Unveiling the Transport Dynamics of Neural Networks : a Least Action Principle for Deep Learning

Ahmed Skander Karkar

Sorbonne Université - LIP6 & ISIR - EDITE de Paris

Thèse de doctorat

Unveiling the Transport Dynamics of Neural Networks: A Least Action Principle for Deep Learning

Skander Karkar

soutenue publiquement le 13 octobre 2023 devant le jury composé de

Mme Laetitia Chapel Professeure, Institut Agro Rennes-Angers	Rapportrice
M. Ievgen Redko Principal research scientist, Huawei	Rapporteur
M. Alasdair Newson Maître de conférences, Sorbonne Université	Examinateur
M. David Picard Senior research scientist, École des Ponts	Président
M. Patrick Gallinari Professeur, Sorbonne Université	Directeur
M. Alain Rakotomamonjy Principal researcher, Criteo	Directeur

Abstract

Residual connections are ubiquitous in deep learning, since besides residual networks [He et al., 2016b, He et al., 2016a] and their variants, they are also present in Transformers [Liu et al., 2021, Dosovitskiy et al., 2021]. The dynamic view of residual networks views them as analogous to a forward Euler scheme for an ordinary differential equation [Weinan, 2017]. We can then say that residual networks transport input points in space, time being represented by the depth of the network. This viewpoint has, for example, led to new architecture inspired by other numerical schemes for differential equations [Lu et al., 2018, Haber et al., 2019]. On the other hand, a bias of residual networks towards small perturbations of the input has been observed [Greff et al., 2016, Hauser, 2019, Jastrzebski et al., 2018, De and Smith, 2020, Chang et al., 2018b]. In the context of the dynamic view of residual networks mentioned above, this means a bias towards a small transport cost. In a first paper [Karkar et al., 2020], we experimentally verify that this bias is beneficial and should be encouraged and we show that forcing the network to approximate an optimal transport map by regularizing its transport cost improves its generalization ability and training stability. In a second paper [Karkar et al., 2023a], we show that applying this transport regularization to successive neural modules that don't back-propagate to each other amounts to following a gradient flow for minimizing the loss in distribution space, thus improving the performance of module-wise training, which consumes a lot less memory than end-to-end training. In a third paper [Karkar et al., 2023b], we propose a detector of adversarial and out-of-distribution samples that is based on the view of residual networks as discrete dynamical systems and show that transport regularization makes adversarial detection easier.

Contents

1 Introduction **1**
- 1.1 Context . . . 1
- 1.2 Motivation . . . 1
- 1.3 Contributions . . . 2
- 1.4 Outline . . . 3

2 Background **5**
- 2.1 Residual Networks . . . 5
- 2.2 Residual Block Architectures . . . 6
- 2.3 Interpretations of Residual Networks . . . 9
- 2.4 The Iterative Refinement View of Residual Networks . . . 10
- 2.5 Numerical Methods for Differential Equations . . . 11
- 2.6 Residual Networks as Discrete Dynamical Systems . . . 12
- 2.7 Neural Networks as Transport Systems . . . 14
- 2.8 Optimal Transport Theory . . . 15
- 2.9 Optimal Transport for the Study of Neural Networks . . . 18

3 A Least Action Principle for the Training of Neural Networks **21**
- 3.1 Introduction . . . 21
- 3.2 Empirical Analysis of Transport Dynamics in Residual Networks 22
  - 3.2.1 Transport Cost and Generalization in Image Classification 22
  - 3.2.2 Visualizing Network Dynamics in 2 Dimensions . . . 23
- 3.3 Method and Theory . . . 25
  - 3.3.1 General Setting . . . 25
  - 3.3.2 Formulation . . . 26
  - 3.3.3 Regularity . . . 28
- 3.4 Practical Implementation . . . 29
- 3.5 Experiments . . . 30
  - 3.5.1 Generalization . . . 30
  - 3.5.2 Transport Visualization . . . 32
  - 3.5.3 Stability . . . 32
  - 3.5.4 Training Dynamics . . . 33
- 3.6 Discussion and Conclusion . . . 34

4 Module-wise Training via the Minimizing Movement Scheme **37**
- 4.1 Introduction . . . 37
- 4.2 Related Work . . . 40
- 4.3 Method and Theory . . . 42
  - 4.3.1 Formulation . . . 42
  - 4.3.2 Background on the Minimizing Movement Scheme . . . 43
  - 4.3.3 Link with the Minimizing Movement Scheme . . . 44
  - 4.3.4 Regularity Result . . . 45
- 4.4 Practical implementation . . . 46
  - 4.4.1 Multi-block Modules . . . 46

4.4.2 Solving the Module-wise Problems . . . 47
4.4.3 Varying the Regularization Weight . . . 47
4.5 Experiments . . . 48
4.5.1 Parallel Module-wise Training with Few Modules . . . 49
4.5.2 Ablation Study and Sensitivity to Hyperparameter τ . . . 51
4.5.3 Memory Usage of Parallel TRGL . . . 52
4.5.4 Training Time of Parallel TRGL . . . 53
4.5.5 Sequential Full Block-wise Training . . . 53
4.5.6 Avoiding Early Overfitting . . . 55
4.6 Discussion and Conclusion . . . 56

5 Adversarial Detection Through Transport Dynamics 57
5.1 Introduction . . . 57
5.2 Related Work . . . 58
5.3 Method and Theory . . . 62
5.3.1 Detection . . . 62
5.3.2 Regularization . . . 63
5.4 Experiments . . . 67
5.4.1 Preliminary Experiments . . . 68
5.4.2 Detection of Seen Attacks . . . 70
5.4.3 Detection of Unseen Attacks . . . 72
5.4.4 Detection of Out-of-Distribution Samples . . . 73
5.4.5 Attacking the Detector . . . 75
5.4.6 Time Comparison . . . 76
5.5 Discussion and Conclusion . . . 76

6 Conclusion 79
6.1 Summary . . . 79
6.2 Future Directions . . . 79
6.2.1 Adaptive Depth During Training . . . 79
6.2.2 Similarity of Neural Network Representations . . . 82
6.2.3 Optimal Transport between Different Dimensions . . . 82
6.2.4 Simulating Continuity Partial Differential Equations . . . 82
6.3 Conclusion . . . 83

References 85

List of Acronyms 111

List of Tables 115

List of Figures 117

A Background on Numerical Methods for Differential Equations 119
A.1 The Euler Method . . . 119
A.2 Runge-Kutta Methods . . . 120
A.3 Adaptive Runge-Kutta Methods . . . 122

B Background on Optimal Transport Theory **125**
B.1 Monge and Kantorovich Problems . . . 125
B.2 Duality . . . 128
B.3 The Wasserstein Space . . . 129
B.4 The Dynamic Formulation of Optimal Transport . . . 133
B.5 Regularity of Optimal Transport Maps . . . 136

C Background on Gradient Flows **139**
C.1 Gradient Flows in Euclidean Space . . . 139
C.2 Gradient Flows in Metric Spaces . . . 140
C.3 Gradient Flows in the Wasserstein Space . . . 141

D Background on Module-wise Training **145**
D.1 Additional Loss Terms . . . 145
D.2 Architectural Approaches . . . 146

E Background on Adversarial Attacks **149**
E.1 Adversarial Attacks . . . 149
E.1.1 White-box Attacks . . . 149
E.1.2 Black-box Attacks . . . 150
E.2 Adversarial Detectors . . . 151
E.2.1 Statistical Approaches . . . 151
E.2.2 Computer Vision Approaches . . . 152

F Additional Experiments from Section 3 **155**
F.1 Implementation Details . . . 155
F.2 Transport Visualization . . . 155
F.3 Experiments with Fixed λ . . . 159
F.4 Impact on Weight Distribution of Classifier . . . 160
F.5 Additional Experiments in 2 Dimensions . . . 161
F.5.1 Comparison to Exact Wasserstein Distances . . . 161
F.5.2 Comparison to Batch Normalization . . . 161
F.5.3 Additional Transport Visualizations in 2 Dimensions . . . 163

G Additional Experiments from Section 4 **169**
G.1 Implementation Details . . . 169
G.2 Additional Experiments . . . 169
G.3 Memory Usage . . . 172
G.4 Sensitivity to Hyperparameter τ . . . 173

H Additional Experiments from Section 5 **175**
H.1 Implementation Details . . . 175
H.2 Detection of Seen Attacks . . . 175
H.3 Detection of Unseen Attacks . . . 176
H.4 Detection Rate of Successful Adversarial Samples . . . 177
H.5 False Positive Rate . . . 178

H.6 AUROC 179
H.7 Detection of Out-of-Distribution Samples 179

1 Introduction

1.1 Context

Machine learning is the prevailing current approach in the field of artificial intelligence. It focuses on designing systems that automatically learn from observed data [Russell and Norvig, 2020]. In the past twenty years, deep learning, i.e. the use of deep artificial neural networks, has emerged as a crucial advancement in machine learning [Goodfellow et al., 2016]. It has become the state-of-the-art in nearly all tasks, from classification to generation, in the fields of computer vision [Krizhevsky et al., 2012], natural language [Wu et al., 2016], sound [Purwins et al., 2019], speech [Mehrish et al., 2023] and video [Sharma et al., 2021] processing, and has been applied successfully to other tasks such as dynamical system forecasting [de Bezenac et al., 2018], biology [Jumper et al., 2021] and reinforcement learning [Silver et al., 2016].

In all these applications, the neural network architectures used rely on a set of heuristics, techniques and components that are omnipresent. These components, and especially the interactions between them, are not yet fully understood theoretically. They include back-propagation [Rumelhart et al., 1986] and stochastic gradient descent [Bottou, 2012, Amari, 1967], or a variant of it [Bottou et al., 2018], for training the model, non-linear activations (ReLU [Fukushima, 1975] very often), normalization layers (most often batch normalization [Ioffe and Szegedy, 2015] or layer normalization [Ba et al., 2016]) and the importance of a good initialization [Saxe et al., 2014, He et al., 2015]. Among these ever-present elements is the architectural component known as a skip (or residual) connection [He et al., 2016b, He et al., 2016a].

Indeed, residual networks and their variants are a very popular neural network architecture, especially in computer vision. Their main building block, the residual connection, has become ubiquitous in state-of-the-art architectures such as Transformers used in natural language processing.

1.2 Motivation

Many works have tried to explain the success of residual networks through both empirical and theoretical explorations, and to propose new improvements (architectures, initializations, training procedures...) suggested by these explanations.

A point that stands out is that it is precisely their initialization as a perturbation of the identity function that biases residual networks towards finding good solutions to learning problems that only minimally move the inputs.

We position ourselves in this field of research that tries to explain the importance of skip connections and then improve the performances of networks that use them.

1.3 Contributions

The particular point of view of the functioning of residual networks that we consider is that of discrete dynamical systems. Indeed, residual networks can be viewed as a discrete Euler scheme for a differential equation. We further link this to the theory of optimal transport in its dynamical formulation. This allows us to see residual networks as discrete transport systems that move their input points through space in time, time being represented by the depth, to disentangle them before they are classified by a linear classifier.

Through this understanding, we propose a regularization that is particularly well-adapted to residual networks and that improves their training stability and generalization ability. Indeed, from this viewpoint, the bias towards small residuals means a bias towards a small transport cost. Since we verify that this bias is beneficial, a natural idea that follows is to regularize residual networks during training by penalizing their transport cost, which is the distance by which the input points travel through the residual blocks, thereby reinforcing the desirable bias towards remaining close to the identity function. This regularization forces the solution network to be an optimal transport map in the sense of optimal transport theory, thus giving it some regularity. In practice, this is showcased by increased stability (less variance in the results and the ability to train without batch normalization for example) and better accuracy at test time.

This transport regularization has other interesting properties that we exploit for two applications: module-wise training and adversarial detection.

Firstly, the quantity we regularize is in fact a distance between probability distributions (the Wasserstein distance). Therefore, this regularization allows to split neural networks into modules and to train them module-wise, i.e. without back-propagation between the modules, in a theoretically sound way and with good performances. This type of training is interesting in practice in constrained settings as it does without many of the back-propagated gradients of end-to-end back-propagation, thus consuming less memory. The transport regularization keeps the early modules from overfitting and makes the modules take small proximal steps to minimize the loss in the Wasserstein metric. This allows the modules to build upon each other in accuracy without back-propagation between them, which is exactly the property desired in module-wise training.

Secondly, The transport regularization also has the particularity of having a value for each input image. The transport statistics therefore allow us to study the behavior of the network on a particular image. We can therefore use these transport statistics that describe the trajectory of the input in space for the purpose of detecting adversarial attacks. We propose a novel detector of adversarial and out-of-distribution samples that tells clean inputs from abnormal ones by comparing the discrete vector fields they follow throughout the network's layers before the final classification layer. We also show that minimizing the transport cost gives the network more regularity only on the support of the clean data distribution. This makes the network's activations on clean samples

more distinguishable from those on abnormal samples, and thus improves the performance of adversarial detection methods that use the internal embeddings as inputs.

1.4 Outline

In Section 2 we present the related work and necessary background on residual networks, their dynamic interpretation and the advances that followed from it, numerical methods for ordinary differential equations and optimal transport theory.

In Section 3, we present the work done in [Karkar et al., 2020]. We first verify experimentally that the bias of residual networks towards minimal transformations is beneficial, and that it means a bias of residual networks towards being good approximators of optimal transport maps. We then formalize the idea of encouraging small transport cost by linking this bias to optimal transport theory in its dynamic formulation and implement it as a transport regularization. This means that we seek to retrieve a network that solves the task at hand while moving the input points as little as possible. We theoretically prove that this leads to regular networks. Experimentally, our method improves the generalization of residual-type networks, especially in small data and overfitting regimes. It also improves training stability, allowing for example to train deep networks without batch normalization.

In Section 4, we present the work done in [Karkar et al., 2023a], where we show that applying this transport regularization to successive neural modules that don't back-propagate to each other amounts to following a gradient flow for minimizing the loss in distribution space. This offers a new formulation and method for greedily training neural networks module-wise, i.e. splitting the network into modules and training these modules without back-propagation between them. This requires less memory than standard end-to-end training and has therefore been used in constrained settings such as training on mobile devices. Experimentally, regularizing the transport cost of each module leads to better performances of these networks by avoiding the problem of stagnation or collapse in accuracy along the depth observed in greedily-trained modules, whereby earlier modules overfit and deeper ones stop improving the test accuracy or even degrade it.

In Section 5, we present the work done in [Karkar et al., 2023b], where we propose a detector of adversarial and out-of-distribution samples that uses the spatiotemporal dynamics of the inputs throughout the network to detect abnormal inputs. We also show that the increased regularity guarantee given by the transport regularization only on the support of the data makes detection of adversarial attacks, that tend to lie outside the data manifold, easier.

In Section 6, we conclude and discuss future avenues of research. The main idea for future work is to extend the analogy between neural networks and numerical schemes for differential equations to include adaptive schemes. Indeed,

architectures that imitate schemes that are of higher order than the Euler method have been proposed, but the possibilities and advantages of including adaptive methods, which rely on imbedded schemes of different orders, have not been sufficiently explored. These possibilities include ensembling different forward passes at test time to improve performance, and adaptively adjusting the depth of the network during training to the complexity of the task at hand.

2 Background

We present in this section the background and notions that are necessary to all the following sections. The notions that are specific only to a particular section are presented in it, and more detailed background can be found in the Appendix. Here, we define residual networks, stress their importance in modern deep learning and present some interpretations of their functioning, focusing on the iterative refinement and dynamic interpretations. We then discuss recent advances and ideas that were inspired by these interpretations. Finally, we present the essential notions of optimal transport theory that we will use later.

2.1 Residual Networks

Residual networks (ResNets) [He et al., 2016b, He et al., 2016a] and their variants such as ResNeXt [Xie et al., 2017] and WideResNet [Zagoruyko and Komodakis, 2016] have become an important architecture in deep learning, especially for computer vision. They allowed for training very deep models: up to 1000 layers, while training compositional (i.e. non-residual) networks deeper than 20 layers remains challenging [Jastrzebski et al., 2018, Balduzzi et al., 2017]. Residual networks and their variants reach state-of-the-art performances on many computer vision tasks such as image classification [Xie et al., 2017, Mahajan et al., 2018, Wightman et al., 2021], unsupervised domain translation [Zhu et al., 2017], image segmentation [Pohlen et al., 2017, Chen et al., 2017a] and action recognition [Tran et al., 2018]. They have also been used with success in speech and natural language processing applications [et al., 2016, Xiong et al., 2017, van den Oord et al., 2016].

Furthermore, residual connections are a basic building block in many state-of-the-art architectures such as EfficientNet [Tan and Le, 2019] and MobileNetV2 [Sandler et al., 2018], which use a so-called inverted residual block. Residual connections have also been added to prior architectures such as Inception to improve their performances [Szegedy et al., 2017], and are used in some U-Net architectures [Janner et al., 2022]. If we call a *residual network* any neural network whose architecture is made up of successive residual blocks (preceded by an initial encoding, separated by pooling layers and followed by a classification head), then vision Transformers such as Swin [Liu et al., 2021] and ViT [Dosovitskiy et al., 2021], video Transformers such as TimeSFormer [Bertasius et al., 2021], and natural language processing Transformers such as GPT [Radford et al., 2018] can also be considered residual networks, with attention and multilayer perceptrons instead of convolutions in their residual functions.

A *residual block* (ResBlock) is an architectural component in a neural network that has as input x (an embedding at a certain depth) and outputs $x + r(x)$. We will call the function r associated with a residual block its *residual function* (or *forcing function*), and $r(x)$ the *residue*. Consecutive residual blocks with

residual functions r_m applied to an input x_0 can then be written

$$x_{m+1} = x_m + r_m(x_m) \tag{1}$$

The addition of x to $r(x)$ is called a *skip connection* or a *residual connection*. More generally, a skip connection saves the value of the input to some layer and uses it later in the network. The residual point of view is different from the traditional one where the data is transformed compositionally by applying $f_1 \circ f_2 \circ f_3 \circ ... f_m$ to the input. In a residual architecture, the input is transformed in an additive manner by adding residues to it.

This rewriting of course does not change the representation power of the network (i.e. the functions it is able to approximate). Networks with layers that apply $x_{m+1}{=}f_m(x_m)$ where the input and output of f_m have the same dimension can be reformulated as residual blocks by taking as residual functions $r_m(x_m){=}f_m(x_m){-}x_m$ so that $x_{m+1}{=}x_m{+}r_m(x_m)$. The theory and implementation developed apply therefore directly to networks with stages made up of such layers, which includes for examples VGG networks [Simonyan and Zisserman, 2014]. The particularity of residual blocks in residual networks and Transformers is then that they are initialized to functions of the form $\mathrm{id} + r$, which confers certain biases to the training procedure and to the solution that will be found. It is these biases that we aim to uncover and exploit.

2.2 Residual Block Architectures

Typically the residual function of a residual block used for a vision task contains two or three convolutions with activations and batch normalisations. The *basic residual block* architecture is in Figure 1 below.

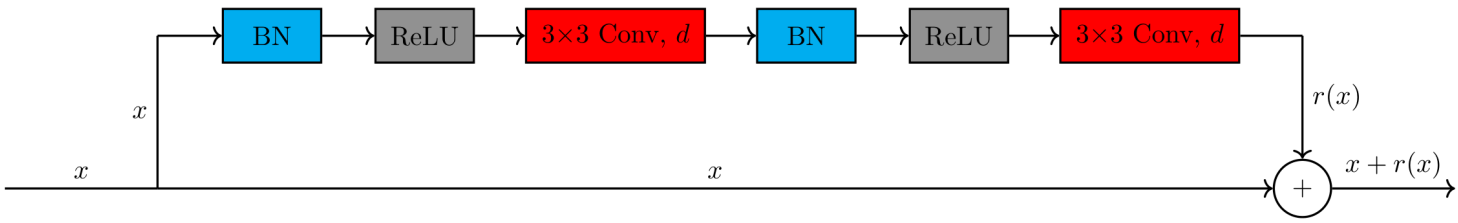

Figure 1: Basic residual block architecture proposed in [He et al., 2016a]. BN: batch normalization, ReLU: $x \mapsto (x)_+$. Conv: convolution. d: number of channels.

The other common residual block architecture is called the *bottleneck residual block* [He et al., 2016b, He et al., 2016a]. While the convolutions in the basic block keep the same number of channels, the bottleneck block has three convolutions, the first of which (a 1×1 convolution) reduces the number of channels. The

third convolution (also 1×1) restores it to that of the block's input. This structure, applied across many paths, is present in the residual block used in the ResNeXt architecture. The bottleneck block and the ResNeXt block are illustrated in Figure 2 below.

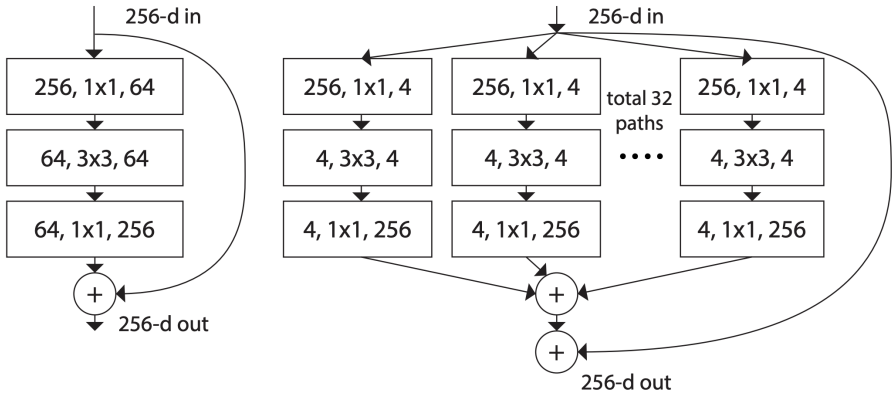

Figure 2: Bottleneck residual block (left) and ResNeXt block (right) architectures. From [Xie et al., 2017]. A convolutional layer is shown as (number of input channels, filter size, number of output channels).

Finally, we mention the *inverted residual block*. It was proposed initially for the MobileNetV2 [Sandler et al., 2018] architecture for efficiency reasons and has been used since for mobile-optimized networks. It uses an inverse structure to the bottleneck block, as the input and output now have few channels, and the first 1×1 convolution increases the number of channels before the third convolution reduces it back to that of the input. It also uses a depth-wise convolution for the second 3×3 convolution, which reduces the number of parameters. This architecture is compared to the bottleneck architecture in Figure 3 below.

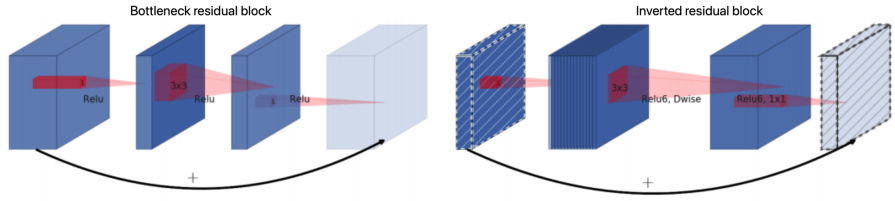

Figure 3: Bottleneck residual block (left) and inverted residual block (right) architectures. From [Sandler et al., 2018]. The thickness indicates the number of channels.

Residual blocks can be separated by layers that change the dimensions of the embedding (for example pooling layers and patch merging layers). If the same embedding dimension is kept, we call the network a *single representation* ResNet.

If the embedding dimension changes as in the original ResNet of [He et al., 2016a], we call each group of residual blocks that keep the same dimension a *residual stage*. In Figure 4, we represent a residual stage containing three consecutive basic residual blocks.

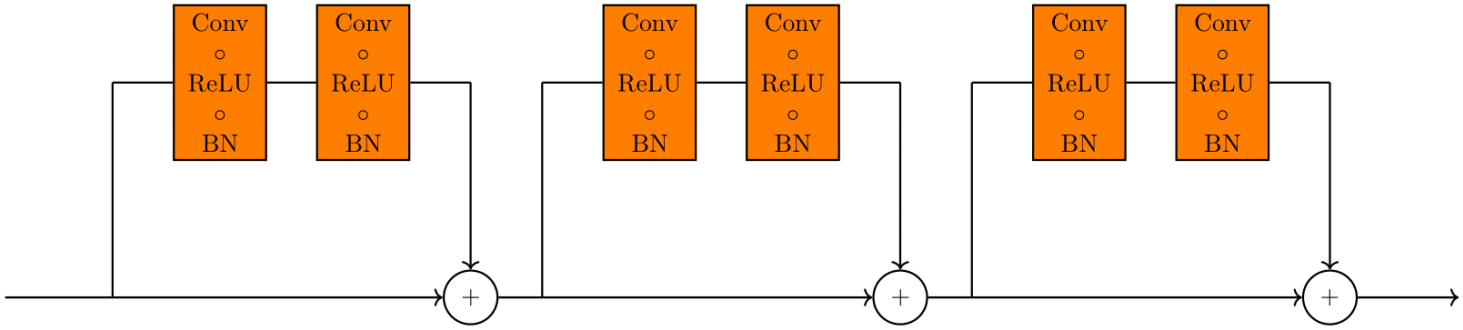

Figure 4: Three consecutive residual blocks.

Below in Figure 5 is the Swin Transformer architecture with 4 residual stages separated by patch merging. We see that the typical *Transformer block* is made up of two residual blocks, the first containing self-attention and the second containing a multilayer perceptron. The Transformer block in ViT (Vision Transformer, [Dosovitskiy et al., 2021]) is very similar.

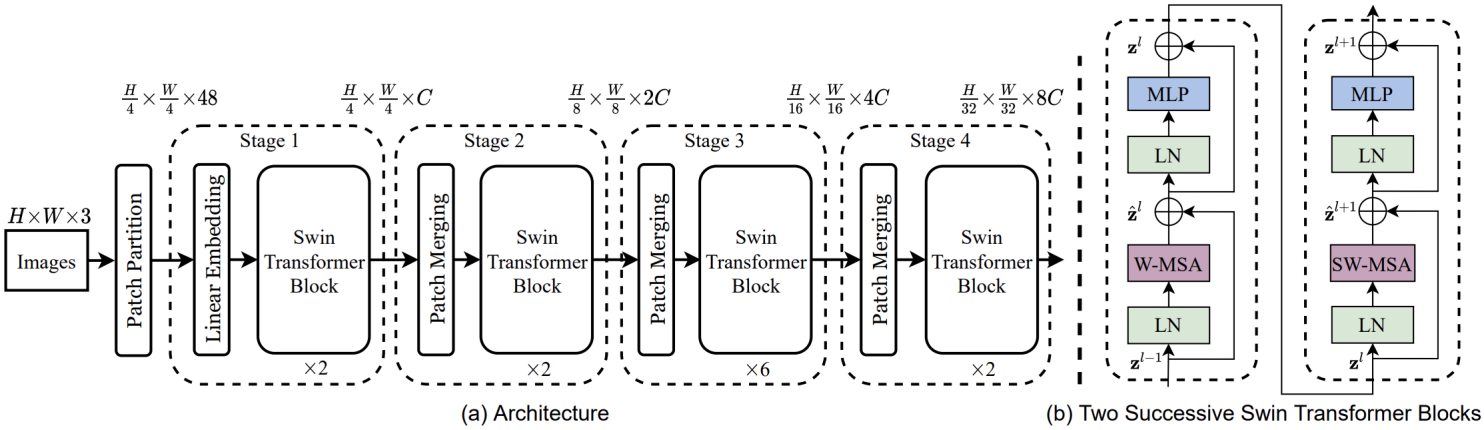

Figure 5: Swin Transformer architecture with 4 residual stages from [Liu et al., 2021]. MLP: multi-layer perception. LN: layer normalization. W-MSA: window multi-head self-attention. SW-MSA: shifted window multi-head self-attention.

2.3 Interpretations of Residual Networks

The initial intuition behind the design of the ResBlock in [He et al., 2016b] is that adding more layers to a network should not decrease its performance, as the deeper network contains the shallower one by simply adding identity mappings.

However, increased depth in compositional networks is observed to cause a decrease in training accuracy. The residual block architecture was proposed as a way of forcing deep networks to only learn perturbations from identity, as learning the zero function is easier than learning the identity function. This also helped with the problems of vanishing and exploding gradients (exacerbated in deeper networks as gradients multiply).

Many perspectives try to further explain how residual connections work and why they perform so well. Once an interesting explanation is found, it is often possible to deduce from it new methods, architectures and improvements to residual networks and to neural networks in general.

In [Balduzzi et al., 2017], the authors identify the problem of *shattered gradients*. They show, experimentally and theoretically, that correlations between gradients of deeper layers at initialization decay exponentially with depth in compositional networks, making them resemble white noise. More precisely, they consider the gradient of the network's output with respect to a fixed neuron's output for different inputs. Because of the chain rule, these gradients are strongly linked (via terms that are independent of the network's architecture) to the gradients with respect to the weights computed during optimization. This decay in correlations between gradients of deeper layers makes training difficult as it leads to averaging white-noise-like gradients across minibatches, hinders optimization algorithms that assume gradients at nearby points are similar (e.g. momentum-based and accelerated methods) and causes the effect of a neuron on the output to become increasingly unstable as the depth increases. The authors show that, in contrast, gradients in residual networks behave more like brown noise and that the correlations between them decrease sublinearly with depth (when batch normalization is used). From this, they propose a new initialization that makes training deep non-residual networks easier.

In the ensemble view [Veit et al., 2016, Huang et al., 2018], residual networks are thought to learn an exponential ensemble of shallower models. For a residual network of depth 3, the output x_3 can indeed be written

$$x_3 = x_0 + r_0(x_0) + r_1(x_0 + r_0(x_0)) + r_2(x_0 + r_0(x_0) + r_1(x_0 + r_0(x_0)))$$

The input x_0 then has many paths to reach the final classification layer. If this classification layer is linear, then its output can be seen as the sum of its outputs on many different shallower learned representations of the input x_0, starting from x_0 itself to the deepest representation $r_2(x_0 + r_0(x_0) + r_1(x_0 + r_0(x_0)))$. [Huang et al., 2018] combine this point of view with boosting theory [Freund and Schapire, 1997] to propose an algorithm for sequentially training residual networks, as we do in Section 4.

2.4 The Iterative Refinement View of Residual Networks

A view of residual network that is of use to us is that of unrolled iterative estimation, where residual blocks are thought to iteratively refine representations instead of learning new ones [Greff et al., 2016, Liao and Poggio, 2016, Hauser, 2019, De and Smith, 2020]. This view essentially verifies that residual blocks, especially deeper ones, do remain close to the identity function. Paper [Jastrzebski et al., 2018] explores this view further, calling it *iterative refinement* (or *iterative inference*). The authors rewrite the loss L of a residual network with M residual blocks by applying recursive Taylor expansions:

$$\begin{aligned} L(x_M) &= L(x_{M-1} + r_{M-1}(x_{M-1})) \\ &= L(x_{M-1}) + \nabla L(x_{M-1}) \cdot r_{M-1}(x_{M-1}) + \mathcal{O}(r^2_{M-1}(x_{M-1})) \\ &= L(x_m) + \sum_{m \leq i < M} \nabla L(x_i) \cdot r_i(x_i) + \mathcal{O}(r_i^2(x_i)) \end{aligned}$$

which holds for any $0 \leq m < M$. This is a good approximation if the magnitude of the residual functions r_i is small. Thus, for these blocks, the training can be thought of as in part minimizing the terms $\nabla L(x_i) \cdot r_i(x_i)$, which can be done by making $r_i(x_i)$ point in the opposite half-space to $\nabla L(x_i)$. The block operations $x_i + r_i(x_i)$ are therefore moving the input towards the half-space of $-\nabla L(x_i)$. A residual block is then said to perform iterative refinement or inference if, on average on the dataset, its residual function r_i has a small magnitude and satisfies $\nabla L(x_i) \cdot r_i(x_i) < 0$. A residual block whose residual function has a high magnitude is said to learn a new representation of the data, which is slowly refined by later blocks performing iterative inference.

It is observed in [Greff et al., 2016, Hauser, 2019, Jastrzebski et al., 2018, Chang et al., 2018b] that only few early residual blocks in each residual stage learn a new representation and that the deeper blocks in a residual stage perform iterative refinement according to the definition above. Figure 6 shows the ratio $\frac{\|r_i(x_i)\|}{\|x_i\|}$ and the cosine loss $\frac{\nabla \mathcal{L}(x_i) \cdot r_i(x_i)}{\|\nabla \mathcal{L}(x_i)\| \|r_i(x_i)\|}$ on CIFAR10 of 4 ResNets.

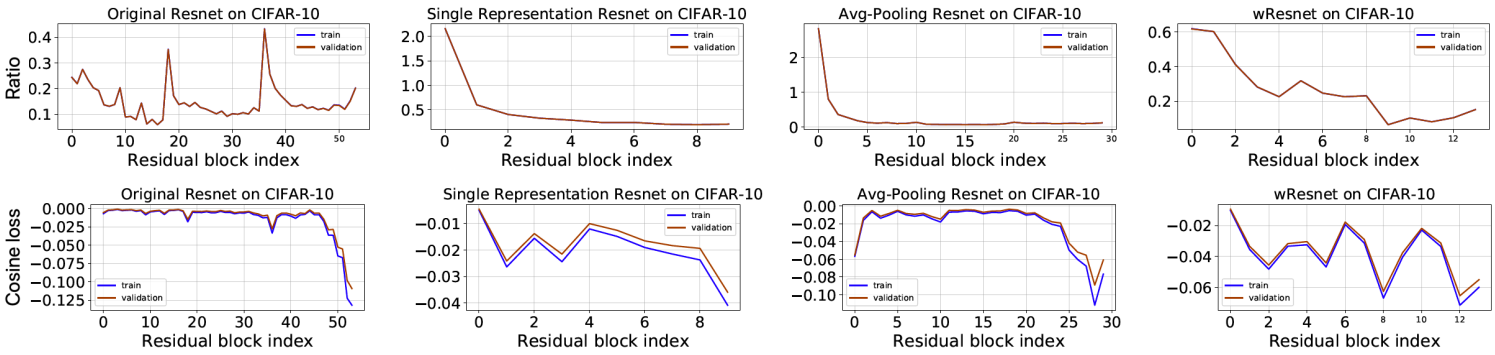

Figure 6: Ratio of residue norm to input norm (first row) and cosine loss (second row) of 4 residual architectures on CIFAR10 from [Jastrzebski et al., 2018].

We see that the peaks in the ratio come in fact in the early residual blocks of a residual stage, especially on the first architecture on the left. This means that residual blocks are indeed biased towards minimally modifying their input. Furthermore, [Zhang and Wynter, 2018, Zhang et al., 2019] add a fixed positive weight h to the residual function, i.e. consider residual blocks of the form $x + h\ r(x)$, and find that taking a small positive $h < 1$ makes residual networks more stable. This indicates that making the residual blocks even closer to identity is beneficial.

In [Jastrzebski et al., 2018], two suggested modifications to residual networks follow from this finding that later residual blocks all perform similar iterative refinement. First, they propose to unroll the last block at inference time (i.e. apply it many times) to improve the accuracy of the model on borderline inputs, that is, inputs that are very close to the decision boundary. Secondly, they propose to make the later residual blocks share the same parameters to reduce memory footprint.

The fact that residual blocks only minimally build upon each other instead of learning new representations also explains why stochastic depth training [Huang et al., 2016, Hayou and Ayed, 2021], i.e dropping a random subset of layers at each training epoch and replacing them with the identity function, is particularly effective on residual networks. It increases their test accuracy as it functions as a regularization [Hayou and Ayed, 2021], and of courses reduces training time. In [Huang et al., 2016], stochastic depth training allowed for training residual networks that have more than 1200 layers on CIFAR10 while still improving test accuracy, i.e without overfitting.

Finally, [De and Smith, 2020] also find that the bias towards small residuals is beneficial and allows one to successfully train deeper networks. They further show that batch normalization encourages this bias by downscaling the variance of the residue at initialization compared to the variance of the identity branch of the residual block. From this they develop an initialization that allows to train deep residual networks without batch normalization. Our transport regularization in Section 3 also allows to train deep residual networks without batch normalization.

2.5 Numerical Methods for Differential Equations

Before introducing the dynamic interpretation of residual networks, we need to introduce numerical schemes for ordinary differential equations (ODEs). Given a time-dependent vector field $v_t : \mathbb{R}^d \to \mathbb{R}^d$, consider the *initial value problem* of finding a function $x : I \to \mathbb{R}^d$ that satisfies

$$x'(t) = v_t(x(t)) \tag{2}$$

with initial condition $x(t_0)=x_0$, where I is the interval $[t_0, t_0+T]$.

Equivalently, the integral form of this problem is finding x that satisfies

$$x(t) = x_0 + \int_{t_0}^{t} v_t(x(t))\,\mathrm{d}t \tag{3}$$

Finding closed-formed solutions to this problem can be difficult. We wish then to approximate the values x takes at certain time points. Given a subdivision $t_0{<}t_1{<}..{<}t_N{=}t_0{+}T$ of I, we denote the *step-sizes* $h_n{:=}t_{n+1}{-}t_n$ for $0{\leq}n{<}N$ and define $h_{\max} := \max h_n$. The *explicit Euler method* is a numerical approximation method that approximates the values $(x(t_n))_n$ through the values $(x_n)_n$ given by

$$x_{n+1} = x_n + h_n v_{t_n}(x_n) \tag{4}$$

starting from the initial condition x_0. This method simply comes from a first-order Taylor expansion of x, or from a left-hand rectangle method for approximating the integral in (3). The right-hand rectangle method for approximating the integral in (3) leads to the *implicit Euler method*:

$$x_{n+1} = x_n + h_n v_{t_{n+1}}(x_{n+1}) \tag{5}$$

A fixed-point method [Shashkin, 1991] or the Newton-Raphson method [Ypma, 1995] can be used to solve (5) for x_{n+1}.

A numerical method *converges* if

$$\max_n \|x(t_n){-}x_n\| \xrightarrow[h_{\max}\to 0]{} 0$$

where $\max_n \|x(t_n){-}x_n\|$ is the *global error* of the method. Euler's method converges if v is $\mathcal{C}^0$ and Lipschitz in x. If v is also $\mathcal{C}^1$ then it converges with speed $O(h_{\max})$. See Appendix A.1 for more details.

