Neural Networks Regularization Through Representation Learning

Soufiane Belharbi

Normandie Université

Pour obtenir le diplôme de doctorat

Spécialité Informatique

Préparée au sein de « l'INSA Rouen Normandie »

Neural Networks Regularization Through Representation Learning

Présentée et soutenue par
Soufiane BELHARBI

Thèse soutenue publiquement le 06 Juillet 2018 devant le jury composé de		
Sébastien ADAM	Professeur à l'Université de Rouen Normandie	Directeur de thèse
Clément CHATELAIN	Maître de conférence à l'INSA Rouen Normandie	Encadrant de thèse
Romain HÉRAULT	Maître de conférence à l'INSA Rouen Normandie	Encadrant de thèse
Elisa FROMONT	Professeur à l'Université de Rennes 1	Rapporteur de thèse
Thierry ARTIÈRES	Professeur à l'École Centrale Marseille	Rapporteur de thèse
John LEE	Professeur à l'Université Catholique de Louvain	Examinateur de thèse
David PICARD	Maître de conférences à l'École Nationale Supérieure de l'Électronique et de ses Applications	Examinateur de thèse
Frédéric JURIE	Professeur à l' Université de Caen Normandie	Invité

Thèse dirigée par Sébastien ADAM, laboratoire LITIS

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES ROUEN NORMANDIE

UNIVERSITÉ DE ROUEN NORMANDIE

MIIS

litis

Neural Networks Regularization Through Representation Learning

Soufiane BELHARBI

Supervisor: **Prof. Sébastien ADAM**

Advisor: Assistant Prof. Clément CHATELAIN
Assistant Prof. Romain HÉRAULT

LITIS laboratory
INSA Rouen Normandie

This dissertation is submitted for the degree of
Doctor of Philosophy

Normandie Université July 2018

I would like to dedicate this thesis to my parents and my grandparents.

In memory of Frank Rosenblatt.

Fig. 1 Frank Rosenblatt, with the image sensor of the Mark I Perceptron (Source: Arvin Calspan Advanced Technology Center; Hecht-Nielsen, R. Neurocomputing (Reading, Mass.: Addison-Wesley, 1990); Cornell Library)

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except when specified in the text.

Soufiane BELHARBI
July 2018

Acknowledgements

I would like to thank many people who helped me along my thesis.

I would like to thank my thesis supervisor Sébastien Adam, for taking me as his PhD student. I would like also to thank my advisors Clément Chatelain and Romain Hérault for all their advice and knowledge they have shared with me. I would like to thank as well Romain Modzelewski at Henri Becqurel Center at Rouen for his availability and help for a whole year. I would like to thank as well my collaborators at the same center Sébastien Thureau, and Mathieu Chastan.

Writing this manuscript was a separate challenge. I would like to thank, again, my supervisor Sébastien Adam, and my advisors Clément Chatelain, and Romain Hérault, for their patience, comments, constant criticism, and availability all along the 7 months of writing. It was through their help that this manuscript has reached such maturity.

Preparing the presentation of my PhD defense was the last challenge. Again, my supervisor Sébastien Adam, and my advisors, Clément Chatelain, and Romain Hérault were a major help. It was through their comments and criticism that I have successfully prepared my presentation in terms of content, pedagogy, and speech.

I would like to thank all the jury members of my PhD defense: John Lee, Elisa Fromont, Thierry Artières, David Picard, Frédéric Jurie, Clément Chatelain, Romain Hérault, and Sébastien Adam. I would like to thank all of them for their presence, constructive, detailed, and helpful questions and criticism that helped improving my manuscript and showed their interest to my work. I would like to thank them as well for their valuable time, and their availability to judge my work. It was an honor, and a pleasure defending my work in front of them.

I would like to thank Clément Chatelain for helping me managing my administrative procedure at INSA Rouen Normandie.

I would like to thank Stéphan Canu for his constant insightful conversations and his many offered opportunities and help. He knocks on my office door every morning to say hi and to start the day with another interesting conversation. I thank him for his encouragement, guidance, constructive criticism, and inspiration all along these years.

I would like to thank Alain Rakotomamonjy for his interesting conversations when we have the chance to have one.

I would like to thank Gilles Gasso for all the interesting conversations that we had including science, and running.

I would like to thank Carole Le Guyader for her interesting conversations and her help.

I would like to thank Samia Ainouz for her constant encouragement and help.

I would like to thank Aziz Bensrhair for his encouragement and advice.

I would like to thank John Lee at the Université Catholique de Louvain who admitted me for two weeks in his laboratory. I would like to thank as well Frédéric

Precioso at University of Nice-Sophia Antipolis for inviting me to give a talk in his deep learning summer school.

I would like to thank the staff at the computing centers for their constant support at INSA Rouen Normandie: Jean-François Brulard; Université de Rouen Normandie: Arnaud Citerin; and the CRIANN computing center (www.criann.fr): Benoist Gaston, Patrick Bousquet-Melou, Beatrice Charton, and all the technical support team.

I would like to thank the secretariat staff for their help, and assistance including but not limited to: Brigitte Diara, Sandra Hague, Florence Aubry, Isabelle Poussard, Fabienne Bocquet, Marion Baudesson, Laure Paris, Leila Lahcen, and Rachel Maeght.

I would like to thank Alexis Lechervy at the Université de Caen Normandie, Kamal D S Oberoi, and Lorette Noiret for their time, and availability to revise many of my papers.

I would like to thank Alexis Lechervy for his help and advise since I met him when I was his student.

I would like to thank Gilles Gasso, Antoine Godichon, Alain Rakotomamonjy, Djamila Boukehil, and Yuan Liu for their time, availability, and insightful conversations that helped me improve one of my papers.

I would like to thank the ASI department at INSA Rouen Normandie for opening the door for me to integrate with them, including but not limited to: Nicolas Malandain, Nicolas Delestre, Geraldine Del Mondo, Gilles Gasso, Benoit Gaüzère, Sébastien Bonnegent, Damien Guesdon, Elsa Planterose, Alexandre Pauchet, Michel Mainguenaud, and from the Université de Rouen Normandie: Pierrick Tranouez and Daniel Antelme and all the members of the "miam" group.

I would like to thank all the administration staff at INSA Rouen Normandie and at the Université de Rouen Normandie for their help and assistance.

I would like to thank my previous teachers at the Université de Rouen Normandie for their encouragement, including but not limited to: Thierry Paquet, Laurent Heutte, Stéphane Nicolas, Caroline Petitjean, Maxime Berar, Pierre Héroux, Su Ruan, and Christele Lecomte. I would like also to thank Simon Bernard. I would like to thank my previous teachers at the Université de Caen Normandie as well, including but not limited to: Gaël Dias, Frédéric Jurie, and Alexis Lechervy.

I would like to thank my office colleagues with whom I had insightful conversations and wonderful time inside and outside the office, including but not limited to: Yuan Liu, Linlin Jia, Imad Rida, Cyprien Ruffino, Djamila Boukehil, Denis Rousselle, Sokol Koço, and Kawtar Bomohamed.

I would like to thank Jean-Baptiste Louvet, and Mathieu Bourgais for all the fun conversations we had. Good luck to Jean-Baptiste in saving nature. I hope he can survive without meat.

I Would like to thank all the students, and PhD students that I have met within LITIS laboratory or outside, including but not limited to: Bruno Stuner, Wassim Swaileh, Rivière Marc-Aurèle, Sovann En, Meriem El Azami, Safaa Dafrallah, Imen Beji, Cloé Cabot, Manon Ansart, Laetitia Jeancolas, Tongxue Zhou, Noémie Debroux, Fan Wang, Fadila Taleb, Imene khames, Romaric Pighetti, Rémi Cadène, Mélanie Ducoffe, Jean baptiste Charpentier, Nezha Bouhafs, Ennassiri Hamza, Harik Elhoussein Chouaib, Mohammed Ali Benhamida, Barange Mukesh, Tatiana Poletaeva, Christophe Tirel, Danut Pop, Sahba Zojaji, Alexander Dydychkin, Julien Lerouge, Grégoire Mesnil, Quentin Lerebours, Florian Leriche, Clara Gainon de Forsan.

I would like to thank my colleagues from my engineer school www.esi.dz for their help and encouragement, including but not limited to: Arab Ghedamsi, Boualem Bouldja, Lyas Said Aissa , Sofiane Bellil, Moussa Ailane, and Yasmina Chikhi.

I would like to thank Hanene Haouam for her encouragement.

I would like to thank Amine Frikha for his constant help during all my three years of studies at Rouen and Caen.

I would like to thank my colleagues and PhD representatives at the LITIS laboratory committee: Fabien Bonardi and Riadh Saada. I would like also to thank the members of ADDED association (association-added.fr) who I had a wonderful time working with them for almost two years: Claire Debreux, Xavier Monnier, Steven Araujo, and Javier Anez Perdomo. I would like to thank as well all the PhD students who subscribed and participated into the animation of my club DOC-AC (doctorants-actifs.fr, doctorants-actifs.github.io).

I would like to thank everyone who helped in preparing the reception of my PhD defense, including but not limited to: Cyprien Ruffino, Linlin Jia, Ismaila Seck, Djamila Boukehil, Yuan Liu, Benoit Gaüzère, Pierrick Tranouez, Antoine Godichon, Sandra Hague, Alain Rakotomamonjy, and Simon Bernard. Particular thanks to Brigitte Diara who organized and prepared the whole defense.

I thank everyone that helped me in my PhD thesis, people that I know and do not know of.

In this short acknowledgment, I am certain that I have forgot many other people. For that, I apologize.

This was a wonderful, and a rich experience.

Context

The research that led to this PhD thesis was conducted at the "Institut National des Sciences Appliquées Rouen Normandie (INSA Rouen Normandie)", at the "Laboratoire d'Informatique, de Traitement de l'Information et des Systèmes (LITIS)"[1], in over the course of 3 years, between 2014 and 2017. This work was done in close collaboration with my supervisor Prof. Sébastien ADAM at "l'Université de Rouen", and with my advisors Clément CHATELAIN and Romain HÉRAULT at INSA de Rouen.

[1] Avenue de l'Université, 76801 Saint Etienne du Rouvray Cedex

Summary

Neural network models and deep models are one of the leading and state of the art models in machine learning. They have been applied in many different domains. Most successful deep neural models are the ones with many layers which highly increases their number of parameters. Training such models requires a large number of training samples which is not always available. One of the fundamental issues in neural networks is overfitting which is the issue tackled in this thesis. Such problem often occurs when the training of large models is performed using few training samples. Many approaches have been proposed to prevent the network from overfitting and improve its generalization performance such as data augmentation, early stopping, parameters sharing, unsupervised learning, dropout, batch normalization, etc.

In this thesis, we tackle the neural network overfitting issue from a representation learning perspective by considering the situation where few training samples are available which is the case of many real world applications. We propose three contributions. The first one presented in chapter 2 is dedicated to dealing with structured output problems to perform multivariate regression when the output variable $\mathbf{y}$ contains structural dependencies between its components. Our proposal aims mainly at exploiting these dependencies by learning them in an unsupervised way. Validated on a facial landmark detection problem, learning the structure of the output data has shown to improve the network generalization and speedup its training. The second contribution described in chapter 3 deals with the classification task where we propose to exploit prior knowledge about the internal representation of the hidden layers in neural networks. This prior is based on the idea that samples within the same class should have the same internal representation. We formulate this prior as a penalty that we add to the training cost to be minimized. Empirical experiments over MNIST and its variants showed an improvement of the network generalization when using only few training samples. Our last contribution presented in chapter 4 showed the interest of transfer learning in applications where only few samples are available. The idea consists in re-using the filters of pre-trained convolutional networks that have been trained on large datasets such as ImageNet. Such pre-trained filters are plugged into a new convolutional network with new dense layers. Then, the whole network is trained over a new task. In this contribution, we provide an automatic system based on such learning scheme with an application to medical domain. In this application, the task consists in localizing the third lumbar vertebra in a 3D CT scan. A pre-processing of the 3D CT scan to obtain a 2D representation and a post-processing to refine the decision are included in the proposed system. This work has been done in collaboration with the clinic "*Rouen Henri Becquerel Center*" who provided us with data.

Keywords: neural network, deep learning, regularization, overfitting, feedforawrd networks, convolutional networks, multi-task learning, unsupervised learning, repre-

sentation learning, transfer learning, classification, univariate regression, multivariate regression, structured output prediction, prior knowledge.

Résumé

Les modèles de réseaux de neurones et en particulier les modèles profonds sont aujourd'hui l'un des modèles à l'état de l'art en apprentissage automatique et ses applications. Les réseaux de neurones profonds récents possèdent de nombreuses couches cachées ce qui augmente significativement le nombre total de paramètres. L'apprentissage de ce genre de modèles nécessite donc un grand nombre d'exemples étiquetés, qui ne sont pas toujours disponibles en pratique. Le sur-apprentissage est un des problèmes fondamentaux des réseaux de neurones, qui se produit lorsque le modèle apprend par cœur les données d'apprentissage, menant à des difficultés à généraliser sur de nouvelles données. Le problème du sur-apprentissage des réseaux de neurones est le thème principal abordé dans cette thèse. Dans la littérature, plusieurs solutions ont été proposées pour remédier à ce problème, tels que l'augmentation de données, l'arrêt prématuré de l'apprentissage ("early stopping"), ou encore des techniques plus spécifiques aux réseaux de neurones comme le "dropout" ou la "batch normalization".

Dans cette thèse, nous abordons le sur-apprentissage des réseaux de neurones profonds sous l'angle de l'apprentissage de représentations, en considérant l'apprentissage avec peu de données. Pour aboutir à cet objectif, nous avons proposé trois différentes contributions. La première contribution, présentée dans le chapitre 2, concerne les problèmes à sorties structurées dans lesquels les variables de sortie sont à grande dimension et sont généralement liées par des relations structurelles. Notre proposition vise à exploiter ces relations structurelles en les apprenant de manière non-supervisée avec des autoencodeurs. Nous avons validé notre approche sur un problème de régression multiple appliquée à la détection de points d'intérêt dans des images de visages. Notre approche a montré une accélération de l'apprentissage des réseaux et une amélioration de leur généralisation. La deuxième contribution, présentée dans le chapitre 3, exploite la connaissance *a priori* sur les représentations à l'intérieur des couches cachées dans le cadre d'une tâche de classification. Cet *a priori* est basé sur la simple idée que les exemples d'une même classe doivent avoir la même représentation interne. Nous avons formalisé cet *a priori* sous la forme d'une pénalité que nous avons rajoutée à la fonction de perte. Des expérimentations empiriques sur la base MNIST et ses variantes ont montré des améliorations dans la généralisation des réseaux de neurones, particulièrement dans le cas où peu de données d'apprentissage sont utilisées. Notre troisième et dernière contribution, présentée dans le chapitre 4, montre l'intérêt du transfert d'apprentissage ("transfer learning") dans des applications dans lesquelles peu de données d'apprentissage sont disponibles. L'idée principale consiste à pré-apprendre les filtres d'un réseau à convolution sur une tâche source avec une grande base de données (ImageNet par exemple), pour les insérer par la suite dans un nouveau réseau sur la tâche cible. Dans le cadre d'une collaboration avec le centre de lutte contre le cancer "*Henri Becquerel de Rouen*", nous avons construit un système automatique basé sur ce type de transfert d'apprentissage pour une application médicale où l'on

dispose d'un faible jeu de données étiquetées. Dans cette application, la tâche consiste à localiser la troisième vertèbre lombaire dans un examen de type scanner. L'utilisation du transfert d'apprentissage ainsi que de prétraitements et de post traitements adaptés a permis d'obtenir des bons résultats, autorisant la mise en œuvre du modèle en routine clinique.

Mots clés: réseaux de neurones, apprentissage profond, régularisation, sur-apprentissage, réseau de neurones à passe avant, réseaux de neurones convolutifs, apprentissage multi-tâches, apprentissage non supervisé, apprentissage des représentations, transfert d'apprentissage, classification, régression univariée, régression multiple, prédiction à sortie structurée, connaissances à priori.

Publications

Journals:

- **Deep Neural Networks Regularization for Structured Output Prediction**. Soufiane Belharbi, Romain Hérault[2], Clément Chatelain[2], and Sébastien Adam. Neurocomputing Journal, 281C:169-177, 2018.
- **Spotting L3 Slice in CT Scans using Deep Convolutional Network and Transfer Learning**. Soufiane Belharbi, Clément Chatelain[2], Romain Hérault[2], Sébastien Adam, Sébastien Thureau, Mathieu Chastan, and Romain Modzelewski. *Computers in Biology and Medicine, 87: 95-103, 2017.*
- **Neural Networks Regularization Through Class-wise Invariant Representation Learning**. Soufiane Belharbi, Clément Chatelain, Romain Hérault, and Sébastien Adam. **Under review**, *2017.*

International conferences/workshops:

- **Learning Structured Output Dependencies Using Deep Neural Networks**. Soufiane Belharbi, Clément Chatelain, Romain Hérault, and Sébastien Adam. Deep Learning Workshop in the 32^{nd} International Conference on Machine Learning (ICML), 2015.
- **Deep Multi-Task Learning with Evolving Weights**. Soufiane Belharbi, Romain Hérault, Clément Chatelain, and Sébastien Adam. European Symposium on Artificial Neural Networks (ESANN), 2016.

French conferences:

- **A Unified Neural Based Model For Structured Output Problems**. Soufiane Belharbi, Clément Chatelain, Romain Hérault, and Sébastien Adam. Conférence Francophone sur l'Apprentissage Automatique (CAP), 2015.
- **Pondération Dynamique dans un Cadre Multi-Tâche pour Réseaux de Neurones Profonds**. Soufiane Belharbi, Romain Hérault, Clément Chatelain, and Sébastien Adam. Reconnaissance des Formes et l'Intelligence Artificielle (RFIA) (Session spéciale: Apprentissage et vision), 2016.

[2]Authors with equal contribution.

Table of Contents

Summary xiii

Résumé xv

List of Figures xxiii

List of Tables xxv

List of Symbols xxvii

General Introduction 1

1 Background 5
1.1 Machine Learning . . . 5
1.1.1 Learning from Data . . . 5
1.1.2 Statistical Learning and Generalization: PAC Learning . . . 7
1.1.3 Empirical Risk Minimization with Inductive Bias . . . 11
1.1.4 Summary . . . 15
1.2 Introduction to Feedforward Neural Networks . . . 16
1.2.1 Early History . . . 16
1.2.2 Perceptron . . . 17
1.2.3 Gradient Based Learning . . . 19
1.2.4 Multilayer Perceptron and Representation Learning . . . 20
1.2.5 Deep Learning: from Late 60′s to Today . . . 22
1.2.6 Summary . . . 27
1.3 Improving Neural Networks Generalization . . . 28
1.3.1 Explicit Regularization: Explicit Complexity Reduction . . . 28
1.3.2 Implicit Regularization . . . 29
1.3.3 Summary . . . 45
1.4 Conclusion . . . 45

2 Deep Neural Networks Regularization for Structured Output Prediction 47
2.1 Prologue . . . 47
2.2 Introduction . . . 49
2.3 Related Work . . . 50
2.3.1 Graphical Models Approaches . . . 50
2.3.2 Deep Neural Networks Approaches . . . 51

2.4 Multi-task Training Framework for Structured Output Prediction . . . 52
2.5 Implementation . . . 55
2.6 Experiments . . . 55
2.6.1 Datasets . . . 56
2.6.2 Metrics . . . 57
2.6.3 General training setup . . . 57
2.7 Conclusion . . . 64

3 Neural Networks Regularization Through Class-wise Invariant Representation Learning 67
3.1 Prologue . . . 67
3.2 Introduction . . . 69
3.3 Related Work . . . 70
3.4 Proposed Method . . . 72
3.4.1 Model Decomposition . . . 72
3.4.2 General Training Framework . . . 73
3.4.3 Implementation and Optimization Details . . . 74
3.5 Experiments . . . 75
3.5.1 Classification Problems and Experimental Methodology . . . 76
3.5.2 Results . . . 77
3.5.3 On Learning Invariance within Neural Networks . . . 79
3.6 Conclusion . . . 81

4 Application: Spotting L3 Slice in CT Scans using Deep Convolutional Network and Transfer Learning 83
4.1 Prologue . . . 83
4.2 Introduction . . . 85
4.3 Related Work . . . 86
4.4 Proposed Approach . . . 88
4.4.1 MIP Transformation . . . 89
4.4.2 Learning the TL-CNN . . . 90
4.4.3 Decision Process using a Sliding Window over the MIP Images . 93
4.5 Experimental Protocol . . . 95
4.5.1 CT Exams Database Description . . . 95
4.5.2 Datasets Preparation . . . 95
4.5.3 Neural Networks Models . . . 95
4.6 Results . . . 96
4.6.1 Data View: Frontal Vs. Lateral . . . 96
4.6.2 Detection Performance . . . 97
4.6.3 Processing Time Issues . . . 98
4.6.4 Comparison with Radiologists . . . 98
4.7 Conclusion . . . 99

General Conclusion and Perspectives 101

References 107

Appendix A Definitions and Technical Details **135**
A.1 Machine Learning Definitions 135
A.1.1 Applications 135
A.1.2 Terminology 136
A.1.3 Learning Scenarios 138
A.2 Technical Details 139
A.2.1 Bias-variance Tradeoff 139
A.2.2 Feedforward Neural Networks 142
A.2.3 Regularization 157

List of Figures

1 Frank Rosenblatt, with the image sensor of the Mark I Perceptron. . . iii

1.1 Illustration of structural risk minimization (SRM). 14
1.2 Perceptron model. 17
1.3 Multiclass perceptron. 18
1.4 Multilayer perceptron. 22
1.5 *Lenet* convolutional network. 24
1.6 Example of *Deep Residual* network architectures for image recognition. 26
1.7 Training and validation learning curves. 30
1.8 Common architecture for multi-task learning in neural networks. 33
1.9 Transfer learning scheme. 35
1.10 Three ways in which transfer learning might improve learning. 36
1.11 Transfer learning vs. multi-task learning. 36
1.12 A 2-dimensional convolution layer. 40
1.13 Dropout applied to a neural network. 41

2.1 Examples of facial landmarks from LFPW training set. 49
2.2 Proposed MTL framework. 54
2.3 Samples from HELEN dataset. 57
2.4 Linear adaptation of the importance weights during training. 59
2.5 Examples of prediction on LFPW test set. 60
2.6 Examples of prediction on HELEN test set. 60
2.7 MSE during training epochs over HELEN train. 62
2.8 CDF curves. 63

3.1 Input/Hidden representations in an MLP. 70
3.2 Decomposition of the neural network. 73
3.3 Constraining the intermediate representations of an MLP. 74
3.4 Samples from training set of each benchmark. 76
3.5 Measuring the hint term of the different layers. 80

4.1 Finding the L3 slice within a whole CT scan. 86
4.2 Two slices from the same patient. 87
4.3 System overview. 89
4.4 Examples of normalized frontal MIP 90
4.5 System overview. 92
4.6 CNN output and post-processing. 94

A.1 Complexity vs. generalization. 138

A.2 Relationship between model capacity, its bias and variance and the generalization error. . . . 142
A.3 The backward pass of the backpropagation algorithm. . . . 145
A.4 The backpropagation algorithm in a matrix form. . . . 147
A.5 Examples of nonlinear activation functions. . . . 148
A.6 Rectified linear unit function. . . . 149
A.7 Effect of depth on the generalization of neural networks. . . . 154
A.8 Effect of the number of parameters on the generalization of neural networks. . . . 155
A.9 Effect of L_2 norm regularization. . . . 159
A.10 Effect of L_1 norm regularization. . . . 162
A.11 An illustration of the effect of early stopping. . . . 164

List of Tables

2.1 MSE over LFPW. 60
2.2 MSE over HELEN. 61
2.3 **AUC** and $\mathbf{CDF_{0.1}}$ performance over LFPW test set. 61
2.4 **AUC** and $\mathbf{CDF_{0.1}}$ performance over HELEN test set. 61
2.5 Size of augmented LFPW and HELEN train sets. 64

3.1 Mean ± standard deviation error over validation and test sets. 78
3.2 Mean ± standard deviation error over validation and test sets. 79
3.3 Mean ± standard deviation error over validation and test sets. 79

4.1 Test error. 97
4.2 Error expressed in slice over all the folds. 97
4.3 Number of parameters vs. processing time. 98
4.4 Comparison of the performance. 99

List of Symbols

The next list describes several notations and symbols that will be later used within the body of the document, unless we redefine the notations depending on the context.

Numbers and Arrays

a A scalar (integer or real)

$\boldsymbol{a}$ A vector

$\boldsymbol{A}$ A matrix

$\boldsymbol{I}_n$ Identity matrix with n rows and n columns

$\boldsymbol{I}$ Identity matrix with dimentionality implied by the context

$\text{diag}(\boldsymbol{a})$ A square, diagonal matrix with diagonal entries given by $\boldsymbol{a}$

a A scalar random variable

$\mathbf{a}$ A vector-valued random variable

$\mathbf{A}$ A matrix-valued random variable

$\mathbf{0}$ A vector or a matrix, depending on the context, full of 0

Sets

$\mathbb{A}$ A set

$a \in \mathbb{A}$ Element a in set $\mathbb{A}$

$\mathbb{A} \subset \mathbb{B}$ Set $\mathbb{A}$ is a subset of set $\mathbb{B}$

$|\mathbb{A}|$ Number of elements in set $\mathbb{A}$

$\mathbb{R}$ Set of real numbers

$\mathbb{R}_+$ Set of non-negative real numbers

$\mathbb{R}^n$ Set of n-dimensional real-valued vectors

$\mathbb{R}^{n \times m}$ Set of $n \times m$-dimensional real-valued matrices

$\mathbb{N}$ Set of natural numbers, i.e., $\{0, 1, \cdots\}$

$\{0, 1, \cdots, n\}$ The set of all integers between 0 and n

$[a, b]$ Closed interval between a and b

$\{a, b, c\}$ Set containing elements a, b and c

Indexing

a_i Element i of vector $\boldsymbol{a}$

$A_{i,j}$ Element i, j of matrix $\boldsymbol{A}$

$\boldsymbol{A}_{i,:}$ Row i of matrix $\boldsymbol{A}$

$\boldsymbol{A}_{:,i}$ Column i of matrix $\boldsymbol{A}$

Linear Algebra Operations

$\boldsymbol{A}^\top$ Transpose of matrix $\boldsymbol{A}$

$\boldsymbol{A} \odot \boldsymbol{B}$ Element-wise (Hadamard) product of $\boldsymbol{A}$ and $\boldsymbol{B}$

$\langle \boldsymbol{a}, \boldsymbol{b} \rangle$ Inner product between vectors $\boldsymbol{a}$ and $\boldsymbol{b}$

Calculus

$\frac{\partial y}{\partial x}$ Partial derivative of y with respect to x

$\nabla_{\boldsymbol{x}} y$ Gradient of y with respect to $\boldsymbol{x}$

$\frac{\partial f}{\partial \boldsymbol{x}}$ Jacobian matrix $\boldsymbol{J} \in \mathbb{R}^{m \times n}$ of $f : \mathbb{R}^n \to R^m$

Probability and Information Theory

$P(a)$ A probability distribution over a discrete variable

$p(a)$ A probability distribution over a continuous variable, or over a variable whose type has not been specified

$\mathrm{a} \sim P$ Random variable a has distribution P

$\mathop{\mathbb{E}}_{x \sim D}[\cdot]$ Expectation over x drawn from distribution D

$\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$ Gaussian distribution over $\boldsymbol{x}$ with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$

$\mathcal{D}$ Unspecified probability distribution

Functions

$f : \mathcal{X} \to \mathcal{Y}$ The function f with domain $\mathcal{X}$ and range $\mathcal{Y}$

$f(\boldsymbol{x}; \boldsymbol{\theta})$ A function of $\boldsymbol{x}$ parametrized by $\boldsymbol{\theta}$. (Sometimeswe write $f(\boldsymbol{x})$ and omit the argument $\boldsymbol{\theta}$ to lighten the notation)

$\log x$ Natural logarithm of x

$\log_a$ Logarithm with base a

$\|\boldsymbol{x}\|$ L_2 norm of $\boldsymbol{x}$

$\|\boldsymbol{x}\|_p$ L_p norm of $\boldsymbol{x}$

$1_{\mathbb{A}}$ Indicator function indicating membership in subset $\mathbb{A}$

$R(\cdot)$ Generalization error or risk

$\hat{R}(\cdot)$ Empirical error or risk

Datasets and Distributions

$\mathcal{X}$ Input space

$\mathcal{Y}$ Target space

p_{data} The data generating distribution

$\hat{p}_{\text{data}}$ The empirical distribution defined by the training set

$\mathbb{D}$ A set of samples

$\boldsymbol{x}^{(i)}$ The i-th example (input) from a dataset

$y^{(i)}$ or $\boldsymbol{y}^{(i)}$ The target associated with $\boldsymbol{x}^{(i)}$ for supervised learning

$\boldsymbol{X}$ The $m \times n$ matrix with input examples $\boldsymbol{x}^{(i)}$ in row $\boldsymbol{X}_{i,:}$

General Introduction

In the last years, the neural network field has seen large success for different applications that require large number of features to solve complex tasks. This success has made neural networks one of the leading models in machine learning as well as the state of the art for different applications. Among the tasks that have been well modeled using neural networks, one can mention image classification, image labeling, object detection, image description, speech recognition, speech synthesis, query answering, text generation, etc. However, in order to solve such complex tasks, neural network models rely on a large number of parameters that may easily reach millions. Consequently, such models require a large number of training data in order to avoid overfitting, a case where the model becomes too specific to the training data and looses its ability to generalize to unseen data. In practice, one usually deals with applications with few training samples. Therefore, one has the choice either to use neural network models with small capacity and loose much of their power, or stick with models with large capacity and employ what it is known as regularization techniques to save the generalization ability and to prevent the network from overfitting. We provide in chapter 1 a background on machine learning with a focus on regularization aspects. The same chapter contains a brief introduction to neural network models and their regularization techniques. The chapter is completed by Appendix A that provides some basic definitions in machine learning, and more technical details on neural networks and regularization.

In the literature of neural networks, we find different approaches of regularization that we describe in Sec.1.3. Such regularization methods either aim at explicitly reducing the model complexity using for example L_p parameter norm (Sec.1.3.1), or implicitly reducing its complexity using for instance early stopping (Sec.1.3.2.1). Other methods do not alter the model complexity but tackle the issue of overfitting using different angles where the aim is to improve the model's generalization. One can mention the following approaches:

- parameter sharing (Sec.1.3.2.5),
- data augmentation (Sec.1.3.2.2),
- batch normalization (Sec.1.3.2.7),
- and smart ensemble methods such as dropout (Sec.1.3.2.6).

[154] covers most of the technical approaches used to regularize neural networks.

In this thesis-by-article, we provide three different ways of regularizing neural network models when trained using small dataset for the task of classification, univariate, and multivariate regression. The aim of such approaches is to directly improve the model's generalization without affecting its complexity. Our research direction to

improve neural networks generalization in this thesis is throughout learning good internal representations within the network.

Data representation within neural networks is a key component of their success. Such models are able to self-learn adequate internal representations to solve the task in hand. This ability is a major factor that separates such models from other type of models in machine learning. Although models with high capacity are able to build complex features that allow to solve complex tasks, they are more likely to fall into overfitting, particularly when they are trained with few samples. In this thesis, we provide three approaches to regularize neural networks. Two of them are based on theoretical frameworks which adapt learning internal representations to the task in hand. This includes structured output prediction through unsupervised learning (chapter 2), and classification task through prior knowledge (chapter 3). The last approach is based on transfer learning with an applicative aspect in medical domain (chapter 4). Each of our contributions is presented in a chapter.

- **Structured output prediction**
  In chapter 2, we explore the use of unsupervised learning to build robust features in order to solve structured output problems, i.e., to perform multivariate regression where the output variable is high dimensional with possible structural relations between its components. The motivation behind this work is that feedforward networks lack, by their structural design, the ability to learn dependencies among the output variable. We recall that each output neuron in a feedforward network performs the prediction independently from the rest of the other output neurons. In order to learn the output structure, local or global, the network needs a large amount of training data in order to learn the output variations. Otherwise, the network falls into overfitting or in the case of outputting the mean structure. In this work, we want the network to focus explicitly on learning the output structure. In order to do so, we propose a unified multi-task framework to regularize the training of the network when dealing with structured output data. The framework contains a supervised task that maps the input to the output and two unsupervised tasks where the first learns the input distribution while the second learns the output distribution. The later one, which is the key component of the framework, allows learning explicitly the output structure in an unsupervised way. This allows the use of labels only data to learn such structure. We validate our framework over a facial landmark detection problem where the goal is to predict a set of key points on face images. Our experiments show that the use of our framework speeds up the training of neural networks and improves their generalization.

- **Prior knowledge and classification**
  In chapter 3, we explore another aspect of learning representations within neural network. We investigate the use of prior knowledge. Using prior knowledge can be considered as a regularization. It allows to promote generalization without the need to see large amount of training data. For instance, in the case of a classification task, if someone provides us the information that "generally, a car has four wheels", it could save us the need to see a large number of car images to understand such a key concept in order to figure out what a car is. In the context of classification, a general prior knowledge about the internal representation of

a network is that samples within the same class should have the same internal representation. Based on this belief, we propose a new regularization penalty that constrains the hidden layers to build similar features for samples within the same class. Validated on MNIST dataset and its variants, incorporating such prior knowledge in the network training allows improving its generalization while using few training samples.

- **Transfer learning and medical domain**
  Our last contribution, presented in chapter 4, consists in a different strategy to boost learning complex features using high capacity models while using few training samples. The hope is to maintain the model generalization. As it seems in contradiction with the generalization theory (Sec.1.1) that states that to well fit a model with high capacity we need a large number of training data, empirical evaluation shows the possibility of such approach. In this work, we provide a real life application of a learning scheme that allows us to train a model with high capacity using only few training samples. The learning scheme consists in training a model with high capacity over a task that has abundant training samples such as training large convolutional network over ImageNet dataset. Then, we re-use part of the trained model, responsible for features building, in the second task which has only few training samples. This training scheme, known as transfer learning, allows the second task which has a lack of data to benefit from another task with abundant data. We validate this learning scheme over a real life application based on medical image data. The task consists in localizing a specific vertebra, precisely the third lumbar vertebra, in a 3D CT scan of a patient's body. We provide adequate pre-processing and post-processing procedures. We report satisfying results. This work was done in collaboration with the clinic "Rouen Henri Becquerel Center" which provided us with the data.

How to Read This Thesis

This is a thesis by article. It is composed of 4 chapters. chapter 1 is an introduction which is composed of four sections: Sec.1.1 presents machine learning backgrounds with more focus on regularization; Sec.1.2 presents an introduction to feedforward neural networks; Sec.1.3 contains methods that are used to improve neural networks generalization; Sec.1.4 contains the conclusion of the first chapter. If the reader is familiar with these subjects, we recommend skipping the first chapter. However, we recommend reading Sec.1.1.3 that presents the regularization concept and Sec.1.3.2.8 which presents unsupervised and semi-supervised learning in neural networks as a regularization.

chapter 2, chapter 3, and chapter 4 contain our three contributions.

In order to keep this thesis short, straightforward, and self-contained, we provide one appendix that covers some basic definitions in machine learning and some technical details on neural networks and regularization (Appendix A).

Chapter 1

Background

We discuss in this first chapter three important subjects that constitute the background of this thesis: machine learning, neural networks, and methods to improve neural networks generalization. In the first section (Sec.1.1), we provide an introduction to machine learning problem with an emphasis on the generalization aspect and how one can improve it. As this thesis concerns specifically neural networks, we then provide an introduction to such models in the second section (Sec.1.2). Our main concern behind this thesis is to provide new techniques to improve the generalization performance of neural network, particularly when dealing with small training sets. Therefore, the last section (Sec.1.3) contains a presentation of the most common methods used to improve the generalization performance of neural networks.

This chapter is inspired from machine learning and neural networks literature [305, 385, 5, 232, 154, 103] including precise definitions and theorems.

1.1 Machine Learning

This section contains an introduction to machine learning with a focus on the generalization aspect. Appendix A.1 contains basic definitions of machine learning if needed.

1.1.1 Learning from Data

If a three-year-old kid is shown a picture and asked if there is a tree in it, it is high likely that we get the right answer. If we ask the three-year-old kid what is a tree, we likely get an inconclusive answer. We, humans, did not learn what is a tree by studying the mathematical definition of trees. We learned it by looking at trees. In other words, we learned from *data*, i.e., *examples*.

The term *machine learning* refers to the automated detection of meaningful patterns in data. In the past couple of decades, it has become a common tool in almost any task that requires information extraction from large datasets. Nowadays, we are surrounded by machine learning based technology almost anywhere: search engines learn how to bring us the best results, antispam software learns how to filter our email inbox, and credit card transactions are secured by a software that learns to detect frauds. Digital cameras learn to detect faces and intelligent personal assistance applications

on smart-phones learn to recognize voice commands. Cars are equipped with accident prevention systems that are built using machine learning algorithms. Hospitals are equipped with programs that assist doctors in their diagnostics. Machine learning is also widely used in scientific applications such as bioinformatics, medicine, and astronomy.

A common feature of all these applications is that, in contrast to the traditional use of computers, in these cases, due to complexity of the patterns that need to be detected, a human programmer can not provide an explicit, fine-detailed specification of how such tasks should be executed. Taking example from intelligent beings, many of our skills are acquired or refined through *learning* from our experience. Machine learning tools are concerned with endowing programs with the ability to "learn" and adapt.

Learning can be thought of as the process of converting experience into expertise or knowledge. Machine learning aims at incorporating such a concept into computers or any other computing device such as phones, tablets, etc. Now, why do we need machine learning tools? one of the reasons is to automate processes and eliminate humans routines to free them to do more intelligent and delicate tasks. Another important reason is to fill the gap in the human capabilities to perform tasks that go beyond their abilities. Such tasks require analyzing very large and complex data: astronomical data, turning medical archives into medical knowledge, weather prediction, analysis of genomic data, web search engines, electronic commerce, etc. With more and more available digitally recorded data, it becomes obvious that there is meaningful information buried in such data that are too large and too complex for human to process them or make sense of them. The last reason that we mention here is the limiting feature of traditional programmed tools which is their rigidity. Once installed, a programmed tool does not change. However, many tasks change over time or from a user to another. Machine learning based tools provide a solution to such issues. They are, by nature, adaptive to changes in the environment they interact with. A typical successful application of machine learning to such problems include programs that decode handwritten text, where a fixed program can adapt to variations between the handwriting of different users.

Machine learning covers a large spectrum of tasks to adapt to the human needs. Such tasks include classification, regression, ranking, clustering, dimensionality reduction, etc. Machine learning also provides different learning strategies as an adaptive way to the nature of available data and the environment including supervised/unsupervised/semi-supervised learning, transductive inference, online learning, reinforcement learning, active learning, etc. We assume the reader is familiar with basic definitions of machine learning. In the other case, Appendix A.1 contains such definitions.

Let us go back to our problem of tree detection in images. To solve this problem we decide to use a machine learning tool in a supervised context where we collect a set of images and annotate them to indicate if they contain a tree or not. One may ask the following questions: is it possible to learn the concept of trees? which machine learning algorithm should we use? how many pictures of trees do we need for learning? is there any guarantee that our algorithm will succeed to detect a tree that was not seen during training? such questions are legitimate and they construct the foundation of machine learning. Computational learning theory [232], which is a subfield of artificial intelligence devoted to studying the design and analysis of machine learning algorithms, provides us with well studied and formal frameworks to answer the aforementioned

questions and more. For the sake of simplicity, and in order to address such questions, we decide to use in this thesis the *probably approximately correct* learning framework, also known as PAC learning, proposed by Leslie Valiant [436] in the 80′s.

A fundamental pillar in the learning concept is the ability of the learner to perform well for unseen situations. This aspect in learning is known as *generalization*. It is likely in practice that the learner fails to acquire such capability and falls into what is known as *overfitting*, a situation where the learner performs well on the training data but fails to generalize to unseen data. *Regularization* is a well known approach in machine learning that we use to deal with such issue. For the sake of simplicity and coherence with our contributions in this thesis, we will present in the next section only some selected concepts in the PAC learning framework in order to answer the aforementioned questions while building our way and justifications toward the concept of regularization.

1.1.2 Statistical Learning and Generalization: PAC Learning

In this section, we present a simplified version of PAC learning framework in order to define the learnability of a concept, the generalization aspect, and the number of samples needed to learn. For illustration, we consider PAC learning framework for learning a binary classification task. Extension to other tasks is possible as well [305, 385, 5].

We denote by $\mathcal{X}$ the set of all possible examples or instances. $\mathcal{X}$ is also referred to as the input space. The set of all possible labels or target values is denoted by $\mathcal{Y}$ which is referred to as the output space. $\mathcal{Y}$ is limited to two labels, $\mathcal{Y} = \{0, 1\}$, where 1 refers to the class `tree` while 0 refers to the class `not-tree`.

A concept $c : \mathcal{X} \to \mathcal{Y}$ is a mapping function. A concept class is a set of concepts we may wish to learn one of them and is denoted by $\mathbb{C}$. We note by h a hypothesis which is also a mapping function: $h : \mathcal{X} \to \mathcal{Y}$. $\mathbb{H}$ is the hypothesis set. Concepts and hypotheses have the same nature, both of them are mapping functions that take an element from $\mathcal{X}$ and map it into $\mathcal{Y}$. Therefore, the terms hypothesis and concept can be interchangeable. In this context, the main difference between a concept and a hypothesis is that the learner does not have access to the concept c while it does know the hypothesis h. Also, this distinction makes it possible that the set of possible hypotheses may not contain c, the true hypothesis that associates a label to a random input. In this context, the aim of the learning algorithm is to pick a hypothesis h that approximates c. We note that in the case of classification, a hypothesis h and a concept c can be called a *classifier*.

Let us assume that examples are independently and identically distributed (i.i.d.) according to some fixed but unknown distribution $\mathcal{D}$. The learning problem is then formulated as follows. The learning algorithm considers a fixed set of possible concepts $\mathbb{H}$, which may not coincide with $\mathbb{C}$. It receives N samples $\mathbb{S} = (\boldsymbol{x}^{(1)}, \cdots, \boldsymbol{x}^{(N)})$ drawn i.i.d. according to $\mathcal{D}$ as well as the labels $(c(\boldsymbol{x}^{(1)}), \cdots, c(\boldsymbol{x}^{(N)}))$, which are based on a specific target concept $c \in \mathbb{C}$ to learn. The set $\{(\boldsymbol{x}^{(i)}, c(\boldsymbol{x}^{(i)}))\}_{i=1}^{N}$ is known as the training set. In order to measure the success of a selected hypothesis h, we need an error measure. In the case of classification, we can use a 0-1 loss function defined as an

indicator function that indicates whether h makes a mistake or not over a sample $\boldsymbol{x}$,

$$1_{h(\boldsymbol{x})\neq c(\boldsymbol{x})} = \begin{cases} 1 & \text{if } h(\boldsymbol{x}) \neq c(\boldsymbol{x}) \\ 0 & \text{if } h(\boldsymbol{x}) = c(\boldsymbol{x}) \ . \end{cases} \tag{1.1}$$

The loss function, defined to measure the error committed by h, depends on the task in hand. For instance, in a regression task, a possible loss is the square loss $(h(x) - c(x))^2$. The error of a hypothesis over all examples that can be sampled from $\mathcal{D}$ is referred to as *generalization error*.

The generalization error of a hypothesis $h \in \mathbb{H}$, also referred to as the true error, or just error of h, is denoted by $R(h)$ and defined as follows [305, 385, 5]

Definition 1.1. ***Generalization error***
Given a hypothesis $h \in \mathbb{H}$, a target concept $c \in \mathbb{C}$, and an underlying distribution $\mathcal{D}$, the generalization error or risk of h is defined by

$$R(h) = \Pr_{\mathbf{x}\sim\mathcal{D}}[h(\boldsymbol{x}) \neq c(\boldsymbol{x})] = \mathop{\mathbb{E}}_{\mathbf{x}\sim\mathcal{D}}[1_{h(\boldsymbol{x})\neq c(\boldsymbol{x})}] \ . \tag{1.2}$$

The learning algorithm task is to use the labeled samples $\mathbb{S}$ to select a hypothesis $h_{\mathbb{S}} \in \mathbb{H}$ that has a small generalization error with respect to the concept c.

The generalization error of a hypothesis is not directly accessible to the learning algorithm since both distribution $\mathcal{D}$ and the target concept c are unknown. Therefore, another method is required in order to measure how well does a hypothesis h. A useful notion of error that can be calculated by the learning algorithm is the *training error*, which is the error of the classifier h over the training set. The term *empirical error* or *empirical risk* are also used for this error. It is defined as follows [305, 385, 5]

Definition 1.2. ***Empirical error***
Given a hypothesis $h \in \mathbb{H}$, a target concept $c \in \mathbb{C}$, and a set of samples $\mathbb{S} = (\boldsymbol{x}^{(1)}, \cdots, \boldsymbol{x}^{(N)})$, the empirical error or the empirical risk of h is defined by

$$\hat{R}(h) = \frac{1}{N}\sum_{i=1}^{N} 1_{h(\boldsymbol{x}^{(i)})\neq c(\boldsymbol{x}^{(i)})} \ . \tag{1.3}$$

Thus, the empirical error of $h \in \mathbb{H}$ is the average error over the samples $\mathbb{S}$, while the generalization error is the expected error based on the distribution $\mathcal{D}$. Since the training samples $\mathbb{S}$ is a snapshot of $\mathcal{D}$ that is available to the learning algorithm, it makes sense to search for a solution that works well on $\mathbb{S}$. This learning paradigm that outputs a hypothesis h that minimizes $\hat{R}(h)$ is called *Empirical Risk Minimization* or ERM for short.

The following introduces the PAC learning framework. We denote by $O(n)$ an upper bound on the cost of the computational representation of any $\boldsymbol{x} \in \mathcal{X}$ and by size(c) the maximal cost of the computational representation of $c \in C$. For example, $\boldsymbol{x}$ may be a vector in $\mathbb{R}^n$, for which the cost of an array-based representation would be $O(n)$.

Definition 1.3. ***PAC-learning***
A concept class $\mathbb{C}$ is said to be PAC-learnable if there exists an algorithm $\mathcal{A}$ and a polynomial function poly$(\cdot,\cdot,\cdot,\cdot)$ such that for any $\epsilon > 0$ and $\delta > 0$, for all distributions

$\mathcal{D}$ on $\mathcal{X}$ and for any target concept $c \in \mathbb{C}$, the following holds for any number of samples $N \geq poly(1/\epsilon, 1/\delta, n, size(c))$:

$$\Pr_{\mathbf{x}\sim\mathcal{D}^N}[R(h_{\mathbb{S}}) \leq \epsilon] \geq 1 - \delta \ . \tag{1.4}$$

A concept class $\mathbb{C}$ is thus PAC-learnable if the hypothesis returned by the learning algorithm $\mathcal{A}$ after observing a number of points is *approximately correct* (with error at most ϵ) with high *probability* (at least $1 - \delta$), which justifies the PAC-learning terminology. The definition of PAC-learning contains two approximation parameters which are predefined beforehand. The accuracy parameter ϵ determines how far the output hypothesis (classifier) is from the optimal one, and a confidence parameter δ which indicates how likely the classifier to meet that accuracy requirement.

Several key points of the PAC-learning definition are worth mentioning. First, the PAC-learning framework is a distribution-free model, no particular assumption is made about the distribution from which samples are drawn. Second, the training data and the test data are drawn from the same distribution $\mathcal{D}$. This is a necessary assumption for the generalization to be possible in most cases. Finally, the PAC-learning framework deals with the learnability of a concept class $\mathbb{C}$ not a particular concept c.

Up to now, we have provided only the definition of a learnable concept class. The next paragraph provides a condition on the sample complexity [305, 385], i.e., the minimal number of samples needed in order to guarantee a probably approximately correct solution. An upper bound of the generalization error is as well provided. First, let us consider the case of a consistent hypothesis $h_{\mathbb{S}}$ which is a hypothesis that admits no error on the training samples $\mathbb{S}$: $\hat{R}(h_{\mathbb{S}}) = 0$. Whereas, $h_{\mathbb{S}}$ is said to be inconsistent when it has errors on the training samples: $\hat{R}(h_{\mathbb{S}}) > 0$.

Theorem 1.1. ***Learning bounds: finite $\mathbb{H}$, consistent case***
Let $\mathbb{H}$ be a finite set of functions mapping from $\mathcal{X}$ to $\mathcal{Y}$. Let $\mathcal{A}$ be an algorithm that for any target concept $c \in \mathbb{H}$ and any i.i.d. samples set $\mathbb{S}$ returns a consistent hypothesis $h_{\mathbb{S}} : \hat{R}(h_{\mathbb{S}}) = 0$. Then, for any $\epsilon, \delta > 0$, the inequality $\Pr_{\mathbb{S}\sim D^N}[R(h_{\mathbb{S}}) \leq \epsilon] \geq 1 - \delta$ holds if

$$N \geq \frac{1}{\epsilon}\left(\log|\mathbb{H}| + \log\frac{1}{\delta}\right) \ . \tag{1.5}$$

This sample complexity result admits the following equivalent statement as a generalization bound: for any $\epsilon, \delta > 0$, with probability at least $1 - \delta$,

$$R(h_{\mathbb{S}}) \leq \frac{1}{N}\left(\log|\mathbb{H}| + \log\frac{1}{\delta}\right) \ . \tag{1.6}$$

Theorem 1.1 shows that when the hypothesis set $\mathbb{H}$ is finite, $\mathcal{A}$, that returns a consistent hypothesis, is a PAC-learning algorithm under the condition of the availability of enough training samples. As shown by Eq.1.6, the generalization error of consistent hypotheses is upper bounded by a term that decreases as a function of the number of training samples N. This is a general fact, as expected, learning algorithms benefit from large labeled training samples. The decrease rate of $O(1/N)$ guaranteed by this theorem, however, is particularly favorable. The price to pay for coming up with a consistent hypothesis is the use of a larger hypothesis set $\mathbb{H}$ in order to increase the chance to find target concepts. As shown in Eq.1.6, the upper bound increases with the cardinality $|\mathbb{H}|$. However, that dependency is only logarithmic. We note that the

term $\log|\mathbb{H}|$ can be interpreted as the number of bits needed to represent $\mathbb{H}$. Therefore, the generalization guarantee of the theorem is controlled by the ratio of this number of bits $\log|\mathbb{H}|$ and the number of training samples N.

In most general case, there may be no hypothesis in $\mathbb{H}$ consistent with the labeled training samples. This, in fact, is the typical case in practice where the learning problem may be somewhat difficult or the concept class may be more complex than the hypothesis set used by the learning algorithm. The learning guarantees in this more general case can be derived under an inequality form that relates the generalization error and empirical error for all the hypotheses $h \in \mathbb{H}$ [305, 385].

Theorem 1.2. ***Learning bound: finite*** $\mathbb{H}$***, inconsistent case***
Let $\mathbb{H}$ *be a finite hypothesis set. Then, for any* $\delta > 0$*, with probability at least* $1 - \delta$*, the following inequality holds:*

$$\forall h \in \mathbb{H}, \quad R(h) \leq \hat{R}(h) + \sqrt{\frac{\log(|\mathbb{H}|) + \log(\frac{2}{\delta})}{2N}} \,. \tag{1.7}$$

Thus, for a finite hypothesis set $\mathbb{H}$,

$$R(h) \leq \hat{R}(h) + O\left(\sqrt{\frac{\log_2(|\mathbb{H}|)}{N}}\right). \tag{1.8}$$

The sample complexity required in this case is,

$$N \geq \frac{1}{2\epsilon^2}\left(\log|\mathbb{H}| + \log\frac{1}{\delta}\right) \,. \tag{1.9}$$

Several remarks similar to those made on the generalization bound in the consistent case can be made here: a larger number of training samples N guarantees better generalization, and the bound increase with the size $|\mathbb{H}|$, but only logarithmically. But, here, the bound is a less favorable function of $\frac{\log_2|\mathbb{H}|}{N}$; it varies as the square root of this term. This is not a minor price to pay: for a fixed $|\mathbb{H}|$, to attain the same guarantees as in the consistent case, a quadratically larger labeled samples is needed.

We note that the bound suggests seeking a trade-off between reducing the empirical error versus controlling the size of the hypothesis set: larger hypothesis set is penalized by the second term but could help reducing the empirical error, that is the first term. But, for a similar empirical error, it suggests using a small hypothesis set. This can be viewed as an instance of the so-called *Occam's razor* principle[1] which states that: all other things being equal, a simpler (smaller) hypothesis set is better [305].

This concludes our introduction of the PAC-learning framework. We have seen during this short review the definition of learnability which indicates if an algorithm has learned to do some task. We investigated as well the number of training samples required for a learning algorithm to achieve some predefined accuracy. We have seen

[1] *Occam's razor* principle, after William of Ockham, a 14th-century English Logician, is a problem-solving approach that, when presented with competing explanations, a short explanation (that is, a hypothesis with short length) tends to be more valid than a long explanation. In computational learning theory context, Theorem 1.1 and Theorem 1.2 provide a justification of such principle [305, 385].

also the relation between the generalization error and the empirical error as a function of the size of the hypothesis space $\mathbb{H}$ and the number of training samples N. This relation suggests that in order to obtain better generalization error, it is better to choose small hypothesis set and use large number of training samples.

As we mentioned in the beginning of this section, PAC-learning framework is a theoretical learning framework that provides learning guarantees for finite hypothesis sets. However, in practice, we mostly deal with infinite hypothesis set such as the set of all hyperplanes. While this framework remains theoretical, it provides us with general guidelines to build good learning algorithms. In the case of infinite $\mathbb{H}$, other learning frameworks can be required to provide learning guarantees. We can mention Vapnick and Chervonenkis learning framework that introduces the concept of Vapnik–Chervonenkis dimension [438], also known as VC-dimension which is covered in [305, 385].

As we saw earlier, the generalization of a set of hypothesis $\mathbb{H}$ depends on two main aspects: the number of training samples and the size of the hypothesis set. In practice, we usually have a fixed number of training samples. The only variable that we can control is $|\mathbb{H}|$. Now, given this fixed number of samples, how one can choose the right size of a hypothesis set $|\mathbb{H}|$? In the following, we address such issue by introducing the concept of *regularization*.

1.1.3 Empirical Risk Minimization with Inductive Bias

We discuss here some model selection and algorithmic ideas based on the theoretical results presented in the previous section. Let us assume an i.i.d. labeled training set $\mathbb{S}$ with N samples and denote the empirical error of a hypothesis h on $\mathbb{S}$ by $\hat{R}_{\mathbb{S}}(h)$ to explicitly indicate its dependency on $\mathbb{S}$.

In the following, we show that an Empirical Risk Minimization algorithm can result in the hypothesis that best fits the training data but lacks the generalization aspect. As a possible solution, we present *hypothesis selection* approaches that, while seeking a hypothesis that minimizes the ERM, it prevents overfitting the training data and promotes generalization for unseen situations [305, 385]. Such hypothesis selection methods are known under the name of *inductive biases* [305, 385, 299, 159].

1.1.3.1 Inductive Bias

Empirical Risk Minimization algorithm (ERM), which only seeks to minimize the error on the training samples [305]

$$h_{\mathbb{S}}^{ERM} = \arg\min_{h \in \mathbb{H}} \hat{R}_{\mathbb{S}}(h) \ , \tag{1.10}$$

might not be successful, since it disregards the complexity term of $\mathbb{H}$ [305]. In practice, the performance of the ERM algorithm is typically poor, particularly when using limited training data, since the learning algorithm may fail to find a hypothesis that is able to generalize well for data outside the training set [305, 385, 299, 159]. This situation is known as *overfitting*. Additionally, in many cases, determining the ERM solution is computationally intractable. For example, finding a linear hypothesis with smallest error on the training samples is NP-hard (as a function of the dimension of the space) [305].

A common solution to the overfitting issue of ERM algorithm is to apply the ERM learning rule over a restricted search space [385]. Formally, a set of predictors $\mathbb{H}$ is chosen in advance before seeing the data. Such prior restrictions are often called an *inductive bias*. Since the choice of such restrictions is determined before the learning algorithm sees the training data, it should ideally be based on some prior knowledge about the problem in hand. The hope is that the learning algorithm will search for a hypothesis such that when the ERM predictor has good performance with respect to the training data, it is more likely to perform well over the underlying data distribution.

Inductive bias of a learning algorithm, also known as *learning bias*, can be defined as the set of assumptions that the learner uses to predict correct outputs given inputs that have not been seen before [299]. Therefore, the aim of inductive bias is to promote the generalization. A *bias* refers to any basis for choosing a generalization hypothesis over another, other than strict consistency with the observed training instances [299, 435]. A fundamental question is raised here: what kind of assumptions and over which hypothesis classes an ERM learning algorithm will not result in an overfitting?

In machine learning, different inductive biases can be found [300, 299, 435] including:

- Factual knowledge about the domain and the training data.
- Maximum margin when attempting to separate two classes. The assumption behind is that distinct classes tend to be separated by wide boundaries. Such assumption is behind support vector machines [93].
- Minimum features: unless there is a good justification that a feature is useful, it should be deleted. This is the assumption behind feature selection approach [184, 222, 53].
- Manifold assumption and nearest neighbors: assumes that most of the samples in a small neighborhood in representation space belong to the same class [71]. Given a sample with an unknown class label, guessing that it belongs to the same class as the majority in its immediate neighborhood is a direct application to such assumption. k-nearest neighbors algorithm [222, 305, 385] is based on such assumption.
- Minimum description length and the bias toward simplicity and generality: the minimum description length principal (MDL) is a formalization of *Occam's razor* principle in which the best hypothesis for a given set of data is the one that leads to the best compression of data. This paradigm was introduced in [362] and it is an important concept in information theory and computational learning theory [174]. Considering that any set of data can be represented by a string of symbols from a finite alphabet, the MDL principle is based on the intuition that any regularity in a given set of data can be used to compress it, i.e., such data can be described using fewer symbols than needed to describe the data literally [175]. In inductive and statistical inference theory, such concept promotes the idea that all statistical learning is about finding regularities in data, and the best hypothesis to describe the regularities in data is also the one that is able to compresses the data the most. Theorem 1.1 and Theorem 1.2 are based on such concept where the notion $\log_2(|\mathbb{H}|)$ is used as a description language to measure the length of a hypothesis set. Such theorems suggest that having two hypotheses sharing the same empirical risk, the true error of the one that has shorter description

can be bounded by a lower error value. Seeking a hypothesis that best fits the data while keeping its complexity low is known as *regularization*. This aspect is described more in details in the next section.

A bias needs to be justified in order to be used to constrain the hypothesis search [299]. While inductive biases can help preventing overfitting the training data, strong biases can prevent learning from data [385]. Therefore, a tradeoff is required.

In the following, we present one of the well known and well studied inductive biases which based on preferring hypothesis with low complexity [184, 385, 305]. Such bias is justified theoretically as well (Sec.1.1.2).

1.1.3.2 Example of Inductive Bias: Preference of Hypothesis Sets with Low Complexity (Regularization)

While guarantees of Theorem 1.2 and Theorem 1.1 hold for finite hypothesis sets, they already provide some useful insights for the design of learning algorithms. Similar guarantees hold in the case of infinite hypothesis sets [305]. Such results invite us to consider in a learning context two terms: the empirical error and a complexity term, which here are a function of $|\mathbb{H}|$ and the sample size N.

Structural Risk Minimization learning paradigm (SRM) comes to alleviate the overfitting issue of the ERM learning paradigm. While the ERM considers only the empirical error, the SRM considers the empirical error and the hypothesis set complexity which are both involved in bounding the generalization error. SRM consists in considering an infinite sequence of hypothesis sets with increasing sizes [305]

$$\mathbb{H}_0 \subset \mathbb{H}_1 \subset \cdots \mathbb{H}_t \cdots \ , \tag{1.11}$$

and find the ERM solution h_t^{ERM} for each $\mathbb{H}_t$. The selected hypothesis is the one among the h_t^{ERM} solutions with the smallest sum of the empirical error *and* a complexity term $complexity(\mathbb{H}_t, N)$ that depends on the size (or more generally the capacity, that is, another measure of the richness of $\mathbb{H}$) of $\mathbb{H}_t$, and the sample size N [305]

$$h_{\mathbb{S}}^{SRM} = \arg\min_{\substack{h \in \mathbb{H} \\ t \in \mathbb{N}}} \hat{R}_{\mathbb{S}}(h) + complexity(\mathbb{H}_t, N) \ . \tag{1.12}$$

Fig.1.1 illustrates the SRM. While SRM benefits from strong theoretical guarantees, it is typically computationally expensive, since it requires to determine the solution of multiple ERM problems [305].

Instead of performing an exhaustive search among different hypothesis sets $\mathbb{H}_t$ with increasing sizes $|\mathbb{H}_t|$, an alternative family of algorithms is based on a more straightforward optimization that consists in minimizing simultaneously the sum of the empirical error *and* a regularization term that penalizes the complexity of the hypothesis that belongs to a fixed hypothesis set $\mathbb{H}^*$. Therefore, it is natural to choose a hypothesis set $\mathbb{H}^*$ with large size. One way to measure the complexity of a hypothesis is by counting its number of parameters. Let us consider $\boldsymbol{w}_h \in \mathbb{R}^m$ the set of parameters of a linear hypothesis $h \in \mathbb{H}^*$. Therefore, hypothesis with few parameters are less complex than hypothesis with more parameters. Given that the optimal number of parameters is usually unknown in practice, a hypothesis set with high complexity is usually used. However, such hypothesis is more likely to overfit the data, therefore fails

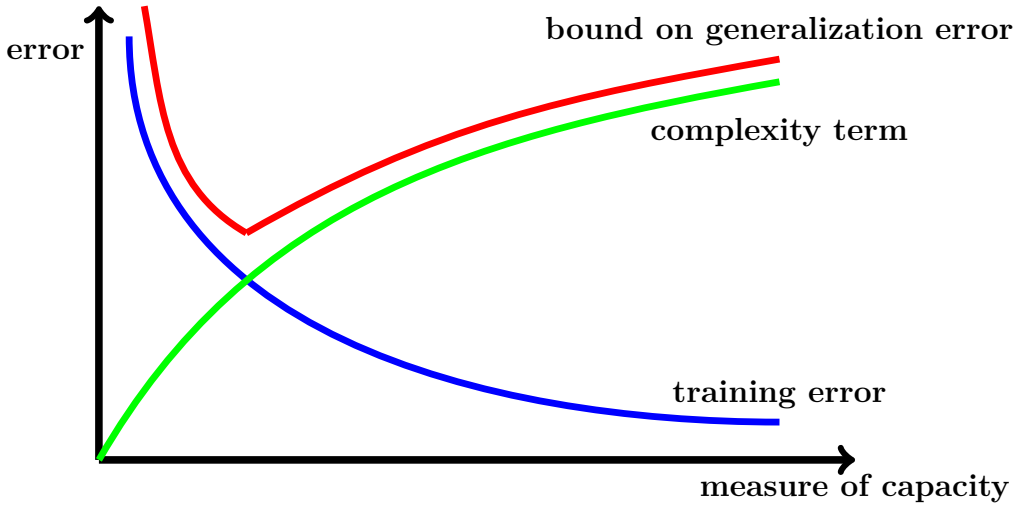


Fig. 1.1 Illustration of structural risk minimization (SRM). The plots of three errors are shown as a function of a measure of capacity. As the size or capacity of the hypothesis set increases, the training error decreases, while the complexity term increases. SRM (shown in red) selects the hypothesis minimizing a bound on the generalization error, which is a sum of the empirical error, and the complexity term. (Reference: [305])

to generalize. An intuitive solution to this issue is to constrain the search algorithm to find a hypothesis h with $\boldsymbol{w}_h$ that has only few non-zero entries. Setting a subset of the parameters to zero is a way of omitting them, i.e., reducing the complexity of the hypothesis since now it has less non-zero parameters. This is known as sparsity and it is motivated from signal approximation and compressed sensing domain [184]. In practice, many classes of data are sparse [184], which means that only few components of the input data are relevant. Hence, sparse parameters are needed. We note that sparsity and feature selection are two related subjects [184].

A straightforward way to constrain the search algorithm to find sparse solutions is to count the non-zero entries of the parameters $\boldsymbol{w}_h$ which can be done using L_0 parameter norm

$$L_0(\boldsymbol{w}_h) = \|\boldsymbol{w}_h\|_0 = \sum_{i=1}^{m} 1_{w_{i_h} \neq 0} \ , \tag{1.13}$$

which counts the non-zero elements of $\boldsymbol{w}_h$ [67].

In terms of computational complexity, constraining the search algorithm using L_0 norm was shown to be NP-hard [312]. It was shown in [67, 109, 184] that L_0 and L_p norm minimization,

$$L_p(\boldsymbol{w}_h) = \|\boldsymbol{w}_h\|_p = \sum_{i=1}^{m} \left(|w_{i_h}|^p\right)^{1/p} \ , \tag{1.14}$$

for $p = 1$, has identical solution under some conditions over $\boldsymbol{w}_h$. Hence, L_0 norm minimization can be relaxed with L_1 norm minimization. However, these conditions may be too strong for practical use. Instead, one may consider the sparse solution

provided by a relaxed problem for a fixed p with $0 < p < 1$ [72, 73, 77, 125, 309]. Nonetheless, L_p norm minimization for $0 < p < 1$ is strongly NP-hard [140]. However, any basic feasible solution of L_0 norm minimization is a local minimizer of L_p norm minimization with $0 < p < 1$ [140]. This is motivated by the fact that local minimizers are easy to certify and compute [140]. Although L_1 is one of the most common norm used to constrain the hypothesis complexity [184], other norms with $p > 1$, can be used [425, 468] such as the popular L_2 norm [429, 305].

To sum up, the regularized optimization problem which is composed of the empirical error and a regularization term, that is typically defined as $L_p(\boldsymbol{w}_h)$, can be written as [305]

$$h_{\mathbb{S}}^{REG} = \arg\min_{h \in \mathbb{H}^*} \hat{R}_{\mathbb{S}}(h) + \lambda L_p(\boldsymbol{w}_h) \ , \tag{1.15}$$

where $\lambda \geq 0$ is a regularization parameter, which can be used to determine the trade-off between empirical error minimization and the model complexity. In practice, λ is typically selected using n-fold cross validation.

Although, in the context of machine learning, regularization is best known for reducing the hypothesis complexity [184, 385], it can go beyond that to reach the inductive bias definition (Sec.1.1.3.1). Therefore, regularization can be defined as any process that allows reducing the generalization error without necessarily reducing the empirical error. This is usually done by introducing prior knowledge that allows narrowing down the hypothesis space to specific subset. This prior knowledge may concern the complexity of the model, the data, or anything related to the task in hand [154].

In the following, we provide a summary of this section about machine learning and generalization.

1.1.4 Summary

In this section, we have seen that learning from data is a crucial part of today's technology. In order to make it work, some important questions must be addressed including what can a machine learn? how many samples are needed to guarantee a better generalization? how to deal with failure of generalization?. In the context of learning theory, we have presented the concept of learnability throughout the PAC learning framework which provides us general guidelines to build good learning algorithms.

We have seen that the generalization error upper bound depends on the number of training samples and the complexity of the hypothesis. Using the ERM learning paradigm to find the best hypothesis that fits the training samples can lead to poor results due to overfitting since ERM takes in consideration only the training samples. A possible and efficient solution to deal with the overfitting issue of the ERM is to restrict the hypothesis search space [385]. For instance, one can choose a priori a hypothesis set $\mathbb{H}^*$ before seeing the training samples. Such priors are known as *inductive bias*. Restricting the search space by picking such hypothesis set independently from data should ideally be done based on some prior knowledge about the problem to be learned. As an example of inductive bias, we have considered the prior that consists in preferring hypotheses with low complexity which is justified theoretically. As we have seen, a hypothesis with low complexity is in favor of generalization. However, such hypotheses

may not be able to reduce the empirical error, i.e., explain the observed data. In the other hand, a hypothesis with high complexity may have low empirical error but it will increase the upper bound of the generalization error leading to overfitting. Therefore, a tradeoff between reducing the empirical error and the hypothesis complexity must be achieved. The search for a hypothesis with low complexity in the context of learning is known as regularization. This definition can be extended to include methods that promote the generalization aspect but without necessarily affecting the hypothesis complexity. We cover in Appendix A.2.1 technical details on the bias-variance tradeoff which is related to the empirical error and the hypothesis complexity tradeoff.

In this thesis, we deal exclusively with neural network models. Therefore, we provide in the next section an introduction to the subject. We provide more technical details on such models in Appendix A.2.2. If the reader is familiar with the neural network field, we recommend skipping toward Sec.1.3 where we present different methods to improve the generalization performance of such models.

1.2 Introduction to Feedforward Neural Networks

Artificial neural networks (ANN) are a particular type of parametrized models in machine learning. We provide in this section an introduction to such models. We cover more technical details on neural networks in Appendix A.2.2.

While providing a presentation of ANN, we highlight also important historical key moments in neural networks origins. Extensive tracking of the history of neural networks and deep learning can be found in [380].

1.2.1 Early History

In 1943, neurophysiologist Warren McCulloch and mathematician Walter Pitts proposed a mathematical model [288] of neurons in the brain based on threshold logic and demonstrated that combined together they can compute logic functions. They modeled a simple neural network through electrical circuit. Their model lacked a learning mechanism which is important to solve artificial intelligent problems.

In 1949, psychologist Donald Olding Hubb proposed a fundamental work [189] about the learning process. His hypothesis states that knowledge and learning in the brain occurs primarily through formation and changes of synapses between neurons, known as synaptic plasticity.

In 1957, psychologist Frank Rosenblat proposed the perceptron [364] model based on the work of Warren Mcculloch, Walter Pitts [288] and Donald O. Hubb [189]. Later, he published a book where he described in depth the perceptrons and their related proofs [365].

After 1967, research in neural networks stagnated after the work of Marvin Minsky and Seymour Papert [296] who showed the limitation of the perceptrons which are unable to solve problems which are not linearly separable such as the simple XOR problem.

In the following, we present a brief and formal description of perceptrons.

1.2.2 Perceptron

A perceptron models one simple neuron. Formally, it is a simple function with an argument that is linear with respect to its input,

$$\hat{y} = f(\boldsymbol{x}) = \phi(\boldsymbol{x} \cdot \boldsymbol{w} + b) = \phi(\sum_{i=1}^{D} x_i w_i + b) \ , \tag{1.16}$$

where $\boldsymbol{x} \in \mathbb{R}^D$ is an input vector, $\boldsymbol{w}$ is a vector of parameters known as weights. b is a scalar parameter known as bias. $\phi(\cdot)$ is called an activation function and is typically nonlinear. If $\phi(\cdot)$ is the Heaviside step function, the perceptron has only two states: 0 or 1 which might indicate two different classes. An illustration of a simple perceptron is depicted in Fig.1.2.

To simplify notations, the weights vector $\boldsymbol{w}$ are extended by adding extra component to represent the bias b. Then, the input $\boldsymbol{x}$ is also extended by adding an additional component with value of 1. From now on, we consider the extended notation unless we state the opposite.

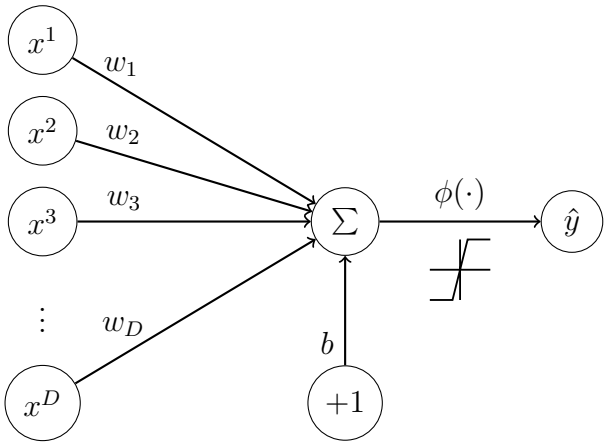


Fig. 1.2 Perceptron model. (Notation: x^i is the i^{th} component of $\boldsymbol{x}$ the same as x_i in Eq.1.16.)

Learning consists in searching for the best weights and bias that make $\hat{y}$ close to the target y. Rosenblatt training algorithm consists in updating the weights by increasing or decreasing $\boldsymbol{w}$ if the output $\hat{y}$ is smaller or greater than the target y as follows,

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - (\hat{y} - y)\boldsymbol{x} \ , \tag{1.17}$$

where $y \in \{0, 1\}$, and $\hat{y}$ is the prediction. This algorithm continues learning as long as the model commits a mistake in classification. It has been shown in [322] that in the case of two linearly separable classes, this algorithm converges in a finite number of iterations. However, in the case where the two classes are not linearly separable, the algorithm never converges.

One perceptron can work only with two classes. It is possible to extend a perceptron to work with more than two classes, i.e., $M > 2$. This can be done using M perceptrons

connected to the same input $\boldsymbol{x}$ but each one has its own weights and bias. By doing so, an output vector is obtained instead of one scalar,

$$\hat{\boldsymbol{y}} : \hat{y}_j = f_j(\boldsymbol{x}), \forall j = 1, \ldots, M \ . \tag{1.18}$$

In this case, the predicted class k is given by the class of the maximum output,

$$k : \hat{y}_k \geq \hat{y}_j, \forall j = 1, \ldots, M \ . \tag{1.19}$$

Fig.1.3 illustrates an example of multiclass perceptron.

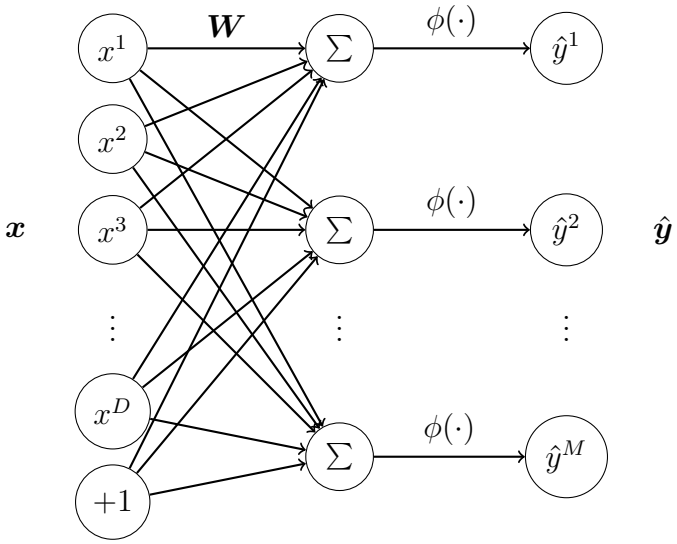


Fig. 1.3 Multiclass perceptron. (Notation: x^i is the i^{th} component of $\boldsymbol{x}$. $\hat{y}^j$ is the j^{th} component of $\hat{\boldsymbol{y}}$).

The training of multiclass perceptron is done by applying the peceptron learning rule described previously for each perceptron. Let us use a matrix notation where $\boldsymbol{W} \in \mathbb{R}^{(D+1)\times M}$ is the weights of the multiclass perceptron. $\boldsymbol{x}$ is vector of $1 \times (D+1)$, $\hat{\boldsymbol{y}} - \boldsymbol{y}$ is a vector of errors with size $1 \times M$, $y_{j=1,\ldots,M} \in \{0, 1\}$ is a vector of labels where 1 indicates the correct class, and $\sum_{j=1}^{M} y_j = 1$ to make sure that every sample belongs to only one class. The update rule can be written as,

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \boldsymbol{x}^\top \cdot (\hat{\boldsymbol{y}} - \boldsymbol{y}) \ . \tag{1.20}$$

In 1957, Frank Rosenblatt implemented a multiclass perceptron in custom-built hardware under the name "Mark I" perceptron with 20×20 inputs and 8 output classes. Preceding this work by many years, in 1951, Marvin Minsky has implemented a hardware neural network based on memory and rewards named "SNARC" (Stochastic Neural Analog Reinforcement Calculator). This machine is considered one of the first pioneering works in the field of artificial intelligence.

In the next paragraph, we present a slightly different approach to train a perceptron based on gradient descent.

1.2.3 Gradient Based Learning

In 1960, Bernard Widrow and Marcian Hoff developed a linear model [459] similar to the perceptron but without the thresholding activation function. This allows using gradient descent method and using the derivatives. They named their model ADALINE for ADAptive LINear Elements. The model is an electrical circuit based on a new circuit called memistor which is a resistor with memory. For training their model, Widrow and Hoff proposed a slightly modified version of the perceptron learning rule. Instead of using the misclassification error, they proposed to use the squared error as an error measure for each sample as follows,

$$\ell(f(\boldsymbol{x}), \boldsymbol{y}) = \frac{1}{2}\sum_{j=1}^{M}(y_j - f_j(\boldsymbol{x}))^2 \ . \tag{1.21}$$

Therefore, minimizing the mean squared error over all the training samples N can be formulated as follows,

$$L(\boldsymbol{W}) = \frac{1}{N}\sum_{i=1}^{N}\ell(f(\boldsymbol{x}^{(i)}), \boldsymbol{y}^{(i)}) \ . \tag{1.22}$$

In this case, $\hat{\boldsymbol{y}}^{(i)} = f(\boldsymbol{x}^{(i)}) = \boldsymbol{x}^{(i)} \cdot \boldsymbol{W}$.

To train the ADALINE model, they propose to use gradient descent as follows,

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \alpha\frac{\partial L(\boldsymbol{W})}{\partial \boldsymbol{W}} \ , \tag{1.23}$$

where α is a learning rate which controls the speed of convergence. This algorithm suggested to move the parameters weights in the direction that decreases the total error $L(\boldsymbol{W})$. This direction is obtained by computing the derivative of the total train loss $L(\boldsymbol{W})$ with respect to the weight vector using the chain rule. Using the linearity of the derivatives of Eq.1.22, we can compute the derivatives of one example $(\boldsymbol{x}, \boldsymbol{y})$ in Eq.1.21, then we take the average over all the samples to obtain $\frac{\partial L}{\partial \boldsymbol{W}}$, the derivatives of the total loss in Eq.1.22. For one training sample, deriving the loss in Eq.1.21 with respect to a parameter weight gives,

$$\frac{\partial \ell}{\partial W^{ij}} = \frac{\partial \ell}{\partial \hat{y}^j} \cdot \frac{\partial \hat{y}^j}{\partial W^{ij}}, \quad \forall i = 1, \ldots, D+1, \forall j = 1, \ldots, M \quad , \tag{1.24}$$

where,

$$\frac{\partial \ell}{\partial \hat{y}^j} = \frac{\partial \frac{1}{2}(\hat{y}^j - y^j)^2}{\partial \hat{y}^j} = \hat{y}^j - y^j \Rightarrow \frac{\partial \ell}{\partial \hat{y}} = \hat{y} - y \ . \tag{1.25}$$

The notation x^i is the i^{th} component of $\boldsymbol{x}$, $\hat{y}^j$ is the j^{th} component of $\hat{\boldsymbol{y}}$, and W^{ij} is the component at the i^{th} row and the j^{th} coloumn of $\boldsymbol{W}$.
In the case where there is no activation function,

$$\frac{\partial \hat{y}^j}{\partial W^{ij}} = \frac{\partial\left(\sum_k x^k W^{kj}\right)}{\partial W^{ij}} = x^i, \quad \forall i = 1, \ldots, D+1, \forall j = 1, \ldots, M \ . \tag{1.26}$$

This gives the delta rule learning algorithm,

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \alpha \frac{\partial L}{\partial \boldsymbol{W}} = \boldsymbol{W} - \alpha \boldsymbol{x}^\top \cdot (\hat{\boldsymbol{y}} - \boldsymbol{y}) \; . \tag{1.27}$$

One can see that the delta rule (Eq.1.27) is similar to the perceptron learning rule (Eq.1.20) except for the learning rate α.

In the case where there is a differentiable activation function $\hat{y}^j = \phi(h^j = \sum_k x^k W^{kj})$, Eq.1.26 can be developed using chain rule as follows,

$$\frac{\partial \hat{y}^j}{\partial W^{ij}} = \frac{\partial \hat{y}^j}{\partial h^j} \cdot \frac{\partial h^j}{\partial W^{ij}} \tag{1.28}$$

$$= \frac{\partial \phi(h^j)}{\partial h^j} \cdot \frac{\partial \left(\sum_k x^k W^{kj} \right)}{\partial W^{ij}} \tag{1.29}$$

$$= \frac{\partial \phi(h^j)}{\partial h^j} \cdot x^i \; . \tag{1.30}$$

Therefore, the delta rule in this case is,

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \alpha \boldsymbol{x}^\top \cdot \nabla_{\boldsymbol{h}} \phi(\boldsymbol{h}) \cdot (\hat{\boldsymbol{y}} - \boldsymbol{y}) \; . \tag{1.31}$$

In the case of the perceptron, which uses the Heaviside step function, $\nabla_{\boldsymbol{h}} \phi(\boldsymbol{h})$ is not defined at zero and it is equal to zero everywhere else which makes the application of the delta rule on the perceptron impossible. This led to the use of differentiable functions such as the sigmoid functions. Appendix A.2.2.2 covers more details about other activation functions.

In the following, we present the extension of the perceptron to multilayer perceptron which is a critical change in the history of neural networks. We briefly highlight the historical reasons for such a major change.

1.2.4 Multilayer Perceptron and Representation Learning

Although perceptrons seemed promising at the beginning, it was quickly shown that they could not be trained to separate every type of classes. In 1969, Marvin Minsky and Seymour Papert published their book "Perceptrons: An Introduction to Computational Geometry" [296] which put an end to perceptrons. In this book, the authors pointed out fundamental limitations of the perceptron. For instance, a single perceptron can not solve an XOR problem. Moreover, they conjectured, mistakenly, that similar results would be found when using multilayer perceptrons. This book caused a significant decline in interest and funding of neural networks research. This led to an abandonment of connectionism which was the other part of Artificial Intelligence with concurrence with symbolic reasoning which Minsky and Papert were part of. This major criticism participated in starting the AI winter[2]. Three years later, Stephen Grossborg published a series of work introducing neural networks modeling XOR [172]. In 1987, Minsky and Papert reprinted their book with the name "Perceptrons - Expanded Edition"

[2]Between 1974-1980: AI winter is a period of reduced funding and interest in artificial intelligence research. At this period, AI has experienced several hype cycles, disappointment and criticism, followed by funding cuts. Years later, interest into AI was back.

[297] where some errors of the original book were shown and corrected. Despite this controversy, the reprinted version contains a handwritten dedication to Frank Rosenblatt who did not live to see it. As a side note, Minsky and Rosenblatt knew each other since adolescence. They studied at the same high school with one different year. However, they pursued different paths in AI research. While Minsky promoted symbolism, Rosenblatt promoted connectionism and learning. More on this controversy can be found in [326].

Despite this pessimism toward perceptrons, the book of Minsky and Papert provided new insights and research directions to improve them. The main result is that the perceptron fails in many recognition tasks not because of the learning algorithm but because of its lack of represention the required knowledge about the task to be solved. The authors stated that no machine can learn to recognize an object unless it possesses, at least potentially, some scheme for representing the object. In the case of the perceptron, Minsky and Papert pointed out that if there is a layer of simple perceptron-like hidden units which can recode the input pattern into an internal representation, there is always a recoding in this hidden representation that can support any required mapping from the input to the output. We note that at that time, few networks use this technique such as MADALINE [463] which has a different training algorithm than the perceptron rule referred to as MRII algorithm for MADALINE Rule II. MADALINE consists basically of two sequential layers where each one is composed of multiple ADALINE neurons [459] that are followed by a threshold function. The use of threshold functions prevents MADALINE from using gradient based training algorithms. MRII algorithm uses instead the principle of minimal disturbance [215] where the network parameters are disturbed whenever there is a mistake in the output. MADALINE is an extension of the two-layer network of Ridgway [215] which is based on two layers: an adaptive layer that contains multiple ADALINE neurons followed each by a threshold function; then a fixed logic layer that takes the output of the previous ADALINEs as input to provide the final output. The logic layer is a simple logic function: "AND", "OR", and majority vote. MADALINE goes further by implementing such logic functions using ADALINE neurons and provides a learning algorithm for a multilayer network. Earlier to that, particularly in the beginning of the 60′s, we find Gamaba machines [137] as a two perceptrons machine where the output of the first one is fed to the second one. The main issue at the time is that there was no strong learning algorithm to learn networks with hidden units. Moreover, the computation power required by such networks exceeds what was available at the moment. Neural networks field had to wait until the arrival of the backpropagation algorithm (Appendix A.2.2.1) for training multilayer networks. An illustration of a multilayer perceptron is depicted in Fig.1.4.

We discuss in Appendix A.2.2 more details on neural networks including backpropagation algorithm (Sec.A.2.2.1), nonlinear activations (Sec.A.2.2.2), universal approximation properties and depth (Sec.A.2.2.4), and other neural architectures (Sec.A.2.2.5).

Neural network domain kept struggling in the late of 90′s and the beginning of 2000′s for many reasons including the lack of data, practical issues in optimization algorithms, and most importantly the lack of computation power which slowed down research. It was until around 2006 that neural network field entered a new era which moved neural network models from shallow, i.e., few layers, to deep, i.e., many layers, which led to a spiking success in the history of neural networks. Such success has

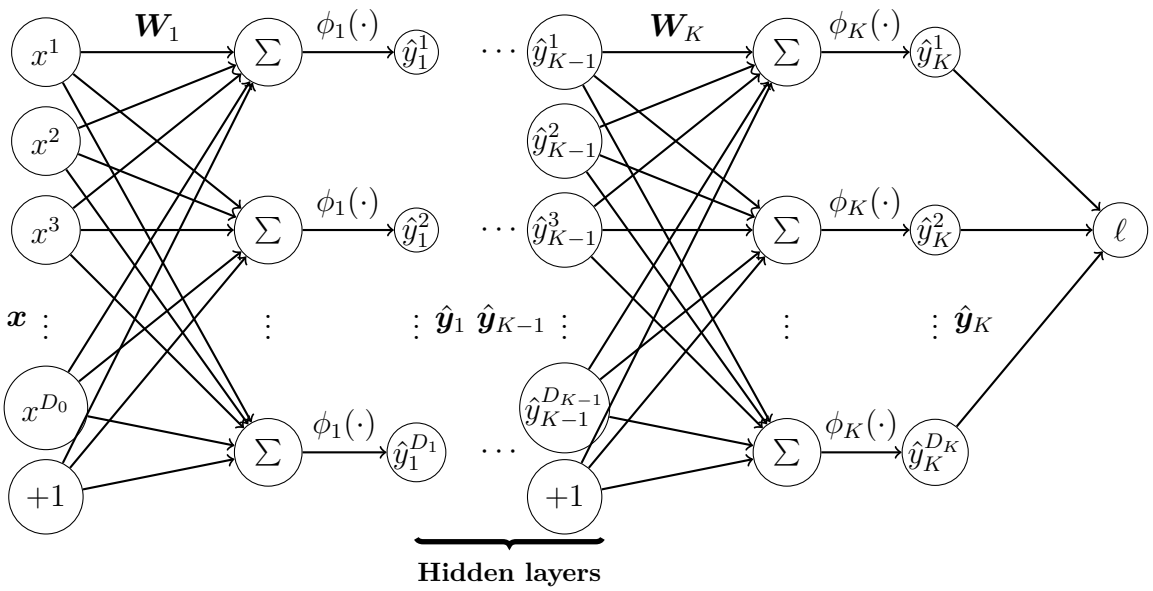


Fig. 1.4 Multilayer perceptron. (Notation: $D_k, k = 1, \cdots, K$ is the dimension of the output of the layer k. $D_0 = D$ is the dimension of the input $\boldsymbol{x}$ of the network. $D_K = M$ is the dimension of the output $\hat{\boldsymbol{y}}$ of the network, i.e., M. x^i is the i^{th} component of $\boldsymbol{x}$. $\hat{y}_k^j$ is the j^{th} component of the output representation $\hat{\boldsymbol{y}}_k$ at the layer k).

attracted the research community to start working again on such models. Started in the beginning of 2000′s, the expression "*deep learning*"[3] was coined to broadly describe neural based models that use many layers to learn hierarchical representations. Aside from the advances in optimization algorithms that pushed deep learning field forward, such success could not be done without: the modern computational power that speeds up drastically the training and inference of deep models and the availability of massive supervised data in many domains such as computer vision, natural language processing, and voice recognition. In the next paragraph, we present a modern version and current advances in deep learning field.

1.2.5 Deep Learning: from Late 60′s to Today

Deep learning, also known as deep structural learning or hierarchical learning, is part of a broader family of machine learning algorithms based on learning data representations where learning can be achieved through supervised, semi-supervised or unsupervised approaches.

[106] defines deep learning as a class of machine learning algorithms that: • use a cascade of multiple layers of nonlinear processing units for feature extraction and transformation. In most cases, each layer uses the output of the previous layer as input. • learning such layers is done in a supervised and/or unsupervised fashion. • each layer

[3]The term "*deep learning*" was introduced to the machine learning community by Rina Dechter in 1986 [102], and to artificial neural networks by Igor Aizenberg [8, 153] in 2000.

is seen as an abstraction level of representation. Stacking the layers forms a hierarchy of concepts that are built up from lower levels toward more abstract ones [47, 44]. The assumption underlying distributed representations is that observed data are generated by the interaction of layered factors [197, 47]. Deep learning adds the assumption that such layers of factors correspond to levels of abstraction or composition. Varying number of layers and layer sizes can provide different degrees of abstraction [47].

[380] introduce the notion of *credit assignment path (CAP)* to define what is a *shallow* and *deep* model. The CAP is defined as a chain of transformations from the input to the output to describe potentially causal connections between the input and the output. For instance, feedforward neural networks have a CAP with a depth equals to the number of hidden layers plus one (the output layer). For recurrent networks (Appendix A.2.2.5), the CAP depth is potentially unlimited [380]. Although there is no universal agreement upon threshold of CAP depth to divide shallow learning from deep learning, most researchers agree that deep learning involves a CAP depth greater than 2 which has been shown to be a universal approximation in a sense that it can emulate any function (Appendix A.2.2.4).

Neural networks trained by the *Group Method of Data Handling* (GMDH) [219, 220] were perhaps the first deep learning models based on feedforward multilayer perceptron. Although, the units of their networks may have polynomial activation functions implementing *Kolmogorov-Gabor polynomials* [218] to introduce nonlinearity. Such activation functions are different than the other widely used nowadays. Their deep networks are incrementally trained then pruned. [218] describe a network with 8 layers. A presentation of *Kolmogorov-Gabor polynomials* can be found in [449].

Aside from deep GMDH networks, the *Neocognitron* [131], by Fukushima, was maybe the first artificial neural network that deserves the name *deep*. It was the first to incorporate the neurophysiological insights about the visual cortex found around the 60′s [460, 213] into a learning framework. *Neocognitron* introduced convolution layers composed of a set of convolution operators parametrized with a weights, under the form of rectangular matrix, that are duplicated over the 2D input through a shifting process. The same model introduced subsampling[4], known also as downsampling, to promote a certain insensitivity to small shifts in the 2D input image. *Neocognitron* is very similar to the feedforward, gadient-based, and backpropagation-based convolutional neural networks. However, Fukushima did not set the weights by backpropagation but by local Winner-Take-All-based unsupervised learning rule[5] [134] or by pre-wiring. Therefore, deep learning issues did not matter (Appendix A.2.2.1). For down-sampling, Fukushima used spatial averaging [132, 133] instead of max-pooling[6] that is well known in modern convolutional networks.

[4] A subsampling layer is an average pooling layer performs down-sampling by dividing the input into rectangular pooling regions and computing the average values of each region.

[5] Winner-Take-All (WTA) is a computational principle applied in computational models of neural networks by which neurons in a layer compete with each other for activation. In a classical setup, only the neuron with the highest activation stays active while all the other neurons shut down. However, other variations may allow more than one neuron to be active. In the theory of artificial neural networks, WTA networks are a case of competitive learning in recurrent neural networks. Output nodes in the network mutually inhibit each other, while simultaneously activating themselves through reflexive connections [173, 328].

[6] A max-pooling layer is a layer that performs down-sampling by dividing the input into rectangular pooling regions and computing the maximum values of each region.

In 1989, backpropagation algorithm (Appendix .A.2.2.1) was successfully applied to *Neocognitron*-like model with weights sharing and convolutional layers [255–257] (Fig.1.5). The purpose of the application is to recognize handwritten zip codes on mail. Their algorithm required 3 days of training.

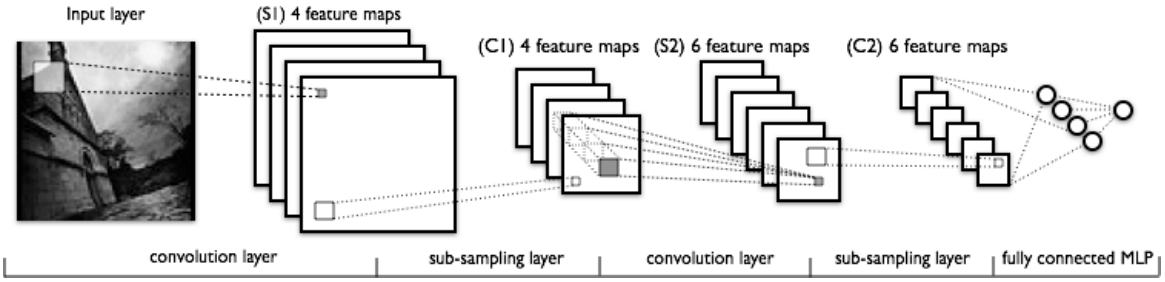


Fig. 1.5 Architecture of *lenet* convolutional network [257] which is composed of: two convolutional layers followed by a pooling layer each; followed by two dense layers. (Source: Deep Learning Tutorial: http://deeplearning.net/tutorial/lenet.html).

Inspired by *Neocognitron*, *Cresceptron* was born in 1992 [452] which adapts its topology during training. Instead of using local subsampling or WTA methods [132, 377], the *Cresceptron* introduced, for the first time, max-pooling layers. A more complex version of *Cresceptron* was proposed which includes blurring layers to improve object location tolerance [453].

By the late of the 80′s, experiments had shown that traditional deep forward networks or recurrent networks (Appendix A.2.2.1) are hard to train using backpropagation. It was until 1991, that this issue was understood and the reason for such issue is now known by the name of the vanishing/explosion of the gradient (Appendix A.2.2.1). Unsupervised pre-training was a major tool to deal with such an issue in feedforward networks and recurrent networks as well [23, 378]. Long Short-term memory (LSTM) recurrent networks helped as well avoiding gradient problems and allow training very deep learning models [206, 380].

Closely related works to [23] have appeared in post-2000. For instance, in 2006, [198, 194, 45] showed that many layered feedforward networks could be effectively trained by pre-training one layer at a time, treating each layer as an unsupervised restricted Boltzmann machine, then fine-tune the whole network using supervised backpropagation. This is referred to as training of deep belief networks. Similar ideas were proposed in [361, 360] which helped training deep networks.

For a long time, the slow computation power stepped in the way of training deep neural models [380, 106]. Advances in hardware renewed the interest in such domain. In 2009, Nvidia[7] was involved in what was called the "*big bang*" of deep learning. Their GPUs have speed up drastically the training of deep networks and allowed going deeper in a reasonable time. In particular, GPUs are well-suited for the matrix/vector operations involved in machine learning [408, 86, 350]. Specialized hardware and optimization algorithms can be used for efficient processing [418].

[7]Nvidia Corporation is an American technology company that designs graphics processing units (GPUs) for the gaming, cryptocurrency, and professional markets, as well as system on a chip units (SoCs) for the mobile computing and automotive market. Website: http://www.nvidia.com

Aside from the increase of computational power that speeds up training deep networks and the increase of the available supervised data in certain applications, which make training deep architectures practical [380], there have been, in the last few years, many advances in optimization approaches that helped: • speeding the training by improving gradient descent approach such as momentum [414, 342], Adagrad [114, 101], Adadelta [477], Adam [237, 111], and possibly the use of second order optimization such as Hessian-free optimization [307, 284, 285]. • improving the generalization and avoid overfitting such as dropout [403, 404] and batch normalization [216]. • avoiding the vanishing of the gradient by introducing new activation functions that do not saturate such as rectifier [310, 149, 158] (Appendix A.2.2.2).

Nowadays, different deep neural models including feedforward and recurrent networks have been successfully applied to different tasks including: computer vision [88, 244], speech recognition [106, 380], natural language processing and machine translation [400, 387], visual art processing[8] [420, 395], recommendation systems [450, 437], image restoration [433, 484], social network filtering [315], bioinformatics and drug design [79, 374], where they have produced results comparable and in some cases superior to human experts [88, 244].

In the last four years, generative models based on neural networks have been a hot topic particularly using Generative Adversarial Network (GANs) [156] (Appendix A.2.2.5) which is an area of deep learning that is growing rapidly. Hinton, one of the founders of neural networks, has recently proposed a new architecture named "*capsules*" [370, 200] in an attempt to solve a convolutional network issue related to its lack of taking in consideration the spatial relations between the parts of an object. Deep reinforcement learning is another breakthrough of deep learning models that is making a big step to improve artificial intelligence and get computers to learn like humans, without explicit instructions [16, 12, 303]. Many research teams focus now on developing systems capable of learning how to play *ATARI* video games using only pixels as data input [302, 210]. Autonomous vehicles driving is also an area where deep reinforcement learning is making progress [128, 78, 474]. Deep reinforcement learning has been used to learn *Go*[9] game well enough to beat a professional *Go* player [391].

[380, 106] provide a detailed and extensive presentation of the history and applications of deep learning methods.

We mention that most of the success of neural networks today is due to the depth of their architectures and not the width of their layers [154] (Fig.1.6). We cover this aspect with more details in Appendix A.2.2.4. [330] provide a discussion on the question of which one to use: deep or wide learning mechanism?

[8] "*DeepDream*" is a computer vision program created by Google engineer Alexander Mordvintsev which uses a convolutional network to find and enhance patterns in images via algorithmic pareidolia, thus creating a dream-line hallucinogenic appearance in the deliberately over-processed image.

[9] *Go* is an abstract strategy board game for two players, in which the aim is to surround more territory than the opponent.

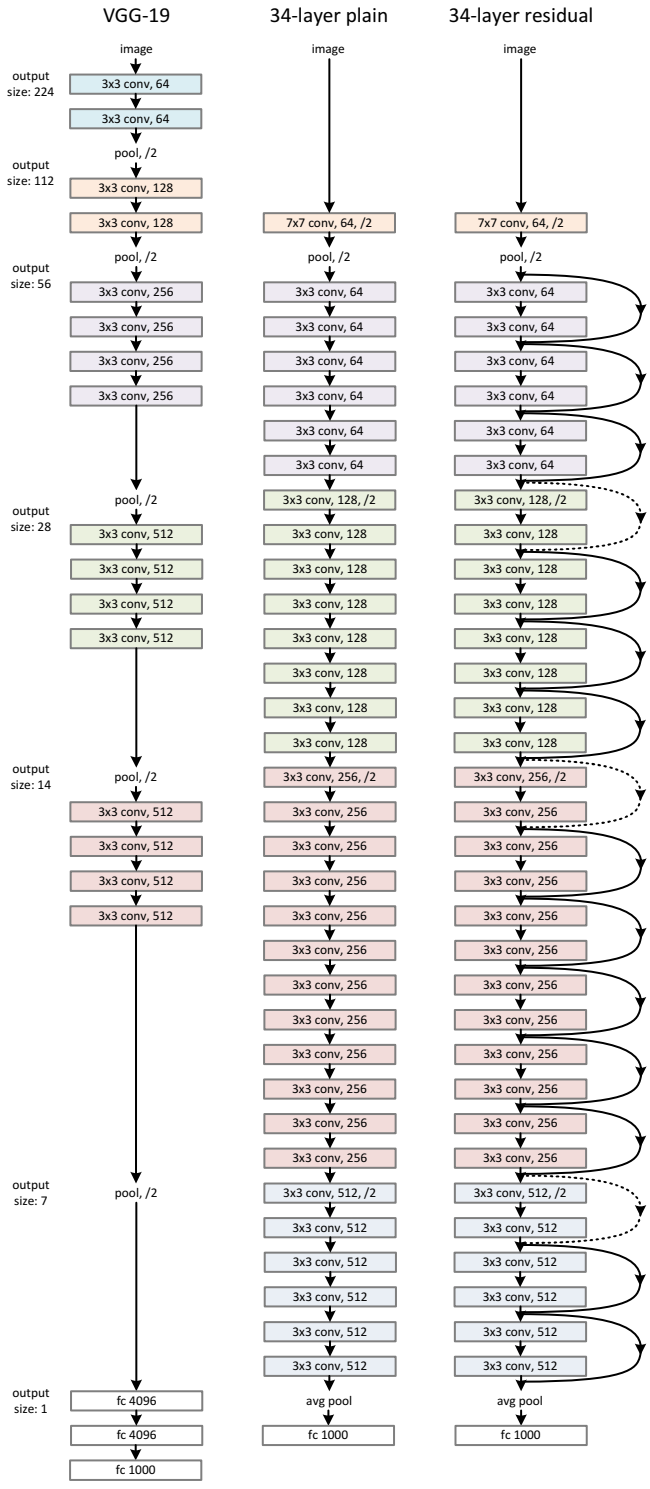


Fig. 1.6 Example of *Deep Residual* network architectures for image recognition. *Left*: the VGG-19 model [396]. *Middle*: a plain network with 34 parameter layers. *Right*: a residual network with 34 parameter layers. (Credit: [187])

Deep Learning Issues and Criticism

Although deep learning models have achieved high performance in many applications, they still display problematic and questionable behaviors such as classifying unrecognizable images as belonging to a familiar category of ordinary images or misclassifying small perturbed images that have been correctly classified [151, 314, 422]. As deep learning models move from the laboratory to the real world, such misbehavior can cause a serious security threat. For instance, an attacker can add a specific type of noise to an image so that the human eye does not notice it, but it will cause the neural network to produce a completely wrong prediction. Such attacks are known by "*adversarial attacks*". A neural network can be trained to minimize its error over such attacks [422, 157]. Another serious and old issue that is known to the research community is that neural networks memorize chunks of training data [181, 15, 68]. This phenomenon is not well-understood. Language models are probably the most vulnerable type of models at the moment while it is harder over images [68]. This shows again how deep learning models are vulnerable to information leakage. Using particular search algorithms, an attacker can retrieve sensitive data such as text messages, emails, medical data, etc. An important work that changed the research community understanding of the generalization aspects in deep learning models has suggested that "brute-force memorization" may be part of an effective learning strategy for deep neural networks on real data, implying that generalization and memorization are not necessarily opposed [482]. However, Not all the community seems to agree that neural networks memorize the training data [245]. There may be ways to get around the memorization issue. The researchers recommend developers to use "*differential privacy learning algorithms*"[10] [1, 331] which are currently applied in big companies that often deal with private information of the users which is mostly textual. Another way is to scramble and randomize private information so that it is difficult to reproduce.

Neural network field has seen many criticism, and still do, since its birth [296, 297]. The main criticism concerns the lack of theoretical foundations. Deep learning models are often looked at as a "*black box*", with most confirmations are done empirically rather than theoretically. Modern critics such as Gary Marcus, pointed out that deep learning should be looked at as a step towards realizing strong artificial intelligence, and not as an entire solution [282, 283]. Despite the power and the success of deep learning methods, they still lack much of the functionalities needed for attaining the goal of artificial intelligence such as their lack of representing causal relationships, logic inference, and integrating abstract knowledge [282, 283]. Hopefully, such criticism will lead to improvements in neural networks domain.

1.2.6 Summary

We have presented in this section a brief, technical, and historical introduction to neural networks from the early 40′s to its modern version under the name of *deep learning*. We have presented also some of the last advances in the field.

[10] In cryptography, *Differential privacy* is a mathematical definition for the privacy loss that results to individuals when their private information is used in the creation of a data product. It aims to provide means to maximize the accuracy of queries from statistical databases while minimizing the chances of identifying its records [117, 116, 118].

As we saw earlier, adding more hidden layers to encode the input is one possible way to allow perceptrons to perform better on different tasks. This idea has led to what is known today as deep models, i.e., models composed of many hidden layers. This makes learning representations within neural networks an important aspect to obtain a better generalization error [51, 46]. However, increasing the depth of a network leads to an increase of the number of its parameters which requires a large number of training data to fit the model and avoid overfitting. Unfortunately, in practice, one usually deals with small datasets which causes deep models to overfit. Different approaches were proposed in the literature to deal with such issue using variant regularization methods. We present the most common used methods in the following section (Sec.1.3). As it is impractical to cite most the approaches in this chapter, we refer the reader to [154] where most neural networks regularization techniques are covered, while we continue our discussion of neural networks in Appendix A.2.2.

1.3 Improving Neural Networks Generalization

Training neural networks and particularly deep architectures is known to be difficult [148, 154]. Moreover, deep architectures are known to overfit the data particularly when using few training samples. Regularization is one of the commonly known solutions to deal with the overfitting issue by providing tools to reduce the generalization error. We describe in this section selected methods to regularize neural networks.

In Sec.1.1.3, we provided the definition of the regularization as any process that allows reducing the generalization error without necessarily reducing the error over the training samples. In machine learning, and for long time, regularization consists generally in reducing the model complexity using L_p parameters norm penalty. However, with the advances in machine learning, other approaches have been proposed to reduce the generalization error. Nevertheless, such methods do not necessarily reduce the model complexity. We attempt in this section to separate these two approaches, even though the frontier between them is still vague:

- Explicit regularization where the aim is to explicitly reduce the model complexity.
- Implicit regularization where the aim is to reduce the generalization error. However, the model complexity may or may not be reduced.

1.3.1 Explicit Regularization: Explicit Complexity Reduction

Many regularization approaches are based on limiting the capacity of the models by adding a parameter norm penalty to the training objective function J. Let us denote the regularized objective function by $\widetilde{J}$

$$\widetilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{\theta}), \quad \Omega(\boldsymbol{\theta}) = \frac{1}{p}\|\boldsymbol{\theta}\|_p^p \,. \tag{1.32}$$

In this context, we provide the study for the case where $p \in \{1, 2\}$. $\alpha \in [0, \infty)$ is a hyperparameter that weights the relative contribution of the norm penalty term, Ω, to the standard training objective function J. Large values of α result in more

regularization. This approach of regularization is referred to as L_p norm which was widely used in machine learning with a variety of models. Although, L_p parameters norm regularization is not specific to neural networks.

When considering the L_p norm for neural networks regularization [244], one typically chooses a parameter norm penalty that penalizes only the weights vector $\boldsymbol{w}$ at each layer and leaves the biases unregularized [154]. It is desirable to use a different α per layer. However, as this can make it computationally expensive to compute each optimal value, one may consider using the same α across all the layers [154].

1.3.1.1 L_2 Parameters Norm

The L_2 norm penalty is commonly known as weight decay. This regularization approach drives the weights closer to the origin by adding a regularization term

$$\Omega(\boldsymbol{\theta}) = \frac{1}{2}\|\boldsymbol{w}\|_2^2 \,, \tag{1.33}$$

to the objective function. It is also known in other communities as ridge regression or Tikhonov regularization [429]. The impact of using the L_2 parameter norm as a regularization is to shrink the components of $\boldsymbol{w}$ where the entries that do not contribute to reduce the objective function are shrunk to have *nearly* zero magnitude, while the rest of the entries are slightly reduced. More technical details are provided in Appendix A.2.3.1.1.

1.3.1.2 L_1 Parameters Norm

As we mentioned in Sec.1.1.3, L_1 parameters norm minimization is an approximation to the L_0 parameters norm. Formally, it is defined as

$$\Omega(\theta) = \|\boldsymbol{w}\|_1 = \sum_i |w_i| \,. \tag{1.34}$$

Compared to L_2 norm, L_1 norm promotes sparsity by providing a solution with subset of the entries set *exactly* to zero while the rest of the entries are shrunk toward zero. The sparsity aspect of L_1 regularization plays an important role in feature selection in machine learning [184, 428]. More technical details on L_1 parameters norms and a comparaison with L_2 parameters norm are provided in Appendix A.2.3.1.2.

1.3.2 Implicit Regularization

We discuss here some approaches that are used to reduce the generalization error of neural network without necessarily reducing their complexity. However, some of these methods may have an effect on the model complexity.

1.3.2.1 Early Stopping

Early stopping assumes that minimizing iteratively an objective function J is stopped before it reaches its minimum. If the model's capacity exceeds the optimal capacity, early stopping may prevent overfitting.

When training such models with over capacity, one often observes that the training error decreases steadily through the learning epochs, while the validation set error begins to raise again at some point (Fig.1.7). This means that one can obtain a model with a better validation set error and hopefully a better test set error by returning the model's parameters at the point with the lowest validation set error. In practice, this can be achieved by keeping a copy of the model's parameters at each time a better validation set error is found.

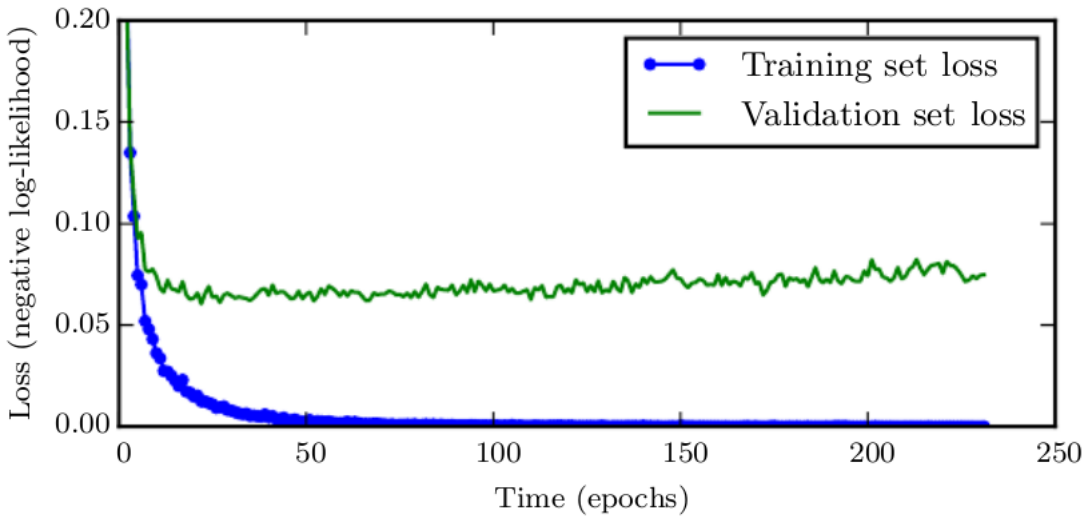


Fig. 1.7 Learning curves showing how the negative log-likelihood loss changes over epochs. In this example, a maxout network is trained over MNIST. One can observe how the validation set average loss begins to increase again, forming an asymmetric U-shape curve while the training objective loss keeps decreasing. (Credit: [154])

Early stopping is one of the most commonly form of regularization in training neural networks [154]. This is due to both its effectiveness and simplicity. It can be seen as an efficient hyperparameter selection algorithm that requires one run of the training process to find the best value when to stop training while most hyperparameters need multiple runs. Besides its training time reduction, it provides a regularization aspect without adding a penalty to the objective cost. However, early stopping requires evaluating the model over the validation set periodically which may slow down the training. Moreover, it requires storing a copy of the model's parameters.

We mentioned that early stopping is a form of regularization. In this section, we provide an intuition to this idea while formal demonstration is presented in Appendix A.2.3.2 following the work of [55, 397].

Consider a network with weights initialized from a distribution with zero mean. This means that initially most of the neurons provide a value close to zero for most the input vector. Thus, the network can be seen as a linear nilpotent operator with a very low complexity. During the training process, the weights are more likely to increase their magnitude which gradually increase the complexity of the network. Early stopping comes to prevent this increase of magnitude. Therefore, it serves as a method of a weight decay. Also, it prevents the network of becoming too complex [55, 397].

1.3.2.2 Data Augmentation

One way to promote generalization in machine learning is to train the model using a large number of training samples. However, in practice, the amount of training data is limited. A possible solution to deal with this issue is to create new data using the original one, mix them up and feed them to the model as a large training set [319, 21, 87, 394]. This approach assumes the generation of new training samples using some invariant parametric transformation function $g(\boldsymbol{x}; \boldsymbol{\theta})$ applied to the existing object $(\boldsymbol{x}, \boldsymbol{y})$. The invariance of this function is considered with respect to the sample target $\boldsymbol{y}$. This technique can be applied on a wide range of tasks. For instance in classification, when considering the input as an image, the label of the sample is invariant to a number of transformations applied on the input image such as translation, rotation, scaling, elastic deformation, contrast, etc.

Generating new training samples is a simple way to incorporate prior knowledge about the problem into the learning algorithm [319], which is the aim of most regularization approaches. For instance, when presenting a sample with different rotation angles and the same label, we are indicating to the learning algorithm that the label is invariant to rotation, thus, we introduce a domain knowledge. Generating new samples is also motivated by the bias-variance decomposition (Appendix A.2.1) where increasing the size of the training set will reduce the variance, but keep the bias the same. However, adding new samples does not reduce the complexity of the model.

Although, data augmentation might be helpful to improve generalization, it must be carried out with care. For example, in classification task, some transformations may change the class of the sample. For instance, in optical character recognition tasks, the model is required to recognize the difference between "n" and "u", "b" and "d". Therefore, 180° rotation and horizontal flip are not appropriate for all characters.

Neural networks are known to be sensitive to noise [424]. One way to improve their robustness is by training them with random noise applied to the input [390]. This can be seen as data augmentation. Adding noise to the input is a well known mechanism for some unsupervised learning methods [439, 440]. One can also carefully add noise to the hidden units [343] which can be seen as an augmentation of the data at multiple levels of abstraction.

Depending on the number of generated samples, data augmentation may increase the training time [244]. For models that need the whole training samples at once, augmenting the training data can raise the issue of the memory. However, when training neural networks, one needs only few samples at once. This allows to generate examples as much as one wants. Moreover, one can generate new samples on the fly without the need to store them.

1.3.2.3 Multi-task Learning

Multi-task learning (MTL) is a learning scenario where multiple learning tasks are solved at the same time, while exploiting commonalities and differences across tasks. This can result in improving the learning efficiency and prediction accuracy of the task-specific models, when compared to training models separately [70, 26, 426, 332].

[70] define MTL as an approach to inductive transfer[11] that improves generalization by using domain information contained in the training signals of related tasks as an inductive bias. This is achieved by learning tasks in parallel while using a shared representation. Therefore, the learned representation for a task can be helpful to learn other tasks. Using an MLT framework introduces a bias in the model selection in order to prefer hypotheses that explain more than one task. This shows to improve the generalization of the model and prevents its overfitting since it is required to solve many tasks at once which makes it less likely to overfit one of the tasks. Hence, MTL approaches are considered as a regularization. We mention that MTL framework does not only concern deep learning algorithms but a broad learning algorithms in machine learning [487]. Aside from designing MTL algorithms, there are many works that study the theoretical aspects of MTL and their generalization bounds [26, 42, 43, 41]. For instance, [26] showed that the generalization bounds can be improved through an MTL framework due to the parameters sharing which prevents overfitting. This holds when some assumptions about the statistical relationship between the different tasks are valid, meaning that there is something shared across some of the tasks.

In an MTL framework, a task can be any general learning task such as supervised task, unsupervised task, semi-supervised task, reinforcement learning task, or multi-view learning tasks. [487] provide a recent and detailed survey that contains different approaches of MTL algorithms.

In deep learning, MTL is a common approach and goes back to the 90′s [69]. It is generally applied by sharing the hidden layers between all the tasks, while keeping several task-specific output layers (Fig.1.8). This is known as *hard parameter sharing*. Another MTL scenario consists in considering each task has its own model. However, the different models are encouraged to have similar parameters by constraining the distance between the parameters to be small [115, 471]. This is referred to as *soft parameter sharing*. Nevertheless, some works escape such traditional MTL schemes. [273] improve upon hard parameter sharing by placing matrix priors on each level of dense layers, which allow to learn the relationship between tasks. [274] propose a bottom-up approach that starts with a thin network and dynamically widens it greedily during training using a criterion that promotes grouping of similar tasks. [298] propose *cross-stitch networks*. The process starts with two separate models just as in soft parameter sharing. Then, the authors use what is referred to as *cross-stitch* units to allow the model to determine in what way the task-specific networks leverage the knowledge of the other task by learning a linear combination of the output of the previous layers. Such units are placed after the pooling and dense layers. [367] propose *sluice networks*, a generalization of *cross-stitch networks*. Other innovations in MTL exist and depend on the task in hand or they are inspired from non-neural MTL frameworks [401, 182, 234, 470].

[11] *Inductive transfer* or *transfer learning* is a research problem in machine learning that focuses on storing knowledge gained while solving one problem and applying it to different but related problem.