2.6 Residual Networks as Discrete Dynamical Systems

The view we consider is the dynamic view. It views residual network as an Euler discretization of a differential equation on the interval $I = [0, 1]$.

$$x_{m+1} = x_m + r_m(x_m) \iff x'(t) = v_t(x(t)) \tag{6}$$

where r_m approximates the velocity field v_t at time $t = t_m$ and we suppose that the time-steps are learned inside r_m. This observation was first made in [Weinan, 2017] and allows to draw on the rich theory of differential equations and on the numerical methods developed to simulate them. New network architectures were designed by considering other numerical schemes for the approximation of differential equations such as linear multi-step methods [Lu et al., 2018], implicit-explicit methods [Haber et al., 2019], and Runge-Kutta methods [Zhu et al., 2022]. We present some of these architectures below.

The IMEX architecture [Haber et al., 2019] considers a parametrized differential equation $x'(t) = v(x(t); \theta(t))$ where v is a neural network parametrized by θ

and discretizes it using the implicit-explicit numerical method [Ascher et al., 1995, Ascher et al., 1997, Ascher and Petzold, 1998]. This method rewrites the differential equation as follows $x'(t) = v(x(t); \theta(t)) + Ax(t) - Ax(t)$ where A is a matrix to be chosen or learned. The method then uses an explicit Euler scheme for the first part $v(x(t); \theta(t)) + Ax(t)$ and an implicit Euler scheme for the second part $Ax(t)$. This leads to the following numerical step and block architecture

$$x_{n+1} = (\mathtt{Id} + hA)^{-1}(x_n + hAx_n + hv(x_n, \theta_n)) \tag{7}$$

The matrix A is a trainable group convolution and the left-multiplication by an inverse matrix is done in the Fourier domain. According to [Haber et al., 2019], this architecture has two advantages in theory: the increased stability of implicit methods, and the global field of view connecting all pixels thanks to the multiplication by $(\mathtt{Id} + hA)^{-1}$, since this matrix is dense. Experiments on tasks where the field of view is important such as image segmentation [Minaee et al., 2022] and depth estimation [Masoumian et al., 2022] confirm this second advantage over ResNets.

[Lu et al., 2018] build architectures called LM-ResNet and LM-ResNeXt that mimic linear multi-step numerical schemes [Dahlquist, 1963], while FractalNet [Larsson et al., 2017] and RKCNN [Zhu et al., 2022] mimic Runge-Kutta numerical methods (see Appendix A.2 for a definition of these methods). PolyNet [Zhang et al., 2017b] use a polynomial approximation to the implicit Euler method (5) to design their architecture. The increased stability of implicit methods allows them to increase the step-size and therefore to reduce the depth of the network.

Forward passes that imitate partial differential equations were also proposed in [Ruthotto and Haber, 2018]. The stability of the forward pass was also studied through the lens of dynamical systems, leading to considering particular forms of differential equations to imitate such as Hamiltonian systems [Haber and Ruthotto, 2017], and to enforcing particular conditions on the weight matrices [Ciccone et al., 2018].

Reversibility, which means that the dynamics of a differential equation can also be simulated backwards in time, is an important notion in the study of dynamic systems. The dynamic view was therefore linked to the possible reversibility of residual networks in [Chang et al., 2018a]. The possibility of making residual networks reversible is one of their advantages. The reversible residual network called RevNet designed in [Gomez et al., 2017] allows to reconstruct the activations of a layer from those of the next layer. It is then possible to perform back-propagation without storing the activations in memory, except for those of a handful of non-reversible layers. These residual networks therefore have an activation storage complexity that is independent of depth and an order of magnitude lower than that of non-reversible networks of equal size. Reversible networks called i-RevNets [Jacobsen et al., 2018] are an extension of these RevNets.

Some of the architectures discussed above are illustrated in Figure 7 below.

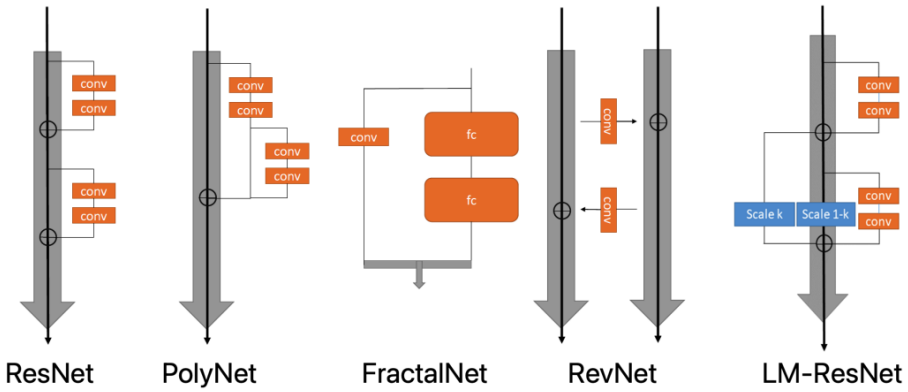

Figure 7: Some architectures inspired by numerical schemes. From [Lu et al., 2018].

2.7 Neural Networks as Transport Systems

The dynamic view also allows one to consider that neural networks are transporting their inputs in space, with depth representing time, before a linear classifier is applied (if the task at hand is classification). Here, we discuss other works that share this point of view.

NeuralODE [Chen et al., 2018, Ruiz-Balet and Zuazua, 2021, Yan et al., 2020] take the continuous limit of a residual network and parameterize and learn the velocity field of an ordinary differential equation that is then numerically integrated to get the output. More precisely, since a residual network is analogous to an Euler scheme, taking its limit as the step size goes to zero suggests directly learning the continuous velocity field v_t that moves the input points in space so that they land in a configuration that solves the task. So if L is the loss, and x_0 is the input, we look for a vector field v_t (in practice a neural network parametrized by θ) such that the initial value problem $x'(t) = v_t(x_t)$ on the interval $[0, 1]$ with initial condition $x(0) = x_0$ has a solution curve x that ends up at a value $x(1)$ that has a small loss. So we minimize (through gradient descent) in θ

$$L(x(1)) = L\left(x_0 + \int_0^1 v(t, x(t); \theta)\,\mathrm{d}t\right) \approx L(\texttt{Solver}(v(.,.;\theta), x_0)) \tag{8}$$

where `Solver` applies a numerical solver such as Euler to the initial value problem and returns an approximation of $x(1)$. If the task is classification, a linear classifier is applied to $x(1)$ before the loss is computed and an embedding might precede going through the numerical solver. The classifier and encoder are then jointly learned with the dynamics network v.

Differentiating the loss (8) in θ can be done in two ways. The first is to discretize then differentiate, i.e. to differentiate $L(\texttt{Solver}(v(.,.;\theta), x_0))$ with respect to

θ by simply using automatic differentiation to back-propagate through the operations of the numerical solver (which are all differentiable). The second is to differentiate then discretize, i.e. to use the so-called *adjoint state method*, which can be thought of as an instantaneous analog of the chain rule, to differentiate $L(x_0 + \int_0^1 v(t, x(t); \theta)\,\mathrm{d}t)$ in θ. The formula for this gradient includes an integral that is then also discretized through a numerical solver.

In [Li et al., 2018], the parameter θ in (8) is allowed to depend on t and the resulting problem is looked at as the *optimal control* problem [Athans and Falb, 2007] of finding a *control function* $\theta(t)$:$[0, 1] \to \Theta$ that is going to guide the solution curve x to the ordinary differential equation $x'(t) = v(t, x(t); \theta(t))$ towards a ending point $x(1)$ with a small loss. The authors then use Pontryagin's maximum principle (PMP, [Pontryagin, 1987]) to state the necessary optimality conditions for this optimal control problem, and then use the method of successive approximations (MSA, [Chernousko and Lyubushin, 1982]) to discretize and numerically solve for these optimality conditions. This leads to an alternative training algorithm for residual networks.

To improve adversarial robustness, [Wang et al., 2019a] add a diffusion term to the transport/continuity equation (equation (11) in Section 2.8 below) that a residual network approximates, which amounts to training an ensemble of Gaussian-perturbed residual networks.

2.8 Optimal Transport Theory

We quickly introduce here the theory of optimal transport which we will use in the following sections to analyze residual networks as transport maps. More details are in Appendix B. Optimal transport deals with the problem of displacing mass from one configuration to another while expending the least amount of effort possible. This minimal effort required to move mass from one distribution to another is actually a distance between distributions that we can equip the space of probability distributions with to get the so-called *Wasserstein space*. Let us first define the *push-forward measure*:

Definition 1. For a function $T : X \to Y$ and a measure μ on X, the push-forward of μ by T is the measure $T_\sharp\mu$ on Y defined by $T_\sharp\mu(A) = \mu(T^{-1}(A))$ for every measurable set $A \subset Y$.

The metric Wasserstein space $\mathbb{W}_2(\Omega) := (\mathcal{P}(\Omega), W_2)$, with Ω a convex and compact subset of $\mathbb{R}^d$, is the space $\mathcal{P}(\Omega)$ of probability measures over Ω, equipped with the distance W_2 given by the solution to the *Kantorovitch problem*

$$W_2^2(\mu, \nu) = \inf_{\gamma \in \Pi(\mu,\nu)} \int_{\Omega\times\Omega} \|x - y\|^2 \,\mathrm{d}\gamma(x, y) \tag{9}$$

where $\Pi(\mu, \nu)$ is the set of probability distribution over $\Omega \times \Omega$ with first marginal equal to μ and second marginal equal to ν, i.e

$$\Pi(\mu, \nu) = \{\gamma \in \mathcal{P}(\Omega \times \Omega) \mid \pi_{1\sharp}\gamma = \mu,\ \pi_{2\sharp}\gamma = \nu\}$$

where $\pi_1(x, y) = x$ and $\pi_2(x, y) = y$ are projections. Here, $\gamma \in \Pi(\mu, \nu)$ is called a *transport plan* between μ and ν, and $\gamma(x, y)$ indicates how much mass from position x should be sent to position y (see Figure 8 below). This problem has a solution in our setting and W_2 is a geodesic distance (see Appendix B).

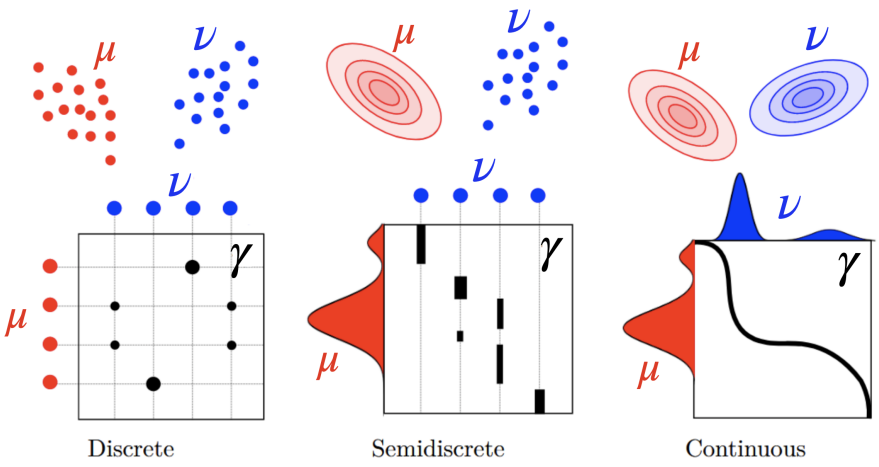

Figure 8: The Kantorovitch problem from μ to ν. From [Peyre and Cuturi, 2019]. The black dots and curve represent the transport plan γ.

If μ is absolutely continuous and $\partial\Omega$ is μ-negligible then the minimization problem in (9) has a unique solution and is equivalent to the *Monge problem*

$$W_2^2(\mu, \nu) = \min_{T \text{ s.t. } T_\sharp \mu = \nu} \int_\Omega \|T(x) - x\|^2 \, \mathrm{d}\mu(x) \tag{10}$$

In the Monge problem, mass is no longer split and all the mass in x goes to $T(x)$ (see Figure 9 below). A function T that satisfies $T_\sharp \mu = \nu$ is called a *transport map* between μ and ν.

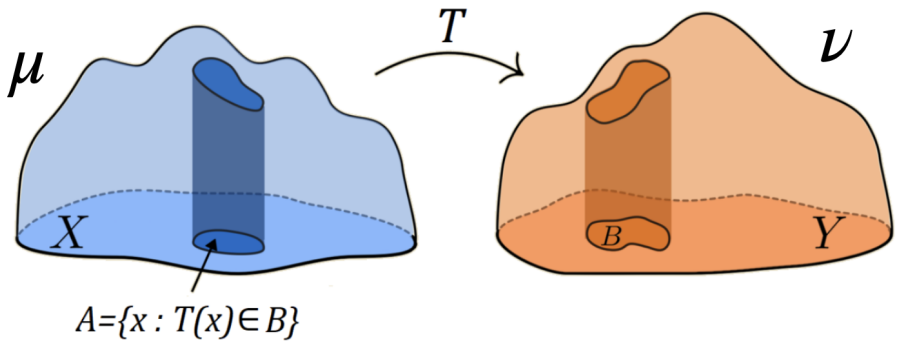

Figure 9: The Monge problem from μ to ν. From [Kolouri et al., 2017].

The Monge problem has a unique solution $T^\star$ linked to the solution $\gamma^\star$ of (9) through $\gamma^\star = (\mathrm{id}, T^\star)_\sharp \mu$, where $(\mathrm{id}, T^\star)$ is the function $x \mapsto (x, T^\star(x))$. The Monge formulation is more suitable to the classification problems we consider, since we don't have a specific target position y to learn a transport plan $\pi(x, y)$.

Another equivalent formulation of the optimal transport problem in this setting is the dynamical formulation [Benamou and Brenier, 2000]. Here, instead of directly pushing samples of μ to ν using T, we can equivalently displace mass, according to a continuous flow with velocity $v_t : \mathbb{R}^d \to \mathbb{R}^d$. This implies that the density μ_t at time t satisfies the *continuity equation*

$$\partial_t \mu_t + \nabla \cdot (\mu_t v_t) = 0 \tag{11}$$

assuming that initial and final conditions are given by $\mu_0 = \mu$ and $\mu_1 = \nu$ respectively. The continuity equation describes the evolution of a probability distribution under a velocity field. In this case, the optimal displacement is the one that minimizes the total action caused by the velocity field v :

$$\begin{aligned} W_2^2(\mu, \nu) = &\min_v \int_0^1 \|v_t\|^2_{L^2(\mu_t)} \, \mathrm{d}t \\ &\text{s.t. } \partial_t \mu_t + \nabla \cdot (\mu_t v_t) = 0, \ \mu_0 = \mu, \mu_1 = \nu \end{aligned} \tag{12}$$

Instead of describing the evolution of the density through the continuity equation, we can describe the paths ϕ_t^x taken by particles at position x from μ when displaced along the flow v. Here ϕ_t^x is the position at time t of the particle that was at $x \sim \mu$ at time 0. The continuity equation is then equivalent to $\partial_t \phi_t^x = v_t(\phi_t^x)$. Rewriting the conditions as necessary, Problem (12) becomes

$$\begin{aligned} W_2^2(\mu, \nu) = &\min_v \int_0^1 \|v_t\|^2_{L^2((\phi_t)_\sharp \mu)} \, \mathrm{d}t \\ &\text{s.t. } \partial_t \phi_t^x = v_t(\phi_t^x), \ \phi_0 = \mathrm{id}, (\phi_1)_\sharp \mu = \nu \end{aligned} \tag{13}$$

and the optimal $T^\star$ that solves (10) is in fact $T^\star(x) = \phi_1^x$ for ϕ that solves $\partial_t \phi_t^x = v_t^\star(\phi_t^x)$ together with the optimal $v^\star$ from (13). This formulation is the most useful to us, as we will discretize $\partial_t \phi_t^x = v_t^\star(\phi_t^x)$ to get a residual architecture.

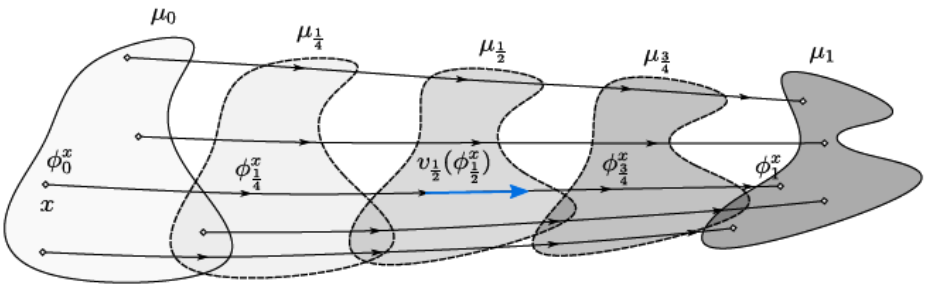

Figure 10: Dynamic transport of μ_0 to μ_1 according to velocity field v (ϕ_t^x is $\phi_t(x)$ in the text). From [de Bézenac et al., 2021].

Since we will work with a discrete set of input points, this second dynamic formulation (13) is more useful to us.

Optimal transport maps have some regularity properties under some boundedness assumptions. We include one such result here in Theorem 1, and another we will use in Appendix B.5, along with more background on optimal transport theory. First let us define Hölder continuity, which is equivalent to Lipschitz continuity when $\eta = 1$:

Definition 2. A function f is η-Hölder on a set $X \subset \mathbb{R}^d$ if $\forall\ a, b \in X$, we have

$$\|f(a) - f(b)\| \leq C\|a - b\|^\eta$$

for some constants $C{>}0$ and $0{<}\eta{\leq}1$. We denote this $f \in \mathcal{C}^{0,\eta}(X)$. If all derivatives of f up to the k-th order are η-Hölder on X, we write $f \in \mathcal{C}^{k,\eta}(X)$.

We can now state the following regularity theorem for optimal transport maps:

Theorem 1. Suppose there are X, Y, bounded open sets in $\mathbb{R}^d$, such that the densities of μ and ν are null in their respective complements and bounded away from zero and infinity over them respectively.

Then, if Y is convex, there exists $0 < \eta \leq 1$ such that the optimal transport map T between μ and ν is in $\mathcal{C}^{0,\eta}(X)$.

If Y isn't convex, there exists two relatively closed sets A in X and B in Y such that $T \in \mathcal{C}^{0,\eta}(X \setminus A, Y \setminus B)$, where A and B are of null Lebesgue measure.

Moreover, if the densities are $\mathcal{C}^{k,\eta}$, then $\mathcal{C}^{0,\eta}$ can be replaced by $\mathcal{C}^{k+1,\eta}$ in the conclusions above. In particular, if the densities are smooth, then the transport map is a diffeomorphism.

2.9 Optimal Transport for the Study of Neural Networks

We verify in Section 3.2.2 and Appendix F.5.1 that residual networks are indeed good approximators of optimal transport maps. This was also confirmed in [Gai and Zhang, 2021], which was released shortly after our first work [Karkar et al., 2020], and which finds that residual networks are closer to being optimal transport maps between their input and output distributions than non-residual networks.

Also released shortly after our first work, [Finlay et al., 2020] accelerate the NeuralODE model mentioned in Section 2.7 by adding an optimal transport regularization to the loss. This makes the learned dynamics v well-behaved, which means that less numerical integration steps are needed. While NeuralODE has been tested on classification tasks, it is mainly used for generative and dynamic modeling tasks.

Optimal transport theory was used in [Sonoda and Murata, 2019] to analyze deep gaussian denoising autoencoders (not necessarily implemented through

residual networks) as transport systems. In the continuous limit, they are shown to transport the data distribution so as to decrease its entropy.

Finally, [de Bézenac et al., 2021] reformulate the unsupervised domain translation problem tackled by CycleGAN [Zhu et al., 2017, Sim et al., 2020] as an optimal transport problem that is solved by discretizing its dynamic formulation using residual networks.

Wasserstein gradient flows in the data space (as in this thesis) appeared recently in deep learning [Alvarez-Melis and Fusi, 2021, Gao et al., 2019, Liutkus et al., 2019, Arbel et al., 2019, Ansari et al., 2021], but with a focus on transfer learning and generation, while we focus on classification (see Section 4.3.2 for more details). Rather unrelated to our work, [Chizat and Bach, 2018, Chizat, 2020] used Wasserstein gradient flows in the parameter space to prove convergence of the training of two-layer neural networks.

3 A Least Action Principle for the Training of Neural Networks

Abstract

Neural networks have been achieving high generalization on many tasks despite being highly over-parameterized. Since classical statistical learning theory struggles to explain this behavior, much effort has been focused on uncovering the mechanisms behind it, in the hope of developing a more adequate theoretical framework and having a better control over the trained models. We adopt an alternative perspective, viewing the neural network as a dynamical system displacing input particles over time. We conduct a series of experiments and, by analyzing the network's behavior through its displacements, we show the presence of a low kinetic energy bias in the transport map of the network, and link this bias with generalization performance. From this observation, we reformulate the learning problem as follows: find networks that solve the task while transporting the data as efficiently as possible. This offers a novel formulation of the learning problem which allows us to provide regularity results for the solution network, based on Optimal Transport theory. From a practical viewpoint, this allows us to propose a new learning algorithm, which automatically adapts to the complexity of the task, and leads to networks with a high generalization ability even in low data regimes. The work presented in this section led to the following publication [Karkar et al., 2020] which won the runner-up best student paper award at ECML 2020.

3.1 Introduction

Deep neural networks have repeatedly shown their ability to solve a wide range of challenging tasks, while often having many more parameters than there are training samples. Such a performance of over-parametrized models is counter-intuitive. They seem to adapt their complexity to the given task, systematically achieving a low training error without suffering from over-fitting as could be expected [Belkin et al., 2019, Nakkiran et al., 2020, Zhang et al., 2017a]. This is in contradiction with the classical statistical practice of selecting a class of functions complex enough to represent the coherent patterns in the data, and simple enough to avoid spurious correlations [Belkin et al., 2018, Hastie et al., 2001]. Although this behavior has sparked much recent work towards explaining neural networks' success [De Palma et al., 2019, Jacot et al., 2018, Novak et al., 2018, Rahaman et al., 2019], it still remains poorly understood. Among the factors to consider are the implicit biases present in the choices made for the parametrization, the architecture, the initialization and the optimization algorithm, and that contribute all to this success. Our aim is to uncover some of these hidden biases and highlight their link with generalization.

We will focus on residual networks and adopt the dynamical point of view discussed in Section 2.6, which allows us to leverage the theories and mathematical tools developed to study, approximate and apply differential equations. We

first conduct experiments to observe how neural networks displace their inputs, seen as particles, through time. We measure a strong empirical correlation between good test performance and neural networks with low kinetic energy along their transport flow. From this, we reformulate the training problem as follows: retrieve the network which solves the task using the principle of least action, i.e. expending as little kinetic energy as possible. This problem, in its probabilistic formulation, is tightly linked with and inspired by the well-known problem of finding an optimal transportation map. This yields new insights into neural networks' generalization capabilities, and provides a novel algorithm that automatically adapts to the complexity of the data and robustly improves the network's performance, including in low data regimes, without slowing down the training. To summarize, our contributions are the following:

- We highlight a low-energy bias in ResNets.
- We formulate a Least Action Principle for the training of neural networks.
- We prove existence and regularity results for networks with minimal energy.
- We provide an algorithm for retrieving minimal energy networks that leads to better generalization performance on different classification tasks, without complexifying the architecture.

3.2 Empirical Analysis of Transport Dynamics in Residual Networks

Before introducing our framework, we conduct an exploratory analysis of the impact of the network's inner dynamics on generalization. We present below two experiments. The first one highlights how good generalization performance is closely related to low transport cost for classification tasks on MNIST [LeCun et al., 2010] and CIFAR10 [Krizhevsky, 2009]. This cost therefore appears as a natural characterization of the complexity and disorder of a network. The second experiment, performed on a toy 2D dataset, visualizes the transport induced by the blocks of a ResNet.

We consider ResNets where, after encoding, a data point x_0 is transported by applying $x_{m+1} = x_m + r_m(x_m)$ for M residual blocks and then classified using F. We measure the disorder/complexity of a network by its transport cost which is the sum $\mathcal{C} = \sum_m \|r_m(x_m)\|_2^2$ of the displacements induced by its residual blocks. This quantity corresponds to the kinetic energy of the displacement.

3.2.1 Transport Cost and Generalization in Image Classification

In order to study the correlation between the transport cost of a residual network and its generalization ability on image data, we train convolutional 9-block ResNets with different initializations (orthogonal and normal with different gains), for 10-class classification tasks MNIST and CIFAR10. In Figure 11, each

point represents a trained network and gives the transport cost $\mathcal{C}$ as a function of the test accuracy of the network.

This experiment clearly highlights the strong negative correlation between transport cost and good generalization. This illustrates the importance of the implicit initialization bias and motivates initialization schemes which favor a low kinetic energy. A number of factors contribute to this low energy bias: small initialization gains tend to bias $\|r_m(x_m)\|_2^2$ towards small values, and stochastic gradient descent does not change this much.

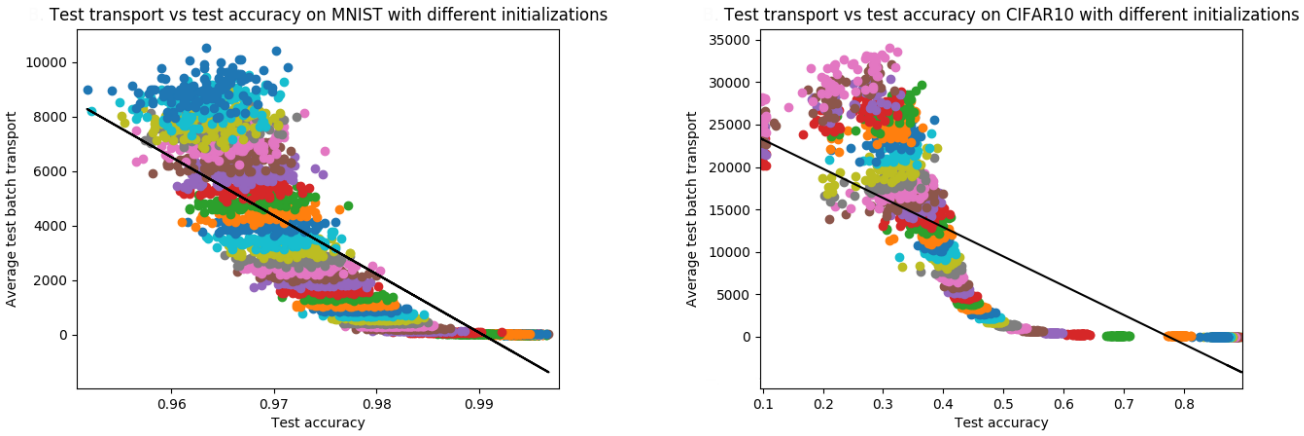

Figure 11: Test transport cost $\mathcal{C}$ against test accuracy of ResNet-9 models on MNIST (left) and CIFAR10 (right) with fitted linear regressions, where each color indicates a different initialization (either orthogonal or normal with varying gains)

3.2.2 Visualizing Network Dynamics in 2 Dimensions

We visualize in 2 dimensions the transport dynamics inside a network. The task is 2-class classification of a non-linearly separable dataset (two concentric circles, from `sklearn` [Pedregosa et al., 2011]) that contains 1000 points with a train-test split of 80%-20%, see Figure 12 top left. The network is a ResNet containing 9 residual blocks, followed by a fixed linear classifier. Each residual block contains two fully connected layers separated by a batch normalization and a ReLU activation.

With the cross-entropy loss alone, the behavior of a well trained and carefully initialized network achieving 100% test accuracy is illustrated in the first row of Figure 12. With a $\mathcal{N}(0, 5)$ initialization, significantly bigger than a good initialization, the test accuracy drops to 98% (average of 100 runs) and the transport becomes chaotic (Figure 12, second row). Adding the transport cost to the loss improves the test accuracy (99.7% on average) of this badly initialized network and the movement becomes more controlled (third row of Figure 12).

Thus, controlling transport improves the behavior and generalization of the net-

work. This allows to explicitly control the network whereas implicit biases such as good initialization rely on heuristics. In Appendix F.5.2, more experiments show that in other situations that deviate from the ideal setting where the task is perfectly solved, e.g. when using a network which is too large or too small, or a small training set, controlling the transport cost also improves generalization.

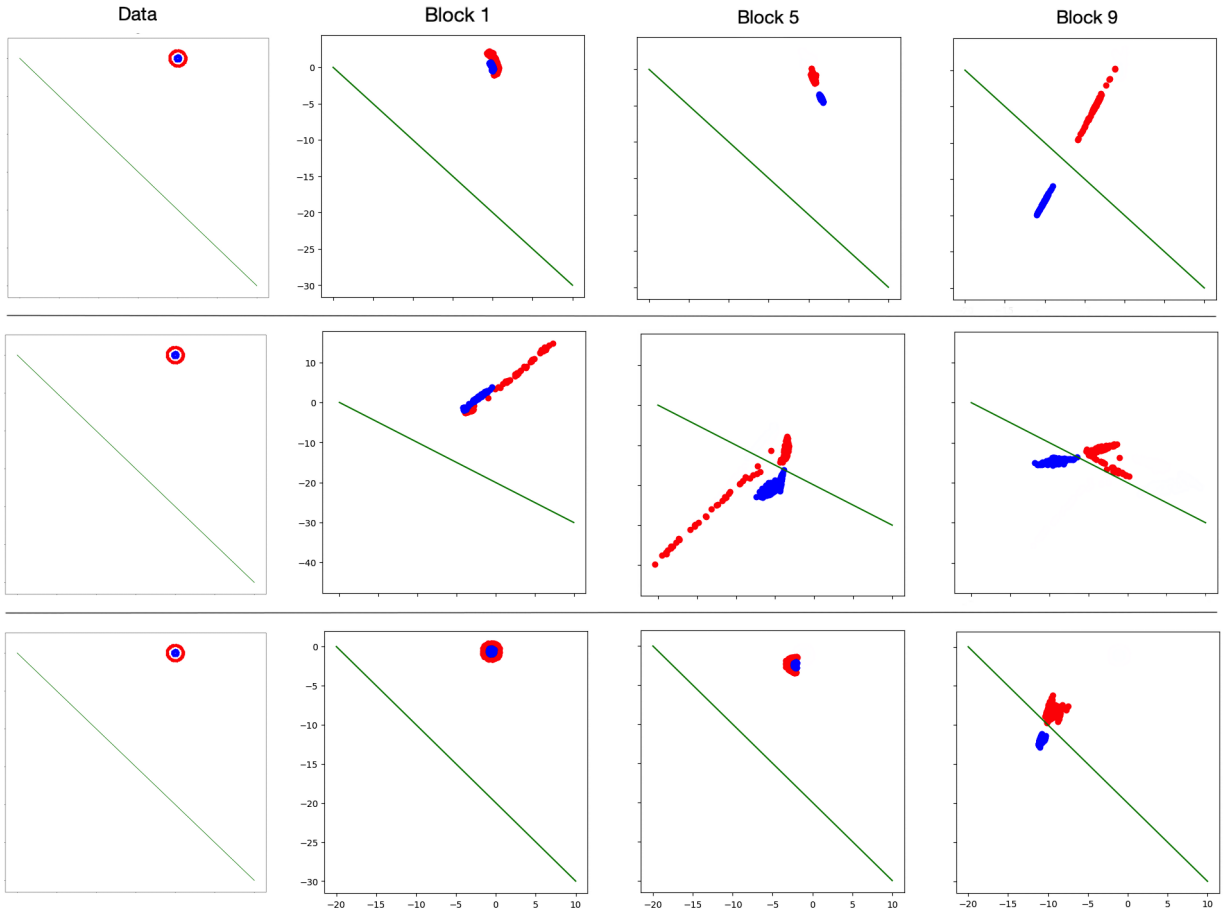

Figure 12: Transformed circles test set by a ResNet-9 after blocks 1, 5 and 9 after training; first row with good initialization; second row with a $\mathcal{N}(0, 5)$ initialization; third row with a $\mathcal{N}(0, 5)$ initialization and the transport cost added to the loss.

In 2 dimensions, we can compute the W_2 distance between successive point clouds before and after each block, and compare it, in Figure 37 in Appendix F.5.1, to the displacement that the block causes. This confirms that ResNets are already close to being optimal transport maps between their input and output distributions, as the distance traveled by the points is close to the minimal distance possible. We see that the transport regularization encourages this bias.

3.3 Method and Theory

3.3.1 General Setting

In order to better understand the inner workings of a neural network, we adopt a viewpoint in which the different mechanisms become apparent and are decoupled.

Decomposing a neural network. We consider the following model of a neural network where computations are separated into the three steps, i.e. $F \circ T \circ E$ (this corresponds to the general structure of recent deep models or to the structure of components of a deep model):

1. **Dimensionality change:** starting from an input distribution ρ on $\mathbb{R}^n$, a transformation $E : \mathbb{R}^n \to \Omega \subset \mathbb{R}^d$ is applied, transforming it into $\mu = E_\sharp \rho$, a distribution on Ω. This corresponds to the first few layers in most architectures and represents a change of dimension. We call E the *encoder*.
2. **Data transport:** then μ is transformed by a mapping $T : \Omega \to \Omega$, which we see as a transport map. Here, the dimensionality doesn't change and, if this part of the network is a sequence of residual blocks, T can be written as the discretized flow of an ODE.
3. **Task-specific final layers:** a final function $F : \mathbb{R}^d \to \mathcal{Y}$ is applied to $T_\sharp \mu$ in order to compute the loss $\mathcal{L}$ associated with the task at hand, e.g. F could be a linear classifier. Like E, F is typically made up of a few layers.

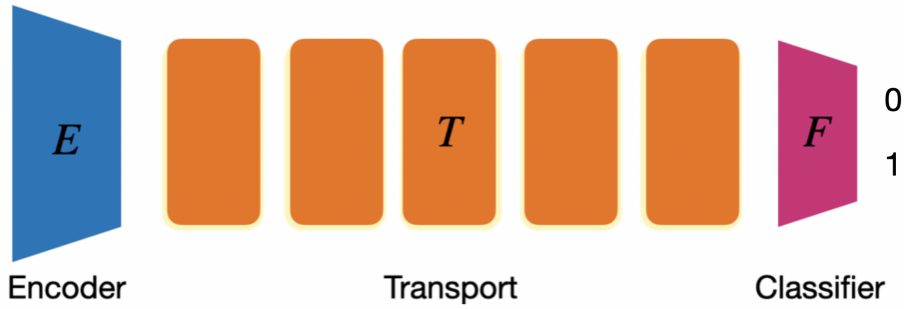

Figure 13: Decomposition of a neural network

Our focus is on analyzing the second phase, data transport, and we assume that the encoder E is pretrained and fixed (this will be relaxed in the experiments). To solve a complex non-linear task for which a neural network is needed, the data has to be transformed in a non-trivial way, meaning that this is an essential phase, for example in the case of classification, $T_\sharp \mu$ needs to be linearly separable if F is linear. This model is quite general, as many ResNet-like architectures [He et al., 2016b, He et al., 2016a, Xie et al., 2017, Zagoruyko and Komodakis, 2016] alternate modules that change the dimensionality (step 1, typically pooling or patch merging) and transport modules that keep the dimensionality fixed (step 2) and according to [Jastrzebski et al., 2018], the transport modules have similar

behavior. The model can then be considered as a simplified ResNet, sometimes called a *single representation* ResNet. Note that [Sandler et al., 2019] finds that networks that keep the same resolution remain competitive.

The set of admissible targets. As recent neural architectures have systematically achieved near-zero training error [Belkin et al., 2019, Belkin et al., 2018, Jacot et al., 2018, Zhang et al., 2017a], we place ourselves in this regime, which makes it possible to model this as a hard constraint. For some tasks, this constraint over T is obvious: in a generative setting for example, $T_\sharp\mu$ must be equal to some prescribed distribution ν which is the target of the generation process. But in general, T is less strictly constrained and the condition depends on F and $\mathcal{L}$. This leads us to define a *set of admissible targets* for the task:

$$S_{F,\mathcal{L}} = \{\nu \in \mathcal{P}(\Omega) \mid \mathcal{L}(F, \nu) = 0\} \tag{14}$$

In general, $\mathcal{L}$ is fixed while F is learned jointly with T. This set is supposed to be non-empty for some F and, in general, it will contain many distributions. The goal of the learning task can then be reformulated as:

$$\text{Find } (T, F) \text{ such that } T_\sharp\mu \in S_{F,\mathcal{L}} \tag{15}$$

An important observation is that, even when $S_{F,\mathcal{L}}$ is reduced to a singleton, the problem is still strongly under-constrained and it is possible to obtain many such (T, F) that lead to poor generalization. One can then ask why this is not the case in practice, as good generalization performance is usually achieved.

The case of classification. Even though our framework is general, we focus our experiments on classification tasks, with $\mathcal{L}$ being the cross entropy loss. The task consists in separating N classes. We denote μ_i the class distributions which are supposed to be distributions in $\mathbb{R}^d$ of mutually disjoint supports, meaning that there is no ambiguity in the class of data points, and such that $\mu = \sum w_i\mu_i$ with $\sum w_i = 1$ and $w_i \in]0, 1[$. We want to find a transformation T of these distributions such that all transported distributions can be correctly classified by a classifier F. When F is linear, $S_{F,\mathcal{L}}$ is the set of distributions which have N components that are linearly separated by F. Note that we place ourselves in a noiseless ideal setting where perfect classification is possible. In addition to the problem of finding T being strongly under-constrained, finding such a mapping is difficult, especially when only few data samples are available, and one might wonder whether some mappings are easier to calculate than others.