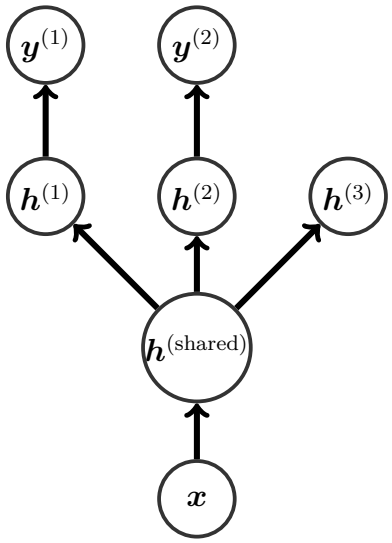


Fig. 1.8 Typical architecture for multi-task learning in neural networks where the tasks share a common input but involve different output targets. The low layers, $\boldsymbol{h}^{\text{shared}}$, can be shared among the different tasks while task-specific layers, $\boldsymbol{h}^{(1)}$ and $\boldsymbol{h}^{(2)}$, can be learned on top of the shared representation. The underlying assumption is that there exists a common pool of factors that explain the variations of $\boldsymbol{x}$, while each task is associated with a subset of such factors. In an unsupervised task, it is possible to pool top-level factors, $\boldsymbol{h}^{(3)}$, to be associated with none of the output task. Such factors may explain some of the variations of the input $\boldsymbol{x}$, but they are irrelevant to predict the targets $\boldsymbol{y}^{(1)}$ and $\boldsymbol{y}^{(2)}$. (Reference: [154])

MTL framework is an adequate choice where we are interested in obtaining predictions for multiple tasks at the same time. Such scenarios are common for instance in finance or economics forecasting, where we might want to predict the value of many possibly related variables, or in bioinformatics where we might want to predict symptoms for multiple diseases simultaneously. In the case of drug discovery, where tens or hundreds of active compounds may be predicted, MTL increases the prediction accuracy with the increase of the number of tasks [352]. Although MTL is constructed to improve the performance of many tasks at once, in some situations we only care about the performance of one task, that we call *main task*, whereas, the other tasks, named *auxiliary tasks*, are not important in the inference time. They are useful only during training in order to prevent the main task from overfitting. Using such auxiliary tasks as an MTL framework is a classical choice. In the case of using T auxiliary tasks, denoted $f_j(\cdot)$, along with a main task $f(\cdot)$, a standard optimization formula can be cast as follows. Let us consider $\boldsymbol{\theta}_{MTL} = \{\boldsymbol{\theta}_{sh}, \boldsymbol{\theta}, \boldsymbol{\theta}_j, \cdots, \boldsymbol{\theta}_T\}$ a set of parameters of the whole MTL framework, where $\boldsymbol{\theta}_{sh}$ is a shared set of parameters among all tasks, $\boldsymbol{\theta}$ is the set of parameters of the main task, and $\boldsymbol{\theta}_j$ is the set of parameters associated with the j^{th} auxiliary task. We note $\mathcal{C}(\cdot,\cdot)$ as the cost of the main task, while $\mathcal{C}_j(\cdot,\cdot)$ is the cost of the j^{th} auxiliary task. We consider that the same input $\boldsymbol{x}_i$ is fed to each task, while a different label $\boldsymbol{y}_i^j$ is associated to it depending on the task j. The training set

has N samples, with $\mathbb{D}_j = \{(\boldsymbol{x}_i, \boldsymbol{y}_i^j)\}_{i=1}^N$ a training set of the j^{th} auxiliary task, and $\mathbb{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^N$ is a training set for the main task. Each auxiliary task j is weighted using a coefficient λ_j. The optimization consists in solving

$$\arg\min_{\boldsymbol{\theta}_{MTL}} \sum_{i=1}^{N} \mathcal{C}(f(\boldsymbol{x}_i; \boldsymbol{\theta}_{sh}, \boldsymbol{\theta}), \boldsymbol{y}_i) + \sum_{j=1}^{T} \lambda_j \mathcal{C}_j(f_j(\boldsymbol{x}_i; \boldsymbol{\theta}_{sh}, \boldsymbol{\theta}_j), \boldsymbol{y}_i^j) \ . \tag{1.35}$$

Optimizing Eq.1.35 can be done easily in parallel using stochastic gradient descent. However, in practice, alternating between tasks seems to work better [91, 488].

Auxiliary tasks have been used in different setups in an MTL framework. For instance, [70] use tasks that predict different characteristics of the road in order to predict the steering direction in a self-driving car. [488] use head pose estimation and facial attributes inference as auxiliary tasks to predict facial landmarks. [138] use an adversarial loss in a domain adaptation[12] scenario in order to constrain the model to build internal representations that do not distinguish between domains. In some cases, one can encode some prior knowledge as an auxiliary task, mostly to learn better representations. This is known as *hints* and it is the old name of an MTL framework [410, 2, 3]. Reconstructing the input [36] or/and the output [35] data can be used as well as auxiliary tasks.

Although auxiliary tasks are helpful in an MTL framework, it is still unclear what tasks should be used in practice. Finding a good auxiliary task is often based on the assumption that the auxiliary task should be related to the main task somehow and that it should be helpful for the prediction of the main task. However, the relatedness of two tasks is still unclear. [70] define two tasks to be related if they use the same features to make a decision. [26] argue that related tasks share a common optimal hypothesis class, i.e., have the same inductive bias. [43] propose that two tasks are related if the data for both tasks can be generated from a fixed probability distribution using a set of transformations. [469] define that two tasks are related if their classification boundaries, i.e., parameters, are close. Despite the theoretical lack of our understanding to task relatedness, such concept is not binary but a spectrum. Allowing the models to learn what to share with each task might be a way to build better MTL frameworks and make better use of related or loosely related tasks [487].

1.3.2.4 Transfer Learning

Since transfer learning is not specific to deep learning methods, we decide to provide a general background on transfer learning as a learning paradigm in machine learning. Then, we detail its applications and how to achieve it in a deep learning based model.

Transfer Learning: Background

Transfer learning (TL) is the improvement of learning in a new task through the transfer of knowledge from a related task that has already been learned [327, 7, 329].

[12] *Domain adaptation* is a field associated with machine learning and transfer learning. It consists in a learning scenario where we aim at learning from a source data distribution a well performing model on a different, but related, target data distribution [62, 39, 94]. For instance, in spam filtering task, domain adaptation consists in adapting a model from one use (the source distribution) to a new user who receives significantly different email, i.e., the target distribution.

Common machine learning algorithms traditionally address isolated tasks such as classification, regression, and clustering, etc, under the assumption that training and test data are draw from the same feature space and the same distribution. When the distribution changes, most statistical models need to be rebuilt from scratch using newly collected data. In many real world applications, it is expensive or impossible to re-collect the needed data and rebuild the models. TL domain attempts to change this by developing methods to transfer knowledge learned in one or more *source tasks* and use it to improve learning a related *target task* (Fig.1.9). Techniques that enable knowledge transfer represent progress toward making machine learning as efficient as human learning. Extensive and detailed study of TL, its applications, and issues can be found in [329, 7, 327].

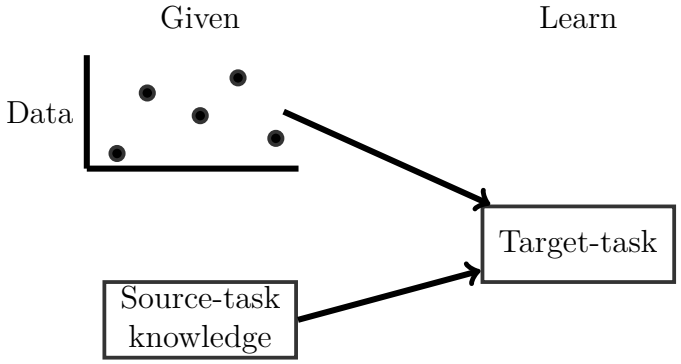


Fig. 1.9 Transfer learning is a machine learning algorithm with an additional source of information apart from the standard training data: knowledge extracted from one or more related tasks. (Reference: [327])

The study of TL can be motivated biologically by the fact that people can intelligently apply knowledge learned previously to solve new problems faster with better solutions [120, 465]. The fundamental motivation for TL in the field of machine learning is the need for lifelong machine learning methods that retain and reuse previously learned knowledge.

TL techniques have been applied in different learning algorithms including inductive learning and reinforcement learning [327, 7, 329]. In this section, we focus on its application to the former.

The goal of TL is to improve learning in the target task by leveraging knowledge from the source task. [327] refer to three common measures by which TL might improve learning, illustrated in Fig.1.10: • The initial performance achievable in the target task using only the transferred knowledge, before any further learning is done, compared to the initial performance of a model with a fresh start. • The amount of time it takes to fully learn the target task given the transferred knowledge compared to the amount of time to learn it from scratch. • The final performance level achievable in the target task compared to the final level without transfer.

It is important to distinguish the difference between TL and multi-task learning discussed in Sec.1.3.2.3, where several tasks are learned simultaneously (Fig.1.11). Multi-task learning is clearly closely related to TL, but it does not involve designated source and target tasks; instead the learning algorithm receives several tasks at once.

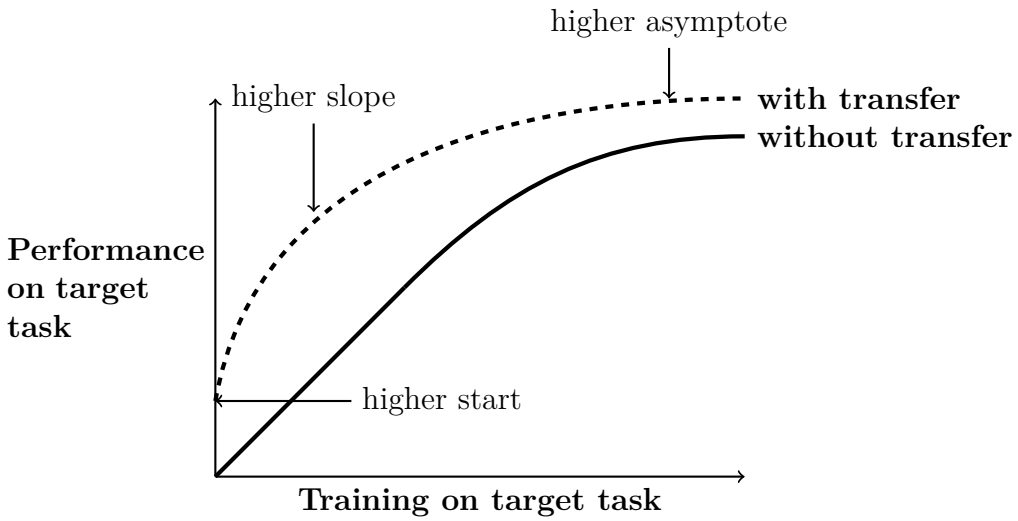


Fig. 1.10 Three ways in which transfer learning might improve learning the target task. (Reference: [327])

In contrast, in TL, the learning algorithm in the source task knows nothing about the target task. Moreover, TL aims at boosting the performance of the target domain by using the source domain data, i.e., its knowledge.

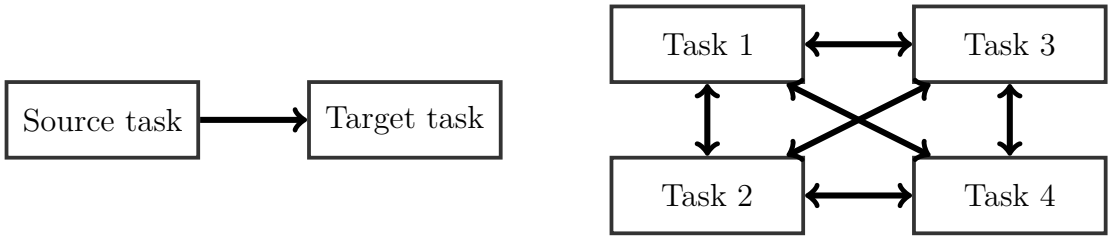


Fig. 1.11 *left*: in transfer learning, the information flows in one direction only, from the source task toward the target task. *right*: in multi-task learning, information can flow freely among all tasks. (Reference: [327])

In the case of inductive learning methods [300], the target-task inductive bias is chosen or adjusted based on the source-task knowledge [327]. It is usually concerned with improving the speed with which a model is learned, or with improving its generalization capability. The way this is done varies depending on which inductive is used to learn the source and target tasks. Some TL algorithms narrow down the hypothesis space, limiting the possible hypotheses, or remove search steps from consideration. Other methods broaden the space, allowing the search to discover more complex hypotheses, or add new search steps [26, 427, 292].

[7, 329] provide an extensive categorization of TL techniques. In the case of inductive learning, four possible ways of achieving a knowledge transfer: • Transferring knowledge of instances: where certain parts of the source data are reused together

with the labeled data in the target task [99, 224]. • Transferring knowledge of representations: in this case, the aim is to find a good representation in source and target tasks which is similar to finding common features in a multi-task learning scheme (Sec.1.3.2.3). Depending on the available labeled or unlabeled data in the source task, TL in this category can be done either in a supervised way by using an MTL framework to learn low-dimensional representations that are shared across the tasks [13, 14] or in unsupervised way to learn higher level representations [349, 448]. • Transferring knowledge of parameters: the underlying assumption in this approach is that individual models for related tasks should share some parameters such as in SVMs [123] , ensemble learning [139], or prior distributions of hyper-parameters such as in Bayesian frameworks [251, 57]. • Transferring relational knowledge: this approach deals with the data that is non-i.i.d. and can be represented by multiple relations, such as networked data and social network data. Its aim is to transfer the relationship among data from a source domain to a target domain. Statistical relational learning techniques are required to deal with such context [291, 293].

An important issue in TL is to recognize its limitations. Different works address theoretical aspects of TL such as its generalization bounds [278, 40, 56, 39, 142]. Although the generalization bounds are slightly different, under some conditions, the bound consists generally in two terms: the first is the error bound of the model on the source domain, the second is the bound on the distance between the source and the target domains, i.e., the distance between their marginal probability distributions which explains the relatedness of both domains. The aspect of relatedness of domains is an open issue in TL. When transferring knowledge between unrelated domains, *negative transfer* may happen: a case where the performance of the model in the target domain decrease when applying TL compared to its performance without TL. Often, when applying TL from a domain to another, it is necessary to *map* the characteristics/feature/representation of one task to another. In much cases, this is achieved by hand. Other methods have been propose to perform automatic *mapping* between domains. We intentionally skip the discussion of such important aspects in this thesis in order to stay focused on its main subject. [327, 7, 329] provide further details on theses subjects.

In the following, we present some aspects of using TL in deep learning methods.

Transfer Learning in Deep Learning

TL is a learning paradigm that can be applied to different machine learning models including deep learning. Most success in deep learning methods in academic research or industry has been driven by the use of large models that have be trained using a huge amount of supervised data [154]. In real life applications, nor the large amount of labeled data, nor the computation power required for training are usually available to conduct such large scale learning. We often encounter a situation where only few training samples are available in a particular target domain of interest. TL seems to be an alternative way to classical supervised learning in order to obtain better performance on the target domain by leveraging knowledge extracted from a source domain with abundant training samples.

We present in this section two main approaches to apply TL in deep learning, both of them are based on representation learning paradigm: either through using pre-trained models, or through learning domain-invariant representations.

- The application of convolutional neural networks (CNNs) has seen a large success, mostly in computer vision tasks [154]. In such models, low convolutional layers in the network tend to capture low level image features such as edges, while higher convolutional layers tend to capture more complex and task dependent details such as body parts, faces, and more compositional patterns [473, 480]. To perform TL in a target domain in computer vision tasks using CNNs, it is common [31, 357, 225] to use off-the-shelf pre-trained CNNs on ImageNet [104]. In practice, this is achieved by reusing the pre-trained convolutional layers, preferably low layers [473], and adding on top of it new layers specific to the target task. Then, the new network, as a whole or partially, is trained over the target data. A practical issue in such approach is that, often, the CNN model is over-parametrized for the target task and it may cause a slow in running speed mainly due to unnecessary computations. A practical solution is to drop useless filters through a pruning process [264, 306, 333] while tolerating slight reduction of the performance. While such reuse of pre-trained low layers of CNNs has seen large success in computer vision, pre-trained models have limited use in natural language processing (NLP) [351, 295, 235]. Low layers of an NLP model tend to learn task specific aspects such as syntax, which can not be helpful to perform cross domain adaptation [228]. What is needed in NLP is features that capture more fine-grained rules which are located at the top layers [154, 228]. While object recognition may be a prototype task that is shared among most computer vision tasks, language modeling [228] may be the closest analogy in NLP, where in order to predict the next word, a model needs: to possess knowledge of how a language is structured, understand what words likely are related to and likely to follow each other, and to model long-term dependencies. Such aspects may be shared among different tasks in NLP.

A model trained on ImageNet seems to capture details about the way animals and objects are structured and composed which is generally relevant when dealing with images. As such, the classification task on ImageNet seems to be a good proxy for general computer vision problems, as the same knowledge that is required to excel in it is also relevant for many other computer vision tasks. A similar assumption is used to motivate the use of generative model, that is, when training generative models, it is assumed that the ability to generate realistic images, for instance, requires an understanding of the underlying structure of images. Such knowledge about the structure can be used in different task where the structure is relevant. Such assumption relies itself on the premise that all images lie on a low-dimensional manifold, i.e., that there is some underlying structure to images that can be extracted by a model. [347] indicate that such a structure might indeed exist, and demonstrate it by generating realistic images describing transitions points in an image.

Unsupervised layer-wise pre-training technique [154, 50, 198] is another practical example of TL in deep learning. In such learning approach, the source task consists in learning in an unsupervised and incremental way hidden representations that disentangle the variation factors of the input data throughout reconstruction of the raw data.

- The second method of using TL in deep learning consists in learning domain-invariant representations. Creating representations that do not change based on the domain is very interesting since it should capture the variations of the data independently of the domain. This is less expensive and more feasible for non-vision tasks than generating representations that are useful for all tasks. In such scenario, only unlabeled data of each domain are needed in order to create domain-invariant

representations. Such representations are generally learned using stacked denoising auto-encoders and have seen success in NLP [150, 76] as well as in computer vision [489]. Other approaches encourage the data representation in different domains to be more similar and avoid domain-specific representations [214, 58, 138, 432].

TL is perhaps a potential learning paradigm that may allow breakthrough of deep learning techniques in a large number of small-data settings, which is the case in most real life applications. We hope that it will get more attention in deep learning research community in order to democratize the use of such powerful models.

1.3.2.5 Parameter Sharing: Particular Case of Convolutional Networks and Auto-encoders

As we mentioned earlier in Sec.1.3.1, regularization can be obtained by introducing prior knowledge of the domain or of the model's architecture into the way of estimating the parameters values. For instance, one may introduce some prior knowledge about the dependency between the values of different parameters. In pattern recognition applied to images, one can assume that an elementary pattern such as small corners may appear multiple times in an image. Therefore, it make sense to use the same sensor, i.e., parameter, all over the image in order to detect the pattern, instead of using different sensors at different locations. This is referred to as parameter sharing which can be seen as a regularization [154]. The first obvious advantage of parameter sharing over regularizing the parameters to be close to some predefined ideal parameters is the significant reduction of the model size in terms of memory use. Moreover, some models have more natural way, in certain applications, to use shared parameters such as the case of convolutional networks [131, 254].

Convolutional networks were motivated by the neurophysiological insights from [460, 213] that showed that simple and complex cells, which are found in the cat's visual cortex, fire in response to certain properties of visual sensory of inputs such as the orientation of edges. This led to convolutional networks as we know them today where the receptive field of a convolutional unit with given weight, which is typically a square filter, is shifted step by step across a 2 dimensional array of input values such as an image (Fig.1.12). The resulting 2D array of a subsequent activation events of this unit can be provided as inputs to higher level units, and so on. Usually, there are many filters at one representation level where each one learns to respond to specific properties. In the other hand, sharing filters is also motivated by some properties of the input signal. For instance, natural images have many statistical properties that are invariant to translation: a picture of a car remains a picture of car even if it is translated one pixel to any direction. Convolutional networks can take this property into account by sharing parameters across different locations of the image where the same feature is computed over different positions in the input. This means we can find a car with the same car detector even if we moved the car slightly. Another example that concerns low level features such as edges. One filter can learn to detect a specific type of edges with different angle. It is intuitive to apply the same filter across the whole input in order to find similar edges. Parameter sharing has enabled convolutional networks to dramatically lower the number of parameters (Fig.1.12) and to significantly increase network size without requiring a corresponding increase of the number of training samples.

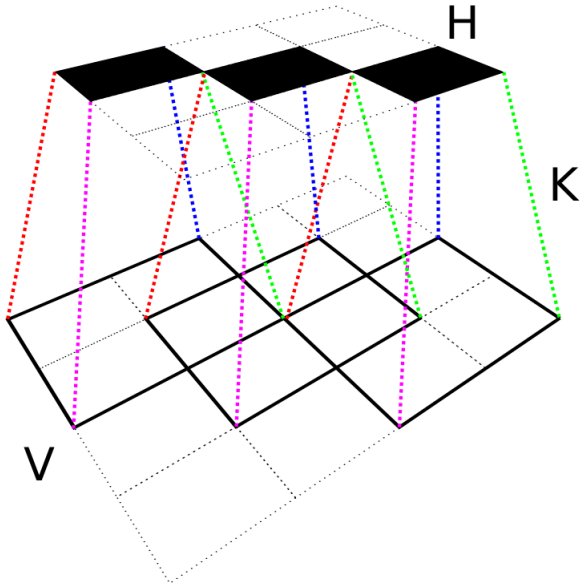


Fig. 1.12 A 2-dimensional convolution layer. The 2×2 filter $\boldsymbol{K}$ is applied to the 4×4 input $\boldsymbol{V}$ in order to get a 3×3 output $\boldsymbol{H}$. The number of weights is reduced from $4 \times 4 \times 3 \times 3 = 144$ to 4. (Credit: [103])

Auto-encoders [49, 353] employ as well parameter sharing between the encoder and the decoder layer. However, instead of using the same weight of the encoder, the decoder uses its transpose. This has the advantage to reduce the number of parameters of the auto-encoder. Depending on using a non-linearity in both layers, this type of parameter sharing may prevent the auto-encoder from learning linear transformation similarly to principal component analysis (PCA) method [48]. Furthermore, sharing the parameters between the encoder and the decoder may have come originally from Restricted Boltzmann Machines (RBMs) [399] where the same weight and its transpose are used to infer the hidden and the visible states.

1.3.2.6 Dropout

Dropout [403, 404] provides a computationally inexpensive but powerful method for regularizing a broad family of models and prevent their overfitting. The main idea of dropout consists in randomly omitting a subset of neurons during training for each sample by setting the output of these neurons to zero. This can prevent the remaining neurons from co-adaptation in which a feature extractor is only helpful in the context of several other specific feature detectors. Instead, dropout pushes each neuron to learn to detect a feature that is generally helpful to produce the correct answer independently of the internal context, i.e., the presence or the absence of other features. During the test phase, a scaling of the neuron output is necessary.

In practice, dropout is performed as follows. Let consider a multilayer perceptron with the layers: $\boldsymbol{y}_0, \cdots, \boldsymbol{y}_M$. Then, the dropout on the layer $\boldsymbol{y}_i$ of the size N can be described as follows:

1. For each training case, generate a binary vector mask $\boldsymbol{\mu}$ of length N, where each element is sampled from the Bernoulli distribution with probability $0 < p < 1$.
2. On the forward pass, multiply the values of $\boldsymbol{y}_i$ by $\boldsymbol{\mu}$.
3. On the backward pass, multiply the gradients $\boldsymbol{dy}_i$ by $\boldsymbol{\mu}$.
4. During the test phase, multiply all values of $\boldsymbol{y}_i$ by p.

Fig.1.13 illustrates the application of dropout across all the layers of a feedforward network.

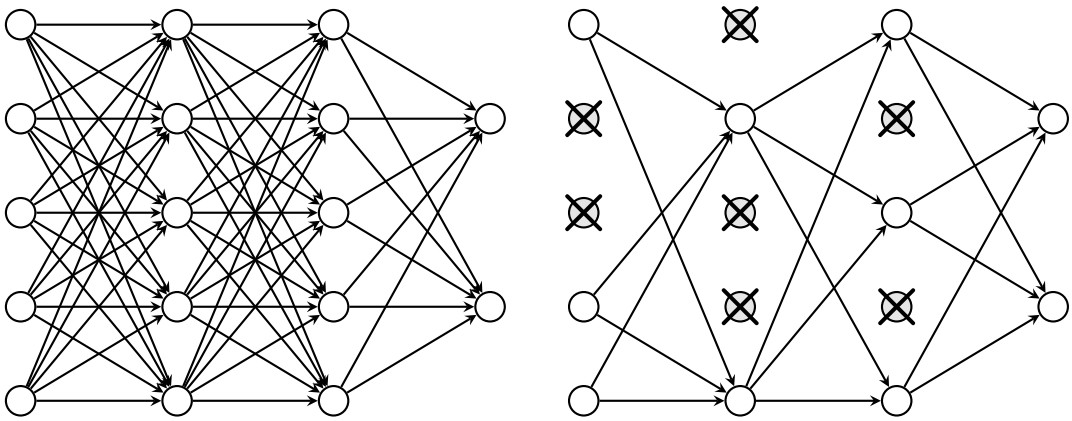


Fig. 1.13 *left*: a standard neural network with two hidden layers. *right*: An example of thinned network produced by applying dropout to the network on the left with $p = 0.5$ across all the layers. Crossed units have been dropped. (Reference: [404])

Dropout can be interpreted as a biological behavior. It pressures the hidden unit to be able to perform well regardless of which other hidden units are available. Hidden units must be ready to be swapped and interchanged between the subnetworks. Dropout [404] was inspired by an idea from biology: sexual reproduction, which involves swapping genes between two different organisms, creates evolutionary pressure for genes to become not just good but readily swapped between different organisms. Such genes and such features are robust to changes in their environment because they are not able to incorrectly adapt to unusual features of any one organism. Dropout thus regularizes each hidden unit to be merely a good feature but a feature that is good in many contexts. Other aspects of dropout with more details can be found in [154].

As a second interpretation, dropout can be seen as a method of making bagging [60] practical for ensembles of very many large neural networks [451, 154]. However, this seems impractical when each model is a large neural network, since training and evaluating such networks is costly in term of runtime and memory. Dropout provides a hack to perform bagging on an ensemble of exponentially many neural networks.

Applying dropout to a neural network during training consists into sampling a "thinned" network from it. The thinned network consists of all the units that survived dropout. A neural network with N units, can be seen as an ensemble of 2^N possible thinned network that share weights. For each representation of each training case, a

new thinned network is sampled and trained. Therefore, training a neural network with dropout can be seen as training a collection of 2^N subnetworks with extensive weight sharing, where each subnetwork gets trained rarely, if at all. The first hack in dropout is to use a binary mask as a way of approximating sampling a subnetwork from the total neural network. The mask of each unit is sampled independently from the others. The probability of sampling a mask with value 1 (causing a unit to be included in the subnetwork) is a hyperparameter p fixed before the training begins. Typical choice for input unit is 0.8 and for the hidden unit is 0.5 [154, 403, 404].

The second hack involved in dropout is at the inference time. Instead of averaging a bench of subnetworks, dropout infers the prediction by evaluating only one model: the model with all units, but with the weights going out of unit i multiplied by the probability of including unit i [403, 404, 154]. The motivation of this modification is to capture the right expected value of the output from that unit at the test phase. This is referred to as weight scaling inference rule. There is not yet any theoretical argument for the accuracy of this approximation inference rule in deep nonlinear networks, but, empirically, it performs very well [403, 404, 154]. Theoretical demonstration is provided only for $p = \frac{1}{2} = 0.5$ [403, 404, 154].

While dropout showed satisfying improvements on a large number of tasks, it also has its limitations. Due to randomness in the architecture, it increases the convergence time. Considering dropout as a regularization technique, it reduces the effective capacity of a model [154]. To offset this effect, one must increase the size of the model to compensate the missed units. However, this comes at the cost of a larger model and longer training time. Using larger model may allow the model to remember the dropout noise, which makes it worse. In this case, using other regularization techniques may be better. When extremely few labeled training examples are available, dropout is less effective [404]. On very large datasets, the obtained improvement is negligible, so for computational reasons it is recommended to not use it [154]. It was shown [443] that when applying dropout to linear regression, it is equivalent to L_2 weight decay, with a different weight decay coefficient for each input feature. The magnitude of each feature's weight decay coefficient is determined by its variance. Similar results hold for other linear models. For deep models, dropout is not equivalent to weight decay [154].

Dropout has motivated other stochastic approaches to training exponentially large ensembles of models that share weights such as DropConnect [447], and stochastic pooling [479]. So far, dropout remains the most widely used implicit ensemble method.

1.3.2.7 Batch Normalization

Batch normalization [216] is one of the most recent innovations in deep learning. Its primary purpose is to improve the optimization speed of deep neural networks. It is considered as a reparameterization of the model in a way that introduces both additive and multiplicative noise to the hidden units during training time. This noise can have a regularization effect and sometimes it makes dropout [403, 404] unnecessary.

Covariance shift [388] is a well known issue in machine learning. The problem raises when the input distribution of a model changes. For instance, when the test distribution is different than the train distribution, the model will perform poorly. For a long time, it has been known [258, 462] that the neural network training converges

faster if its input are whitened, i.e. linearly transformed to have zero mean and unit variances, and decorrelated.

Deep neural models consist in the composition of several layers. The computed gradient indicates how to update the parameters of each layer assuming that the other layers do not change. However, in practice, all the parameters are updated at once. When the updates are done, unexpected results can happen because many functions composed together have changed simultaneously. In other words, the stream of information in this hidden layer changes constantly. One possible way to deal with this is to consider higher level of interactions between layers, i.e., their parameters, such as second or higher order optimization. However, such optimization for deep models is impractical due to the computational cost. Batch normalization came up with a solution in order to stabilize the hidden distributions and thus prevent what is known as *internal covariance shift* issue. Its strategy consists in maintaining the distribution at each hidden unit fixed, i.e., normalized.

Given an input minibatch, let $\boldsymbol{H}$ be a matrix that contains the activations of the layer to be normalized, with the activations for each example appearing in a row of the matrix. To normalize $\boldsymbol{H}$, it is replaced with

$$\boldsymbol{H}' = \frac{\boldsymbol{H} - \boldsymbol{\mu}}{\boldsymbol{\sigma}} , \tag{1.36}$$

where $\boldsymbol{\mu}$ is a vector containing the mean of each unit and $\boldsymbol{\sigma}$ is a vector containing the standard deviation of each unit. The calculation here are based on broadcasting the vector $\boldsymbol{\mu}$ and the vector $\boldsymbol{\sigma}$ to be applied to every row of the matrix $\boldsymbol{H}$. Within each row, the arithmetic is element-wise. Therefore, $H_{i,j}$ is normalized by subtracting μ_j and dividing by σ_j. The rest of the network then operates on $\boldsymbol{H'}$ instead of $\boldsymbol{H}$. $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ are estimated over each minibatch during train time

$$\boldsymbol{\mu} = \frac{1}{m} \sum_i \boldsymbol{H}_{i,:} , \tag{1.37}$$

$$\boldsymbol{\sigma} = \sqrt{\delta + \frac{1}{m} \sum_i (\boldsymbol{H} - \boldsymbol{\mu})^2_{i,:}} , \tag{1.38}$$

where δ is a small positive value to avoid numerical instability. The most important thing is that the backpropagation is performed through $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ to compute the updates which means that the computed gradient will not change the units distributions which is a crucial innovation in this approach. Previous approaches [461, 348, 344, 107] had involved adding penalties to the cost function to encourage units to have normalized activation statistics or involved intervening to re-normalize unit statistics after each gradient descent step. The former approach usually resulted in imperfect normalization and the latter usually resulted in significant waste of time, as the learning algorithm repeatedly proposed changing the mean and variance, and the normalization step repeatedly undid this change. Batch normalization reparametrizes the model to make some units always standardized by definition.

At the test time, $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ may be replaced by running averages that were collected during train time. This allows the model to be evaluated on a single sample at a time without the need to re-estimating these statistics that depend on an entire minibatch.

Normalizing the mean and standard deviation of a unit can reduce the expressive power of the network that contains that unit. To maintain the expressive power of the network, it is common to replace the batch of hidden unit activation $\boldsymbol{H}$ with

$$\gamma \mathbf{H}' + \beta \ , \tag{1.39}$$

rather than simply the normalized $\boldsymbol{H}'$. γ and β are learned parameters that allow the new variable to have any possible mean and standard deviation depending on the optimization problem. Setting the mean as a learnable parameter makes it easier than estimate it stochastically based on a minbatch.

1.3.2.8 Unsupervised and Semi-supervised Learning

In machine learning, learning better representations usually leads to better generalization [51, 46]. In neural networks field, representations learning has been a hot topic for a long time and still is [154]. The main idea is to learn a model that provides adequate representations for the task in hand. In standard supervised learning setup, learning such representations requires large number of labeled data which may not be available. A possible solution to deal with this is to use unlabeled data either alone or combined with labeled data. Semi-supervised learning refers to a context where the learning is based on both: labeled and unlabeled data. Unsupervised learning refers to a learning context where only data without labels are used.

In semi-supervised learning, the main idea is to use unlabeled data to discover the underlying input domain distribution in an unsupervised way. In most cases, the supervised task partially shares its parameters with an unsupervised task. This may be seen as a regularization of the supervised task where the cost is constrained by the unsupervised cost [121, 50]. Most importantly, the unsupervised task introduces a generalization and prevents the supervised task from overfitting. This can be achieved through sharing general representations [154]. For instance, let us consider an unsupervised task that learns representation of trucks such as car, bus, motor-cycle. On the other hand, let us consider a supervised task that learns to recognize each of the previous trucks. One of the possible learned features in the unsupervised task is the concept of a "wheel" and more complicated feature "counting the number of wheels". Learning both these tasks can be helpful because they are important features in the supervised task that aims at distinguishing between the different trucks. This type of unsupervised learning can be very helpful in the case where only few labeled data is available with a large number of unlabeled data [154].

Since 2006, many works showed that unsupervised learning techniques can help to efficiently train deep feedforward networks [201, 199, 49] using multilayer perceptrons (MLP). Such methods are mostly based on layer-wise pre-training where layers are pre-trained sequentially using a reconstruction criterion. This allows to train Deep Belief Networks [201, 199] using RBMs and deep feedforward networks using different variants of auto-encoders [353, 354, 360]. Pre-training techniques can be seen as a regularization to the supervised task where it allows to learn intermediate feature functions that provide general features. It may also be seen as a better initialization of the weights by pushing them toward better regions in the parameters space. Thus, it avoids local minima. More insights on the unsupervised learning can be found in [121]. [261] provide a technical description of pre-training approach and auto-encoders.

Recently, there was an attempt to pre-train convolutional networks in an unsupervised way through reconstruction [287, 485, 110], however these methods are still facing open issues such as the deconvolution.

Pre-training technique has been abandoned in the last few years due to its greedy approach, the availability of large supervised data, and also to the improvements and accessibility to less greedy regularization techniques. One can also mention its lack of efficient stopping criterion to end the pre-training phase. Moreover, we did not see new advances in auto-encoders aspects in the last years. Today, learning better representations is achieved using a large number of supervised data and more complicated/deep models such as convolutional networks [244, 384, 126].

In this thesis, we focus more on regularizing neural networks through learning better representations using unsupervised learning, prior knowledge, or transfer learning.

1.3.3 Summary

Throughout this section, we have presented different variant of tools that are commonly used to prevent the overfitting of neural networks. We have divided such approaches into two categories: methods that reduce the model complexity, and methods that promote the model generalization without necessarily affecting its complexity.

1.4 Conclusion

We have presented in this first chapter of this thesis an introduction to machine learning with a focus on the generalization aspect (Sec.1.1), followed with an introduction to neural networks (Sec.1.2) and their overfitting issue. Finally, we closed this chapter by presenting some selected approaches used to improve the generalization of such models and alleviate their overfitting (Sec.1.3). Such methods may aim directly at reducing the network complexity or aim directly at improving the generalization performance of the model.

In this thesis, we tackled the overfitting issue of neural networks by focusing more on representation learning within the model, particularly when using few labeled training samples. We performed the regularization of learning representations in neural networks either through learning input/output representations using unsupervised approach [29, 33, 35], incorporating prior knowledge on learning internal representations [30], or using transfer learning [32]. All these contributions were made under the angle of tackling the overfitting issue of deep neural networks when dealing with small training sets which is the case in real life applications. We can divide our contributions into three different approaches:

- Approach based on unsupervised learning (chapter 2): This provides an access to large unsupervised samples and allows to learn the structure of the data without the need to large labeled data. This leads to improvements in the network performance [29, 33, 35].

- Approach based on prior knowledge about the task (chapter 3): The key idea is to exploit a prior belief about the distribution of the internal representation, in the case of a classification task, which is: samples within the same class should have the same internal representation. Incorporating this prior knowledge into

the network training allows improving its generalization while using small dataset for training [30].

- Approach based on transfer learning (chapter 4): This idea consists in training a deep model on a large labeled data over a specific task t_1. Then, one takes a subset of the learned parameters and use them, combined with new parameters, to learn a second task t_2 which has few training samples. Usually, only the parameters that learn low representations are used. Such parameters seem to be common among different tasks, particularly in computer vision. This allows to obtain models with large capacity partially trained and use them over small datasets with success [32].

The following chapters describe each of our contributions.

Chapter 2

Deep Neural Networks Regularization for Structured Output Prediction

2.1 Prologue

Article Details:

- **Deep Neural Networks Regularization for Structured Output Prediction**. Soufiane Belharbi, Romain Hérault[1], Clément Chatelain[1], and Sébastien Adam. Neurocomputing Journal, 281C:169-177, 2018.

Other Related Publications:

- **Learning Structured Output Dependencies Using Deep Neural Networks**. Soufiane Belharbi, Clément Chatelain, Romain Hérault, and Sébastien Adam. Deep Learning Workshop in the 32^{nd} International Conference on Machine Learning (ICML), 2015.
- **A Unified Neural Based Model For Structured Output Problems**. Soufiane Belharbi, Clément Chatelain, Romain Hérault, and Sébastien Adam. Conférence Francophone sur l'Apprentissage Automatique (CAP), 2015.
- **Deep Multi-Task Learning with Evolving Weights**. Soufiane Belharbi, Romain Hérault, Clément Chatelain, and Sébastien Adam. European Symposium on Artificial Neural Networks (ESANN), 2016.
- **Pondération Dynamique dans un Cadre Multi-Tâche pour Réseaux de Neurones Profonds**. Soufiane Belharbi, Romain Hérault, Clément Chatelain, and Sébastien Adam. Reconnaissance des Formes et l'Intelligence Artificielle (RFIA) (Session spéciale: Apprentissage et vision), 2016.

Context:
We provide in this chapter our first contribution which concerns regularization of

[1] Authors with equal contribution.

neural network in the context of structured output problems. Structured output problems is a set of problems where the output variable $\mathbf{y}$ is multi-dimensional and structural relations exist between its components. In this work, we aim at providing a regularization framework that makes use of unsupervised learning on both input $\mathcal{X}$ and more importantly on output $\mathcal{Y}$. Learning the input and output distribution allows to speed the training of the neural network and improves its generalization error. Moreover, this allows exploiting unlabeled inputs and/or label only outputs.

This work comes after a series of related works [29, 28, 33, 34]. Our framework is composed of three tasks: two unsupervised tasks and a main supervised task. Our first proposition [29, 28] of this framework had a sequential optimization scheme. Although it shows significant improvements, it still has issues with respect to the optimization schedule which is difficult to tune and can easily lead to overfitting for the unsupervised tasks. Our first attempt to fix this issue was through the works [33, 34] where we proposed a parallel optimization technique of pre-training [201, 199, 49, 353] instead of sequential one in hope to avoid overfitting and to reduce the number of hyperparameters one needs to setup. Through the work [33], we succeed to show that parallel optimization in this context of multi-tasking is better than sequential one. Later on, we extend this parallel optimization setup to our framework for solving structured output problems where the three tasks are optimized at once which led to this work [35]. Evaluated on a facial landmark detection problem, it allows to improve the generalization of the network and add more speed to their training. Furthermore, we show experimentally the possibility to use label-only data in an unsupervised way to improve more the generalization. We present this final work [35] as a chapter of this thesis under the form of one contribution. This chapter contains the original paper as it was accepted in Neurocomputing journal with slight adaptation of notation.

Contributions:
The contribution of this paper is to provide a parallel strategy to optimize the framework proposed initially in [29, 28] to solve structured output problems. The proposed strategy showed a significant improvement in the generalization of the network. Furthermore, we showed the possibility to use label-only data in an unsupervised fashion over the output which allowed to improve more the generalization.

2.2 Introduction

In the machine learning field, the main task usually consists in learning general regularities over the input space in order to provide a specific output. Most of machine learning applications aim at predicting a single value: a label for classification or a scalar value for regression. Many recent applications address challenging problems where the output lies in a multi-dimensional space describing discrete or continuous variables that are most of the time interdependent. A typical example is speech recognition, where the output label is a sequence of characters which are interdependent, following the statistics of the considered language. These dependencies generally constitute a regular structure such as a sequence, a string, a tree or a graph. As it provides constraints that may help the prediction, this structure should be either discovered if unknown, or integrated in the learning algorithm using prior assumptions. The range of applications that deal with structured output data is large. One can cite, among others, image labeling [126, 272, 320, 363, 486, 207, 265, 402], statistical natural language processing (NLP) [221, 323, 398, 376], bioinformatics [226, 417], speech processing [346, 481] and handwriting recognition [166, 409]. Another example which is considered in the evaluation of our proposal in this paper is the facial landmark detection problem. The task consists in predicting the coordinates of a set of keypoints given the face image as input (Fig.2.1). The set of points are interdependent throughout geometric relations induced by the face structure. Therefore, facial landmark detection can be considered as a structured output prediction task.

Fig. 2.1 Examples of facial landmarks from LFPW [37] training set.

One main difficulty in structured output prediction is the exponential number of possible configurations of the output space. From a statistical point of view, learning to predict accurately high dimensional vectors requires a large amount of data where in practice we usually have limited data. In this article we propose to consider structured output prediction as a representation learning problem, where the model must i) capture the discriminative relation between $\mathbf{x}$ (input) and $\mathbf{y}$ (output), and ii) capture the interdependencies laying between the variables of each space by efficiently modeling the input and output distributions. We address this modelization through a regularization scheme for training neural networks. Feedforward neural networks lack exploiting the structural information between the $\mathbf{y}$ components. Therefore, we incorporate in our framework an unsupervised task which aims at discovering this hidden structure. The advantage of doing so is there is no need to fix beforehand any prior structural information. The unsupervised task learns it on itself.

Our contributions is a multi-task framework dedicated to train feedforward neural networks models for structured output prediction. We propose to combine unsupervised tasks over the input and output data in parallel with the supervised task. This parallelism can be seen as a regularization of the supervised task which helps it to generalize better. Moreover, as a second contribution, we demonstrate experimentally the benefit of using the output labels $\mathbf{y}$ without their corresponding inputs $\mathbf{x}$. In this work, the multi task framework is instantiated using auto-encoders [440, 50] for both representations learning and exploiting unlabeled data (input) and label-only data (output). We demonstrate the efficiency of our proposal over a real-world facial landmark detection problem.

The rest of the paper is organized as follows. Related works about structured output prediction is proposed in section 2.3. Section 2.4 presents the proposed formulation and its optimization details. Section 2.5 describes the instantiation of the formulation using a deep neural network. Finally, section 2.6 details the conducted experiments including the datasets, the evaluation metrics and the general training setup. Two types of experiments are explored: with and without the use of unlabeled data. Results are presented and discussed for both cases.

2.3 Related work

We distinguish two main categories of methods for structured output prediction. For a long time, graphical models have showed a large success in different applications involving 1D and 2D signals. Recently, a new trend has emerged based on deep neural networks.

2.3.1 Graphical Models Approaches

Historically, graphical models are well known to be suitable for learning structures. One of their main strength is an easy integration of explicit structural constraints and prior knowledge directly into the model's structure. They have shown a large success in modeling structured data thanks to their capacity to capture dependencies among relevant random variables. For instance, Hidden Markov Models (HMM) framework has a large success in modeling sequence data. HMMs make an assumption that the output random variables are supposed to be independent which is not the case in many real-world applications where strong relations are present. Conditional Random Fields (CRF) have been proposed to overcome this issue, thanks to its capability to learn large dependencies of the observed output data. These two frameworks are widely used to model structured output data represented as a 1-D sequence [119, 346, 54, 248]. Many approaches have also been proposed to deal with 2-D structured output data as an extension of HMM and CRF. [316] propose a Markov Random Field (MRF) for document image segmentation. [423] provide an adaptation of CRF to 2-D signals with hand drawn diagrams interpretation. Another extension of CRF to 3-D signal is presented in [431] for 3-D medical image segmentation. Despite the large success of graphical models in many domains, they still encounter some difficulties. For instance, due to their inference computational cost, graphical models are limited to low dimensional structured output problems. Furthermore, HMM and CRF models

are generally used with discrete output data where few works address the regression problem [321, 129].

2.3.2 Deep Neural Networks Approaches

More recently, deep learning based approaches have been widely used to solve structured output prediction, especially proposed for image labeling problems. Deep learning domain provides many different architectures. Therefore, different solutions were proposed depending on the application in hand and what is expected as a result.

In image labeling task (also known as semantic segmentation), one needs models able to adapt to the large variations in the input image. Given their large success in image processing related tasks [244], convolutional neural networks is a natural choice. Therefore, they have been used as the core model in image labeling problems in order to learn the relevant features. They have been used either combined with simple post-processing in order to calibrate the output [89] or with more sophisticated models in structure modeling such as CRF [126] or energy based models [318]. Recently, a new trend has emerged, based on the application of convolution [272, 363] or deconvolutional [320] layers in the output of the network which goes by the name of fully convolutional networks and showed successful results in image labeling. Despite this success, these models does not take in consideration the output representation.

In many applications, it is not enough to provide the output prediction, but also its probability. In this case, Conditional Restricted Boltzmann Machines, a particular case of neural networks and probabilistic graphical models have been used with different training algorithms according to the size of the plausible output configurations [304]. Training and inferring using such models remains a difficult task. In this same direction, [27] tackle structured output problems as an energy minimization through two feed-forward networks. The first is used for feature extraction over the input. The second is used for estimating an energy by taking as input the extracted features and the current state of the output labels. This allows learning the interdependencies within the output labels. The prediction is performed using an iterative backpropagation-based method with respect to the labels through the second network which remains computationally expensive. Similarly, Recurrent Neural Networks (RNN) are a particular architecture of neural networks. They have shown a great success in modeling sequence data and outputing sequence probability for applications such as Natural Language Processing (NLP) tasks [271, 415, 18] and speech recognition [164]. It has also been used for image captioning [231]. However, RNN models doe not consider explicitly the output dependencies.

In [261], our team proposed the use of auto-encoders in order to learn the output distribution in a pre-training fashion with application to image labeling with promising success. The approach consists in two sequential steps. First, an input and output pre-training is performed in an unsupervised way using autoencoders. Then, a finetune is applied on the whole network using supervised data. While this approach allows incorporating prior knowledge about the output distribution, it has two main issues. First, the alteration of a network output layer is critical and must be performed carefully. Moreover, one needs to perform multiple trial-error loops in order to set the autoencoder's training hyper-parameters. The second issue is overfitting. When pre-training the output auto-encoder, there is actually no information that indicates if the pre-training is helping the supervised task, nor when to stop the pre-training.

The present work proposes a general and easy to use multi-task training framework for structured output prediction models. The input and the output unsupervised tasks are embedded into a regularization scheme and learned in parallel with the supervised task. The rationale behind is that the unsupervised tasks should provide a generalization aspect to the main supervised task and should limit overfitting. This parallel transfer learning which includes an output reconstruction task constitutes the main contribution of this work. In structured output context, the role of the output task is to learn the hidden structure within the original output data, in an unsupervised way. This can be very helpful in models that do not consider the relations between the components of the output representation such as feedforward neural networks. We also show that the proposed framework enables to use labels without input in an unsupervised fashion and its effect on the generalization of the model. This can be very useful in applications where the output data is abundant such as in a speech recognition task where the output is ascii text which can be easily gathered from Internet. In this article, we validate our proposal on a facial landmark prediction problem over two challenging public datasets (LFPW and HELEN). The performed experiments show an improvement of the generalization of deep neural networks and an acceleration of their training.

2.4 Multi-task Training Framework for Structured Output Prediction

Let us consider a training set $\mathbb{D}$ containing examples with both features and targets $(\boldsymbol{x}, \boldsymbol{y})$, features without target $(\boldsymbol{x}, \_)$, and targets without features $(\_, \boldsymbol{y})$. Let us consider a set $\mathbb{F}$ which is the subset of $\mathbb{D}$ containing examples with at least features $\boldsymbol{x}$, a set $\mathbb{L}$ which is the subset of $\mathbb{D}$ containing examples with at least targets $\boldsymbol{y}$, and a set $\mathbb{S}$ which is the subset of $\mathbb{D}$ containing examples with both features $\boldsymbol{x}$ and targets $\boldsymbol{y}$. One can note that all examples in $\mathbb{S}$ are also in $\mathbb{F}$ and in $\mathbb{L}$.

Input task The input task $\mathcal{R}_{in}$ is an unsupervised reconstruction task which aims at learning global and more robust input representation based on the original input data $\boldsymbol{x}$. This task projects the input data $\boldsymbol{x}$ into an intermediate representation space $\tilde{\boldsymbol{x}}$ through a coding function $\mathcal{P}_{in}$, known as encoder. Then, it attempts to recover the original input by reconstructing $\hat{\boldsymbol{x}}$ from $\tilde{\boldsymbol{x}}$ through a decoding function $\bar{\mathcal{P}}_{in}$, known as decoder

$$\hat{\boldsymbol{x}} = \mathcal{R}_{in}\left(\boldsymbol{x}; \boldsymbol{w}_{in}\right) = \bar{\mathcal{P}}_{in}\left(\tilde{\boldsymbol{x}} = \mathcal{P}_{in}\left(\boldsymbol{x}; \boldsymbol{w}_{cin}\right); \boldsymbol{w}_{din}\right) \ , \tag{2.1}$$

where $\boldsymbol{w}_{in} = \{\boldsymbol{w}_{cin}, \boldsymbol{w}_{din}\}$. The decoder parameters $\boldsymbol{w}_{din}$ are proper to this task however the encoder parameters $\boldsymbol{w}_{cin}$ are shared with the main task (see Fig.2.2). This multi-task aspect will attract, hopefully, the shared parameters in the parameters space toward regions that build more general and robust input representations and avoid getting stuck in local minima. Therefore, it promotes generalization. This can be useful to start the training process of the main task.

The training criterion for this task is given by

$$J_{in}(\mathbb{F}; \boldsymbol{w}_{in}) = \frac{1}{\operatorname{card} \mathbb{F}} \sum_{\boldsymbol{x} \in \mathbb{F}} \mathcal{C}_{in}(\mathcal{R}_{in}(\boldsymbol{x}; \boldsymbol{w}_{in}), \boldsymbol{x}) \ , \tag{2.2}$$

where $\mathcal{C}_{in}(\cdot, \cdot)$ is an unsupervised learning cost which can be computed on all the samples with features (i.e. on $\mathbb{F}$). Practically, it can be the mean squared error.

Output task The output task $\mathcal{R}_{out}$ is an unsupervised reconstruction task which has the same goal as the input task. Similarly, this task projects the output data $\boldsymbol{y}$ into an intermediate representation space $\tilde{\boldsymbol{y}}$ through a coding function $\mathcal{P}_{out}$, i.e. a coder. Then, it attempts to recover the original output data by reoncstructing $\hat{\boldsymbol{y}}$ based on $\tilde{\boldsymbol{y}}$ through a decoding function $\bar{\mathcal{P}}_{out}$, i.e. a decoder. In structured output data, $\tilde{\boldsymbol{y}}$ can be seen as a code that contains many aspect of the original output data $\boldsymbol{y}$, most importantly, its hidden structure that describes the global relation between the components of $\boldsymbol{y}$. This hidden structure is discovered in an unsupervised way without priors fixed beforehand which makes it simple to use. Moreover, it allows using labels only (without input $\boldsymbol{x}$) which can be helpful in tasks with abundant output data such as in speech recognition task (Sec.2.3)

$$\hat{\boldsymbol{y}} = \mathcal{R}_{out}\left(\boldsymbol{y}; \boldsymbol{w}_{out}\right) = \bar{\mathcal{P}}_{out}\left(\tilde{\boldsymbol{y}} = \mathcal{P}_{out}\left(\boldsymbol{y}; \boldsymbol{w}_{cout}\right); \boldsymbol{w}_{dout}\right) \ . \tag{2.3}$$

where $\boldsymbol{w}_{out} = \{\boldsymbol{w}_{cout}, \boldsymbol{w}_{dout}\}$. In the opposite of the input task, the encoder parameters $\boldsymbol{w}_{cout}$ are proper to this task while the decoder parameters $\boldsymbol{w}_{dout}$ are shared with the main task (see Fig.2.2).

The training criterion for this task is given by

$$J_{out}(\mathcal{L}; \boldsymbol{w}_{out}) = \frac{1}{\operatorname{card} \mathbb{L}} \sum_{\boldsymbol{y} \in \mathbb{L}} \mathcal{C}_{out}(\mathcal{R}_{out}(\boldsymbol{y}; \boldsymbol{w}_{out}), \boldsymbol{y}) \ , \tag{2.4}$$

where $\mathcal{C}_{out}(\cdot, \cdot)$ is an unsupervised learning cost which can be computed on all the samples with labels (i.e. on $\mathbb{L}$), typically, the mean squared error.

Main task The main task is a supervised task that attempts to learn the mapping function $\mathcal{M}$ between features $\boldsymbol{x}$ and labels $\boldsymbol{y}$. In order to do so, the first part of the mapping function is shared with the encoding part $\mathcal{P}_{in}$ of the input task and the last part is shared with the decoding part $\mathcal{P}_{out}$ of the output task. The middle part $\mathcal{L}$ of the mapping function $\mathcal{M}$ is specific to this task

$$\hat{\boldsymbol{y}} = \mathcal{M}\left(\boldsymbol{x}; \boldsymbol{w}_{sup}\right) = \bar{\mathcal{P}}_{out}\left(\mathcal{L}\left(\mathcal{P}_{in}\left(\boldsymbol{x}; \boldsymbol{w}_{cin}\right); \boldsymbol{w}_{s}\right); \boldsymbol{w}_{dout}\right) \ . \tag{2.5}$$

where $\boldsymbol{w}_{sup} = \{\boldsymbol{w}_{cin}, \boldsymbol{w}_{s}, \boldsymbol{w}_{dout}\}$. Accordingly, $\boldsymbol{w}_{cin}$ and $\boldsymbol{w}_{dout}$ parameters are respectively shared with the input and output tasks.

Learning this task consists in minimizing its learning criterion J_s,

$$J_{s}(\mathbb{S}; \boldsymbol{w}_{sup}) = \frac{1}{\operatorname{card} \mathbb{S}} \sum_{(\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{S}} \mathcal{C}_{s}(\mathcal{M}(\boldsymbol{x}; \boldsymbol{w}_{sup}), \boldsymbol{y}) \ , \tag{2.6}$$

where $\mathcal{C}_{s}(\cdot, \cdot)$ can be the mean squared error.

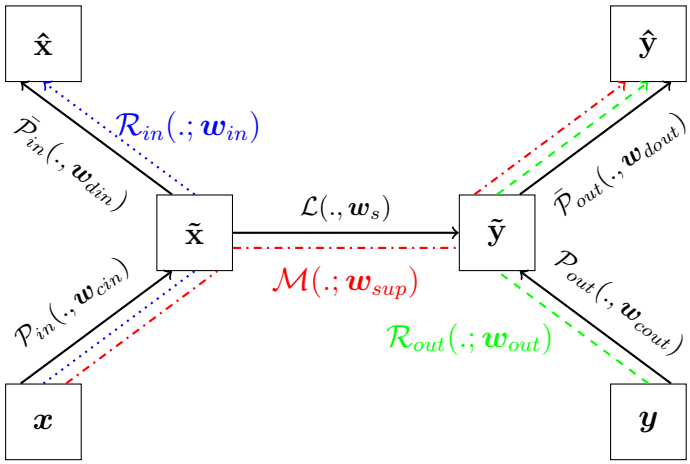


Fig. 2.2 Proposed MTL framework. Black plain arrows stand for intermediate functions, blue dotted arrow for input auxiliary task $\mathcal{R}_{in}$, green dashed arrow for output auxiliary task $\mathcal{R}_{out}$, and red dash-dotted arrow for the main supervised task $\mathcal{M}$.

As a synthesis, our proposal is formulated as a multi-task learning framework (MTL) [70], which gathers a main task and two secondary tasks. This framework is illustrated in Fig. 2.2.

Learning the three tasks is performed in parallel. This can be translated in terms of training cost as the sum of the corresponding costs. Given that the tasks have different importance, we weight each cost using a corresponding importance weight λ_{sup}, λ_{in} and λ_{out} respectively for the supervised, the input and output tasks. Therefore, the full objective of our framework can be written as

$$J(\mathbb{D};\boldsymbol{w}) = \lambda_{sup} \times J_s(\mathbb{S};\boldsymbol{w}_{sup}) + \lambda_{in} \times J_{in}(\mathbb{F};\boldsymbol{w}_{in}) + \lambda_{out} \times J_{out}(\mathbb{L};\boldsymbol{w}_{out}) \ , \tag{2.7}$$

where $\boldsymbol{w} = \{\boldsymbol{w}_{cin}, \boldsymbol{w}_{din}, \boldsymbol{w}_s, \boldsymbol{w}_{cout}, \boldsymbol{w}_{dout}\}$ is the complete set of parameters of the framework.

Instead of using fixed importance weights that can be difficult to optimally set, we adapt them through the learning epochs. In this context, Eq. 2.7 is modified as follows

$$\begin{aligned} J(\mathbb{D};\boldsymbol{w}) = \lambda_{sup}(t) \times J_s(\mathbb{S};\boldsymbol{w}_{sup}) \\ + \lambda_{in}(t) \times J_{in}(\mathbb{F};\boldsymbol{w}_{in}) + \lambda_{out}(t) \times J_{out}(\mathbb{L};\boldsymbol{w}_{out}) \ , \end{aligned} \tag{2.8}$$

where $t \geq 0$ indicates the learning epochs. Our motivation to adapt the importance weights is that we want to use the secondary tasks to start the training and avoid the main task to get stuck in local minima early in the beginning of the training by moving the parameters towards regions that generalize better. Then, toward the end of the training, we drop the secondary tasks by annealing their importance toward zero because they are no longer necessary for the main task. The early stopping of the secondary tasks is important in this context of mult-tasking as shown in [488] otherwise, they will overfit, therefore, they will harm the main task. The main advantage of Eq.2.8 is that it allows an interaction between the main supervised task and the secondary

tasks. Our hope is that this interaction will promote the generalization aspect of the main task and prevent it from overfitting.

2.5 Implementation

In this work, we implement our framework throughout a deep neural network. The main supervised task is performed using a deep neural network (DNN) with K layers. Secondary reconstruction tasks are carried out by auto-encoders (AE): the input task is achieved using an AE that has K_{in} layers in its encoding part, with an encoded representation of the same dimension as $\tilde{\boldsymbol{x}}$. Similarly, the output task is achieved using an AE that has K_{out} layers in its decoding part, with an encoded representation of the same dimension as $\tilde{\boldsymbol{y}}$. At least one layer must be dedicated in the DNN to link $\tilde{\boldsymbol{x}}$ and $\tilde{\boldsymbol{y}}$ in the intermediate spaces. Therefore, $K_{in} + K_{out} < K$.

Parameters $\boldsymbol{w}_{in}$ are the parameters of the whole input AE, $\boldsymbol{w}_{out}$ are the parameters of the whole output AE and $\boldsymbol{w}_{sup}$ are the parameters of the main neural network (NN). The encoding layers of the input AE are tied to the first layers of the main NN, and the decoding layers of the output AE are in turn tied to the last layers of the main NN. If $\boldsymbol{w}_i$ are the parameters of layer i of a neural network, then $\boldsymbol{w}_1$ to $\boldsymbol{w}_{K_{in}}$ parameters of the input AE are shared with $\boldsymbol{w}_1$ to $\boldsymbol{w}_{K_{in}}$ parameters of the main NN. Moreover, if $\boldsymbol{w}_{-i}$ are the parameters of last minus $i-1$ layer of a neural network, then parameters $\boldsymbol{w}_{-K_{out}}$ to $\boldsymbol{w}_{-1}$ of the output AE are shared with the parameters $\boldsymbol{w}_{-K_{out}}$ to $\boldsymbol{w}_{-1}$ of the main NN.

During training, the loss function of the input AE is used as J_{in}, the loss function of the output AE is used as J_{out}, and the loss function of the main NN is used as J_s.