3.3.2 Formulation

Given the empirical observations of Section 3.2, a natural way to select a robust model is to select, among the maps which transport μ to $S_{F,\mathcal{L}}$ and thus solve the task, one with a minimal transport cost. This gives us the following optimization problem:

$$\begin{aligned} \inf_{T,F} \quad & \mathcal{C}(T) = \int_\Omega c(x, T(x))\, \mathrm{d}\mu(x) \\ \text{subject to} \quad & T_\sharp\mu \in S_{F,\mathcal{L}} \end{aligned} \tag{16}$$

We consider ground costs $c(x, y) = \|x - y\|^p$ with $p > 1$, and suppose Ω compact with negligible boundary and μ absolutely continuous. We assume that the space of classifiers is compact, that the loss $\mathcal{L}$ is continuous, that the set $\cup_{F\in\mathcal{F}} S_{F,\mathcal{L}}$ is at a finite p-Wasserstein distance W_p from μ (in particular, it is non-empty) and that all its bounded subsets are totally bounded (*i.e.* can be covered by finitely many subsets of any fixed size). These properties depend on the choice of the loss $\mathcal{L}$ and of a class of functions $\mathcal{F}$ for the classifier F. The equivalent dynamical version of (16) is then, as per Section 2.8,

$$\begin{aligned} \inf_{v,F} \quad & \int_0^1 \|v_t\|^p_{L^p((\phi_t^\cdot)_\sharp\mu)} \, \mathrm{d}t \\ \text{subject to} \quad & \partial_t \phi_t^x = v_t(\phi_t^x) \\ & \phi_0^\cdot = \mathrm{id} \\ & (\phi_1^\cdot)_\sharp \mu \in S_{F,\mathcal{L}} \end{aligned} \tag{17}$$

where $\|v_t\|^p_{L^p((\phi_t^\cdot)_\sharp\mu)} = \int_{\mathbb{R}^d} \|v_t\|^p \, \mathrm{d}(\phi_t^\cdot)_\sharp\mu$. The result below shows that Problems (16) and (17) are equivalent and that the infima are realized as minima when $p = 2$:

Theorem 2. The infima of (16) and (17) are finite and are realized through a map T which is (or a velocity field v which induces) an optimal transportation map. When $c(x, y) = \|x - y\|^p$, then (16) and (17) are equivalent.

Proof. From the hypothesis above, there exists $\nu \in S_{F,\mathcal{L}}$ at a finite distance from μ. Taking any transport map between μ and ν, we see that the infima are finite (see Appendix B.1 for existence of optimal transport maps).

Consider (16) and take a minimizing sequence $(T_i, F_i)_i$. Set $\beta_i = (T_i)_\sharp\mu$. Then $(\mathcal{C}(T_i))_i$ converges to the infimum which is strictly bounded by $M > 0$. Then, by definition, for i large enough, $W_p^p(\mu, \nu_i) \leq \mathcal{C}(T_i) \leq M$. So that $(\nu_i)_i$ is a bounded sequence in $\cup_F S_{F,\mathcal{L}}$. By the hypothesized total boundedness of bounded subsets and as $\mathcal{P}_p(\mathbb{R}^d)$ endowed with W_p is a complete metric space (see [Bolley, 2008] for a proof), up to an extraction, $(\nu_i)_i$ converges to $\nu^\star$ in the closure of $\cup_F S_{F,\mathcal{L}}$. Moreover, up to an extraction, $(F_i)_i$ also converges to $F^\star$ by compactness of the class of classifiers. Taking $T^\star$ the optimal transport map between μ and $\nu^\star$, we then have, by continuity of $\mathcal{L}$,

$$T^\star_\sharp \mu = \nu^\star \in S_{F^\star,\mathcal{L}}$$

and $\mathcal{C}(T^\star) \leq \lim \mathcal{C}(T_i)$ by optimality of $T^\star$, which means, since $(\mathcal{C}(T_i))_i$ is a minimizing sequence, that $\mathcal{C}(T^\star)$ minimizes (16). So $(T^\star, F^\star)$ is a minimizer and $T^\star$ is an optimal transport map.

Finally, there exists, (Appendix B.4), a velocity field $v_t^\star$ inducing the optimal transport map between μ and $\nu^\star$ which then gives a minimizer $(v^\star, F^\star)$ for (17). By the same reasoning, taking a minimizing sequence $(v^{(i)}, F_i)_i$ and the induced maps T_i shows that both problems are equivalent. □

Note that uniqueness doesn't hold anymore, as the constraint $T_\sharp\mu \in S_{F,\mathcal{L}}$ in (17) is looser than in standard OT. However, as we show in the following section, the fact that the optimization problems are solved by OT maps will give regularity properties for the models induced by these optimization problems.

3.3.3 Regularity

Intuitively, the fact that we minimize the energy of the transport map transforming the data is akin to the idea of Occam's razor: among all the possible networks that correctly solve the the task, the one transforming the data in the simplest way is selected. Moreover, it is possible to show that this optimal transformation is regular: our formulation provides an alternate view on generalization for modern deep learning architectures in the overparametrized regime.

Optimal maps can be as irregular as needed in order to fit the target distribution, however in much the same way as successfully trained DNNs, optimal maps are still surprisingly regular. In a way, they are as regular as possible given the constraints which is exactly the type of flexibility needed. However, the constraints in (16) and (17) are looser than in the standard definitions of Optimal Transport. Still, supposing that the input data distribution has a nicely behaved density, namely bounded and of compact support, with the same hypothesis as above, we have the following, which is mainly a corollary of Theorem 2:

Proposition 1. Consider $T^\star$ the OT map induced by (16) (or (17)) given by Theorem 2. Take X, respectively Y, an open neighborhood of the support of μ, respectively of $T^\star_\sharp\mu$, then $T^\star$ is differentiable, except on a set of null μ measure.

Additionally, if X is convex and $T^\star$ doesn't have singularities, then there exists A, respectively B, relatively closed in X, respectively Y, both of null measure, such that $T^\star$ is η-Hölder continuous from $X\backslash A$ to $Y\backslash B$, i.e. for constants $0 < \eta \leq 1$ and $C > 0$ and $\forall\ x, y \in X \setminus A$ we have

$$\|T(x) - T(y)\| \leq C\|x - y\|^\eta$$

Proof. This is a consequence of Theorem 2, the hypothesis made in this section and the regularity theorems in Appendix B.5. □

Remark 3. If Y in Proposition 1 is convex then $T^\star$ is η-Hölder on all of X, i.e. $T \in \mathcal{C}^{0,\eta}(X)$ (see Theorem 1).

Remark 4. If the distributions μ and $\nu = T^\star_\sharp\mu$ in Proposition 1 are $\mathcal{C}^{k,\eta}$ (i.e all derivatives up to the k-th derivative are η-Hölder), then the optimal transport map T is $\mathcal{C}^{k+1,\eta}$ (see Theorem 1). This means that the more regular the data, the more regular the network we find.

There are two results in Proposition 1. The first gives α-*a.e.* differentiability. This is already as strong as might be expected from a classifier: there are necessarily discontinuities at the frontiers between different classes. The second is even more interesting: it gives η-Hölder continuity over as large a domain

as possible, and even a diffeomorphism if the data distribution is well-behaved enough. Here η measures smoothness, the higher its value, the better. In particular, in the case of classification, this means that the Hausdorff dimension along the frontiers between the different classes is scaled by less than a factor of $1/\eta$ in the transported domain. If the densities are smooth, the dimension even becomes provably smaller by this result.

Intuitively, this means that the data is transported in a way that preserves and simplifies the patterns in the input distribution. In the following, we propose a practical algorithm implementing these models and use it for standard classification tasks, showing an improvement over standard models.

3.4 Practical Implementation

We propose here an algorithm for training ResNets using the least action principle by minimizing the kinetic energy. Starting from problem (17) with $p = 2$ and the Euclidean norm, we first discretize the differential equation via a forward Euler scheme, which yields $\phi^x_{m+1} = \phi^x_m + r_m(\phi^x_m)$. The discretized flow r_m approximates v_t at time $t = m/M$ and is parameterized by a residual function, giving a standard residual architecture. The residual blocks, along with a classifier F, are parametrized by θ. Next, the constraint $(\phi^{\cdot}_1)_\sharp\mu \in S_{F,\mathcal{L}}$ is rewritten as $\mathcal{L}(F, (\phi^{\cdot}_1)_\sharp\mu) = 0$, denoted $\mathcal{L}(\theta) = 0$ below. Finally, as we only have access to a finite set $\mathcal{D}$ of samples x from μ, we use a Monte-Carlo approximation of the integral with respect to the distributions $(\phi^{\cdot}_t)_\sharp\mu$. This gives us

$$\begin{aligned} \min_\theta \quad & \mathcal{C}(\theta) = \sum_{x\in\mathcal{D}} \sum_{m=0}^{M-1} \|r_m(\phi^x_m)\|^p \\ \text{subject to} \quad & \phi^x_{m+1} = \phi^x_m + r_m(\phi^x_m) \text{ and } \phi^x_0 = x \text{ for all } x \in \mathcal{D} \\ & \mathcal{L}(\theta) = 0 \end{aligned} \tag{18}$$

Is is easy to see that the min-max problem $\min_\theta \max_{\lambda>0}\ \mathcal{C}(\theta) + \lambda\ \mathcal{L}(\theta)$ yields the same solution as Problem 18, as the first two constraints are satisfied trivially. If the constraint $\mathcal{L}(\theta) = 0$ corresponding to solving the task, which includes the classifier F, is not verified, this will cause the second term to grow unbounded, and the solution will thus be avoided by the minimization. This min-max problem can be solved using an iterative approach, starting from some initial λ_0 and θ_0:

$$\begin{cases} \theta_{i+1} = \arg\min_\theta\ \mathcal{C}(\theta) + \lambda_i\ \mathcal{L}(\theta) \\ \lambda_{i+1} = \lambda_i + \tau\ \mathcal{L}(\theta_{i+1}) \end{cases} \tag{19}$$

The minimization is done via stochastic gradient descent (SGD) for a number of steps s, where a step means a batch, starting from the previous parameter value θ_i. This algorithm is similar to Uzawa's algorithm (also called the method

of multipliers) used in convex optimization [Santambrogio, 2015]. In practice, it is much more stable to divide the minimization objective in (19) by λ_i, yielding the following algorithm:

Algorithm: Training with Least Action Principle (LAP-Net)

Input: Training samples, step size τ, number of steps s, initial weight λ_0
Initialization: Initialize the parameters θ_0 and set $i = 0$
while *not converged* do
1. Starting from θ_i, perform s steps of stochastic gradient descent:
1.1. $\theta_{i+1}^0 = \theta_i$
1.2. $\theta_{i+1}^l = \theta_{i+1}^{l-1} - \epsilon(\nabla \mathcal{C}(\theta_{i+1}^{l-1})/\lambda_i + \nabla \mathcal{L}(\theta_{i+1}^{l-1}))$ for l from 1 to s
1.3. $\theta_{i+1} = \theta_{i+1}^s$
2. Update the weight $\lambda_{i+1} = \lambda_i + \tau\ \mathcal{L}(\theta_{i+1})$ and increment $i \leftarrow i + 1$
Output: Learned parameters θ

While the high non-convexity makes it difficult to ensure exact optimality, we can still have some induced regularity when reaching a good local minimum:

Proposition 2. Suppose the pair $(F^{\theta^\star}, T^{\theta^\star})$ is reached by the optimization algorithm such that $T^{\theta^\star}$ is an $\epsilon-$OT map between μ and its push-forward (meaning $\|T^{\theta^\star} - T^\star\|_\infty \leq \epsilon$ where $T^\star$ is the OT map). Then we have, with the same notations as in Proposition 1,

$$\forall x, y \in X \setminus A,\ \|T^{\theta^\star}(x) - T^{\theta^\star}(y)\| \leq O(\epsilon + \|x - y\|^\eta)$$

Proof. We simply write the decomposition:

$$T^{\theta^\star}(x) - T^{\theta^\star}(y) = T^{\theta^\star}(x) - T^\star(x) + T^\star(x) - T^\star(y) + T^\star(y) - T^{\theta^\star}(y)$$

and use the triangular inequality: the first and third terms are smaller than ϵ by hypothesis while Hölder continuity applies for the second by Proposition 1. □

This shows that minimizing the transport cost still endows the model with some regularity, even in situations where the global minimum is not reached.

We also remark is that we can also consider a relaxation of the optimization problem 18 by assigning a fixed weight λ to $\mathcal{L}$ and minimizing $\mathcal{C}(\theta) + \lambda \mathcal{L}(\theta)$. This provides a simpler and quite competitive benchmark (see Appendix F.3).

3.5 Experiments

3.5.1 Generalization

MNIST Experiments. The base model is a ResNet with 9 residual blocks. Two convolutional layers first encode the image of shape $1 \times 28 \times 28$ into shape $32 \times 14 \times 14$. A residual block contains two convolutional layers, each preceded by a ReLU activation and batch normalization. The classifier is made up of two fully connected layers separated by batch normalization and a ReLU activation.

We use an orthogonal initialization [Saxe et al., 2014] with gain 0.01. This and all vanilla models and their training regimes are implemented by following closely the cited papers that first introduced them, and our method is added over these training regimes. More implementation details are in Appendix F.1.

From the experiments in two dimensions, we suspect that adding the transport cost helps when the training set is small. For performance comparisons, we average the highest test accuracy achieved over 30 training epochs (over random orthogonal weight initializations and random subsets of the complete training set). We find that adding the transport cost improves generalization when the training set is very small (Table 1). We see that the improvement becomes more important as the training set becomes smaller and reaches an increase of almost 14 percentage points in the average test accuracy.

Table 1: Average highest test accuracy and 95% confidence interval of ResNet-9 over 50 instances on MNIST with training sets of different sizes (in %)

Train size	ResNet	LAP-ResNet
500	90.8 ± 0.4	**90.9** ± 0.2
400	88.4 ± 0.5	**88.4** ± 0.4
300	83.5 ± 0.5	**86.2** ± 0.4
200	74.9 ± 1.0	**82.0** ± 0.5
100	56.4 ± 1.5	**70.0** ± 1.0

CIFAR10 Experiments. We run the same experiments on CIFAR10. The architecture is the same except that the encoder transforms the input which is of shape $3 \times 32 \times 32$ into shape $100 \times 16 \times 16$. We use $\lambda_0 = 0.1$, $\tau = 0.1$ and $s = 50$. We average the highest test accuracy achieved over 200 training epochs over random orthogonal weight initializations and random subsets of the complete train set. Here, we find that adding the transport cost helps for all sizes of the train set (which has 50 000 images in total). The increase in average precision becomes more important as the train set becomes smaller (Table 2).

Table 2: Average highest test accuracy and 95% confidence interval of ResNet-9 over 20 instances on CIFAR10 with training sets of different sizes (in %)

Train size	ResNet	LAP-ResNet
50 000	91.49 ± 0.10	**91.94** ± 0.10
30 000	88.61 ± 0.14	**89.41** ± 0.10
20 000	85.73 ± 0.14	**86.74** ± 0.13
10 000	79.25 ± 0.25	**80.90** ± 0.16
5 000	70.32 ± 0.32	**72.58** ± 0.21
4 000	67.80 ± 0.26	**70.12** ± 0.30

CIFAR100 experiments. On CIFAR100 [Krizhevsky, 2009], results using a ResNet are in Appendix F.3. We also used the ResNeXt [Xie et al., 2017] architecture. The residual block of a ResNeXt applies $x + \sum_i w_i(x)$ with the functions w_i having the same architecture but independent weights, followed by a ReLU activation. We used the ResNeXt-50-32×4d architecture detailed in [Xie et al., 2017]. This is a much bigger and state-of-the-art network, as compared with the single representation ResNets used so far. It also extends the experimental results beyond the theoretical framework in three ways: the embedding dimension changes between the residual blocks, a block applies $x_{m+1} = \texttt{ReLU}(x_m + \sum_i w_{m,i}(x_m))$ and the encoder is no longer fixed. We found that penalizing $\sum_i w_{m,i}(x_m)$ or the true residue $x_{m+1} - x_m$ is essentially equivalent. Table 3 shows consistent accuracy gains as our method (with $\lambda_0 = 1$, $\tau = 0.1$ and $s = 5$) corrects the overfitting of the bigger ResNeXt network compared to ResNet.

Table 3: Average highest test accuracy and 95% confidence interval of ResNeXt50 over 10 instances on CIFAR100 with training sets of different sizes (in %)

Train size	ResNeXt50	LAP-ResNeXt50
50 000	72.97 ± 1.18	**76.11** ± 0.78
25 000	62.55 ± 2.37	**64.11** ± 1.85
12 500	45.90 ± 2.74	**48.23** ± 1.84

3.5.2 Transport Visualization

We verify in Appendix F.2 using a pretrained autoencoder for visualization that the embeddings learned through LAP training on image datasets do indeed remain closer than the embeddings learned through vanilla training to the initial embedding that the first residual block receives.

More precisely, if we pretrain an autoencoder on an image dataset, we can use its encoder as the encoder of the residual network and freeze it during training. This makes it possible to visualize, in the same space as the input data (so as images), how the embeddings evolve by decoding, using the pretrained decoder, the outputs of the residual blocks. And indeed we find on MNIST, CIFAR10 and CIFAR100 that the decodings of the network's embeddings remain closer to the initial reconstruction by the autoencoder when using LAP training.

3.5.3 Stability

The stability of networks trained with our method is hinted at by the regularity results of Propositions 1 and 2 and is also seen in the lower variance in the performances of LAP-ResNets in Tables 1, 2 and 3 above, and in the tables in Appendices F.5 and F.3. This is expected as the model becomes more constrained and can be seen as an advantage, especially in cases where the results vary more with the initialization.

To explore this increased training stability, we try training ResNets that get deeper without batch-normalization on CIFAR10. We see in Figure 14 that training deep ResNets without batch-normalization is near impossible, whereas LAP-ResNets maintain the same performance and stability without batch-normalization for up to 50 blocks.

LAP-ResNets are also compared in this regard to the small step method of [Zhang and Wynter, 2018, Zhang et al., 2019], which simply adds a small weight h of around 0.1 in front of the residue function to make ResNets more stable. The Least Action Principle has the same stable test accuracy when training without batch-normalization in Figure 14 as this method, while also improving the test accuracy when using batch-normalization.

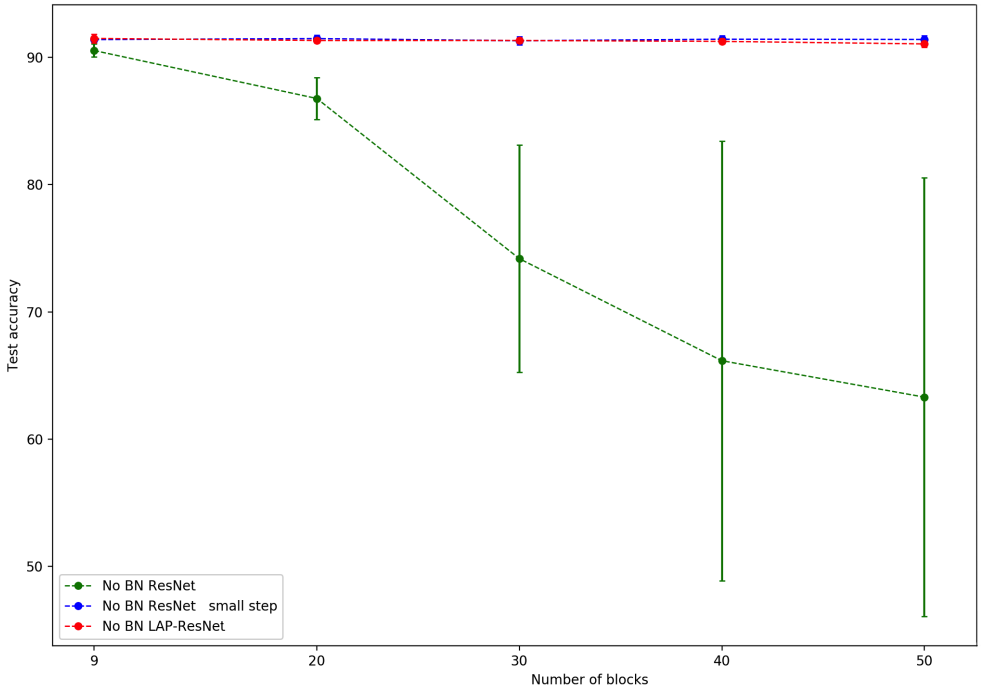

Figure 14: Test accuracy of ResNets of various depths without batch-normalization on CIFAR10.

3.5.4 Training Dynamics

We now look at the evolution of the test accuracy during training in Figure 15 below. We see that the Least Action Principle acts by speeding up training in the first epochs and more generally when a jump in test accuracy happens.

The orange curve in Figure 15 is that of a network trained with the transport cost simply as a regularizer. That means that instead of using the adaptive

optimization algorithm (19), we solve $\mathcal{L}(\theta) + \lambda\, \mathcal{C}(\theta)$ with a fixed λ. Results using this training are in Appendix F.3. While it does not perform as well as LAP training, it has fewer hyperparameters to tune and still leads to better test accuracy than vanilla training.

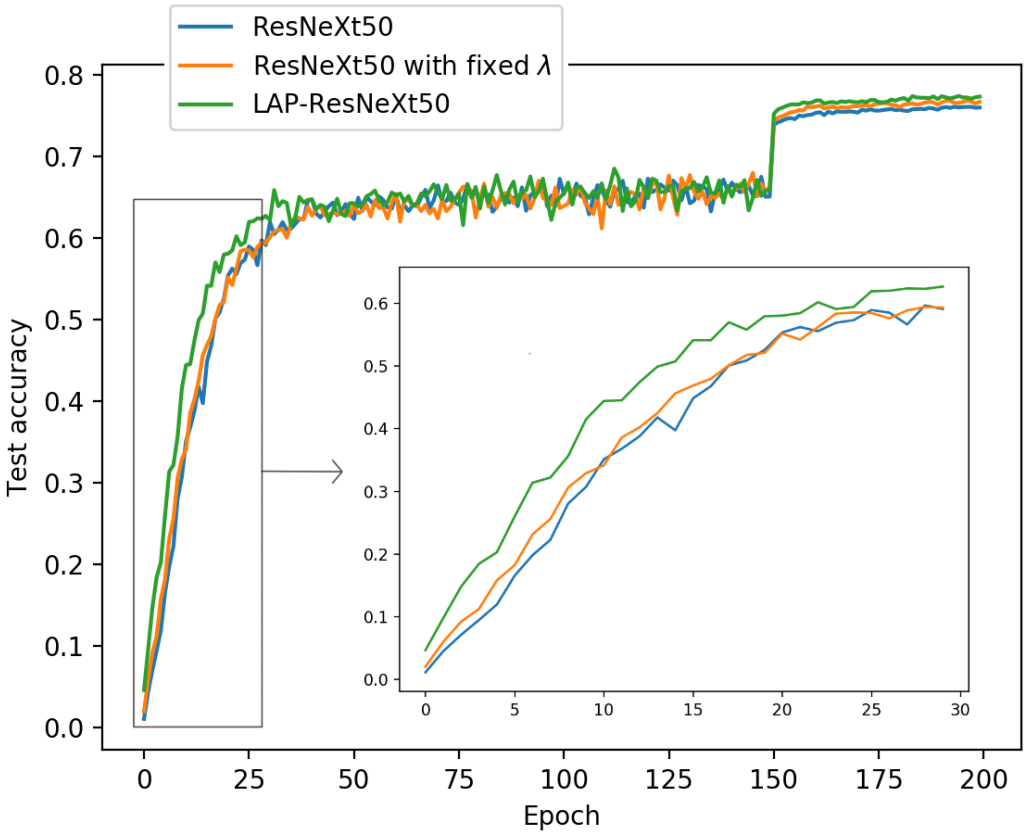

Figure 15: Test accuracy during training of ResNeXt50 models on CIFAR100.

3.6 Discussion and Conclusion

We can summarize this section as follows. We have empirically motivated and then formulated the following learning problem: among all the neural networks that correctly solve the task, select the one that transforms the data with the lowest transport cost. We have proven general results of existence and regularity for models trained this way, studied their behavior in low-dimensional settings when compared to vanilla models and shown their efficiency on standard classification tasks. We also found that the training is stabilized in an adaptive fashion without being slowed down.

Note that even though we have only considered residual networks, this framework can be applied to any architecture by considering the static formulation (16) of the problem. The improved stability of LAP-Nets suggests also looking into their robustness to adversarial attacks. Another avenue of research would be to experiment with alternative ground transport costs.

4 Module-wise Training via the Minimizing Movement Scheme

Abstract

Greedy layer-wise or module-wise training of neural networks is compelling in constrained and on-device settings where memory is limited, as it circumvents a number of problems of end-to-end back-propagation. However, it suffers from a stagnation problem, whereby early layers overfit and deeper layers stop increasing the test accuracy after a certain depth. We propose to solve this issue by introducing a simple module-wise regularization inspired by the minimizing movement scheme for gradient flows in distribution space. We call the method TRGL for Transport Regularized Greedy Learning and study it theoretically, proving that it leads to greedy modules that are regular and that progressively solve the task. Experimentally, we show improved accuracy of module-wise trained networks when our regularization is added, superior to that of other module-wise training methods and often to end-to-end training, with as much as 60% less memory usage. The work presented in this section has led to the following publication [Karkar et al., 2023a] at NeurIPS 2023.

4.1 Introduction

End-to-end backpropagation is the standard training method of neural nets. But there are reasons to look for alternatives. First, it is deemed biologically implausible [Gupta, 2020, Mostafa et al., 2018, Liao et al., 2016]. Second, it requires loading the whole model during training which can be impossible in constrained settings such as on mobile devices [Teng et al., 2020, Tang et al., 2021]. Thirdly, it forces the training of systems of cooperative networks to be sequential and synchronous, which is not flexible when the networks are distributed between a central agent and clients that operate at different rates [Jaderberg et al., 2017]. Fourthly, it prohibits training the layers in parallel. These limitations follow from the locking problems [Jaderberg et al., 2017] that end-to-end backpropagation suffers from: forward locking (each layer must wait for the previous layers to process its input), update locking (each layer must wait for the end of the forward pass to be updated) and backward locking (each layer must wait for errors to backpropagate from the last layer to be updated).

Dividing the network into modules, a module being made up of one or more layers, and greedily solving module-wise optimization problems sequentially (i.e. one after the other fully) or in parallel (i.e. batch-wise, see Section 4.3.4), solves update locking (and so also backward locking). When combined with batch buffers, parallel module-wise training solves all three problems [Belilovsky et al., 2020] and allows parallel training of the modules. Module-wise training is appealing in memory-constrained settings as it works without storing all activations at the same time, and when done sequentially, only requires loading and training one module at a time.

Despite its simplicity, module-wise training has been shown to scale well [Belilovsky et al., 2020, Pyeon et al., 2021], outperforming more complicated methods addressing the locking problems e.g. synthetic [Jaderberg et al., 2017, Czarnecki et al., 2017] and delayed [Huo et al., 2018b, Huo et al., 2018a] gradients. We can also deduce theoretical results about networks of greedily-trained shallow modules [Belilovsky et al., 2019, Belilovsky et al., 2020, Arora et al., 2014, Malach and Shalev-Shwartz, 2018, Huang et al., 2018] from the existing results about the convergence of the training of shallow networks [Arora et al., 2018, Bach, 2017, Janzamin et al., 2016, Ge et al., 2018, Du and Goel, 2018].

The typical setting of (sequential) module-wise training for minimizing a loss L, is, given a dataset $\mathcal{D}$, to solve one after the other, for $1 \leq k \leq K$, Problems

$$(T_k, F_k) \in \arg\min_{T,F} \sum_{x \in \mathcal{D}} L(F, T \circ G_{k-1}(x)) \tag{20}$$

where $G_k = T_k \circ ... \circ T_1$ for $1 \leq k \leq K$ and $G_0 = \texttt{id}$. Here, T_k is the module (one or many layers) and it receives the output of module T_{k-1}, and F_k is an auxiliary classifier that processes the outputs of T_k so that the loss can be computed. The inputs are x and L has access to their labels y to calculate the loss, i.e. $L(F, T \circ G_{k-1}(x)) = l(F \circ T \circ G_{k-1}(x), y)$ where l is a machine learning loss such as cross-entropy or mean squared error. See Figure 16 for a representation of this training. The final network trained this way is $F_K \circ G_K$, but we can stop the forward pass at any depth k and keep the network $F_k \circ G_k$ for inference if it performs better.

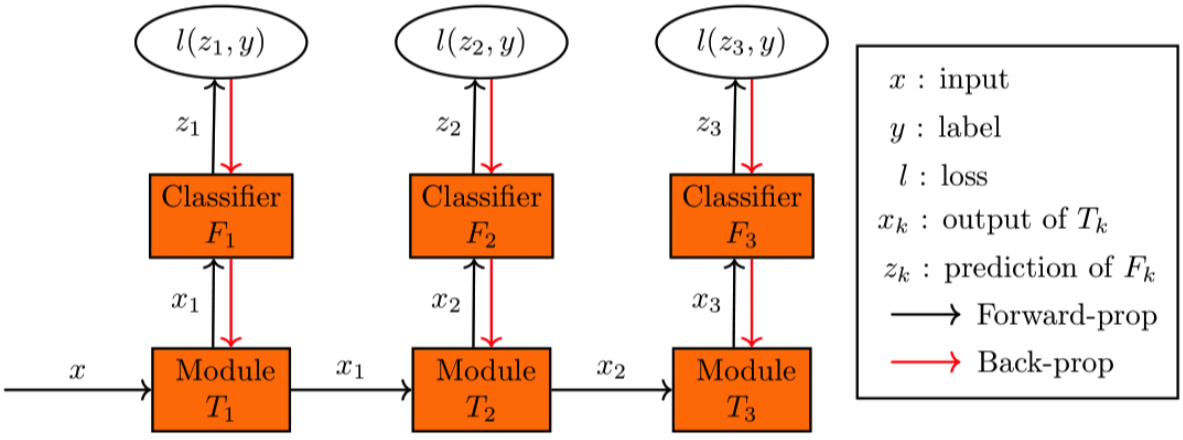

Figure 16: Module-wise training.

Indeed, module-wise training suffers often from a well-documented *stagnation problem* [Marquez et al., 2018, Belilovsky et al., 2019, Wang et al., 2021, Pyeon

et al., 2021], whereby greedy early modules overfit and learn more discriminative features than end-to-end training, and deeper modules don't improve the test accuracy, or even degrade it. [Wang et al., 2021] verify this and also find (see Figure 17 below) that early modules destroy too much information between the features they learn and the inputs, and between the features and the labels. Later in Section 4.5.6, we find a large degradation in the accuracy of deeper modules when training on small datasets. Our methods avoids this degradation.

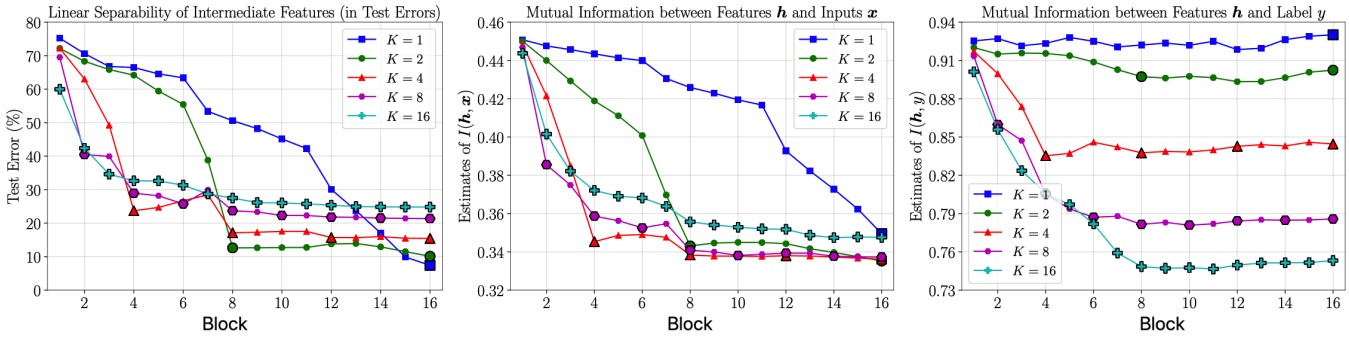

Figure 17: Linear separability (left), mutual information with input (center), and with output (right) of the features at different depths learned with module-wise training with K modules. The experiment is training a ResNet-32 on CIFAR10. K=1 is end-to-end training. K=2 means training two modules with 16 blocks each. From [Wang et al., 2021].

To solve this issue, InfoPro [Wang et al., 2021] maximizes the mutual information that each module keeps with the input, in addition to minimizing the loss. While this method shows good results in practice, the intuition behind it was criticized in [Du et al., 2021]. To avoid early overfitting, we propose a different, transport-based perspective, leveraging the analogy between ResNets and the Euler scheme for ODEs [Weinan, 2017]. To preserve input information, we minimize the kinetic energy of the modules along with the training loss. This has tight connections with the theories of gradient flows in distribution space and optimal transport. Our approach is particularly well-adapted to ResNets [He et al., 2016a, He et al., 2016b], which remain competitive to this day [Wightman et al., 2021], and similar models such as ResNeXts [Xie et al., 2017], Wide ResNets [Zagoruyko and Komodakis, 2016] and vision transformers that are made up essentially of residual connections [Liu et al., 2021, Dosovitskiy et al., 2021], but is immediately usable on other architectures where many layers have the same input and output dimension (which allows the computation of residues) such as VGG [Simonyan and Zisserman, 2014]. Our contributions are the following:

– We propose a new method for module-wise training. Being a regularization, it is easier to implement and lighter than many recent methods such as PredSim [Nøkland and Eidnes, 2019] and InfoPro [Wang et al., 2021], that train another auxiliary network besides the auxiliary classifier for each

module (similarity matching loss for PredSim and reconstruction loss for InfoPro to estimate the mutual information with the input).

- We theoretically justify our method, proving that it is a transport regularization that forces the module to be an optimal transport map making it more regular and stable. We also show that it amounts to a discretization of the gradient flow of the loss in probability space, which means that the modules progressively minimize the loss and explains why the method avoids the accuracy collapse observed in module-wise training.
- Additionally, we propose *multi-lap sequential* training, a variant of sequential module-wise training, that for the same time and number of epochs, and the same memory usage, performs better in many cases.
- Experimentally, we improve the test accuracy of module-wise trained networks (ResNets, VGG and Swin Transformer), both in sequential and parallel module-wise training, beating 8 other methods. Our method also works when the modules are few and deep, and when they are numerous and shallow, when many methods focus on the first setting. Our regularization makes parallel module-wise training superior or comparable in accuracy to end-to-end training, while consuming 10% to 60% less memory.

4.2 Related Work

Module-wise training methods. Local layer-wise training was initially considered as an early training algorithm for unsupervised learning [Hinton et al., 1995], as a pre-training and initialization method [Bengio et al., 2006, Marquez et al., 2018], and as a complement to end-to-end training to avoid vanishing gradients in the Inception architecture [Szegedy et al., 2017].

However, it was recently shown to be competitive with end-to-end training on supervised image classification tasks [Belilovsky et al., 2019, Nøkland and Eidnes, 2019]. This has led to it being considered in practical settings with limited resources such as embedded training on mobile devices [Teng et al., 2020, Tang et al., 2021] and dealing with very large whole slide images in histopathology [Zhang et al., 2022].

Many papers consider using a different auxiliary loss, instead of or in addition to the classification loss: kernel similarity [Mandar Kulkarni, 2016], pairwise losses [Duan et al., 2022], class separability [Lengellé and Denœux, 1996], information-theory-inspired losses [Xu and Principe, 1999, Löwe et al., 2019, Nguyen and Choi, 2019, Ma et al., 2020, Wang et al., 2021] and biologically plausible losses [Löwe et al., 2019, Nøkland and Eidnes, 2019, Gupta, 2020, Bernd Illing, 2020, Mostafa et al., 2018]. Paper [Belilovsky et al., 2019] reports the best experimental results when solving the layer-wise problems sequentially. Methods PredSim [Nøkland and Eidnes, 2019], DGL [Belilovsky et al., 2020], Sedona [Pyeon et al., 2021] and InfoPro [Wang et al., 2021] report the best results when solving the module-wise problems in parallel. [Belilovsky et al., 2019, Belilovsky et al., 2020] do it simply

through architectural considerations regarding the auxiliary networks. Sedona [Pyeon et al., 2021] uses architecture search to chose where to split the network into modules and what auxiliary classifier to use. We describe how some of these methods work in Appendix D.

However, [Belilovsky et al., 2019] do not consider ResNets and PredSim state that their method does not perform well on them. Sedona applies architecture search to decide on where to split the network into up to 16 modules and what auxiliary classifier to use before module-wise training. Only BoostResNet [Huang et al., 2018] also proposes a block-wise training idea geared for ResNets. However, their results only show better early performance on limited experiments and end-to-end fine-tuning is required to be competitive. These methods can all be combined with our regularization, and we use the auxiliary architecture from [Belilovsky et al., 2019, Belilovsky et al., 2020].

In more recent developments, [Patel et al., 2022] attempts to also split the layers width-wise into groups of neurons and [Ren et al., 2023, Fournier et al., 2023] combine layer-wise training with forward gradient learning [Baydin et al., 2022], another alternative to end-to-end backpropagation. Finally, we mention that module-wise training has been extended to the training of spiking neural networks [Guo et al., 2022b] and of neural search networks [Chen et al., 2022].

Reduced backpropagation methods. Some methods allow some backward communication between the modules to improve performance, but therefore limit the computational advantages of module-wise training. For example, [Guo et al., 2022a] allow some backpropagation from a module to the last few layers of the preceding module and [Xiong et al., 2020, Gomez et al., 2020, Laskin et al., 2021] make the last layer in a module and the first in the next module the same.

Delayed and predicted gradient methods. Besides module-wise training, methods such as DNI [Jaderberg et al., 2017, Czarnecki et al., 2017], DDG [Huo et al., 2018b], FR [Huo et al., 2018a] and ADL [Zhuang et al., 2021], solve the update and backward locking problems with an eye towards parallelization by using delayed or predicted gradients, or even predicted inputs to address forward locking, which is what [Sun et al., 2021] do. But they only split networks into a small number of sub-modules (less than five) that don't backpropagate to each other and observe training issues with more sub-modules [Huo et al., 2018a]. This makes them compare unfavorably to module-wise training [Belilovsky et al., 2020]. The high dimension of the predicted gradient which scales with the size of the network renders [Jaderberg et al., 2017, Czarnecki et al., 2017] challenging in practice. Therefore, despite its simplicity, greedy module-wise training is more appealing when working in a constrained setting.

Other alternatives to end-to-end backpropagation. Other similar approaches include ResIST [Dun et al., 2021] and DPA [Ni et al., 2022]. Resist randomly assigns residual blocks to modules that are trained independently and reassembled before another random partition. More of a distributed training

method, it is only compared with local SGD [Stich, 2019]. DPA surrounds modules by fixed small pre-trained layers and trains them before assembling them. See [Duan and Príncipe, 2022] for a survey on alternatives to end-to-end training.