Optimizing Eq.2.8 can be performed using Stochastic Gradient Descent. In the case of task combination, one way to perform the optimization is to alternate between the tasks when needed [91, 488]. In the case where the training set does not contain unlabeled data, the optimization of Eq.2.8 can be done in parallel over all the tasks. When using unlabeled data, the gradient for the whole cost can not be computed at once. Therefore, we need to split the gradient for each sub-cost according to the nature of the samples at each mini-batch. For the sake of clarity, we illustrate our optimization scheme in Algorithm 1 using on-line training (i.e. training one sample at a time). Mini-batch training can be performed in the same way.

2.6 Experiments

We evaluate our framework on a facial landmark detection problem which is typically a structured output problem since the facial landmarks are spatially inter-dependent. Facial landmarks are a set of key points on human face images as shown in Fig. 2.1. Each key point is defined by the coordinates (x, y) in the image ($(x, y) \in \mathbb{R}^2$). The number of landmarks is dataset or application dependent.

It must be emphasized here that the purpose of our experiments in this paper was not to outperform the state of the art in facial landmark detection but to show that learning the output dependencies helps improving the performance of DNN on that task. Thus, we will compare a model with/without input and output training. [483] use a cascade of neural networks. In their work, they provide the performance

Algorithm 1 Our training strategy for one epoch

1: $\mathbb{D}$ is the shuffled training set. B a sample.
2: **for** B in $\mathbb{D}$ **do**
3: **if** B contains $\boldsymbol{x}$ **then**
4: `Update` $\boldsymbol{w}_{in}$: Make a gradient step toward $\lambda_{in} \times J_{in}$ using B (Eq.2.2).
5: **end if**
6: **if** B contains $\boldsymbol{y}$ **then**
7: `Update` $\boldsymbol{w}_{out}$: Make a gradient step toward $\lambda_{out} \times J_{out}$ using B (Eq.2.4).
8: **end if**
9: `# parallel parameters update`
10: **if** B contains $\boldsymbol{x}$ and $\boldsymbol{y}$ **then**
11: `Update` $\boldsymbol{w}$: Make a gradient step toward J using B (Eq.2.8).
12: **end if**
13: `Update` λ_{sup}, λ_{in} `and` λ_{out}.
14: **end for**

of their first global network. Therefore, we will use it as a reference to compare our performance (both networks have the same number of layers) except they use larger training dataset.

We first describe the datasets followed by a description of the evaluation metrics used in facial landmark problems. Then, we present the general setup of our experiments followed by two types of experiments: without and with unlabeled data. An opensource implementation of our MTL deep instantiation is available online[2].

2.6.1 Datasets

We have carried out our evaluation over two challenging public datasets for facial landmark detection problem: LFPW [37] and HELEN [252].

LFPW dataset consists of 1132 training images and 300 test images taken under unconstrained conditions (in the wild) with large variations in the pose, expression, illumination and with partial occlusions (Fig.2.1). This makes the facial point detection a challenging task on this dataset. From the initial dataset described in LFPW [37], we use only the 811 training images and the 224 test images provided by the ibug website[3]. Ground truth annotations of 68 facial points are provided by [371]. We divide the available training samples into two sets: validation set (135 samples) and training set (676 samples).

HELEN dataset is similar to LFPW dataset, where the images have been taken under unconstrained conditions with high resolution and collected from Flikr using text queries. It contains 2000 images for training, and 330 images for test. Images and face bounding boxes are provided by the same site as for LFPW. The ground truth annotations are provided by [371]. Examples of dataset are shown in Fig.2.3.

All faces are cropped into the same size (50×50) and pixels are normalized in [0,1]. The facial landmarks are normalized into [-1,1].

[2] https://github.com/sbelharbi/structured-output-ae
[3] 300 faces in-the-wild challenge http://ibug.doc.ic.ac.uk/resources/300-W/

Fig. 2.3 Samples from HELEN [252] dataset.

2.6.2 Metrics

In order to evaluate the prediction of the model, we use the standard metrics used in facial landmark detection problems.

The Normalized Root Mean Squared Error (NRMSE) [95] (Eq.2.9) is the Euclidean distance between the predicted shape and the ground truth normalized by the product of the number of points in the shape and the inter-ocular distance D (distance between the eyes pupils of the ground truth),

$$NRMSE(s_p, s_g) = \frac{1}{N * D} \sum_{i=1}^{N} ||s_{pi} - s_{gi}||_2 \ , \tag{2.9}$$

where s_p and s_g are the predicted and the ground truth shapes, respectively. Both shapes have the same number of points N. D is the inter-ocular distance of the shape s_g.

Using the NMRSE, we can calculate the Cumulative Distribution Function for a specific NRMSE (CDF_{NRMSE}) value (Eq.2.10) overall the database,

$$CDF_x = \frac{card(NRMSE \leq x)}{n} \ , \tag{2.10}$$

where $card(.)$ is the cardinal of a set. n is the total number of images.

The CDF_{NRMSE} represents the percentage of images with error less or equal than the specified NRMSE value. For example a $CDF_{0.1} = 0.4$ over a test set means that 40% of the test set images have an error less or equal than 0.1. A CDF curve can be plotted according to these CDF_{NRMSE} values by varying the value of $NRMSE$.

These are the usual evaluation criteria used in facial landmark detection problem. To have more numerical precision in the comparison in our experiments, we calculate the Area Under the CDF Curve (AUC), using only the NRMSE range [0,0.5] with a step of 10^{-3}.

2.6.3 General training setup

To implement our framework, we use - a DNN with four layers $K = 4$ for the main task; - an input AE with one encoding layer $K_{in} = 1$ and one decoding layer; - an output AE with one encoding layer and one decoding layer $K_{out} = 1$. Referring to Fig.2.2, the size of the input representation $\boldsymbol{x}$ and estimation $\hat{\boldsymbol{x}}$ is $2500 = 50 \times 50$; the size of the output representation $\boldsymbol{y}$ and estimation $\hat{\boldsymbol{y}}$ is $136 = 68 \times 2$, given the 68

landmarks in a 2D plane; the dimension of intermediate spaces $\tilde{\boldsymbol{x}}$ and $\tilde{\boldsymbol{y}}$ have been set to 1025 and 64 respectively; finally, the hidden layer in the $\mathcal{L}$ link between $\tilde{\boldsymbol{x}}$ and $\tilde{\boldsymbol{y}}$ is composed of 512 units. The size of each layer has been set using a validation procedure on the LFPW validation set.

Sigmoid activation functions are used everywhere in the main NN and in the two AEs, except for the last layer of the main NN and the tied last layer of output AE which use a hyperbolic tangent activation function to suite the range $[-1,\ 1]$ for the output $y_i \in \boldsymbol{y}$.

We use the same architecture through all the experiments for the different training configurations. To distinguish between the multiple configurations we set the following notations:

1. **MLP**, a DNN for the main task with no concomitant training;
2. **MLP + in**, a DNN with input AE parallel training;
3. **MLP + out**, a DNN with output AE parallel training;
4. **MLP + in + out**, a DNN with both input and output reconstruction secondary tasks.

We recall that the auto-encoders are used only during the training phase. In the test phase, they are dropped. Therefore, the final test networks have the same architecture in all the different configurations.

Beside these configurations, we consider the mean shape (the average of the $\boldsymbol{y}$ in the training data) as a simple predictive model. For each test image, we predict the same estimated mean shape over the train set.

To clarify the benefit of our approach, all the configurations must start from the same initial weights to make sure that the obtained improvement is due to the training algorithm, not to the random initialization.

For the input reconstruction tasks, we use a denoising auto-encoder with a corruption level of 20% for the first hidden layer. For the output reconstruction task, we use a simple auto-encoder. To avoid overfitting, the auto-encoders are trained using L_2 regularization with a weight decay of 10^{-2}.

In all the configurations, the update of the parameters of each task (supervised and unsupervised) is performed using Stochastic Gradient Descent with momentum [414] with a constant momentum coefficient of 0.9. We use mini-batch size of 10. The training is performed for 1000 epochs with a learning rate of 10^{-3}.

In these experiments, we propose to use a simple linear adaptation scheme for the importance weights λ_{sup} (supervised task), λ_{in} (input task) and λ_{out} (output task). We retain the adaptation scheme proposed in [36], and presented in Fig.2.4.

The hyper-parameters (learning rate, batch size, momentum coefficient, weight decay, the importance weights) have been optimized on the LFPW validation set. We apply the same optimized hyper-parameters for HELEN dataset.

Using these configurations, we perform two types of experiments: with and without unlabeled data. We present in the next sections the obtained results.

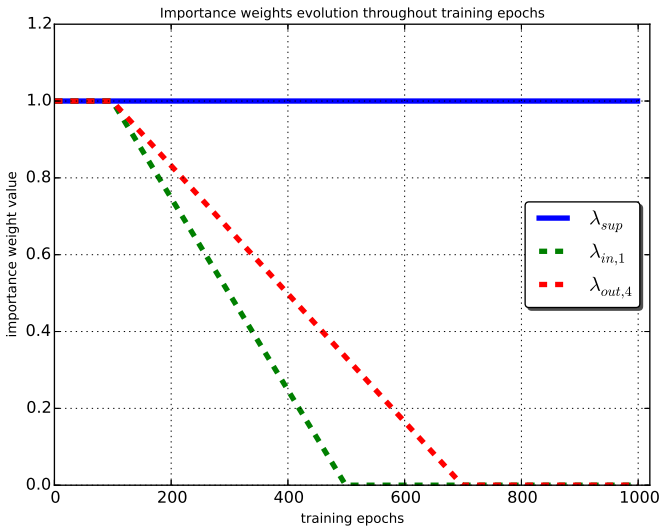


Fig. 2.4 Linear adaptation of the importance weights during training.

2.6.3.1 Experiments with fully labeled data

In this setup, we use the provided labeled data from each set in a classical way. For LFPW set, we use the 676 available samples for training and 135 samples for validation. For HELEN set, we use 1800 samples for training and 200 samples for validation.

In order to evaluate the different configurations, we first calculate the Mean Squared Error (MSE) of the best models found using the validation during the training. Column 1 (no unlabeled data) of Tab.2.1, 2.2 shows the MSE over the train and valid sets of LFPW and HELEN datasets, respectively. Compared to an MLP alone, adding the input training of the first hidden layer slightly reduces the train and validation error in both datasets. Training the output layer also reduces the train and validation error, with a more important factor. Combining the input train of the first hidden layer and output train of the last layer gives the best performance. We plot the tracked MSE over the train and valid sets of HELEN dataset in Fig.2.7a, 2.7b. One can see that the input training reduces slightly the validation MSE. The output training has a major impact over the training speed and the generalization of the model which suggests that output training is useful in the case of structured output problems. Combining the input and the output training improves even more the generalization. Similar behavior was found on LFPW dataset.

At a second time, we evaluate each configuration over the test set of each datasets using the $CDF_{0.1}$ metric. The results are depicted in Tab.2.3, 2.4 in the first column for LFPW and HELEN datasets, respectively. Similarly to the results previously found over the train and validation set, one can see that the joint training (supervised, input, output) outperforms all the other configurations in terms of $CDF_{0.1}$ and AUC. The CDF curves in Fig.2.8 also confirms this result. Compared to the global DNN in [483] over LFPW test set, our joint trained MLP performs better ([483]: $CDF_{0.1} = 65\%$, ours: $CDF_{0.1} = 69.64\%$), despite the fact that their model was trained using larger supervised dataset (combination of multiple supervised datasets beside LFPW).

An illustrative result of our method is presented in Fig.2.5, 2.6 for LFPW and HELEN using an MLP and MLP with input and output training.

Fig. 2.5 Examples of prediction on LFPW test set. For visualizing errors, red segments have been drawn between ground truth and predicted landmark. Top row: MLP. Bottom row: MLP+in+out. (no unlabeled data)

Fig. 2.6 Examples of prediction on HELEN test set. Top row: MLP. Bottom row: MLP+in+out. (no unlabeled data)

Table 2.1 MSE over LFPW: train and valid sets, at the end of training with and without unlabeled data.

	No unlabeled data		**With unlabeled data**	
	MSE train	MSE valid	MSE train	MSE valid
Mean shape	7.74×10^{-3}	8.07×10^{-3}	7.78×10^{-3}	8.14×10^{-3}
MLP	3.96×10^{-3}	4.28×10^{-3}	-	-
MLP + in	3.64×10^{-3}	3.80×10^{-3}	1.44×10^{-3}	2.62×10^{-3}
MLP + out	2.31×10^{-3}	2.99×10^{-3}	1.51×10^{-3}	2.79×10^{-3}
MLP + in + out	$\mathbf{2.12 \times 10^{-3}}$	$\mathbf{2.56 \times 10^{-3}}$	$\mathbf{1.10 \times 10^{-3}}$	$\mathbf{2.23 \times 10^{-3}}$

Table 2.2 MSE over HELEN: train and valid sets, at the end of training with and without data augmentation.

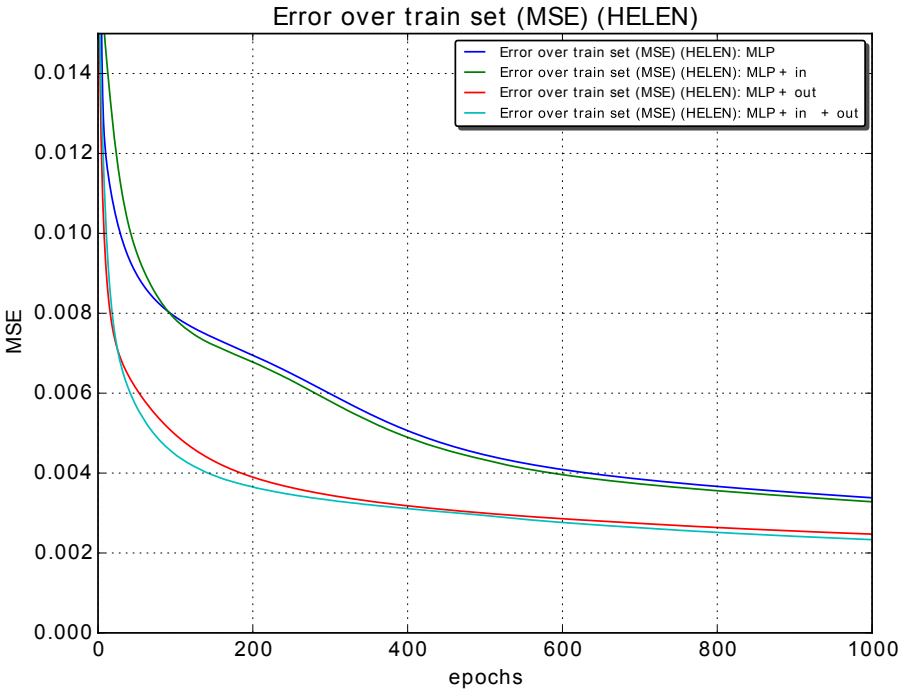
	Fully labeled data only		**Adding unlabeled or label-only data**	
	MSE train	MSE valid	MSE train	MSE valid
Mean shape	7.59×10^{-3}	6.95×10^{-3}	7.60×10^{-3}	0.95×10^{-3}
MLP	3.39×10^{-3}	3.67×10^{-3}	-	-
MLP + in	3.28×10^{-3}	3.42×10^{-3}	2.31×10^{-3}	2.81×10^{-3}
MLP + out	2.48×10^{-3}	2.90×10^{-3}	2.00×10^{-3}	2.74×10^{-3}
MLP + in + out	$\mathbf{2.34 \times 10^{-3}}$	$\mathbf{2.53 \times 10^{-3}}$	$\mathbf{1.92 \times 10^{-3}}$	$\mathbf{2.40 \times 10^{-3}}$

Table 2.3 **AUC** and $\mathbf{CDF_{0.1}}$ performance over LFPW test dataset with and without unlabeled data.

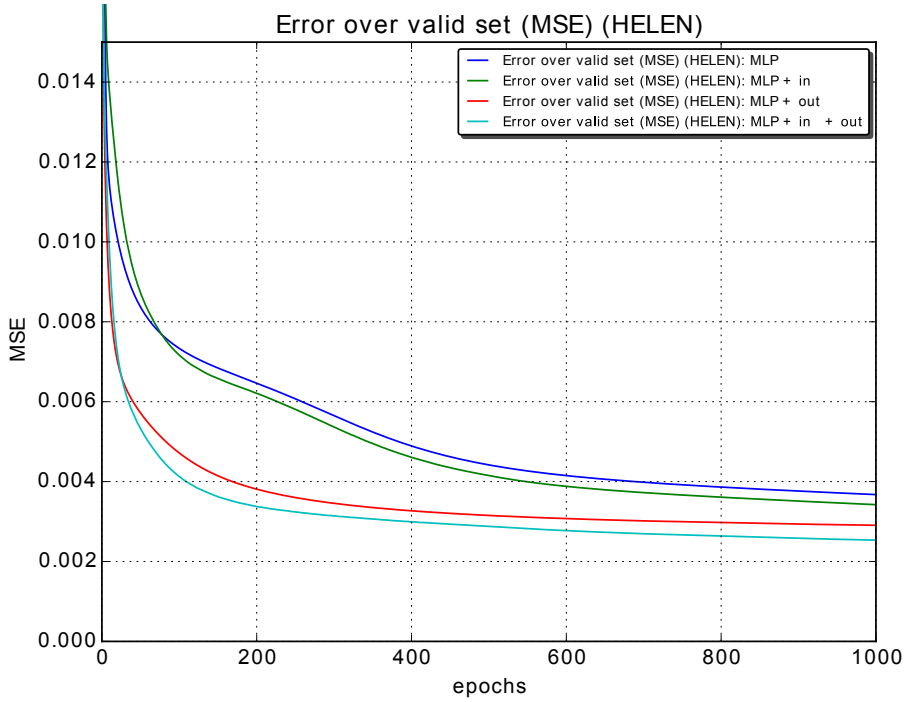
	Fully labeled data only		**Adding unlabeled or label-only data**	
	AUC	$\mathbf{CDF_{0.1}}$	**AUC**	$\mathbf{CDF_{0.1}}$
Mean shape	68.78%	30.80%	77.81%	22.33%
MLP	76.34%	46.87%	-	-
MLP + in	77.13%	54.46%	80.78%	67.85%
MLP + out	80.93%	66.51%	81.77%	67.85%
MLP + in + out	**81.51%**	**69.64%**	**82.48%**	**71.87%**

Table 2.4 **AUC** and $\mathbf{CDF_{0.1}}$ performance over HELEN test dataset with and without unlabeled data.

	Fully labeled data only		**Adding unlabeled or label-only data**	
	AUC	$\mathbf{CDF_{0.1}}$	**AUC**	$\mathbf{CDF_{0.1}}$
Mean shape	64.60%	23.63%	64.76%	23.23%
MLP	76.26%	52.72%	-	-
MLP + in	77.08%	54.84%	79.25%	63.33%
MLP + out	79.63%	66.60%	80.48%	65.15%
MLP + in + out	**80.40%**	**66.66%**	**81.27%**	**71.51%**

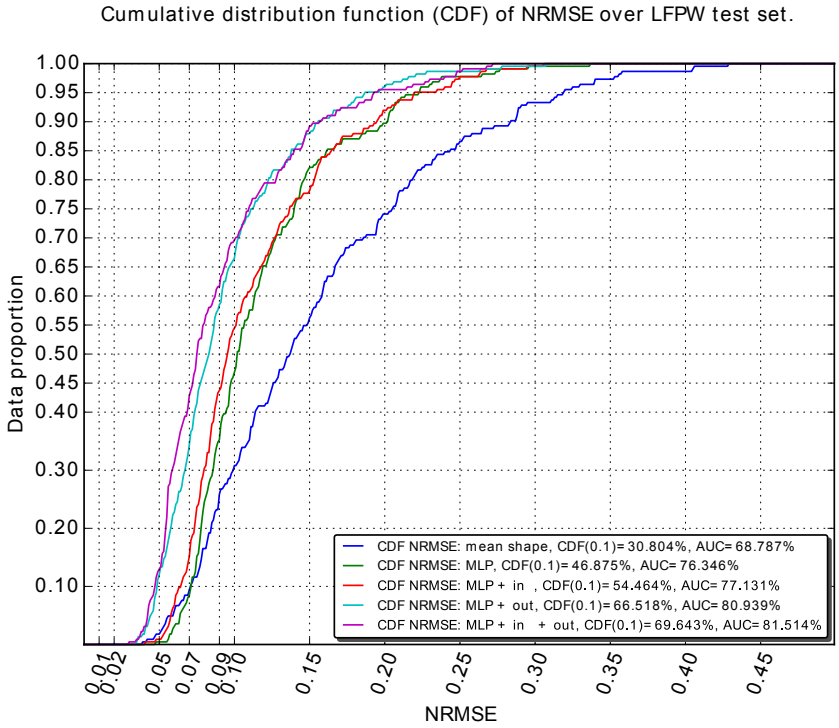


(a)

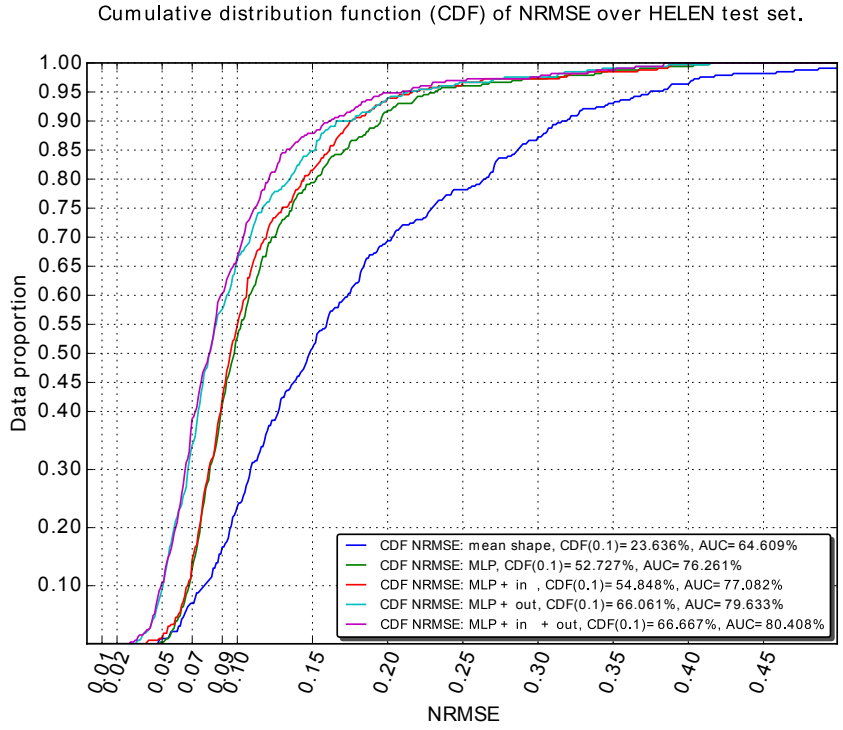


(b)

Fig. 2.7 MSE during training epochs over HELEN train (a) and valid (b) sets using different training setups for the MLP.



(a)



(b)

Fig. 2.8 CDF curves of different configurations on: (a) LFPW, (b) HELEN.

2.6.3.2 Data augmentation using unlabeled data or label-only data

In this section, we experiment our approach when adding unlabeled data (input and output). Unlabeled data (i.e. image faces without the landmarks annotation) are abundant and can be found easily for example from other datasets or from the Internet which makes it practical and realistic. In our case, we use image faces from another dataset.

In the other hand, label-only data (i.e. the landmarks annotation without image faces) are more difficult to obtain because we usually have the annotation based on the image faces. One way to obtain accurate and realistic facial landmarks without image faces is to use a 3D face model as a generator. We use an easier way to obtain facial landmarks annotation by taking them from another dataset.

In this experiment, in order to add unlabeled data for LFPW dataset, we take all the image faces of HELEN dataset (train, valid and test) and vice versa for HELEN dataset by taking all LFPW image faces as unlabeled data. The same experiment is performed for the label-only data using the facial landmarks annotation. We summarize the size of each train set in Tab.2.5..

Table 2.5 Size of augmented LFPW and HELEN train sets.

Train set / size of	Supervised data	Unsupervised input $\boldsymbol{x}$	Unsupervised output $\boldsymbol{y}$
LFPW	676	2330	2330
HELEN	1800	1035	1035

We use the same validation sets as in Sec.2.6.3.1 in order to have a fair comparison. The MSE are presented in the second column of Tab.2.1, 2.2 over LFPW and HELEN datasets. One can see that adding unlabeled data decreases the MSE over the train and validation sets. Similarly, we found that the input training along with the output training gives the best results. Identically, these results are translated in terms of $CDF_{0.1}$ and AUC over the test sets (Tab.2.3, 2.4). All these results suggest that adding unlabeled input and output data can improve the generalization of our framework and the training speed.

2.7 Conclusion

In this paper, we tackled structured output prediction problems as a representation learning problem. We have proposed a generic multi-task training framework as a regularization scheme for structured output prediction models. It has been instantiated through a deep neural network model which learns the input and output distributions using auto-encoders while learning the supervised task $\mathcal{X} \rightarrow \mathcal{Y}$. Moreover, we explored the possibility of using the output labels $\mathbf{y}$ without their corresponding input data $\mathbf{x}$ which showed more improvement in the generalization. Using a parallel scheme allows an interaction between the main supervised task and the unsupervised tasks which helped preventing the overfitting of the main task.

We evaluated our training method on a facial landmark detection task over two public datasets. The obtained results showed that our proposed regularization scheme improves the generalization of neural networks model and speeds up their training. We believe that our approach provides an alternative for training deep architectures for

structured output prediction where it allows the use of unlabeled input and label of the output data.

As a future work, we plan to adapt automatically the importance weights of the tasks. For that and in order to better guide their adaptation, we can consider the use of different indicators based on the training and the validation errors instead of the learning epochs only. Furthermore, one may consider other kind of models instead of simple auto-encoders in order to learn the output distribution. More specifically, generative models such as variational and adversarial auto-encoders [280] could be explored.

Acknowledgement

This work has been partly supported by the grant ANR-11-JS02-010 LeMon and the grant ANR-16-CE23-0006 “Deep in France”.

Chapter 3

Neural Networks Regularization Through Class-wise Invariant Representation Learning

3.1 Prologue

Article Details:

- **Neural Networks Regularization Through Class-wise Invariant Representation Learning**. Soufiane Belharbi, Clément Chatelain, Romain Hérault, and Sébastien Adam. **Under review**, *2017*.

Context:
Neural network models, particularly deep models, have seen a large success in different applications. However, training such models requires a large number of training data which are not available in many real world applications. Despite this issue, one would like to be able to use deep neural networks using only few samples which is the challenge that we tackle in this contribution.

Not far away from the time of writing this work, there was an urge toward learning unsupervised representations. Many of these works suggested using different approach of regularizations to learn better representations. This motivated us to explore a different and intuitive approach to regularize supervised learning of neural networks, especially in the case where only few training samples are available. In this work, we present a new approach to regularize neural networks when trained over few data for classification task. The key idea here is the use of a prior belief about the internal representation within a neural network. The idea simply states that samples within the same class should have the same representation. We formulate this idea as a cost function, under the form of a dissimilarity measure, which we integrate with the training cost to be minimized. We note that our regularization requires supervised samples.

Empirical results over different classification tasks showed improvements of the generalization error especially in the case where only few training samples are available. Moreover, we showed an intriguing behavior in learning intermediate representations within neural networks which is: the hidden layers do not tend, on its own, to learn

invariant features. This confirms that the propagated classification error does not necessarily train the hidden layers to learn meaningful and understandable features. This brings us one step further to make neural network more understandable and shed more light on the comprehension of the information stream within the layers to gain more control over it.

We present this work [30] as a chapter of this thesis under the form of one contribution. This chapter contains the original paper as it was submitted to Neural Networks journal with slight adaptation.

Contributions:
The contribution of this work is to provide a new regularization approach for training supervised neural networks for a classification task by guiding learning the internal representation of the network using a prior belief. This new approach showed to be more useful when only few training samples are available.

3.2 Introduction

For a long time, it has been understood in the field of deep learning that building a model by stacking multiple levels of non-linearity is an efficient way to achieve good performance on complicated artificial intelligence tasks such as vision [244, 396, 420, 187] or natural language processing [92, 457, 236, 162]. The rationale behind this statement is the hierarchical learned representations throughout the depth of the network which circumvent the need of extracting handcrafted features.

For many years, the non-convex optimization problem of learning a neural network has prevented going beyond one or two hidden layers. In the last decade, deep learning has seen a breakthrough with efficient training strategies of deeper architectures [199, 356, 49], and a race toward deeper models has began [244, 396, 420, 187]. This urge to deeper architectures was due to (i) large progress in optimization, (ii) the powerful computation resources brought by GPUs[1] and (iii) the availability of huge datasets such as ImageNet [104] for computer vision problems. However, in real applications, few training samples are usually available which makes the training of deep architectures difficult. Therefore, it becomes necessary to provide new learning schemes for deep networks to perform better using few training samples.

A common strategy to circumvent the lack of annotated data is to exploit extra informations related to the data, the model or the application domain, in order to guide the learning process. This is typically carried out through regularization which can rely for instance on data augmentation, L_2 regularization [430], dropout[404], unsupervised training [199, 356, 49, 361, 360, 36], shared parameters [255, 359, 135], etc.

Our research direction in this work is to provide a new regularization framework to guide the training process in a supervised classification context. The framework relies on the exploitation of prior knowledge which has already been used in the literature to train and improve models performance when few training samples are available [300, 381, 192, 319, 246, 475, 466, 476].

Indeed, prior knowledge can offer the advantage of more consistency, better generalization and fast convergence using less training data by guiding the learning process [300]. By using prior knowledge about the target function, the learner has a better chance to generalize from few data [300, 2–4]. For instance, in object localization such as part of the face, knowing that the eyes are located above the nose and the mouth can be helpful. One can exploit this prior structure about the data representation: to constrain the model architecture, to guide the learning process, or to post-process the model's decision.

In classification task, although it is difficult to define what makes a representation good, two properties are inherent to the task: Discrimination , i.e., representations must allow to separate samples of distinct classes. Invariance, i.e., representations must allow to obtain robust decision despite some variations of input samples. Formally, given two samples $\boldsymbol{x}^{(1)}$ and $\boldsymbol{x}^{(2)}$, a representation function $\Gamma(\cdot)$ and a decision function $\Psi(\cdot)$; when $\boldsymbol{x}^{(1)} \approx \boldsymbol{x}^{(2)}$, we seek invariant representations that provide $\Gamma(\boldsymbol{x}^{(1)}) \approx \Gamma(\boldsymbol{x}^{(2)})$, leading to smooth decision $\Psi(\Gamma(\boldsymbol{x}^{(1)})) \approx \Psi(\Gamma(\boldsymbol{x}^{(2)}))$. In this work, we are interested in the invariance aspect of the representations. This definition can be extended to more elaborated transformations such as rotation, scaling, translation, etc. However, in real life there are many other transformations which are difficult to formalize or

[1] Graphical Processing Units.

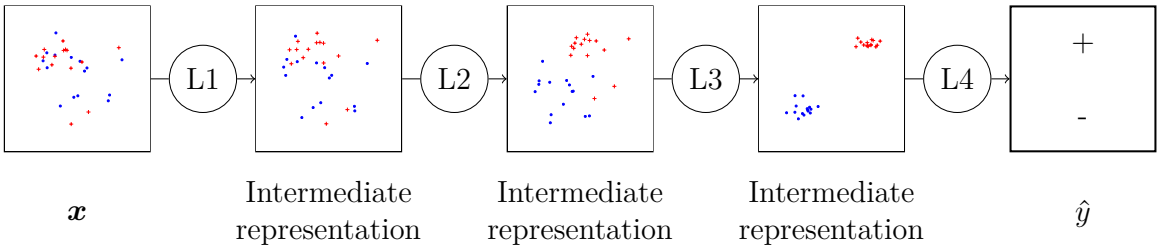


Fig. 3.1 Input/Hidden representations of samples from an artificial dataset along 4 layers of a MLP. Each representation is projected into a 2D space.

even enumerate. Therefore, we extend in this work the definition of the invariant representations to the class membership, where samples within the same class should have the same representation. At a representation level, this should generate homogeneous and tighter clusters per class.

In the training of neural networks, while the output layer is guided by the provided target, the hidden layers are left to the effect of the propagated error from the output layer without a specific target. Nevertheless, once the network is trained, examples may form (many) modes on hidden representations, i.e. outputs of hidden layers, conditionally to their classes. Most notably, on the penultimate representation before the decision stage, examples should agglomerate in distinct clusters according to their label as seen on Figure 3.1. From the aforementioned prior perspective about the hidden representations, we aim in this work to provide a learning scheme that promotes the hidden layers to build representations which are class-invariant and thus agglomerate in restricted number of modes. By doing so, we constrain the network to build invariant intermediate representations per class with respect to the variations in the input samples without explicitly specifying these variations nor the transformations that caused them.

We express this class-invariance prior as an explicit criterion combined with the classification training criterion. It is formulated as a dissimilarity between the representations of each pair of samples within the same class. The average dissimilarity over all the pairs of all the classes is considered to be minimized. To the best of our knowledge, none has used this class membership to build invariant representations. Our motivation in using this prior knowledge, as a form of regularization, is to be able to train deep neural networks and obtain better generalization error using less training data. We have conducted different experiments over MNIST benchmarck using two models (multilayer perceptrons and convolutional networks) for different classification tasks. We have obtained results that show important improvements of the model's generalization error particularly when trained with few samples.

The rest of the paper is organized as follows: Sec.3.3 presents related works for invariance learning in neural networks. We present our learning framework in Sec.3.4 followed by a discussion of the obtained results in Sec.3.5.

3.3 Related Work

Learning general invariance, particularly in deep architectures, is an attractive subject where different approaches have been proposed. The rational behind this framework is

to ensure the invariance of the learned model toward the variations of the input data. In this section, we describe three kinds of approaches of learning invariance within neural networks. Some of these methods were not necessarily designed to learn invariance however we present them from the invariance perspective. For this description, f is the target function to be learned.

***Invariance through data transformations*:**
It is well known that generalization performance can be improved by using larger quantity of training samples. Enlarging the number of samples can be achieved by generating new samples through the application of small random transformations such as rotation, scaling, random noise, etc [21, 87, 394] to the original examples. Incorporating such transformed data within the learning process has shown to be helpful in generalization [319]. [2] proposes the use of prior information about the behavior of f over perturbed examples using different transformations where f is constrained to be invariant over all the samples generated using these transformations. While data transformations successfully incorporate certain invariance into the learned model, they remain limited to some predefined and well known transformations. Indeed, there are many other transformations which are either unknown or difficult to formalize.

***Invariance through model architectures*:**
In some neural network models, the architecture implicitly builds a certain type of invariance. For instance, in convolutional networks [255, 359, 135], combining layers of feature extractors using weight sharing with local pooling of the feature maps introduces some degree of translation invariance [355, 259]. These models are currently state of the art strategies for achieving invariance in computer vision tasks. However, it is unclear how to explicitly incorporate in these models more complicated invariances such as large angle rotation and complex illumination. Moreover, convolutional and max-pooling techniques are somewhat specialized to visual and audio processing, while deep architectures are generally task independent.

***Invariance through analytical constraints*:**
Analytical invariance consists in adding an explicit penalty term to the training objective function in order to reduce the variations of f or its sub-parts when the input varies. This penalty is generally based on the derivatives of a criterion related to f with respect to the input. For instance, in unsupervised representation learning, [361] introduces a penalty for training auto-encoders which encourages the intermediate representation to be robust to small changes of the input around the training samples, referred to as contractive auto-encoders. This penalty is based on the Frobenius norm of the first order derivative of the hidden representation of the auto-encoder with respect to the input. Later, [360] extended the contractive auto-encoders by adding another penalty using the norm of an approximation of the second order derivative of the hidden representation with respect to the input. The added term penalizes curvatures and thus favors smooth manifolds. [176] exploit the idea that solving adversarial examples is equivalent to increase the attention of the network to small perturbation for each example. Therefore, they propose a layer-wise penalty which creates flat invariance regions around the input data using the contractive penalty proposed

in [361]. [393, 392] penalize the derivatives of f with respect to perturbed inputs using simple distortions in order to ensure local invariance to these transformations. Learning invariant representations through the penalization of the derivatives of the representation function $\Gamma(\cdot)$ is a strong mathematical tool. However, its main drawback is that the learned invariance is local and is generally robust toward small variations.

Learning invariance through explicit analytical constraints can also be found in metric learning. For instance, [82, 178] use a contrastive loss which constrains the projection in the output space as follows: input samples annotated as similar must have close (adjacent) projections and samples annotated as dissimilar must have far projections. In the same way, Siamese networks [63] proceed in learning similarity by projecting input points annotated as similar to be adjacent in the output space. This approach of analytical constraints is our main inspiration in this work, where we provide a penalty that constrains the representation function $\Gamma(\cdot)$ to build similar representation for samples from the same class, i.e., in a supervised way.

In the following section, we present our proposal with more details.

3.4 Proposed Method

In deep neural networks, higher layers tend to learn the most abstract features. We would like that samples of the same class have the same features. In order to do so, we add a penalty to the training criterion of the network to constrain the intermediate representations to be class-invariant. We first describe our regularization framework by providing basic definitions and our training criterion. Then, we discuss three measures of invariance studied in this work followed by the implementation of our framework.

3.4.1 Model Decomposition

Let us consider a parametric mapping function for classification: $\mathcal{M}(.;\boldsymbol{\theta}) : \mathcal{X} \rightarrow \mathcal{Y}$, represented here by a neural network model, where $\mathcal{X}$ is the input space and $\mathcal{Y}$ is the label space. This neural network is arbitrarily decomposed into two parametric sub-functions

1. $\Gamma(\cdot;\boldsymbol{\theta}_\Gamma) : \mathcal{X} \rightarrow \mathcal{Z}$, a representation function parameterized with the set $\boldsymbol{\theta}_\Gamma$. This sub-function projects an input sample $\boldsymbol{x}$ into a representation space $\mathcal{Z}$.
2. $\Psi(\cdot;\boldsymbol{\theta}_\Psi) : \mathcal{Z} \rightarrow \mathcal{Y}$, a decision function parameterized with the set $\boldsymbol{\theta}_\Psi$. It performs the classification decision over the representation space $\mathcal{Z}$.

The network decision function can be written as follows

$$\mathcal{M}(\boldsymbol{x}^{(i)};\boldsymbol{\theta}) = \Psi(\Gamma(\boldsymbol{x}^{(i)};\boldsymbol{\theta}_\Gamma);\boldsymbol{\theta}_\Psi) \ , \tag{3.1}$$

where $\boldsymbol{\theta} = \{\boldsymbol{\theta}_\Gamma, \boldsymbol{\theta}_\Psi\}$.

Such a possible decomposition of a neural network with $K = 4$ layers is presented in Fig.3.2. Here, the decision function $\Psi(\cdot)$ is composed of solely the output layer while the rest of the hidden layers form the representation function $\Gamma(\cdot)$.

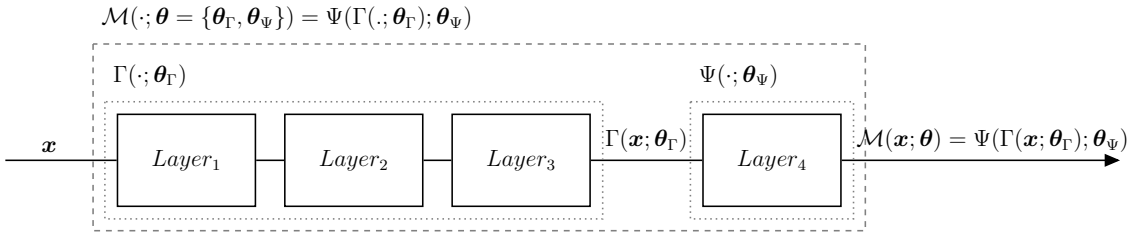


Fig. 3.2 Decomposition of the neural network $\mathcal{M}(\cdot)$ into a representation function $\Gamma(\cdot)$ and a decision function $\Psi(\cdot)$.

3.4.2 General Training Framework

In order to constrain the intermediate representations $\Gamma(\cdot)$ to form clusters over all the samples within the same class we modify the training loss by adding a regularization term. Thus, the training criterion J is composed of the sum of two terms. The first term J_{sup} is a standard supervised term which aims at reducing the classification error. The second and proposed regularization term J_H is a hint penalty that aims at constraining the intermediate representations of samples within the same class to be similar. By doing so, we constrain $\Gamma(\cdot)$ to lean invariant representations with respect to the class membership of the input sample.

Proposed Hint Penalty

Let $\mathbb{D} = \{(\boldsymbol{x}^{(i)}, y^{(i)})\}$ be a training set for classification task with S classes and N samples; $(\boldsymbol{x}^{(i)}, y^{(i)})$ denotes an input sample and its label. Let $\mathbb{D}_s$ be the sub-set of $\mathbb{D}$ that consists in all the examples of class s, i.e. $\mathbb{D}_s = \{(\boldsymbol{x}, y) \in \mathbb{D} \ s.t. \ y = s\}$. By definition, $\mathbb{D} = \bigcup_{s=1}^{S} \mathbb{D}_s$. For the sake of simplicity, even if $\mathbb{D}$ and $\mathbb{D}_s$ contains tuples of (feature,target), $\boldsymbol{x}$ represents only the feature part in the notation $\boldsymbol{x} \in \mathbb{D}$.

Let $\boldsymbol{x}^{(i)}$ be an input sample. We want to reduce the dissimilarity over the space $\mathcal{Z}$ between the projection of $\boldsymbol{x}^{(i)}$ and the projection of every sample $\boldsymbol{x}^{(i)} \in \mathbb{D}_s$ with $j \neq i$. For this sample $\boldsymbol{x}^{(i)}$, our hint penalty can be written as follows

$$J_h(\boldsymbol{x}^{(i)};\boldsymbol{\theta}_\Gamma) = \frac{1}{|\mathbb{D}_s| - 1} \sum_{\substack{\boldsymbol{x}^{(j)} \in \mathbb{D}_s \\ j \neq i}} \mathcal{C}_h(\Gamma(\boldsymbol{x}^{(i)};\boldsymbol{\theta}_\Gamma), \Gamma(\boldsymbol{x}^{(j)};\boldsymbol{\theta}_\Gamma)) \ , \tag{3.2}$$

where $\mathcal{C}_h(\cdot,\cdot)$ is a loss function that measures how much two projections in $\mathcal{Z}$ are dissimilar and $|\mathbb{D}_s|$ is the number of samples in $\mathbb{D}_s$.

Fig.3.3 illustrates the procedure to measure the dissimilarity in the intermediate representation space $\mathcal{Z}$ between two input samples $\boldsymbol{x}^{(i)}$ and $\boldsymbol{x}^{(j)}$ with the same label. Here, we constrained only one hidden layer to be invariant. Extending this procedure for multiple layers is straightforward. It can be done by applying a similar constraint over each concerned layer.

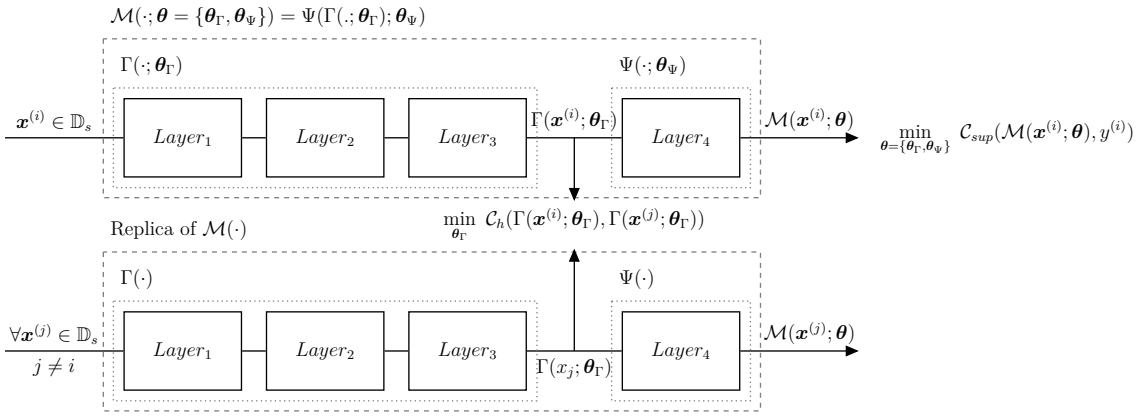


Fig. 3.3 Constraining the intermediate learned representations to be similar over a decomposed network $\mathcal{M}(\cdot)$ during the training phase.

Regularized Training Loss

The full training loss can be formulated as follows

$$J(\mathbb{D};\boldsymbol{\theta})=\underbrace{\frac{\gamma}{N}\sum_{(\boldsymbol{x}^{(i)},y^{(i)})\in\mathbb{D}}\mathcal{C}_{sup}(\Psi(\Gamma(\boldsymbol{x}^{(i)};\boldsymbol{\theta}_\Gamma);\boldsymbol{\theta}_\Psi),y^{(i)})}_{\text{Supervised loss } J_{sup}}+\underbrace{\frac{\lambda}{S}\sum_{s=1}^{S}\frac{1}{|\mathbb{D}_s|}\sum_{\boldsymbol{x}^{(i)}\in\mathbb{D}_s}J_h(\boldsymbol{x}^{(i)};\boldsymbol{\theta}_\Gamma)}_{\text{Hint penalty } J_H}\ , \tag{3.3}$$

where γ and λ are regularization weights, $\mathcal{C}_{sup}(\cdot,\cdot)$ the classification loss function. If one use a dissimilarity measure $\mathcal{C}_h(\cdot,\cdot)$ in J_h that is symmetrical such as typically a distance, summations in the term J_H could be rewritten to prevent the same sample couple to appear twice.

Eq.3.3 shares a similarity with the contrastive loss [82, 178, 63]. This last one is composed of two terms. One term constrains the learned model to project similar inputs to be closer in the output space. In Eq.3.3, this is represented by the hint term. In [82, 178, 63], to avoid collapsing all the inputs into one single output point, the contrastive loss uses a second term which projects dissimilar points far from each other by at least a minimal distance. In Eq.3.3, the supervised term prevents, implicitly, this collapsing by constraining the extracted representations to be discriminative with respect to each class in order to minimize the classification training error.

3.4.3 Implementation and Optimization Details

In the present work, we have chosen the cross-entropy as the classification loss $\mathcal{C}_{sup}(\cdot,\cdot)$.

In order to quantify how much two representation vectors in $\mathcal{Z}$, $\boldsymbol{a},\boldsymbol{b}\in\mathcal{Z}\subset\mathbb{R}^V$, are dissimilar we proceed using a distance based approach for $\mathcal{C}_h(\cdot,\cdot)$. We study three different measures: the squared Euclidean distance (SED),

$$\mathcal{C}_h(\boldsymbol{a},\boldsymbol{b})=\|\boldsymbol{a}-\boldsymbol{b}\|_2^2=\sum_{v=1}^{V}(a_v-b_v)^2\ , \tag{3.4}$$

the normalized Manhattan distance (NMD),

$$\mathcal{C}_h(\boldsymbol{a}, \boldsymbol{b}) = \frac{1}{V} \sum_{v=1}^{V} |a_v - b_v| \ , \tag{3.5}$$

and the angular similarity (AS),

$$\mathcal{C}_h(\boldsymbol{a}, \boldsymbol{b}) = \arccos\left(\frac{\langle \boldsymbol{a}, \boldsymbol{b} \rangle}{\|\boldsymbol{a}\|_2 \, \|\boldsymbol{b}\|_2}\right) \ . \tag{3.6}$$

Minimizing the loss function of Eq.3.3 is achieved using Stochastic Gradient Descent (SGD). Eq.3.3 can be seen as multi-tasking where two tasks represented by the supervised term and the hint term are in concurrence. One way to minimize Eq.3.3 is to perform a parallel optimization of both tasks by adding their gradient. Summing up the gradient of both tasks can lead to issues mainly because both tasks have different objectives that do not steer necessarily in the same direction. In order to avoid these issues, we propose to separate the gradients by alternating between the two terms at each mini-batch which showed to work well in practice [70, 458, 91, 36]. Moreover, we use two separate optimizers where each term has its own optimizer. By doing so, we make sure that both gradients are separated.

On a large dataset, computing all the dissimilarity measures in J_H in Eq.3.3 over the whole training dataset is computationally expensive due to the large number of pairs. Therefore, we propose to compute it only over the mini-batch presented to the network. Consequently, we need to shuffle the training set $\mathbb{D}$ periodically in order to ensure that the network has seen almost all the possible combinations of the pairs. We describe our implementation in Alg.2.

Algorithm 2 Our training strategy

$\mathbb{D}$ is the training set. B_s a mini-batch. B_r a mini-batch of all the possible pairs in B_s (Eq.3.3). OP_s an optimizer of the supervised term. OP_r an optimizer of the dissimilarity term. `max_epochs`: maximum epochs. γ, λ are regularization weights.
for i=1..`max_epoch` **do**
    Shuffle $\mathbb{D}$. Then, split it into mini-batches.
    for (B_s, B_r) in $\mathbb{D}$ **do**
        Make a gradient step toward J_{sup} using B_s and OP_s. (Eq.3.3)
        Make a gradient step toward J_H using B_h and OP_r. (Eq.3.3)
    end for
end for

3.5 Experiments

In this section, we evaluate our regularization framework for training deep networks on a classification task as described in Sec.3.4. In order to show the effect of using our regularization on the generalization performance, we will mainly compare the

generalization error of a network trained with and without our regularizer on different benchmarks of classification problems.

3.5.1 Classification Problems and Experimental Methodology

In our experiments, we consider three classification problems. We start by the standard MNIST digit dataset. Then, we complicate the classification task by adding different types of noise. We consider the three following problems:

- The standard MNIST digit classification problem with *50000*, *10000* and *10000* training, validation and test set. We refer to this benchmark as *mnist-std*. (Fig.3.4, top row).
- MNIST digit classification problem where we use a background mask composed of a random noise followed by a uniform filter. The dataset is composed of *100000*, *20000* and *50000* samples for train, validation and test set. Each set is generated from the corresponding set in the benchmark *mnist-std*. We refer to this benchmark as *mnist-noise*. (Fig.3.4, middle row).
- MNIST digit classification problem where we use a background mask composed of a random picture taken from CIFAR-10 dataset [243]. This benchmark is composed of *100000* samples for training built upon *40000* training samples of CIFAR-10 training set, *20000* samples for validation built upon the rest of CIFAR-10 training set (i.e. *10000* samples) and *50000* samples for test built upon the *10000* test samples of CIFAR-10. We refer to this benchmark as *mnist-img*. (Fig.3.4, bottom row).

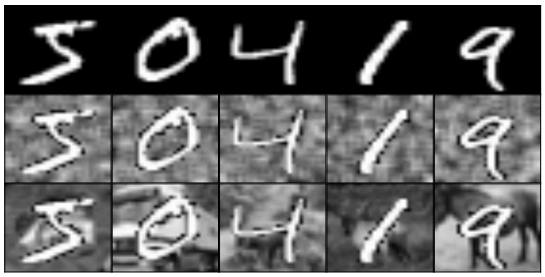


Fig. 3.4 Samples from training set of each benchmark. *Top row*: *mnist-std* benchmark. *Middle row*: *mnist-noise* benchmark. *Bottom row*: *mnist-img* benchmark.

All the images are 28×28 gray-scale values scaled to $[0, 1]$. In order to study the behavior of our proposal where we have few training samples, we use different configurations for the training set size. We consider four configurations where we take only *1000*, *3000*, *5000*, *50000* or *100000* training samples from the whole available training set. We refer to each configuration by *1k*, *3k*, *5k*, *50k* and *100k* respectively. For the benchmark *mnist-std*, only the configurations *1k*, *3k*, *5k* and *50k* are considered.

For all the experiments, we consider the two following neural network architectures:

- Multilayer perceptron with 3 hidden layers followed by a classification output layer. We use the same architecture as in [100] which is $1200 - 1200 - 200$. This model is referred to as *mlp*.

- LeNet convolutional network [257], which is well known in computer vision tasks, (with similar architecture to LeNet-4) with 2 convolution layers with 20 and 50 filters of size 5×5, followed by a dense layer of size 500, followed by a classification output layer. This model is referred to as *lenet*.

Each model has three hidden layers, we refer to each layer from the input toward the output layer by: h_1, h_2 and h_3 respectively. The output layer is referred to as h_4. When using our hint term, we refer to the model by *mlp + hint* and *lenet + hint* for the *mlp* and *lenet* models respectively.

Each experiment is repeated 7 times. The best and the worst test classification error cases are discarded. We report the *mean* $\pm$ *standard deviation* of the validation (vl) and the test (tst) classification error of each benchmark. Models without regularization are trained for *2000* epochs. All the models regularized with our proposal are trained for *400* epochs which we found enough to converge and find a better model over the validation set. All the trainings are performed using stochastic gradient descent with an adaptive learning rate applied using AdaDelta [478], with a batch size of *100*.

Technical Details:

- We found that layers with bounded activation functions such as the logistic sigmoid or the hyperbolic tangent function are more suitable when applying our hint term. Applying the regularization term over a layer with unbounded activation function such as the Relu [310] did not show an improvement.

- In practice, we found that setting $\gamma = 1, \lambda = 1$ works well.

The source code of our implementation is freely available [2].

3.5.2 Results

As we have described in Sec.3.4, our hint term can be applied at any hidden layer of the network. In this section, we perform a set of experiments in order to have an idea about which one is more adequate to use our regularization. To do so, we trained the *mlp* model for classification task over the benchmark *mnist-std* using different configurations with and without regularization. The regularization is applied for one hidden layer at a time h_1, h_2 or h_3. We used the squared Euclidean distance (Eq.3.4) as a dissimilarity measure. The obtained results are presented in Tab.3.1.

From Tab.3.1, it seems that the proposed method decreases systematically the performance when used in layers 1, and 2 in the configuration $1k$. This may be explained by the fact that low layers in neural networks tend to learn low representations which are *shared* among high representations. This means that these representations are not ready yet to discriminate between the classes. Therefore, they can not be used to describe each class separately. This makes our regularization inadequate at these levels because we aim at constraining the representations to be similar within each class

[2] https://github.com/sbelharbi/learning-class-invariant-features

Model/train data size	*1k*		*3k*		*5k*		*50k*	
	vl	tst	vl	tst	vl	tst	vl	tst
	mlp							
	10.49 ± 0.031	11.24 ± 0.050	6.69 ± 0.039	7.17 ± 0.010	5.262 ± 0.030	5.63 ± 0.126	$\mathbf{1.574 \pm 0.016}$	$\mathbf{1.66 \pm 0.016}$
	mlp + hint							
h_3	$\mathbf{8.80 \pm 0.093}$	$\mathbf{9.50 \pm 0.093}$	$\mathbf{5.81 \pm 0.104}$	$\mathbf{6.24 \pm 0.069}$	$\mathbf{4.74 \pm 0.065}$	$\mathbf{5.05 \pm 0.035}$	1.67 ± 0.043	1.73 ± 0.080
h_2	11.48 ± 0.081	12.32 ± 0.090	6.72 ± 0.031	7.29 ± 0.038	5.33 ± 0.031	5.84 ± 0.030	1.88 ± 0.043	1.97 ± 0.071
h_1	12.15 ± 0.043	12.74 ± 0.189	6.75 ± 0.041	7.26 ± 0.049	5.35 ± 0.028	5.87 ± 0.050	1.83 ± 0.033	1.95 ± 0.025

Table 3.1 Mean $\pm$ standard deviation error over validation and test set of the benchmark *mnist-std* using the model *mlp* and the SED as dissimilarity measure over the different hidden layers: h_1, h_2, h_3. (**bold font indicates lowest error.**)

while these layers are incapable to deliver such representations. Therefore, regularizing these layers may hamper their learning. As a future work, we think that it would be beneficial to use at low layers a regularization term that constrains the representations of samples within different classes be dissimilar such as the one in the contrastive loss [82, 178, 63].

In the case of regularizing the last hidden layer h_3, we notice from Tab.3.1 an important improvement in the classification error over the validation and the test set in most configurations. This may be explained by the fact that the representations at this layer are more abstract, therefore, they are able to discriminate the classes. Our regularization term constrains these representations to be tighter by re-enforcing their invariance which helps in generalization. Therefore, applying our hint term over the last hidden layer makes more sense and supports the idea that high layers in neural networks learn more abstract representations. Making these discriminative representations invariant helps the linear output layer in the classification task. For all the following experiments, we apply hint term over the last hidden layer. Moreover, one can notice that our regularization has less impact when adding more training samples. For instance, we reduced the classification test error by: 1.74%, 0.92% and 0.58% in the configurations $1k$, $3k$ and $5k$. This suggests that our proposal is more efficient in the case where few training samples are available. However, this does not exclude using it for large training datasets as we will see later (Tab.3.2, 3.3). We believe that this behavior depends mostly on the model's capacity to learn invariant representations. For instance, from the invariance perspective, convolutional networks are more adapted, conceptually, to process visual content than multilayers perceptrons.

In another experimental setup, we investigated the effect of the measure used to compute the dissimilarity between two feature vectors as described in Sec.3.4.3. To do so, we applied our hint term over the last hidden layer h_3 using the measures SED, NMD and AS over the benchmark *mnist-std*. The obtained results are presented in Tab.3.2. These results show that the squared Euclidean distance performs significantly better than the other measures and has more stability when changing the number of training samples (*1k*, *3k*, *5k*, *50k*) or the model (*mlp*, *lenet*).

In another experiment, we evaluated the benchmarks *mnist-noise* and *mnist-img*, which are more difficult compared to *mnist-std*, using the model *lenet* which is more suitable to process visual content. Similarly to the previous experiments, we applied our regularization term over the last hidden layer h_3 using the SED measure. The results depicted in Tab.3.3 show again that using our proposal improves the generalization error of the network particularly when only few training samples are available. For

Model/train data size	*1k*		*3k*		*5k*		*50K*	
	vl	tst	vl	tst	vl	tst	vl	tst
	MLP							
2-9 *mlp*	10.49 ± 0.031	11.24 ± 0.050	6.69 ± 0.039	7.17 ± 0.010	5.262 ± 0.030	5.63 ± 0.126	1.574 ± 0.016	1.66 ± 0.016
mlp + hint (SED)	$\mathbf{8.80 \pm 0.093}$	$\mathbf{9.50 \pm 0.093}$	$\mathbf{5.81 \pm 0.104}$	$\mathbf{6.24 \pm 0.069}$	$\mathbf{4.74 \pm 0.065}$	$\mathbf{5.05 \pm 0.035}$	1.67 ± 0.043	1.73 ± 0.080
mlp + hint (NMD)	10.32 ± 0.028	10.92 ± 0.094	6.69 ± 0.075	7.22 ± 0.059	5.34 ± 0.035	5.79 ± 0.045	1.44 ± 0.020	1.47 ± 0.020
mlp + hint (AS)	10.27 ± 0.068	10.71 ± 0.123	6.52 ± 0.044	6.89 ± 0.013	4.96 ± 0.041	5.25 ± 0.051	$\mathbf{1.37 \pm 0.023}$	$\mathbf{1.37 \pm 0.025}$
	Lenet							
lenet	6.25 ± 0.016	7.27 ± 0.033	3.65 ± 0.085	4.02 ± 0.073	2.62 ± 0.031	2.90 ± 0.058	1.31 ± 0.028	1.23 ± 0.024
lenet + hint (SED)	$\mathbf{4.54 \pm 0.150}$	5.05 ± 0.115	$\mathbf{2.70 \pm 0.124}$	$\mathbf{2.85 \pm 0.082}$	$\mathbf{2.06 \pm 0.113}$	$\mathbf{2.37 \pm 0.105}$	$\mathbf{0.97 \pm 0.087}$	$\mathbf{1.04 \pm 0.060}$
lenet + hint (NMD)	6.70 ± 0.040	$\mathbf{4.60 \pm 0.065}$	3.85 ± 0.032	4.30 ± 0.036	2.87 ± 0.045	3.14 ± 0.035	1.99 ± 0.043	2.075 ± 0.079
lenet + hint (AS)	6.72 ± 0.024	7.66 ± 0.024	3.86 ± 0.049	4.26 ± 0.049	2.80 ± 0.033	3.12 ± 0.021	1.75 ± 0.123	1.97 ± 0.063

Table 3.2 Mean ± standard deviation error over validation and test set of the benchmark *mnist-std* using different dissimilarity measures (SED, NMD, AS) over the layer h_3. (**bold font indicates lowest error.**)

example, our regularization allows to reduce the classification error over the test set by 2.98% and by 4.16% over the benchmark *mnist-noise* and *mnist-img*, respectively when using only *1k* training samples.