Wasserstein gradient flows in the data space. Wasserstein gradient flows operating in the data space recently appeared in deep learning. In [Alvarez-Melis and Fusi, 2021], the focus is on functionals of measures whose first variations are known in closed form and used, through their gradients, in the algorithm. This limits the scope of their applications to transfer learning and similar tasks. Likewise, [Gao et al., 2019, Liutkus et al., 2019, Arbel et al., 2019, Ansari et al., 2021] use the continuous gradient flow of f-divergences and other distances between measures for generation and generator refinement. In contrast, we use the discrete minimizing movement scheme which does not require computation of the first variation and allows to consider classification.

4.3 Method and Theory

In this section we state the module-wise optimization problems we solve and show that successively solving these problems means following a minimizing movement scheme in distribution space that maximizes the separability of the embedding distributions. We then show that the solution modules exist and have some regularity as they are optimal transport maps. Appendices B and C give the necessary background on optimal transport and gradient flows.

4.3.1 Formulation

To keep greedily-trained modules from overfitting and destroying information needed later, we penalize their kinetic energy to force them to preserve the geometry of the problem. If each module is a single ResBlock (i.e. a function $T = \texttt{id}+r$), its kinetic energy is simply the squared norm of its residue $r = T-\texttt{id}$, which we add to the loss L in the target of the greedy problems (20). Given $\tau>0$ used to weight the regularization, we now solve, for $1\leq k\leq K$, Problems

$$(T_k^\tau, F_k^\tau) \in \arg\min_{T,F} \sum_{x\in\mathcal{D}} L(F, T\circ G_{k-1}^\tau(x)) + \frac{1}{2\tau}\|T\circ G_{k-1}^\tau(x) - G_{k-1}^\tau(x)\|^2 \quad (21)$$

where $G_k^\tau = T_k^\tau \circ .. \circ T_1^\tau$ for $1\leq k\leq K$ and $G_0^\tau = \texttt{id}$. The final network is now $F_K^\tau \circ G_K^\tau$. Intuitively, we can think that this biases the modules towards moving the points as little as possible, thus at least keeping the performance of the previous module. We focus on ResNets because they are already biased towards small displacements and that this bias is desirable and should be encouraged [Jastrzebski et al., 2018, Zhang et al., 2019, Hauser, 2019, De and Smith, 2020], and because $T(x)-x$ can always be computed as both terms have the same dimension. But the method can be applied to any module where this quantity can be computed.

To facilitate the theoretical analysis, we rewrite the method in a more general formulation using data distribution ρ, which can be discrete or continuous,

and the distribution-wide loss $\mathcal{L}$ that arises from the point-wise loss L. Then Problem (21) is equivalent in the discrete case to the following problem

$$(T_k^\tau, F_k^\tau) \in \arg\min_{T,F} \ \mathcal{L}(F, T_\sharp \rho_k^\tau) + \frac{1}{2\tau} \int_\Omega \|T(x) - x\|^2 \, \mathrm{d}\rho_k^\tau(x) \tag{22}$$

with $\rho_{k+1}^\tau = (T_k^\tau)_\sharp \rho_k^\tau$ and $\rho_1^\tau = \rho$, i.e data embedding distributions ρ_k^τ are pushed forward by maps T_k^τ, and $\mathcal{L}(F, T_\sharp \rho_k^\tau) = \int L(F, T(x)) \, \mathrm{d}\rho_k^\tau(x) = \int L(F, .) \, \mathrm{d}T_\sharp \rho_k^\tau$. The difference with Problem (16) in Section 3, is that now we have a regularized problem instead of a problem with a hard constraint. But we will show in Section 4.3.4 that this still forces the solution to be an optimal transport map.

4.3.2 Background on the Minimizing Movement Scheme

Our main result is that solving problems (22) successively means following a *minimizing movement scheme* (MMS) in distribution space for minimizing $\mathcal{Z}(\mu) = \min_F \mathcal{L}(F, \mu)$, which represents the loss of the best classifier on distribution μ. If we restrict ourselves to linear classifiers, $\mathcal{Z}(\rho_k^\tau)$ represents the linear separability of representation ρ_k^τ at module k of the data distribution ρ. When auxiliary networks are not necessary, for example in generative tasks where the output and the target have the same shape, $\mathcal{Z}$ is simply $\mathcal{L}$. When the auxiliary network F is pre-trained and fixed, $\mathcal{Z} = \mathcal{L}(F, \mu)$. The MMS was introduced in [Giorgi et al., 1980, Gianazza et al., 1994] as a metric counterpart to Euclidean gradient descent for minimizing functionals defined on metric spaces. The task we want to solve is minimizing $\mathcal{Z}$ in the Wasserstein space. We define the minimizing movement scheme in Definition 3. See Appendix C for more details.

We work in the metric Wasserstein space $\mathbb{W}_2(\Omega) = (\mathcal{P}(\Omega), W_2)$, where $\Omega \subset \mathbb{R}^d$ is a convex compact set, $\mathcal{P}(\Omega)$ is the set of probability distributions over Ω and W_2 is the Wasserstein distance over $\mathcal{P}(\Omega)$ derived from the optimal transport problem with Euclidean cost:

$$W_2^2(\mu, \nu) = \min_{T \text{ s.t. } T_\sharp \mu = \nu} \int_\Omega \|T(x) - x\|_2^2 \, \mathrm{d}\mu(x) \tag{23}$$

where we further assume that $\partial\Omega$ is negligible and that the distributions we are dealing with are absolutely continous.

Definition 3. Given $\mathcal{Z} : \mathbb{W}_2(\Omega) \to \mathbb{R}$, the minimizing movement scheme is a discretized gradient flow that is well-defined in non-Euclidean metric spaces and minimizes (under some conditions) $\mathcal{Z}$ starting from $\rho_1^\tau \in \mathcal{P}(\Omega)$. It is given by

$$\rho_{k+1}^\tau \in \arg\min_{\rho \in \mathcal{P}(\Omega)} \ \mathcal{Z}(\rho) + \frac{1}{2\tau} W_2^2(\rho, \rho_k^\tau) \tag{24}$$

Equation (24) can be seen as a proximal step in the Wasserstein distance for minimizing $\mathcal{Z}$. Equation (24) can also be interpreted as a non-Euclidean version

of the implicit Euler scheme for following the gradient flow of $\mathcal{Z}$. Indeed, the Euclidean version of (24) for $F : \mathbb{R}^d \to \mathbb{R}$ is

$$x_{k+1}^\tau \in \arg\min_{x\in\mathbb{R}^d} \; F(x) + \frac{1}{2\tau}\|x - x_k^\tau\|^2 \tag{25}$$

And the optimality condition of problem (25) is

$$\frac{x_{k+1}^\tau - x_k^\tau}{\tau} = -\nabla F(x_{k+1}^\tau) \tag{26}$$

which is exactly the formula of the implicit Euler scheme (see Appendix A.1) with step size τ for the Cauchy problem of continuous-time gradient descent:

$$x'(t) = -\nabla F(x(t)) \tag{27}$$

A curve x solution to (27) is called a *gradient flow* of F and under convexity assumptions on F tends to a minimizer of F as t grows. Under more conditions on F, the minimizing movement scheme (equivalently the implicit Euler scheme in the Euclidean case), converges as τ goes to 0 to a gradient flow x of F (i.e. to a solution of $x'(t) = -\nabla F(x(t))$). Therefore if both sets of assumptions on F are satisfied, x_k^τ converges to a minimizer of F as t tens to infinity and τ tends to 0. The advantage of the minimizing movement scheme formulation (25) compared to the implicit Euler scheme (26) is that it does not require the computation of a gradient and therefore can be more easily extended to non-Euclidean metric spaces such as the Wasserstein space in (24).

In the Wasserstein case, if $\mathcal{Z}$ is lower-semi continuous then problems (24) always admit a solution because $\mathcal{P}(\Omega)$ is compact. If $\mathcal{Z}$ is also λ-geodesically convex for $\lambda{>}0$, we have convergence of ρ_k^τ as $k{\to}\infty$ and $\tau{\to}0$ to a minimizer of $\mathcal{Z}$, potentially under more technical conditions (see Appendix C for details). Even though a machine learning loss will usually not satisfy these conditions, this analyses offers hints as to why our method avoids in practice the problem of stagnation or collapse in performance of module-wise training along the depth k, as we are making proximal local steps in Wasserstein space to minimize the loss. This convergence discussion also suggests taking τ as small as possible and many modules T_k^τ in practice.

4.3.3 Link with the Minimizing Movement Scheme

So under the mentioned assumptions on Ω in the second paragraph of Section 4.3.2 and absolute continuity of the distributions, we have:

Proposition 3. The distributions $\rho_{k+1}^\tau = (T_k^\tau)_\sharp \rho_k^\tau$, where the functions T_k^τ are found by solving (22) and $\rho_1^\tau = \rho$ is the data distribution, coincide with the minimizing movement scheme (24) for $\mathcal{Z} = \min_F \mathcal{L}(F, .)$.

Proof. The minimizing movement scheme (24) is equivalent to taking $\rho_{k+1}^\tau = (T_k^\tau)_\sharp \rho_k^\tau$ where

$$T_k^\tau \in \arg\min_{T:\Omega\to\Omega} \mathcal{Z}(T_\sharp \rho_k^\tau) + \frac{1}{2\tau} W_2^2(T_\sharp \rho_k^\tau, \rho_k^\tau) \tag{28}$$

under conditions that guarantee the existence of a transport map between ρ_k^τ and any other measure. Absolute continuity of ρ_k^τ suffices for this, and the loss can ensure that ρ_{k+1}^τ is absolutely continuous. Among the functions T_k^τ that solve problem (28), is the optimal transport map from ρ_k^τ to ρ_{k+1}^τ. To solve specifically for this optimal transport map, we have to solve the equivalent Problem

$$T_k^\tau \in \arg\min_T \mathcal{Z}(T_\sharp \rho_k^\tau) + \frac{1}{2\tau}\int_\Omega \|T(x) - x\|^2 \,\mathrm{d}\rho_k^\tau(x) \tag{29}$$

Problems (28) and (29) have the same minimum value, but the minimizer of (29) is now an optimal transport map between ρ_k^τ and ρ_{k+1}^τ. This is immediate from the definition (23) of the W_2 distance. Equivalently minimizing first over the auxiliary F in (22), and from the definition of $\mathcal{Z}$, Problems (22) and (29) are equivalent, which concludes. □

Since we solve Problems (22) over neural networks, their representation power shown by universal approximation theorems [Cybenko, 1989, Hornik et al., 1989] is important to get close to equivalence between (24) and (22), as we need to approximate an optimal transport map. We also know that the training of each module, if it is shallow, converges [Arora et al., 2018, Bach, 2017, Janzamin et al., 2016, Ge et al., 2018, Du and Goel, 2018].

4.3.4 Regularity Result

As in Section 3, we can show, by adapting the proof of Theorem 2, that Problem (22) has a solution and that the solution module T_k^τ is an optimal transport map between its input and output distributions, which means that it comes with some regularity. The difference is that we have a regularized problem instead of a hard constraint. We assume that the minimization in F is over a compact set $\mathcal{F}$, that ρ_k^τ is absolutely continuous, that $\mathcal{L}$ is continuous and non-negative, that Ω is convex and compact and that $\partial\Omega$ is negligible. We then have

Theorem 5. Problem (22) has a minimizer (T_k^τ, F_k^τ) such that T_k^τ is an optimal transport map. For any minimizer (T_k^τ, F_k^τ), T_k^τ is an optimal transport map.

Proof. Take a minimizing sequence $(\tilde{F}_i, \tilde{T}_i)$, i.e. such that $\mathcal{C}(\tilde{F}_i, \tilde{T}_i) \to \min \mathcal{C}$, where $\mathcal{C} \geq 0$ is the target function in (22) and denote $\beta_i = \tilde{T}_{i\sharp}\rho_k^\tau$. Then by compacity $\tilde{F}_i \to F^\star$ and $\beta_i \rightharpoonup \beta^\star$ in duality with $\mathcal{C}_b(\Omega)$ by Banach-Alaoglu. There exists $T^\star$ an optimal transport map between ρ_k^τ and $\beta^\star$ and we have that $\mathcal{C}(F^\star, T^\star) \leq \lim_{i\to\infty} \mathcal{C}(\tilde{F}_i, \tilde{T}_i) = \min \mathcal{C}$ by continuity of $\mathcal{L}$ and because

$$\begin{aligned}\int_\Omega \|T^\star(x) - x\|^2 \,\mathrm{d}\rho_k^\tau(x) = W_2^2(\rho_k^\tau, \beta^\star) &= \lim_{i\to\infty} W_2^2(\rho_k^\tau, \beta_i) \\ &\leq \lim_{i\to\infty} \int_\Omega \|\tilde{T}_i(x) - x\|^2 \,\mathrm{d}\rho_k^\tau(x)\end{aligned}$$

as W_2 metrizes weak convergence of measures. We take (F_k^τ, T_k^τ)=$(F^\star, T^\star)$. It is also immediate that for any minimizing pair, the transport map is optimal. □

As in Section 3, regularity properties follow for the solution modules. Given Theorem 1 in Appendix B and taken from [Figalli, 2017], T_k^τ is η-Hölder continuous almost everywhere and if the optimization algorithm we use to solve the discretized problem (21) returns an approximate solution pair $(\tilde{F}_k^\tau, \tilde{T}_k^\tau)$ such that $\tilde{T}_k^\tau$ is an ϵ-optimal transport map, i.e. $\|\tilde{T}_k^\tau - T_k^\tau\|_\infty \leq \epsilon$, then we have (using the triangle inequality) the following stability property of the module $\tilde{T}_k^\tau$:

$$\|\tilde{T}_k^\tau(x) - \tilde{T}_k^\tau(y)\| \leq 2\epsilon + C\|x-y\|^\eta \tag{30}$$

for almost every $x, y \in \text{supp}(\rho_k^\tau)$ and a constant $C > 0$. As we have seen in Section 3, these networks generalize better and overfit less in practice. Naively composing these stability bounds on T_k^τ and $\tilde{T}_k^\tau$ allows to get stability bounds for the composition networks G_k^τ and $\tilde{G}_k^\tau = \tilde{T}_k^\tau \circ .. \circ \tilde{T}_1^\tau$.

To summarize Section 4.3, the transport regularization makes each module more regular and it allows the modules to build on each other as k increases to solve the task, which is the property we desire in module-wise training.

4.4 Practical implementation

4.4.1 Multi-block Modules

For simplicity, we presented in (21) the case where each module is a single ResBlock. However, in practice, we often split the network into modules that contain many ResBlocks each. We show here that regularizing the kinetic energy of such modules still amounts to a transport regularization, which means that Theorem 5, the regularity bound (30) and the link with gradient flows still apply.

If each module T_k is made up of M ResBlocks, i.e. applies $x_{m+1} = x_m + r_m(x_m)$ for $0 \leq m < M$, we have seen in Section 3 that $\sum \|r_m(x_m)\|^2$ is its discrete kinetic energy. If ϕ_m^x denotes the position of a point x after m ResBlocks, then regularizing the kinetic energy of multi-block modules means solving

$$\begin{aligned}(T_k^\tau, F_k^\tau) \in \arg\min_{T,F} \sum_{x\in\mathcal{D}} (L(F, T(G_{k-1}^\tau(x)) + \frac{1}{2\tau}\sum_{m=0}^{M-1} \|r_m(\phi_m^x)\|^2) \\ \text{s.t. } T = (\texttt{id} + r_{M-1}) \circ ... \circ (\texttt{id} + r_0),\ \phi_0^x = G_{k-1}^\tau(x) \\ \phi_{m+1}^x = \phi_m^x + r_m(\phi_m^x)\end{aligned} \tag{31}$$

We also want to minimize this sum of squared residue norms instead of $\|T(x)-x\|^2$ (the two no longer coincide) as it works better in practice, which we assume is because it offers a better and more localized control of the transport. As expressed in (6), a ResNet can be seen as an Euler discretization of a differential equation and this new Problem (31) is then the discretization of Problem

$$\begin{aligned}(T_k^\tau, F_k^\tau) \in \arg\min_{T,F}\ \mathcal{L}(F, T_\sharp\rho_k^\tau) + \frac{1}{2\tau}\int_0^1 \|v_t\|^2_{L^2((\phi_t^\cdot)_\sharp\rho_k^\tau)}\,\mathrm{d}t \\ \text{s.t. } T = \phi_1^\cdot,\ \partial_t\phi_t^x = v_t(\phi_t^x),\ \phi_0^\cdot = \text{id}\end{aligned} \tag{32}$$

where $\rho_{k+1}^\tau = (T_k^\tau)_\sharp \rho_k^\tau$ and r_m is the discretization of vector field v_t at time $t = m/M$. Here, distributions ρ_k^τ are pushed forward through the maps T_k^τ which correspond to the flow ϕ at time $t = 1$ of the kinetically-regularized velocity field v_t. We recognize in the second term in the target of (32) the optimal transport problem in its dynamic formulation [Benamou and Brenier, 2000], and given the equivalence between the Monge OT problem (23) and the dynamic OT problem (13) in Appendix B, Problem (32) is in fact equivalent to the original continuous formulation (22), and the theoretical results in Section 4.3 follow (Proposition 3 follows immediately and take a minimizing sequence $(\tilde{F}_i, \tilde{v}^i)$ and the corresponding maps $\tilde{T}_i$ as in the proof of Theorem 5 to get the result).

4.4.2 Solving the Module-wise Problems

The module-wise problems can be solved in one of two ways. In *sequential* module-wise training, we completely train each module with its auxiliary classifier for N epochs before training the next module, which receives as input the output of the previous trained module. In *parallel* module-wise training, the modules are trained synchronously batch-wise, i.e. we do a complete forward pass on each batch but without a full backward pass, rather a backward pass that only updates the current module T_k^τ and its auxiliary classifier F_k^τ, meaning that T_k^τ forwards its output to T_{k+1}^τ immediately after it computes it. This is called *decoupled* greedy training in [Belilovsky et al., 2020], which shows that combining it with batch buffers solves all three locking problems and allows a linear training parallelization in the depth of the network. Since it is observed that parallel module-wise training performs better than sequential training (see [Wang et al., 2021] and Section 4.5.5 for example), we propose a variant of sequential module-wise training that we call *multi-lap sequential* module-wise training that introduces more synchronicity in sequential training. Here, instead of training each module for N epochs, we train each module from the first to the last sequentially for N/R epochs, then go back and train from the first module to the last for N/R epochs again, and we do this for R laps. For the same total number of epochs and training time, and the same advantages (loading and training one module at a time) this provides a non-negligible improvement in accuracy over normal sequential module-wise training in most cases, as shown in Section 4.5.5. Despite our theoretical framework being that of sequential module-wise training, our method improves the test accuracy of all three module-wise training regimes.

4.4.3 Varying the Regularization Weight

The discussion in Section 4.3.3 suggests taking a fixed τ that is as small as possible. However, instead of using a fixed τ, we might want to vary it along the depth k to further constrain with a smaller τ_k the earlier modules to avoid that they overfit or the later modules to maintain the accuracy of earlier modules. We might also want to regularize the networks further in earlier epochs when the data is more entangled.

To unify and formalize this varying weight $\tau_{k,i}$ across modules k and SGD iterations i, we use a scheme inspired by the method of multipliers as in Section 3.4 to solve Problems (21) and (31). To simplify the notations, we will instead consider the weight $\lambda_{k,i}{:=}2\tau_{k,i}$ given to the loss. We denote $\theta_{k,i}$ the parameters of both T_k and F_k at SGD iteration i. We also denote $L(\theta, x)$ and $C(\theta, x)$ respectively the loss and the transport regularization as functions of parameters θ and data x. We now increase the weight $\lambda_{k,i}$ of the loss every s iterations of SGD by a value that is proportional to the current loss. Given increase factor $h{>}0$, initial parameters $\theta_{k,1}$, initial weights $\lambda_{k,1}{\geq}0$, learning rates (η_i) and batches (x_i), we apply for module k and $i{\geq}1$:

$$\begin{cases} \theta_{k,i+1} & = \theta_{k,i} - \eta_i \nabla_\theta (\lambda_{k,i}\ L(\theta_{k,i}, x_i) + C(\theta_{k,i}, x_i)) \\ \lambda_{k,i+1} & = \lambda_{k,i} + hL(\theta_{k,i+1}, x_{i+1}) \text{ if } i \bmod s = 0 \text{ else } \lambda_{k,i} \end{cases} \tag{33}$$

The weights $\lambda_{k,i}$ will vary along modules k even if we use the same initial weights $\lambda_{k,1} = \lambda_1$ because they will evolve differently with iterations i for each k. They will increase more slowly with i for larger k because deeper modules will have smaller loss. This method can be seen as a method of multipliers for the problem of minimizing the transport under the constraint of zero loss. Therefore it is immediate by slightly adapting the proof of Theorem 5 or from Section 3 that we are still solving a problem that admits a solution whose non-auxiliary part is an optimal transport map with the same regularity as stated above. We use the same initial value $\lambda_1 = \lambda_{k,1}$ for all modules so that this method requires choosing three hyper-parameters (h, s and λ_1), and the number of hyper-parameters does not go up as the number of modules increases.

In practice (see Section 4.5.1 and Appendix G.1), this algorithm works best in only one experiment, but its behavior suggested the following heuristic which we use in the other experiments: use a value τ for the first $K/2$ modules and double it (i.e use 2τ) for the last $K/2$ modules. A single hyper-parameter has then to be chosen.

4.5 Experiments

We test our method on classification tasks, L being cross-entropy. We call our method TRGL for Transport-Regularized Greedy Learning. For the auxiliary classifiers, we use the architecture from DGL [Belilovsky et al., 2019, Belilovsky et al., 2020], that is a convolutional layer followed by an average pooling layer and a fully connected layer. This auxiliary classifier architecture is also very similar to that used by InfoPro [Wang et al., 2021]. The code is available at `github.com/block-wise/module-wise` and implementation details are in Appendix G.1.

We focus first on parallel module-wise training in Section 4.5.1 as it performs better in practice and has therefore been more explored in recent works. The methods we compare to are:

- VanGL is simply vanilla greedy module-wise training with the same architecture but without our regularization. We include its results for ablation study purposes.
- InfoPro [Wang et al., 2021] which maximize the mutual information each module keeps with the input to the network in addition to the classification loss. This therefore requires an additional auxiliary network besides the classifier. This method has two variants, depending on whether the cross-entropy loss (InfoPro Softmax) is used for the auxiliary classifiers, or a contrastive loss (InfoPro Softmax) inspired by [Chen et al., 2020b, Khosla et al., 2020, He et al., 2020].
- InfoProL [Pathak et al., 2022], which is a variant of InfoPro that maximize the mutual information each module keeps with its own input, instead of the input to the network. This requires slightly less memory.
- Sedona [Pyeon et al., 2021], which uses an architecture search phase to decide where to split the network into modules and which auxiliary classifier to use. This leads to larger early modules, reducing the memory savings of module-wise training.
- DGL [Belilovsky et al., 2020], which only focuses on the architecture of the auxiliary classifier, and whose auxiliary classifier we use.
- PredSim [Nøkland and Eidnes, 2019], which adds a similarity matching loss to the loss of every module, requiring a second auxiliary network.
- DDG [Huo et al., 2018b] and FR [Huo et al., 2018a], which are delayed gradient methods that aim to break the locking problems for parallelization and not for saving memory.

We describe how some of these methods work in Appendix D. We find that TRGL always has a better test accuracy than the other methods. Except on Transformers, is has a better test accuracy than end-to-end training, with as much as almost 60% less memory usage.

We then run experiments on sequential module-wise training in Section 4.5.5 with each module being a single residual block, which allows for the largest memory savings, as only one block and its classifier have to be loaded at a time. We reach comparable performances to [Belilovsky et al., 2019].

Finally, we verify in Section 4.5.6 that our method does avoid the early overfitting and subsequent stagnation or collapse in test accuracy.

4.5.1 Parallel Module-wise Training with Few Modules

The first experiment is training in parallel 3 residual architectures and a VGG-19 [Simonyan and Zisserman, 2014] divided into 4 modules on TinyImageNet. We compare in Table 4 below our results in this setting to three of the best recent parallel module-wise training methods: DGL [Belilovsky et al., 2020], PredSim

[Nøkland and Eidnes, 2019] and Sedona [Pyeon et al., 2021], from Table 2 in [Pyeon et al., 2021]. We find that our method has a much better test accuracy than the three other methods, especially on the smaller architectures. It also performs better than end-to-end training on the three ResNets. Parallel TRGL in this case with 4 modules consumes 10 to 21% less memory than end-to-end training.

Table 4: Accuracy of Parallel TRGL with 4 modules (average and 95% confidence over 5 runs) on TinyImageNet, compared to DGL, PredSim, Sedona and E2E from Table 2 in [Pyeon et al., 2021], with memory saved compared to E2E as a percentage of E2E memory in red.

Dataset	Architecture	K	Parallel VanGL	Parallel TRGL (ours)	PredSim	DGL	Sedona	E2E
TinyImageNet	VGG-19	4	56.17 ± 0.29 (↓ 27%)	**57.28** ± 0.20 (↓ 21%)	44.70	51.40	56.56	58.74
	ResNet-50	4	58.43 ± 0.45 (↓ 26%)	**60.30** ± 0.58 (↓ 20%)	47.48	53.96	54.40	58.10
	ResNet-101	4	63.64 ± 0.30 (↓ 24%)	**63.71** ± 0.40 (↓ 11%)	53.92	53.80	59.12	62.01
	ResNet-152	4	63.87 ± 0.16 (↓ 21%)	**64.23** ± 0.14 (↓ 10%)	51.76	57.64	64.10	62.32

The second experiment is training in parallel two residual architectures divided into 2 modules on CIFAR100 [Krizhevsky, 2009]. We compare in Table 5 our results in this setting to the two delayed gradient methods DDG [Huo et al., 2018b] and FR [Huo et al., 2018a] from Table 2 in [Huo et al., 2018a]. Here again, parallel TRGL has a better accuracy than the other two methods and than end-to-end training. With only two modules, the memory gains from less backpropagation are neutralized by the weight of the extra classifier and the memory savings compared to end-to-end training are negligible, but TRGL has a better accuracy by up to almost 2 percentage points.

Table 5: Accuracy of Parallel TRGL with 2 modules (average and 95% confidence over 3 runs) on CIFAR100, compared to DDG, FR and E2E from Table 2 in [Huo et al., 2018a].

Dataset	Architecture	K	Par VanGL	Par TRGL (ours)	DDG	FR	E2E
CIFAR100	ResNet-101	2	77.31 ± 0.27	**77.87** ± 0.44	75.75	76.90	76.52
	ResNet-152	2	75.40 ± 0.75	**76.55** ± 1.90	73.61	76.39	74.80

The third experiment is training in parallel a ResNet-110 divided into two, four, eight and sixteen modules on STL10 [Coate et al., 2011]. We compare in Table 6 our results in this setting to the recent methods InfoPro [Wang et al., 2021] and DGL [Belilovsky et al., 2020] from Table 2 in [Wang et al., 2021]. Our TRGL clearly outperforms the other methods. It also outperforms end-to-end training in all but one case (that with 16 modules). Memory savings of parallel TRGL compared to end-to-end training (with a batch size of 64) reach around 48% and 58.5% with 8 and 16 modules respectively, with comparable test accuracy. With four modules, the TRGL training weighs 24% less than end-to-end-training, and has a test accuracy that is better by 2 percentage points (see Section 4.5.3 below for more details and comparison to the memory saving of InfoPro).

Table 6: Accuracy of Parallel TRGL with K modules (average and 95% interval over 5 runs) on STL10, compared to DGL, InfoPro and E2E from Table 2 in [Wang et al., 2021].

Dataset	Architecture	K	Par VanGL	Par TRGL (ours)	DGL	InfoPro Softmax	InfoPro Contrast	E2E
STL10	ResNet-110	2	79.85 ± 0.93	**80.04** ± 0.85	75.03 ± 1.18	78.98 ± 0.51	79.01 ± 0.64	77.73 ± 1.61
		4	77.11 ± 2.31	**79.72** ± 0.81	73.23 ± 0.64	78.72 ± 0.27	77.27 ± 0.40	77.73 ± 1.61
		8	75.71 ± 0.55	**77.82** ± 0.73	72.67 ± 0.24	76.40 ± 0.49	74.85 ± 0.52	77.73 ± 1.61
		16	73.57 ± 0.95	**77.22** ± 1.20	72.27 ± 0.58	73.95 ± 0.71	73.73 ± 0.48	77.73 ± 1.61

The fourth experiment is training (from scratch) in parallel a Swin-Tiny Transformer [Liu et al., 2021] divided into 4 modules on three datasets. We compare in Table 7 our results in this setting with those of InfoPro [Wang et al., 2021] and InfoProL, a variant of InfoPro proposed in [Pathak et al., 2022], from Table 3 in [Pathak et al., 2022]. TRGL outperforms the other module-wise training methods. It does not outperform end-to-end training in this case, but consumes 30% less memory on CIFAR10 and CIFAR100 and 50% less on STL10, compared to 38% for InfoPro and 45% for InfoProL in [Pathak et al., 2022].

Table 7: Test accuracy of parallel TRGL with 4 modules (average and 95% confidence interval over 5 runs) on a Swin-Tiny Transformer, compared to InfoPro, InfoProL and E2E from Table 3 in [Pathak et al., 2022], with memory saved compared to E2E as a percentage of E2E memory consumption in red.

Architecture	Dataset	K	Parallel VanGL	Parallel TRGL (ours)	InfoPro	InfoProL	E2E
Swin-Tiny	STL10	4	67.00 ± 1.36 (↓ 55%)	**67.92** ± 1.12 (↓ 50%)	64.61 (↓ 38%)	66.89 (↓ 45%)	72.19
	CIFAR10	4	83.94 ± 0.42 (↓ 33%)	**86.48** ± 0.54 (↓ 29%)	83.38 (↓ 38%)	86.28 (↓ 45%)	91.37
	CIFAR100	4	69.34 ± 0.91 (↓ 33%)	**74.11** ± 0.31 (↓ 29%)	68.36 (↓ 38%)	73.00 (↓ 45%)	75.03

To show that our method works well with all types of module-wise training when using few modules, we train a ResNet-101 split in 2 modules on CIFAR100, sequentially and multi-lap sequentially. In Table 8, we see that our idea of multi-lap sequential training adds one percentage point of accuracy to sequential training, and that the regularization further improves the accuracy by about half a percentage point. As only one module has to be trained at a time, this requires only around half the memory end-to-end training requires.

Table 8: Test accuracy of sequential (Seq) and multi-lap sequential (MLS) TRGL and VanGL with 2 modules on CIFAR100 using ResNet-101 (average of 2 runs).

Seq VanGL	Seq TRGL	MLS VanGL	MLS TRGL
73.31	73.61	74.34	**74.78**

4.5.2 Ablation Study and Sensitivity to Hyperparameter τ

We see in the Tables 4, 5 and 6 in Section 4.5.1 that the improvement in accuracy from the regularization compared to vanilla module-wise training (VanGL) tends

to increase as the number of modules increases and reaches almost 4 percentage points with 16 modules. We further show in Figure 43 in Appendix G.4 that our method is not very sensitive to the choice of τ over a large scale of values.

4.5.3 Memory Usage of Parallel TRGL

As seen above, parallel TRGL is lighter than end-to-end training by up to more than 50%. We compare here the memory needs of our method to that of InfoPro [Wang et al., 2021] on a ResNet-110 split into K modules trained in parallel on STL10 with a batch size of 64 (so the same setting as in Table 6).

InfoPro [Wang et al., 2021] also proposes to split the network into K modules that have the same weight but not necessarily the same number of layers, which leads to even less memory usage. Indeed, the earlier layers of a neural networks usually have more parameters and are heavier. Therefore, by counting the number of parameters and making the earlier modules shallower than the deeper ones we can split the network into modules that are equally heavy instead of equally deep. InfoPro only implement this for $K \leq 4$ modules. When the modules are even in weight and not in depth, keeping the notations of [Wang et al., 2021], we call the training methods VanGL*, TRGL* and InfoPro*. As said above, this leads to shallower early modules which hurts performance according to [Pyeon et al., 2021]. We verify that this degradation is slight with our regularization in Table 30 in Appendix G.3 (to be compared with Table 6), and that TRGL* still outperforms VanGL*, InfoPro* and end-to-end training.

We report in Table 9 the memory saved as a percentage of the 6230 MiB memory required by end-to-end training with the same batch size. We see in Table 9 that TRGL saves more memory than InfoPro in two out of three cases (4 and 8 modules), and about the same in the third case (16 modules), with much better test accuracy in all cases. Likewise, TRGL* is lighter than InfoPro*, with better accuracy. We also see that the added memory cost of the regularization compared to vanilla greedy learning (VanGL) is small.

Table 9: Memory savings using a ResNet-110 on STL10 split into K modules trained in parallel with a batch size of 64, as a percentage of the weight of end-to-end training. Average test accuracy over 5 runs is between brackets. Test accuracy of end-to-end training is 77.73%.

	Equally deep modules			Equally heavy modules		
K	Par VanGL	Par TRGL (ours)	InfoPro	Par VanGL*	Par TRGL* (ours)	InfoPro*
4	27% (77.11)	24% (**79.72**)	18% (78.72)	41% (77.14)	39% (**78.94**)	33 % (78.78)
8	50% (75.71)	48% (**77.82**)	37% (76.40)			
16	61% (73.57)	58% (**77.22**)	59% (73.95)			

We summarize in Table 31 in Appendix G.3, memory savings of parallel TRGL with 4 modules that range between 8 and 30% of E2E memory, on the 4 networks trained on TinyImageNet in Table 4 and a the Swin Transformer from Table 7. Note that methods DDG [Huo et al., 2018b] and FR [Huo et al., 2018a],

being delayed gradient methods and not module-wise training methods, do no save memory (they actually increase memory usage, see FR [Huo et al., 2018a]). Sedona [Pyeon et al., 2021] also does not claim to save memory, as their first module (the heaviest) is deeper than the others, but rather to speed up computation. Finally, DGL [Belilovsky et al., 2020] is architecture-wise essentially identical to VanGL and consumes the same amount of memory.

4.5.4 Training Time of Parallel TRGL

Since we do not implement forward unlocking with batch buffers as in DGL (i.e. only the backward passes of the modules happen in parallel), parallel module-wise training does slightly slow down training in this case. Epoch time increases by 6% with 2 modules and by 16% with 16 modules compared to end-to-end training. TRGL is only slower than VanGL by 2% for all number of modules due to the additional regularization term. This is comparable to InfoPro which report a time overhead between 1 and 27% compared to end-to-end training.

4.5.5 Sequential Full Block-wise Training

Full block-wise sequential training, meaning that each module is a single residual block and that the blocks are trained sequentially, allows to load and train only one block at a time. The memory needed is therefore divided by approximately the number of modules. The minimal memory needed to train the model is now the size of the heaviest module, which is the first one if the modules are equally deep, but it is possible to make the modules equally heavy (see Section 4.5.3 above). So even though it has been less explored in recent module-wise training methods because of its inferior accuracy compared to parallel module-wise training, it has been used in practice in very constrained settings such as on-device training of sensors [Teng et al., 2020, Tang et al., 2021]. We test our regularization therefore in this section on sequential block-wise training.

We propose in this setting to use shallower and initially wider ResNets with a downsampling and 256 filters initially and a further downsampling and doubling of the number of filters at the midpoint, no matter the depth. In these ResNets, we use the ResBlock from [He et al., 2016a] with two convolutional layers. If such a network is divided in K modules of M ResBlocks each, we call the network a $K-M$ ResNet. These wider shallower architectures are well-adapted to layer-wise training as seen in [Belilovsky et al., 2019].

We check in Table 24 in Appendix G that this architecture works well with parallel module-wise training by comparing on CIFAR10 a 2-7 ResNet with DGL, InfoPro and DDG. The 2-7 ResNet has 45 millions parameters, which is about the same as the ResNet-110 divided in two used by the other methods, but vanilla parallel module-wise training with this architecture already performs better than the other methods.

We now train a 10-block ResNet block-wise on CIFAR100 (a 10-1 ResNet in our notations). We report even the small improvements in accuracy to show

that our method works, or at worst does not fail, in all settings (parallel or sequential with many or few splits), which other methods so far don't do. For sequential training, block k is trained for $50{+}10k$ epochs where $0{\leq}k{\leq}10$, block 0 being the encoder. This idea of increasing the number of epochs per layer along with the depth is found in [Marquez et al., 2018]. For multi-lap sequential training, block k is trained for $10{+}2k$ epochs, and this is repeated for 5 laps. In block-wise training, the last block does not always perform the best and we report the accuracy of the best block. In Table 10, we see that multi-lap sequential (MLS) training improves the test accuracy of sequential training by around 0.8 percentage points when the training dataset is full, but works less well on small training sets. Of the two, the regularization mainly improves the test accuracy of MLS training. The improvement increases as the training set gets smaller and reaches 1 percentage point. That is also the case for parallel module-wise training, which already performs quite close to end-to-end training in the full data regime and much better in the small data regime.

Table 10: Average highest test accuracy and 95% confidence interval of 10-1 ResNet over 10 runs on CIFAR100 with different train sizes and sequential (Seq), multi-lap sequential (MLS) and parallel (Par) TRGL and VanGL, compared to E2E training.

Train size	Seq VanGL	Seq TRGL	MLS VanGL	MLS TRGL	Par VanGL	Par TRGL	E2E
50000	68.74 ± 0.45	**68.79** ± 0.56	69.48 ± 0.53	**69.95** ± 0.50	72.59 ± 0.40	**72.63** ± 0.40	75.85 ± 0.70
25000	60.48 ± 0.15	**60.59** ± 0.14	61.33 ± 0.23	**61.71** ± 0.32	64.84 ± 0.19	**65.01** ± 0.27	65.36 ± 0.31
12500	51.64 ± 0.33	**51.74** ± 0.26	51.30 ± 0.22	**51.89** ± 0.30	55.13 ± 0.24	**55.40** ± 0.35	52.39 ± 0.97
5000	36.37 ± 0.33	**36.40** ± 0.40	33.68 ± 0.48	**34.61** ± 0.59	39.45 ± 0.23	**40.36** ± 0.23	36.38 ± 0.31

We report further results for block-wise training on MNIST [LeCun et al., 2010] and CIFAR10 [Krizhevsky, 2009], but now we report the last block's accuracy. We see again greater improvement from the regularization as the training set gets smaller, gaining as much as 6 percentage points (Table 11 below, and Tables 25 and 26 in Appendix G.2).

Table 11: Average last block test accuracy and 95% confidence interval of 10-1 ResNet over 10 runs on CIFAR10 with different train sizes and sequential (Seq) TRGL and VanGL.

Train	Seq VanGL	Seq TRGL	E2E
50000	88.02 ± .18	**88.20** ± .24	91.88 ± .18
25000	83.95 ± .13	**84.28** ± .22	88.75 ± .27
10000	76.00 ± .39	**77.18** ± .34	82.61 ± .35
5000	67.74 ± .49	**69.67** ± .44	73.93 ± .67
1000	45.67 ± .88	**51.34** ± .90	50.63 ± .98

The 88% accuracy of sequential training on CIFAR10 in Table 11 is the same in table 2 of [Belilovsky et al., 2019], which is the best method for layer-wise sequential training available, with VGG [Simonyan and Zisserman, 2014] networks of comparable depth and width.

The results here show a few limitations of our method, as the improvements from the regularization are often minimal on sequential training. However, the results show that our approach works, or at least does not hurt performance, in all settings (parallel and sequential with many or few modules), whereas other papers don't test their methods in all settings, and some show problems in different settings from the original one in subsequent papers (e.g. delayed gradients methods when the number of modules increases [Huo et al., 2018a], and PredSim in [Pyeon et al., 2021]).

4.5.6 Avoiding Early Overfitting

We verify that our method avoids the stagnation in test accuracy with increasing depth. In Figure 18 below, we look at the test accuracy of each module after module-wise training with and without the regularization.

On the left, from the experiment with parallel module-wise training with 16 modules on STL10 from Table 6 in Section 4.5.1, we see that TRGL performs worse than vanilla module-wise training in the early modules, but surpasses it greatly in later modules. This indicates that it does avoid early overfitting and the subsequent stagnation in test accuracy. On the right, from experiments with sequential block-wise training from Table 11 in Appendix G, we see a large decline in the test accuracy of vanilla block-wise training after the first block that adding our transport regularization completely avoids. We see similar patterns in Figure 42 in Appendix G with parallel and multi-lap sequential block-wise training.

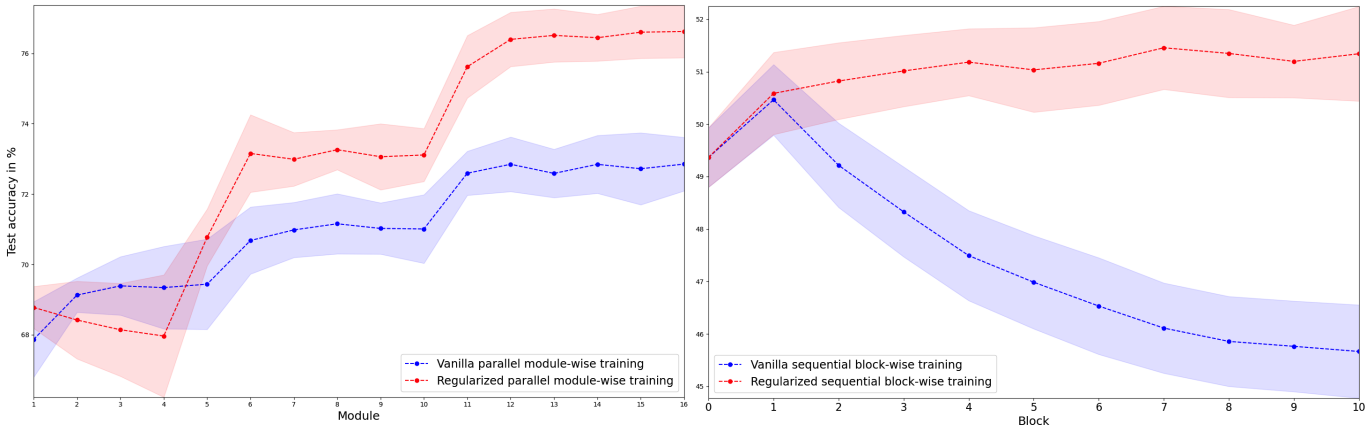

Figure 18: Accuracy after each module averaged over 10 runs with 95% confidence intervals. Left: parallel vanilla (VanGL, in blue) and regularized (TRGL, in red) module-wise training of a ResNet-110 with 16 modules on STL10. Right: sequential vanilla (VanGL, in blue) and regularized (TRGL, in red) block-wise training of a 10-1 ResNet on 2% of CIFAR10.

4.6 Discussion and Conclusion

We introduced a transport regularization for module-wise training that theoretically links it to gradient flows of the loss in distribution space. Our method provably leads to more regular modules and experimentally consistently improves the test accuracy of module-wise and block-wise sequential, parallel and multi-lap sequential (a variant of sequential training that we introduce) training. Through this simple method that does not complexify the architecture, we aim at making module-wise training competitive with end-to-end training while benefiting from its computational advantages: reduced memory usage and parallelism that is complementary to model and data parallelism in the case of parallel module-wise training, and training only one module at a time in constrained settings such as training on mobile devices in the case of sequential module-wise training.

The method can easily be combined with other methods of layer-wise training. Future work can also experiment with working in Wasserstein space W_p for $p \neq 2$, i.e. regularizing with a norm $\|.\|_p$ with $p \neq 2$. One can also ask how far the obtained composition network G_K is from being an OT map itself, which could provide a better stability bound than the one obtained by naively chaining the stability bounds (30) that follow from each module T_k being an OT map.

5 Adversarial Detection Through Transport Dynamics

Abstract

We propose a detector of adversarial samples that is based on the view of residual networks as discrete dynamical systems. The detector tells clean inputs from abnormal ones by comparing the discrete vector fields they follow throughout the network's layers. We also show that regularizing this vector field during training makes the network more regular on the data distribution's support, thus making the network's activations on clean samples more distinguishable from those on abnormal samples. Experimentally, we compare our detector favorably to other detectors using seen and unseen attacks, and show that the regularization of the network's dynamics improves the performance of adversarial detectors that use the internal embeddings as inputs, while also improving the network's test accuracy. The work presented in this section has led to the following publication [Karkar et al., 2023b] at ECML 2023.

5.1 Introduction

Neural networks have improved performances on many learning tasks, including image classification. They are however vulnerable to adversarial attacks [Szegedy et al., 2013]. These attacks modify an image in a way that is imperceptible to the human eye but that fools the network into wrongly classifying the modified image. These attacks can transfer to and fool other networks [Moosavi-Dezfooli et al., 2017a], can be carried out in the physical word, for example causing autonomous cars to misclassify road signs [Eykholt et al., 2018], and can be generated without having access to the network [Liu et al., 2017]. Developing networks that are robust to such attacks or accompanied by detectors that can detect them is therefore indispensable to deploying them safely in the real word [Amodei et al., 2016].

We focus here on the task of detecting adversarial samples. Neural networks trained with a softmax classifier produce overconfident predictions even for out-of-distribution and adversarial inputs [Nguyen et al., 2015]. This makes it difficult to detect such inputs via the softmax outputs. An adversarial detector is a system capable of predicting whether an input at test time has been adversarially modified. Adversarial detectors are trained on a dataset made up of clean and attacked inputs, after network training. While simply training the detector on the inputs has been tried, using their embeddings works better [Carlini and Wagner, 2017a]. Detection methods vary by which activations to use and how to process them to extract the features that are fed to the classifier that tells clean samples from attacked ones.

We make two contributions in this work. First, we propose an adversarial detector that is based on the view of neural networks as dynamical systems that move inputs in space, time represented by depth, to separate them before applying a

linear classifier. Given their resemblance to the Euler scheme for ODEs, ResNets [He et al., 2016b, He et al., 2016a, Weinan, 2017] are particularly amenable to this analysis. But the analysis and implementation extend immediately to any network where most layers have the same input and output dimensions. Our detector follows the trajectory of samples in space, through time, to differentiate clean and adversarial images. The statistics that we extract are the positions of the internal embeddings in space approximated by their norms and cosines to a fixed vector. This means that the extraction time and size are very low and allows to consider all layers, when many methods have to restrict themselves to a subset of layers.

[Wu et al., 2020] show an increased vulnerability of residual architectures to transferable attacks, precisely because of the skip connections. Given that skip connections are present in almost all state-of-the-art architectures, this motivates the need for a detector that is well adapted to residual architectures.

We test our detector on adversarial samples generated by 8 attacks, on 3 datasets and networks, comparing it favorably to the Mahalanobis [Lee et al., 2018] and natural scene statistics [Kherchouche et al., 2020] detectors. We also show that our detector can be used effectively for out-of-distribution detection even after only training it on adversarial samples.

Building on this dynamical viewpoint, our second contribution is to use the transport regularization during training proposed in Section 3 to make the activations of adversarial samples more distinguishable from those of clean samples, thus making adversarial detectors perform better, while also improving generalization.

We prove that the regularization achieves this by making the network more regular on the support of the data distribution. This does not make it more robust to attacks, but it will make the activations of the clean samples closer to each other and further from those of abnormal out-of-distribution samples, thus making adversarial detection easier, since adversarial attacks tend to lie outside the data manifold. We favorably compare the effect of our regularization in this regard to RCE training [Pang et al., 2018].

5.2 Related Work

An adversarial attack aims at slightly modifying an input image at test time by no more than ϵ in a certain metric (an L_p norm usually) so that the perturbation is undetectable to the human and that the perturbed image is misclassified by the network. More formally, given a classifier f in a multi-class classification task and $\epsilon > 0$, an adversarial sample y constructed from a clean sample x is $y = x + \delta$, such that $\|\delta\| \leq \epsilon$ and $f(y) \neq f(x)$. The maximal perturbation size ϵ has to be so small as to be almost imperceptible to a human (see Figure 19 below). An adversarial attack is an algorithm that finds such adversarial samples. We present below some popular adversarial attacks that are used to

find such adversarial samples. We present in more detail how these attacks work in Appendix E.1.

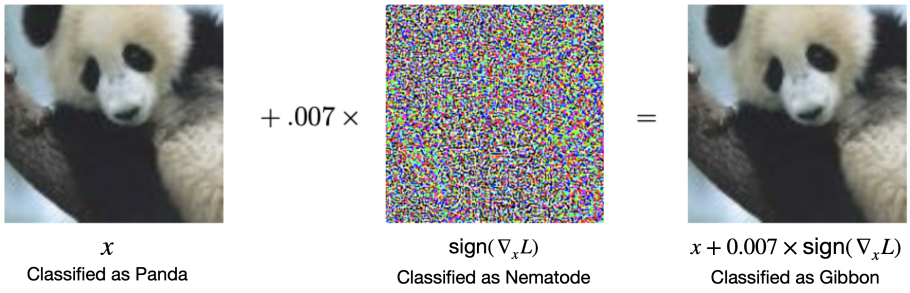

Figure 19: Example of an adversarial attack from [Goodfellow et al., 2015a].

White-box attacks are attacks where the adversary has access to the network's architecture, weights and gradients. The Fast Gradient Method (FGM) [Goodfellow et al., 2015b] takes a perturbation step in the direction of the gradient that maximizes the loss (see Figure 19 above). Projected Gradient Descent (PGD) [Madry et al., 2018] and the Basic Iterative Method (BIM) [Kurakin et al., 2017] are iterative versions of FGM. Auto-PGD (APGD) [Croce and Hein, 2020b] is a variant of PGD that uses an adaptive step size. Two slower but more effective attacks are DeepFool (DF) [Moosavi-Dezfooli et al., 2016], which iteratively perturbs an input in the direction of the closest decision boundary, and Carlini-Wagner (CW) [Carlini and Wagner, 2017b], which solves an optimization problem to find the perturbation.

Black-box attacks don't have any knowledge about the the network and can only query it. They include the Boundary Attack (BA) [Brendel et al., 2018], which starts from a large adversarial sample and moves towards the boundary decision to minimize the perturbation, and Hop-Skip-Jump (HSJ) [Chen et al., 2020a], which estimates the gradient direction at the decision boundary.

AutoAttack [Croce and Hein, 2020b] is combination of 3 white-box attacks (two versions of Auto-PGD and the FAB attack [Croce and Hein, 2020a]), and of the black-box Square Attack (SA) [Andriushchenko et al., 2020].

Two main defense mechanisms exist against adversarial attacks: adversarial robustness and adversarial detection.

Adversarial robustness means training a network that is not easily fooled by adversarial samples, for example through adversarial training (augmenting the training set with adversarial samples [Szegedy et al., 2013, Goodfellow et al., 2015b]), regularization [Gu and Rigazio, 2015], distillation [Papernot et al., 2016c] and denoising [Meng and Chen, 2017]. In the context of adversarial robustness, the view of ResNets as transport systems was used in [Wang et al., 2019b] to relate injecting noise into residual networks and ensembling them

to approximating the Feynman-Kac formula for convection-diffusion, which makes the level sets more regular and the network more robust to adversarial attacks. These robustness-enhancing defense mechanisms can generally be avoided by newer attacks [Carlini and Wagner, 2017a], and a theoretical debate exists over a trade-off that is observed in practice between test accuracy and adversarial robustness [Yang et al., 2020b, Raghunathan et al., 2020, Schmidt et al., 2018, Dohmatob, 2019].

Given these limitations of adversarial robustness, a detector capable of detecting adversarial samples is an important addition to a neural network in deployment. An advantage to having an adversarial detector is that it can, at least in principle, be used to also detect out-of-distribution samples, for which the notion of robustness does not make sense. While simply training the detector on a dataset of clean and adversarial inputs has been tried, using their embeddings as they go through the network works better for most detection methods [Carlini and Wagner, 2017a]. But using all the intermediate embeddings of an input as features to train an adversarial detector on is impossible given the huge memory and time it would require. Detection methods therefore vary by which activations they use and how to process them to extract the features that are fed to the classifier that tells clean samples from attacked ones.

An early idea for adversarial detection was to use a second network [Metzen et al., 2017]. However, this network can be adversarially attacked as easily as the first network. More recent popular statistical approaches include LID [Ma et al., 2018], Mahalanobis [Lee et al., 2018, Ren et al., 2021], MMD [Grosse et al., 2017] and KD [Feinman et al., 2017]. LID trains the detector on the local intrinsic dimensionality (LID, [Houle, 2013, Amsaleg et al., 2015, Tempczyk et al., 2022] of activations approximated over a batch. Mahalanobis train the detector on the Mahalanobis distances between the activations and a Gaussian fitted to them during network training, assuming they are normally distributed. MMD employs a statistical test that uses the maximum mean discrepancy distance (MMD, [Gretton et al., 2012]) between shuffled inputs. KD combines kernel density (KD, [Parzen, 1962, Rosenblatt, 1956]) estimates and Bayesian uncertainty estimates obtained via dropout [Srivastava et al., 2014] as detection features.

Our detector is not a statistical approach and does not need batch-level statistics, nor statistics from the training data of the network. [Carrara et al., 2019] is the most similar approach as they monitor the evolution of the positions of adversarial samples by comparing them to representative pivot positions of clean points from each class. This also requires the detector to have access to the training data of the network to compute the pivot positions as a centroid or a medoid, and makes the extraction time and size potentially much larger as it increases linearly with the number of classes.

Detectors trained in the Fourier domain [Harder et al., 2021] or on natural scene statistics [Kherchouche et al., 2020, Kherchouche et al., 2022] have been proposed. See Appendix E.2 for details and [Aldahdooh et al., 2022] for a review.

Our second contribution is to regularize the network in a way that makes it Hölder-continuous, but only on the input distribution's support. Estimations of the Lipschitz constant of a network have been used as estimates of its robustness to adversarial samples in [Weng et al., 2018, Szegedy et al., 2013, Virmaux and Scaman, 2018, Hein and Andriushchenko, 2017], and making the network more Lipschitz (e.g. by penalizing an upper bound on its Lipschitz constant) has been used to make it more robust to adversarial samples in [Hein and Andriushchenko, 2017, Cisse et al., 2017]. These regularizations [Cisse et al., 2017, Gu and Rigazio, 2015] often work directly on the weights of the neural network, therefore making the neural network more regular on the entire input space. The main difference with our method is that we only endue the network with regularity on the support of the clean samples. This won't make it more robust to adversarial samples who lie outside the support of the data distribution, but it will make the activations of the clean samples closer to each other and further from those of abnormal samples, thus making adversarial detection easier.

That adversarial samples tend to lie outside the data manifold, particularly in its co-dimensions, is a common observation and explanation for why adversarial attacks are easy to find in high dimensions [Gilmer et al., 2018, Tanay and Griffin, 2016, Song et al., 2018, Ma et al., 2018, Samangouei et al., 2018, Khoury and Hadfield-Menell, 2018, Alemany and Pissinou, 2022, Feinman et al., 2017, Chattopadhyay et al., 2019]. According to [Tanay and Griffin, 2016], adversarial samples are easy to find in high dimensions because the decision boundary intersects the data manifold but extends beyond it while remaining close to it (see Figure 20 below).

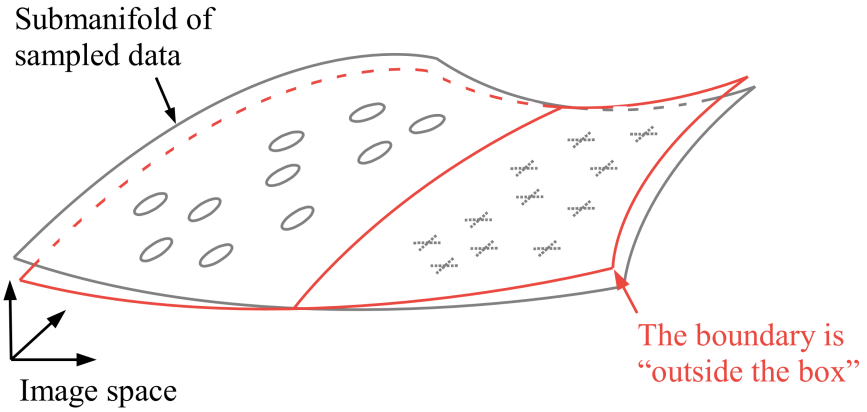

Figure 20: Boundary tilting perspective of adversarial samples from [Tanay and Griffin, 2016]. Adversarial samples lie on the other side of the boundary plane in the co-dimension of the data manifold, and not on other side of the boundary line that intersects the data manifold.

To the best of our knowledge, [Pang et al., 2018] is the only other method that

also attempts to improve detection by encouraging the network during (normal not adversarial) training to learn latent representations that are more different between clean and adversarial samples. They do this by replacing the cross-entropy loss by reverse cross-entropy (RCE) that encourages uniform softmax outputs among the non-predicted classes. We find that our regularization leads to better classification accuracy and adversarial detection than RCE.

5.3 Method and Theory

We take the view that a residual network moves its inputs according to a discrete vector field to separate them. Points in each class follow similar trajectories. Heuristically, for a successful adversarial sample that lies close to clean samples, the vector field it follows has to be different at some step from that of the clean samples (in its original class and in other classes), so that it can join the trajectory of the points in another class. In Section 5.3.1, we present how to detect adversarial attacks by looking at these trajectories. In Section 5.3.2, we discuss the link between the regularity of neural networks and their robustness to adversarial attacks and use the transport regularization (18) to improve detectability of adversarial attacks.

5.3.1 Detection

Given a residual network that applies $x_{m+1} = x_m + hr_m(x_m)$ to an input x_0 for $0 \leq m < M$, we consider the embeddings x_m for $0 < m \leq M$, or the residues $r_m(x_m)$ for $0 \leq m < M$. To describe their positions in space, we take their norms and their cosine similarities with a fixed vector as features to train our adversarial detector on. Using only the norms already gave good detection accuracy. Cosines to other orthogonal vectors can be added to better locate the points at the price of increasing the number of features. We found that using only one vector already gives state-of-the-art detection, so we only use the norms and cosines to a fixed vector of ones. We train the detector (a random forest in practice) on these features. The embeddings x_m and the residues $r_m(x_m)$ can equivalently describe the trajectory of x_0 in space through the blocks. In practice, we use the residues $r_m(x_m)$, with their norms squared and averaged. So the feature vector given to the random forest for each x_0 that goes through a network that applies $x_{m+1} = x_m + h\ r_m(x_m)$ is

$$\left(\frac{1}{d_m}\|r_m(x_m)\|^2,\ \cos\Big(r_m(x_m), \mathbf{1}_m\Big)\right)_{m=0}^{M-1} \tag{34}$$

and the label is 0 if x_0 is clean and 1 if it is adversarial. Here cos is the cosine similarity between two vectors and $\mathbf{1}_{d_m}$ is a vector of ones of size d_m where d_m is the size of $r_m(x_m)$. For any non-residual architecture $x_{m+1} = g_m(x_m)$, the vector $x_{m+1} - x_m$ can be used instead of $r_m(x_m)$ on layers that have the same input and output dimension, allowing to apply the method to any network with many such layers. And we do test the detector on a ResNeXt, which

does not fully satisfy the dynamic view, as the activation is applied after the skip-connection, i.e. $x_{m+1} = \mathrm{ReLU}(x_m + h\ r_m(x_m))$.

The number of features is twice that of residual blocks (a norm and a cosine per block). This is of the same order as for other popular detectors such as Mahalanobis ([Lee et al., 2018]) and LID ([Ma et al., 2018]) that extract one feature per residual stage (a residual stage is a group of blocks that keep the same dimension). Even for common large architectures, twice the number of residual blocks is still a small number of features for training a binary classifier (ResNet152 has 50 blocks). More importantly, the features we extract (norms and cosines) are quick to calculate, whereas those of other methods require involved statistical computations on the activations. We include in Section 5.4.6 a favorable time comparison of our detector to the Mahalanobis detector. Another advantage is that our detector does not have a hyper-parameter to tune unlike the Mahalanobis and LID detectors.

5.3.2 Regularization

Regularity of neural networks (typically Lipschitz continuity) has been used as a measure of their robustness to adversarial attacks [Weng et al., 2018, Szegedy et al., 2013, Virmaux and Scaman, 2018, Hein and Andriushchenko, 2017, Cisse et al., 2017]. Indeed, the smaller the Lipschitz constant L of a function f satisfying $\|f(x) - f(y)\| \leq L\|x - y\|$, the less f changes its output $f(y)$ for a perturbation (adversarial or not) y of x. Regularizing a network to make it more Lipschitz and more robust has therefore been tried in [Hein and Andriushchenko, 2017] and [Cisse et al., 2017]. For this to work, the regularization has to apply to adversarial points, i.e. outside the support of the clean data distribution. Indeed, the Lipschitz continuity obtained though most of these methods and analyses apply on the entire input space $\mathbb{R}^d$ as they penalize the network's weights directly. Likewise, choosing a small step size $0<h<1$ for the residual blocks as in [Zhang and Wynter, 2018, Zhang et al., 2019], while improving stability, will have the same effect on all inputs, adversarial or not.

We propose an alternative approach where we regularize the network only on the support of the input distribution, making it η-Hölder on it. Since this result does not apply outside the input distribution's support, particularly in the adversarial spaces, then this regularity can serve to make adversarial samples more distinguishable from clean ones, and therefore easier to detect. We show experimentally that the behavior of the network will be more distinguishable between clean and adversarial samples in practice in Section 5.4.1.

So as in Section 3.4, we train our network by solving

$$\begin{aligned} \min_{\theta} \quad & \mathcal{C}(\theta) = \sum_{x \in \mathcal{D}} \sum_{m=0}^{M-1} \|r_m(\phi_m^x)\|^p \\ \text{subject to} \quad & \phi_{m+1}^x = \phi_m^x + h r_m(\phi_m^x) \text{ and } \phi_0^x = x \text{ for all } x \in \mathcal{D} \\ & \mathcal{L}(\theta) = 0 \end{aligned} \tag{35}$$

which is the discretization of

$$\begin{aligned} \inf_{v,F} \quad & \int_0^1 \|v_t\|^p_{L^p((\phi_t^\cdot)_\sharp \mu)} \, \mathrm{d}t \\ \text{subject to} \quad & \partial_t \phi_t^x = v_t(\phi_t^x) \\ & \phi_0^\cdot = \mathrm{id} \\ & (\phi_1^\cdot)_\sharp \mu \in S_{F,\mathcal{L}} \end{aligned} \tag{36}$$

which is equivalent to

$$\begin{aligned} \inf_{T,F} \quad & \mathcal{C}(T) = \int_\Omega c(x, T(x)) \, \mathrm{d}\mu(x) \\ \text{subject to} \quad & T_\sharp \mu \in S_{F,\mathcal{L}} \end{aligned} \tag{37}$$

We solve again (35) through the method of multipliers

$$\begin{cases} \theta_{i+1} = \arg\min_\theta \ \mathcal{C}(\theta) + \lambda_i \ \mathcal{L}(\theta) \\ \lambda_{i+1} = \lambda_i + \tau \ \mathcal{L}(\theta_{i+1}) \end{cases} \tag{38}$$

We had a regularity result on the transport part T of the network in Section 3.4, but we now look at individual residual blocks. We first prove a regularity result on each residual r_m that applies on the clean data points moving according to v solution to (17).

We take $\Omega \subset \mathbb{R}^d$ convex and compact and the data distribution $\alpha \in \mathcal{P}(\Omega)$ absolutely continuous and such that $\delta\Omega$ is α-negligible. We suppose that there exists an open bounded convex set $X \subset \Omega$ such that α is bounded away from zero and infinity on X and is zero on its complement $X^\complement$. From Theorem 2 in Section 3.3.2, we know that (37) and (36) are equivalent and have solutions (T, F) and (v, F) such that T is an optimal transport map between α and $\beta := T_\sharp \alpha$. We suppose that β is absolutely continuous and that there exists an open bounded convex set $Y \subset \Omega$ such that β is bounded away from zero and infinity on Y and is zero on $Y^\complement$. In the rest of this section, v solves (36) and we suppose that we find a solution to the discretized problem (35) that is an $\varepsilon/2$-approximation of v, i.e. $\|r_m - v_{t_m}\|_\infty \leq \varepsilon/2$ for all $0 \leq m < M$, with $t_m = m/M$.

In Theorem 6, we show that the regularization makes the residual blocks of the network η-Hölder (with an error of ε) on the support of the input distribution as it moves according to the theoretical vector field solution v. The results hold for all norms on $\mathbb{R}^d$.

Theorem 6. For $a, b \in \mathrm{support}(\alpha_{t_m})$ where $\alpha_t := (\phi_t^\cdot)_\sharp \alpha$ with ϕ solving (36) along with v, we have

$$\begin{aligned} \|r_m(a) - r_m(b)\| &\leq \varepsilon + K\|a - b\|^{\zeta_1} \text{ if } \|a - b\| \leq 1 \\ \|r_m(a) - r_m(b)\| &\leq \varepsilon + K\|a - b\|^{\zeta_2} \text{ if } \|a - b\| > 1 \end{aligned}$$

for constants $K > 0$ and $0 < \zeta_1 \leq \zeta_2 \leq 1$,

Proof. A solution v to (36) exists and is linked to an optimal transport map T that is a solution to (37) through $v_t = (T - \mathtt{id}) \circ T_t^{-1}$ where $T_t := (1-t)\mathtt{id} + tT$ which is invertible (see Appendix B). By Theorem 1 in Appendix B, being an optimal transport map, T is η-Hölder on X. So for all $t \in [0, 1[$ and all $a, b \in \text{support}(\alpha_t)$, where $\alpha_t = (\phi_t^{\cdot})_\sharp \alpha = (T_t)_\sharp \alpha$ with ϕ solving (36) together with v, we have

$$\|v_t(a) - v_t(b)\| \leq \|T_t^{-1}(a) - T_t^{-1}(b)\| + C\|T_t^{-1}(a) - T_t^{-1}(b)\|^\eta \tag{39}$$

Since $(\alpha_t)_{t=0}^1$ is a geodesic between α and $\beta = \alpha_1 = T_\sharp \alpha$, then $(\alpha_s)_{s=0}^t$ is a geodesic between α and α_t (modulo reparameterization to $[0, 1]$). And since $\alpha_s = (T_s)_\sharp \alpha$, the map T_t is an optimal transport map between α and α_t. Therefore its inverse T_t^{-1} is an optimal transport map (see Theorem 25 in Appendix B) and is η_t-Hölder with $0 < \eta_t \leq 1$ (being a push-forward by T_t, the support of α_t satisfies the conditions of Theorem 1 in Appendix B). Therefore, for all $a, b \in \text{support}(\alpha_t)$

$$\|v_t(a) - v_t(b)\| \leq C_t\|a - b\|^{\eta_t} + CC_t^\eta \|a - b\|^{\eta\eta_t} \tag{40}$$

and for all $a, b \in \text{support}(\alpha_{t_m})$

$$\|r_m(a) - r_m(b)\| \leq \varepsilon + C_{t_m}\|a - b\|^{\eta_{t_m}} + CC_{t_m}^\eta \|a - b\|^{\eta\eta_{t_m}} \tag{41}$$

by the hypothesis on r and the triangle inequality. Let $K := \max_m C_{t_m} + CC_{t_m}^\eta$, $\zeta_1 := \eta \min_m \eta_{t_m}$ and $\zeta_2 := \max_m \eta_{t_m}$. Then, we have the desired result immediately from (41). □

We use Theorem 6 to now bound the distance between the residues at depth m as a function of the distance between the network's inputs. For inputs a_0 and b_0 to the network, the intermediate embeddings are $a_{m+1} = a_m + hr_m(a_m)$ and $b_{m+1} = b_m + hr_m(b_m)$, and the residues used to compute features for adversarial detection are $r_m(a_m)$ and $r_m(b_m)$. So we want to bound $\|r_m(a_m) - r_m(b_m)\|$ as a function of $\|a_0 - b_0\|$. This is usually done by multiplying the Lipschitz constants of each block up to depth m, which leads to an overestimation [Huster et al., 2019], or through more complex estimation algorithms [Virmaux and Scaman, 2018, Latorre et al., 2020, Bhowmick et al., 2021]. Bound (39) allows through T_t^{-1} to avoid multiplying the Hölder constants of the blocks. If a_0 and b_0 are on the clean data support X, we get Theorem 7, which we prove together with Theorem 8 below.

Theorem 7. For $a_0, b_0 \in X$ and constants $C, L > 0$,

$$\begin{aligned}\|r_m(a_m) - r_m(b_m)\| \leq\ & \varepsilon + \|a_0 - b_0\| + C\|a_0 - b_0\|^\eta + \\ & + L(\|a_m - \phi_{t_m}^{a_0}\| + \|b_m - \phi_{t_m}^{b_0}\|)\end{aligned}$$

The term $\mu(a_0) := \|a_m - \phi_{t_m}^{a_0}\|$ (and $\mu(b_0) := \|b_m - \phi_{t_m}^{b_0}\|$) is the distance between the point a_m after m residual blocks and the point $\phi_{t_m}^{a_0}$ we get by following the

theoretical solution vector field v up to time t_m starting from a_0. If a_0 and b_0 are not on the data support X, an extra term has to be introduced to use bound (39). Bounding the terms $\mu(a_0)$ and $\mu(b_0)$ is possible under more regularity assumptions on v. We assume then that v is $\mathcal{C}^1$ and Lipschitz in x, which is not stronger than the regularity we get on v through our regularization, as it does not give a similar result to bound (39). We have for all inputs a_0 and b_0, whether they are clean or not, Theorem 8:

Theorem 8. For $a_0, b_0 \in \mathbb{R}^d$ and constants $R, S > 0$,

$$\begin{aligned}\|r_m(a_m) - r_m(b_m)\| \leq \varepsilon + LS\varepsilon + LSRh + \|a_0 - b_0\| + C\|a_0 - b_0\|^\eta + \\ + LS(\mathrm{dist}(a_0, X) + \mathrm{dist}(b_0, X))\end{aligned}$$

Proof. Since $T_t^{-1}(\phi_t^x) = x$, we have for any $a_0, b_0 \in X$ by the triangle inequality

$$\begin{aligned}\|r_m(a_m) - r_m(b_m)\| &\leq \|r_m(a_m) - r_m(\phi_{t_m}^{a_0})\| + \|r_m(\phi_{t_m}^{a_0}) - v_{t_m}(\phi_{t_m}^{a_0})\| + \\ &+ \|v_{t_m}(\phi_{t_m}^{a_0}) - v_{t_m}(\phi_{t_m}^{b_0})\| + \|r_m(\phi_{t_m}^{b_0}) - v_{t_m}(\phi_{t_m}^{b_0})\| + \\ &+ \|r_m(b_m) - r_m(\phi_{t_m}^{b_0})\| \\ &\leq \varepsilon + \|a_0 - b_0\| + C\|a_0 - b_0\|^\eta + \\ &+ L(\|a_m - \phi_{t_m}^{a_0}\| + \|b_m - \phi_{t_m}^{b_0}\|)\end{aligned}$$

where $L = \max_m L_m$ and L_m is the Lipschitz constant of r_m (which is Lipschitz being a composition of matrix multiplications and activations such as ReLU). This the bound in Theorem 7.

In this bound, the term $\|a_m - \phi_{t_m}^{a_0}\|$ (and likewise $\|b_m - \phi_{t_m}^{b_0}\|$) represents the distance between the point a_m we get after m residual blocks (i.e. after m Euler steps using approximation r of v) and the point $\phi_{t_m}^{a_0}$ we get by following the solution vector field v up to time t_m. By the properties of the Euler scheme (consistency and zero-stability, see Corollaries 11, 13 and 15 in Section A.1), under more conditions on v, it is possible to bound this term. Indeed, if v is $\mathcal{C}^1$ and M-Lipschitz in x (this is not stronger than the regularity we get on v through our regularization, because we still need to use (39)), we have for constants $R, S > 0$,

$$\|\phi_{t_m}^{a_0} - a_m\| \leq \|\phi_{t_m}^{a_0} - \tilde{a}_m\| + \|\tilde{a}_m - a_m\| \leq S\varepsilon + SRh$$

where $\tilde{a}_m$ comes from the Euler scheme with access to v ($\tilde{a}_{m+1} := \tilde{a}_m + hv_{t_m}(\tilde{a}_m)$ and $\tilde{a}_0 := a_0$), R is the consistency constant of the Euler method and S is its zero-stability constant. If $a_0, b_0 \notin X$, we need to introduce $\hat{a}_0 := \mathrm{Proj}_X(a_o)$ and $\hat{b}_0 := \mathrm{Proj}_X(b_o)$ to apply (39). We now get

$$\|r_m(a_m) - r_m(b_m)\| \leq \varepsilon + \|a_0 - b_0\| + C\|a_0 - b_0\|^\eta + L(\|a_m - \phi_{t_m}^{\hat{a}_0}\| + \|b_m - \phi_{t_m}^{\hat{b}_0}\|)$$

Bounding the terms $\|a_m - \phi_{t_m}^{\hat{a}_0}\|$ and $\|b_m - \phi_{t_m}^{\hat{b}_0}\|$ now gives

$$\|\phi_{t_m}^{\hat{a}_0} - a_m\| \leq \|a_m - \tilde{a}_m\| + \|\tilde{a}_m - \phi_{t_m}^{\hat{a}_0}\| \leq S(\|a_0 - \hat{a}_0\| + \varepsilon) + SRh$$

where $\tilde{a}_m$ now comes from the Euler scheme with access to v that starts at $\hat{a}_0$ (meaning $\tilde{a}_{m+1}{:=}\tilde{a}_m{+}hv_{t_m}(\tilde{a}_m)$ and $\tilde{a}_0{:=}\hat{a}_0$). Likewise, we get the same bound for $\|b_m - \phi_{t_m}^{\hat{b}_0}\|$.

Since $\|a_0 - \hat{a}_0\| = \text{dist}(a_0, X)$ and $\|b_0 - \hat{b}_0\| = \text{dist}(b_0, X)$, we get the bound in Theorem 8. Note that if we use the stability of the differential equation instead of the stability of the Euler method to bound $\|a_m - \phi_{t_m}^{\hat{a}_0}\|$ we get the same result. Indeed, if $\tilde{a}_m$ again comes from the Euler scheme with access to v that starts at a_0 (meaning $\tilde{a}_{m+1}{:=}\tilde{a}_m{+}hv_{t_m}(\tilde{a}_m)$ and $\tilde{a}_0{:=}a_0$), we can write, for some constant $F > 0$

$$\begin{aligned}\|\phi_{t_m}^{\hat{a}_0} - a_m\| \leq \|a_m - \tilde{a}_m\| + \|\tilde{a}_m - \phi_{t_m}^{a_0}\| + \|\phi_{t_m}^{a_0} - \phi_{t_m}^{\hat{a}_0}\| \leq S\varepsilon + SRh + \\ + F\|a_0 - \hat{a}_0\|\end{aligned}$$

since

$$\begin{aligned}\|\phi_{t_m}^{a_0} - \phi_{t_m}^{\hat{a}_0}\| &\leq \|a_0 - \hat{a}_0\| + \int_0^{t_m} \|v_s(\phi_s^{a_0}) - v_s(\phi_s^{\hat{a}_0})\| \,\mathrm{d}s \\ &\leq \|a_0 - \hat{a}_0\| + M \int_0^{t_m} \|\phi_s^{a_0} - \phi_s^{\hat{a}_0}\| \,\mathrm{d}s \\ &\leq F\|a_0 - \hat{a}_0\|\end{aligned}$$

where we get the last line by Gronwall's lemma. □

The terms $\text{dist}(a_0, X)$ and $\text{dist}(b_0, X)$ in Theorem 8 show that the regularity guarantee is increased for inputs in X. The trajectories of clean points are then closer to each other and more different from those of abnormal samples outside X as in Figure 22 in Section 5.4.1.

5.4 Experiments

We evaluate our method on adversarial samples found by 8 adversarial attacks. The threat model is as follows. We use 2 popular white-box attacks that have access to the network and to its weights and architecture but not to its training data: FGM [Goodfellow et al., 2015b], PGD [Madry et al., 2018] (specifically the Auto-PGD-CE (APGD) variant [Croce and Hein, 2020b]), BIM [Kurakin et al., 2017], DF [Moosavi-Dezfooli et al., 2016], CW [Carlini and Wagner, 2017b], and AA [Croce and Hein, 2020b], and 2 black-box attacks that can only query the network: HSJ [Chen et al., 2020a] and BA [Brendel et al., 2018]. We assume the attacker has no access to the detector and use the untargeted (i.e. not seeking to direct the mistake towards a particular class) versions of the attacks.