Model/train data size	*1k*		*3k*		*5k*		*100k*	
	vl	tst	vl	tst	vl	tst	vl	tst
	mnist-noise							
lenet	9.62 ± 0.123	10.72 ± 0.116	5.95 ± 0.059	6.39 ± 0.032	4.92 ± 0.036	5.11 ± 0.012	1.90 ± 0.020	2.011 ± 0.018
lenet + hint	$\mathbf{7.12 \pm 0.200}$	$\mathbf{7.74 \pm 0.148}$	$\mathbf{4.09 \pm 0.130}$	$\mathbf{4.62 \pm 0.059}$	$\mathbf{3.53 \pm 0.117}$	$\mathbf{3.98 \pm 0.167}$	$\mathbf{1.60 \pm 0.107}$	$\mathbf{1.64 \pm 0.116}$
	mnist-img							
lenet	13.88 ± 0.114	15.34 ± 0.124	8.34 ± 0.030	8.66 ± 0.024	6.64 ± 0.057	6.46 ± 0.033	2.53 ± 0.080	2.55 ± 0.007
lenet + hint	$\mathbf{10.30 \pm 0.425}$	$\mathbf{11.18 \pm 0.290}$	$\mathbf{6.19 \pm 0.281}$	$\mathbf{6.61 \pm 0.212}$	$\mathbf{5.37 \pm 0.358}$	$\mathbf{5.65 \pm 0.310}$	$\mathbf{2.15 \pm 0.105}$	$\mathbf{2.21 \pm 0.032}$

Table 3.3 Mean ± standard deviation error over validation and test set of the benchmarks *mnist-noise* and *mnist-img* using *lenet* model (regularization applied over the layer h_3). (**bold font indicates lowest error.**)

Based on the above results, we conclude that using our hint term in the context of classification task using neural networks is helpful in improving their generalization error particularly when only few training samples are available. This generalization improvement came at the price of an extra computational cost due the dissimilarity measures between pair of samples. Our experiments showed that regularizing the last hidden layer using the squared Euclidean distance give better results. More generally, the obtained results confirm that guiding the learning process of the intermediate representations of a neural network can be helpful to improve its generalization.

3.5.3 On Learning Invariance within Neural Networks

We show in this section an intriguing property of the learned representations at each layer of a neural network from the invariance perspective. For this purpose and for the sake of simplicity, we consider a binary classification case of the two digits "1" and "7". Furthermore, we consider the *mlp* model over the *lenet* in order to be able to measure the features invariances over all the layers. We trained the *mlp* model over the benchmark *mnist-std* where we used all the available training samples of both digits. The model is trained without our regularization. However, we tracked, at each layer and at the same time, the value of the hint term J_H in Eq.3.3 over the training set using the normalized

Manhattan distance as a dissimilarity measure. This particular dissimilarity measure allows comparing the representations invariance between the different layers due to the normalization of the measure by the representations dimension. The obtained results are depicted in Fig.3.5 where the x-axis represents the number of mini-batches already processed and the y-axis represents the value of the hint term J_H at each layer. Low value of J_H means high invariance (better case) whereas high value of J_H means low invariance.

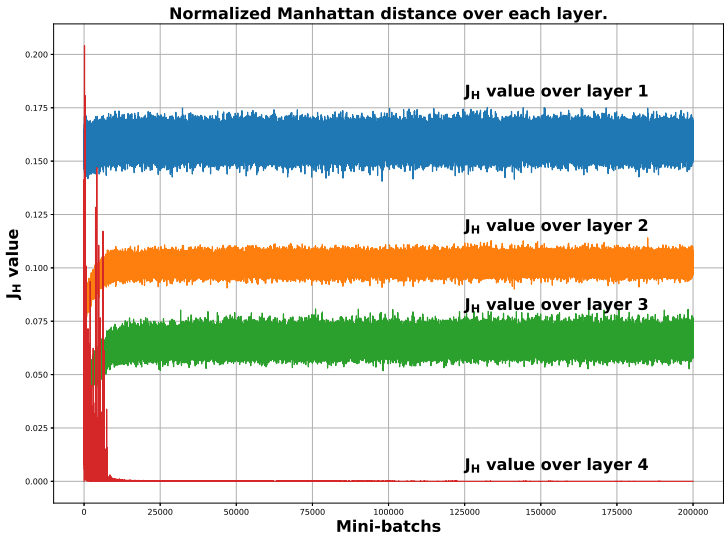


Fig. 3.5 Measuring the hint term J_H of Eq.3.3 over the training set within each layer (simultaneously) of the *mlp* over the train set of *mnist-std* benchmark for a binary classification task: the digit "1" against the digit "7".

In Fig.3.5, we note two main observations:

- The value of the hint term J_H is reduced through the depth of the network which means that the network learns more invariant representations at each layer in this order: layer 1, 2, 3, 4. This result supports the idea that abstract representations, which are known to be more invariant, are learned toward the top layers.
- At each layer, the network does not seem to learn to improve the invariance of the learned representations by reducing J_H. It appears that the representations invariance is kept steady all along the training process. Only the output layer has learned to reduce the value of J_H term because minimizing the classification term J_{sup} reduces automatically our hint term J_H. This shows a flaw in the back-propagation procedure with respect to learning intermediate representations. Assisting the propagated error through regularization can be helpful to guide the hidden layers to learn more suitable representations.

These results show that relying on the classification error propagated from the output layer does not necessarily constrain the hidden layers to learn better representations for classification task. Therefore, one would like to use different prior knowledge to guide the internal layers to learn better representations which is our future work. Using these guidelines can help improving neural networks generalization especially when trained with few samples.

3.6 Conclusion

We have presented in this work a new regularization framework for training neural networks for classification task. Our regularization constrains the hidden layers of the network to learn class-wise invariant representations where samples of the same class have the same representation. Empirical results over MNIST dataset and its variants showed that the proposed regularization helps neural networks to generalize better particularly when few training samples are available which is the case in many real world applications.

Another result based on tracking the representation invariance within the network layers confirms that neural networks tend to learn invariant representations throughout staking multiple layers. However, an intriguing observation is that the invariance level does not seem to be improved, within the same layer, through learning. We found that the hidden layers tend to maintain a certain level of invariance through the training process.

All the results found in this work suggest that guiding the learning process of the internal representations of a neural network can be helpful to train them and improve their generalization particularly when few training samples are available. Furthermore, this shows that the classification error propagated from the output layer does not necessarily train the hidden layers to provide better representations. This encourages us to explore other directions to incorporate different prior knowledge to constrain the hidden layers to learn better representations in order to improve the generalization of the network and be able to train it with less data.

Acknowledgment

This work has been partly supported by the grant ANR-16-CE23-0006 “Deep in France” and benefited from computational means from CRIANN, the contributions of which are greatly appreciated.

Chapter 4

Application: Spotting L3 Slice in CT Scans using Deep Convolutional Network and Transfer Learning

4.1 Prologue

Article Details:

- **Spotting L3 Slice in CT Scans using Deep Convolutional Network and Transfer Learning**. Soufiane Belharbi, Clément Chatelain[1], Romain Hérault[1], Sébastien Adam, Sébastien Thureau, Mathieu Chastan, and Romain Modzelewski. *Computers in Biology and Medicine, 87: 95-103 (2017).*

Context:
We saw previously that the generalization error is bounded by two terms: a training error term and a complexity term (Sec.1.1.3). Moreover, we saw that these two terms are antagonist. Therefore, one needs to strike a balance between these two terms in order to get a better generalization error. Moreover, we concluded that in order to well train models with high capacity, one needs large number of training samples. In this work, we provide a real life application of an idea that allows us to "cheat", i.e., train a model with high capacity using only few samples.

In the last years, many neural network models have seen large success in many tasks such as pattern recognition, particularly deep convolution networks, e.g., Alexnet [244], VGG16 [396], VGG19 [396], Googlenet (Inception V1) [419], which were trained on enormous corpus of labeled data such as ImageNet [105]. This success has attracted many people and motivated the use of such models. However, training such models is time consuming and most importantly requires millions of labeled data. Luckily, the authors of the original models have made available the parameters of the trained models. Many researchers started experimenting using these parameters and adjusting them for their own tasks following a transfer learning paradigm. This idea started

[1] Authors with equal contribution.

to spread to different applications such as character recognition [225, 90], signature identification [179] and medical imaging [24, 389]. The idea consists in 1. taking low layers of the pre-trained network, 2. plug them into a new network, 3. stack on top random fully connected layers, and finally 4. train the whole network on the new task. Using this transfer learning approach, we applied a deep convolutional network to a medical domain problem that lacks data.

We present this work [32] as a chapter of this thesis under the form of one single contribution. This chapter contains the original paper as it was accepted in Computers in Biology and Medicine journal.

Technical context:
This work came as a part of a project developed in the clinic "*Rouen Henri Becquerel Center*" to analyze 3D Computed Tomography (CT) scans. The idea consists in locating a particular vertebra slice (the third lumbar vertebra, i.e., L3) in the 3D CT scan. Then, perform an imagery analysis on it [261]. Our task is to locate the L3 slice. In this work, we provide a complete automated system to locate the L3 slice in a 3D CT scan without any assumptions on which part of the patient's body is covered by the scan.

Contributions:
The contribution of this work is to provide a complete automated system to locate the third lumbar vertebra in a 3D CT scan. The system was validated on a real world data. This work shows that transfer learning can be helpful in the case where only few training samples are available. Moreover, it shows the possibility to apply deep neural networks, particularly convolutional neural networks, on medical images. Furthermore, the provided system is a generic solution which can be used to locate any organ of the patient's body, providing the necessary data.

4.2 Introduction

In recent years, there has been an increasing interest in the analysis of body composition for estimating patient outcomes in many pathologies. For instance, sarcopenia (loss of muscle), visceral and subcutaneous obesity are known prognostic factors in cancers [286, 472], cardiovascular diseases [17] and surgical procedures [339, 230]. Body composition can also be used to improve individual nutritional care and chemotherapy dose calculation [160, 250]. It is usually assessed by CT and Magnetic Resonance Imaging (MRI). Moreover, It has been shown that the composition of the third lumbar vertebra (L3) slice is a good estimator of the whole body measurements [301, 386]. To assess the patient's body composition, radiologists usually have to manually find the corresponding L3 slice in the whole CT exam (spotting step, see Figure 4.1), and then to segment the fat and muscle on a dedicated software platform (segmentation step). These two operations take more than 5 minutes for an experienced radiologist and are prone to errors. Therefore, there is a need for automating these two tasks.

The segmentation step has been extensively addressed in the literature among the medical imaging community [341, 289]. Dedicated approaches for L3 slice have been proposed such as atlas based methods [83] or deep learning [261]. On the other hand, to the best of our knowledge, the automatic spotting of a specific slice within the whole CT scan has not been investigated in the literature. The spotting task is particularly challenging since it has to handle:

- The intrinsic variability in the patient's anatomy (genders, ages, morphologies or medical states).
- The various acquisition/reconstruction protocols (low/high X-rays dose, slice thickness, reconstruction filtering, enhanced/non enhanced contrast agent).
- The arbitrary field-of-view scans, displaying various anatomical regions.
- The strong similarities between the L3 slice and other slices, due to the repetitive nature of vertebrae (Fig.4.2).

In the literature, spotting tasks are often achieved using ad hoc approaches such as registration which are not suitable for high variability problems [147, 97]. In particular, a 3D registration on a whole CT scan would require a large amount of computation at decision time [375]. Here, we suggest a more generic strategy based on machine learning in order to handle high variability context, while maintaining a fast decision process.

In this work, spotting a slice within a CT scan is tackled as a regression problem, where we try to estimate the slice position height. An efficient processing flow is proposed, including a Convolutional Neural Network (CNN) learned using transfer learning. Our approach tackles the classical issues faced in medical image analysis: the data representation issue is addressed using Maximum Intensity Projection (MIP); the variability of the shapes in CT scans is handled using a CNN; and the lack of annotated data is circumvented using transfer learning.

The article is organized as follows: Section 4.3 presents the related work and the general framework for applying machine learning for L3 detection in a CT scan. Section 4.4 presents the proposed approach and describes each stage of the whole processing flow. Section 4.5 describes the experiments and the obtained results.

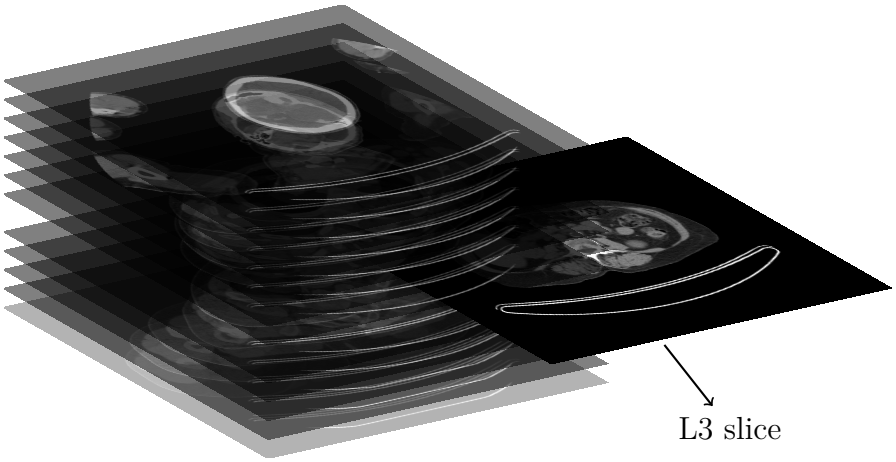


Fig. 4.1 Finding the L3 slice within a whole CT scan.

4.3 Related Work

Machine learning approaches provide generic and flexible systems, provided enough annotated data is available. From a machine learning perspective, the localization of the L3 slice given a whole CT scan can either be considered as a slice-classification problem, a sequence labeling problem or a regression problem. Let us now consider these three options.

The classification paradigm consists of deciding for each slice of the whole CT scan whether the L3 vertebra is present or not. However, the repetitive nature of individual vertebra induces a similarity between the L3 slice and its neighbors, which prevents to efficiently classify an isolated slice without any context (see Fig. 4.2). This explains why even experienced radiologists need to browse the CT scan to infer the relative position and precisely identify the L3 slice. To the best of our knowledge, the classification paradigm has not been used in the literature to detect the L3 slice within a whole CT scan.

The sequence labeling paradigm consists of estimating the label (L1, L2, etc.) of every slice of a complete CT scan, then, choose the one that is more likely to correspond to the L3. The advantage of this approach is that the decision is globally taken on the whole CT scan by analyzing the dependencies between the slices. This kind of approach has been recently investigated for labeling the vertebrae of complete spine images [143, 152, 290, 145, 229, 146, 212, 276, 325]. The dependencies are modeled using graphical models, such as Hidden Markov Models (HMMs) [146] or Markov Random Fields (MRFs) [229]. A full review of the spine labelization methods can be found in [279]. The major drawback of sequence labeling approaches is that they require a fully annotated learning database where every slice of the CT scan is labeled, which is very time consuming. Such a dataset is proposed by [147], but this dataset cannot be easily exploited for our problem since i) the data are cropped images of the whole spine, and ii) it contains only 224 CT scan.

Fig. 4.2 Two slices from the same patient: a L3 (up) and a non L3 (L2) (down). The similar shapes of both vertebrae prevent from taking a robust decision given a single slice.

The regression problem consists of directly estimating a real value that indicates the L3 slice position (i.e., the number of the slice) given the whole CT scan, in a spotting fashion. Like the previous paradigm, it has the advantage of performing a global decision by taking into account the dependencies within the entire exam. Another major advantage of a spotting approach is that it does not require a full labeling of the exams. Indeed, the only annotation needed for learning such a model is the L3 position within the whole exam. For radiologists, this annotation is more lightweight than a full annotation and may lead to creating large datasets easily.

In this work, we retain the third paradigm and propose a machine learning approach for spotting the L3 slice in heterogeneous arbitrary field-of-view CT scans. To the best of our knowledge, this is the first time that slice spotting is addressed as a machine learning regression problem.

Usually, traditional machine learning methods exploit generic hand-designed features which are fed to a learning model with the assumption that they are suitable for describing the image. To achieve high accuracy, usually one ends up combining many types of features which require extensive computation, more time and large memory size. Ideally, it would be better if the model is capable of learning on its own task-dependent features.

Deep neural networks (DNN) are a specific category of models in machine learning which are capable of learning on their own hierarchical features based on the raw image. Convolutional neural networks (CNN) are a particular type of DNN which gained a large reputation in computer vision due to their high performance for many tasks on natural scene images [421, 122, 358, 244].

In the last years, the use of machine learning, in general, and using CNN, in particular, has grown in various medical domains such as cancer diagnosis [366, 434], segmentation [211, 185, 249] or histological [281] and drusen identification [74]. In all these works, the authors are faced with a common issue which is the lack of annotated data. Although extremely powerful, CNN architectures require a huge amount of data to avoid the "learning by heart" phenomenon, also known as overfitting in machine learning. The classical techniques to limit these issues are dropout, data augmentation or the use of regularization. All these technical tricks are exploited in [249], but the lack of data is still a limitation to train such large models. Recently, a more efficient way has been proposed to circumvent the lack of annotated data in vision. This method consists of exploiting models that have been pre-trained on a huge amount of annotated data on another task and is known as "transfer learning".

In this work, we explore the idea of using a CNN model for the localization of the L3 slice using transfer learning. A full description of our approach is presented in section 4.4.

4.4 Proposed Approach

Using a CNN for solving the L3 detection task formulated as a regression problem (see fig. 4.1) is not straightforward, and requires the alleviation of some constraints which are inherent to the medical domain and to the data that is being processed (i) Training a CNN on 3D data such as CT scans requires very large computing and

memory resources that can even exceed the memory limit of most accelerator cards, while such cards are essential for learning a CNN in a reasonable time; (ii) Training a CNN requires fixed size inputs, while the size of the CT scans can vary from one exam to another because of an arbitrary field of view; (iii) Training a CNN requires a large amount of labeled data.

In this paper, we propose to overcome these limitations by using the approach depicted in figure 4.3. In this approach, the CT scan is first converted into another representation using Maximum Intensity Projection (MIP), in order to reduce the dimension of the input from 3D to 2D, without loss of important information. Then, the MIP image is processed in a sliding window fashion to be fed to a CNN with a fixed-size input. This CNN is trained with Transfer Learning (TL-CNN) to solve the requirement of a large amount of labeled examples. Once the trained TL-CNN has computed its prediction for each position of a sliding window, the resulting prediction sequence is processed in order to estimate the final L3 position in the full CT scan. The following subsections detail the three important contributions of the proposed system.

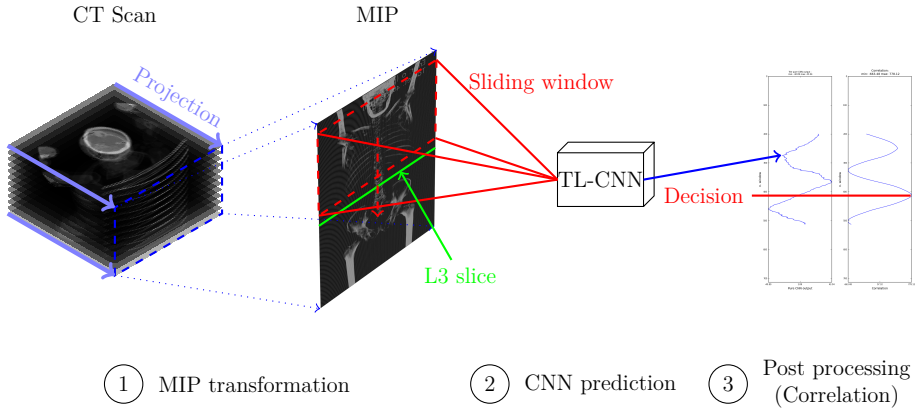


Fig. 4.3 System overview describing the three important stage of our approach : MIP transformation, TL-CNN prediction, and post processing.

4.4.1 MIP Transformation

Ideally, one can use the raw 3D scan image to feed the CNN. If N is the number of slices of the arbitrary field of view CT scan, the input size is $512^2 \times N$. For example, a CT-scan with 1000 slices represents 262M inputs. However, the input size of CNN models strongly impacts their number of parameters. Therefore it would require a very large number of training samples to efficiently learn the CNN. Thus, in the case of few training samples, using the 3D scan directly as an input is not efficient. We believe that the patient's skeleton carries enough visual information in order to detect the L3.

For these reasons, we propose to use a different data representation which focuses on the patient's skeleton and dramatically reduces the size of the input space. This representation is based on a frontal Maximum Intensity Projection (MIP) [446, 444, 445]. The idea is to project a line from a frontal view of the CT scan and retain the maximum intensity over all the voxels that fall into that line. We experimented using different

views such as frontal and lateral views, as well as their combination but they did not work well as compared to the frontal view alone.

Since the slice thickness can vary within the same scan and the voxels are not squared, the projection often generates a distorted MIP. Visually, this gives an unrealistic image where the skeleton is shrunk or enlarged. The cause of this distortion is that, often, the resulting pixel from the projection does not correspond to one voxel. Often, one voxel can be represented by more than one pixel. In order to obtain an equal correspondence (i.e. one pixel corresponds to one voxel), we resize (normalize) the 2D MIP image using an estimated ratio r and average slice thickness s where r represents the number of pixels corresponding to one voxel (slice).

Fig.4.4 shows an example of a normalized frontal MIP image. The MIP transformation reduces the input size from $512^2 \times N$ to $512 \times N$.

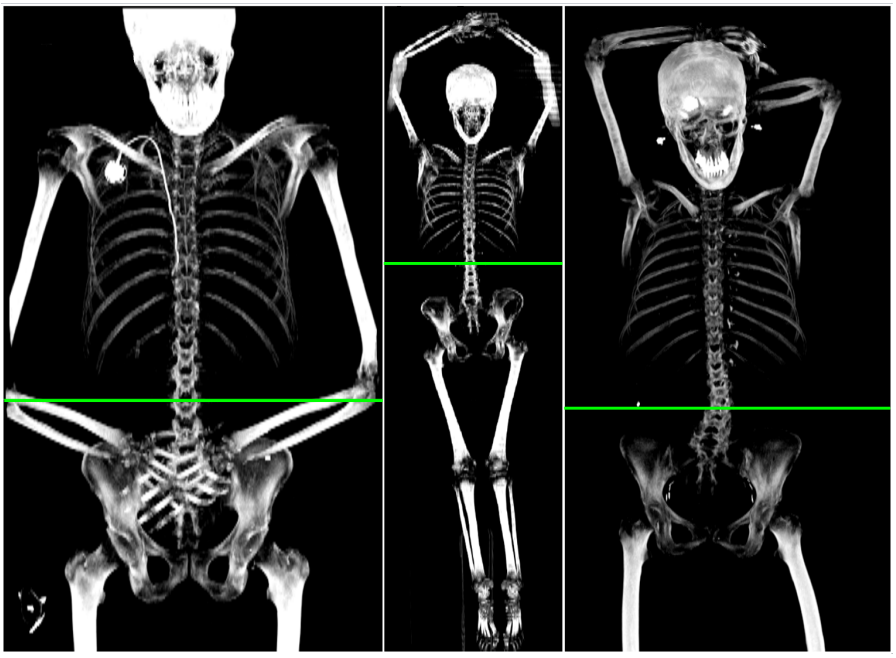


Fig. 4.4 Examples of normalized frontal MIP images with the L3 slice position.

4.4.2 Learning the TL-CNN

Convolutional neural networks (CNN) are particular architecture of neural networks. Their main building block is a convolution layer that performs a non-linear filtering operation. This convolution can be viewed as a feature extractor applied identically over a plane. The values of the convolution kernel constitute the layer parameters. Several convolution layers can be stacked to extract hierarchical features, where each layer builds a set of features from the previous layer. After the convolutional layers, fully connected layers can be stacked to perform the adequate task such as the classification or the regression.

In the learning phase, both parameters of convolutional layers and fully connected layers are optimized according to a loss function. The optimization of these huge

number of parameters is generally performed using stochastic gradient descent method. This process requires a very large number of training samples.

Recently, there has been a growing interest in the exploration of transfer learning methods to overcome the lack of training data. Transfer learning consists in adapting models, trained for different task, to the task in hand (target). It has been applied with success for various applications such as character recognition [225, 90], signature identification [179] or medical imaging [24, 389]. All these contributions exploit CNN architectures which have been pre-trained on computer vision problems, where huge labeled datasets exist. In this framework, the weights of the convolutional layers are initialized with the weights of a pre-trained CNN on another dataset, and then fine-tuned to fit the target application. The fine-tuning starts by transferring only the weights of the convolutional layers from a pre-trained network to the target network. Then, randomly initialized fully connected layers are stacked over the pre-trained convolutional layers and the optimization process is performed on the whole network. This transfer learning framework carried out for our application is illustrated by Figure 4.5 .

A well-known difficulty when using the transfer learning paradigm is to fit the data to the input size of the pre-trained architecture. Since the size of the normalized MIP images varies from one patient to another, two solutions can be considered. The first one consists of resizing the whole scan to a given fixed size. This solution is straightforward but it dramatically impacts the image quality and the output precision. The second solution consists in decomposing the input MIP into a set of fixed-size windows with a sampling strategy. In this paper, we adopt the second approach which enables to preserve the initial quality of the image data.

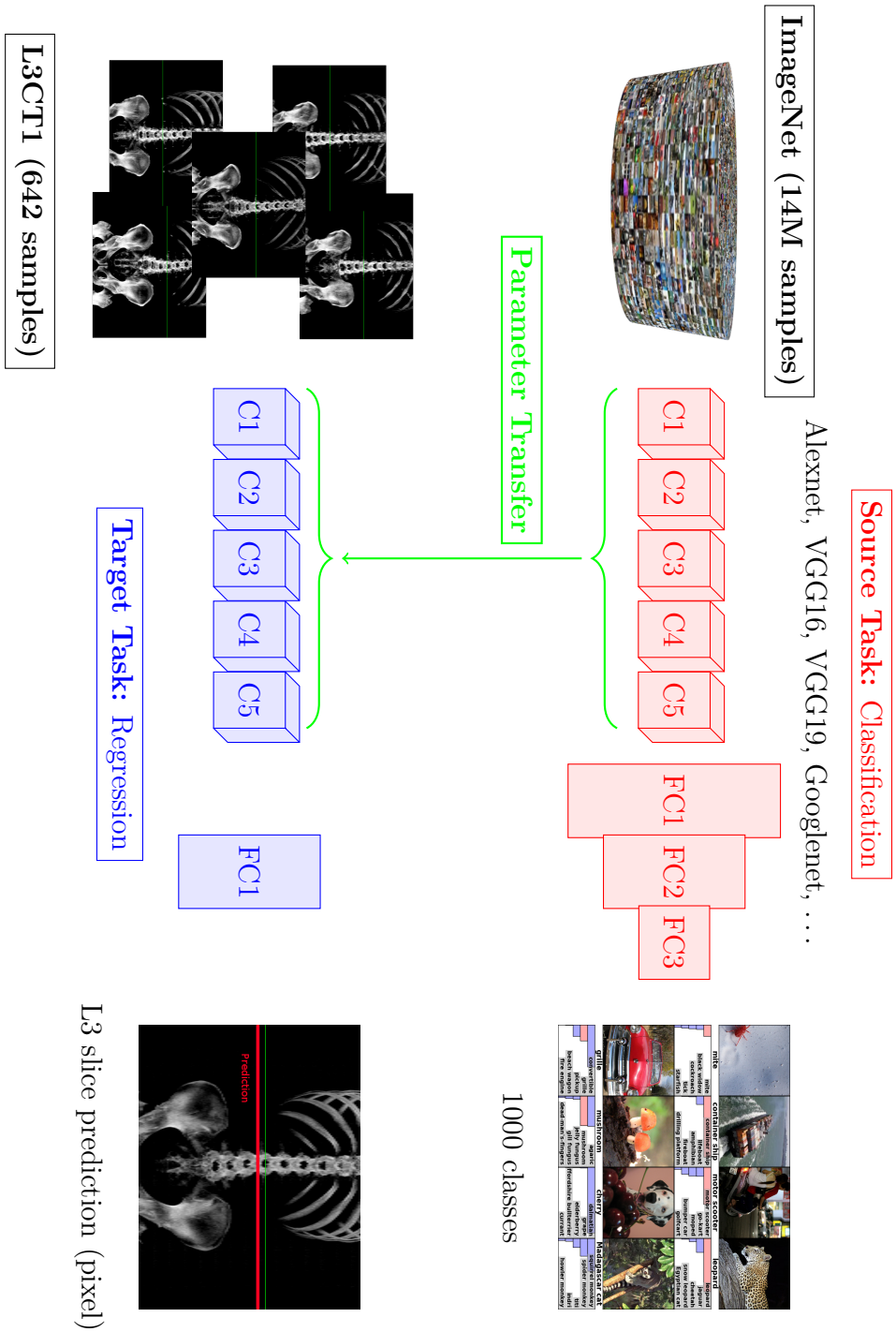


Fig. 4.5 System overview. Layers C_i are Convolutionnal layers, while FC_i denote Full Connected layers. Convolution parameters of previously learnt ImageNet classifier are used as initial values of corresponding L3 regressor layers to overcome the lack of CT examples.

When sampling windows from the MIP image, two sets of window images can be produced. The first one is made of windows containing the L3, and the other one is made of windows without the L3. This raises the question whether the windows without L3 should be present or not in the CNN learning dataset. As we propose a regression approach, adding the non-L3 images in the learning dataset would imply that the CNN learns (and outputs in the decision stage) the offset of the L3 with respect to the current window. Obviously, this offset can be very difficult to learn, particularly if the current window is far from the L3 position. Thus, we have decided to include only the windows containing the L3 in the learning dataset.

Thus, for building the training dataset, we sample all the possible windows of height H such that the L3 position is in the support $[-a, +a]$ where 0 denotes the center of the window. This leads to $2a + 1$ possible windows from each MIP image to be included in the training set. All windows from all MIP are then shuffled: it is highly improbable that two neighboring windows from the same MIP will appear next to each other in the optimization procedure.

4.4.3 Decision Process using a Sliding Window over the MIP Images

A sliding window procedure is applied at the decision phase on the entire MIP image, leading to a sequence of relative L3 position predictions. Such a sequence is illustrated in the left of figure 4.6.

In this sequence, one can observe two distinct behaviors depending on the presence of the L3 in the corresponding window: i) If the L3 is not in the window, the CNN tends to output random values since it has been trained only on images containing L3. This behavior is illustrated in Figure 4.6 at the beginning and (less clearly) at the end of the sequence. ii) If the L3 is within the window, the CNN is expected to predict (correctly) the relative L3 position within the window. Since the L3 position is fixed in the MIP and the window slides line by line on the region of interest, the true relative L3 position should decrease one by one. In consequence, the CNN output should evolve linearly along the sequence of windows, leading to a noisy straight line with a slope of -1[2]. The noise may come from local imprecision or error on an individual slide. This behavior can be observed in figure 4.6 between offset 500 and 600, and it is highlighted with a theoretical orange line.

Therefore, at decision stage, the L3 position can be estimated through the localization of the middle of this particular straight segment. This estimation can easily be achieved by searching the maximum of a simple correlation between the sequence and the expected slope. This procedure, illustrated at the bottom of Fig. 4.6, easily filters out boundary windows which do not contain the L3, and shows robustness by averaging several predictions of the CNN.

[2] $\frac{\Delta y}{\Delta x} = -1$ is the slope of the line where Δx is the moved distance (slided distance caused by moving the window down which is always positive). If we move the window from line x_1 to line $x_2 = x_1 + s$ where s is the stride (i.e., how many lines we move the window down). y is the relative prediction inside the window. If the network predicts y_1 at the window sampled at x_1, therefore, we expected that when we slide the window down by s lines, the relative prediction should move by $-s$. Therefore, $y_2 = y_1 - s$ which means that $\Delta y = -s$. Therefore, we find that $\frac{\Delta y}{\Delta x} = \frac{-s}{s} = -1$.

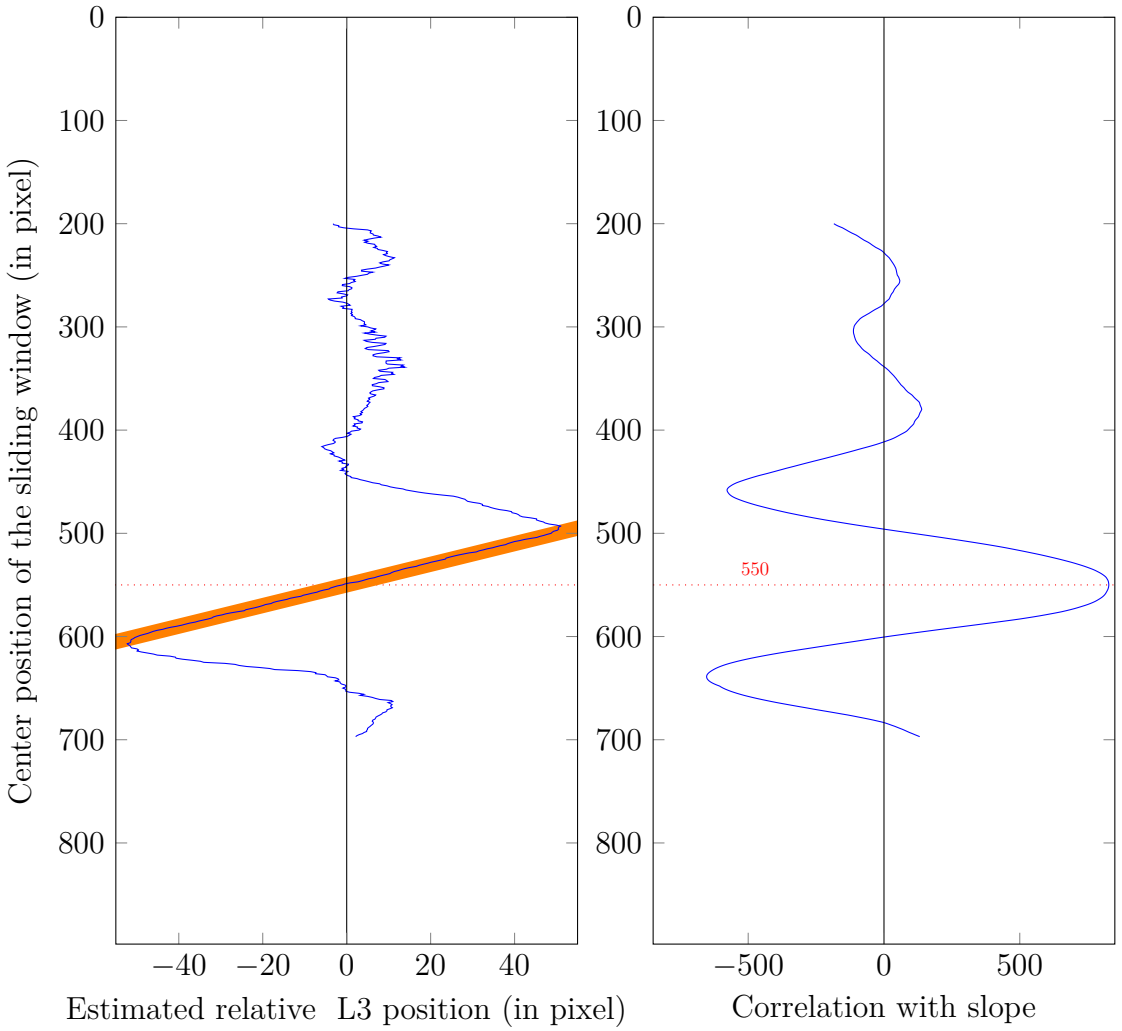


Fig. 4.6 [left]: CNN output sequence obtained for $H = 400$ and $a = 50$ on a test CT scan. The sequence contains the typical straight line of slope -1 centered on the L3 (the theoretical line is plotted in orange), surrounded by random values. [right]: correlation between the CNN output sequence and the theoretical slope. We retain the maximum of correlation as an estimation of the L3 position.

4.5 Experimental Protocol

4.5.1 CT Exams Database Description

In order to validate the proposed approach, a database named L3CT1 has been collected[3]. The main part of the dataset is composed of 642 CT exams from different patients. All patients were included in this study after being informed of the possible use of their images in a retrospective research. The institutional ethical board of the Rouen Henri Becquerel Center approved this study [4]. The CT exams show a high heterogeneity of patients in terms of anatomy, sex, cancer pathologies, position and properties of the reconstructed CT images: 4 scanner models (PET/CT modalities) and 2 manufacturer, acquisition protocols (low dose acquisition (100 to 120 kV) and modulated mAs along the body) axial field of view (FOV) (400 to 500 mm), reconstruction algorithms (Filtered Back Projection (FBP) or iterative reconstruction) and slice thickness (2 to 5 mm).

On each CT scan, the L3 slice was located by an expert radiologist on a dedicated software [250], providing the annotation for the position of the L3 through its distance in (mm) from the first slice in the scan (top).

Moreover, 43 supplementary CT scans have been annotated by the same radiologist and 3 other experts, in order to evaluate the variability of annotations among experts.

To be as reproducible and precise as possible, detailed guidelines were given to all radiologists for annotation.

From all the scans, frontal MIP images have been computed using the process described in 4.4.1. This results in a set of 642 images of constant width (512 pixels) and variable height, varying from 659 to 1862 pixels. Fig 4.4 shows some examples of frontal MIP images extracted from three patients of the L3CT1 database.

4.5.2 Datasets Preparation

The first step consists in splitting the dataset into 5 folds, in order to allow a cross-validation procedure. The split is applied at the patient level, in order to prevent that a given CT-scan provides windows in different sets (learning, validation, test), what should lead to biased results. Moreover, due to variable slice thickness in the dataset, we make sure when dividing the dataset to obtain stratified folds. Thus, we end up with the same number of samples from each slice thickness in each set.

Once the MIP images folds have been generated, learning, validation and test windows are sampled as explained in section 4.4.3, where the value of a has been experimentally set to $a = 50$ using a cross validation procedure. For the validation set, in order to speed up the training, we take only 300 random windows from different patients.

4.5.3 Neural Networks Models

In order to conduct our experiments, two types of convolutional neural networks have been compared:

[3]This dataset is available on demand, please contact the corresponding author

[4]IRB Number 1604B.

- **Homemade CNN (CNN4):** We have designed and trained a CNN from scratch, with specific architecture of four convolutional layers followed by a fully connected output layer. In each convolution layer, a horizontal max-pooling is performed. We found in practice that vertical max-pooling distorts the target position. The number of kernels that we used in the four convolution layers are $[10, 3, 3, 5]$, with respective sizes $[5, 7, 9, 3]$. The hyper-parameters of our CNN were tuned on the validation set [45]. We refer to our model as $CNN4$.

- **Pre-trained CNNs:** In our study, we have collected a set of pre-trained convolutional neural networks over ImageNet dataset [105]: Alexnet [244], VGG16 [396], VGG19 [396], Googlenet (Inception V1) [419][5]. The models are created using the library Keras [81]. For each model, we keep only the convolutional layers which are considered as shared perception layers that may be used for different tasks. On top of that, we add one fully connected layer to be specialized in our specific task (i.e. L3 detection). Our experiments have shown that adding more fully connected layers does not improve the results.

  The input of pre-trained models is supposed to be an RGB image (i.e. a 3D matrix), while in the other hand, our sampled windows are 2D matrix. In order to match the required input, we duplicate the 2D matrix in each color channel. Then, each channel is normalized using its mean from the ImageNet Dataset.

  We use L_2 regularization for training all the models with value of $\lambda = 10^{-3}$, except for Googlenet where we used the original regularization values.

4.6 Results

4.6.1 Data View: Frontal Vs. Lateral

The use of the MIP representation allows us to access to different views of the CT scan, such as the frontal and lateral views (other views with different angles are possible). In order to choose the best view, we re-train a VGG16 model with one fully connected layer using different input views. We recall that the input of the VGG16 is an image with 3 plans. We experimented three configurations. In the first and second cases, we repeat the frontal and lateral views, respectively, in the three input channels. In the last case, we mixed the frontal and the lateral view. The motivation behind the combination of the views is that each view will provide an additional information (hopefully complementary) that will help the model to decide. The sampling margin of the windows is done over the range $[-50, +50]$. Tab.4.1 shows that using frontal view alone is more suitable. One possible explanation of this results is that the frontal view contains more structural context (ribs, pelvis) which helps to locate the L3 slice, in the opposite of the lateral view. Combining lateral and frontal views gave better results than lateral alone but worse than frontal alone. One may think that lateral view adds noise to the frontal view.

[5]The weights of Googlenet were obtained from: https://gist.github.com/joelouismarino/a2ede9ab3928f999575423b9887abd14, and the weights of the rest of the models were obtained from https://github.com/heuritech/convnets-keras

View	VGG16
	Error m_c (slices)
Frontal	$\mathbf{1.71 \pm 1.59}$
Lateral	4.29 ± 14.90
Frontal Lateral Frontal	1.89 ± 2.05

Table 4.1 Test error (mean ± standard deviation) over the test set of fold 0, expressed in slices, using VGG16 model with frontal and lateral views.

4.6.2 Detection Performance

All the models described in section 4.5.3 have been evaluated in a cross validation procedure on the L3CT1 dataset by computing the prediction error. The prediction error for one CT scan is computed as the absolute difference between the prediction y_{pred} and the target y: $e = |y - y_{pred}|$. The error is expressed in slices. We report the mean and the standard deviation of the test error (μ_e, σ_e), respectively in the form $\mu_e \pm \sigma_e$, over the entire test set. Obtained results are reported in Tab.4.2.

For the sake of comparison, we used Random Forest Regression (RF) [61, 203] as a regressor instead of our CNN. As in most pattern recognition problems, we need to extract input features to train our Random Forest Regression. Local Binary Patterns (LBP) features have shown to be very efficient in many computer vision tasks [324], especially in medical imaging [311]. Therefore, we have retained this feature descriptor. To extract the LBP features we used a number of neighbors of 8 and a radius of 3 which creates an input feature vector with dimension of $2^8 = 256$. From each sampled window, we extract LBP features. We investigated different number of trees: 10, 100 and 500. The obtained results showed that random forests do not perform well over this task. We report in Tab.4.2 the results using 500 ($RF500$) trees which are in the same order of performance compared the other cases (i.e. 10 and 100 trees).

	RF500	CNN4	Alexnet	VGG16	VGG19	Googlenet
fold 0	7.31 ± 6.52	2.85 ± 2.37	2.21 ± 2.11	2.06 ± 4.39	1.89 ± 1.77	1.81 ± 1.74
fold 1	11.07 ± 11.42	3.12 ± 2.90	2.44 ± 2.41	1.78 ± 2.09	1.96 ± 2.10	3.84 ± 12.86
fold 2	13.10 ± 13.90	3.12 ± 3.20	2.47 ± 2.38	1.54 ± 1.54	1.65 ± 1.73	2.62 ± 2.52
fold 3	12.03 ± 14.34	2.98 ± 2.38	2.42 ± 2.23	1.96 ± 1.62	1.76 ± 1.75	2.22 ± 1.79
fold 4	8.99 ± 7.83	1.87 ± 1.58	2.69 ± 2.41	1.74 ± 1.96	1.90 ± 1.83	2.20 ± 2.20
Average	10.50 ± 10.80	2.78 ± 2.48	2.45 ± 2.42	$\mathbf{1.82 \pm 2.32}$	1.83 ± 1.83	2.54 ± 4.22

Table 4.2 Error expressed in slice over all the folds using different models: RF500, CNN4 (Homemade model), and Alexnet/VGG16/VGG19/GoogleNet (Pre-trained models).

From Tab.4.2, one can see that pre-trained models perform better than our homemade CNN4 with an improvement of about 35%[6]. In particular, VGG16 showed the best results by an average error of 1.82 ± 2.32 followed by VGG19 with 1.83 ± 1.83.

[6] $(2.78 - 1.82)/2.78 \approx 0.3453 \approx 35\%$.

This result confirms the strong benefit of transfer learning between two different tasks. Moreover, it shows that the convolutional layers can be shared as a perception tool between different tasks with slight adaptation. On the other hand, this illustrates the capability for modeling such task using the pre-trained models.

4.6.3 Processing Time Issues

One must mention that the price we paid in order to reach the performance mentioned above is to increase the complexity of the model. In Table 4.3, we present the number of parameters of each model and the average required time for the prediction of the L3 slice. We observe that VGG16 contains approximately 264 times more parameters than CNN4. Beside the required memory for such models, the real paid cost is the evaluation time during the test phase. Computed on a GPU (Tesla K40), VGG16 requires an average of 13.28 seconds per CT scan while our CNN4 only needs 4.46 second per CT scan.

	Number of parameters	Average forward pass time (seconds/CT scan)
CNN4	55,806	04.46
Alexnet	2,343,297	06.37
VGG16	14,739,777	13.28
VGG19	20,049,473	16.02
Googlenet	6,112,051	17.75

Table 4.3 Number of parameters for different models and average forward pass time per CT scan.

An important factor which affects the evaluation time in these experiments is the number of windows processed by the CNN for a given CT scan. Thus, it is possible to dramatically reduce the computation time by shifting the window by a bigger value than 1 pixel. An experimental evaluation of this strategy with VGG16 has shown that a good compromise between processing time and performance could be obtained for a shift value up to 6 pixels without affecting the localization precision. This sub-sampling reduces the evaluation time from 13.28 seconds/CT scan to 2.36 seconds/CT scan and moved the average localization error from 1.82 ± 2.32 slices to 1.91 ± 2.69 slices, respectively. This shows the robustness of the proposed correlation post-processing.

4.6.4 Comparison with Radiologists

In order to further assess the performance of the proposed approach, an extra set of 43 CT scans was used for test. This particular dataset was annotated by the same radiologist who annotated L3CT1 dataset and also by three other experts. Each annotation was performed at two different times, in order to evaluate the intra-annotator variability. We refer to both annotations by the same expert by *Review 1* and *Review 2*.

Obtained results are illustrated in Tab.4.4. It compares the error made by CNN models with those made by the radiologists, using the radiologist who annotated the L3CT1 dataset as reference. These results corroborate the results provided in Table

4.2 since VGG16 is better than CNN4 with an improvement of about 35% in average for both reviews. The results also demonstrate that radiologists are in average more precise than automatic models with an improvement of about 50%. However, they also show that there exists some variabilities among radiologist annotations and even an intra-annotator variability. This latter is visible in Tab. 4.4 since computed errors for automatic systems vary between both reviews while the automatic system gives the same output, showing that reference values have changed. This illustrates the difficulty of the task of precisely locating the L3 slice and the interest of CNN which does not change its prediction.

Errors (slices) / operator	CNN4	VGG16	Ragiologist #1	Radiologist #2	Radiologist #3
Review1	2.37 ± 2.30	1.70 ± 1.65	0.81 ± 0.97	0.72 ± 1.51	0.51 ± 0.62
Review2	2.53 ± 2.27	1.58 ± 1.83	0.77 ± 0.68	0.95 ± 1.61	0.86 ± 1.30

Table 4.4 Comparison of the performance of both the automatic systems and radiologists. The L3 annotations given by the reference radiologist vary between the two reviews.

4.7 Conclusion

In this paper, we proposed a new and generic pipeline for spotting a particular slice in a CT scan. In our work, we applied our approach to the L3 slice, but it can easily be generalized to other slices, provided a labeled dataset is available.

First, the CT scan is converted into a frontal Maximum Intensity Projection (MIP) image. Afterwards, this representation is processed in a sliding window fashion to be fed to a CNN which is trained using Transfer Learning. In the test phase, all the predictions concerning the position of the L3 within the sliding windows are merged into a robust post-processing stage to take the final decision about the position of the L3 slice in the full CT scan.

Obtained results show that the approach is efficient to precisely detect the target slice. Using a fine-tuned VGG16 network coupled with an adequate decision strategy, the average error is under 2 slices where experienced radiologists can provide annotations that differ of about 1 slice. The computing time is within an acceptable range for clinical applications, and can be further reduced by (i) increasing the shift value (ii) adapting the network architecture by pre-training smaller networks over ImageNet, for example, which has not been studied in this work (iii) and prune the final trained CNN by dropping the less important filters. Recently, pruning CNNs [264, 306, 333] has seen a lot of attention in order to deploy large CNNs on devices with less computation resource. We are currently working on this idea to speedup more the computation.

This contribution confirms the interest of using machine learning and more particularly deep learning in medical problems. One of the main reasons deep learning is not popular in medical domain is the lack of training data. Pre-training the networks over other large dataset will strongly alleviate this problem and encourage the use of such efficient models.

Acknowledgement

This work has been partly supported by the grant ANR-11-JS02-010 LeMon and the grant ANR-16-CE23-0006 “Deep in France”.

General Conclusion and Perspectives

In this thesis, we have tackled the overfitting issue in neural network models particularly in learning scenarios where only few labeled data are available. Regularization is the most common used approach to deal with such issue. In the literature, different regularization methods have been proposed. While each regularization method tackles the overfitting issue differently, we can distinguish a class of methods that uses representation learning as a fundamental mechanism such as dropout, sparse representations, unsupervised/semi-supervised learning, tangent propagation and manifold learning. Following the success of such methods, we address in this thesis the overfitting of neural networks by proposing three different regularization methods based on representation learning paradigm, where each method is adapted to the task in hand. Such methods were designed and validated mainly in learning scenarios where only few labeled data are available which is a challenging task since most successful neural networks are trained using very large number of samples that reaches easily millions. In most real life applications, such large number of samples is not available. However, one still wants to exploit the performance of such models.

In the following, we present a brief description of each of our proposals along with their respective perspectives.

1. Unsupervised learning for structured output problems

In the first contribution, we addressed structured output problems, i.e., a mapping $\mathcal{X} \rightarrow \mathcal{Y}$ where the output is multidimensional and where there are some relations among its components. In this contribution, we proposed to use the unsupervised learning paradigm to learn/discover the hidden structure in the output data. To do so, we proposed a multi-task framework which is composed of a supervised task and two unsupervised tasks; the first unsupervised task learns the input distribution data while the second one learns the output distribution. Explicit incorporating learning the output data structure into the network learning, which is rarely used, has shown to speedup its training, and more importantly, to improve its generalization. Moreover, learning could be achieved in an unsupervised way where the network discovers on its own the underlying structure that may help to perform accurate prediction. Therefore, no need to a supervised intervention to specify what type of relations the network should learn. Furthermore, it allows using unlabeled data and labels only data which showed to help further more improving the generalization error. The application of our framework to facial landmark detection problem showed a speedup of neural networks training and an improvement of their generalization performance.

Although our framework has shown improvements, it still can be improved. For instance, the adaptation scheme of the importance weights of the three tasks can

be achieved differently. Instead of adapting them in terms of training epochs, we can consider an automatic scheme that uses different indicators based on the training and validation errors. Furthermore, one may consider another type of models to learn the output distribution instead of simple auto-encoders. Generative models, such as generative auto-encoders and adversarial auto-encoders [280], seem a good choice to start with. This will lead to probabilistic outputs in the case where multiple decisions are required.

2. The use of prior knowledge for classification

In the second contribution, neural networks regularization is achieved through the use of prior knowledge about the internal representations of the network for a classification task. Prior knowledge can be helpful, in terms of generalization, when dealing with few training data. Since the prior explains a decision rule that helps in generalization, it can guide the learning process to choose a better solution to avoid overfitting. More precisely, to deal with a classification task, we have proposed to integrate the following prior knowledge about the internal representation within a neural network: "samples within the same class should have the same internal representation". In this contribution, our suggestion consists in formulating this prior knowledge as a penalty which is added to the supervised cost in order to be minimized. The proposed penalty constrains the hidden representations to be class-wise invariant. Empirical evidence has showed that incorporating such prior knowledge helps in improving the network generalization when trained with few samples.

Moreover, this work has shown that such class-wise invariance is learned with the increase of depth, i.e., the more the network is deep the more the internal representation is class-wise invariant. However, tracking this invariance in a network did not show its improvement during the learning. This means that the backpropagation algorithm does not always learns understandable/reasonable internal representations and one can do better using prior knowledge. In this work, we exploited only the invariance property of the internal representation in a neural network. Such property creates compact classes where samples within the same class are close to each other. As an extension to this work, which showed promising results, we plan to add a discrimination term that constrains the classes to be far from each other which is an important property in a classification task. For instance, we are planning to use non-linear discriminant analysis tool to learn efficient internal representations. Moreover, we plan to address the issue of multi-modality to choose which pairs are important for optimization. Many works in linear discriminant analysis showed that considering the distance between samples with a uniform importance degrades the performance [411, 127]. Giving high importance to particular pairs leads to better performance. Based on such results, we plan to use a probabilistic framework that provides dynamically a probabilistic importance to each pair. Such probabilistic framework is inspired from ideas in optimal transport [340]. Moreover, we plan to tackle the optimization problem when dealing with multi-task learning where we have the following setup: a main task, and a set of secondary tasks. In such setup, we usually end up with unbalanced gradients between the main task and the secondary tasks. For instance, in low layers in a neural network (the ones close to its input), the supervised (main) gradient is low compared to the secondary tasks (at the same

layer). In practice, this scenario usually leads to degrading the performance of the network. We plan to use a suitable normalization of the gradients to avoid such issue.

3. **Transfer learning and medical domain**
   In our last contribution, we have presented a real-life application to deal with the lack of labeled data in medical domain. The idea consists in using a network with high capacity, such as convolutional networks, which was pre-trained over a different task with abundant data. Then, a subset of its parameters responsible for feature building are extracted and re-used into a new fresh network. This learning procedure falls into the transfer learning paradigm where learned knowledge from a source task is transfered to a target task. This gives more advantage to the target task to start with the parameters of a complex network partially trained. Therefore, only few data are required to adapt the network to fit the target task.

   We applied this learning process to localize the third lumbar vertebra, i.e., L3, over a 3D CT scan. Instead of using the 3D CT scan as a raw input data, we used its frontal projection to obtain a 2D image. This allows reducing processing time. To locate the target, we slide our trained model over the whole CT scan, and perform a prediction over each window. This makes the prediction independent of which part of the patient's body is covered by the scan. Then, we perform a post-processing using a correlation to detect where the network spikes to localize the target. The framework was designed to be task independent, i.e., it could be used to localize any other organ.

   The obtained results from this contribution showed the interest of applying transfer learning in machine learning and exploiting pre-trained deep architectures. However, as a real world application, this work raised an issue which is the running time. Running large networks such as VGG in a production software where many images are processed requires a lot of computation power which is not available in most cases, at least in the clinic. We note that such high complex models are over-parameterized with respect to our task. This motivated us to explore a solution to speed up the computation. Recently, a trend of speeding up convolutional networks, especially when using transfer learning, has raised. This trend is based on pruning a subset of the filters from the network using different strategies [264, 306, 333]. We developed the necessary code for pruning the models used in this work using minimal norm, i.e., filters with low norm are pruned. However, such approach has yet to be evaluated. We recommended the developers team of the clinic to pursue this path. Another possible way to speedup the computation is to reverse the procedure by including the computation power as a model selection criterion. The idea consists in finding a model with complexity that allows running it over average computation machine in acceptable time. Next, this model is pre-trained over a dataset with enough data. Then, its pre-trained filters are extracted and re-used as described above. The advantage of this approach is that we are able to control the model size which is directly involved into the required computation power.

Perspectives

We have presented in this thesis different approaches of using inductive bias to improve the generalization of neural networks. Such inductive bias is based on representation learning paradigm that is one the main reasons of the success of neural networks. In the following, based on the promising results, and the extensive literature study conducted in this thesis, we present two main research directions that we believe they will improve the generalization of neural networks and help us build more understandable neural networks:

1. **Prior knowledge and domain knowledge to improve deep learning**
   Although deep learning based on neural networks has shown impressive results in different domains, we believe that sooner or later, the performance of such models will reach a saturation regime. A situation where adding more data will not improve their performance, even though such models are known to be incredibly eager for data. For now, the hype that we see in the use of deep learning is just a start. This issue is the result of neural network limitations. From our understanding that is based on the reviewed literature and the history of neural networks, we concluded that neural networks are not that *intelligent* as machine learning models. They learn through a mechanical process by repeating over and over the same process until they *memorize* patterns of data. We note that other machine learning models, if not all of them, do the same thing. Humans seem to learn way faster and smarter. One of the main reasons is their heavy use of *prior knowledge* [113] which allows them to learn new tasks in a short time with less effort and less data. We think that using generic prior knowledge in training neural networks is inevitable in order to introduce some intelligence aspects into their learning/behavior. Learning in such framework will more likely require less training samples. One can go further by using domain knowledge, and benefit from the knowledge of experts. Therefore, neural networks need to be modified in order to ease introducing any type of domain knowledge. The main difficulty here is how one can introduce different types of priors to the network ? We suggest two possible ways to do that: either through the architecture by designing new models that take in consideration the prior such as in convolutional networks where the same feature detector is applied all over the image following the prior that the feature may appear in different positions. The other way is through learning, by using constraints such as in regularization.

   Neural networks and deep learning are by far to be considered as intelligent models nor an instance of Artificial Intelligence. We do not think that such models are ready yet to be integrated to drive a car or take control over a robot. Such models are purely mechanical and have an extreme lack in reasoning. Up to now, we do not exactly know how a neural network takes a decision, and it seems reasonable not to trust its decisions particularly in critical tasks. One can qualify deep learning models by *brute-force models*. Unfortunately, in such domain, the performance goes first, and the justification and the proof do not matter. It is an experimental-guided field.