For FGM, APGD and BIM, we use a maximal perturbation size of $\epsilon = 0.03$. We use the L_2 norm for CW and HSJ and the L_∞ norm for all the other attacks. We use the Adversarial Robustness Toolbox (ART) Python package [Nicolae et al., 2018] to generate the adversarial samples, except for AA for which we use the authors' original code [Croce and Hein, 2020b]. We use the default

values from the ART package and the code of [Croce and Hein, 2020b] for the hyper-parameters of the attacks that are not mentioned here. The number of iterations is 50 for HSJ, 5000 for BA, 10 for CW and 100 for APGD and DF.

We compare our detector (which we call the Transport detector or TR in the tables below) to the Mahalanobis detector (MH in the tables below) of [Lee et al., 2018] and to the detector of [Kherchouche et al., 2020, Kherchouche et al., 2022] that uses natural scene statistics (NS in the tables below). We compare our transport regularization to reverse cross entropy (RCE) training of [Pang et al., 2018], which is also meant to improve detection of adversarial examples.

We use three architectures and datasets for our experiments: a ResNeXt50 trained on CIFAR100, a ResNet110 trained on CIFAR10 and a WideResNet trained on TinyImageNet. For each architecture-dataset pair, the network is trained normally with cross entropy, with our regularization added to cross entropy (called a LAP-network) and with reverse cross entropy instead of cross entropy (called an RCE-network). For our regularization, we use (19) with τ=1, s=1 and λ_0=1 for all networks. These hyper-parameters are chosen to improve validation accuracy during training and not adversarial detection. More training details are in Appendix H.1. The code is available at `github.com/advadvadvadvadv/adv`.

We find that our detector consistently outperforms the other detectors, also on out-of-distribution detection, and especially on generalization to unseen attacks. LAP training mainly improves, sometimes largely, the accuracy of the Mahalanobis detector on seen attacks, and consistently does better than RCE.

5.4.1 Preliminary Experiments

We find again that LAP training improves generalization. The vanilla ResNeXt50 has a test accuracy of 74.38% on CIFAR100 while the LAP-ResNeXt50 has an accuracy of 77.2%. The vanilla ResNet110 has a test accuracy of 92.52% on CIFAR10 while the LAP-ResNet110 has an accuracy of 93.52% and the RCE-ResNet110 of 93.1%. The vanilla WideResNet has a test accuracy of 65.14% on TinyImageNet while the LAP-WideResNet has an accuracy of 65.34%.

We see in Figure 21 below that regularizing the transport cost $\mathcal{C}$ through LAP training makes it more distinguishable between clean and adversarial samples. Indeed, the histograms of $\mathcal{C}$ on adversarial samples move away from the histogram of $\mathcal{C}$ on clean samples when using LAP training instead of vanilla training. Simply using the empirical quantiles of $\mathcal{C}$ on clean samples to reject abnormal samples with a transport cost $\mathcal{C}$ that falls outside of these quantiles allows then to detect samples from some simple attacks with near perfect recall and a chosen fixed false positive rate, without training on or even seeing adversarial samples. In Figure 21, using the 0.02 and 0.98 empirical quantiles of $\mathcal{C}$ on the clean samples allows to detect adversarial samples from the FGM attack with big enough maximal perturbation ϵ almost perfectly on the LAP-ResNeXt50 trained on CIFAR100, but not on the vanilla ResNeXt50.

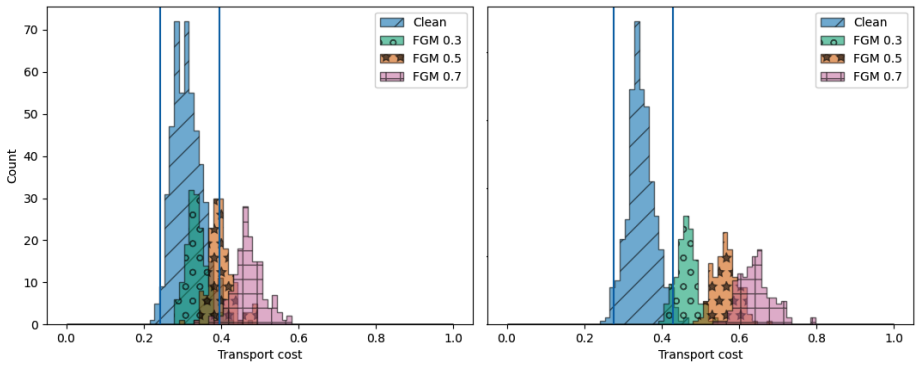

Figure 21: Histogram of the transport cost $\mathcal{C}$ for clean and FGM-attacked test samples with different values of ϵ on CIFAR100. The vertical lines represent the 0.02 and 0.98 empirical quantiles of the transport cost of the clean samples. Left: ResNeXt50. Right: LAP-ResNeXt50.

We visualize this increased difference in behavior between clean and out-of-distribution (OOD) points with LAP training in 2 dimensions in Figure 22. The experiments is binary classification of the circles dataset from `scikit-learn` [Pedregosa et al., 2011], where we add OOD points at test time.

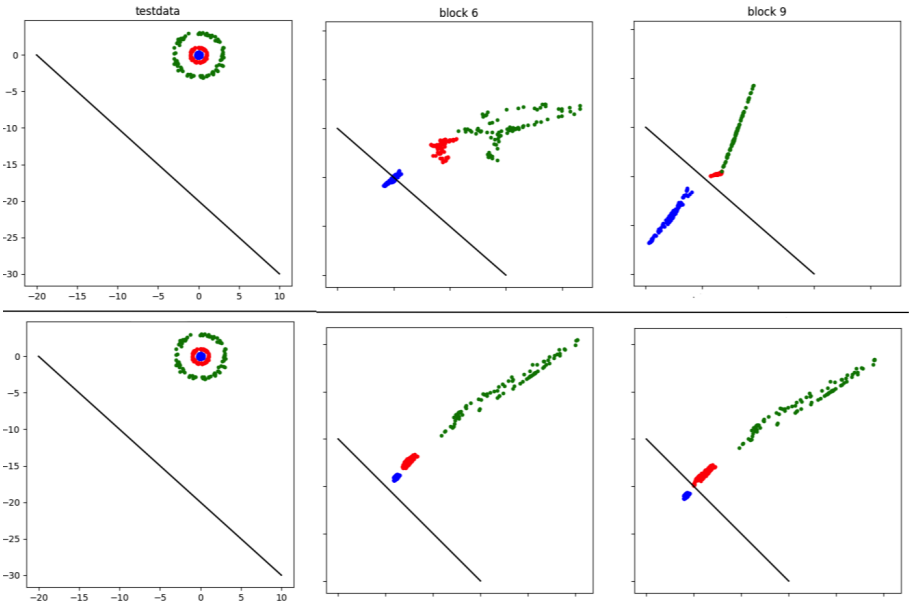

Figure 22: Transformed circles test set from scikit-learn (red and blue) and out-of-distribution points (green) after blocks 6 and 9 of a ResNet with 9 blocks. We use LAP training in the second row, which makes the movements of the clean points (red and blue) more similar to each other and more different from the movements of the green out-of-distribution points than when using the vanilla network in the first row. In particular, without the regularization, the green points are closer to the clean red points after blocks 6 and 9 which is undesirable.

5.4.2 Detection of Seen Attacks

For detection training, the test set is split in 0.9/0.1 proportions into two datasets, B1 and B2. For each image in B1 (respectively B2), an adversarial sample is generated and a balanced detection training set (respectively detection test set) is created. Since adversarial samples are created for a specific network, this is done for the vanilla network and its LAP and RCE versions. We tried augmenting the detection training dataset with a random perturbation of each image, to be considered as clean, as in [Lee et al., 2018], but we found that this usually does not improve detection accuracy. This dataset creation protocol is standard and is depicted in Figure 23 below. We tried limiting the dataset to successfully attacked images only as in [Lee et al., 2018], but did not find a consistent improvement in detection, even only on successfully attacked images.

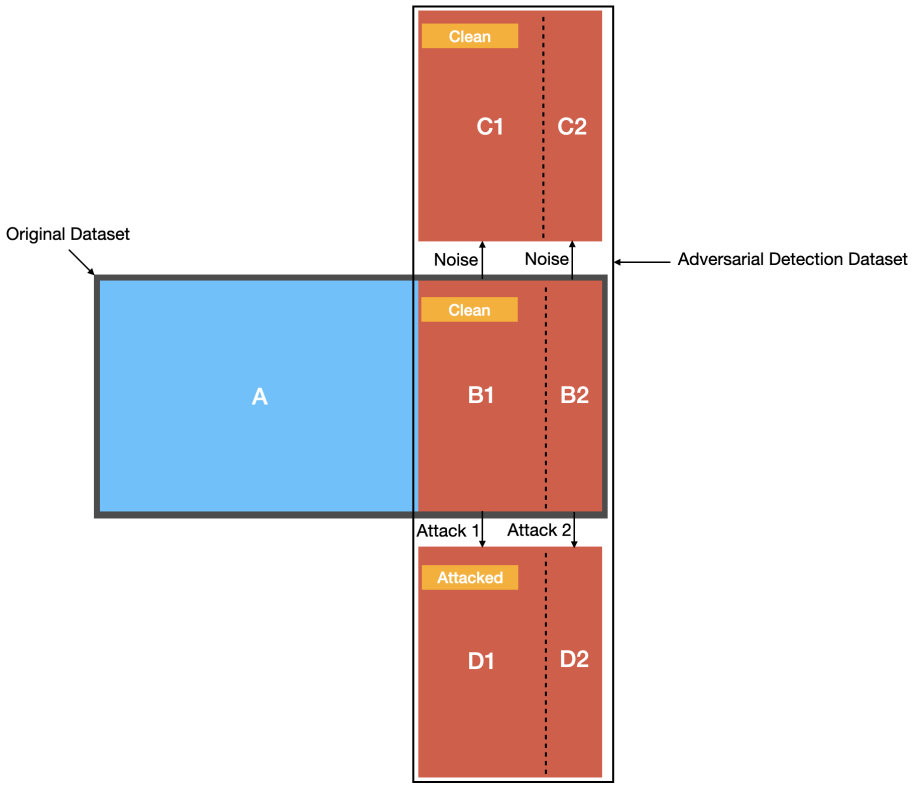

Figure 23: Adversarial detection dataset creation. A∪B1∪B2 is the original dataset, where A is the training set and B1∪B2 is the test set. We create a noisy version of B1∪B2 by adding random noise to each sample in B1∪B2 to get C1∪C2. Noisy samples are considered clean (i.e. not attacked) in adversarial detection training. We create an attacked version of B1∪B2 by creating an attacked image from each image in B1∪B2 to get D1∪D2. In the case of generalization to unseen attacks, Attack 2 used to create D2 from B2 is different from Attack 1 used to create D1 from B1. Otherwise, Attack 1 and Attack 2 are the same. B1∪C1∪D1 is the adversarial detection training set and B2∪C2∪D2 is the adversarial detection test set.

Samples in the detection training set are fed through the network and the features of each detection method are extracted. We tried three classifiers (logistic regression, random forest and SVM from scikit-learn [Pedregosa et al., 2011]) trained on these features for each detection method, and kept the random forest as it always performs best.

We tried two methods to improve all detectors: class-conditioning and ensembling. In class-conditioning, the features are grouped by the class predicted by the network, and a detector is trained for every class. At test time, the detector trained on the features of the predicted class is used. A detector is also trained on all samples regardless of the predicted class and is used in case a certain class is never targeted by the attack. We also tried ensembling the class-conditional detector with the general all-class detector: an input is considered an attack if at least one detector says so. This ensemble of the class-conditional detector and the general detector performs best for all detection methods, and is the one we use.

We report the detection accuracy of each detector on the detection test set for both the vanilla version of the network and its LAP version in Table 12 below. In each cell, the first number corresponds to the vanilla network and the second to the regularized LAP-network. Since the NS detector takes the image and not its embeddings as input, the impact of LAP and RCE training on its performance is minimal and we report its performance on the vanilla network only.

These results are averaged over 5 runs and the standard deviations are in Tables 32, 33 and 34 in Appendix H.2, along with results on RCE-networks. Since they are much slower to generate, we test detection of black-box attacks HSJ and BA only on the network-data pair of ResNet110-CIFAR10 and of the DF and the CW attacks only on ResNet110-CIFAR10 and ResNeXt50-CIFAR100.

Results in Table 12 show two things. First, our detector performs better than both other detectors, with or without the regularization. Second, both the TR and MH detectors work better on the LAP-networks most times. The MH detector benefits more from the regularization, but on all attacks, the best detector is always the Transport detector. In Tables 32 and 33 in Appendix H.2, RCE often improves detection accuracy in this experiment, but clearly less than LAP training.

Results for ResNet110 on CIFAR10 are in the first row of Table 12. Our detector outperforms the Mahalanobis detector by 8 to 16 percentage points depending on the attack on the vanilla ResNet110, and the NS detector by up to 5 percentage points. LAP training improves the detection accuracy of our detector by 1.5 percentage points on average and that of the Mahalanobis detector by a substantial 8.2 percentage points on average.

Results for ResNeXt50 on CIFAR100 are in the second row of Table 12. Our detector outperforms the Mahalanobis detector by 1 to 5 percentage points

depending on the attack on the vanilla ResNeXt50, and the NS detector by up to 3 percentage points. LAP training improves the detection accuracy of both detectors by 1 percentage point on average.

Results for a WideResNet on TinyImageNet are in the third row of Table 12. Our detector outperforms the Mahalanobis detector by 3 to 15.5 percentage points depending on the attack on the vanilla WideResNet, and the NS detector slightly. LAP training improves the detection accuracy of the Mahalanobis detector by 0.6 percentage points on average.

Detection rates of successful adversarial samples (i.e. those that fool the network) are in Table 38 in Appendix H.4 and are higher than 95% on our detector. False positive rates are in Table 40 in Appendix H.5 and are always less than 5% on our detector. The AUROC is in Table 42 in Appendix H.6. On all these metrics, our detector outperforms the other detectors, and LAP-training greatly improves the performance of the Mahalanobis detector.

Table 12: Average detection accuracy of seen attacks on Network/LAP-Network.

		Attack							
Network/Data	Det	FGM	APGD	BIM	AA	DF	CW	HSJ	BA
ResNet110 CIFAR10	TR	97.1/**98.7**	94.1/**97.5**	97.5/**99.3**	88.9/**94.1**	**99.9**/99.8	**98.0**/**98.0**	**99.9**/**99.9**	96.6/**97.0**
	MH	87.8/95.6	82.1/90.7	86.8/95.4	80.5/90.0	91.5/96.7	85.6/93.4	85.5/94.6	80.2/89.6
	NS	94.6	94.3	95.0	88.8	99.8	93.9	99.7	92.1
ResNeXt50 CIFAR100	TR	97.3/**98.3**	96.0/**97.8**	98.0/**98.9**	84.9/**87.6**	**99.8**/99.6	97.0/**97.8**		
	MH	95.8/96.8	93.9/94.6	96.1/97.8	83.9/86.6	97.3/97.1	95.4/96.4		
	NS	94.7	94.2	94.7	84.8	99.6	90.7		
WideResNet TinyImageNet	TR	**95.4**/95.1	**95.2**/**95.2**	**95.3**/95.1	**81.4**/81.2				
	MH	81.1/82.3	79.7/80.6	81.2/82.5	78.4/78.4				
	NS	94.9	94.9	95.0	81.3				

5.4.3 Detection of Unseen Attacks

An important setting is when we don't know which attack might be used at test time or only have time to train on one attack. We still want our detector to generalize well to other attacks. To test this, we use the same vanilla networks as above but only train the detectors on the detection training set created using the simplest and quickest attack (FGM) and test them on the detection test sets created using the six other attacks.

Results are in Table 13 below. We see that our detection method has very good detection generalization to unseen attacks, even those very different from FGM, comfortably better than the Mahalanobis detector, by up to 19 percentage points, while the NS detector only generalizes to variants of FGM (APGD and BIM), and fails on the other attacks. These results are averaged over 5 runs and the standard deviations are in Tables 35, 36 and 37 in Appendix H.3.

Table 13: Average detection accuracy of unseen attacks after training on FGM.

		Attack						
Network/Data	Detector	APGD	BIM	AA	DF	CW	HSJ	BA
ResNet110 CIFAR10	TR	89.32	**96.02**	**85.10**	**91.02**	**93.18**	**93.00**	**90.92**
	MH	77.34	77.24	72.12	80.12	79.92	79.70	79.32
	NS	**92.08**	93.88	51.82	51.40	50.84	52.12	59.88
ResNeXt50 CIFAR100	TR	91.94	**95.02**	**73.32**	**85.16**	**78.18**	**85.04**	**92.14**
	MH	90.86	93.16	73.08	82.72	76.44	82.82	84.46
	NS	**92.16**	93.88	51.32	51.62	51.02	52.04	57.90
WideResNet TinyImageNet	TR	93.26	**94.66**	**77.04**	**90.62**	**91.42**		
	MH	76.96	77.02	60.36	73.18	75.52		
	NS	**94.06**	94.62	65.60	72.82	71.96		

On our detector, the detection rate of successful adversarial samples from unseen attacks remains higher than 90% in most cases (Table 39 in Appendix H.4) and the FPR is always lower than 10% (Table 41 in Appendix H.5). The AUROC is in Table 43 in Appendix H.6. Our detector almost always outperforms the other detectors on all these metrics. On the attacks that are not variants of FGM, the NS detector almost never predicts that an input is adversarial.

However, this experiment shows that our approach has some limitations. We see in Tables 35, 36 and 37 in Appendix H.3 that LAP training does not improve detection accuracy as much anymore in this setting, and reduces it in some cases. It improves it for the Mahalanobis detector on all attacks on the ResNet110-CIFAR10 and WideResNet-TinyImageNet experiments (by up to 10 percentage points), but not on most attacks on the ResNeXt50-CIFAR100 experiment. It often reduces detection generalization on our detector. But it always does better than RCE training in Tables 35 and 36 in Appendix H.3.

We claim this is because these methods reduce the variance of features extracted during training on the seen attack, thus harming generalization to unseen attacks. This explains why detection of PGD and BIM, which are variants of FGM, still often improves. Fixing this could be an area of future research. That LAP training improves detection of unseen OOD samples in some cases in the out-of-distribution experiment in Tables 14 and 15 in Section 5.4.4 indicates that this problem is not general.

5.4.4 Detection of Out-of-Distribution Samples

Since our analysis applies to all out-of-distribution (OOD) samples, we test detection of OOD samples in a similar setting to the Mahalanobis paper [Lee et al., 2018]. We use the same ResNet110 and ResNeXt50 models trained on CIFAR10 and CIFAR100 respectively. Since the detectors need to be trained, we are in the OOD setting where we have a first dataset for training the network (CIFAR10 in Table 14 and CIFAR100 in Table 15) and a second dataset from another distribution that is not the test OOD distribution to train the detector

on. This could be another dataset (CIFAR100 in the first column of Table 14), some images found in the wild, or a perturbation of our dataset that we generate using an adversarial attack (CW on CIFAR10 in the second column of Table 14, and AA and CW on CIFAR100 in Table 15). Detectors can then be used by training them to distinguish between these first two datasets, and then testing them on distinguishing between the first dataset and a third unseen dataset (SVHN [Netzer et al., 2011] in both tables). Results are in Tables 14 and 15.

Table 14: Average OOD detection accuracy and standard deviation over 5 runs using ResNet110 trained on CIFAR10.

	OOD Experiment 1		OOD Experiment 2	
Detector	CIFAR100 (seen)	SVHN (unseen)	CW-CIFAR10 (seen)	SVHN (unseen)
VAN TR	98.30 ± 0.46	97.46 ± 0.49	**97.42** ± 0.57	**91.38** ± 0.95
RCE TR	**98.42** ± 0.40	98.20 ± 0.39	91.54 ± 6.06	77.58 ± 6.72
LAP TR	98.30 ± 0.22	**98.50** ± 0.47	97.28 ± 0.62	85.46 ± 2.64
VAN MH	86.88 ± 1.52	91.28 ± 0.92	81.80 ± 1.96	83.76 ± 1.13
RCE MH	94.82 ± 0.45	92.16 ± 0.57	76.74 ± 2.75	54.24 ± 3.46
LAP MH	94.84 ± 0.41	90.46 ± 1.45	89.68 ± 0.65	76.72 ± 1.73

Table 15: Average OOD detection accuracy and standard deviation over 5 runs using ResNeXt50 trained on CIFAR100.

	OOD Experiment 1		OOD Experiment 2	
Detector	AA-CIFAR100 (seen)	SVHN (unseen)	CW-CIFAR100 (seen)	SVHN (unseen)
VAN TR	84.48 ± 0.59	75.32 ± 0.62	95.82 ± 0.67	**92.92** ± 1.36
RCE TR	50.04 ± 0.07	55.44 ± 5.76	76.48 ± 0.75	75.66 ± 0.68
LAP TR	**87.10** ± 0.12	72.98 ± 2.72	**95.94** ± 0.57	85.94 ± 2.88
VAN MH	83.44 ± 0.48	78.82 ± 0.48	94.96 ± 0.81	85.10 ± 1.50
RCE MH	50.04 ± 0.07	58.74 ± 2.36	76.20 ± 0.72	72.20 ± 1.47
LAP MH	86.04 ± 0.31	**86.84** ± 0.68	94.82 ± 0.34	88.92 ± 1.26

Our detector performs very well and better than the MH detector in three of the four experiments, and in the fourth case, the MH detector benefits from LAP training by 8 percentage points (first experiment in Table 15).Without any extra data available, using the CW adversarial attack allows to detect OOD samples from an unseen distribution with more than 90% accuracy and an FPR of less than 10% at a fixed TPR of 95%. The choice of the attack is also important, as CW allows for much better detection of unseen samples from SVHN than AA.

The AUROC and the false positive rate at a fixed true positive rate of 95% are in Appendix H.7, in Tables 44 and 45 for ResNet100, and in Tables 46 and 47 for ResNeXt50. They show the same advantage for our detector and regularization over the Mahalanobis detector and RCE training respectively.

5.4.5 Attacking the Detector

We consider here the case where the attacker also attacks the detector (adaptive attacks). We try two such attacks on the TR and MH detectors on ResNet110 trained on CIFAR10. Both attacks are white-box with respect to the network. The first is black-box with respect to the detector. It only knows if an adversarial sample has been detected or not. The second has some knowledge about the detector. It knows what features it uses and can attack it directly to find adversarial features. We test these attacks by looking at the percentage of detected successful adversarial samples that they turn into undetected successful adversarial samples that fool both the network and the detector.

The first attack proceeds as follows. A strong white-box attack (CW) is used on the network on image x that has label y. If it finds a successful adversarial image $\tilde{x}$ that fools the network into predicting $\tilde{y} \neq y$ but is detected by the detector, the attacker will attempt to modify this image $\tilde{x}$ so that the network and the detector are both fooled. For this, the image $\tilde{x}$ is used as the initialization for an attack (HSJ with a budget of 50 iterations and 10000 evaluations) on a black-box Network-Detector system. The attacker considers that the Network-Detector behaves as follows: it outputs the class prediction of the network if the detector does not detect an attack and outputs an additional 'detected' class if the detector detects an attack. The attacker attacks this Network-Detector on image $\tilde{x}$ targeting the $\tilde{y}$ label. This way the network makes a mistake and the 'detected' class is avoided. On the vanilla ResNet110, this attack turns 16.5% of 1700 detected successful adversarial samples $\tilde{x}$ into undetected successful adversarial samples on our detector, compared to 25.7% on the Mahalanobis detector. These percentages are lower on the LAP-ResNet110 as they drop to 6.8% on our detector and 12.9% on the Mahalanobis detector. This shows that LAP training improves the robustness of both adversarial detectors to being attacked themselves, and that the Transport detector is more robust than the MH detector.

The second attack is very similar to the adaptive attack used in [Carlini and Wagner, 2017a] to break the Kernel Density detector of [Feinman et al., 2017]. It proceeds as follows. A strong white-box attack (CW) is used on the network on image x that has label y. If it finds a successful adversarial image $\tilde{x}$ that fools the network but is detected by the detector, the detection features $\tilde{z}$ that $\tilde{x}$ generates when run through the network are used as the initialization for a black-box attack (HSJ with a budget of 50 iterations and 10000 evaluations) on the detector. If successful adversarial detection features z^* that fool the detector are found, the attacker has to find an adversarial perturbation of x that still fools the network and that generates these features z^* (or close features that also fool the detector) when run through the network. We do this as in [Carlini and Wagner, 2017a] by solving the following optimization problem:

$$\min_{x^*} -L(N(x^*), y) + c_1 \|D(x^*) - z^*\| + c_2 \|x^* - x\| \tag{42}$$

where L is the cross-entropy loss, N is the network, and D is the (differentiable)

function that returns the detection features of its input. This optimization problem is differentiable and we try differentiable optimization algorithms such as BFGS and NR to solve it. The initial detected successful adversarial image $\tilde{x}$ is used as initialization as in [Carlini and Wagner, 2017a]. This attack turns 14% of detected successful adversarial samples $\tilde{x}$ into undetected successful adversarial samples on our detector on the LAP-ResNet110.

Given that initial detection rates of successful adversarial samples are very often close to 100% (see Tables 38 and 39 in Appendix H.4), this shows that adaptive attacks do not (at least not easily) circumvent the detector, as detection rates drop to 85% at worst.

The second attack is stronger than the first one, but it can probably still be improved by using a white-box attack that is specific to random forests for attacking the detector such as [Kantchelian et al., 2016] or [Zhang et al., 2020], or a different loss than cross-entropy such as the one used in the CW attack. The choice of the differentiable term representing the detector (here $\|D(x^*) - z^*\|$ in (42)) is also important. However, the difficulty of combining the attack on the network with that on the detector remains. It is the non-differentiability of the random forest that forces either this separate treatment of network and detector then the use of a proxy differentiable term for the detector in (42) to combine both, or the use of a black-box method as in the first attack.

Also, we did not consider here the ensemble of the class-conditional detector and the general detector, which is the best performing version of the detector (see Section 5.4.2), and should be more robust to adaptive attacks, as the attacker will have to fool two random forest detectors at once and target a particular label, constraining further the optimization problem he solves. This is similar to [Eniser et al., 2020] where a random detector is selected at test time from a pool of trained detectors to render adaptive attacks on the detector more difficult.

5.4.6 Time Comparison

With a ResNeXt50 on CIFAR100 and a Tesla V100 GPU, it takes our method (including the time to generate FGM attacks) 66 seconds to extract its features from both the clean and the adversarial samples, while it takes the Mahalanobis method 110 seconds. Mahalanobis also extracts some statistics from the training set prior to adversarial training, which takes an additional 35 seconds. Our feature vector is of size 32, compared to 5 for the Mahalanobis detector. So our random forest takes only 4 more seconds to train than the Mahalanobis one (7 vs 3 seconds). Computation of the features our detector uses (norms and cosines) is in $O(MD)$, where M is the number of residual blocks and D is the largest embedding dimension inside the network.

5.5 Discussion and Conclusion

To conclude, we proposed a method for detecting abnormal samples, based on the dynamical viewpoint of neural networks. The method examines the discrete

vector field moving the inputs to distinguish clean and adversarial samples. The detector requires minimal processing and computation to extract the features it needs for detection and achieves state-of-the-art detection performances on 3 different architectures and datasets, on seen and unseen attacks, and also on out-of-distribution detection.

The low dimension of the features we extract (2 per residual block) allows to keep the features from all blocks as opposed to other detection methods that only experimentally select which layers to use. This could explain the superior performance of our detector, as we closely monitor the evolution of the input's embeddings inside the network.

We have also used the transport regularization that was previously shown to improve test accuracy to improve the detection accuracy of adversarial detectors. This regularization provably makes the embeddings closer to each other on the support of the input distribution, thus making them more distinguishable from those of abnormal samples. The main difference with other regularizations that make the network more Lipschitz by regularizing its weights directly is that we only endue the network with additional regularity on the support of the data distribution and not on the entire space.

A possible simple extension of our detection method would be to add cosines to other orthogonal vectors as detection features. A trade-off between the accuracy of the detector and the size and computation time of the features would then have to be taken into account.

6 Conclusion

6.1 Summary

Residual connections are among the workhorses that make neural networks successful. We have expanded on interpretations of their functioning, namely, the iterative refinement bias (i.e. the bias to remain close to the identity function) and the discrete dynamical system interpretation. We use optimal transport theory to theoretically analyze their functioning. In optimal transport terms, these two interpretations are unified to show that residual connections induce a desirable bias towards networks with low kinetic energy and transport cost. This analysis allowed us to propose improvements to the accuracy and stability of residual networks and applications to module-wise training that make it competitive with end-to-end training with only a fraction of the memory requirements and to detection of adversarial and out-of-distribution samples.

In a first work [Karkar et al., 2020] detailed in Section 3, we empirically motivated and formulated the following learning problem: among all the networks that solve the task, find the one that transforms the data with the lowest cost. This amounts to regularizing the transport cost of the network. We proved existence and regularity of models trained this way, studied their behavior in low-dimensional settings and shown their efficiency on standard classification tasks. Training is stabilized in an adaptive fashion without being slowed down.

In a second work [Karkar et al., 2023a] detailed in Section 4, we used this regularization for module-wise training of neural networks without end-to-end backpropagation, including on Transformers, saving up to 60% memory usage. We proved that this makes every module apply a proximal step in the Wasserstein space to maximize the separability of the data.

In a third work [Karkar et al., 2023b] detailed in Section 5, we proposed a detector of adversarial and out-of-distribution samples based on the dynamic viewpoint of neural networks. We also used our detector successfully on detection of unseen out-of-distribution samples. The transport regularization provably makes the activations of out-of-distribution samples more distinguishable from those of clean samples, thus improving the accuracy of adversarial detectors.

6.2 Future Directions

Besides the direct possible extensions mentioned in the conclusions to Sections 3 to 5, other avenues of research follow from our analysis and point of view.

6.2.1 Adaptive Depth During Training

6.2.1.1 Introduction

The dynamical point of view of neural networks, initiated by the analogy between residual networks and the Euler scheme, has led to new architectures [Lu et al.,

2018, Haber et al., 2019]. While these new architectures often imitate numerical solvers of higher order than the Euler scheme through skip connections, adaptive solvers have only yet been explored in a few papers. We propose to use them to adjust the step size and the depth of the network during training, possibly reducing training cost. Given that adaptive methods are made up of multiple imbedded schemes, this also allows to ensemble different pre-classification forward passes at inference, without an additional training cost.

6.2.1.2 Method

This is currently a work in progress with positive results on the CIFAR100 dataset. We build a neural network that imitates the structure of an adaptive Runge-Kutta scheme (see Appendices A.2 and A.3). The block we use contains 4 convolutional functions denoted f_i and does the following computations for an input x with a step size h:

$$\begin{cases} p_1 & = f_1(x) \\ y_{\text{euler}} & = x + hp_1 \\ p_2 & = f_2(x + \frac{1}{2}hp_1) \\ y_{\text{midpoint}} & = x + hp_2 \\ p_3 & = f_3(x + \frac{3}{4}hp_2) \\ y_{\text{order3}} & = x + \frac{2}{9}hp_1 + \frac{1}{3}hp_2 + \frac{4}{9}hp_3 \\ p_4 & = f_4(y_{\text{order3}}) \\ y_{\text{order2}} & = x + \frac{7}{24}hp_1 + \frac{1}{4}hp_{n,2} + \frac{1}{3}hp_3 + \frac{1}{8}hp_4 \end{cases} \tag{43}$$

Depending on the numerical scheme we want to use, the block forwards one of the outputs y to the next block. Only layers f_1 and f_2 will be trained for example if we want to use the midpoint scheme. This allows to compare the efficiency of the different schemes for solving classification tasks, i.e. is it better to train 1 block of order 3 or 3 blocks of order 1 (since the order 3 method contains 3 functions f_i)?

If all the functions f_i are learned, and since all the values y approximate the same differential equation, it makes sense to average them after the last block before the classification layer as an ensembling method.

The depth is adaptively adjusted during training as follows. First, a tolerance hyperparameter $\varepsilon > 0$ is fixed. Then, on the validation set, every S epochs, the average error $\texttt{err} = \|y_{\text{order3}} - y_{\text{order2}}\|_2$ for each block is computed. If it is larger than ε then the step size is halved and the block is duplicated, i.e. a block with the same weights is added right after it. If the error is smaller than $\varepsilon/3$, then the step size is doubled and the following block is deleted.

To summarize, the contributions we aim to verify on larger datasets are:

– Using neural blocks of higher order is more efficient than using normal Euler residual blocks, meaning that even though the blocks are larger,

fewer are required to reach higher accuracy with less memory usage.

- Since adaptive scheme are made-up of interwoven schemes of different order, different forward passes representing the schemes of different order can be ensembled at test time, through averaging the final embeddings before classification, to improve accuracy.
- Adaptively adjusting the step-sizes and depth of the network allows to reduce, since the first training epochs, the size of a large network, while still obtaining the same performance at the end of training.
- This architecture also benefits from LAP training as in Section 3.

6.2.1.3 Related Work

[Yang et al., 2020a] introduce a time-step controller that is an LSTM [Hochreiter and Schmidhuber, 1997] which is dependent on the convolution weights of the current and previous layers and that is jointly optimized with the network. In contrast, our time-step is adjusted as in adaptive numerical schemes, therefore requiring no extra training, and also adjusts the depth of the network during training. [Chang et al., 2018b] use the analogy between ResNets and numerical schemes for differential equations to accelerate training by starting the training with a shallow network with a large step-size and then doubling the depth and halving the step-size during training. This is not an adaptive method to adjust the depth and step-size, as it is not dependent on the stability of the approximated differential equation. [Zhu et al., 2022] simply implement an architecture that imitates a Runge-Kutta scheme. However, the coefficients are not fixed to the theoretical values of the scheme and are instead learned inside the convolutions. The relations between them necessary to the theoretical validity of the scheme are therefore not enforced. There is also no modification of the step-size and depth. Likewise, [Lu et al., 2018] use an architecture that imitates the linear multi-step numerical method and [Haber et al., 2019] imitate the implicit-explicit method. Modifying the step-size has shown its benefits in [Zhang et al., 2019, Zhang and Wynter, 2018], where fixing it to a small value around 0.1 makes training more stable.

Somewhat similar to our proposed method is the adaptive computation time method applied to residual networks [Figurnov et al., 2017], inspired by a similar idea applied to recurrent neural networks [Graves, 2017, Fojo et al., 2018, Jernite et al., 2017]. In these methods, secondary neural networks are trained to choose (minimize) the number of times the state transition is applied for an input step in a recurrent network, and the number of residual blocks applied for an input in a residual network. The goal in [Figurnov et al., 2017] is faster computation time and not the adjustment during training of the size of the network to the dataset at hand, as the number of residual blocks applied is chosen for each input (or even patch of the input), so the entire network still has to be trained.

6.2.2 Similarity of Neural Network Representations

We propose to use the study of the dynamics inside a neural network, especially as used in Section 5 to detect adversarial and out-of-distribution samples, to consider the problem of measuring similarity between the representations learned by neural networks. This area of research [Raghu et al., 2017, Kornblith et al., 2019, Laakso and Cottrell, 2000, Li et al., 2015, Morcos et al., 2018, Wang et al., 2018] tries to answer questions such as:

- Do different architectures learn similar representations when trained on the same dataset?
- Does the same architecture tend to learn similar representations when trained on similar datasets ?
- How much does changing the initialization change the learned representations?
- Can we establish correspondences between layers of different networks?

The available methods usually propose similarity measures (e.g. SVCCA [Raghu et al., 2017] and CKA [Kornblith et al., 2019]) between activations that have certain desirable invariances such as invariance to orthogonal transformations.

Optimal transport theory was used in [Singh and Jaggi, 2020] to establish correspondences between the neurons of two networks, which is then useful for fusing these networks into a network that keeps as much of the capabilities of each network as possible. However, [Singh and Jaggi, 2020] do this only layer by layer and not across layers, which could be an extension of their method.

6.2.3 Optimal Transport between Different Dimensions

Another avenue of investigation is to use optimal transport between distributions that are supported in spaces of different dimensions [McCann and Pass, 2019, Pass, 2010] to regularize pooling and embedding layers that change the dimension of their input. This could extend the work done here to different architectures, and could be particularly useful to regularizing generative network where we would like to start generating in lower resolutions in the earlier layers before upsampling to use less resources and computation time.

6.2.4 Simulating Continuity Partial Differential Equations

In Appendix C.3, the minimizing movement scheme defined in Definition 3 in Section 4.3.2 as

$$\rho_{k+1}^{\tau} \in \arg\min_{\rho \in \mathcal{P}(\Omega)} \ \mathcal{Z}(\rho) + \frac{1}{2\tau} W_2^2(\rho, \rho_k^{\tau})$$

for $\mathcal{Z} : \mathbb{W}_2(\Omega) \to \mathbb{R}$, is shown to converge, under some conditions on $\mathcal{Z}$, as τ goes to 0, to a curve $\tilde{\rho}_t$ in $\mathbb{W}_2(\Omega)$ that solves the continuity equation

$$\partial_t \rho_t(x) - \nabla \cdot (\rho_t(x) \nabla (\frac{\delta \mathcal{Z}}{\delta \rho}(\rho_t))(x)) = 0$$

where $\frac{\delta \mathcal{Z}}{\delta \rho}$ is the *first variation* of $\mathcal{Z}$ defined in Definition 20 in Appendix C.3.

For example, if $\mathcal{Z}(\rho) = \int g(\rho(x)) \, \mathrm{d}x$ with $g(x) = x \log x$, we obtain the *heat equation* $\partial_t \rho_t - \Delta \rho = 0$. And if $\mathcal{Z}(\rho) = \int g(\rho(x)) \, \mathrm{d}x + \int V \, \mathrm{d}\rho$, we obtain the *Fokker-Planck equation* $\partial_t \rho_t - \Delta \rho - \nabla \cdot (\rho \nabla V) = 0$.

This suggests that solving the problems in the minimizing movement scheme by approximating optimal transport maps through residual neural networks could be a way to discretize and numerically solve such continuity partial differential equations.

6.3 Conclusion

This principle of least action is central in physics. It can be found in classical and relativistic mechanics, thermodynamics and quantum mechanics [García-Morales et al., 2008, Feynman, 1942, Gray, 2009]. It states that the dynamical trajectory of a system is one that makes a certain action locally stationary. Since essentially all the laws of physics can be derived through some version of this principle, this suggests that nature operates in an optimal and efficient way. The principle of least action was therefore extended to derive laws of mathematical economy [Kombarov, 2021] and of evolution and natural selection [Kaila and Annila, 2008]. Neural networks that respect conservation laws have been proposed to approximate physical systems where conservation of energy is important [Greydanus et al., 2019].