2. **Fundamental research: dictionary learning and deep learning**
   During this thesis, we found out that representation learning paradigm is an important aspect in deep learning models that gives them a powerful capacity to learn complex tasks. While writing this thesis, we had to go back to the 40′s,

the age of birth of neural networks. Years later, Minsky and Papert [296, 297] showed the limitations of shallow neural networks, i.e., perceptrons. The main message of their criticism is that shallow networks can be able to solve complex tasks when they are able to *represent* differently the input signal in hidden layers. Minsky and Papert proposed, at the time, to use perceptron-like layers to find such representations. As much as this proposition is interesting, we think that we can do better. There are many reasons to think so. We recall that back in the 40′s, the goal behind creating perceptrons and neural networks is to build machines, more precisely, intelligent machines. Therefore, such attentions had a large impact on most research directions including the architecture of such models where a clear attempt has appeared to mimic an electronic circuit to be easy to implement in real life. Hence, neural networks, old and modern version, have inherited most of their aspect from circuits. Seeing deep learning today's success, this certainly has advantages. However, we think that such aspect has carried with it many disadvantages as well which may be the reason behind most deep learning issues today. Using perceptron-like as an encoder in the hidden layer can have advantages in, at least, two cases: • Dealing with binary predicates as input: Using a perceptron-like in the hidden layer will allow to learn new predicates based on the input predicate. Such hidden representation can be built by combining different boolean inputs. • Using handcrafted features: In this case, the input signal is composed of different pre-computed features. The most important aspect here is that each dimension represents the same feature. Therefore, combining different features does make sense in order to build a new feature.

For long time, the research community has started using continues inputs, and raw data as input. Therefore, each component of the input signal changes from an example to another, taking as example an image as raw input. This makes perceptron-like hidden layer inefficient.

Another critical aspect in using perceptron-like as a hidden layer is information loss. A neuron is fundamentally used to take a decision. Aligning a set of a neurons to learn a new representation, based a continuous input, is not really optimal especially in low layers. It is clear that there will be a large information loss. We think that one of the reasons that today neural networks are deep is because there is a need to many layers to recover the lost information.

In order to alleviate this issue, and in order to give neural networks a more solid theoretical background, we suggest to keep the idea of Minsky and Papert, i.e., to use hidden representations to represent in a different way the raw input signal. However, instead of using perceptron-like layers, we suggest to use well studied, interpretable, and solid concepts based on approximation theory and data representation such as dictionary learning-like methods [407]. The main idea consists in learning how to well represent data in the hidden layers using flexible tools that we know and control what they do exactly. Dictionary learning [406] provides a strong tool to represent a signal using a fixed number of atoms. In the case of a classification task, one can image a pool of a shared dictionary to represent all the samples independently from the class membership, followed by class-wise dictionaries, followed by a perceptron-like layer for decision step. Possible adaptation to the output can be done by considering dictionary learning

properties in a classification task. The main idea here is to find a new common space to represent all samples. This common space could be modeled using dictionaries, and could exploit their ability to be trained using unsupervised data through reconstruction. It is clear that one can build easily hierarchical representations following this scheme. However, the first issue of using dictionaries is security, since the model will explicitly memorize chunks of data.

Following this approach, one may go further to exploit more solid theoretical frameworks for data representation such as tensor decomposition [345, 240] in order to build more intelligent layers that are able to decompose a complex signal into elementary elements.

References

[1] Abadi, M., Chu, A., Goodfellow, I., McMahan, H. B., Mironov, I., Talwar, K., and Zhang, L. (2016). Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 308–318, New York, NY, USA. ACM.

[2] Abu-Mostafa, Y. S. (1990). Learning from hints in neural networks. *Journal of Complexity*, 6(2):192–198.

[3] Abu-Mostafa, Y. S. (1992). A method for learning from hints. In *Advances in Neural Information Processing Systems 5, [NIPS Conference, Denver, Colorado, USA, November 30 - December 3, 1992]*, pages 73–80.

[4] Abu-Mostafa, Y. S. (1993). Hints and the vc dimension. *Neural Computation*, 5(2):278–288.

[5] Abu-Mostafa, Y. S., Magdon-Ismail, M., and Lin, H.-T. (2012). *Learning From Data*. AMLBook.

[6] Ackley, D. H., Hinton, G. E., and Sejnowski, T. J. (1985). A learning algorithm for Boltzmann machines. *Cognitive Science*, 9:147–169.

[7] Aggarwal, C. C. (2014). *Data Classification: Algorithms and Applications*. Chapman & Hall/CRC, 1st edition.

[8] Aizenberg, I. N., Aizenberg, N. N., and Vandewalle, J. P. (2000). *Multi-Valued and Universal Binary Neurons: Theory, Learning and Applications*. Kluwer Academic Publishers, Norwell, MA, USA.

[9] Anastassiou, G. A. (2016). *Intelligent Systems II: Complete Approximation by Neural Network Operators*. Springer Publishing Company, Incorporated, 1st edition.

[10] Anastassiou, G. A. and Duman, O. (2016). *Intelligent Mathematics II: Applied Mathematics and Approximation Theory*, volume 441. Springer.

[11] Anastassiou, G. A. and Gal, S. G. (2002). Approximation theory. moduli of continuity and global smoothness preservation.

[12] Andersen, P. (2018). Deep reinforcement learning using capsules in advanced game environments. *CoRR*, abs/1801.09597.

[13] Argyriou, A., Evgeniou, T., and Pontil, M. (2006). Multi-task feature learning. In *Advances in Neural Information Processing Systems 19, Proceedings of the Twentieth Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 4-7, 2006*, pages 41–48.

[14] Argyriou, A., Micchelli, C. A., Pontil, M., and Ying, Y. (2007). A spectral regularization framework for multi-task structure learning. In *Advances in Neural Information Processing Systems 20, Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 3-6, 2007*, pages 25–32.

[15] Arpit, D., Jastrzebski, S. K., Ballas, N., Krueger, D., Bengio, E., Kanwal, M. S., Maharaj, T., Fischer, A., Courville, A. C., Bengio, Y., and Lacoste-Julien, S. (2017). A closer look at memorization in deep networks. In *ICML*, volume 70 of *Proceedings of Machine Learning Research*, pages 233–242. PMLR.

[16] Arulkumaran, K., Deisenroth, M. P., Brundage, M., and Bharath, A. A. (2017). Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38.

[17] Atkins, J. L., Whincup, P. H., Morris, R. W., Lennon, L. T., Papacosta, O., and Wannamethee, S. G. (2014). Sarcopenic obesity and risk of cardiovascular disease and mortality: a population-based cohort study of older men. *Journal of the American Geriatrics Society*, 62(2):253–60.

[18] Auli, M., Galley, M., Quirk, C., and Zweig, G. (2013). Joint language and translation modeling with recurrent neural networks. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, EMNLP 2013, 18-21 October 2013, Grand Hyatt Seattle, Seattle, Washington, USA, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1044–1054.

[19] Ba, L. J. and Caruana, R. (2014). Do deep nets really need to be deep? In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, pages 2654–2662, Cambridge, MA, USA. MIT Press.

[20] Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473.

[21] Baird, H. (1990). Document image defect models. In *Proceddings, IAPR Workshop on Syntactic and Structural Pattern Recognition*, Murray Hill, NJ.

[22] Baldi, P., Brunak, S., Frasconi, P., Soda, G., and Pollastri, G. (1999). Exploiting the past and the future in protein secondary structure prediction. *Bioinformatics*, 15(11):937–946.

[23] Ballard, D. H. (1987). Modular learning in neural networks. In *Proc. AAAI*, pages 279–284.

[24] Bar, Y., Diamant, I., Wolf, L., and Greenspan, H. (2015). Deep learning with non-medical training used for chest pathology identification. *Proc. SPIE, Medical Imaging: Computer-Aided Diagnosis*, 9414:94140V–7.

[25] Barron, A. R. (1993). Universal approximation bounds for superpositions of a sigmoidal function. *Information Theory, IEEE Transactions on*, 39(3):930–945.

[26] Baxter, J. (2000). A model of inductive bias learning. *J. Artif. Int. Res.*, 12(1):149–198.

[27] Belanger, D. and McCallum, A. (2016). Structured prediction energy networks. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pages 983–992.

[28] Belharbi, S., Chatelain, C., Hérault, R., and Adam, S. (2015a). Learning structured output dependencies using deep neural networks. *Deep Learning Workshop in the 32nd International Conference on Machine Learning (ICML)*.

[29] Belharbi, S., Chatelain, C., Hérault, R., and Adam, S. (2015b). A unified neural based model for structured output problems. *Conférence Francophone sur l'Apprentissage Automatique (CAP)*.

[30] Belharbi, S., Chatelain, C., Hérault, R., and Adam, S. (2017a). Neural networks regularization through class-wise invariant representation learning. *XXX*, XX(X):XXXX–XXXX.

[31] Belharbi, S., Chatelain, C., Hérault, R., Adam, S., Thureau, S., Chastan, M., and Modzelewski, R. (2017b). Spotting l3 slice in ct scans using deep convolutional network and transfer learning. *Computers in Biology and Medicine*, 87:95 – 103.

[32] Belharbi, S., Chatelain, C., Hérault, R., Adam, S., Thureau, S., Chastan, M., and Modzelewski, R. (2017c). Spotting l3 slice in ct scans using deep convolutional network and transfer learning. *Computers in Biology and Medicine*, 87:95 – 103.

[33] Belharbi, S., Hérault, R., Chatelain, C., and Adam, S. (2016a). Deep multi-task learning with evolving weights. In *European Symposium on Artificial Neural Networks (ESANN)*.

[34] Belharbi, S., Hérault, R., Chatelain, C., and Adam, S. (2016b). Pondération dynamique dans un cadre multi-tâche pour réseaux de neurones profonds. *Reconnaissance des Formes et l'Intelligence Artificielle (RFIA) (Session spéciale "Apprentissage et vision")*.

[35] Belharbi, S., Hérault, R., Chatelain, C., and Adam, S. (2018). Deep neural networks regularization for structured output prediction. *Neurocomputing*, 281C:169 – 177.

[36] Belharbi, S., R.Hérault, Chatelain, C., and Adam, S. (2016c). Deep multi-task learning with evolving weights. In *European Symposium on Artificial Neural Networks (ESANN)*.

[37] Belhumeur, P. N., Jacobs, D. W., Kriegman, D. J., and Kumar, N. (2011). Localizing parts of faces using a consensus of exemplars. In *CVPR*, pages 545–552. IEEE.

[38] Bellman, R. (1957). *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1st edition.

[39] Ben-David, S., Blitzer, J., Crammer, K., Kulesza, A., Pereira, F., and Vaughan, J. W. (2010). A theory of learning from different domains. *Machine Learning*, 79(1):151–175.

[40] Ben-David, S., Blitzer, J., Crammer, K., and Pereira, F. (2006). Analysis of representations for domain adaptation. In *Advances in Neural Information Processing Systems 19, Proceedings of the Twentieth Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 4-7, 2006*, pages 137–144.

[41] Ben-David, S. and Borbely, R. S. (2008). A notion of task relatedness yielding provable multiple-task learning guarantees. *Machine Learning*, 73(3):273–287.

[42] Ben-David, S., Gehrke, J., and Schuller, R. (2002). A theoretical framework for learning from a pool of disparate data sources. In *KDD*, pages 443–449. ACM.

[43] Ben-David, S. and Schuller, R. (2003). Exploiting task relatedness for multiple task learning. In Schölkopf, B. and Warmuth, M. K., editors, *Learning Theory and Kernel Machines*, pages 567–580, Berlin, Heidelberg. Springer Berlin Heidelberg.

[44] Bengio, Y. (2009). Learning Deep Architectures for AI. *Found. Trends Mach. Learn.*, 2(1):1–127.

[45] Bengio, Y. (2012). Practical recommendations for gradient-based training of deep architectures. In Montavon, G., Orr, G. B., and Müller, K.-R., editors, *Neural Networks: Tricks of the Trade (2nd ed.)*, volume 7700 of *Lecture Notes in Computer Science*, pages 437–478. Springer.

[46] Bengio, Y. (2013). Deep learning of representations: Looking forward. *CoRR*, abs/1305.0445.

[47] Bengio, Y., Courville, A., and Vincent, P. (2013a). Representation learning: A review and new perspectives. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(8):1798–1828.

[48] Bengio, Y., Courville, A. C., and Vincent, P. (2013b). Representation Learning: A Review and New Perspectives. *IEEE PAMI*, 35(8):1798–1828.

[49] Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. (2006). Greedy layer-wise training of deep networks. In *Advances in Neural information Processing Systems 19, NIPS 2006*, pages 153–160.

[50] Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. (2007). Greedy Layer-Wise Training of Deep Networks. In Schölkopf, B., Platt, J., and Hoffman, T., editors, *NIPS*, pages 153–160.

[51] Bengio, Y. and Lecun, Y. (2007). *Scaling learning algorithms towards AI*. MIT Press.

[52] Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *Trans. Neur. Netw.*, 5(2):157–166.

[53] Bermingham, M. L., Pong-Wong, R., Spiliopoulou, A., Hayward, C., Rudan, I., Campbell, H., Wright, A. F., Wilson, J. F., Agakov, F., Navarro, P., et al. (2015). Application of high-dimensional feature selection: evaluation for genomic prediction in man. *Scientific reports*, 5:10312.

[54] Bikel, D. M., Schwartz, R., and Weischedel, R. M. (1999). An algorithm that learns what's in a name. *Machine learning*, 34(1-3):211–231.

[55] Bishop, C. (1995). Regularization and complexity control in feed-forward networks. In *Proceedings International Conference on Artificial Neural Networks ICANN'95*, volume 1, page 141–148. EC2 et Cie.

[56] Blitzer, J., Crammer, K., Kulesza, A., Pereira, F., and Wortman, J. (2007). Learning bounds for domain adaptation. In *Advances in Neural Information Processing Systems 20, Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 3-6, 2007*, pages 129–136.

[57] Bonilla, E. V., Chai, K. M. A., and Williams, C. K. I. (2007). Multi-task gaussian process prediction. In *Advances in Neural Information Processing Systems 20, Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 3-6, 2007*, pages 153–160.

[58] Bousmalis, K., Trigeorgis, G., Silberman, N., Krishnan, D., and Erhan, D. (2016). Domain separation networks. In *NIPS*, pages 343–351.

[59] Boyd, S. and Vandenberghe, L. (2004). *Convex Optimization*. Cambridge University Press, New York, NY, USA.

[60] Breiman, L. (1996). Bagging predictors. *Machine Learning*, 24(2):123–140.

[61] Breiman, L. (2001). Random forests. *Mach. Learn.*, 45(1):5–32.

[62] Bridle, J. S. and Cox, S. (1990). Recnorm: Simultaneous normalisation and classification applied to speech recognition. In *NIPS*, pages 234–240. Morgan Kaufmann.

[63] Bromley, J., Guyon, I., LeCun, Y., Säckinger, E., and Shah, R. (1994). Signature verification using a "siamese" time delay neural network. In Cowan, J. D., Tesauro, G., and Alspector, J., editors, *Advances in Neural Information Processing Systems 6*, pages 737–744. Morgan-Kaufmann.

[64] Bryson, A. and Ho, Y. (1969). *Applied optimal control: optimization, estimation, and control*. Blaisdell Pub. Co.

[65] Bryson, A. E. (1961). A gradient method for optimizing multi-stage allocation processes. In *Proc. Harvard Univ. Symposium on digital computers and their applications*.

[66] Bryson, Jr., A. E. and Denham, W. F. (1961). A steepest-ascent method for solving optimum programming problems. Technical Report BR-1303, Raytheon Company, Missle and Space Division.

[67] Candes, E. J. and Tao, T. (2005). Decoding by linear programming. *IEEE Trans. Inf. Theor.*, 51(12):4203–4215.

[68] Carlini, N., Liu, C., Kos, J., Erlingsson, Ú., and Song, D. (2018). The secret sharer: Measuring unintended neural network memorization & extracting secrets. *CoRR*, abs/1802.08232.

[69] Caruana, R. (1993). Multitask learning: A knowledge-based source of inductive bias. In *ICML*, pages 41–48. Morgan Kaufmann.

[70] Caruana, R. (1997). Multitask learning. *Machine Learning*, 28(1):41–75.

[71] Chapelle, O., Schölkopf, B., and Zien, A. (2006). *Semi-supervised learning*. Adaptive computation and machine learning. MIT Press.

[72] Chartrand, R. (2007). Exact reconstruction of sparse signals via nonconvex minimization. *IEEE Signal Processing Letters*, 14(10):707–710.

[73] Chartrand, R. (2009). Fast algorithms for nonconvex compressive sensing: Mri reconstruction from very few data. In *2009 IEEE International Symposium on Biomedical Imaging: From Nano to Macro*, pages 262–265.

[74] Checco, P. and Corinto, F. (2006). Cnn-based algorithm for drusen identification. In *International Symposium on Circuits and Systems*.

[75] Chen, J. and Chaudhari, N. S. (2004). Capturing long-term dependencies for protein secondary structure prediction. In *Advances in Neural Networks - ISNN 2004, International Symposium on Neural Networks, Dalian, China, August 19-21, 2004, Proceedings, Part II*, pages 494–500.

[76] Chen, M., Xu, Z. E., Weinberger, K. Q., and Sha, F. (2012). Marginalized denoising autoencoders for domain adaptation. In *ICML*. icml.cc / Omnipress.

[77] Chen, X., Xu, F., and Ye, Y. (2009). Lower Bound Theory of Nonzero Entries in Solutions of l2-lp Minimization. Technical report, The Hong Kong Polytechnic University.

[78] Chi, L. and Mu, Y. (2017). Deep steering: Learning end-to-end driving model from spatial and temporal visual cues. *CoRR*, abs/1708.03798.

[79] Chicco, D., Sadowski, P., and Baldi, P. (2014). Deep autoencoder neural networks for gene ontology annotation predictions. In *Proceedings of the 5th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics*, BCB '14, pages 533–540, New York, NY, USA. ACM.

[80] Cho, K., van Merrienboer, B., Bahdanau, D., and Bengio, Y. (2014). On the properties of neural machine translation: Encoder-decoder approaches. In *Proceedings of SSST@EMNLP 2014, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation, Doha, Qatar, 25 October 2014*, pages 103–111.

[81] Chollet, F. (2015). Keras. https://github.com/fchollet/keras.

[82] Chopra, S., Hadsell, R., and LeCun, Y. (2005). Learning a similarity metric discriminatively, with application to face verification. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2005), 20-26 June 2005, San Diego, CA, USA*, pages 539–546.

[83] Chung, H., Cobzas, D., Birdsell, L., Lieffers, J., and Baracos, V. (2009). Automated segmentation of muscle and adipose tissue on CT images for human body composition analysis. *Proceedings of SPIE*, 7261:72610K–72610K–8.

[84] Chung, J., Gülçehre, Ç., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv e-prints*, abs/1412.3555. Presented at the Deep Learning workshop at NIPS2014.

[85] Chung, J., Çaglar Gülçehre, Cho, K., and Bengio, Y. (2015). Gated feedback recurrent neural networks. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 2067–2075.

[86] Cireşan, D. C., Meier, U., Gambardella, L. M., and Schmidhuber, J. (2010a). Deep, big, simple neural nets for handwritten digit recognition. *Neural Comput.*, 22(12):3207–3220.

[87] Cireşan, D. C., Meier, U., Gambardella, L. M., and Schmidhuber, J. (2010b). Deep, big, simple neural nets for handwritten digit recognition. *Neural Computation*, 22(12):3207–3220.

[88] Ciresan, D., Meier, U., and Schmidhuber, J. (2012a). Multi-column deep neural networks for image classification. In *IN PROCEEDINGS OF THE 25TH IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION (CVPR 2012*, pages 3642–3649.

[89] Ciresan, D. C., Giusti, A., Gambardella, L. M., and Schmidhuber, J. (2012b). Deep neural networks segment neuronal membranes in electron microscopy images. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States.*, pages 2852–2860.

[90] Cireşan, D. C., Meier, U., and Schmidhuber, J. (2012). Transfer learning for latin and chinese characters with deep neural networks. In *International Joint Conference on Neural Networks*, pages 1–6.

[91] Collobert, R. and Weston, J. (2008a). A unified architecture for natural language processing: deep neural networks with multitask learning. In *Machine Learning, Proceedings of he 25th International Conference, ICML 2008*, pages 160–167.

[92] Collobert, R. and Weston, J. (2008b). A unified architecture for natural language processing: deep neural networks with multitask learning. In *Machine Learning, Proceedings of the Twenty-Fifth International Conference (ICML 2008), Helsinki, Finland, June 5-9, 2008*, pages 160–167.

[93] Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20(3):273–297.

[94] Crammer, K., Kearns, M. J., and Wortman, J. (2008). Learning from multiple sources. *Journal of Machine Learning Research*, 9:1757–1774.

[95] Cristinacce, D. and Cootes, T. (2006). Feature Detection and Tracking with Constrained Local Models. In *BMVC*, pages 95.1–95.10.

[96] Csáji, B. C. (2001). Approximation with artificial neural networks. Master's thesis, Faculty of Sciences, Etvs Lornd University, Hungary, Hungary.

[97] Cunliffe, A., White, B., Justusson, J., Straus, C., Malik, R., Hallaq, A.-H., and Armato, S. (2015). Comparison of Two Deformable Registration Algorithms in the Presence of Radiologic Change Between Serial Lung CT Scans. *Journal of Digital Imaging*, 28(6):755–760.

[98] Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314.

[99] Dai, W., Yang, Q., Xue, G., and Yu, Y. (2007). Boosting for transfer learning. In *Machine Learning, Proceedings of the Twenty-Fourth International Conference (ICML 2007), Corvallis, Oregon, USA, June 20-24, 2007*, pages 193–200.

[100] De Vries, H., Memisevic, R., and Courville, A. (2016). Deep learning vector quantization. In *European Symposium on Artificial Neural Networks (ESANN)*.

[101] Dean, J., Corrado, G. S., Monga, R., Chen, K., Devin, M., Le, Q. V., Mao, M. Z., Ranzato, M., Senior, A., Tucker, P., Yang, K., and Ng, A. Y. (2012). Large scale distributed deep networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1223–1231, USA. Curran Associates Inc.

[102] Dechter, R. (1986). Learning while searching in constraint-satisfaction-problems. In Kehler, T., editor, *AAAI*, pages 178–185. Morgan Kaufmann.

[103] Demyanov, S. (2015). *Regularization methods for neural networks and related models*. PhD thesis, The University of Melbourne, Department of Computing and Information Systems.

[104] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009a). ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*.

[105] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Li, F.-F. (2009b). Imagenet: A large-scale hierarchical image database. In *CVPR*, pages 248–255.

[106] Deng, L. and Yu, D. (2014). Deep learning: Methods and applications. *Found. Trends Signal Process.*, 7(3–4):197–387.

[107] Desjardins, G., Simonyan, K., Pascanu, R., and kavukcuoglu, k. (2015). Natural neural networks. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems 28*, pages 2071–2079. Curran Associates, Inc.

[108] Doersch, C. (2016). Tutorial on variational autoencoders. *CoRR*, abs/1606.05908.

[109] Donoho, D. (2004). For most large underdetermined systems of linear equations the minimal l1 -norm solution is also the sparsest Solution. Technical report, Stanford University.

[110] Dosovitskiy, A., Springenberg, J. T., Tatarchenko, M., and Brox, T. (2017). Learning to generate chairs, tables and cars with convolutional networks. *IEEE Trans. Pattern Anal. Mach. Intell.*, 39(4):692–705.

[111] Dozat, T. (2016). Incorporating nesterov momentum into adam.

[112] Dreyfus, S. E. (1962). The numerical solution of variational problems. *Journal of Mathematical Analysis and Applications*, 5(1):30–45.

[113] Dubey, R., Agrawal, P., Pathak, D., Griffiths, T. L., and Efros, A. A. (2018). Investigating human priors for playing video games.

[114] Duchi, J., Hazan, E., and Singer, Y. (2010). Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. In *COLT*, pages 257–269.

[115] Duong, L., Cohn, T., Bird, S., and Cook, P. (2015). Low resource dependency parsing: Cross-lingual parameter sharing in a neural network parser. In *ACL (2)*, pages 845–850. The Association for Computer Linguistics.

[116] Dwork, C. (2011). A firm foundation for private data analysis. *Commun. ACM*, 54(1):86–95.

[117] Dwork, C., McSherry, F., Nissim, K., and Smith, A. D. (2006). Calibrating noise to sensitivity in private data analysis. In *TCC*, volume 3876 of *Lecture Notes in Computer Science*, pages 265–284. Springer.

[118] Dwork, C. and Roth, A. (2014). The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4):211–407.

[119] El-Yacoubi, M., Gilloux, M., and Bertille, J.-M. (2002). A statistical approach for phrase location and recognition within a text line: An application to street name recognition. *IEEE PAMI*, 24(2):172–188.

[120] Ellis, H. C. (1965). *Invariant Subspaces*. Macmillan, New York.

[121] Erhan, D., Bengio, Y., Courville, A., Manzagol, P.-A., Vincent, P., and Bengio, S. (2010). Why does unsupervised pre-training help deep learning? *J. Mach. Learn. Res.*, 11:625–660.

[122] Erhan, D., Szegedy, C., Toshev, A., and Anguelov, D. (2014). Scalable object detection using deep neural networks. In *CVPR*, pages 2155–2162.

[123] Evgeniou, T. and Pontil, M. (2004). Regularized multi–task learning. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Seattle, Washington, USA, August 22-25, 2004*, pages 109–117.

[124] Fahlman, S. E., Hinton, G. E., and Sejnowski, T. J. (1983). Massively parallel architectures for AI: netl, thistle, and boltzmann machines. In *Proceedings of the National Conference on Artificial Intelligence. Washington, D.C., August 22-26, 1983.*, pages 109–113.

[125] Fan, J. and R., L. (2001). Variable selection via nonconcave penalized likelihood and its oracle properties. *Journal of the American Statistical Association*, 96:1348–1360.

[126] Farabet, C., Couprie, C., Najman, L., and LeCun, Y. (2013). Learning Hierarchical Features for Scene Labeling. *IEEE PAMI*, 35(8):1915–1929.

[127] Flamary, R., Cuturi, M., Courty, N., and Rakotomamonjy, A. (2018). Wasserstein discriminant analysis. *Machine learning*.

[128] Fridman, L., Jenik, B., and Terwilliger, J. (2018). Deeptraffic: Driving fast through dense traffic with deep reinforcement learning. *CoRR*, abs/1801.02805.

[129] Fridman, M. (1993). *Hidden markov model regression*. PhD thesis, Graduate School of Arts and Sciences, University of Pennsylvania.

[130] Fukada, T., Schuster, M., and Sagisaka, Y. (1999). Phoneme boundary estimation using bidirectional recurrent neural networks and its applications. *Systems and Computers in Japan*, 30(4):20–30.

[131] Fukushima, K. (1979). Neural network model for a mechanism of pattern recognition unaffected by shift in position - Neocognitron. *Trans. IECE*, J62-A(10):658–665.

[132] Fukushima, K. (1980). Neocognitron: A self-organizing neural network for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202.

[133] Fukushima, K. (2011). Increasing robustness against background noise: visual pattern recognition by a Neocognitron. *Neural Networks*, 24(7):767–778.

[134] Fukushima, K. (2013). Training multi-layered neural network Neocognitron. *Neural Networks*, 40:18–31.

[135] Fukushima, K. and Miyake, S. (1982). Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position. *Pattern Recognition*, 15(6):455–469.

[136] Funahashi, K. (1989). On the approximate realization of continuous mappings by neural networks. *Neural Networks*, 2(3):183–192.

[137] Gamba, A., Gamberini, L., Palmieri, G., and Sanna, R. (1961). Further experiments with papa. *Il Nuovo Cimento (1955-1965)*, 20(2):112–115.

[138] Ganin, Y., Ustinova, E., Ajakan, H., Germain, P., Larochelle, H., Laviolette, F., Marchand, M., and Lempitsky, V. (2016). Domain-adversarial training of neural networks. *Journal of Machine Learning Research*, 17(59):1–35.

[139] Gao, J., Fan, W., Jiang, J., and Han, J. (2008). Knowledge transfer via multiple model local structure mapping. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Las Vegas, Nevada, USA, August 24-27, 2008*, pages 283–291.

[140] Ge, D., Jiang, X., and Ye, Y. (2011). A note on the complexity of lp minimization. *Mathematical Programming*, 129(2):285–299.

[141] Geman, S., Bienenstock, E., and Doursat, R. (1992). Neural networks and the bias/variance dilemma. *Neural Comput.*, 4(1):1–58.

[142] Germain, P., Habrard, A., Laviolette, F., and Morvant, E. (2017). Pac-bayes and domain adaptation. *arXiv*.

[143] Ghosh, S., Alomari, R. S., Chaudhary, V., and Dhillon, G. (2011). Automatic lumbar vertebra segmentation from clinical CT for wedge compression fracture diagnosis. *Proceedings of the SPIE*, 3:796303–9.

[144] Girosi, F. and Poggio, T. (1989). Representation properties of networks: Kolmogorov's theorem is irrelevant. *Neural Computation*, 1(4):465–469.

[145] Glocker, B., D.Zikic, E.Konukoglu, Haynor, D., and Criminisi, A. (2013). Vertebrae localization in pathological spine CT via dense classification from sparse annotations. *MICCAI*, 16(Pt 2):262–70.

[146] Glocker, B., Feulner, J., Criminisi, A., Haynor, D. R., and Konukoglu, E. (2012). *Automatic Localization and Identification of Vertebrae in Arbitrary Field-of-View CT Scans*, pages 590–598. Springer Berlin Heidelberg, Berlin, Heidelberg.

[147] Glocker, B., Zikic, D., and Haynor, D. R. (2014). *Robust Registration of Longitudinal Spine CT*, pages 251–258. Springer International Publishing.

[148] Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *International conference on artificial intelligence and statistics*, pages 249–256.

[149] Glorot, X., Bordes, A., and Bengio, Y. (2011a). Deep sparse rectifier neural networks. In Gordon, G. J. and Dunson, D. B., editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS-11)*, volume 15, pages 315–323. Journal of Machine Learning Research - Workshop and Conference Proceedings.

[150] Glorot, X., Bordes, A., and Bengio, Y. (2011b). Domain adaptation for large-scale sentiment classification: A deep learning approach. In *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, pages 513–520.

[151] Goertzel, B. (2015). Are there deep reasons underlying the pathologies of today's deep learning algorithms? In Bieger, J., Goertzel, B., and Potapov, A., editors, *Artificial General Intelligence*, pages 70–79, Cham. Springer International Publishing.

[152] Golodetz, S., Voiculescu, I., and Cameron, S. (2009). Automatic spine identification in abdominal CT slices using image partition forests. *International Symposium on Image and Signal Processing and Analysis*.

[153] Gomez, F. J. and Schmidhuber, J. (2005). Co-evolving recurrent neurons learn deep memory pomdps. In *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation*, GECCO '05, pages 491–498, New York, NY, USA. ACM.

[154] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[155] Goodfellow, I., Bulatov, Y., Ibarz, J., Arnoud, S., and Shet, V. (2014a). Multi-digit number recognition from street view imagery using deep convolutional neural networks. In *International Conference on Learning Representations (ICLR2014)*.

[156] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014b). Generative adversarial nets. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc.

[157] Goodfellow, I. J., Shlens, J., and Szegedy, C. (2014c). Explaining and harnessing adversarial examples. *CoRR*, abs/1412.6572.

[158] Goodfellow, I. J., Warde-Farley, D., Mirza, M., Courville, A. C., and Bengio, Y. (2013). Maxout networks. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, pages 1319–1327.

[159] Gordon, D. F. and Desjardins, M. (1995). Evaluation and selection of biases in machine learning. *Machine Learning*, 20(1):5–22.

[160] Gouérant, S., Leheurteur, M., Chaker, M., Modzelewski, R., Rigal, O., Veyret, C., Lauridant, G., and Clatot, F. (2013). A higher body mass index and fat mass are factors predictive of docetaxel dose intensity. *Anticancer research*, 33(12):5655.

[161] Graves, A. (2012). *Supervised Sequence Labelling with Recurrent Neural Networks*, volume 385 of *Studies in Computational Intelligence*. Springer.

[162] Graves, A. (2013a). Generating sequences with recurrent neural networks. *CoRR*, abs/1308.0850.

[163] Graves, A. (2013b). Generating sequences with recurrent neural networks. *CoRR*, abs/1308.0850.

[164] Graves, A. and Jaitly, N. (2014a). Towards end-to-end speech recognition with recurrent neural networks. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, pages 1764–1772.

[165] Graves, A. and Jaitly, N. (2014b). Towards end-to-end speech recognition with recurrent neural networks. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, pages II–1764–II–1772. JMLR.org.

[166] Graves, A., Liwicki, M., Fernández, S., Bertolami, R., Bunke, H., and Schmidhuber, J. (2009). A novel connectionist system for unconstrained handwriting recognition. *IEEE transactions on pattern analysis and machine intelligence*, 31(5):855–868.

[167] Graves, A., Mohamed, A., and Hinton, G. E. (2013). Speech recognition with deep recurrent neural networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2013, Vancouver, BC, Canada, May 26-31, 2013*, pages 6645–6649.

[168] Graves, A. and Schmidhuber, J. (2009). Offline handwriting recognition with multidimensional recurrent neural networks. In Koller, D., Schuurmans, D., Bengio, Y., and Bottou, L., editors, *Advances in Neural Information Processing Systems 21*, pages 545–552. Curran Associates, Inc.

[169] Graves, A., Wayne, G., and Danihelka, I. (2014). Neural turing machines. *arXiv preprint arXiv:1410.5401*.

[170] Graves, A., Wayne, G., Reynolds, M., Harley, T., Danihelka, I., Grabska-Barwińska, A., Colmenarejo, S. G., Grefenstette, E., Ramalho, T., Agapiou, J., Badia, A. P., Hermann, K. M., Zwols, Y., Ostrovski, G., Cain, A., King, H., Summerfield, C., Blunsom, P., Kavukcuoglu, K., and Hassabis, D. (2016). Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476.

[171] Griewank, A. (2012). *Documenta Mathematica - Extra Volume ISMP*, pages 389–400.

[172] Grossberg, S. (1973). Contour enhancement, short term memory, and constancies in reverberating neural networks. *Studies in Applied Mathematics*, 52(3):213–257.

[173] Grossberg, S. (1982). *Contour Enhancement, Short Term Memory, and Constancies in Reverberating Neural Networks*, pages 332–378. Springer Netherlands, Dordrecht.

[174] Grünwald, P. D. (2007). *The Minimum Description Length Principle (Adaptive Computation and Machine Learning)*. The MIT Press.

[175] Grünwald, P. (2005). A tutorial introduction to the minimum description length principle. In *Advances in Minimum Description Length: Theory and Applications*. MIT Press.

[176] Gu, S. and Rigazio, L. (2014). Towards deep neural network architectures robust to adversarial examples. *CoRR*, abs/1412.5068.

[177] Hadamard, J. (1908). *Mémoire sur le problème d'analyse relatif à l'équilibre des plaques élastiques encastrées*, volume 33. Imprimerie nationale.

[178] Hadsell, R., Chopra, S., and LeCun, Y. (2006). Dimensionality reduction by learning an invariant mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2006), 17-22 June 2006, New York, NY, USA*, pages 1735–1742.

[179] Hafemann, L. G., Sabourin, R., and Oliveira, L. S. (2016). Writer-independent feature learning for offline signature verification using deep convolutional neural networks. *CoRR*, abs/1604.00974.

[180] Hammer, B. (1998). On the approximation capability of recurrent neural networks. In *In International Symposium on Neural Computation*, pages 12–4.

[181] Hansel, D., Mato, G., and Meunier, C. (1992). Memorization without generalization in a multilayered neural network. *EPL (Europhysics Letters)*, 20(5):471.

[182] Hashimoto, K., Xiong, C., Tsuruoka, Y., and Socher, R. (2017). A joint many-task model: Growing a neural network for multiple NLP tasks. In *EMNLP*, pages 1923–1933. Association for Computational Linguistics.

[183] Hassoun, M. H. (1995). *Fundamentals of Artificial Neural Networks*. MIT Press, Cambridge, MA, USA, 1st edition.

[184] Hastie, T., Tibshirani, R., and Wainwright, M. (2015). *Statistical Learning with Sparsity: The Lasso and Generalizations*. Chapman & Hall/CRC.

[185] Havaei, M., Davy, A., Warde-Farley, D., Biard, A., Courville, A., Bengio, Y., Pal, C., Jodoin, P., and Larochelle, H. (2015). Brain tumor segmentation with deep neural networks. *CoRR*, abs/1505.03540.

[186] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV 2015*, pages 1026–1034.

[187] He, K., Zhang, X., Ren, S., and Sun, J. (2016a). Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778.

[188] He, K., Zhang, X., Ren, S., and Sun, J. (2016b). Identity mappings in deep residual networks. In *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part IV*, pages 630–645.

[189] Hebb, D. O. (1949). *The organization of behavior: A neuropsychological theory*. Wiley, New York.

[190] Hecht-Nielsen, R. (1989a). *Neurocomputing*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[191] Hecht-Nielsen, R. (1989b). Theory of the backpropagation neural network. In *International Joint Conference on Neural Networks (IJCNN)*, pages 593–605. IEEE.

[192] Heckerman, D., Geiger, D., and Chickering, D. M. (1995). Learning bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20(3):197–243.

[193] Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. *Neural Comput.*, 14(8):1771–1800.

[194] Hinton, G. E. (2007a). Learning multiple layers of representation. *Trends in Cognitive Sciences*, 11:428–434.

[195] Hinton, G. E. (2007b). Learning multiple layers of representation. *Trends in Cognitive Sciences*, 11:428–434.

[196] Hinton, G. E. (2012). A practical guide to training restricted boltzmann machines. In Montavon, G., Orr, G. B., and Müller, K.-R., editors, *Neural Networks: Tricks of the Trade (2nd ed.)*, volume 7700 of *Lecture Notes in Computer Science*, pages 599–619. Springer.

[197] Hinton, G. E., McClelland, J. L., and Rumelhart, D. E. (1986). Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. chapter Distributed Representations, pages 77–109. MIT Press, Cambridge, MA, USA.

[198] Hinton, G. E., Osindero, S., and Teh, Y.-W. (2006a). A fast learning algorithm for deep belief nets. *Neural Comput.*, 18(7):1527–1554.

[199] Hinton, G. E., Osindero, S., and Teh, Y. W. (2006b). A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554.

[200] Hinton, G. E., Sabour, S., and Frosst, N. (2018). Matrix capsules with EM routing. In *International Conference on Learning Representations*.

[201] Hinton, G. E. and Salakhutdinov, R. (2006). Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507.

[202] Hinton, G. E., Sejnowski, T. J., and Ackley, D. H. (1984). Boltzmann machines: Constraint satisfaction networks that learn. Technical Report CMU-CS-84-119, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA.

[203] Ho, T. K. (1995). Random decision forests. In *Proceedings of the Third International Conference on Document Analysis and Recognition (Volume 1) - Volume 1*, ICDAR '95, pages 278–, Washington, DC, USA. IEEE Computer Society.

[204] Hochreiter, S. (1991). Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, Institut für Informatik, Lehrstuhl Prof. Brauer, Technische Universität München. Advisor: J. Schmidhuber.

[205] Hochreiter, S., Bengio, Y., Frasconi, P., and Schmidhuber, J. (2001). Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In Kremer and Kolen, editors, *A Field Guide to Dynamical Recurrent Neural Networks*. IEEE Press.

[206] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Comput.*, 9(8):1735–1780.

[207] Hoffman, J., Wang, D., Yu, F., and Darrell, T. (2016). Fcns in the wild: Pixel-level adversarial and constraint-based adaptation. *CoRR*, abs/1612.02649.

[208] Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366.

[209] Hornik, K., Stinchcombe, M., and White, H. (1990). Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. *Neural Netw.*, 3(5):551–560.

[210] Hosu, I. and Rebedea, T. (2016). Playing atari games with deep reinforcement learning and human checkpoint replay. *CoRR*, abs/1607.05077.

[211] Huang, G. B. and Jain, V. (2013). Deep and wide multiscale recursive networks for robust image labeling. *CoRR*, abs/1310.0354.

[212] Huang, S. H., Chu, Y. H., Lai, S. H., and Novak, C. L. (2009). Learning-Based Vertebra Detection and Iterative Normalized-Cut Segmentation for Spinal MRI. *IEEE Transactions on Medical Imaging*, 28(10):1595–1605.

[213] Hubel, D. H. and Wiesel, T. (1962). Receptive fields, binocular interaction, and functional architecture in the cat's visual cortex. *Journal of Physiology (London)*, 160:106–154.

[214] III, H. D. (2007). Frustratingly easy domain adaptation. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 256–263, Prague, Czech Republic. Association for Computational Linguistics.

[215] (III.), W. C. R. (1962). *An Adaptive Logic System with Generalizing Properties*. PhD thesis, Stanford University, Stanford Electronics Labs.

[216] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 448–456.

[217] Irie, B. and Miyake, S. (1988). Capabilities of three-layered perceptrons. In *IEEE International Conference on Neural Networks*, volume 1, page 218.

[218] Ivakhnenko, A. G. (1971). Polynomial theory of complex systems. *IEEE Transactions on Systems, Man and Cybernetics*, (4):364–378.

[219] Ivakhnenko, A. G. and Lapa, V. G. (1965). *Cybernetic Predicting Devices*. CCM Information Corporation.

[220] Ivakhnenko, A. G., Lapa, V. G., and McDonough, R. N. (1967). *Cybernetics and forecasting techniques*. American Elsevier, NY.

[221] Jaderberg, M., Simonyan, K., Vedaldi, A., and Zisserman, A. (2014). Deep structured output learning for unconstrained text recognition. *CoRR*, abs/1412.5903.

[222] James, G., Witten, D., Hastie, T., and Tibshirani, R. (2014). *An Introduction to Statistical Learning: With Applications in R*. Springer Publishing Company, Incorporated.

[223] Jarrett, K., Kavukcuoglu, K., Ranzato, M., and LeCun, Y. (2009). What is the best multi-stage architecture for object recognition? In *ICCV 2009*, pages 2146–2153.

[224] Jiang, J. and Zhai, C. (2007). Instance weighting for domain adaptation in NLP. In *ACL 2007, Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics, June 23-30, 2007, Prague, Czech Republic*.

[225] Jiang, X. (2015). Representational transfer in deep belief networks. In *28th Canadian Conference on Artificial Intelligence*, pages 338–342.

[226] Jones, D. T. (1999). Protein secondary structure prediction based on position-specific scoring matrices. *Journal of Molecular Biology*, 292(2):195–202.

[227] Joulin, A. and Mikolov, T. (2015). Inferring algorithmic patterns with stack-augmented recurrent nets. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 190–198.

[228] Józefowicz, R., Vinyals, O., Schuster, M., Shazeer, N., and Wu, Y. (2016). Exploring the limits of language modeling. *CoRR*, abs/1602.02410.

[229] Kadoury, S., Labelle, H., and Paragios, N. (2011). Automatic inference of articulated spine models in CT images using high-order markov random fields. *Medical Image Analysis*, 15(4):426–437.

[230] Kaido, T., Ogawa, K., Fujimoto, Y., Ogura, Y., Hata, K., Ito, T., Tomiyama, K., Yagi, S., Mori, A., and Uemoto, S. (2013). Impact of sarcopenia on survival in patients undergoing living donor liver transplantation. *American Journal of Transplantation*, 13(6):1549–1556.

[231] Karpathy, A. and Li, F. (2015). Deep visual-semantic alignments for generating image descriptions. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 3128–3137.

[232] Kearns, M. J. and Vazirani, U. V. (1994). *An Introduction to Computational Learning Theory*. MIT Press, Cambridge, MA, USA.

[233] Kelley, H. J. (1960). Gradient theory of optimal flight paths. *Ars Journal*, 30(10):947–954.

[234] Kendall, A., Gal, Y., and Cipolla, R. (2017). Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. *CoRR*, abs/1705.07115.

[235] Kim, Y. (2014a). Convolutional neural networks for sentence classification. *CoRR*, abs/1408.5882.

[236] Kim, Y. (2014b). Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1746–1751.

[237] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

[238] Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *CoRR*, abs/1312.6114.

[239] Kiros, R., Salakhutdinov, R., and Zemel, R. S. (2014). Unifying visual-semantic embeddings with multimodal neural language models. *CoRR*, abs/1411.2539.

[240] Kolda, T. G. and Bader, B. W. (2009). Tensor decompositions and applications. *SIAM Rev.*, 51(3):455–500.

[241] Kolmogorov, A. K. (1957). On the representation of continuous functions of several variables by superposition of continuous functions of one variable and addition. *Doklady Akademii Nauk SSSR*, 114:369–373.

[242] Kolmogorov, A. N. (1965). On the representation of continuous functions of several variables by superposition of continuous functions of one variable and addition. *Doklady Akademii. Nauk USSR,*, 114:679–681.

[243] Krizhevsky, A. (2009). Learning multiple layers of features from tiny images. Technical report.

[244] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In Pereira, F., Burges, C., Bottou, L., and Weinberger, K., editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc.

[245] Krueger, D., Ballas, N., Jastrzebski, S., Arpit, D., Kanwal, M. S., Maharaj, T., Bengio, E., Fischer, A., and Courville, A. (2017). Deep nets don't learn via memorization.

[246] Krupka, E. and Tishby, N. (2007). Incorporating prior knowledge on features into learning. In *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics, AISTATS 2007, San Juan, Puerto Rico, March 21-24, 2007*, pages 227–234.

[247] Kumar, A., Irsoy, O., Ondruska, P., Iyyer, M., Bradbury, J., Gulrajani, I., Zhong, V., Paulus, R., and Socher, R. (2016). Ask me anything: Dynamic memory networks for natural language processing. In Balcan, M. F. and Weinberger, K. Q., editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1378–1387, New York, New York, USA. PMLR.

[248] Lafferty, J. D., McCallum, A., and Pereira, F. C. N. (2001). Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In *ICML*, pages 282–289.

[249] Lai, M. (2015). Deep learning for medical image segmentation. *CoRR*, abs/1505.02000.

[250] Lanic, H., Kraut-Tauzia, J., Modzelewski, R., Clatot, F., Mareschal, S., Picquenot, J. M., Stamatoullas, A., Leprêtre, S., Tilly, H., and Jardin, F. (2014). Sarcopenia is an independent prognostic factor in elderly patients with diffuse large b-cell lymphoma treated with immunochemotherapy. *Leukemia & Lymphoma*, 55(4):817–823.

[251] Lawrence, N. D. and Platt, J. C. (2004). Learning to learn with the informative vector machine. In *Machine Learning, Proceedings of the Twenty-first International Conference (ICML 2004), Banff, Alberta, Canada, July 4-8, 2004*.

[252] Le, V., Brandt, J., Lin, Z., Bourdev, L. D., and Huang, T. S. (2012). Interactive Facial Feature Localization. In *ECCV, 2012, Proceedings, Part III*, pages 679–692.

[253] LeCun, Y. (1985). Une procédure d'apprentissage pour réseau à seuil asymétrique. *Proceedings of Cognitiva 85, Paris*, pages 599–604.

[254] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989a). Back-propagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551.

[255] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989b). Backpropagation applied to handwritten zip code recognition. *Neural Comput.*, 1(4):541–551.

[256] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1990). Handwritten digit recognition with a back-propagation network. In Touretzky, D. S., editor, *Advances in Neural Information Processing Systems 2*, pages 396–404. Morgan Kaufmann.

[257] Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324.

[258] LeCun, Y., Bottou, L., Orr, G. B., and Müller, K.-R. (1998). Efficient backprop. In *Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop*, pages 9–50, London, UK, UK. Springer-Verlag.

[259] Lee, H., Grosse, R., Ranganath, R., and Ng, A. Y. (2009). Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 609–616, New York, NY, USA. ACM.

[260] Leibniz, G. W. (1676). Memoir using the chain rule (cited in TMME 7:2&3 p 321-332, 2010).

[261] Lerouge, J., Herault, R., Chatelain, C., Jardin, F., and Modzelewski, R. (2015). IODA : An input / output deep architecture for image labeling. *Pattern Recognition*, 48(9):2847–2858.

[262] Leshno, M., Lin, V. Y., Pinkus, A., and Schocken, S. (1993). Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, 6(6):861–867.

[263] L'Hôpital, G. F. A. (1696). *Analyse des infiniment petits, pour l'intelligence des lignes courbes.* Paris: L'Imprimerie Royale.

[264] Li, H., Kadav, A., Durdanovic, I., Samet, H., and Graf, H. P. (2016a). Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*.

[265] Li, X., Uricchio, T., Ballan, L., Bertini, M., Snoek, C. G. M., and Bimbo, A. D. (2016b). Socializing the semantic gap: A comparative survey on image tag assignment, refinement, and retrieval. *ACM Comput. Surv.*, 49(1):14:1–14:39.

[266] Light, W. (1992). Ridge functions, sigmoidal functions and neural networks. *Approximation theory VII*, pages 163–206.

[267] Linnainmaa, S. (1970). The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors. Master's thesis, Univ. Helsinki.

[268] Linnainmaa, S. (1976). Taylor expansion of the accumulated rounding error. *BIT Numerical Mathematics*, 16(2):146–160.

[269] Lippmann, R. P. (1988). An introduction to computing with neural nets. *SIGARCH Computer Architecture News*, 16(1):7–25.

[270] Liu, D. C. and Nocedal, J. (1989). On the limited memory bfgs method for large scale optimization. *Math. Program.*, 45(3):503–528.

[271] Liu, S., Yang, N., Li, M., and Zhou, M. (2014). A recursive recurrent neural network for statistical machine translation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1491–1500, Baltimore, Maryland. Association for Computational Linguistics.

[272] Long, J., Shelhamer, E., and Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 3431–3440.

[273] Long, M. and Wang, J. (2015). Learning multiple tasks with deep relationship networks. *CoRR*, abs/1506.02117.

[274] Lu, Y., Kumar, A., Zhai, S., Cheng, Y., Javidi, T., and Feris, R. S. (2017). Fully-adaptive feature sharing in multi-task networks with applications in person attribute classification. In *CVPR*, pages 1131–1140. IEEE Computer Society.

[275] Luenberger, D. G. (1969). *Optimization by vector space methods*. Decision and control. Wiley, New York, NY.

[276] Ma, J. and Lu, L. (2013). Hierarchical segmentation and identification of thoracic vertebra using learning-based edge detection and coarse-to-fine deformable model. *Computer Vision and Image Understanding*, 117(9):1072–1083.

[277] Maas, A. L., Hannun, A. Y., and Ng, A. Y. (2013). Rectifier nonlinearities improve neural network acoustic models. In *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*.

[278] Mahmud, M. M. and Ray, S. R. (2007). Transfer learning using kolmogorov complexity: Basic theory and empirical evaluations. In *Advances in Neural Information Processing Systems 20, Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 3-6, 2007*, pages 985–992.

[279] Major, D., Hladůvka, J., Schulze, F., and Bühler, K. (2013). Automated landmarking and labeling of fully and partially scanned spinal columns in CT images. *Medical Image Analysis*, 17(8):1151–1163.

[280] Makhzani, A., Shlens, J., Jaitly, N., and Goodfellow, I. J. (2015). Adversarial autoencoders. *CoRR*, abs/1511.05644.

[281] Malon, C., Miller, M., Burger, H. C., Cosatto, E., and Graf, H. P. (2008). Identifying histological elements with convolutional neural networks. In *Int. Conf. on Soft Computing As Transdisciplinary Science and Technology*, pages 450–456.

[282] Marcus, G. (2018). Deep learning: A critical appraisal. *CoRR*, abs/1801.00631.

[283] Marcus, G. F. (2003). *The algebraic mind: Integrating connectionism and cognitive science*. MIT press.

[284] Martens, J. (2010). Deep learning via hessian-free optimization. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, pages 735–742.

[285] Martens, J. and Sutskever, I. (2011). Learning recurrent neural networks with Hessian-free optimization. In *ICML 2011*, pages 1033–1040.

[286] Martin, L., Birdsell, L., MacDonald, N., Reiman, T., Clandinin, M. T., McCargar, L. J., Murphy, R., Ghosh, S., Sawyer, M. B., and Baracos, V. E. (2013). Cancer cachexia in the age of obesity: Skeletal muscle depletion is a powerful prognostic factor, independent of body mass index. *Journal of Clinical Oncology*, 31(12):1539–1547.

[287] Masci, J., Meier, U., Cireşan, D., and Schmidhuber, J. (2011). *Stacked Convolutional Auto-Encoders for Hierarchical Feature Extraction*, pages 52–59. Springer Berlin Heidelberg, Berlin, Heidelberg.

[288] McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133.

[289] McInerney, T. and Terzopoulos, D. (1996). Deformable models in medical image analysis: a survey. *Medical image analysis*, 1(2):91–108.

[290] Michael Kelm, B., Wels, M., Kevin Zhou, S., Seifert, S., Suehling, M., Zheng, Y., and Comaniciu, D. (2013). Spine detection in CT and MR using iterated marginal space learning. *Medical Image Analysis*, 17(8):1283–1292.

[291] Mihalkova, L., Huynh, T. N., and Mooney, R. J. (2007). Mapping and revising markov logic networks for transfer learning. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, July 22-26, 2007, Vancouver, British Columbia, Canada*, pages 608–614.

[292] Mihalkova, L. and Mooney, R. (2006). Transfer learning with markov logic networks. In *ICML workshop on structural knowledge transfer for machine learning*.

[293] Mihalkova, L. and Mooney, R. J. (2008). Transfer learning by mapping with minimal target data. In *Proceedings of the AAAI-08 workshop on transfer learning for complex tasks*.

[294] Mikolov, T. (2012). *Statistical language models based on neural networks*. PhD thesis, Brno University of Technology.

[295] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *NIPS*, pages 3111–3119.

[296] Minsky, M. and Papert, S. (1969). *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, USA.

[297] Minsky, M. L. and Papert, S. A. (1988). *Perceptrons: Expanded Edition*. MIT Press, Cambridge, MA, USA.

[298] Misra, I., Shrivastava, A., Gupta, A., and Hebert, M. (2016). Cross-stitch networks for multi-task learning. In *CVPR*, pages 3994–4003. IEEE Computer Society.

[299] Mitchell, T. M. (1980). The need for biases in learning generalizations. Technical report, Rutgers University, New Brunswick, NJ.

[300] Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition.

[301] Mitsiopoulos, N., Baumgartner, R. N., Heymsfield, S. B., Lyons, W., Gallagher, D., and Ross, R. (1998). Cadaver validation of skeletal muscle measurement by magnetic resonance imaging and computerized tomography. *Journal of applied physiology*, 85(1):115–122.

[302] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. In *NIPS Deep Learning Workshop*.

[303] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M. A., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

[304] Mnih, V., Larochelle, H., and Hinton, G. E. (2011). Conditional restricted boltzmann machines for structured output prediction. In *UAI 2011, Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence, Barcelona, Spain, July 14-17, 2011*, pages 514–522.

[305] Mohri, M., Rostamizadeh, A., and Talwalkar, A. (2012). *Foundations of Machine Learning*. The MIT Press.

[306] Molchanov, P., Tyree, S., Karras, T., Aila, T., and Kautz, J. (2016). Pruning convolutional neural networks for resource efficient transfer learning. *arXiv preprint arXiv:1611.06440*.

[307] Moller, M. F. (1993). Exact calculation of the product of the Hessian matrix of feed-forward network error functions and a vector in O(N) time. Technical Report PB-432, Computer Science Department, Aarhus University, Denmark.

[308] Montúfar, G., Pascanu, R., Cho, K., and Bengio, Y. (2014). On the number of linear regions of deep neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, pages 2924–2932, Cambridge, MA, USA. MIT Press.

[309] Mourad, N. and Reilly, J. P. (2010). Minimizing nonconvex functions for sparse vector reconstruction. *IEEE Trans. Signal Processing*, 58(7):3485–3496.

[310] Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In Fürnkranz, J. and Joachims, T., editors, *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814. Omnipress.

[311] Nanni, L., Lumini, A., and Brahnam, S. (2010). Local binary patterns variants as texture descriptors for medical image analysis. *Artificial Intelligence in Medicine*, 49(2):117 – 125.

[312] Natarajan, B. K. (1995). Sparse approximate solutions to linear systems. *SIAM J. Comput.*, 24(2):227–234.

[313] Nesterov, Y. (1983). A method of solving a convex programming problem with convergence rate O(1/sqr(k)). *Soviet Mathematics Doklady*, 27:372–376.

[314] Nguyen, A. M., Yosinski, J., and Clune, J. (2015). Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *CVPR*, pages 427–436. IEEE Computer Society.

[315] Nguyen, D. T., Alam, F., Ofli, F., and Imran, M. (2017). Automatic image filtering on social networks using deep learning and perceptual hashing during crises. *CoRR*, abs/1704.02602.

[316] Nicolas, S., Paquet, T., and Heutte, L. (2006). A Markovian Approach for Handwritten Document Segmentation. In *ICPR (3)*, pages 292–295.

[317] Nielsen, R. H. (1987). Kolmogorov's mapping neural network existence theorem. In *Proceedings of the IEEE First International Conference on Neural Networks* (San Diego, CA), volume III, pages 11–13. Piscataway, NJ: IEEE.

[318] Ning, F., Delhomme, D., LeCun, Y., Piano, F., Bottou, L., and Barbano, P. E. (2005). Toward automatic phenotyping of developing embryos from videos. *IEEE Trans. Image Processing*, 14(9):1360–1371.

[319] Niyogi, P., Girosi, F., and Poggio, T. (1998). Incorporating prior information in machine learning by creating virtual examples. *Proceedings of the IEEE*, 86(11):2196–2209.

[320] Noh, H., Hong, S., and Han, B. (2015). Learning deconvolution network for semantic segmentation. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 1520–1528.

[321] Noto, K. and Craven, M. (2012). Learning Hidden Markov Models for Regression using Path Aggregation. *CoRR*, abs/1206.3275.

[322] Novikoff, A. B. (1962). On convergence proofs on perceptrons. In *Proceedings of the Symposium on the Mathematical Theory of Automata*.

[323] Och, F. J. (2003). Minimum error rate training in statistical machine translation. In *Proceedings of the ACL*, volume 1.

[324] Ojala, T., Pietikainen, M., and Maenpaa, T. (2002). Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):971–987.

[325] Oktay, A. B. and Akgul, Y. S. (2011). Localization of the lumbar discs using machine learning and exact probabilistic inference. In *Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics*, pages 158–165.

[326] Olazaran, M. (1996). A sociological study of the official history of the perceptrons controversy. *Social Studies of Science*, 26(3):611–659.

[327] Olivas, E. S., Guerrero, J. D. M., Sober, M. M., Benedito, J. R. M., and Lopez, A. J. S. (2009). *Handbook Of Research On Machine Learning Applications and Trends: Algorithms, Methods and Techniques - 2 Volumes*. Information Science Reference - Imprint of: IGI Publishing, Hershey, PA.

[328] Oster, M., Douglas, R. J., and Liu, S. (2009). Computation with spikes in a winner-take-all network. *Neural Computation*, 21(9):2437–2465.

[329] Pan, S. J. and Yang, Q. (2010). A survey on transfer learning. *IEEE Trans. on Knowl. and Data Eng.*, 22(10):1345–1359.

[330] Pandey, G. and Dukkipati, A. (2014). To go deep or wide in learning? In Kaski, S. and Corander, J., editors, *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*, volume 33 of *Proceedings of Machine Learning Research*, pages 724–732, Reykjavik, Iceland. PMLR.

[331] Papernot, N., Abadi, M., Erlingsson, Ú., Goodfellow, I. J., and Talwar, K. (2016). Semi-supervised knowledge transfer for deep learning from private training data. *CoRR*, abs/1610.05755.

[332] Paredes, B. R., Argyriou, A., Berthouze, N., and Pontil, M. (2012). Exploiting unrelated tasks in multi-task learning. In Lawrence, N. D. and Girolami, M., editors, *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics*, volume 22 of *Proceedings of Machine Learning Research*, pages 951–959, La Palma, Canary Islands. PMLR.

[333] Park, J., Li, S., Wen, W., Tang, P. T. P., Li, H., Chen, Y., and Dubey, P. (2016). Faster cnns with direct sparse convolutions and guided pruning.

[334] Parker, D. B. (1985). Learning-logic. Technical Report TR-47, Center for Comp. Research in Economics and Management Sci., MIT.

[335] Pascanu, R., Mikolov, T., and Bengio, Y. (2012). Understanding the exploding gradient problem. *CoRR*, abs/1211.5063.

[336] Pascanu, R., Mikolov, T., and Bengio, Y. (2013a). On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, pages 1310–1318.

[337] Pascanu, R., Mikolov, T., and Bengio, Y. (2013b). On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, pages III–1310–III–1318. JMLR.org.

[338] Pearlmutter, B. A. (1994). Fast exact multiplication by the Hessian. *Neural Computation*, 6(1):147–160.

[339] Peng, P., Van Vledder, M., Tsai, S., De Jong, M., Makary, M., Ng, J., Edil, B., Wolfgang, C., Schulick, R., Choti, M., Kamel, I., and Pawlik, T. (2011). Sarcopenia negatively impacts short-term outcomes in patients undergoing hepatic resection for colorectal liver metastasis. *HPB*, 13(7):439–446.

[340] Peyré, G., Cuturi, M., et al. (2017). Computational optimal transport. Technical report.

[341] Pham, D. L., Xu, C., and Prince, J. L. (2000). Current methods in medical image segmentation 1. *Annual review of biomedical engineering*, 2(1):315–337.

[342] Polyak, B. T. (1964). Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17.

[343] Poole, B., Sohl-Dickstein, J., and Ganguli, S. (2014). Analyzing noise in autoencoders and deep networks. *CoRR*, abs/1406.1831.

[344] Povey, D., Zhang, X., and Khudanpur, S. (2014). Parallel training of deep neural networks with natural gradient and parameter averaging. *CoRR*, abs/1410.7455.

[345] Rabanser, S., Shchur, O., and Günnemann, S. (2017). Introduction to tensor decompositions and their applications in machine learning. *CoRR*, abs/1711.10781.

[346] Rabiner, L. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286.

[347] Radford, A., Metz, L., and Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434.

[348] Raiko, T., Valpola, H., and Lecun, Y. (2012). Deep learning made easier by linear transformations in perceptrons. In Lawrence, N. D. and Girolami, M., editors, *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics*, volume 22 of *Proceedings of Machine Learning Research*, pages 924–932, La Palma, Canary Islands. PMLR.

[349] Raina, R., Battle, A., Lee, H., Packer, B., and Ng, A. Y. (2007). Self-taught learning: transfer learning from unlabeled data. In *Machine Learning, Proceedings of the Twenty-Fourth International Conference (ICML 2007), Corvallis, Oregon, USA, June 20-24, 2007*, pages 759–766.

[350] Raina, R., Madhavan, A., and Ng, A. Y. (2009). Large-scale deep unsupervised learning using graphics processors. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 873–880, New York, NY, USA. ACM.

[351] Ramachandran, P., Liu, P. J., and Le, Q. V. (2017). Unsupervised pretraining for sequence to sequence learning. In *EMNLP*, pages 383–391. Association for Computational Linguistics.

[352] Ramsundar, B., Kearnes, S. M., Riley, P., Webster, D., Konerding, D. E., and Pande, V. S. (2015). Massively multitask networks for drug discovery. *CoRR*, abs/1502.02072.

[353] Ranzato, A., Poultney, C., Chopra, S., and Lecun, Y. (2007a). Efficient Learning of Sparse Representations with an Energy-Based Model. In *NIPS*, pages 1137–1144.

[354] Ranzato, M., Boureau, Y., and LeCun, Y. (2007b). Sparse feature learning for deep belief networks. In *Advances in Neural Information Processing Systems 20, Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 3-6, 2007*, pages 1185–1192.

[355] Ranzato, M., Huang, F., Boureau, Y., and LeCun, Y. (2007c). Unsupervised learning of invariant feature hierarchies with applications to object recognition. In *Proc. Computer Vision and Pattern Recognition Conference (CVPR'07)*. IEEE Press.

[356] Ranzato, M., Poultney, C. S., Chopra, S., and LeCun, Y. (2006). Efficient learning of sparse representations with an energy-based model. In *Advances in Neural Information Processing Systems 19, Proceedings of the Twentieth Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 4-7, 2006*, pages 1137–1144.

[357] Razavian, A. S., Azizpour, H., Sullivan, J., and Carlsson, S. (2014). CNN features off-the-shelf: An astounding baseline for recognition. In *CVPR Workshops*, pages 512–519. IEEE Computer Society.

[358] Ren, S., He, K., Girshick, R., and Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. *NIPS 28*, pages 91–99.

[359] Riesenhuber, M. and Poggio, T. (1999). Hierarchical models of object recognition in cortex. *Nature Neuroscience*, 2(11).

[360] Rifai, S., Mesnil, G., Vincent, P., Muller, X., Bengio, Y., Dauphin, Y., and Glorot, X. (2011a). Higher order contractive auto-encoder. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*.

[361] Rifai, S., Vincent, P., Muller, X., Glorot, X., and Bengio, Y. (2011b). Contractive auto-encoders: Explicit invariance during feature extraction. In *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, pages 833–840.

[362] Rissanen, J. (1978). Modeling by shortest data description. *Automatica*, 14(5):465–471.

[363] Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention - MICCAI 2015 - 18th International Conference Munich, Germany, October 5 - 9, 2015, Proceedings, Part III*, pages 234–241.

[364] Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386.

[365] Rosenblatt, F. (1962). *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan Books, Washington.

[366] Roth, H. R., Yao, J., Lu, L., Stieger, J., Burns, J. E., and Summers, R. M. (2014). Detection of sclerotic spine metastases via random aggregation of deep convolutional neural network classifications. *CoRR*, abs/1407.5976.

[367] Ruder, S., Bingel, J., Augenstein, I., and Søgaard, A. (2017). Sluice networks: Learning what to share between loosely related tasks. *CoRR*, abs/1705.08142.

[368] Rudin, W. (1964). *Principles of mathematical analysis*. McGraw-hill New York.

[369] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error propagation. In Rumelhart, D. E. and McClelland, J. L., editors, *Parallel Distributed Processing*, volume 1, pages 318–362. MIT Press.

[370] Sabour, S., Frosst, N., and Hinton, G. E. (2017). Dynamic routing between capsules. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 3859–3869.

[371] Sagonas, C., Tzimiropoulos, G., Zafeiriou, S., and Pantic, M. (2013). A semi-automatic methodology for facial landmark annotation. In *CVPR Workshops*, pages 896–903.

[372] Salakhutdinov, R. and Hinton, G. (2009). Deep Boltzmann machines. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, volume 5, pages 448–455.

[373] Salimans, T., Kingma, D. P., and Welling, M. (2015). Markov chain monte carlo and variational inference: Bridging the gap. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 1218–1226.

[374] Sathyanarayana, A., Joty, S., Fernandez-Luque, L., Ofli, F., Srivastava, J., Elmagarmid, A., Arora, T., and Taheri, S. (2016). Sleep quality prediction from wearable data using deep learning. *JMIR mHealth and uHealth*, 4(4).

[375] Savva, A. D., Economopoulos, T. L., and Matsopoulos, G. K. (2016). Geometry-based vs. intensity-based medical image registration: A comparative study on 3D CT data. *Computers in Biology and Medicine*, 69:120–133.

[376] Schmid, H. (1994). Part-of-speech tagging with neural networks. *conference on Computational linguistics*, 12:44–49.

[377] Schmidhuber, J. (1989). A local learning algorithm for dynamic feedforward and recurrent networks. *Connection Science*, 1(4):403–412.

[378] Schmidhuber, J. (1992). Learning complex, extended sequences using the principle of history compression. *Neural Computation*, 4(2):234–242.

[379] Schmidhuber, J. (2013). My first Deep Learning system of 1991 + Deep Learning timeline 1962-2013. Technical Report arXiv:1312.5548v1 [cs.NE], The Swiss AI Lab IDSIA.

[380] Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117. Published online 2014; based on TR arXiv:1404.7828 [cs.NE].

[381] Scholkopf, B. and Smola, A. J. (2001). *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, USA.

[382] Schuster, M. (1999). *On supervised learning from sequential data with applications for speech recognition*. PhD thesis, Daktaro disertacija, Nara Institute of Science and Technology.

[383] Schuster, M. and Paliwal, K. (1997). Bidirectional recurrent neural networks. *Trans. Sig. Proc.*, 45(11):2673–2681.

[384] Sermanet, P., Kavukcuoglu, K., Chintala, S., and Lecun, Y. (2013). Pedestrian detection with unsupervised multi-stage feature learning. In *Proceedings of the 2013 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '13, pages 3626–3633, Washington, DC, USA. IEEE Computer Society.

[385] Shalev-Shwartz, S. and Ben-David, S. (2014). *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, New York, NY, USA.

[386] Shen, W., Punyanitya, M., Wang, Z., Gallagher, D., St-Onge, M.-P., Albu, J., Heymsfield, S. B., and Heshka, S. (2004). Total body skeletal muscle and adipose tissue volumes: estimation from a single abdominal cross-sectional image. *Journal of applied physiology*, 97(6):2333–2338.

[387] Shen, Y., He, X., Gao, J., Deng, L., and Mesnil, G. (2014). A latent semantic model with convolutional-pooling structure for information retrieval. In *CIKM*, pages 101–110. ACM.

[388] Shimodaira, H. (2000). Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of Statistical Planning and Inference*, 90(2):227–244.

[389] Shin, H. C., Roth, H. R., Gao, M., Lu, L., Xu, Z., Nogues, I., Yao, J., Mollura, D., and Summers, R. M. (2016). Deep convolutional neural networks for computer-aided detection: Cnn architectures, dataset characteristics and transfer learning. *IEEE Transactions on Medical Imaging*, 35(5):1285–1298.

[390] Sietsma, J. and Dow, R. J. (1991). Creating artificial neural networks that generalize. *Neural Networks*, 4(1):67 – 79.

[391] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T. P., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489.

[392] Simard, P., Le Cun, Y., and Denker, J. (1993). Efficient Pattern Recognition Using a New Transformation Distance. In *Advances in Neural Information Processing Systems*, volume 5, pages 50–58.

[393] Simard, P., Victorri, B., Lecun, Y., and Denker, J. (1992). Tangent Prop — a formalism for specifying selected invariances in an adaptive network. In Moody, J. E., Hanson, S. J., and Lippmann, R. P., editors, *Advances in Neural Information Processing Systems 4*, pages 895–903, San Mateo, CA. Morgan Kaufmann.

[394] Simard, P. Y., Steinkraus, D., and Platt, J. C. (2003). Best practices for convolutional neural networks applied to visual document analysis. In *Proceedings of the Seventh International Conference on Document Analysis and Recognition - Volume 2*, ICDAR '03, pages 958–, Washington, DC, USA. IEEE Computer Society.

[395] Simonyan, K., Vedaldi, A., and Zisserman, A. (2013). Deep inside convolutional networks: Visualising image classification models and saliency maps. *CoRR*, abs/1312.6034.

[396] Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556.

[397] Sjoberg, J., Sjoeberg, J., Sjöberg, J., and Ljung, L. (1995). Overtraining, regularization and searching for a minimum, with application to neural networks. *International Journal of Control*, 62:1391–1407.

[398] Sleator, D. D. and Temperley, D. (1993). Parsing English with a link grammar. In *Proc. Third International Workshop on Parsing Technologies*, pages 277–292.

[399] Smolensky, P. (1986). Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. chapter Information Processing in Dynamical Systems: Foundations of Harmony Theory, pages 194–281. MIT Press, Cambridge, MA, USA.

[400] Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A. Y., and Potts, C. (2013). Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Stroudsburg, PA. Association for Computational Linguistics.

[401] Søgaard, A. and Goldberg, Y. (2016). Deep multi-task learning with low level tasks supervised at lower layers. In *ACL (2)*. The Association for Computer Linguistics.

[402] Sohn, K., Lee, H., and Yan, X. (2015). Learning structured output representation using deep conditional generative models. In *NIPS 2015*, pages 3483–3491.

[403] Srivastava, N. (2013). Improving Neural Networks with Dropout. Master's thesis, University of Toronto, Toronto, Canada.

[404] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958.

[405] Srivastava, R. K., Greff, K., and Schmidhuber, J. (2016). Highway Networks.

[406] Starck, J.-L., Murtagh, F., and Fadili, J. (2015). *Dictionary Learning*, page 263–274. Cambridge University Press, 2 edition.

[407] Steffens, K.-G. (2007). *The history of approximation theory: from Euler to Bernstein*. Springer Science & Business Media.

[408] Steinkrau, D., Simard, P. Y., and Buck, I. (2005). Using gpus for machine learning algorithms. In *Proceedings of the Eighth International Conference on Document Analysis and Recognition*, ICDAR '05, pages 1115–1119, Washington, DC, USA. IEEE Computer Society.

[409] Stuner, B., Chatelain, C., and Paquet, T. (2016). Cohort of LSTM and lexicon verification for handwriting recognition with gigantic lexicon. *CoRR*, abs/1612.07528.

[410] Suddarth, S. C. and Kergosien, Y. L. (1990). Rule-injection hints as a means of improving network performance and learning time. In *Neural Networks, EURASIP workshop 1990*, pages 120–129.

[411] Sugiyama, M. (2007). Dimensionality reduction of multimodal labeled data by local fisher discriminant analysis. *J. Mach. Learn. Res.*, 8:1027–1061.

[412] Sukhbaatar, S., Szlam, A., Weston, J., and Fergus, R. (2015). Weakly supervised memory networks. *CoRR*, abs/1503.08895.

[413] Sun, X. and Cheney, E. W. (1992). The fundamentality of sets of ridge functions. *aequationes mathematicae*, 44(2):226–235.

[414] Sutskever, I., Martens, J., Dahl, G., and Hinton, G. (2013). On the importance of initialization and momentum in deep learning. In *ICML*, volume 28, pages 1139–1147.

[415] Sutskever, I., Vinyals, O., and Le, Q. V. (2014a). Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 3104–3112.

[416] Sutskever, I., Vinyals, O., and Le, Q. V. (2014b). Sequence to sequence learning with neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, pages 3104–3112, Cambridge, MA, USA. MIT Press.

[417] Syed, U. and Yona, G. (2009). Enzyme function prediction with interpretable models. *Computational Systems Biology. Humana press*, pages 373–420.

[418] Sze, V., Chen, Y. H., Yang, T. J., and Emer, J. S. (2017). Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329.

[419] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2014). Going Deeper with Convolutions. *CoRR*, abs/1409.4842.

[420] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S. E., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2014). Going deeper with convolutions. *CoRR*, abs/1409.4842.

[421] Szegedy, C., Toshev, A., and Erhan, D. (2013a). Deep neural networks for object detection. In Burges, C. J. C., Bottou, L., Welling, M., Ghahramani, Z., and Weinberger, K. Q., editors, *NIPS 26*, pages 2553–2561.

[422] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I. J., and Fergus, R. (2013b). Intriguing properties of neural networks. *CoRR*, abs/1312.6199.

[423] Szummer, M. and Qi, Y. (2004). Contextual Recognition of Hand-drawn Diagrams with Conditional Random Fields. In *IWFHR*, pages 32–37.

[424] Tang, Y. and Eliasmith, C. (2010). Deep networks for robust visual recognition. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, pages 1055–1062.

[425] Terlaky, T. (1985). On lp programming. *European Journal of Operational Research*, 22(1):70–100.

[426] Thrun, S. (1996). Is learning the n-th thing any easier than learning the first? In *Advances in Neural Information Processing Systems*, pages 640–646. The MIT Press.

[427] Thrun, S. and Mitchell, T. M. (1995). Learning one more thing. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95, Montréal Québec, Canada, August 20-25 1995, 2 Volumes*, pages 1217–1225.

[428] Tibshirani, R. (1994). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B*, 58:267–288.

[429] Tikhonov, A. N. (1963). On solving ill-posed problem and method of regularization. *Doklady Akademii Nauk USSR*, 153:501–504.

[430] Tikhonov, A. N. and Arsenin, V. Y. (1977). *Solutions of Ill-posed problems*. W.H. Winston.

[431] Tsechpenakis, G., Wang, J., Mayer, B., and Metaxas, D. (2007). Coupling CRFs and Deformable Models for 3D Medical Image Segmentation. In *ICCV*, pages 1–8.

[432] Tzeng, E., Hoffman, J., Darrell, T., and Saenko, K. (2017). Adversarial discriminative domain adaptation. In *CVPR*.

[433] Ulyanov, D., Vedaldi, A., and Lempitsky, V. (2017). Deep image prior. *arXiv:1711.10925*.

[434] Urban, G., Bendszus, M., Hamprecht, F. A., and Kleesiek, J. (2014). Multi-modal brain tumor segmentation using deep convolutional neural networks. In *MICCAI BraTS Challenge Proceedings*, pages 31–35.

[435] Utgoff, P. E. (1986). *Machine Learning of Inductive Bias*. Kluwer, B.V., Deventer, The Netherlands, The Netherlands.

[436] Valiant, L. G. (1984). A theory of the learnable. *Commun. ACM*, 27(11):1134–1142.

[437] van den Oord, A., Dieleman, S., and Schrauwen, B. (2013). Deep content-based music recommendation. In *NIPS*, pages 2643–2651.

[438] Vapnik, V. N. and Chervonenkis, A. Y. (1971). On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16(2):264–280.

[439] Vincent, P., Hugo, L., Bengio, Y., and Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, ICML '08, pages 1096–1103, New York, NY, USA. ACM.

[440] Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., and Manzagol, P. (2010). Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion. *JMLR*, 11:3371–3408.

[441] Vinyals, O., Kaiser, L., Koo, T., Petrov, S., Sutskever, I., and Hinton, G. E. (2015a). Grammar as a foreign language. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 2773–2781.

[442] Vinyals, O., Toshev, A., Bengio, S., and Erhan, D. (2015b). Show and tell: A neural image caption generator. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 3156–3164.

[443] Wager, S., Wang, S., and Liang, P. S. (2013). Dropout training as adaptive regularization. In Burges, C. J. C., Bottou, L., Welling, M., Ghahramani, Z., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 26*, pages 351–359. Curran Associates, Inc.

[444] Wallis, J. and Miller, T. (1991). Three-dimensional display in nuclear medicine and radiology. *Journal of nuclear medicine : official publication, Society of Nuclear Medicine*, 32(3):534–546.

[445] Wallis, J. W. (1992). *Cardiovascular Nuclear Medicine and MRI: Quantitation and Clinical Applications*, pages 89–100. Springer Netherlands.

[446] Wallis, J. W., Miller, T. R., Lerner, C. A., and Kleerup, E. C. (1989). Three-dimensional display in nuclear medicine. *IEEE Trans. on Medical Imaging*, 8(4):297–230.

[447] Wan, L., Zeiler, M. D., Zhang, S., LeCun, Y., and Fergus, R. (2013). Regularization of neural networks using dropconnect. In *ICML (3)*, volume 28 of *JMLR Proceedings*, pages 1058–1066. JMLR.org.

[448] Wang, C. and Mahadevan, S. (2008). Manifold alignment using procrustes analysis. In *Machine Learning, Proceedings of the Twenty-Fifth International Conference (ICML 2008), Helsinki, Finland, June 5-9, 2008*, pages 1120–1127.

[449] Wang, X., Li, L., Lockington, D., Pullar, D., and Jeng, D. (2005). Self-organizing polynomial neural network for modelling complex hydrological processes. *Research Report No R861, Department of Civil Engineering*.

[450] Wang, X. and Wang, Y. (2014). Improving content-based and hybrid music recommendation using deep learning. In *ACM Multimedia*, pages 627–636. ACM.

[451] Warde-Farley, D., Goodfellow, I. J., Courville, A. C., and Bengio, Y. (2014). An empirical analysis of dropout in piecewise linear networks. In *International Conference on Learning Representations (ICLR2014)*.

[452] Weng, J., Ahuja, N., and Huang, T. S. (1992). Cresceptron: a self-organizing neural network which grows adaptively. In *International Joint Conference on Neural Networks (IJCNN)*, volume 1, pages 576–581. IEEE.

[453] Weng, J. J., Ahuja, N., and Huang, T. S. (1997). Learning recognition and segmentation using the cresceptron. *International Journal of Computer Vision*, 25(2):109–143.

[454] Werbos, P. J. (1974). *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University.

[455] Werbos, P. J. (1981). Applications of advances in nonlinear sensitivity analysis. In *Proceedings of the 10th IFIP Conference, 31.8 - 4.9, NYC*, pages 762–770.

[456] Weston, J., Chopra, S., and Bordes, A. (2014). Memory networks. *arXiv preprint arXiv:1410.3916*.

[457] Weston, J., Ratle, F., and Collobert, R. (2008). Deep learning via semi-supervised embedding. In *Machine Learning, Proceedings of the Twenty-Fifth International Conference (ICML 2008), Helsinki, Finland, June 5-9, 2008*, pages 1168–1175.

[458] Weston, J., Ratle, F., Mobahi, H., and Collobert, R. (2012). Deep learning via semi-supervised embedding. In Montavon, G., Orr, G., and Muller, K.-R., editors, *Neural Networks: Tricks of the Trade*. Springer.

[459] Widrow, B. (1960). An Adaptive "ADALINE" Neuron Using Chemical "Memistors". Technical report, Solid-State Electronics Laboratory, Stanford Electronics Laboratories, Stanford University, Stanford, California.

[460] Wiesel, D. H. and Hubel, T. N. (1959). Receptive fields of single neurones in the cat's striate cortex. *J. Physiol.*, 148:574–591.

[461] Wiesler, S., Richard, A., Schlüter, R., and Ney, H. (2014). Mean-normalized stochastic gradient for large-scale deep learning. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2014 : 4 - 9 May 2014, Florence, Italy*, International Conference on Acoustics Speech and Signal Processing ICASSP, pages 180–184, Piscataway, NJ. IEEE International Conference on Acoustics, Speech and Signal Processing, Florence (Italy), 4 May 2014 - 9 May 2014, IEEE.

[462] Wiesler, S., Schlüter, R., and Ney, H. (2011). A convergence analysis of log-linear training and its application to speech recognition. In *2011 IEEE Workshop on Automatic Speech Recognition Understanding*, pages 1–6.

[463] Winter, R. and Widrow, B. (1988). Madaline rule ii: a training algorithm for neural networks. In *IEEE 1988 International Conference on Neural Networks*, pages 401–408 vol.1.

[464] Wolpert, D. H. (1996). The lack of a priori distinctions between learning algorithms. *Neural Comput.*, 8(7):1341–1390.

[465] Woodworth, R. S. and Thorndike, E. (1901). The influence of improvement in one mental function upon the efficiency of other functions.(i). *Psychological review*, 8(3):247.

[466] Wu, X. and Srihari, R. (2004). Incorporating prior knowledge with weighted margin support vector machines. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '04, pages 326–333, New York, NY, USA. ACM.

[467] Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A. C., Salakhutdinov, R., Zemel, R. S., and Bengio, Y. (2015). Show, attend and tell: Neural image caption generation with visual attention. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 2048–2057.

[468] Xue, G. and Ye, Y. (2000). An efficient algorithm for minimizing a sum of p-norms. *SIAM Journal on Optimization*, 10(2):551–579.

[469] Xue, Y., Liao, X., Carin, L., and Krishnapuram, B. (2007). Multi-task learning for classification with dirichlet process priors. *Journal of Machine Learning Research*, 8:35–63.

[470] Yang, Y. and Hospedales, T. M. (2016a). Deep multi-task representation learning: A tensor factorisation approach. *CoRR*, abs/1605.06391.

[471] Yang, Y. and Hospedales, T. M. (2016b). Trace norm regularised deep multi-task learning. *CoRR*, abs/1606.04038.

[472] Yip, C., Dinkel, C., Mahajan, A., Siddique, M., Cook, G., and Goh, V. (2015). Imaging body composition in cancer patients: visceral obesity, sarcopenia and sarcopenic obesity may impact on clinical outcome. *Insights into Imaging*, pages 489–497.

[473] Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. (2014). How transferable are features in deep neural networks? In *NIPS*, pages 3320–3328.

[474] Yu, A., Palefsky-Smith, R., and Bedi, R. (2016). Deep reinforcement learning for simulated autonomous vehicle control. *Course Project Reports: Winter*, pages 1–7.

[475] Yu, T., Jan, T., Simoff, S., and Debenham, J. (2007). Incorporating prior domain knowledge into inductive machine learning. *Unpublished doctoral dissertation Computer Sciences*.

[476] Yu, T., Simoff, S., and Jan, T. (2010). VQSVM: A case study for incorporating prior domain knowledge into inductive machine learning. *Neurocomputing*, 73(13-15):2614–2623.

[477] Zeiler, M. (2012a). ADADELTA: An Adaptive Learning Rate Method. *CoRR*, abs/1212.5701.

[478] Zeiler, M. D. (2012b). ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701.

[479] Zeiler, M. D. and Fergus, R. (2013). Stochastic pooling for regularization of deep convolutional neural networks. In *International Conference on Learning Representations (ICLR2013)*.

[480] Zeiler, M. D. and Fergus, R. (2014). Visualizing and understanding convolutional networks. In *ECCV (1)*, volume 8689 of *Lecture Notes in Computer Science*, pages 818–833. Springer.

[481] Zen, H., Tokuda, K., and Black, A. (2009). Statistical parametric speech synthesis. *Speech Communication*, 51(11):1039–1064.

[482] Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. (2016). Understanding deep learning requires rethinking generalization. *CoRR*, abs/1611.03530.

[483] Zhang, J., Shan, S., Kan, M., and Chen, X. (2014a). Coarse-to-Fine Auto-Encoder Networks (CFAN) for Real-Time Face Alignment. In *ECCV, Part II*, pages 1–16.

[484] Zhang, K., Zuo, W., Gu, S., and Zhang, L. (2017a). Learning deep CNN denoiser prior for image restoration. In *CVPR*, pages 2808–2817. IEEE Computer Society.

[485] Zhang, X., Das, S., Neopane, O., and Kreutz-Delgado, K. (2017b). A design methodology for efficient implementation of deconvolutional neural networks on an FPGA. *CoRR*, abs/1705.02583.

[486] Zhang, Y., David, P., and Gong, B. (2017c). Curriculum domain adaptation for semantic segmentation of urban scenes. *CoRR*, abs/1707.09465.

[487] Zhang, Y. and Yang, Q. (2017). A survey on multi-task learning. *CoRR*, abs/1707.08114.

[488] Zhang, Z., Luo, P., Loy, C. C., and Tang, X. (2014b). Facial landmark detection by deep multi-task learning. In *Computer Vision, ECCV 2014, 13th European Conference*, pages 94–108.

[489] Zhuang, F., Cheng, X., Luo, P., Pan, S. J., and He, Q. (2015). Supervised representation learning: Transfer learning with deep autoencoders. In *IJCAI*, pages 4119–4125. AAAI Press.

Appendix A

Definitions and Technical Details

Due to the space limitation in the introductory chapter of this thesis (chapter 1), and in order to ease its reading, we decide to provide a separate appendix that covers some basic definitions in machine learning (Sec.A.1), and some technical details (Sec.A.2) which are not mandatory in order to go through the rest of the thesis. However, they cover some important aspects including:

1. bias-variance tradeoff (Sec.A.2.1),
2. feedforward networks (Sec.A.2.2), including (a) backpropagation, derivatives computation, and issues (Sec.A.2.2.1), (b) nonlinear activation functions (Sec.A.2.2.2, A.2.2.3), (c) universal approximation theorem (Sec.A.2.2.4), (d) and other neural architectures (Sec.A.2.2.5).
3. and the impact of some regularization approaches on the obtained solution (Sec.A.2.3) including L_p norm regularization (Sec.A.2.3.1), and early stopping (Sec.A.2.3.2).

A.1 Machine Learning Definitions

We discuss in this section further details on machine learning (Sec.1.1). We illustrate its use throughout different applications in real life (Sec.A.1.1), while we provide some basic definitions (Sec.A.1.2) and different learning scenarios (Sec.A.1.3).

A.1.1 Applications

Machine learning algorithms have been successfully deployed in a variety of applications, including • Text or document classification, e.g., spam detection; • Natural language processing, e.g., part-of-speech tagging, statistical parsing, name-entity recognition; • Speech recognition, speech synthesis, speaker verification; • Computational biology applications, e.g., protein function or structural prediction; • Computer vision tasks, e.g., image recognition, face detection; • Fraud detection (credit card, telephone), and network intrusion; • Games, e.g., chess, backgammon, go; • Unassisted vehicle control (robots, navigation); • Medical diagnosis; • Recommendation systems, search engines, information extraction systems.

This list is by no means comprehensive, and learning algorithms are applied to new applications every day. Moreover, such applications correspond to a wide variety of learning problems. Some major classes of learning problems are:

- *Classification*: Assign a category to each item. For example, document classification may assign items with categories such as *politics*, *business*, *sports*, or *weather*.
- *Regression*: Predict a real value for each item. Examples of regression include prediction of stock values or variations of economic variables. In this problem, the penalty for an incorrect prediction depends on the magnitude of difference between the true and predicted values.
- *Ranking*: Order items according to some criterion. Web search, e.g., returning web pages relevant to a search query, is the canonical ranking example.
- *Clustering*: Partition items into homogeneous regions. Clustering is often performed to analyze very large data sets. For example, in the context of social network analysis, clustering algorithms attempt to identify "communities" within large groups of people.
- *Dimensionality reduction or manifold learning*: Transform an initial representation of items into a lower-dimensional representation of the items while preserving some properties of the initial representation. A common example involves preprocessing digital images in computer vision tasks.

In the next section, we provide basic definitions and terminology that are used in machine learning.

A.1.2 Terminology

We use the canonical problem of spam detection as a running example to illustrate some basic definitions and to describe the use and evaluation of machine learning algorithms in practice [305]. Spam detection is the problem of learning to automatically classify email messages as either `SPAM` or `not-SPAM`.

- *Examples*: Items or instances of data used for learning or evaluation. In our spam problem, these examples correspond to the collection of email messages we will use for learning and testing.
- *Features*: The set of attributes, often represented as a vector, associated to an example. In the case of email messages, some relevant features may include the length of the message, the name of the sender, various characteristics of the header, the presence of certain keywords in the body of the message, and so on.
- *Labels*: Values or categories assigned to examples. In classification problems, examples are assigned specific categories, for instance, the `SPAM` and `not-SPAM` categories in our binary classification problem. In regression, items are assigned real-valued labels.

- *Training samples*: Examples used to train a model. In our spam problem, the training samples consist of a set of email examples along with their associated labels.

- *Validation samples*: Examples used to tune the parameters of a model when working with labeled data. Models typically have one or more free parameters, and the validation samples are used to select appropriate values for such free parameters.

- *Test samples*: Examples used to evaluate the performance of a learned model. The test samples are separate from the training and validation data and is not made available in the learning stage. In the spam problem, the test samples consist of a collection of email examples for which the learned model must predict labels based on features. These predictions are then compared with the labels of the test samples to measure the performance of the model.

- *Loss function*: A function that measures the difference, or loss, between a predicted label and a true label. Denoting the set of all labels as $\mathcal{Y}$ and the set of possible predictions as $\mathcal{Y}'$, a loss function $\ell : \mathcal{Y}' \times \mathcal{Y} \to \mathbb{R}_+$. In most cases, $\mathcal{Y}' = \mathcal{Y}$ and the loss function is bounded, but these conditions do not always hold. Common examples of loss functions include the 0-1 (or misclassification) loss defined over $\{-1, +1\} \times \{-1, +1\}$ by $\ell(y', y) = 1_{y' \neq y}$ and the squared loss defined over $I \times I$ by $\ell(y', y) = (y' - y)^2$, where $I \subseteq \mathbb{R}$ is typically a bounded interval.

- *Hypothesis set*: A set of functions mapping features (feature vectors) to the set of label $\mathcal{Y}$. In our example, these may be a set of functions mapping email features to $\mathcal{Y} = \{\texttt{SPAM}\ , \texttt{not-SPAM}\}$. More generally, hypotheses may be functions mapping features to a different set $\mathcal{Y}'$. They could be linear functions mapping email feature vector to real numbers interpreted as scores ($\mathcal{Y} = \mathbb{R}$), with higher score values more indicative of `SPAM` than lower ones.

We now define the learning stages of our spam problem. We start with a given collection of labeled examples. We first randomly partition the data into training samples, validation samples, and test samples. The size of each of these samples depends on a number of different considerations. For example, the amount of data reserved for validation depends on the number of free parameters of the model. Also, when labeled samples are relatively small, the amount of training data is often chosen to be larger than that of test data since the learning performance directly depends on the number of training samples.

Next, we associate relevant features to the examples. This is a critical step in the design of machine learning solutions. Useful features can effectively guide the learning of the model, while poor or uninformative ones can be misleading. Although it is critical, to a large extent, the choice of the features is left to the user. This choice reflects the user's prior knowledge about the learning task which in practice can have a dramatic effect on the performance results.

Now, we use the features selected to train our model by fixing different values of its free parameters. For each value of these parameters, the learning algorithm selects a different hypothesis, i.e., model, out of the hypothesis set. We choose among them the

model resulting in the best performance on the validation samples. Finally, using that model, we predict the labels of the examples in the test samples. The performance of the model is evaluated by using the loss function associated to the task, e.g., the 0-1 loss in our spam detection task, to compare the predicted and true value.

Thus, the performance of a model is of course evaluated based on its test error and not its error on the training samples. A model may be consistent, that is it may commit no error on the examples of the training data, and yet have a poor performance on the test data. This occurs for consistent models defined by very complex decision surfaces, as illustrated in Fig.A.1, which tend to memorize a relatively small training samples instead of seeking to generalize well. This highlights the key distinction between memorization and generalization, which is the fundamental property sought for an accurate model.

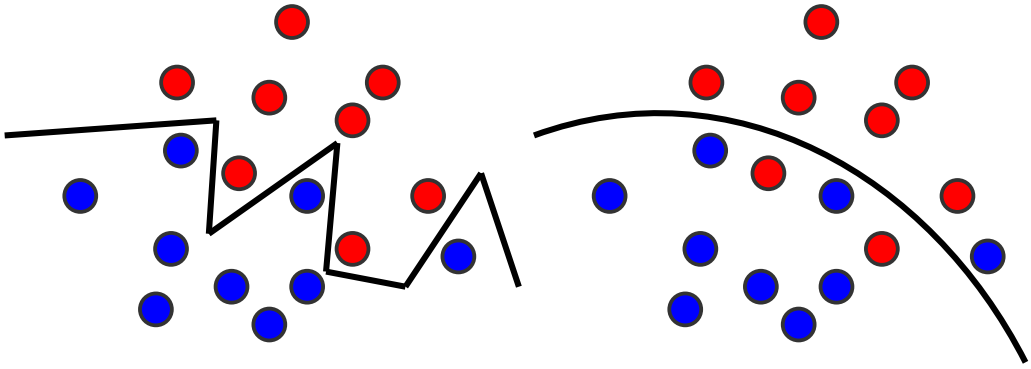


Fig. A.1 The zig-zag line on the left panel is consistent over the blue and red training samples, but it is a complex separation surface that is not likely to generalize well to unseen data. In contrast, the decision surface on the right panel is simpler and might generalize better in spite of its misclassification of few points of the training samples. (Reference: [305])

In the following, we describe some common learning scenarios.

A.1.3 Learning Scenarios

We briefly describe common machine learning scenarios [305]. These scenarios differ in the type of training data available to the learner, the order and method by which training data is received and the test data used to evaluate the model.

- *Supervised learning*: The learner receives a set of labeled examples as training data and makes predictions for unseen points. This is the most common scenario associated with classification, regression, and ranking problems.
- *Unsupervised learning*: The learner exclusively receives unlabeled training data, and makes prediction for unseen points. Since in general no labeled example is available in that setting, it can be difficult to quantitatively evaluate the performance of a learner. Clustering and dimensionality reduction are example of unsupervised learning problems.

- *Semi-supervised learning*: The learner receives some training samples which consist of both labeled and unlabeled data, and makes predictions for unseen points. Semi-supervised learning is common in setting where unlabeled data is easily accessible but labels are expensive to obtain. Various types of problems arising in applications, including classification, regression, or ranking tasks, can be framed as instances of semi-supervised learning. The hope is that the distribution of unlabeled data accessible to the learner can help to achieve a better performance than in the supervised setting.
- *Transductive inference*: As in the semi-supervised scenario, the learner receives labeled training samples along with a set of unlabeled test points. However, the objective of transductive inference is to predict labels only for these particular test points. Transductive inference appears to be an easier task and matches the scenario encountered in a variety of modern applications. However, the assumptions under which a better performance can be achieved in this setting are research questions that have not been fully solved.
- *Online learning*: In contrast with the previous scenarios, the online scenario involves multiple rounds and training and testing phases are intermixed. At each round, the learner receives an unlabeled training point, makes a prediction, receives the true label, and incurs a loss. The objective in the online setting is to minimize the cumulative loss over all rounds. Unlike the previous settings just discussed, no distributional assumption is made in online learning.
- *Reinforcement learning*: The training and the testing phases are also intermixed in reinforcement learning. To collect information, the learner actively interacts with the environment and in some cases affects the environment, and receives an immediate reward for each action. The object of the learner is to maximize its reward over a course of actions and iterations with the environment. However, no long-term reward feedback is provided by the environment, and the learner is faced with the exploration versus exploitation dilemma, since it must choose between exploring unknown actions to gain more information versus exploiting the information already collected.
- *Active learning*: The learner adaptively or interactively collects training examples, typically by querying an oracle to request labels for new points. The goal in active learning is to achieve a performance comparable to the standard supervised learning scenario, but with fewer labeled examples. Active learning is often used in applications where labels are expensive to obtain, for example computational biology applications.

In practice, many other intermediate and somewhat more complex learning scenarios may be encountered.

A.2 Technical Details

A.2.1 Bias-variance Tradeoff

The bias-variance tradeoff is the problem of minimizing two sources of errors that prevent a model from well generalizing beyond the train data. The first error source is

named the bias error which comes from erroneous assumption about the complexity of the model which means this error depends on the performance of the model in average while considering an infinite number of training samples. In the other hand, the variance error which comes from the sensitivity of the model to small variations in the data. This depends on the model capacity to model the random noise (small perturbations) in the data.

The bias-variance decomposition [141] provides a way to analyze the expected generalization error of a model. This decomposition gives the generalization error as a sum of two terms: the bias and the variance.

In order to formalize this decomposition, let us consider a regression problem over the distribution $p_{(\mathbf{x},\mathbf{y})}$. $\mathbb{D}_{\text{train}} \sim p_{(\mathbf{x},\mathbf{y})}$ is a sampled training data. For clearer exposition, let us take $\mathbf{y} = y$ to be a one-dimentional, although the results apply more generally [141]. $\mathbb{E}[y|\boldsymbol{x}]$ denotes as a deterministic function that gives the value y conditioned on a fixed $\boldsymbol{x}$. $\mathbb{E}[y|\boldsymbol{x}]$ can be seen as the best value y for $\boldsymbol{x}$. For any function $f(\boldsymbol{x})$, and any fixed $\boldsymbol{x}$, the regression error is [141]

$$\begin{aligned}
\mathbb{E}\left[(y - f(\boldsymbol{x}))^2|\boldsymbol{x}\right] &= \mathbb{E}\left[((y - \mathbb{E}[y|\boldsymbol{x}]) + (\mathbb{E}[y|\boldsymbol{x}] - f(\boldsymbol{x})))^2|\boldsymbol{x}\right] && \text{(A.1)}\\
&= \mathbb{E}\left[(y - \mathbb{E}[y|\boldsymbol{x}])^2|\boldsymbol{x}\right] + (\mathbb{E}[y|\boldsymbol{x}] - f(\boldsymbol{x}))^2 \\
&\quad + 2\,\mathbb{E}\left[(y - \mathbb{E}[y|\boldsymbol{x}])|\boldsymbol{x}\right] \cdot (\mathbb{E}[y|\boldsymbol{x}] - f(\boldsymbol{x})) && \text{(A.2)}\\
&= \mathbb{E}\left[(y - \mathbb{E}[y|\boldsymbol{x}])^2|\boldsymbol{x}\right] + (\mathbb{E}[y|\boldsymbol{x}] - f(\boldsymbol{x}))^2 \\
&\quad + 2(\mathbb{E}[y|\boldsymbol{x}] - \mathbb{E}[y|\boldsymbol{x}]) \cdot (\mathbb{E}[y|\boldsymbol{x}] - f(\boldsymbol{x})) && \text{(A.3)}\\
&= \mathbb{E}\left[(y - \mathbb{E}[y|\boldsymbol{x}])^2|\boldsymbol{x}\right] + (\mathbb{E}[y|\boldsymbol{x}] - f(\boldsymbol{x}))^2\,. && \text{(A.4)}
\end{aligned}$$

In other words, among all functions of $\boldsymbol{x}$, $f(\boldsymbol{x}) = \mathbb{E}[y|\boldsymbol{x}]$ is the best predictor of y given $\boldsymbol{x}$, in the mean-squared-error.

Now, let us introduce the dependency of $f(\boldsymbol{x})$ to its training sample $\mathbb{D}_{\text{train}}$ and let us note $f(\boldsymbol{x}; \mathbb{D}_{\text{train}})$. For clarity, we refer to $\mathbb{D}_{\text{train}}$ by $\mathbb{D}$. Now, the generalization error over a new fixed example $\boldsymbol{x}$ and fixed training sample $\mathbb{D}$ is computed as follows [141]

$$\mathbb{E}\left[(y - f(\boldsymbol{x}; \mathbb{D}))^2|\boldsymbol{x}, \mathbb{D}\right] = \mathbb{E}\left[(y - \mathbb{E}[y|\boldsymbol{x}, \mathbb{D}])^2|\boldsymbol{x}, \mathbb{D}\right] + (f(\boldsymbol{x}; \mathbb{D}) - \mathbb{E}[y|\boldsymbol{x}])^2\,. \quad \text{(A.5)}$$

The term $\mathbb{E}\left[(y - \mathbb{E}[y|\boldsymbol{x}, \mathbb{D}])^2|\boldsymbol{x}, \mathbb{D}\right]$ does not depend on the data $\mathbb{D}$, nor on f. It is simply the variance of y given $\boldsymbol{x}$. Therefore, only the term $(f(\boldsymbol{x}; \mathbb{D}) - \mathbb{E}[y|\boldsymbol{x}])^2$ measures the effectiveness of f to predict y. The mean-squared error of f as an estimator of the best prediction $\mathbb{E}[y|\boldsymbol{x}]$ is given by [141]

$$\mathop{\mathbb{E}}_{\mathbb{D}}\left[(f(\boldsymbol{x}; \mathbb{D}) - \mathbb{E}[y|\boldsymbol{x}])^2\right]\,, \quad \text{(A.6)}$$

where $\mathop{\mathbb{E}}_{\mathbb{D}}$ is the expectation with respect to the training set, $\mathbb{D}$, that is the average over all possible sampled training samples $\mathbb{D}$.

The error measured in Eq.A.6 can be further developed for any $\boldsymbol{x}$ as follows [141]

$$\mathop{\mathbb{E}}_{\mathbb{D}}\left[(f(\boldsymbol{x};\mathbb{D})-\mathbb{E}[y|\boldsymbol{x}])^2\right]=\mathop{\mathbb{E}}_{\mathbb{D}}\left[((f(\boldsymbol{x};\mathbb{D})+\mathop{\mathbb{E}}_{\mathbb{D}}[f(\boldsymbol{x};\mathbb{D})])+(\mathop{\mathbb{E}}_{\mathbb{D}}[f(\boldsymbol{x};\mathbb{D})]-\mathbb{E}[y|\boldsymbol{x}]))^2\right] \tag{A.7}$$

$$=\mathop{\mathbb{E}}_{\mathbb{D}}\left[(f(\boldsymbol{x};\mathbb{D})-\mathop{\mathbb{E}}_{\mathbb{D}}[f(\boldsymbol{x};\mathbb{D})])^2\right]+\mathop{\mathbb{E}}_{\mathbb{D}}\left[(\mathop{\mathbb{E}}_{\mathbb{D}}[f(\boldsymbol{x};\mathbb{D})]-\mathbb{E}[y|\boldsymbol{x}])^2\right]$$
$$+2\mathop{\mathbb{E}}_{\mathbb{D}}\left[(f(\boldsymbol{x};\mathbb{D})-\mathop{\mathbb{E}}_{\mathbb{D}}[f(\boldsymbol{x};\mathbb{D})])\cdot(\mathop{\mathbb{E}}_{\mathbb{D}}[f(\boldsymbol{x};\mathbb{D})]-\mathbb{E}[y|\boldsymbol{x}])\right] \tag{A.8}$$

$$=\mathop{\mathbb{E}}_{\mathbb{D}}\left[(f(\boldsymbol{x};\mathbb{D})-\mathop{\mathbb{E}}_{\mathbb{D}}[f(\boldsymbol{x};\mathbb{D})])^2\right]+(\mathop{\mathbb{E}}_{\mathbb{D}}[f(\boldsymbol{x};\mathbb{D})]-\mathbb{E}[y|\boldsymbol{x}])^2$$
$$+2\mathop{\mathbb{E}}_{\mathbb{D}}\left[f(\boldsymbol{x};\mathbb{D})-\mathop{\mathbb{E}}_{\mathbb{D}}[f(\boldsymbol{x};\mathbb{D})]\right]\cdot(\mathop{\mathbb{E}}_{\mathbb{D}}[f(\boldsymbol{x};\mathbb{D})]-\mathbb{E}[y|\boldsymbol{x}]) \tag{A.9}$$

$$=\underbrace{(\mathop{\mathbb{E}}_{\mathbb{D}}[f(\boldsymbol{x};\mathbb{D})]-\mathbb{E}[y|\boldsymbol{x}])^2}_{\text{Bias}}+\underbrace{\mathop{\mathbb{E}}_{\mathbb{D}}\left[(f(\boldsymbol{x};\mathbb{D})-\mathop{\mathbb{E}}_{\mathbb{D}}[f(\boldsymbol{x};\mathbb{D})])^2\right]}_{\text{Variance}}. \tag{A.10}$$

From Eq.A.10, one can see that the bias is the mean error of the average models which are trained over infinite samples. Therefore, this error depends only on the model capability to model the data, i.e., model complexity. Similarly, the variance shows the capability of the model to model the variations of the data, again this is related to the complexity of the model.

As a consequence, models with small capacity will tend to have high bias because they are enable to fit well the data and low variance because they do not consider the variation in the data. In the other hand, models with high capacity will have lower bias because they can fit well the data, but they have high variance because they are sensitive to changes in the data. Hence, a tradeoff is necessary to select a model with lower bias and variance. Fig.A.2 shows a typical behavior of model bias and variance with respect to the model capacity. When the capacity of the model increases, the bias tends to decrease and the variance to increase yielding an U-shape of the generalization error. Above the optimum capacity, the model tends to have lower bias and higher variance. This relation is similar to the relation between the capacity, underfitting and overfitting.

When applying the regularization over a model that overfits (high capacity), we attempts to bring it from the overfitting regime toward the optimal regime by reducing its variance but without introduction significant bias.

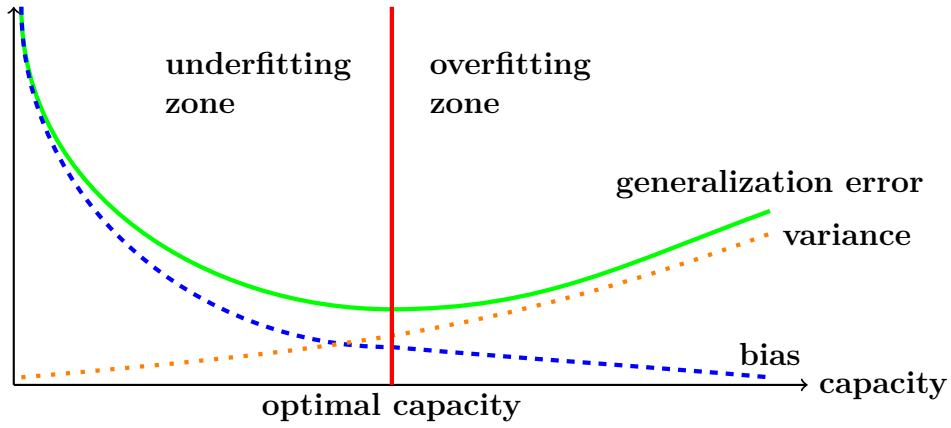


Fig. A.2 Typical relationship between model capacity, its bias and variance and the generalization error. (Reference: [154])

A.2.2 Feedforward Neural Networks

We cover in this section more technical details on neural networks including 1. backpropagation, derivatives computation, and issues, (Sec.A.2.2.1), 2. nonlinear activation functions (Sec.A.2.2.2, A.2.2.3), 3. universal approximation theorem (Sec.A.2.2.4), 4. and other neural architectures (Sec.A.2.2.5).

A.2.2.1 Backpropagation, Computing the Derivatives, and Issues

The error minimization using gradient descent [177] in the parameters space of differentiable systems has been discussed since the 60s [233, 65, 66]. Steepest descent in the weights space of such systems can be performed [65, 233, 64] by iterating the chain rule [260, 263] to dynamic programming [38]. A simplified derivation of this backpropagation using the chain rule can be found in [112].

Explicit Backpropagation for arbitrary and discrete neural network-like systems was first described in 1970 in the master thesis of the Finnish mathematician and computer scientist Seppo Linnainmaa [267, 268] although without referencing neural networks. He did implement it in FORTRAN. Now, this method is mostly known as automatic differentiation [171].

In 1974, Paul Werbos was the first to suggest the possibility to use the backpropagation described by Seppo Linnainmaa, after studying it in depth in his thesis [454], to train neural networks. However, Werbos did not publish his work at the time probably because of the AI winter until 1981 [455]. Related works were published later [334, 253, 254]. In 1986, a paper [369] by David Rumelhart, Geoffrey Hinton, and Ronald Williams made the backpropagation a popular method for training neural networks with multi-layers where they showed useful internal representations learned at the hidden layers.

Up to the day of writing this thesis, backpropagation algorithm is the most dominant approach for training neural networks.

Technical Details

Let us consider a multi-layer perceptron with K layers where $\hat{\boldsymbol{y}}_k, k = 1, \cdots, K$ is the output of each layer. $\hat{\boldsymbol{y}}_0 = \boldsymbol{x}$ is the input vector of size $1 \times D + 1$. The following layers can be computed as

$$\hat{\boldsymbol{y}}_k = \phi_k(\hat{\boldsymbol{y}}_{k-1} \cdot \boldsymbol{W}_k), \forall k = 1, \cdots, K \,, \tag{A.11}$$

with $\phi(\cdot)$ is an activation function, $\hat{\boldsymbol{y}}_K$ is the output vector of size $1 \times M$. $\ell(\cdot, \cdot)$ is the per-sample loss function (Eq.1.21). If $\phi(\cdot)$ is differentiable, one can compute the gradient for each layer $\hat{\boldsymbol{y}}_{k-1}$ using the gradient of the layer $\hat{\boldsymbol{y}}_k$ using the chain rule as follows [154, 103]

$$\frac{\partial \ell}{\partial \hat{y}_{k-1}^i} = \sum_{j=1}^{M} \frac{\partial \ell}{\partial \hat{y}_k^j} \cdot \frac{\partial \hat{y}_k^j}{\partial \hat{y}_{k-1}^i} \tag{A.12}$$

$$= \sum_{j=1}^{M} \frac{\partial \ell}{\partial \hat{y}_k^j} \cdot \left. \frac{\partial \phi_k(z^j)}{\partial z^j} \right|_{z^j = \hat{\boldsymbol{y}}_{k-1} \cdot \boldsymbol{W}_k^j} \cdot \frac{\partial z^j}{\partial \hat{y}_{k-1}^i} \tag{A.13}$$

$$= \sum_{j=1}^{M} \frac{\partial \ell}{\partial \hat{y}_k^j} \cdot \left. \frac{\partial \phi_k(z^j)}{\partial z^j} \right|_{z^j = \hat{\boldsymbol{y}}_{k-1} \cdot \boldsymbol{W}_k^j} \cdot W_k^{ij} \,, \tag{A.14}$$

where $\boldsymbol{W}_k^j$ is the vector of weights connecting the layer $\hat{\boldsymbol{y}}_{k-1}$ with the neuron j at the layer $\hat{\boldsymbol{y}}_k$. Using Eq.A.14, one can compute the gradient of the previous layer $\frac{\partial \ell}{\partial \hat{\boldsymbol{y}}_{k-1}}$ using: the gradient of the current layer $\frac{\partial \ell}{\partial \hat{\boldsymbol{y}}_k}$, the derivative of the activation function $\nabla_{\boldsymbol{z}} \phi_k(\boldsymbol{z})$, and the layer weights W_k. Therefore, it is possible to compute iteratively the gradient of all layers from $K - 1$ to 0 using $\frac{\partial \ell}{\partial \hat{y}_K}$ in Eq.1.25 at the first iteration. Using the chain rule, one can compute the weights gradients $\frac{\partial \ell}{\partial \boldsymbol{W}_k}$ as follows [154, 103]

$$\frac{\partial \ell}{\partial W_k^{ij}} = \frac{\partial \ell}{\partial \hat{y}_k^j} \cdot \frac{\partial \hat{y}_k^j}{\partial W_k^{ij}} \tag{A.15}$$

$$= \frac{\partial \ell}{\partial \hat{y}_k^j} \cdot \left. \frac{\partial \phi_k(z^j)}{\partial z^j} \right|_{z^j = \hat{\boldsymbol{y}}_{k-1} \cdot \boldsymbol{W}_k^j} \cdot \frac{\partial z^j}{\partial W_k^{ij}} \tag{A.16}$$

$$= \frac{\partial \ell}{\partial \hat{y}_k^j} \cdot \left. \frac{\partial \phi_k(z^j)}{\partial z^j} \right|_{z^j = \hat{\boldsymbol{y}}_{k-1} \cdot \boldsymbol{W}_k^j} \cdot \hat{y}_{k-1}^i \,. \tag{A.17}$$

Therefore, using Eq.A.17, one can compute the gradients of all weights at each layer from K to 1 once $\frac{\partial \ell}{\partial \hat{\boldsymbol{y}}_k}$ is available.

The backpropagation algorithm can be divided into three steps: forward, backward and weight gradient computation.

1. *Forward pass*: Initialize the input vector $\hat{\boldsymbol{y}}_0$ to some input samples $\boldsymbol{x}$. Then, iteratively compute the following layers $\hat{\boldsymbol{y}}_k$, for $k = 1, \cdots, K$. Fig.1.4 illustrates the forward pass.

2. *Backward pass*: Initialize the gradients estimations $\frac{\partial \ell}{\partial \hat{\boldsymbol{y}}_K}$ using Eq.1.25. Then, propagate them back through the network layers from $K-1$ to 0 to estimate $\frac{\partial \ell}{\partial \hat{\boldsymbol{y}}_k}$ using Eq.A.14. Fig.A.3 illustrates the backward pass.

3. *Weight updates*: The weights gradients can be obtained in parallel or after finishing the whole backward pass using $\hat{\boldsymbol{y}}_k$ and $\frac{\partial \ell}{\partial \hat{\boldsymbol{y}}_k}$ at each layer (Eq.A.17). The weights gradients are then used to update the weights using Eq.1.23. Besides stochastic gradient descent, different methods have been developed to search the minimum of a loss function, such as AdaGrad [477] which adapts the learning rate for each weight, Nesterov accelerated gradient descent [313] with a convergence rate of $\frac{1}{t^2}$, LBFGS algorithm [270] which uses second order gradients information. However, SGD remains the most commonly used method for training neural networks.

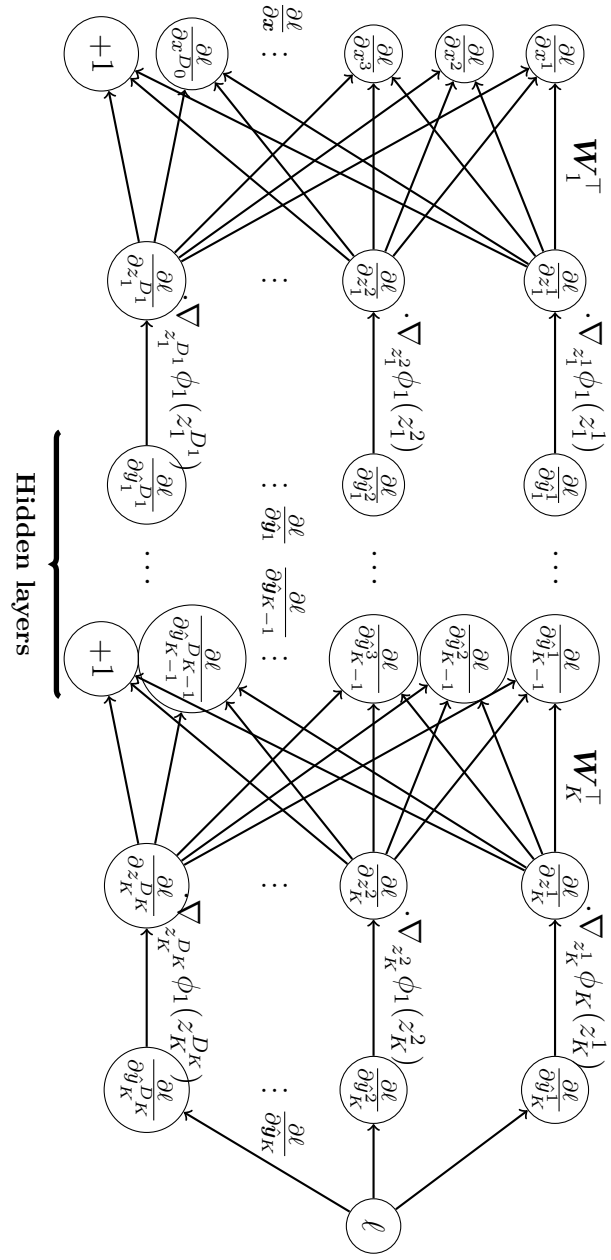


Fig. A.3 The backward pass of the backpropagation algorithm. (Notation: $D_k, k = 1, \cdots, K$ is the dimension of the output of the layer k. $D_0 = D$ is the dimension of the input $\boldsymbol{x}$ of the network. $D_K = M$ is the dimension of the output $\hat{\boldsymbol{y}}$ of the network, i.e. M. x^i is the i^{th} component of $\boldsymbol{x}$. $\hat{y}_k^j$ is the j^{th} component of the output representation $\hat{\boldsymbol{y}}_k$ at the layer k)

Matrix Representation of the Backpropagation

In practice, it is easier to use a matrix representation of the backpropagation for efficient computation. For this, we consider every single transformation linear or nonlinear as a single layer. However, for simplicity, we keep the notation K as the total number of layers. In this case, a layer k can be formalized as a function as follows

$$\hat{\boldsymbol{y}}_k = f_k(\hat{\boldsymbol{y}}_{k-1}, \boldsymbol{W}_k) \ , \tag{A.18}$$

which transforms the the vector $\hat{\boldsymbol{y}}_{k-1}$ into the vector $\hat{\boldsymbol{y}}_k$ using the weights $\boldsymbol{W}_k$. Moreover, the loss function $\ell(\cdot, \cdot)$ is considered as the last layer $\hat{\boldsymbol{y}}_{K+1}$ with dimension 1×1. For simplicity, we refer to it as $\ell(\boldsymbol{x})$. The forward pass is the composition of all functions $\ell(\boldsymbol{x}) = f_{K+1}(f_K(\cdots f_1(\boldsymbol{x}) \cdots))$ applied to the input vector $\boldsymbol{x}$.