Here, we have found that this principle is also central to finding good solutions to machine learning problems outside of physics, and to doing so while minimizing the resources and computations required. In particular, it explains why many techniques such residual connections, batch normalizations and orthogonal initialization have been so successful in deep learning, and opens the door to new considerations when designing novel neural networks and their training methods.

References

[Aldahdooh et al., 2022] Aldahdooh, A., Hamidouche, W., Fezza, S. A., and Déforges, O. (2022). Adversarial example detection for dnn models: a review and experimental comparison. *Artificial Intelligence Review*.

[Alemany and Pissinou, 2022] Alemany, S. and Pissinou, N. (2022). The dilemma between data transformations and adversarial robustness for time series application systems. In *Proceedings of the Workshop on Artificial Intelligence Safety 2022 (SafeAI 2022) co-located with the Thirty-Sixth AAAI Conference on Artificial Intelligence (AAAI2022), Virtual, February, 2022*, volume 3087 of *CEUR Workshop Proceedings*. CEUR-WS.org.

[Alvarez-Melis and Fusi, 2021] Alvarez-Melis, D. and Fusi, N. (2021). Dataset dynamics via gradient flows in probability space. *Proceedings of the International Conference on Machine Learning (ICML 2021)*.

[Amari, 1967] Amari, S. (1967). A theory of adaptive pattern classifiers. *IEEE Transactions on Electronic Computers*, EC-16(3):299–307.

[Ambrosio et al., 2005] Ambrosio, L., Gigli, N., and Savare, G. (2005). *Gradient Flows in Metric Spaces and in the Space of Probability Measures*. Birkhäuser Basel.

[Amodei et al., 2016] Amodei, D., Olah, C., Steinhardt, J., Christiano, P., Schulman, J., and Mané, D. (2016). Concrete problems in ai safety. *arXiv*.

[Amsaleg et al., 2015] Amsaleg, L., Chelly, O., Furon, T., Girard, S., Houle, M. E., Kawarabayashi, K.-i., and Nett, M. (2015). Estimating local intrinsic dimensionality. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, page 29–38, New York, NY, USA. Association for Computing Machinery.

[Andriushchenko et al., 2020] Andriushchenko, M., Croce, F., Flammarion, N., and Hein, M. (2020). Square attack: a query-efficient black-box adversarial attack via random search. In *Computer Vision – ECCV 2020*. Springer.

[Ansari et al., 2021] Ansari, A. F., Ang, M. L., and Soh, H. (2021). Refining deep generative models via discriminator gradient flow. In *International Conference on Learning Representations (ICLR 2021)*.

[Arbel et al., 2019] Arbel, M., Korba, A., SALIM, A., and Gretton, A. (2019). Maximum mean discrepancy gradient flow. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems (NeurIPS 2019)*, volume 32. Curran Associates, Inc.

[Arora et al., 2018] Arora, R., Basu, A., Mianjy, P., and Mukherjee, A. (2018). Understanding deep neural networks with rectified linear units. In *International Conference on Learning Representations (ICLR 2018)*.

[Arora et al., 2014] Arora, S., Bhaskara, A., Ge, R., and Ma, T. (2014). Provable bounds for learning some deep representations. In *Proceedings of the International Conference on Machine Learning (ICML 2014)*.

[Ascher and Petzold, 1998] Ascher, U. and Petzold, L. (1998). *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. Other Titles in Applied Mathematics. Society for Industrial and Applied Mathematics (SIAM, 3600 Market Street, Floor 6, Philadelphia, PA 19104).

[Ascher et al., 1997] Ascher, U. M., Ruuth, S. J., and Spiteri, R. J. (1997). Implicit-explicit runge-kutta methods for time-dependent partial differential equations. *Applied Numerical Mathematics*, 25(2):151–167. Special Issue on Time Integration.

[Ascher et al., 1995] Ascher, U. M., Ruuth, S. J., and Wetton, B. T. R. (1995). Implicit-explicit methods for time-dependent partial differential equations. *SIAM Journal on Numerical Analysis*, 32(3):797–823.

[Athans and Falb, 2007] Athans, M. and Falb, P. (2007). *Optimal Control: An Introduction to the Theory and Its Applications*. Dover Books on Engineering. Dover Publications.

[Ba et al., 2016] Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. *arXiv*.

[Bach, 2017] Bach, F. (2017). Breaking the curse of dimensionality with convex neural networks. *Journal of Machine Learning Research (JMLR 18)*, 18(19):1–53.

[Balduzzi et al., 2017] Balduzzi, D., Frean, M., Leary, L., Lewis, J., Ma, K. W.-D., and McWilliams, B. (2017). The shattered gradients problem: If resnets are the answer, then what is the question? In *Proceedings of the International Conference on Machine Learning (ICML 2017)*.

[Baluja and Fischer, 2017] Baluja, S. and Fischer, I. (2017). Adversarial transformation networks: Learning to generate adversarial examples. *arXiv*.

[Baydin et al., 2022] Baydin, A. G., Pearlmutter, B. A., Syme, D., Wood, F., and Torr, P. (2022). Gradients without backpropagation. *arXiv*.

[Belilovsky et al., 2019] Belilovsky, E., Eickenberg, M., and Oyallon, E. (2019). Greedy layerwise learning can scale to imagenet. In *Proceedings of the International Conference on Machine Learning (ICML 2019)*.

[Belilovsky et al., 2020] Belilovsky, E., Eickenberg, M., and Oyallon, E. (2020). Decoupled greedy learning of cnns. In *Proceedings of the International Conference on Machine Learning (ICML 2020)*.

[Belkin et al., 2018] Belkin, M., Ma, S., and Mandal, S. (2018). To understand deep learning we need to understand kernel learning. In *Proceedings of the International Conference on Machine Learning (ICML 2018)*.

[Belkin et al., 2019] Belkin, M., Ma, S., and Mandal, S. (2019). Reconciling modern machine-learning practice and the classical bias–variance trade-off. In *Proceedings of the National Academy of Sciences (PNAS)*, volume 116, pages 15849–15854.

[Benamou and Brenier, 2000] Benamou, J. and Brenier, Y. (2000). A computational fluid mechanics solution to the monge-kantorovich mass transfer problem. *Numerische Mathematik*.

[Bengio et al., 2006] Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. (2006). Greedy layer-wise training of deep networks. In Schölkopf, B., Platt, J., and Hoffman, T., editors, *Advances in Neural Information Processing Systems (NeurIPS 2006)*, volume 19. MIT Press.

[Bernd Illing, 2020] Bernd Illing, Wulfram Gerstner, G. B. (2020). Towards truly local gradients with clapp: Contrastive, local and predictive plasticity. *arXiv*.

[Bertasius et al., 2021] Bertasius, G., Wang, H., and Torresani, L. (2021). Is space-time attention all you need for video understanding? In Meila, M. and Zhang, T., editors, *Proceedings of the 38th International Conference on Machine Learning (ICML 2021)*, volume 139 of *Proceedings of Machine Learning Research*, pages 813–824. PMLR.

[Bertsekas, 2016] Bertsekas, D. (2016). *Nonlinear Programming*. Athena scientific optimization and computation series. Athena Scientific.

[Bhowmick et al., 2021] Bhowmick, A., D'Souza, M., and Raghavan, G. S. (2021). Lipbab: Computing exact lipschitz constant of relu networks. *arXiv*, abs/2105.05495.

[Bolley, 2008] Bolley, F. (2008). Separability and completeness for the wasserstein distance. In *Séminaire de Probabilités XLI*. Springer, Berlin, Heidelberg.

[Bottou, 2012] Bottou, L. (2012). *Stochastic Gradient Descent Tricks*, pages 421–436. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Bottou et al., 2018] Bottou, L., Curtis, F. E., and Nocedal, J. (2018). Optimization methods for large-scale machine learning. *SIAM Review*, 60(2):223–311.

[Brendel et al., 2018] Brendel, W., Rauber, J., and Bethge, M. (2018). Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. In *International Conference on Learning Representations (ICLR 2018)*.

[Carlini and Wagner, 2017a] Carlini, N. and Wagner, D. (2017a). *Adversarial Examples Are Not Easily Detected: Bypassing Ten Detection Methods*, page 3–14. Association for Computing Machinery, New York, NY, USA.

[Carlini and Wagner, 2017b] Carlini, N. and Wagner, D. (2017b). Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium*

on Security and Privacy (SP), pages 39–57, Los Alamitos, CA, USA. IEEE Computer Society.

[Carrara et al., 2019] Carrara, F., Becarelli, R., Caldelli, R., Falchi, F., and Amato, G. (2019). Adversarial examples detection in features distance spaces. In Leal-Taixé, L. and Roth, S., editors, *Workshop at European Conference on Computer Vision (ECCV 2018)*, pages 313–327, Cham. Springer International Publishing.

[Chang et al., 2018a] Chang, B., Meng, L., Haber, E., Ruthotto, L., Begert, D., and Holtham, E. (2018a). Reversible architectures for arbitrarily deep residual neural networks. In *The AAAI Conference on Artificial Intelligence (AAAI-18)*.

[Chang et al., 2018b] Chang, B., Meng, L., Haber, E., Tung, F., and Begert, D. (2018b). Multi-level residual networks from dynamical systems view. In *International Conference on Learning Representations (ICLR 2018)*.

[Chattopadhyay et al., 2019] Chattopadhyay, N., Chattopadhyay, A., Gupta, S. S., and Kasper, M. (2019). Curse of dimensionality in adversarial examples. In *International Joint Conference on Neural Networks (IJCNN 2019)*.

[Chen et al., 2020a] Chen, J., Jordan, M. I., and Wainwright, M. J. (2020a). Hopskipjumpattack: A query-efficient decision-based attack. In *2020 IEEE Symposium on Security and Privacy*, pages 1277–1294. IEEE.

[Chen et al., 2017a] Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K., and Yuille, A. L. (2017a). Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI 2017)*, 40.

[Chen et al., 2017b] Chen, P.-Y., Zhang, H., Sharma, Y., Yi, J., and Hsieh, C.-J. (2017b). Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, AISec '17, page 15–26, New York, NY, USA. Association for Computing Machinery.

[Chen et al., 2018] Chen, R. T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. (2018). Neural ordinary differential equations. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems (NeurIPS 2018)*, volume 31. Curran Associates, Inc.

[Chen et al., 2020b] Chen, T., Kornblith, S., Norouzi, M., and Hinton, G. (2020b). A simple framework for contrastive learning of visual representations. In III, H. D. and Singh, A., editors, *Proceedings of the 37th International Conference on Machine Learning (ICML 2020)*, volume 119 of *Proceedings of Machine Learning Research*, pages 1597–1607. PMLR.

[Chen et al., 2022] Chen, Y., Guo, Z., Yin, Q., Chen, H., and Huang, K. (2022). Layer-wisely supervised learning for one-shot neural architecture search. In *2022 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8.

[Chernousko and Lyubushin, 1982] Chernousko, F. L. and Lyubushin, A. A. (1982). Method of successive approximations for solution of optimal control problems. *Optimal Control Applications and Methods*, 3(2):101–114.

[Chizat, 2020] Chizat, L. (2020). Sparse optimization on measures with over-parameterized gradient descent. *arXiv*.

[Chizat and Bach, 2018] Chizat, L. and Bach, F. (2018). On the global convergence of gradient descent for over-parameterized models using optimal transport. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems (NeurIPS 2018)*, volume 31. Curran Associates, Inc.

[Ciccone et al., 2018] Ciccone, M., Gallieri, M., Masci, J., Osendorfer, C., and Gomez, F. (2018). Nais-net: Stable deep networks from non-autonomous differential equations. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems (NeurIPS 2018)*, volume 31. Curran Associates, Inc.

[Cisse et al., 2017] Cisse, M., Bojanowski, P., Grave, E., Dauphin, Y., and Usunier, N. (2017). Parseval networks: Improving robustness to adversarial examples. In *Proceedings of the 34th International Conference on Machine Learning (ICML 2017)*, page 854–863. PMLR.

[Coate et al., 2011] Coate, A., Lee, H., and Ng, A. Y. (2011). An analysis of single layer networks in unsupervised feature learning. In *International Conference on Artificial Intelligence and Statistics (AISTATS 2011)*.

[Croce and Hein, 2020a] Croce, F. and Hein, M. (2020a). Minimally distorted adversarial examples with a fast adaptive boundary attack. In *Proceedings of the 37th International Conference on Machine Learning (ICML 2020)*. PMLR.

[Croce and Hein, 2020b] Croce, F. and Hein, M. (2020b). Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. In *Proceedings of the 37th International Conference on Machine Learning (ICML 2020)*. PMLR.

[Cybenko, 1989] Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2:303–314.

[Czarnecki et al., 2017] Czarnecki, W. M., Świrszcz, G., Jaderberg, M., Osindero, S., Vinyals, O., and Kavukcuoglu, K. (2017). Understanding synthetic gradients and decoupled neural interfaces. In *Proceedings of the International Conference on Machine Learning (ICML 2017)*.

[Dahlquist, 1963] Dahlquist, G. G. (1963). A special stability problem for linear multistep methods. *BIT Numerical Mathematics*, 3:27–43.

[De and Smith, 2020] De, S. and Smith, S. (2020). Batch normalization biases residual blocks towards the identity function in deep networks. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H., editors, *Advances in Neural Information Processing Systems (NeurIPS 2020)*, volume 33, pages 19964–19975. Curran Associates, Inc.

[de Bezenac et al., 2018] de Bezenac, E., Pajot, A., and Gallinari, P. (2018). Deep learning for physical processes: Incorporating prior scientific knowledge. In *International Conference on Learning Representations (ICLR 2018)*.

[de Bézenac et al., 2021] de Bézenac, E., Ayed, I., and Gallinari, P. (2021). Cyclegan through the lens of (dynamical) optimal transport. In *European Conference on Machine Learning and Knowledge Discovery in Databases (ECML PKDD 2021)*.

[De Palma et al., 2019] De Palma, G., Kiani, B., and Lloyd, S. (2019). Random deep neural networks are biased towards simple functions. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems (NeurIPS 2019)*, volume 32. Curran Associates, Inc.

[Dohmatob, 2019] Dohmatob, E. (2019). Limitations of adversarial robustness: Strong no free lunch theorem. In *Proceedings of the 36th International Conference on Machine Learning (ICML 2019)*, Proceedings of Machine Learning Research. PMLR.

[Dosovitskiy et al., 2021] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. (2021). An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations (ICLR 2021)*.

[Du and Goel, 2018] Du, S. S. and Goel, S. (2018). Improved learning of one-hidden-layer convolutional neural networks with overlaps. *arXiv*.

[Du et al., 2021] Du, X., Farrahi, K., and Niranjan, M. (2021). Information bottleneck theory based exploration of cascade learning. *Entropy*, 23(10).

[Duan and Príncipe, 2022] Duan, S. and Príncipe, J. C. (2022). Training deep architectures without end-to-end backpropagation: A survey on the provably optimal methods. *IEEE Computational Intelligence Magazine*, 17(4):39–51.

[Duan et al., 2022] Duan, S., Yu, S., and Príncipe, J. C. (2022). Modularizing deep learning via pairwise learning with kernels. *IEEE Transactions on Neural Networks and Learning Systems*, 33(4):1441–1451.

[Dun et al., 2021] Dun, C., Wolfe, C. R., Jermaine, C. M., and Kyrillidis, A. (2021). Resist: Layer-wise decomposition of resnets for distributed training. *arXiv*.

[Engstrom et al., 2019] Engstrom, L., Tran, B., Tsipras, D., Schmidt, L., and Madry, A. (2019). Exploring the landscape of spatial robustness. In Chaudhuri, K. and Salakhutdinov, R., editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 1802–1811. PMLR.

[Eniser et al., 2020] Eniser, H. F., Christakis, M., and Wüstholz, V. (2020). Raid: Randomized adversarial-input detection for neural networks. *arXiv*.

[et al., 2016] et al., Y. W. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *ArXiv*.

[Eykholt et al., 2018] Eykholt, K., Evtimov, I., Fernandes, E., Li, B., Rahmati, A., Xiao, C., Prakash, A., Kohno, T., and Song, D. (2018). Robust physical-world attacks on deep learning visual classification. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1625–1634, Los Alamitos, CA, USA. IEEE Computer Society.

[Feinman et al., 2017] Feinman, R., Curtin, R. R., Shintre, S., and Gardner, A. B. (2017). Detecting adversarial samples from artifacts. *arXiv*.

[Feynman, 1942] Feynman, R. P. (1942). *The principle of least action in quantum mechanics*. PhD thesis, Princeton U.

[Figalli, 2017] Figalli, A. (2017). *The Monge-Ampere Equation and Its Applications*. Zurich lectures in advanced mathematics. European Mathematical Society.

[Figurnov et al., 2017] Figurnov, M., Collins, M. D., Zhu, Y., Zhang, L., Huang, J., Vetrov, D., and Salakhutdinov, R. (2017). Spatially adaptive computation time for residual networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1790–1799.

[Finlay et al., 2020] Finlay, C., Jacobsen, J.-H., Nurbekyan, L., and Oberman, A. M. (2020). How to train your neural ode. In *Proceedings of the International Conference on Machine Learning (ICML 2020)*.

[Fojo et al., 2018] Fojo, D., Campos, V., and i Nieto, X. G. (2018). Comparing fixed and adaptive computation time for recurrent neural networks. In *Workshop at International Conference on Learning Representations (ICLR 2018)*.

[Fournier et al., 2023] Fournier, L., Rivaud, S., Belilovsky, E., Eickenberg, M., and Oyallon, E. (2023). Can forward gradient match backpropagation? In *Proceedings of the International Conference on Machine Learning (ICML 2023)*.

[Freund and Schapire, 1997] Freund, Y. and Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139.

[Fukushima, 1975] Fukushima, K. (1975). Cognitron: A self-organizing multilayered neural network. *Biological Cybernetics*, 20.

[Gai and Zhang, 2021] Gai, K. and Zhang, S. (2021). A mathematical principle of deep learning: Learn the geodesic curve in the wasserstein space. *arXiv*.

[Gao et al., 2019] Gao, Y., Jiao, Y., Wang, Y., Wang, Y., Yang, C., and Zhang, S. (2019). Deep generative learning via variational gradient flow. In *Proceedings of the International Conference on Machine Learning (ICML 2019)*.

[García-Morales et al., 2008] García-Morales, V., Pellicer, J., and Manzanares, J. A. (2008). Thermodynamics based on the principle of least abbreviated action: Entropy production in a network of coupled oscillators. *Annals of Physics*, 323(8):1844–1858.

[Ge et al., 2018] Ge, R., Lee, J. D., and Ma, T. (2018). Learning one-hidden-layer neural networks with landscape design. In *International Conference on Learning Representations (ICLR 2018)*.

[Gianazza et al., 1994] Gianazza, U., Gobbino, M., and Savarè, G. (1994). Evolution problems and minimizing movements. *Atti della Accademia Nazionale dei Lincei. Classe di Scienze Fisiche, Matematiche e Naturali. Rendiconti Lincei. Matematica e Applicazioni*, 5:289–296.

[Gilmer et al., 2018] Gilmer, J., Metz, L., Faghri, F., Schoenholz, S. S., Raghu, M., Wattenberg, M., and Goodfellow, I. (2018). Adversarial spheres: The relationship between high-dimensional geometry and adversarial examples. *arXiv*.

[Giorgi et al., 1980] Giorgi, E. D., Marino, A., and Tosques, M. (1980). Problemi di evoluzione in spazi metrici e curve di massima pendenza. *Atti della Accademia Nazionale dei Lincei. Classe di Scienze Fisiche, Matematiche e Naturali. Rendiconti*, 68(3):180–187.

[Gomez et al., 2020] Gomez, A. N., Key, O., Gou, S., Frosst, N., Dean, J., and Gal, Y. (2020). Interlocking backpropagation: Improving depthwise model-parallelism. *Journal of Machine Learning Research 23 (JMLR 23)*, 23:171:1–171:28.

[Gomez et al., 2017] Gomez, A. N., Ren, M., Urtasun, R., and Grosse, R. B. (2017). The reversible residual network: Backpropagation without storing activations. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems (NeurIPS 2017)*, volume 30. Curran Associates, Inc.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

[Goodfellow et al., 2015a] Goodfellow, I. J., Shlens, J., and Szegedy, C. (2015a). Explaining and harnessing adversarial examples. In *International Conference on Learning Representations (ICLR 2015)*.

[Goodfellow et al., 2015b] Goodfellow, I. J., Shlens, J., and Szegedy, C. (2015b). Explaining and harnessing adversarial examples. In *International Conference on Learning Representations (ICLR 2015)*.

[Graves, 2017] Graves, A. (2017). Adaptive computation time for recurrent neural networks. *arXiv*.

[Gray, 2009] Gray, C. G. (2009). Principle of least action. *Scholarpedia*, 4(12):8291.

[Greff et al., 2016] Greff, K., Srivastava, R. K., and Schmidhuber, J. (2016). Highway and residual networks learn unrolled iterative estimation. *arXiV*.

[Gretton et al., 2012] Gretton, A., Borgwardt, K. M., Rasch, M. J., Schölkopf, B., and Smola, A. (2012). A kernel two-sample test. *Journal of Machine Learning Research 13 (JMLR 13)*, 13.

[Greydanus et al., 2019] Greydanus, S., Dzamba, M., and Yosinski, J. (2019). Hamiltonian neural networks. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems (NeurIPS 2019)*, volume 32. Curran Associates, Inc.

[Grosse et al., 2017] Grosse, K., Manoharan, P., Papernot, N., Backes, M., and McDaniel, P. (2017). On the (statistical) detection of adversarial examples. *arXiv*.

[Gu and Rigazio, 2015] Gu, S. and Rigazio, L. (2015). Towards deep neural network architectures robust to adversarial examples. In *Workshop at International Conference on Learning Representations (ICLR 2015)*.

[Guo et al., 2022a] Guo, W., Fouda, M. E., Eltawil, A. M., and Salama, K. N. (2022a). Backlink: Supervised local training with backward links. *arXiv*.

[Guo et al., 2022b] Guo, W., Fouda, M. E., Eltawil, A. M., and Salama, K. N. (2022b). Efficient training of spiking neural networks with temporally-truncated local backpropagation through time. *arXiv*.

[Gupta, 2020] Gupta, S. K. (2020). A more biologically plausible local learning rule for anns. *arXiv*.

[Haber et al., 2019] Haber, E., Lensink, K., Treister, E., and Ruthotto, L. (2019). Imexnet - a forward stable deep neural network. In *Proceedings of the International Conference on Machine Learning (ICML 2019)*.

[Haber and Ruthotto, 2017] Haber, E. and Ruthotto, L. (2017). Stable architectures for deep neural networks. *CoRR*, abs/1705.03341.

[Harder et al., 2021] Harder, P., Pfreundt, F.-J., Keuper, M., and Keuper, J. (2021). Spectraldefense: Detecting adversarial attacks on cnns in the fourier domain. *arXiv*.

[Hastie et al., 2001] Hastie, T., Tibshirani, R., and Friedman, J. (2001). *The Elements of Statistical Learning*. Springer-Verlag New York.

[Hauser, 2019] Hauser, M. (2019). On residual networks learning a perturbation from identity. *arXiv*.

[Hayou and Ayed, 2021] Hayou, S. and Ayed, F. (2021). Regularization in resnet with stochastic depth. In Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems (NeurIPS 2021)*, volume 34, pages 15464–15474. Curran Associates, Inc.

[He et al., 2020] He, K., Fan, H., Wu, Y., Xie, S., and Girshick, R. (2020). Momentum contrast for unsupervised visual representation learning. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9726–9735, Los Alamitos, CA, USA. IEEE Computer Society.

[He et al., 2016a] He, K., Zhan, X., Ren, S., and Sun, J. (2016a). Identity mappings in deep residual networks. In *European Conference on Computer Vision (ECCV 2016)*.

[He et al., 2015] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034.

[He et al., 2016b] He, K., Zhang, X., Ren, S., and Sun, J. (2016b). Deep residual learning for image recognition. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR 2016)*.

[Hein and Andriushchenko, 2017] Hein, M. and Andriushchenko, M. (2017). Formal guarantees on the robustness of a classifier against adversarial manipulation. In *Advances in Neural Information Processing Systems (NeurIPS 2017)*, page 2263–2273, Red Hook, NY, USA. Curran Associates Inc.

[Hinton et al., 1995] Hinton, G. E., Dayan, P., Frey, B. J., and Neal, R. M. (1995). The "wake-sleep" algorithm for unsupervised neural networks. *Science*, 268(5214):1158–1161.

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9:1735–80.

[Hornik et al., 1989] Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366.

[Houle, 2013] Houle, M. E. (2013). Dimensionality, discriminability, density and distance distributions. In *Workshop at 2013 IEEE 13th International Conference on Data Mining*, pages 468–473.

[Hu and Tan, 2022] Hu, W. and Tan, Y. (2022). Generating adversarial malware examples for black-box attacks based on gan. In Tan, Y. and Shi, Y., editors, *Data Mining and Big Data*, pages 409–423, Singapore. Springer Nature Singapore.

[Huang et al., 2018] Huang, F., Ash, J., Langford, J., and Schapire, R. (2018). Learning deep resnet blocks sequentially using boosting theory. In *Proceedings of the International Conference on Machine Learning (ICML 2018)*.

[Huang et al., 2016] Huang, G., Sun, Y., Liu, Z., Sedra, D., and Weinberger, K. (2016). Deep networks with stochastic depth. *arXiv*.

[Huo et al., 2018a] Huo, Z., Gu, B., and Huang, H. (2018a). Training neural networks using features replay. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems (NeurIPS 2018)*, volume 31. Curran Associates, Inc.

[Huo et al., 2018b] Huo, Z., Gu, B., Yang, Q., and Huang, H. (2018b). Decoupled parallel backpropagation with convergence guarantee. In *Proceedings of the International Conference on Machine Learning (ICML 2018)*.

[Huster et al., 2019] Huster, T., Chiang, C.-Y. J., and Chadha, R. (2019). Limitations of the lipschitz constant as a defense against adversarial examples. In *Workshop at European Conference on Machine Learning and Knowledge Discovery in Databases (ECML PKDD 2019)*, pages 16–29. Springer International Publishing.

[Ioffe and Szegedy, 2015] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Bach, F. and Blei, D., editors, *Proceedings of the 32nd International Conference on Machine Learning (ICML 2015)*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France. PMLR.

[Jacobsen et al., 2018] Jacobsen, J.-H., Smeulders, A., and Oyallon, E. (2018). i-revnet: Deep invertible networks. In *International Conference on Learning Representations (ICLR 2018)*.

[Jacot et al., 2018] Jacot, A., Gabriel, F., and Hongler, C. (2018). Neural tangent kernel: Convergence and generalization in neural networks. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems (NeurIPS 2018)*, volume 31. Curran Associates, Inc.

[Jaderberg et al., 2017] Jaderberg, M., Czarnecki, W. M., Osindero, S., Vinyals, O., Graves, A., Silver, D., and Kavukcuoglu, K. (2017). Decoupled neural interfaces using synthetic gradients. In *Proceedings of the International Conference on Machine Learning (ICML 2017)*.

[Janner et al., 2022] Janner, M., Du, Y., Tenenbaum, J., and Levine, S. (2022). Planning with diffusion for flexible behavior synthesis. In *Proceedings of the 39th International Conference on Machine Learning (ICML)*.

[Janzamin et al., 2016] Janzamin, M., Sedghi, H., and Anandkumar, A. (2016). Beating the perils of non-convexity: Guaranteed training of neural networks using tensor methods. *arXiv*.

[Jastrzebski et al., 2018] Jastrzebski, S., Arpit, D., Ballas, N., Verma, V., Che, T., and Bengio, Y. (2018). Residual connections encourage iterative inference. In *International Conference on Learning Representations (ICLR 2018)*.

[Jernite et al., 2017] Jernite, Y., Grave, E., Joulin, A., and Mikolov, T. (2017). Variable computation in recurrent neural networks. In *International Conference on Learning Representations (ICLR 2017)*.

[Jordan et al., 1998] Jordan, R., Kinderlehrer, D., and Otto, F. (1998). The variational formulation of the fokker–planck equation. *SIAM Journal on Mathematical Analysis*, 29(1):1–17.

[Jumper et al., 2021] Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A., Bridgland, A., Meyer, C., Kohl, S. A. A., Ballard, A. J., Cowie, A., Romera-Paredes, B., Nikolov, S., Jain, R., Adler, J., Back, T., Petersen, S., Reiman, D., Clancy, E., Zielinski, M., and Hassabis, D. (2021). Highly accurate protein structure prediction with alphafold. *Nature*.

[Kaila and Annila, 2008] Kaila, V. R. and Annila, A. (2008). Natural selection for least action. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 464(2099):3055–3070.

[Kantchelian et al., 2016] Kantchelian, A., Tygar, J. D., and Joseph, A. D. (2016). Evasion and hardening of tree ensemble classifiers. In *Proceedings of the 33th International Conference on Machine Learning (ICML 2016)*. PMLR.

[Karkar et al., 2020] Karkar, S., Ayed, I., de Bézenac, E., and Gallinari, P. (2020). A principle of least action for the training of neural networks. In *European Conference on Machine Learning and Knowledge Discovery in Databases (ECML PKDD 2020)*.

[Karkar et al., 2023a] Karkar, S., Ayed, I., de Bézenac, E., and Gallinari, P. (2023a). Module-wise training of neural networks via the minimizing movement scheme. In *Advances in Neural Information Processing Systems (NeurIPS 2023)*.

[Karkar et al., 2023b] Karkar, S., Gallinari, P., and Rakotomamonjy, A. (2023b). Adversarial sample detection through neural network transport dynamics. In *European Conference on Machine Learning and Knowledge Discovery in Databases (ECML PKDD 2023)*.

[Kherchouche et al., 2022] Kherchouche, A., Fezza, S. A., and Hamidouche, W. (2022). Detect and defense against adversarial examples in deep learning using natural scene statistics and adaptive denoising. *Neural Computing and Applications*, 34(24):21567–21582.

[Kherchouche et al., 2020] Kherchouche, A., Fezza, S. A., Hamidouche, W., and Déforges, O. (2020). Detection of adversarial examples in deep neural networks with natural scene statistics. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7.

[Khosla et al., 2020] Khosla, P., Teterwak, P., Wang, C., Sarna, A., Tian, Y., Isola, P., Maschinot, A., Liu, C., and Krishnan, D. (2020). Supervised contrastive learning. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems (NeurIPS 2020)*, volume 33, pages 18661–18673. Curran Associates, Inc.

[Khoury and Hadfield-Menell, 2018] Khoury, M. and Hadfield-Menell, D. (2018). On the geometry of adversarial examples. *arXiv*.

[Kolouri et al., 2017] Kolouri, S., Park, S., Thorpe, M., Slepčev, D., and Rohde, G. K. (2017). Optimal mass transport: Signal processing and machine-learning applications. *IEEE signal processing magazine*.

[Kombarov, 2021] Kombarov, S. (2021). Action in economics: Mathematical derivation of laws of economics from the principle of least action in physics. In *Proceedings of International Conference of Eurasian Economies*.

[Kornblith et al., 2019] Kornblith, S., Norouzi, M., Lee, H., and Hinton, G. (2019). Similarity of neural network representations revisited. In Chaudhuri, K. and Salakhutdinov, R., editors, *Proceedings of the 36th International Conference on Machine Learning (ICML 2019)*, volume 97 of *Proceedings of Machine Learning Research*, pages 3519–3529. PMLR.

[Kotyan and Vargas, 2020] Kotyan, S. and Vargas, D. V. (2020). Adversarial robustness assessment: Why both l_0 and l_∞ attacks are necessary. *arXiv*.

[Krizhevsky, 2009] Krizhevsky, A. (2009). Learning multiple layers of features from tiny images. *University of Toronto Technical Report*.

[Krizhevsky et al., 2012] 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 (NeurIPS 2012*, volume 25. Curran Associates, Inc.

[Kurakin et al., 2017] Kurakin, A., Goodfellow, I., and Bengio, S. (2017). Adversarial examples in the physical world. In *Workshop at International Conference on Learning Representations (ICLR 2017)*.

[Laakso and Cottrell, 2000] Laakso, A. and Cottrell, G. (2000). Content and cluster analysis: Assessing representational similarity in neural systems. *Philosophical Psychology*, 13(1):47–76.

[Larsson et al., 2017] Larsson, G., Maire, M., and Shakhnarovich, G. (2017). Fractalnet: Ultra-deep neural networks without residuals. In *International Conference on Learning Representations (ICLR 2017)*.

[Laskin et al., 2021] Laskin, M., Metz, L., Nabarro, S., Saroufim, M., Noune, B., Luschi, C., Sohl-Dickstein, J., and Abbeel, P. (2021). Parallel training of deep networks with local updates. *arXiv*.

[Latorre et al., 2020] Latorre, F., Rolland, P., and Cevher, V. (2020). Lipschitz constant estimation of neural networks via sparse polynomial optimization. In *International Conference on Learning Representations (ICLR 2020)*.

[LeCun et al., 2010] LeCun, Y., Cortes, C., and Burges, C. J. (2010). MNIST handwritten digit database. *yann.lecun.com/exdb/mnist*.

[Lee et al., 2018] Lee, K., Lee, K., Lee, H., and Shin, J. (2018). A simple unified framework for detecting out-of-distribution samples and adversarial attacks. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems (NeurIPS 2018)*, volume 31. Curran Associates, Inc.

[Lengellé and Denœux, 1996] Lengellé, R. and Denœux, T. (1996). Training mlps layer by layer using an objective function for internal representations. *Neural Networks*, 9(1):83–97.

[Li et al., 2018] Li, Q., Chen, L., Tai, C., and E, W. (2018). Maximum principle based algorithms for deep learning. *Journal of Machine Learning Research (JMLR 2018)*.

[Li et al., 2022] Li, Y., Cheng, M., Hsieh, C.-J., and Lee, T. C. M. (2022). A review of adversarial attack and defense for classification methods. *The American Statistician*, 76(4):329–345.

[Li et al., 2015] Li, Y., Yosinski, J., Clune, J., Lipson, H., and Hopcroft, J. (2015). Convergent learning: Do different neural networks learn the same representations? In Storcheus, D., Rostamizadeh, A., and Kumar, S., editors, *Proceedings of the 1st International Workshop on Feature Extraction: Modern Questions and Challenges at Advances in Neural Information Processing Systems (NeurIPS 2015)*, volume 44 of *Proceedings of Machine Learning Research*, pages 196–212, Montreal, Canada. PMLR.

[Liao et al., 2016] Liao, Q., Leibo, J., and Poggio, T. (2016). How important is weight symmetry in backpropagation? *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1).

[Liao and Poggio, 2016] Liao, Q. and Poggio, T. (2016). Bridging the gaps between residual learning, recurrent neural networks and visual cortex. *arXiv*.

[Liu et al., 2017] Liu, Y., Chen, X., Liu, C., and Song, D. (2017). Delving into transferable adversarial examples and black-box attacks. In *International Conference on Learning Representations (ICLR 2017)*. OpenReview.net.

[Liu et al., 2021] Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., and Guo, B. (2021). Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.

[Liutkus et al., 2019] Liutkus, A., Imşekli, U., Majewski, S., Durmus, A., and Stoter, F.-R. (2019). Sliced-wasserstein flows: Nonparametric generative modeling via optimal transport and diffusions. In *Proceedings of the International Conference on Machine Learning (ICML 2019)*.

[Löwe et al., 2019] Löwe, S., O' Connor, P., and Veeling, B. (2019). Putting an end to end-to-end: Gradient-isolated learning of representations. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems (NeurIPS 2019)*, volume 32. Curran Associates, Inc.

[Lu et al., 2018] Lu, Y., Zhong, A., Li, Q., and Dong, B. (2018). Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations. In *Proceedings of the International Conference on Machine Learning (ICML 2018)*.

[Ma et al., 2020] Ma, W.-D. K., Lewis, J., and Kleijn, W. B. (2020). The hsic bottleneck: Deep learning without back-propagation. In *The AAAI Conference on Artificial Intelligence (AAAI-20)*.

[Ma et al., 2018] Ma, X., Li, B., Wang, Y., Erfani, S. M., Wijewickrema, S., Schoenebeck, G., Song, D., Houle, M. E., and Bailey, J. (2018). Characterizing adversarial subspaces using local intrinsic dimensionality. In *International Conference on Learning Representations (ICLR 2018)*.

[Madry et al., 2018] Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. (2018). Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations (ICLR 2018)*.

[Mahajan et al., 2018] Mahajan, D., Girshick, R., Ramanathan, V., He, K., Paluri, M., Li, Y., Bharambe, A., and van der Maaten, L. (2018). Exploring the limits of weakly supervised pretraining. In *European Conference on Computer Vision (ECCV 2018)*.

[Malach and Shalev-Shwartz, 2018] Malach, E. and Shalev-Shwartz, S. (2018). A provably correct algorithm for deep learning that actually works. *arXiv*.

[Mandar Kulkarni, 2016] Mandar Kulkarni, S. K. (2016). Layer-wise training of deep networks using kernel similarity. In *DLPR workshop at the 25th International Conference on Pattern Recognition (ICPR 2016)*.

[Marquez et al., 2018] Marquez, E. S., Hare, J. S., and Niranjan, M. (2018). Deep cascade learning. *IEEE Transactions on Neural Networks and Learning Systems*.

[Masoumian et al., 2022] Masoumian, A., Rashwan, H. A., Cristiano, J., Asif, M. S., and Puig, D. (2022). Monocular depth estimation using deep learning: A review. *Sensors*, 22(14).

[McCann and Pass, 2019] McCann, R. J. and Pass, B. (2019). Optimal transportation between unequal dimensions. *arXiv*.

[Mehrish et al., 2023] Mehrish, A., Majumder, N., Bhardwaj, R., Mihalcea, R., and Poria, S. (2023). A review of deep learning techniques for speech processing. *arXiv*.

[Meng and Chen, 2017] Meng, D. and Chen, H. (2017). Magnet: A two-pronged defense against adversarial examples. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 135–147, New York, NY, USA. Association for Computing Machinery.