Let us consider the following notations:

1. The vector of derivatives with respect to layer values

$$\boldsymbol{dy}_k = \frac{\partial \ell}{\partial \hat{\boldsymbol{y}}_k} \ . \tag{A.19}$$

2. The backward backpropagation functions, referred to as reverse functions $\bar{f}(\boldsymbol{dy}_k, \boldsymbol{W}_k)$ which computes the vector of derivatives of the previous layer $\hat{\boldsymbol{y}}_{k-1}$ using the derivatives of the next layer $\boldsymbol{dy}_k$ and its parameters $\boldsymbol{W}_k$. From Eq.A.14, one can write

$$\boldsymbol{dy}_{k-1} = \bar{f}(\boldsymbol{dy}_k, \boldsymbol{W}_k) = \boldsymbol{dy}_k \cdot \boldsymbol{J}_{\hat{\boldsymbol{y}}_k}(\hat{\boldsymbol{y}}_{k-1}) \ , \tag{A.20}$$

where $\boldsymbol{J}_{\hat{\boldsymbol{y}}_k}(\hat{\boldsymbol{y}}_{k-1})$ is the Jacobian matrix of the derivatives $\frac{\partial \hat{\boldsymbol{y}}_k}{\partial \hat{\boldsymbol{y}}_{k-1}}$. We note that the first Jacobian $\boldsymbol{J}_{\hat{\boldsymbol{y}}_{K+1}}(\hat{\boldsymbol{y}}_K) = \boldsymbol{dy}_K = \frac{\partial \ell}{\partial \hat{\boldsymbol{y}}_K}$.

Most of the functions f_k are one of the two following types:

- *Linear with weights*: Therefore, $\hat{\boldsymbol{y}}_k = \hat{\boldsymbol{y}}_{k-1} \cdot \boldsymbol{W}_k \Rightarrow \boldsymbol{dy}_{k-1} = \boldsymbol{dy}_k \cdot \boldsymbol{W}_k^\top$.
- *Nonlinear without weights*: Therefore, $\hat{\boldsymbol{y}}_k = \phi_k(\hat{\boldsymbol{y}}_{k-1}) \Rightarrow \boldsymbol{dy}_{k-1} = \boldsymbol{dy}_k \cdot \boldsymbol{J}_{\phi_k}(\hat{\boldsymbol{y}}_{k-1})$. Usually, the nonlinear function $\phi_k(\cdot)$ is element-wise, therefore, they have a squared and diagonal Jacobian $\boldsymbol{J}_{\phi_k}(\hat{\boldsymbol{y}}_{k-1})$.

Now, we introduce the notation of the vector weight gradients

$$\boldsymbol{dW}_k = \frac{\partial \ell}{\partial \boldsymbol{W}_k} \ . \tag{A.21}$$

Then, from Eq.1.24, we can write

$$\boldsymbol{dW}_k = \boldsymbol{J}_{\hat{\boldsymbol{y}}_k}(\boldsymbol{W}_k) \cdot \boldsymbol{dy}_k \ , \tag{A.22}$$

where $\boldsymbol{J}_{\hat{\boldsymbol{y}}_k}(\boldsymbol{W}_k)$ is the Jacobian matrix of the derivatives with respect to the weights $\frac{\partial \hat{\boldsymbol{y}}_k}{\partial \boldsymbol{W}_k}$. In this case, only the linear functions with weights are considered. As a consequence, their Jacobian is equivalent to $\hat{\boldsymbol{y}}_{k-1}^\top$:

$$\boldsymbol{dW}_k = \hat{\boldsymbol{y}}_{k-1}^\top \cdot \boldsymbol{dy}_k \ . \tag{A.23}$$

The backpropagation diagram using a matrix representation is illustrated in Fig.A.4.

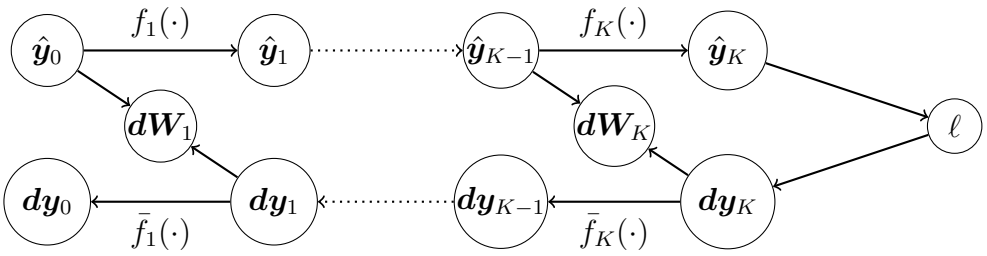


Fig. A.4 The backpropagation algorithm in a matrix form. (Reference: [103])

Backpropagation Issues

By the end of 1980′s, it seems that the backpropagation by itself was not enough to train neural networks with many hidden layers, i.e., deep networks. Most applications focused on neural networks with few hidden layers, i.e., shallow networks. Adding more hidden layers did not often offer empirical benefits. This practical limitation of the backpropagation was accepted at the time. Moreover, the idea of using shallow networks was motivated furthermore by a theorem [242, 191, 208] (Sec.A.2.2.4) that states that a neural network with one hidden layer with enough units can approximate any multivariate continuous function with arbitrary accuracy.

The issue raised in training deep neural networks using backpropagation was fully understood by 1991 where Hochreiter presented in his diploma thesis [204] a breakthrough work which clarified this issue. Hochreiter's work formally identified a major reason of the backpropagation failure to train deep networks. Typically, deep networks suffer from what is now known as vanishing gradients or exploding gradients. With standard activation functions such as the sigmoid function, cumulative backpropagated error signals can either shrink rapidly and vanish, or grow out of bounds and explode. The former is more likely to happen in feedforward networks while the later is more known as an issue in recurrent neural networks.

Over the years, several approaches to partially overcome this fundamental deep learning issue have been proposed and most of them are based on augmenting the backpropagation using unsupervised learning. First, the model is pre-trained using unsupervised learning, then fine-tuned using supervised learning. For example, this technique has been explored in recurrent neural networks [378, 379]. Deep feedforward networks can be also pre-trained in a layer-wise fashion by stacking auto-encoders [23, 50, 439]. Similarly, Deep Belief Networks (DBNs) can be pre-trained [201, 199] by stacking many Restricted Boltzmann Machines (RBMs) [399]. Recurrent neural networks can benefit from gradient clipping [294, 335, 336] to avoid exploding the gradients. Activation functions that can saturate may lead quickly to vanishing the gradients such as the sigmoid or the hyperbolic tangent functions which saturate at either tail (0 or 1 for the sigmoid and -1 and 1 for the hyperbolic tangent). This saturation leads local gradients to be almost zero which causes the gradients to vanish. To avoid this, new activation functions with no saturation regime have been proposed such as the Rectified Linear Unit (ReLU) [223, 310, 149], Leaky ReLU [277], Parametric

ReLU [186] and maxout [158] which generalizes the Leaky and the Parametric ReLU (Sec.A.2.2.2). Hessian-free optimization can help alleviate the problem for feedforward neural networks [307, 338, 284] and recurrent neural networks [285]. Moreover, today's GPUs provide a huge computational power that allows for propagating errors a few layers further down within reasonable time. This makes implementing deep neural network easier, accessible, and helps popularizing such models in different domains.

In the next section, we present some types of nonlinear activation functions.

A.2.2.2 Nonlinear Activation Functions

Training neural networks using gradient based methods requires all the activation functions to be differentiable. For this reason, the Heaviside step function can no longer be used in such setup. Therefore, its approximation with another differentiable function which has similar shape is used instead. We mention two most commonly used approximations of the sigmoid functions which are the logistic function

$$\phi(z) = \text{sigm}(z) = \frac{1}{1 + \exp^{(-z)}} \Rightarrow \forall z,\ \text{sigm}(z) \in [0, 1], \tag{A.24}$$

and hyperbolic tangent function

$$\phi(z) = \tanh(z) = \frac{1 - \exp^{(-2z)}}{1 + \exp^{(-2z)}} \Rightarrow \forall z,\ \tanh(z) \in [-1, 1]\ . \tag{A.25}$$

Both sigmoid and hyperbolic tanget function are related to each other

$$\tanh(z) = 2\,\text{sigm}(2z) - 1\ . \tag{A.26}$$

Fig.A.5 illustrates both functions.

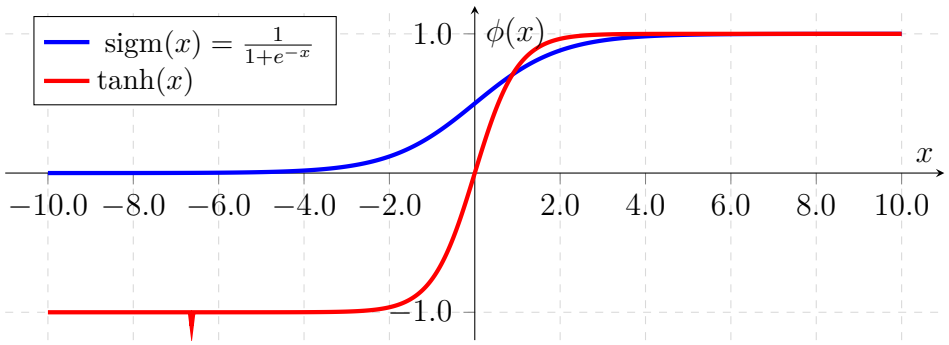


Fig. A.5 Examples of nonlinear activation functions. *blue*: Logistic sigmoid function. *red*: Hyperbolic tangent function

Both activation functions have a convenient gradient $\nabla_z \phi(z)$ that can be computed using the function itself. For logistic function it is formulated as

$$\frac{\partial\,\text{sigm}(z)}{\partial z} = \frac{\exp^{(-z)}}{(1 + \exp^{(-z)})^2} = \text{sigm}(z)(1 - \text{sigm}(z))\ , \tag{A.27}$$

while for the hyperbolic tangent function, it is formulated as

$$\frac{\partial \tanh(z)}{\partial z} = \frac{4 \exp^{(-2z)}}{(1 + \exp^{(-2z)})^2} = 1 - \tanh^2(z) \ . \tag{A.28}$$

The activation functions described in this section are mostly used in the hidden units of a neural network. However, they can still be used at the output layer, depending on the output target. In Sec.A.2.2.3, we discuss a specific type of activation function for the output layer.

Sigmoid activation functions are not always the best choice. They raise an issue for gradient learning methods. Sigmoid functions are sensitive to z only around 0. They have an inconvenient property by falling into a saturation regime once the magnitude of z is very high. This saturation drives the gradient to be very close to 0 which either stops the learning process or makes it very slow.

Designing new activation functions is an active field. Recently, new functions have been proposed with much care about the saturation issue. For instance, REctified Linear Unit [223, 310, 149] (Fig.A.6)

$$\phi(z) = \text{relu}(z) = \max(z, 0) = z \cdot 1_{(z>0)} \ , \tag{A.29}$$

is much faster in computation and does not have the saturation problem. Moreover, it allows a faster training [244]. Its derivative is 1 when z is positive, and 0 when it is negative

$$\frac{\partial \text{relu}(z)}{\partial z} = 1_{(z>0)} \ . \tag{A.30}$$

ReLU function is differentiable everywhere except in $z = 0$. However, this does not seem to raise issues in practice [244]. This issue is dealt with in implementation where usually the value 1 is returned as the value of the derivative at 0 [154].

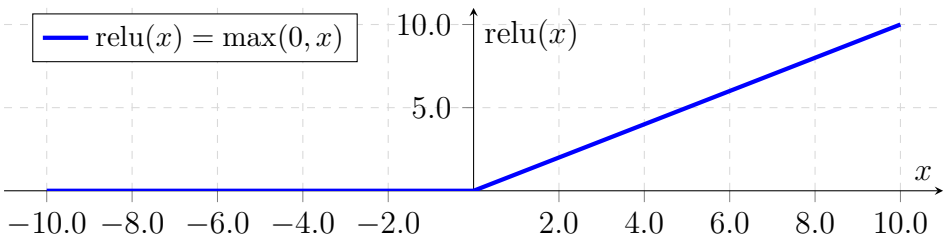


Fig. A.6 Rectified linear unit function.

Later, more improvements were proposed such as Leaky ReLU [277], parametric ReLU [186]. Maxout activation function [158] generalizes ReLU function. Instead of applying an element-wise function, maxout function divides z into k groups of values. Each maxout unit then outputs the maximum element of the group

$$\text{maxout}_i(\boldsymbol{z}) = \max_{j \in [1,k]} z_{ij} \ . \tag{A.31}$$

For instance, in the case of convolutional layer, maxout takes the maximum over a group of feature maps.

Many other types of activation functions are possible but are less commonly used. It is impractical to list them all within this thesis. More details on this matter can be found in [154].

In the next section, we present an activation function for the output layer.

A.2.2.3 Activation Function for Output Units: Classification Case

The activation function of the output layer is considered separately because they must be set according to the target nature $\mathbf{y}$. For instance, in the case of regression with $\boldsymbol{y} \in [0,1]^M$ one can consider using the logistic function. In the case $\boldsymbol{y} \in [-1,1]^M$, the hyperbolic tangent function can be more suitable. In either cases, one can consider a linear output.

In classification case, things are slightly different. In the case of binary classification, one can consider one single output unit to model a Bernoulli distribution $\hat{y} = \hat{y}_K = P(y = 1|\boldsymbol{x})$. In this case, the logistic function, which has the range $[0, 1]$, is enough [154]

$$\hat{y} = \hat{y}_K = \text{sigm}_K(z) = P(y = 1|\boldsymbol{x}) \quad , \text{ such as } z = \hat{\boldsymbol{y}}_{K-1} \cdot \boldsymbol{W}_K \ . \tag{A.32}$$

Therefore,

$$1 - \hat{y}_K = 1 - \text{sigm}_K(z) = P(y = 0|\boldsymbol{x}) \quad , \text{ such as } z = \hat{\boldsymbol{y}}_{K-1} \cdot \boldsymbol{W}_K \ . \tag{A.33}$$

These results are valid under the use of the maximum log-likelihood as a cost function [154].

In the case of M classes, $M > 2$, we need M output units to predict a vector of values to model a multinoulli distribution

$$\hat{y}_K^j = P(y = j|\boldsymbol{x}) \ . \tag{A.34}$$

In this case, the output vector $\hat{\boldsymbol{y}}$ must satisfy two conditions in order to form a probability distribution

$$\forall j \in [1, M], \hat{y}_K^j \geq 0, \ \sum_{j=1}^{M} \hat{y}_K^j = 1 \ . \tag{A.35}$$

The softmax function was designed to fill the two conditions

$$\text{softmax}(\boldsymbol{z})_j = \frac{\exp^{(z_j)}}{\sum_{t=1}^{M} \exp^{(z_t)}} \quad , \ \boldsymbol{z} = \hat{\boldsymbol{y}}_{K-1} \cdot \boldsymbol{W}_K \ . \tag{A.36}$$

To compute the value of one single output unit, the softmax needs to use the value of all the output units.

When using the maximum log-likelihood, we want to maximize [154]

$$\log P(y = j|\boldsymbol{x}) = \log \text{softmax}(\boldsymbol{z})_j = z_j - \log \sum_{t=1}^{M} z_t \ . \tag{A.37}$$

When maximizing the log-likelihood, the first term encourages z_j to be pushed up while the second term encourages all z_t to be pushed down. If we consider that $\exp^{(z_t)}$ is insignificant for any z_t that is noticeably less than $\max_t z_t$, the second term can be approximated as: $\log \sum_{t=1}^{M} z_t \approx \max_t z_t$. Therefore, the maximum log-likelihood always strongly penalizes the most active incorrect prediction. If the correct answer has the strongest activation, then the term $\log \sum_{t=1}^{M} \exp^{(z_t)} \approx \max_t z_t = z_j$ will roughly cancel the first term z_j.

The softmax function has a convenient derivative

$$\frac{\partial \operatorname{softmax}(\boldsymbol{z})_i}{\partial z_j} = \operatorname{softmax}(\boldsymbol{z})_i \times (\delta_{ij} - \operatorname{softmax}(\boldsymbol{z})_j) \ , \tag{A.38}$$

where δ_{ij} is the Kronecker delta function. We note that the Jacobian matrix (Eq.A.20) of the softmax is still squared but no longer diagonal, however it is symmetric.

The cross-entropy or the negative maximum likelihood are the common losses used for optimization combined with the softmax function [154]

$$\ell(f(\boldsymbol{x}), \boldsymbol{y}) = -\sum_{j=1}^{M} y_j \times \log f(\boldsymbol{x})_j, \qquad f(\boldsymbol{x})_j = \operatorname{softmax}(\boldsymbol{z})_j = \operatorname{softmax}(\hat{y}_K^j \times \boldsymbol{W}_K^{:,j}) \ . \tag{A.39}$$

To simplify the notation, let us set $\ell(\boldsymbol{z}) = \ell(f(\boldsymbol{x}), \boldsymbol{y})$ and $\phi(\boldsymbol{z}) = \operatorname{softmax}(\boldsymbol{z})$. Therefore, $l(\boldsymbol{z}) = \sum_{j=1}^{M} y_j \times \log \phi(\boldsymbol{z})_j$. Now let us compute the derivative of the softmax function with respect to its input values $\boldsymbol{z}$:

$$\frac{\partial \ell(\boldsymbol{z})}{\partial z_j} = \sum_{i=1}^{M} \frac{\partial \ell(\boldsymbol{z})}{\partial \phi(\boldsymbol{z})_i} \times \frac{\partial \phi(\boldsymbol{z})_i}{\partial z_j} \tag{A.40}$$

$$= \sum_{i=1}^{M} \frac{y_i}{\phi(\boldsymbol{z})_i} \times \phi(\boldsymbol{z})_i \times (\delta_{ij} - \phi(\boldsymbol{z})_j) \tag{A.41}$$

$$= \sum_{i=1}^{M} y_i \times \delta_{ij} - \phi(\boldsymbol{z})_j \times \sum_{i=1}^{M} y_i \tag{A.42}$$

$$= y_i - \phi(\boldsymbol{z})_i \Rightarrow \frac{\partial \ell(\boldsymbol{z})}{\partial \boldsymbol{z}} = \boldsymbol{y} - \phi(\boldsymbol{z}) \ . \tag{A.43}$$

Therefore, the derivative of the softmax function with respect to the input values $\boldsymbol{z}$ has a simple formulation which simplifies its implementation and improves numerical stability. In practice, other tricks are actually used even to compute the softmax function in order to avoid numerical issues.

In the next section, we present a well known theoretical property of neural networks. We discuss also their depth and its relation to generalization in practice.

A.2.2.4 Universal Approximation Properties and Depth

Neural networks models has acquired the reputation that they lack theoretical foundations. The universal approximation theorem [98, 208, 136] is one of the few theoretical justifications of the capability of neural networks. For instance in [98], Cybenko demon-

strated that a feedforward network with a linear output and at least one hidden layer with any "squashing" activation function such as the logistic function can approximate any continuous function on a compact subset of $[0,1]^m$ with any desired nonzero error, provided that the hidden layer has enough units. A geometric way to understand this result is as follows:

- A continuous function on a compact set can be approximated by a piecewise constant function.
- A piecewise constant function can be represented as a neural network as follows: for each region where the function is constant, use a neural network as an indicator function for that region. Then, build a final layer with a single node, whose input linear combination is the sum of all the indicators multiplied by a weight equals to the constant value of the corresponding region in the original piecewise constant function.

Simply, one can take the continuous function over $[0,1]^m$, and some target error $\epsilon > 0$, then grid the space $[0,1]^m$ at a scale $\rho > 0$ to end up with $(1/\rho)^m$ subcubes so that the function which is constant over each subcube is within ϵ of the target function. A neural network can not precisely represent an indicator function, but it can get very close to it. For $m = 1$, it can be shown that a neural network with one hidden layer containing two hidden units can build a "bump" function which is constant over an interval $[x_0, x_1]$ within the original compact set $[0,1]$. The "squashing" function is important in Cybenko theorem. It must have a known limit when the input goes to $+\infty$ where the limit is 1 and when the input goes to $-\infty$ the limit is 0. One can handcraft the neural network parameters in order to fix x_0, x_1 and in order to build the indicator function. For instance, this can be done by setting the weights of the hidden layer to be extremely huge in order to push the input of the activation function to its limit (0 or 1). Cybenko was aiming at providing a general theorem with minimal layers. He showed that one hidden layer with enough units is enough to construct an approximation with ϵ error.

Formally, let $\phi(\cdot)$ be a non-constant, bounded and monotonically increasing continuous function. Let $\mathbb{I}_m$ denote the m-dimensional unit hypercube $[0,1]^m$. The space of continuous functions on $\mathbb{I}_m$ is denoted by $C(\mathbb{I}_m)$. Then, given any $\epsilon > 0$ and any function $f \in C(\mathbb{I}_m)$, there exists an integer N, real constants $v_i\ , b_i \in \mathbb{R}$ and real vector $\boldsymbol{w}_i \in \mathbb{R}^m$, where $i = 1, \cdots, N$ such that we may define

$$F(\boldsymbol{x}) = \sum_{i=1}^{N} v_i \phi(\boldsymbol{w}_i^\top \boldsymbol{x} + b_i)\ , \tag{A.44}$$

as an approximate realization of the function f where f is independent of ϕ; that is

$$\Big|F(\boldsymbol{x}) - f(\boldsymbol{x})\Big| < \epsilon\ ,\ \forall \boldsymbol{x} \in \mathbb{I}_m\ . \tag{A.45}$$

Cybenko used in his proof the Hahn-Banach theorem [275]. In the same year, Hornik found the same results [208] using the Stone-Weiestrass theorem [368]. Independently, Funahashi proved similar theorem [136] using an integral formula presented in [217].

A year later, Hornik showed [209] that feedforward networks with sigmoid units can approximate not only an unknown function, but also its derivative. Using a theorem

in [413], Light extended [266] Cybenko's results to any continuous function on $\mathbb{R}^n$. Moreover, Light showed that the sigmoid function can be replaced by any continuous function that satisfies some conditions.

Earlier to this, in 1957, Kolmogorov provided a theorem [241] that states that one can express a continuous multivariate function on a compact set in terms of sums and compositions of a finite number of single variable functions. The main difference between Cybenko's theorem [98] is that Kolmogorov [241] uses heterogeneous activation functions at the hidden layer whereas Cybenko [98] uses the same activation function. Kolmogorov formulation can be seen as a composition of different neural networks in parallel where each one has a specific type of activation function. Due to this aspect of the theorem, part of the research community [317, 190, 269] suggested that Kolmogorov [241] theorem provided theoretical support for the universality of neural networks, while others disagreed [144].

More details on the universal approximation theorem can be found in [183, 96, 9]. [11, 407, 10] cover more studies on approximation theory.

As much as the universal approximation theorem [98, 208, 136] provides a strong theoretical support for neural networks, it does not provide much insight on its application. None of the authors of the theorem provides a way to determine how many hidden neurons are required. Most importantly, they did not provide a learning algorithm to find the network parameters. In addition, the authors use sigmoid functions in the hidden units which we know that they are not the best choice as they cause saturation and prevent learning of the network when using gradient based methods. Later, the universal approximation theorem has been proved using a wider class of activation functions which include the rectified linear unit [262].

The universal approximation theorem simply states that independently of what function we are seeking to learn, there exist a multilayer perceptron with one large hidden layer that is able to represent this function with certain precision. [25] provides some bounds on the size of a single layer network needed to approximate a broad class of functions. In the worst case, an exponential number of hidden units may be required. Even if we know the exact number of hidden units which is high-likely to be large, there is no guaranty that the learning algorithm will succeed to learn the right parameters. One of the reasons of this failure is that most likely the network will overfit the training set due to the large number of parameters. In addition, the no free lunch theorem [464] shows that there is no universally superior learning machine. Therefore, the universal approximation theorem proves that feedforward networks are an universal system for representing any function in a sense that given a function, there exists a feedforward network with a one large hidden layer that approximates this function with certain precision. Nor the size of the hidden layer, nor the training procedure to find the network parameters are known. Moreover, there is no guaranty that the network will generalize well to unseen inputs.

Depth of the Network and Generalization

In practice and in many circumstances, it was found that using deeper models can reduce the need of large number of hidden units to represent the desired function and can reduce the generalization error. [308] showed that functions representable with a deep rectifier network can require an exponential number of hidden units with a network with one hidden layer. More precisely, they showed that piecewise networks which can

be obtained using rectifier nonlinearities or maxout unit can represent functions with a number of regions that is exponential in the depth of the network.

Deep models can also be motivated from statistical view [154]. Choosing a deep model encodes a general belief that the function we want to learn should involve the composition of several simple functions. This can be interpreted from a representation point of view as saying that the learning problem consists of discovering a set of underlying factors of variation that can in turn be described in terms of other simpler underlying factors of variation. Alternatively, one may interpret using deep models as computer program that process the data step by step in order to reach the output decision. The intermediate steps are not necessarily factors of variation but may be seen as pointers that the network uses to organize its internal process. Empirically, greater depth seems to result in better generalization for a wide variety of tasks [244, 396] (Fig.A.7, A.8). However, the need to deep model is merely an empirical fact which is not supported by any theoretical foundation. Recently, it was shown that shallow networks can perform as well as deep networks [19].

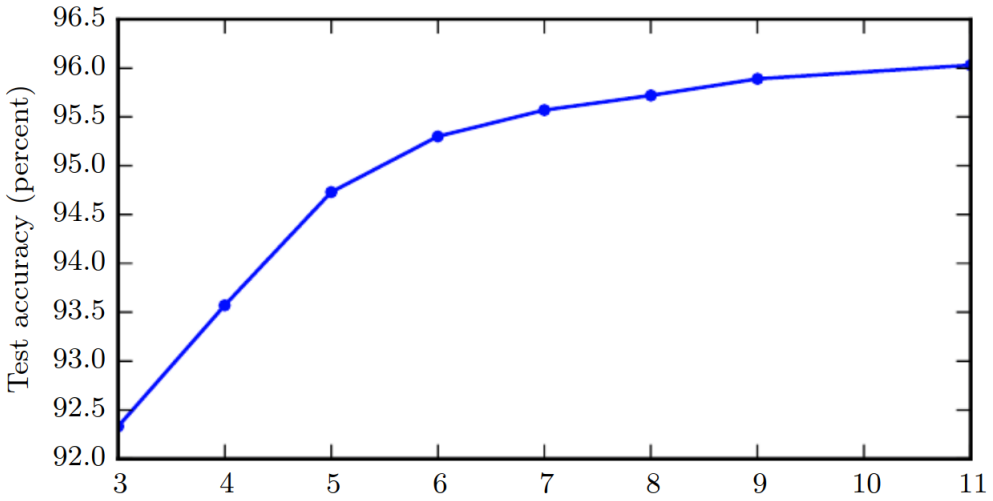


Fig. A.7 Effect of depth. Empirical results showing that deeper networks generalize better when used to transcribe multidigit numbers from photographs of addresses. Image from [155]. (Credit: [154])

Neural network domain has a wide range of architectures that differ mostly depending on the task in hand. In the next section, we describe some of the common architectures.

A.2.2.5 Other Neural Architectures

Up to now, we have discussed only feedforward networks which are composed of sequential layers one after the other such as multilayer perceptron and convolutional neural networks. These models are directed acyclic graphs where the information streams from a layer to the next one from the input layer toward the output layer without loops. Sometimes, the information at a layer i can "skip" the layer $i+1$ to reach the layer $i+2$ or higher [187, 188, 405]. All our work presented in this thesis is

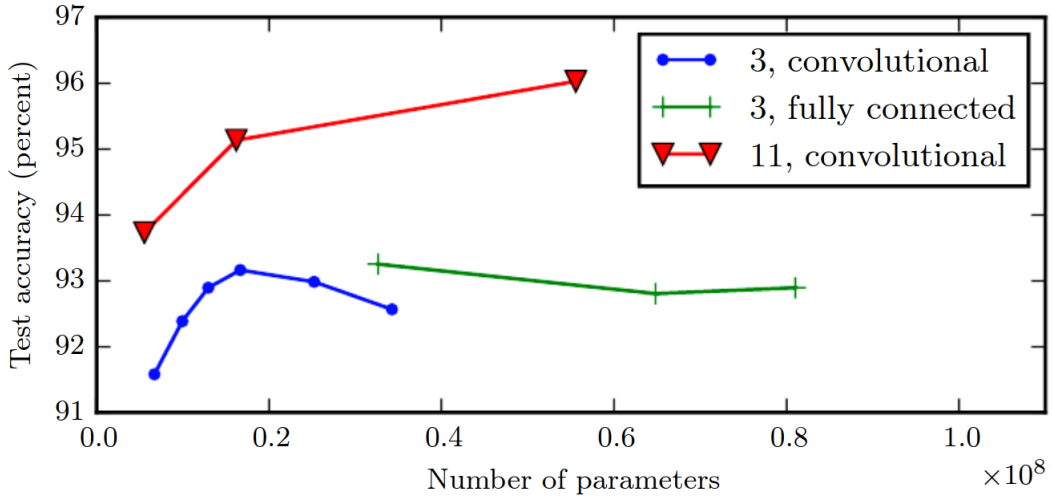


Fig. A.8 Effect of the number of parameters. Deeper models tend to perform better compared to less deep models with the same number of parameters. Image from [155]. (Credit: [154])

validated on feedforward neural networks type.

Boltzmann Machines

One can build a neural model by considering an undirected graph, without direct loop from one neuron to itself such as Boltzmann Machines (BMs) [124, 6, 202]. BMs are a type of neural models built using stochastic neurons where each neuron is directly connected to every neuron in the model. They are used to learn arbitrary probability distributions over binary vectors. The set of neurons (stochastic variables) are usually divided into two sets to indicate that the input signal is partial observable. A set of variables indicates the observable part of the signal referred to as visible units while the hidden part is referred to as hidden units. The joint probability distribution of all the variables is described using an energy function. The hidden units state can be inferred given the visible states. Moreover, the visible states can also be inferred given the hidden states which makes this type of architecture a generative model. BMs can be simplified by removing the connection between units at the same layer which leads to what is known as Restricted Boltzmann Machines (RBMs). RBMs can be stacked to form a deep network which can be used for feature extraction or for discrimination such as in Deep Belief Networks (DBNs) [198, 195]. Unlike RBMs which contain only one layer of hidden units, Deep Boltzmann Machines (DBMs) have several layers of hidden units [372] which have been applied in several tasks including document modeling [403]. Training Boltzmann Machines-like models is known to be difficult [196] which includes using contrastive divergence [193] or stochastic maximum likelihood algorithms [154].

Recurrent Neural Networks

Most feedforward networks such as MLPs can process only fixed length input. In many applications, the input is a sequence and the same goes for the output. Instead of using one MLP per element of the sequence, the MLP is shared across all the

inputs which is similar to the idea of weight sharing in convolutional networks where the same filter is applied across the whole 2D image. Moreover, when dealing with sequence data, it is useful to remember what happened in the past. Therefore, the MLP needs its previous states as input, hence, a recurrence is made. Recurrent Neural Networks (RNNs) [369] are a simple MLP duplicated over the input sequence where they take as input their previous output and the current input in the sequence. In theory an RNN with sufficient number of hidden units can approximate any measurable sequence-to-sequence mapping to arbitrary accuracy [180]. For many sequence labeling tasks, it is beneficial to have access to future as well past context. Bidirectional Recurrent Neural Networks (BRNNs) [383, 382, 22] offer the possibility to do so and showed interesting improvements such as in protein secondary structure prediction [22, 75] and speech processing [382, 130]. Training RNNs is done using a variant of backpropagation algorithm by taking in consideration the dependencies to compute the gradient. It is referred to as BackPropagation Through Time (BPTT). When increasing the size of the input sequence, RNNs trained with BPTT seem to have difficulties learning long-term dependencies [337]. This problem is mostly due the vanishing/exploding gradients issue [204, 205, 52]. Many attempts were proposed to deal with this issue. Among them, we cite the Long Short-Term Memory (LSTM) [206] architecture which is similar to vanilla RNN but with a crucial addition to help the network memorizes long-term dependencies. The LSTM models have been found very successful in many applications, such as unconstrained handwriting recognition [168], speech recognition [167, 165], handwriting generation [163], machine translation [416], image captioning [239, 442, 467], and parsing [441]. Variant architectures based on LSTMs were proposed known as Gated Recurrent Units (GRUs) [80, 84, 85] where the main difference is how to control the network memory. More work has been done in order to improve the memory aspect of RNNs to deal with sequence-data by providing an entire working memory that allows the networks to hold and manipulate the most relevant information. [456] introduced memory networks that contain memory cells that can be addressed via an addressing system using a supervised way. [169, 170] introduced the neural Turing machine which enables learning to read/write arbitrary content in the memory cells without explicit supervision based on an attention mechanism [20]. This addressing mechanism has become a standard [412, 227, 247]. The attention mechanism is probably the next big step in recurrent neural networks. Instead of processing all the input sequence and map it into a fixed length vector, the attention mechanism allows the use of all the internal states (memory cells) of the network by taking their weighted combination in order to produce a single output. This can be interpreted as a memory access in computer. However, in the attention mechanism, the access is performed to all the memory instead of selected cells. More details on recurrent neural networks and the last advances can be found in [154, 161].

Generative Models

Neural networks can also be used to generate new samples. For instance, variational auto-encoders [108, 238, 373] make a strong assumption about the distribution of the latent variables, for example, assuming a Gaussian distribution. One can sample new hidden codes from the learned hidden distribution, then, use these new hidden codes to generate new samples. Generative Adversarial Networks (GANs) [156] are probably one of the hottest topics in neural networks domain these years. GANs are a class of algorithms composed of two networks. A generative network that takes random

noise as input and outputs new samples. A discriminative network that attempts to discriminate between true (real) samples and the samples generated by the generative network which are considered fake. Using this technique, one can learn to generate new samples that look real in an unsupervised way. This topic is an ongoing research subject.

More details on the last advances in neural networks can be found in [154].

A.2.2.6 Experimenting in Deep Learning

Experimenting in science is encouraged which may lead to more understating of the matter and probably new discoveries. However, heavy experimenting may lead to waist of effort, resources and more importantly, it may open the door to personal interpretations which may bias the results and miss-guide the upcoming research. Unfortunately, deep learning domain is a heavily experimental field where most the founded results are empirical which makes it weak compared to concurrent methods in machine learning, despite its practical high performance. There are many unanswered questions about neural networks and its performance. This makes it difficult to understand its results and makes it seem as a "magical black box". Hopefully, neural network field will get more theoretical support in order to set it on the right and solid direction.

In the following, we provide some technical details on regularization.

A.2.3 Regularization

We cover in this section, from a theoretical perspective, the impact of using L_p norm or early stopping, as a regularization, on the parameters of the obtained solution of a learning algorithm.

A.2.3.1 L_p Parameters Norms

We provide more details on the L_p parameters norm regularization for $p \in \{1, 2\}$ (Sec.A.2.3.1.2, Sec.A.2.3.1.1). Let us consider the regularized training objective function

$$\widetilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{\theta}), \quad \Omega(\boldsymbol{\theta}) = \frac{1}{p}\|\boldsymbol{\theta}\|_p^p \,. \tag{A.46}$$

A.2.3.1.1 L_2 Parameters Norm

For L_2 parameters norm regularization, we have

$$\Omega(\boldsymbol{\theta}) = \frac{1}{2}\|\boldsymbol{w}\|_2^2 \,. \tag{A.47}$$

For the next analysis, no bias parameters are assumed, so $\boldsymbol{\theta}$ represents only $\boldsymbol{w}$. Let us consider a general form of the objective function

$$\widetilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \frac{\alpha}{2}\boldsymbol{w}^\top\boldsymbol{w} + J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) \,, \tag{A.48}$$

and its corresponding gradient

$$\nabla_{\boldsymbol{w}} \widetilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha \boldsymbol{w} + \nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) \ . \tag{A.49}$$

Considering ϵ a learning rate, the update rule of the gradient descent at each step is performed as follows [154]

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \epsilon(\alpha \boldsymbol{w} + \nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})) \tag{A.50}$$
$$\boldsymbol{w} \leftarrow (1 - \epsilon\alpha)\boldsymbol{w} - \epsilon \nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) \ . \tag{A.51}$$

The last equation (Eq.A.51) shows that the L_2 regularization modifies the learning rule by multiplicatively shrinking the weight by a constant factor on each step just before performing the updates. In order to get more insights on what happens over the entire training process, further simplification of the analysis can be made by considering a quadratic approximation of the objective function around the value of the weights that obtains minimal unregularized training cost, $\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} J(\boldsymbol{w})$ as follows [154]

$$\hat{J}(\boldsymbol{w}) = J(\boldsymbol{w}^*) + \nabla_{\boldsymbol{w}} J(\boldsymbol{w}^*)^\top (\boldsymbol{w} - \boldsymbol{w}^*) + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^*)^\top \boldsymbol{H} (\boldsymbol{w} - \boldsymbol{w}^*) \tag{A.52}$$
$$\hat{J}(\boldsymbol{w}) = J(\boldsymbol{w}^*) + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^*)^\top \boldsymbol{H} (\boldsymbol{w} - \boldsymbol{w}^*) \ , \tag{A.53}$$

where $\boldsymbol{H}$ is the Hessian matrix of J with respect to $\boldsymbol{w}$ evaluated at $\boldsymbol{w}^*$. By definition, $\nabla_{\boldsymbol{w}} J(\boldsymbol{w}^*) = 0$ at the minimum $\boldsymbol{w}^*$. Given that $\boldsymbol{w}^*$ is a local minimum, $\boldsymbol{H}$ is a positive semidefinite matrix. In order to find the analytic form of the minimum, the gradient of $\hat{J}$ is computed as follows [154]

$$\nabla_{\boldsymbol{w}} \hat{J}(\boldsymbol{w}) = \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*) \ , \tag{A.54}$$

and solve $\nabla_{\boldsymbol{w}} \hat{J}(\boldsymbol{w}) = 0$ for $\boldsymbol{w}$. To study the effect of the weight decay, its gradient is added to Eq.A.54. Now, the regularized version of $\hat{J}$ can be solved. Let $\widetilde{\boldsymbol{w}}$ denotes the location of the minimum, therefore [154]

$$\alpha \widetilde{\boldsymbol{w}} + \boldsymbol{H}(\widetilde{\boldsymbol{w}} - \boldsymbol{w}^*) = 0 \tag{A.55}$$
$$(\boldsymbol{H} + \alpha \boldsymbol{I})\widetilde{\boldsymbol{w}} = \boldsymbol{H}\boldsymbol{w}^* \tag{A.56}$$
$$\widetilde{\boldsymbol{w}} = (\boldsymbol{H} + \alpha \boldsymbol{I})^{-1} \boldsymbol{H}\boldsymbol{w}^* \ . \tag{A.57}$$

As α approaches 0, the regularized solution $\widetilde{\boldsymbol{w}}$ approaches $\boldsymbol{w}^*$. In order to have an idea on what happens when α grows, one can proceed using matrix decomposition. $\boldsymbol{H}$ has an eigen-decomposition using a diagonal matrix $\boldsymbol{\Lambda}$ and an orthogonal basis of eigenvectors $\boldsymbol{Q}$ such that

$$\boldsymbol{H} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top \ . \tag{A.58}$$

Applying this decomposition to Eq.A.57, the following is obtained [154]

$$\widetilde{\boldsymbol{w}} = (\boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top + \alpha\boldsymbol{I})^{-1}\boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top\boldsymbol{w}^* \tag{A.59}$$

$$\widetilde{\boldsymbol{w}} = \left[\boldsymbol{Q}(\boldsymbol{\Lambda} + \alpha\boldsymbol{I})\boldsymbol{Q}^\top\right]^{-1}\boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top\boldsymbol{w}^* \tag{A.60}$$

$$\widetilde{\boldsymbol{w}} = \boldsymbol{Q}(\boldsymbol{\Lambda} + \alpha\boldsymbol{I})^{-1}\boldsymbol{\Lambda}\boldsymbol{Q}^\top\boldsymbol{w}^* \tag{A.61}$$

$$\underbrace{\boldsymbol{Q}^\top\widetilde{\boldsymbol{w}}}_{\text{Projection of } \widetilde{\boldsymbol{w}} \text{ in } \boldsymbol{Q}^\top} = \underbrace{(\boldsymbol{\Lambda} + \alpha\boldsymbol{I})^{-1}\boldsymbol{\Lambda}}_{\text{Scaling factor}= \frac{\Lambda_i}{\Lambda_i+\alpha}} \underbrace{\boldsymbol{Q}^\top\boldsymbol{w}^*}_{\text{Projection of } \boldsymbol{w}^* \text{ in } \boldsymbol{Q}^\top} . \tag{A.62}$$

Therefore, one can see that the effect of weight decay is to rescale $\boldsymbol{w}^*$ along the axes defined by the eigenvectors $\boldsymbol{Q}^\top$. The components of $\boldsymbol{w}^*$ that are aligned with the i-th eigenvector of $\boldsymbol{H}$ are rescaled by a factor of $\frac{\Lambda_i}{\Lambda_i+\alpha}$. Therefore, in the case where $\Lambda_i \gg \alpha$, the effect of the regularization is relatively small. While, in the case where $\Lambda_i \ll \alpha$, the components of the parameters will be shrunk to have nearly zero magnitude. As a result, only directions along which the parameters contribute significantly to reducing the objective function, determined by eigenvalues with high values, are preserved relatively intact. In directions that do not contribute much in reducing the objective function, a small eigenvalue of the Hessian indicates that the movement in this direction will not significantly increase the gradient. Components of the weight vector corresponding to such unimportant directions are pushed toward zero. Fig.A.9 illustrates the effect of L_2 norm regularization on the parameters search.

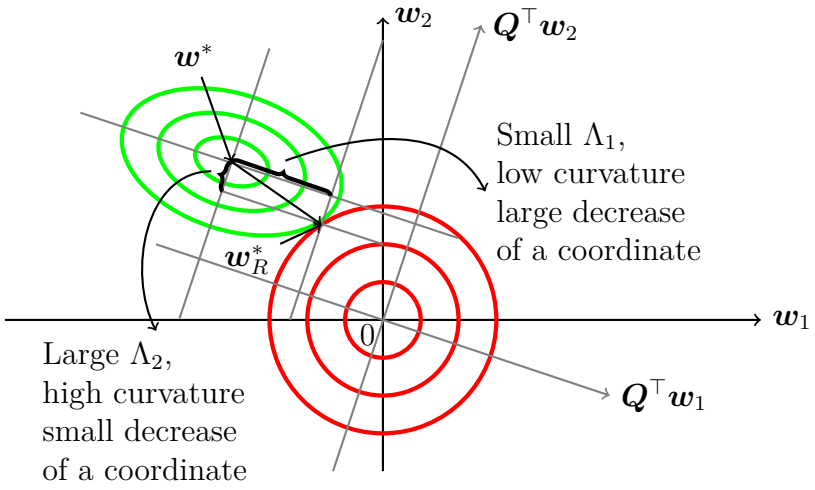


Fig. A.9 Effect of L_2 norm regularization: it scales the weights coordinates depending on the corresponding eigenvalues. (*red contours*): indicates the L_2 cost (L_2 norm of $\boldsymbol{w}$). (*green contours*): indicate the unregularized cost J. One contour indicates a set of parameters $\boldsymbol{w}$ that have the same cost. For instance, all the coordinates (w_1, w_2) that belong to the central red circle have the same L_2 norm. $\boldsymbol{w} = \boldsymbol{0}$ is the optimum solution for the L_2 cost. $\boldsymbol{w} = \boldsymbol{w}^*$ is the minimal solution of J. $\boldsymbol{w} = \boldsymbol{w}_R^*$ represents the optimum solution of the regularized cost $\widetilde{J}$ (Eq.A.48).(Reference: [103])

So far we have seen the effect of the weight decay over a general quadratic cost function. Using the same analysis, one can see its impact on a true quadratic function such as the linear regression

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{w} \ . \tag{A.63}$$

Its unregularized objective function is defined as

$$J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \|\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}\|_2^2 \ , \tag{A.64}$$

and its regularized objective function is defined as

$$\widetilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \|\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}\|_2^2 + \alpha\|\boldsymbol{w}\|_2^2 \ . \tag{A.65}$$

The solution for the normal equation Eq.A.64 is given by

$$\widetilde{\boldsymbol{w}} = (\boldsymbol{X}^\top \boldsymbol{X})^{-1}\boldsymbol{X}^\top \boldsymbol{y} \ , \tag{A.66}$$

while the solution for the regularized form is given as

$$\widetilde{\boldsymbol{w}} = (\boldsymbol{X}^\top \boldsymbol{X} + \alpha \boldsymbol{I})^{-1}\boldsymbol{X}^\top \boldsymbol{y} \ . \tag{A.67}$$

The matrix $(\boldsymbol{X}^\top \boldsymbol{X})$ in Eq.A.66 is proportional to the convariance matrix $\frac{1}{m}(\boldsymbol{X}^\top \boldsymbol{X})$. L_2 norm regularization replaces this matrix by $(\boldsymbol{X}^\top \boldsymbol{X} + \alpha \boldsymbol{I})$ in Eq.A.67. The new matrix is similar to the old one but with the addition of a positive constant α to the diagonal. The diagonal entries in $(\boldsymbol{X}^\top \boldsymbol{X})$ correspond to the variance of each input feature. Therefore, L_2 norm regularization makes the input look like it has a high variance. This shrinks the weights on features whose covariance with the output target is low compared to the added variance. From computation perspective, L_2 norm regularization reduces the numerical instability of inverting $(\boldsymbol{X}^\top \boldsymbol{X})$ by making it non-singular. More interpretations of the L_2 norm regularization can be found in [59].

A.2.3.1.2 L_1 Parameters Norm

L_1 parameters norm regularization is formulated as follows

$$\Omega(\theta) = \|\boldsymbol{w}\|_1 = \sum_i |w_i| \ . \tag{A.68}$$

While we present the L_1 norm regularization, we highlight the differences between L_1 and L_2 forms of regularization by considering a linear regression problem. L_1 weight decay controls the strength of the norm penalty by scaling Ω using a positive hyperparameter α as in L_2 norm regularization. Thus, the regularized objective function is given by

$$\widetilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha\|\boldsymbol{w}\|_1 + J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) \ , \tag{A.69}$$

with the corresponding gradient

$$\nabla_{\boldsymbol{w}}\widetilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha \text{sign}(\boldsymbol{w}) + \nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) \ , \tag{A.70}$$

where $\text{sign}(\boldsymbol{w})$ is the sign of $\boldsymbol{w}$ applied element-wise. Comparing L_1 (Eq.A.70) and L_2 (Eq.A.49), one can see that the L_1 regularization contribution is no longer scales

linearly with each weight w_i as in L_2; instead it is a constant factor with a sign equal to the sign of the parameter w_i.

Now, let us consider approximating the objective function around the minimum $\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} J(\boldsymbol{w})$ by $\hat{J}$ using Taylor expansion. In this case, the gradient of $\hat{J}$ is computed as [154]

$$\nabla_{\boldsymbol{w}} \hat{J}(\boldsymbol{w}) = \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*) \ , \tag{A.71}$$

where $\boldsymbol{H}$ is the Hessian matrix of J with respect to $\boldsymbol{w}$ evaluated at $\boldsymbol{w}^*$. However, this time further simplification are made by assuming that the Hessian is diagonal

$$\boldsymbol{H} = \text{diag}([H_{1,1}, \cdots, H_{n,n}]), \quad \forall i : H_{i,i} > 0 \ . \tag{A.72}$$

This assumption holds if the data for the linear regression problem has been preprocessed to remove all correlation between the input features. Now, the quadratic approximation of the L_1 regularized objective function can be written as [154]

$$\hat{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \mathbf{J}(\boldsymbol{w}^*; \boldsymbol{X}, \boldsymbol{y}) + \sum_i \left[\frac{1}{2} H_{i,i} (w_i - w_i^*)^2 + \alpha |w_i| \right] \ . \tag{A.73}$$

Eq.A.73 has an analytic solution for each dimension w_i under the following form [154]

$$w_i = \text{sign}(w_i^*) \max \left\{ |w_i^*| - \frac{\alpha}{H_{i,i}}, 0 \right\} \ . \tag{A.74}$$

When $w_i^* > 0$, there are two possible outcomes:

1. $w_i^* \leq \frac{\alpha}{H_{i,i}}$: The optimal value of w_i is simply $w_i = 0$. This occurs when the contribution of the L_1 regularization penalty takes over the objective function $J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$.

2. $w_i^* > \frac{\alpha}{H_{i,i}}$: In this case, the regularization does not set the optimal value of w_i to zero but instead shift it in the direction of zero by $\frac{\alpha}{H_{i,i}}$.

In the case where $w_i^* < 0$, the L_1 penalty makes w_i less negative by a distance $\frac{\alpha}{H_{i,i}}$ or set it to zero.

Comparing to L_2, L_1 regularization results in a solution where most of the parameters are zero, i.e., a sparse solution. The sparsity behavior in L_1 is different than the one in L_2 where the parameters are pushed toward zero in some cases. Eq.A.61 gives the solution $\widetilde{\boldsymbol{w}}$ for L_2 regularization. If the same assumption, used in the case of L_1, is considered about the Hessian matrix, one can find that $\widetilde{w}_i = \frac{H_{i,i}}{H_{i,i}+\alpha} w_i^*$. Therefore, if w_i^* is nonzero, $\widetilde{w}_i$ remains nonzero. This shows that L_2 does not promote sparse solutions, while L_1 regularization may set a subset of the parameters to zero for large enough α. Fig.A.10 illustrates the geometric effect introduced by the L_1 regularization on the parameters search. This sparsity aspect plays an important role in machine learning particularly as a feature selection mechanism. Feature selection simplifies machine learning by choosing which subset of input features are relevant to predict the output target. This has a key role in application where the interpretation is highly important. For instance, when building a model to predict a disease based on set of

input features, it is important to know which factors are implicated in the cause of the disease. The sparsity has been used for a long time, for instance, the well known LASSO [428] (Least Absolute Shrinkage and Selection Operator) model integrates an L_1 penalty with a linear model. More details on the L_1 regularization and sparsity can be found in [184].

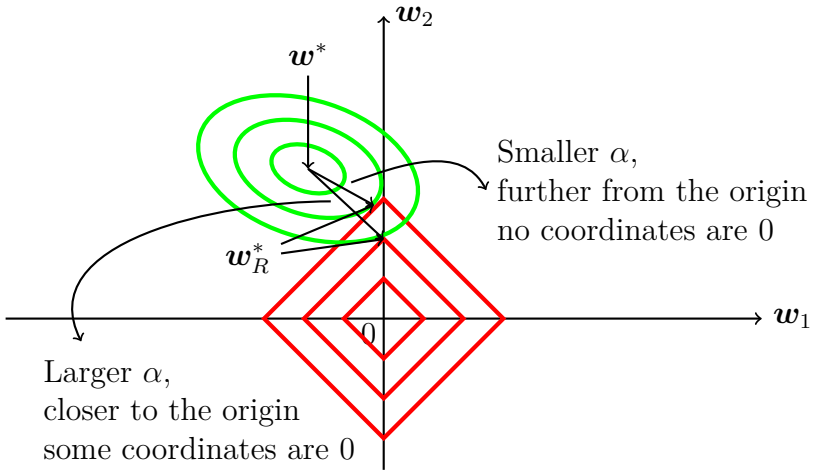


Fig. A.10 Effect of L_1 norm regularization: Large α makes some parameters equal to 0. (*red contours*): indicate the L_1 cost (L_1 norm of $\boldsymbol{w}$). (*green contours*): indicate the unregularized cost J. $\boldsymbol{w} = \boldsymbol{w}^*$ is the minimal solution of J. $\boldsymbol{w} = \boldsymbol{w}_R^*$ represent the optimum solutions of the regularized cost $\widetilde{J}$ (Eq.A.69). In this example, L_1 regularization allows two solutions $\boldsymbol{w}_R^*$ depending on the value of α. Small α results in a solution a little far from the origin. However, large α provides a sparse solution where $w_1 = 0$. (Reference: [103])

A.2.3.2 Early Stopping as a Regularization

We have mentioned in Sec.1.3.2.1 that early stopping can play a role of a regularizer. Formal demonstration is provided in this section.

Let us consider a network with weights initialized from a distribution with zero mean. We will see formally how early stopping can be a regularizer. Many authors [55, 397] argued that early stopping has the effect of restricting the optimization procedure to a relatively small volume of parameter space in the neighborhood of the initial parameter $\boldsymbol{\theta}_0$ (Fig.A.11). More specifically, consider taking τ optimization steps and with a learning rate ϵ. One can view $\epsilon\tau$ as the effective capacity. Restricting the number of iteration and the learning rate limits the volume of the parameter space reachable from $\boldsymbol{\theta}_0$. In this case, $\epsilon\tau$ behaves as if it was the reciprocal of the coefficient used for weight decay.

To compare early stopping with the classical L_2 regularization, let us consider a setting where parameters are linear weights $\boldsymbol{\theta} = \boldsymbol{w}$. One can approximate the objective function J with a quadratic form in the neighborhood of the empirically optimal value

of the weights $\boldsymbol{w}^*$ [154]

$$\hat{J}(\boldsymbol{w}) = J(\boldsymbol{w}^*) + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^*)^\top \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*) \,, \tag{A.75}$$

where $\boldsymbol{H}$ is the Hessian matrix of J with respect to $\boldsymbol{w}$ evaluated at $\boldsymbol{w}^*$. This makes $\boldsymbol{H}$ positive semidefinite. The gradient of the $\hat{J}$ is

$$\nabla_{\boldsymbol{w}} \hat{J}(\boldsymbol{w}) = \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*) \,. \tag{A.76}$$

Now, let us study the trajectory followed by the parameter vector during training. For simplicity, the initial parameters are set to the origin, $\boldsymbol{w}^{(0)} = \boldsymbol{0}$. Then, the gradient descent updates are performed as follows [154]

$$\boldsymbol{w}^{(\tau)} = \boldsymbol{w}^{(\tau-1)} - \epsilon \nabla_{\boldsymbol{w}} \hat{J}(\boldsymbol{w}^{(\tau-1)} \tag{A.77}$$

$$= \boldsymbol{w}^{(\tau-1)} - \epsilon \boldsymbol{H}(\boldsymbol{w}^{(\tau-1)} - \boldsymbol{w}^*) \tag{A.78}$$

$$\boldsymbol{w}^{(\tau)} - \boldsymbol{w}^* = (\boldsymbol{I} - \epsilon \boldsymbol{H})(\boldsymbol{w}^{(\tau-1)} - \boldsymbol{w}^*) \,. \tag{A.79}$$

Now, using the eigendecomposition of $\boldsymbol{H}$: $\boldsymbol{H} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top$, where $\boldsymbol{\Lambda}$ is a diagonal matrix and $\boldsymbol{Q}$ is an orthonormal basis of eigenvectors, it results [154]

$$\boldsymbol{w}^{(\tau)} - \boldsymbol{w}^* = (\boldsymbol{I} - \epsilon \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top)(\boldsymbol{w}^{(\tau-1)} - \boldsymbol{w}^*) \tag{A.80}$$

$$\boldsymbol{Q}(\boldsymbol{w}^{(\tau)} - \boldsymbol{w}^*) = (\boldsymbol{I} - \epsilon \boldsymbol{\Lambda})\boldsymbol{Q}^\top(\boldsymbol{w}^{(\tau-1)} - \boldsymbol{w}^*) \,. \tag{A.81}$$

Assuming that ϵ is chosen to be small enough to grantee $|1 - \epsilon\lambda_i| < 1$, it can be shown [55, 154] that the parameter trajectory during training after τ parameter updates has the following form

$$\boldsymbol{Q}^\top \boldsymbol{w}^{(\tau)} = \left[\boldsymbol{I} - (\boldsymbol{I} - \epsilon\Lambda)^\tau\right] \boldsymbol{Q}^\top \boldsymbol{w}^* \,. \tag{A.82}$$

Eq.A.61 can be rearranged as

$$\boldsymbol{Q}^\top \widetilde{\boldsymbol{w}} = \left[\boldsymbol{I} - (\Lambda + \alpha \boldsymbol{I})^{-1}\alpha\right] \boldsymbol{Q}^\top \boldsymbol{w}^* \,. \tag{A.83}$$

Comparing Eq.A.82 and Eq.A.83, one can see that if ϵ, α and τ are chosen such that [55, 154]

$$(\boldsymbol{I} - \epsilon\Lambda)^\tau = (\Lambda + \alpha \boldsymbol{I})^{-1}\alpha \,, \tag{A.84}$$

then L_2 regularization and early stopping can be seen as equivalent (under the quadratic approximation and the previous stated assumptions). Going further, by approximating both sides of Eq.A.84, one can conclude that if all λ_i are small, then [55, 154]

$$\tau \approx \frac{1}{\epsilon\alpha} \,, \tag{A.85}$$

$$\alpha \approx \frac{1}{\tau\epsilon} \,. \tag{A.86}$$

Thus, under these assumptions, the number of training iterations τ plays a role inversely proportional to the L_2 regularization parameter, and the inverse of $\tau\epsilon$ plays the role of the weight decay. Therefore, the parameter corresponding to the directions of

significant curvature of the objective function are regularized less than directions of less curvature. In the context of early stopping, this means that parameters that correspond to directions of significant curvature tend to learn early relatively to parameters corresponding to directions of less curvature. This is actually very intuitive. Parameters that correspond to high curvature tend to learn faster, which means, that in a short time, they have already learned something. While given the same amount of time, parameters that correspond to low curvature will tend to learn slowly. Therefore, early stopping mimics L_2 regularization by repressing parameters corresponding to low curvature and preventing parameter corresponding to high curvature to significantly grow. One can note that early stopping has the advantage to be determined through one run of the training process while L_2 regularization requires many runs with different values.

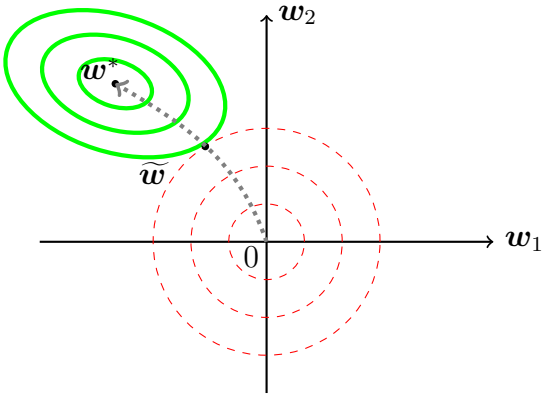


Fig. A.11 An illustration of the effect of early stopping. (*green contours*): indicate the contours the unregularized cost J (no early stopping). (*red dashed contours*): indicate the contours of the L_2 cost, which cause the minimum of the total cost to lie nearer the origin rather the minimum of the unregularized cost. (*gray dotted path*): indicates the trajectory taken by the SGD starting from the origin. Rather than stopping at the optimum point $\boldsymbol{w}^*$ that minimizes the cost, early stopping results in the trajectory stopping at an earlier point $\widetilde{\boldsymbol{w}}$. (Credit: [55, 154])