[Metzen et al., 2017] Metzen, J. H., Genewein, T., Fischer, V., and Bischoff, B. (2017). On detecting adversarial perturbations. In *ICLR*.

[Minaee et al., 2022] Minaee, S., Boykov, Y., Porikli, F., Plaza, A., Kehtarnavaz, N., and Terzopoulos, D. (2022). Image segmentation using deep learning: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(7):3523–3542.

[Mittal et al., 2012] Mittal, A., Moorthy, A. K., and Bovik, A. C. (2012). No-reference image quality assessment in the spatial domain. *IEEE Transactions on Image Processing*, 21(12):4695–4708.

[Moosavi-Dezfooli et al., 2017a] Moosavi-Dezfooli, S., Fawzi, A., Fawzi, O., and Frossard, P. (2017a). Universal adversarial perturbations. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 86–94, Los Alamitos, CA, USA. IEEE Computer Society.

[Moosavi-Dezfooli et al., 2017b] Moosavi-Dezfooli, S.-M., Fawzi, A., Fawzi, O., and Frossard, P. (2017b). Universal adversarial perturbations. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 86–94.

[Moosavi-Dezfooli et al., 2016] Moosavi-Dezfooli, S.-M., Fawzi, A., and Frossard, P. (2016). Deepfool: A simple and accurate method to fool deep neural networks. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2574–2582.

[Morcos et al., 2018] Morcos, A., Raghu, M., and Bengio, S. (2018). Insights on representational similarity in neural networks with canonical correlation. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N.,

and Garnett, R., editors, *Advances in Neural Information Processing Systems (NeurIPS 2018)*, volume 31. Curran Associates, Inc.

[Mostafa et al., 2018] Mostafa, H., Ramesh, V., and Cauwenberghs, G. (2018). Deep supervised learning using local errors. *Frontiers in Neuroscience*.

[Nakkiran et al., 2020] Nakkiran, P., Kaplun, G., Bansal, Y., Yang, T., Barak, B., and Sutskever, I. (2020). Deep double descent: Where bigger models and more data hurt. In *International Conference on Learning Representations (ICLR 2020)*.

[Netzer et al., 2011] Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., and Ng, A. Y. (2011). Reading digits in natural images with unsupervised feature learning. In *Workshop on Deep Learning and Unsupervised Feature Learning at Advances in Neural Information Processing Systems (NeurIPS 2011)*.

[Nguyen et al., 2015] Nguyen, A., Yosinski, J., and Clune, J. (2015). Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 427–436.

[Nguyen and Choi, 2019] Nguyen, T. T. and Choi, J. (2019). Layer-wise learning of stochastic neural networks with information bottleneck. *Entropy*, 21.

[Ni et al., 2022] Ni, Z., Wang, Y., Yu, J., Jiang, H., Cao, Y., and Huang, G. (2022). Deep model assembling. *arXiv*.

[Nicolae et al., 2018] Nicolae, M.-I., Sinn, M., Tran, M. N., Buesser, B., Rawat, A., Wistuba, M., Zantedeschi, V., Baracaldo, N., Chen, B., Ludwig, H., Molloy, I. M., and Edwards, B. (2018). Adversarial robustness toolbox v1.0.0. *arXiv*.

[Novak et al., 2018] Novak, R., Bahri, Y., Abolafia, D. A., Pennington, J., and Sohl-Dickstein, J. (2018). Sensitivity and generalization in neural networks: an empirical study. In *International Conference on Learning Representations (ICLR 2018)*.

[Nøkland and Eidnes, 2019] Nøkland, A. and Eidnes, L. H. (2019). Training neural networks with local error signals. In *Proceedings of the International Conference on Machine Learning (ICML 2019)*.

[Pang et al., 2018] Pang, T., Du, C., Dong, Y., and Zhu, J. (2018). Towards robust detection of adversarial examples. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems (NeurIPS 2018)*, volume 31. Curran Associates, Inc.

[Papernot et al., 2016a] Papernot, N., McDaniel, P., and Goodfellow, I. (2016a). Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. *arXiv*.

[Papernot et al., 2017] Papernot, N., McDaniel, P., Goodfellow, I., Jha, S., Celik, Z. B., and Swami, A. (2017). Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '17, page 506–519, New York, NY, USA. Association for Computing Machinery.

[Papernot et al., 2016b] Papernot, N., McDaniel, P., Jha, S., Fredrikson, M., Celik, Z. B., and Swami, A. (2016b). The limitations of deep learning in adversarial settings. In *2016 IEEE European Symposium on Security and Privacy (EuroSP)*, pages 372–387.

[Papernot et al., 2016c] Papernot, N., McDaniel, P., Wu, X., Jha, S., and Swami, A. (2016c). Distillation as a defense to adversarial perturbations against deep neural networks. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 582–597.

[Parzen, 1962] Parzen, E. (1962). On Estimation of a Probability Density Function and Mode. *The Annals of Mathematical Statistics*, 33(3):1065 – 1076.

[Pass, 2010] Pass, B. (2010). Regularity of optimal transportation between spaces with different dimensions. *arXiv*.

[Patel et al., 2022] Patel, A., Eickenberg, M., and Belilovsky, E. (2022). Local learning with neuron groups. *From Cells to Societies: Collective Learning Across Scales International Conference on Learning Representations (ICLR 2022)*.

[Pathak et al., 2022] Pathak, P., Zhang, J., and Samaras, D. (2022). Local learning on transformers via feature reconstruction. *arXiv*.

[Pedregosa et al., 2011] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research (JMLR 12)*, 12:2825–2830.

[Peyre and Cuturi, 2019] Peyre, G. and Cuturi, M. (2019). *Computational Optimal Transport*. Now Publishers.

[Pohlen et al., 2017] Pohlen, T., Hermans, A., Mathias, M., and Leibe, B. (2017). Full-resolution residual networks for semantic segmentation in street scenes. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR 2017)*.

[Pontryagin, 1987] Pontryagin, L. (1987). *Mathematical Theory of Optimal Processes*. Classics of Soviet Mathematics. Taylor & Francis.

[Purwins et al., 2019] Purwins, H., Li, B., Virtanen, T., Schlüter, J., Chang, S.-Y., and Sainath, T. (2019). Deep learning for audio signal processing. *IEEE Journal of Selected Topics in Signal Processing*, 13(2):206–219.

[Pyeon et al., 2021] Pyeon, M., Moon, J., Hahn, T., and Kim, G. (2021). {SEDONA}: Search for decoupled neural networks toward greedy block-wise learning. In *International Conference on Learning Representations (ICLR 2021)*.

[Quarteroni et al., 2007] Quarteroni, A., Sacco, R., and Saleri, F. (2007). *Numerical Mathematics*. Springer Berlin, Heidelberg.

[Radford et al., 2018] Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. (2018). Improving language understanding with unsupervised learning. *openai.com/research/language-unsupervised*.

[Raghu et al., 2017] Raghu, M., Gilmer, J., Yosinski, J., and Sohl-Dickstein, J. (2017). Svcca: Singular vector canonical correlation analysis for deep learning dynamics and interpretability. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems (NeurIPS 2017)*, volume 30. Curran Associates, Inc.

[Raghunathan et al., 2020] Raghunathan, A., Xie, S. M., Yang, F., Duchi, J., and Liang, P. (2020). Understanding and mitigating the tradeoff between robustness and accuracy. In III, H. D. and Singh, A., editors, *Proceedings of the 37th International Conference on Machine Learning (ICML 2020)*, volume 119 of *Proceedings of Machine Learning Research*, pages 7909–7919. PMLR.

[Rahaman et al., 2019] Rahaman, N., Baratin, A., Arpit, D., Draxler, F., Lin, M., Hamprecht, F., Bengio, Y., and Courville, A. (2019). On the spectral bias of neural networks. In *Proceedings of the International Conference on Machine Learning (ICML 2019)*.

[Ren et al., 2021] Ren, J., Fort, S., Liu, J., Roy, A. G., Padhy, S., and Lakshminarayanan, B. (2021). A simple fix to mahalanobis distance for improving near-ood detection. *arXiv*.

[Ren et al., 2023] Ren, M., Kornblith, S., Liao, R., and Hinton, G. (2023). Scaling forward gradient with local losses. In *The Eleventh International Conference on Learning Representations (ICLR 2023)*.

[Rosenblatt, 1956] Rosenblatt, M. (1956). Remarks on Some Nonparametric Estimates of a Density Function. *The Annals of Mathematical Statistics*, 27(3):832 – 837.

[Ruiz-Balet and Zuazua, 2021] Ruiz-Balet, D. and Zuazua, E. (2021). Neural ode control for classification, approximation and transport. *arXiv*.

[Rumelhart et al., 1986] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323.

[Russell and Norvig, 2020] Russell, S. and Norvig, P. (2020). *Artificial Intelligence: A Modern Approach, 4th edition*. Pearson Series in Artificial Intelligence., Pearson.

[Ruthotto and Haber, 2018] Ruthotto, L. and Haber, E. (2018). Deep neural networks motivated by partial differential equations. *arXiv*.

[Sabour et al., 2016] Sabour, S., Cao, Y., Faghri, F., and Fleet, D. J. (2016). Adversarial manipulation of deep representations. In *International Conference on Learning Representations (ICLR 2016)*.

[Samangouei et al., 2018] Samangouei, P., Kabkab, M., and Chellappa, R. (2018). Defense-GAN: Protecting classifiers against adversarial attacks using generative models. In *International Conference on Learning Representations (ICLR 2018)*.

[Sandler et al., 2019] Sandler, M., Baccash, J., Zhmoginov, A., and Howard, A. (2019). Non-discriminative data or weak model? on the relative importance of data and model resolution. In *Workshop on Real-World Recognition from Low-Quality Images and Videos at International Conference on Computer Vision (ICCV 2019)*.

[Sandler et al., 2018] Sandler, M., Howard, A. G., Zhu, M., Zhmoginov, A., and Chen, L. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 4510–4520. Computer Vision Foundation / IEEE Computer Society.

[Santambrogio, 2015] Santambrogio, F. (2015). *Optimal Transport for Applied Mathematicians*. Birkhäuser.

[Santambrogio, 2016] Santambrogio, F. (2016). Euclidean, metric, and wasserstein gradient flows: an overview. *arXiv*.

[Saxe et al., 2014] Saxe, A. M., Mcclelland, J. L., and Ganguli, S. (2014). Exact solutions to the nonlinear dynamics of learning in deep linear neural network. In *International Conference on Learning Representations (ICLR 2014)*.

[Schmidt et al., 2018] Schmidt, L., Santurkar, S., Tsipras, D., Talwar, K., and Madry, A. (2018). Adversarially robust generalization requires more data. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems (NeurIPS 2018)*, volume 31. Curran Associates, Inc.

[Shannon, 1948] Shannon, C. E. (1948). A mathematical theory of communication. *The Bell System Technical Journal*, 27(4):623–656.

[Sharma et al., 2021] Sharma, V., Gupta, M., Kumar, A., and Mishra, D. (2021). Video processing using deep learning techniques: A systematic literature review. *IEEE Access*, 9:139489–139507.

[Shashkin, 1991] Shashkin, I. (1991). *Fixed Points*. Mathematical world. American Mathematical Soc.

[Silver et al., 2016] 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., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of go with deep neural networks and tree search. nature. *Nature*.

[Sim et al., 2020] Sim, B., Oh, G., Kim, J., Jung, C., and Ye, J. C. (2020). Optimal transport driven cyclegan for unsupervised learning in inverse problems. *SIAM Journal on Imaging Sciences*, 13(4):2281–2306.

[Simonyan and Zisserman, 2014] Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations (ICLR 2014)*.

[Singh and Jaggi, 2020] Singh, S. P. and Jaggi, M. (2020). Model fusion via optimal transport. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems (NeurIPS 2020)*, volume 33, pages 22045–22055. Curran Associates, Inc.

[Song et al., 2018] Song, Y., Kim, T., Nowozin, S., Ermon, S., and Kushman, N. (2018). Pixeldefend: Leveraging generative models to understand and defend against adversarial examples. In *International Conference on Learning Representations (ICLR 2018)*.

[Sonoda and Murata, 2019] Sonoda, S. and Murata, N. (2019). Transport analysis of infinitely deep neural network. *Journal of Machine Learning Research (JMLR 2019)*.

[Srivastava et al., 2014] 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 (JMLR 15)*, 15(56):1929–1958.

[Stich, 2019] Stich, S. U. (2019). Local sgd converges fast and communicates little. In *International Conference on Learning Representations (ICLR 2019)*.

[Su et al., 2019] Su, J., Vargas, D. V., and Sakurai, K. (2019). One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation*, 23(5):828–841.

[Sun et al., 2021] Sun, Q., Dong, H., Chen, Z., Sun, J., Li, Z., and Dong, B. (2021). Layer-parallel training of residual networks with auxiliary-variable networks. *arXiv*.

[Szegedy et al., 2017] Szegedy, C., Ioffe, S., Vanhoucke, V., and Alemi, A. A. (2017). Inception-v4, inception-resnet and the impact of residual connections on learning. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, page 4278–4284. AAAI Press.

[Szegedy et al., 2013] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. (2013). Intriguing properties of neural networks. *arXiv*.

[Söderlind, 2006] Söderlind, G. (2006). Time-step selection algorithms: Adaptivity, control, and signal processing. *Applied Numerical Mathematics*, 56(3):488–502. Selected Papers, The Third International Conference on the Numerical Solutions of Volterra and Delay Equations.

[Tan and Le, 2019] Tan, M. and Le, Q. (2019). EfficientNet: Rethinking model scaling for convolutional neural networks. In Chaudhuri, K. and Salakhutdinov, R., editors, *Proceedings of the 36th International Conference on Machine Learning (ICML 2019)*, volume 97 of *Proceedings of Machine Learning Research*, pages 6105–6114. PMLR.

[Tanay and Griffin, 2016] Tanay, T. and Griffin, L. (2016). A boundary tilting perspective on the phenomenon of adversarial examples. *arXiv*.

[Tang et al., 2021] Tang, Y., Teng, Q., Zhang, L., Min, F., and He, J. (2021). Layer-wise training convolutional neural networks with smaller filters for human activity recognition using wearable sensors. *IEEE Sensors Journal*.

[Tempczyk et al., 2022] Tempczyk, P., Michaluk, R., Garncarek, L., Spurek, P., Tabor, J., and Golinski, A. (2022). LIDL: Local intrinsic dimension estimation using approximate likelihood. In Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G., and Sabato, S., editors, *Proceedings of the 39th International Conference on Machine Learning (ICML 2022)*, volume 162 of *Proceedings of Machine Learning Research*, pages 21205–21231. PMLR.

[Teng et al., 2020] Teng, Q., Wang, K., Zhang, L., and He, J. (2020). The layer-wise training convolutional neural networks using local loss for sensor-based human activity recognition. *IEEE Sensors Journal*.

[Tran et al., 2018] Tran, D., Wang, H., Torresani, L., Ray, J., LeCun, Y., and Paluri, M. (2018). A closer look at spatiotemporal convolutions for action recognition. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR 2018)*.

[van den Oord et al., 2016] van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. (2016). Wavenet: A generative model for raw audio. *arXiv*.

[Veit et al., 2016] Veit, A., Wilber, M. J., and Belongie, S. (2016). Residual networks behave like ensembles of relatively shallow networks. In Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems (NeurIPS 2016)*, volume 29. Curran Associates, Inc.

[Villani, 2008] Villani, C. (2008). *Optimal Transport: Old and New*. Springer-Verlag.

[Villani, 2021] Villani, C. (2021). *Topics in Optimal Transportation*. Graduate Studies in Mathematics. American Mathematical Society.

[Virmaux and Scaman, 2018] Virmaux, A. and Scaman, K. (2018). Lipschitz regularity of deep neural networks: analysis and efficient estimation. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems (NeurIPS 2018)*, volume 31. Curran Associates, Inc.

[Wang et al., 2019a] Wang, B., Shi, Z., and Osher, S. (2019a). Resnets ensemble via the feynman-kac formalism to improve natural and robust accuracies. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems (NeurIPS 2019)*, volume 32. Curran Associates, Inc.

[Wang et al., 2019b] Wang, B., Shi, Z., and Osher, S. (2019b). Resnets ensemble via the feynman-kac formalism to improve natural and robust accuracies. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems (NeurIPS 2019)*, volume 32. Curran Associates, Inc.

[Wang et al., 2018] Wang, L., Hu, L., Gu, J., Hu, Z., Wu, Y., He, K., and Hopcroft, J. (2018). Towards understanding learning representations: To what extent do different neural networks learn the same representation. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems (NeurIPS 2018)*, volume 31. Curran Associates, Inc.

[Wang et al., 2021] Wang, Y., Ni, Z., Song, S., and Le Yang, G. H. (2021). Revisiting locally supervised learning: an alternative to end-to-end training. In *International Conference on Learning Representations (ICLR 2021)*.

[Weinan, 2017] Weinan, E. (2017). A proposal on machine learning via dynamical systems. *Commun. Math. Stat.*

[Weng et al., 2018] Weng, T.-W., Zhang, H., Chen, P.-Y., Yi, J., Su, D., Gao, Y., Hsieh, C.-J., and Daniel, L. (2018). Evaluating the robustness of neural networks: An extreme value theory approach. In *International Conference on Learning Representations (ICLR 2018)*.

[Wightman et al., 2021] Wightman, R., Touvron, H., and Jégou, H. (2021). Resnet strikes back: An improved training procedure in timm. *arXiv*.

[Wu et al., 2020] Wu, D., Wang, Y., Xia, S.-T., Bailey, J., and Ma, X. (2020). Skip connections matter: On the transferability of adversarial examples generated with resnets. In *International Conference on Learning Representations (ICLR 2020)*.

[Wu et al., 2016] Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J.,

Shah, A., Johnson, M., Liu, X., Łukasz Kaiser, Gouws, S., Kato, Y., Kudo, T., Kazawa, H., Stevens, K., Kurian, G., Patil, N., Wang, W., Young, C., Smith, J., Riesa, J., Rudnick, A., Vinyals, O., Corrado, G., Hughe, M., and Dean, J. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv*.

[Xie et al., 2017] Xie, S. et al. (2017). Aggregated residual transformations for deep neural networks. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR 2017)*.

[Xiong et al., 2017] Xiong, W., Droppo, J., Huang, X., Seide, F., Seltzer, M., Stolcke, A., Yu, D., and Zweig, G. (2017). The microsoft 2016 conversational speech recognition system. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2017)*.

[Xiong et al., 2020] Xiong, Y., Ren, M., and Urtasun, R. (2020). Loco: Local contrastive representation learning. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H., editors, *Advances in Neural Information Processing Systems (NeurIPS 2020)*, volume 33, pages 11142–11153. Curran Associates, Inc.

[Xu and Principe, 1999] Xu, D. and Principe, J. (1999). Training mlps layer-by-layer with the information potential. In *International Joint Conference on Neural Networks (IJCNN'99)*, volume 3, pages 1716–1720 vol.3.

[Yan et al., 2020] Yan, H., Du, J., Tan, V., and Feng, J. (2020). On robustness of neural ordinary differential equations. *International Conference on Learning Representations (ICLR 2020*.

[Yang et al., 2020a] Yang, Y., Wu, J., Li, H., Li, X., Shen, T., and Lin, Z. (2020a). Dynamical system inspired adaptive time stepping controller for residual network families. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 6648–6655.

[Yang et al., 2020b] Yang, Y.-Y., Rashtchian, C., Zhang, H., Salakhutdinov, R. R., and Chaudhuri, K. (2020b). A closer look at accuracy vs. robustness. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems (NeurIPS 2020)*, volume 33, pages 8588–8601. Curran Associates, Inc.

[Ypma, 1995] Ypma, T. J. (1995). Historical development of the newton–raphson method. *SIAM Review*, 37(4):531–551.

[Zagoruyko and Komodakis, 2016] Zagoruyko, S. and Komodakis, N. (2016). Wide residual networks. In *Proceedings of the British Machine Vision Conference (BMVC 2016)*.

[Zhang et al., 2017a] Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. (2017a). Understanding deep learning requires rethinking generalization. In *International Conference on Learning Representations (ICLR 2017)*.

[Zhang et al., 2020] Zhang, C., Zhang, H., and Hsieh, C.-J. (2020). An efficient adversarial attack for tree ensembles. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems (NeurIPS 2020)*, volume 33, pages 16165–16176. Curran Associates, Inc.

[Zhang et al., 2019] Zhang, J. et al. (2019). Towards robust resnet: A small step but a giant leap. In *Proceedings of the Twenty-Eight International Joint Conference on Artificial Intelligence (IJCAI-19)*.

[Zhang and Wynter, 2018] Zhang, J. and Wynter, L. (2018). Smooth inter-layer propagation of stabilized neural networks for classification. *arXiv*.

[Zhang et al., 2022] Zhang, J., Zhang, X., Ma, K., Gupta, R., Saltz, J., Vakalopoulou, M., and Samaras, D. (2022). Gigapixel whole-slide images classification using locally supervised learning. In *Medical Image Computing and Computer Assisted Intervention – MICCAI 2022*, pages 192–201, Cham. Springer Nature Switzerland.

[Zhang et al., 2017b] Zhang, X., Li, Z., Loy, C., and Lin, D. (2017b). Polynet: A pursuit of structural diversity in very deep networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2017)*, pages 3900–3908, Los Alamitos, CA, USA. IEEE Computer Society.

[Zhu et al., 2017] Zhu, J.-Y., Park, T., Isola, P., and Efros, A. A. (2017). Unpaired image-to-image translation using cycle-consistent adversarial networks. In *International Conference on Computer Vision (ICCV 2017)*.

[Zhu et al., 2022] Zhu, M., Chang, B., and Fu, C. (2022). Convolutional neural networks combined with runge–kutta methods. *Neural Computing and Applications*.

[Zhuang et al., 2021] Zhuang, H., Weng, Z., Luo, F., Kar-Ann, T., Li, H., and Lin, Z. (2021). Accumulated decoupled learning with gradient staleness mitigation for convolutional neural networks. In Meila, M. and Zhang, T., editors, *Proceedings of the 38th International Conference on Machine Learning (ICML 2021)*, volume 139 of *Proceedings of Machine Learning Research*, pages 12935–12944. PMLR.

List of Acronyms

AA AutoAttack.

ADL Accumulated Decoupled Learning.

APGD Auto Projected Gradient Descent.

ATN Adversarial Transformation Networks.

AUROC Area Under Receiver Operating Curve.

BA Boundary Attack.

BFGS Broyden–Fletcher–Goldfarb–Shanno.

BIM Basic Iterative Method.

BN Batch-Normalization.

BS Bogacki-Shampine.

CKA Centered Kernel Alignment.

Conv Convolution.

CW Carlini-Wagner.

DDG Decoupled Delayed Gradients.

DF DeepFool.

DFT Discrete Fourier Transform.

DGL Decoupled Greedy Learning.

DNI Decoupled Neural Interface.

FA Feature Adversary.

FAB Fast Adaptive Boundary.

FGM Fast Gradient Method.

FPR False Positive Rate.

FR Features Replay.

GGD Generalized Gaussian Distribution.

HSJ HopSkipJump.

IMEX Implicit-Explicit.

InfoPro Information Propagation.

JSMA Jacobian Saliency Map Attack.

KD Kernel Density.

LAP Least Action Principle.

LID Local Intrinsic Dimensionality.

LM Linear Multi-step.

LN Layer Normalization.

MFS Magnitude Fourier Spectrum.

MH Mahalanobis.

MLE Maximum Likelihood Estimator.

MLP Multi-Layer Perception.

MLS Multi-Lap Sequential.

MMD Maximum Mean Discrepancy.

MSA Method of Successive Approximations.

MSCN Mean Subtracted Contrast Normalized.

NODE Neural Ordinary Differential Equation.

NR Newton-Raphson.

NSS Natural Scene Statistics.

ODE Ordinary Differential Equation.

OOD Out-Of-Distribution.

OT Optimal Transport.

PDE Partial Differential Equation.

PFS Phase Fourier Spectrum.

PGD Projected Gradient Descent.

PMP Pontryagin's Maximum Principle.

PredSim Prediction Similarity.

RCE Reverse Cross Entropy.

ReLU Rectified Linear Unit.

ResBlock Residual Block.

ResNet Residual Network.

RK Runge-Kutta.

RMD Relative Mahalanobis Distance.

ROC Receiver Operating Characteristic.

SA Square Attack.

Sedona Searching for Decoupled Neural Architectures.

SGD Stochastic Gradient Descent.

ST Spatial Transformation.

SVCCA Singular Vector Canonical Correlation Analysis.

SW-MSA Shifted Window Multi-head Self-Attention.

TPR True Positive Rate.

TR Transport.

TRGL Transport-Regularized Greedy Learning.

UAP Universal Adversarial Perturbation.

VanGL Vanilla Greedy Learning.

W-MSA Window Multi-head Self-Attention.

ZOO Zeroth Order Optimization.

List of Tables

1 Accuracy of LAP-ResNet-9 on MNIST 31
2 Accuracy of LAP-ResNet-9 on CIFAR10 31
3 Accuracy of LAP-ResNeXt50 on CIFAR100 32
4 Accuracy of parallel TRGL with 4 modules on TinyImageNet . . 50
5 Accuracy of parallel TRGL with 2 modules on CIFAR100 50
6 Accuracy of parallel TRGL with K modules on STL10 51
7 Accuracy of parallel TRGL with 4 modules using a Swin Transformer 51
8 Accuracy of sequential and MLS TRGL on TinyImageNet 51
9 Memory savings of module-wise training on STL10 52
10 Accuracy of 10-1 ResNet with sequential, MLS and parallel TRGL 54
11 Last block accuracy of 10-1 ResNet with sequential TRGL 54
12 Detection accuracy of seen attacks 72
13 Detection accuracy of unseen attacks 73
14 OOD detection accuracy on ResNet110 74
15 OOD detection accuracy on ResNeXt50 74
16 Accuracy of LAP-ResNeXt50 on CIFAR100 159
17 Accuracy of LAP-ResNet-9 on CIFAR100 159
18 Accuracy of LAP-ResNet-9 on CIFAR10 159
19 Accuracy on the circles dataset with 1 block 162
20 Accuracy on the circles dataset with 100 blocks 162
21 Accuracy on the circles dataset with 50 training points 163
22 Accuracy on the circles dataset with a large initialization 163
23 Accuracy of 2-7 ResNet with parallel TRGL 169
24 Accuracy of parallel TRGL with 2 modules 170
25 Last block accuracy of 20-1 ResNet with parallel TRGL 170
26 Last block accuracy of 20-1 ResNet with parallel TRGL 170
27 Accuracy of 10-1 ResNet with sequential TRGL 171
28 Last block accuracy of 50-1 ResNet with sequential and MLS TRGL 171
29 Accuracy of ResNeXt50 with sequential, MLS and parallel TRGL 171
30 Accuracy of parallel TRGL* with K modules on STL10 172
31 Memory savings of module-wise training with 4 modules 172
32 Detection accuracy of seen attacks on ResNet110 175
33 Detection accuracy of seen attacks on ResNeXt50 176
34 Detection accuracy of seen attacks on WideResNet 176
35 Detection accuracy of unseen attacks on ResNet110 176
36 Detection accuracy of unseen attacks on ResNeXt50 177
37 Detection accuracy of unseen attacks on WideResNet 177
38 Detection rate of successful samples from seen attacks 177
39 Detection rate of successful samples from unseen attacks 178
40 False positive rate on seen attacks 178
41 False positive rate on unseen attacks 178
42 AUROC of seen attacks 179
43 AUROC of unseen attacks 179

44 AUROC of OOD detection on ResNet110 180
45 FPR at 95% TPR of OOD detection on ResNet110 180
46 AUROC of OOD detection on ResNeXt50 180
47 FPR at 95% TPR of OOD detection on ResNeXt50 181

List of Figures

1 Basic residual block architecture 6
2 Bottleneck residual block and ResNeXt block architectures . . . 7
3 Bottleneck residual block and inverted residual block architectures 7
4 Consecutive basic residual blocks 8
5 Swin Transformer architecture with 4 residual stages 8
6 Ratio of residue norm to input norm and cosine loss of ResNets . 10
7 Architectures inspired by numerical schemes 14
8 The Kantorovitch problem from μ to ν 16
9 The Monge problem from μ to ν 16
10 Dynamic transport of μ_0 to μ_1 according to velocity field v . . . 17
11 Test transport cost against test accuracy of ResNet-9 models . . 23
12 Circles test set after blocks 1, 5 and 9 24
13 Decomposition of a neural network 25
14 Accuracy of ResNets without batch-normalization 33
15 Accuracy during training of ResNeXt50 models 34
16 Module-wise training 38
17 Early overfitting of vanilla module-wise training 39
18 Accuracy of each module after module-wise training 55
19 Example of an adversarial attack 59
20 Boundary tilting perspective of adversarial samples 61
21 Transport cost histogram for clean and attacked test samples . . 69
22 Circles test set and OOD points after blocks 6 and 9 69
23 Adversarial detection dataset creation 70
24 The Monge problem from μ to ν 125
25 The Kantorovitch problem from μ to ν 126
26 Dynamic transport of μ_0 to μ_1 according to velocity field v . . . 136
27 Early overfitting of vanilla module-wise training 145
28 Decoding the internal representations of a network. 155
29 Internal representations of ResNet-9 on MNIST 156
30 Internal representations of LAP-ResNet-9 on MNIST 156
31 Internal representations of ResNet-9 on CIFAR10 157
32 Internal representations of LAP-ResNet-9 on CIFAR10 157
33 Internal representations of ResNet-9 on CIFAR100 158
34 Internal representations of LAP-ResNet-9 on CIFAR100 158
35 Weight distribution of the classifier of a vanilla ResNet-9 160
36 Weight distribution of the classifier of a LAP-ResNet-9 160
37 Distances between successive point clouds in a ResNet-9 161
38 Circles test set embeddings with LAP 164
39 Circles test set embeddings with large initialization 165
40 Circles test set embeddings with large initialization and BN . . . 166
41 Circles test set embeddings with large initialization, BN and LAP 167
42 Accuracy of each block after block-wise training 171
43 Sensitivity of TRGL to hyperparameter τ 173

A Background on Numerical Methods for Differential Equations

We refer to [Quarteroni et al., 2007] for this quick background on numerical methods for solving ordinary differential equations.

Given an interval $I = [t_0, t_0 + T]$ and a function $f : I \times \mathbb{R}^d \to \mathbb{R}^d$, we consider the *initial value problem* of finding a function $x : I \to \mathbb{R}^d$ that satisfies

$$x'(t) = f(t, x(t))$$

with initial condition $x(t_0) = x_0$, or equivalently in integral form, a function x that satisfies

$$x(t) = x_0 + \int_{t_0}^{t} f(t, x(t))\,\mathrm{d}t$$

It is often difficult to find closed-form solutions to initial value problems. We wish then to approximate the values x takes at certain time points. We consider a subdivision $t_0 < t_1 < .. < t_N = t_0 + T$ of I. Denote the *time-steps* (or *step-sizes*) $h_n := t_{n+1} - t_n$ for $0 \leq n < N$ and define $h_{\max} := \max h_n$.

A.1 The Euler Method

In a one-step method, an approximation of $x(t_n)$ is x_n given iteratively starting from a known initial condition x_0 by

$$\frac{x_{n+1} - x_n}{h_n} = \phi(t_n, x_n, t_{n+1}, x_{n+1}, h_n)$$

If $f(x_{n+1})$ does not appear in the definition of ϕ, the method is called explicit. Otherwise, it is called implicit, as x_{n+1} depends then implicitly on itself through f and an equation needs to be solved to find it, for example, using a fixed-point [Shashkin, 1991] or a Newton-Raphson [Ypma, 1995] method. So for $\phi(t_n, x_n, t_{n+1}, x_{n+1}, h_n) = f(t_n, x_n)$, we get the explicit Euler method:

$$x_{n+1} = x_n + h_n f(t_n, x_n)$$

And for $\phi(t_n, x_n, t_{n+1}, x_{n+1}, h_n) = f(t_{n+1}, x_{n+1})$, we get the implicit Euler method

$$x_{n+1} = x_n + h_n f(t_{n+1}, x_{n+1})$$

We only need the convergence analysis of explicit methods, so in the following we will limit ourselves to methods given by

$$\frac{x_{n+1} - x_n}{h_n} = \phi(t_n, x_n, h_n)$$

Definition 4. For a one-step method, the *consistency errors* e_n, for $0 \leq n < N$, are

$$e_n := \frac{x(t_{n+1}) - \Phi(t_n, x(t_n), h_n)}{h_n} = \frac{x(t_{n+1}) - x(t_n)}{h_n} - \phi(t_n, x(t_n), h_n)$$

where x is solution. The *local (truncation) errors* are $h_n e_n$. The method is *consistent* if $\max|e_n|$ goes to zero as $h_{\max}$ goes to zero. For $p \in \mathbb{N}^*$, the method has *order* p if $\max|e_n| \leq C h_{\max}^p$ for a constant C that depends on f, t_0 and T.

Theorem 9. If f and ϕ are continuous, then the one-step method is consistent if and only if $\phi(t, x, 0) = f(t, x)$ for all (t, x).

Theorem 10. If f is $\mathcal{C}^p$ and ϕ is $\mathcal{C}^p$ in h then the one-step method is of order p if and only if $\partial_h^k \phi(t, x, 0) = \frac{1}{k+1} f^{[k]}(t, x)$ for all (t, x) and $0 \leq k < p$ where $f^{[0]} = f$ and $f^{[k]} = \partial_t f^{[k-1]} + f \partial_x f^{[k-1]}$.

Corollary 11. If f is continuous then Euler's method is consistent. If f is $\mathcal{C}^1$ then Euler's method has order 1.

Definition 5. A one-step method is *zero-stable* (or *stable*) if $\exists\ S > 0$ such that for all $(x_n)_{0 \leq n \leq N}$, $(\tilde{x}_n)_{0 \leq n \leq N}$ and $(\varepsilon_n)_{0 \leq n < N}$ satisfying

$$\frac{x_{n+1} - x_n}{h_n} = \phi(t_n, x_n, h_n) \text{ and } \frac{\tilde{x}_{n+1} - \tilde{x}_n}{h_n} = \phi(t_n, \tilde{x}_n, h_n) + \epsilon_n \qquad (44)$$

for $0 \leq n < N$, we have $\max_n \|\tilde{x}_n - x_n\| \leq S(\|\tilde{x}_0 - x_0\| + T \max_n |\epsilon_n|)$, where $\epsilon_n = \varepsilon_n / h_n$. The constant S is the *stability constant* of the method.

Theorem 12. If ϕ is uniformly L-Lipschitz in its second variable, then the one-step method is stable with constant e^{LT}.

Corollary 13. If f is Lipschitz in its second variable, then Euler's method is stable.

Definition 6. A numerical method *converges* if its *global error* $\max_n \|x(t_n) - x_n\|$ goes to zero as $h_{\max}$ goes to zero.

Theorem 14. If a method is consistent and stable with stability constant S, then it converges and $\max_n \|x(t_n) - x_n\| \leq ST \max|e_n|$. If the method is of order p with constant C, then $\max_n \|x(t_n) - x_n\| \leq STC h_{\max}^p$

Corollary 15. Euler's method converges if f is $\mathcal{C}^0$ and Lipschitz in x. If f is also $\mathcal{C}^1$ then it converges with speed $O(h_{\max})$.

A.2 Runge-Kutta Methods

Definition 7. Given $q \in \mathbb{N}^*$, a *Runge-Kutta* method with q *stages* generates the estimates $(x_n)_{0 \leq n \leq N}$ through, for $0 \leq n < N$,

$$t_{n,i} = t_n + c_i h_n, \quad x_{n,i} = x_n + h_n \sum_{j=1}^{i-1} a_{i,j} p_{n,j}, \quad p_{n,i} = f(t_{n,i}, x_{n,i}), \quad 0 \leq i \leq q$$

$$t_{n+1} = t_n + h_n, \quad x_{n+1} = x_n + h_n \sum_{i=1}^{q} b_i p_{n,i}$$

where $c_i, b_i, a_{i,j} \in [0,1]$ for all $0 \leq i \leq q$ and $1 \leq j < i$ with

$$c_1 = 0 \text{ so that } t_{n,1} = t_n,\ x_{n,1} = x_n \text{ and } p_{n,1} = f(t_n, x_n) \text{ for } 0 \leq n < N$$
$$\sum_{j=1}^{i-1} a_{i,j} = c_i \text{ for } 0 \leq i \leq q \text{ and } \sum_{i=1}^{q} b_i = 1$$

This is an explicit one-step method with ϕ given by

$$\begin{cases} \phi(t,x,h) = \sum_{i=1}^{q} b_i f(t + c_i h, x_i) \\ x_i = x + h \sum_{j=1}^{i-1} a_{i,j} f(t + c_j h, x_j) \text{ for } 1 \leq i \leq q \end{cases} \tag{45}$$

Remark 16. The conditions on $(a_{i,j})$ and (b_i) are so that the numerical integration methods used are of order 1, i.e. exact on constant functions, which is equivalent to the coefficients summing up to the length of the integration interval.

Remark 17. For $q = 1$ we find Euler's method.

Theorem 18. If f is continuous then Runge-Kutta methods are consistent.

Definition 8. The Runge-Kutta method with $q = 2$ stages, $c_2 = a_{2,1} = 1$ and $b_1 = b_2 = 1/2$ is called *Heun's method* and is given by $\phi(t,x,h) = \frac{1}{2} f(t,x) + \frac{1}{2} f(t + h, x + h f(t,x))$.

Proposition 4. If f is continuous then Heun's method is consistent. If f is $\mathcal{C}^2$ then Heun's method has order 2.

Definition 9. The Runge-Kutta method with $q = 4$ stages, $c_2 = c_3 = a_{2,1} = a_{3,2} = 1/2$, $c_4 = a_{4,3} = 1$, $b_1 = b_4 = 1/6$, $b_2 = b_3 = 1/3$ and all other coefficients equal to zero is the classic *RK4 method*. It is given by

$$\begin{cases} t_{n,1} = t_n & x_{n,1} = x_n & p_{n,1} = f(t_{n,1}, x_{n,1}) \\ t_{n,2} = t_n + \frac{h_n}{2} & x_{n,2} = x_n + \frac{h_n}{2} p_{n,1} & p_{n,2} = f(t_{n,2}, x_{n,2}) \\ t_{n,3} = t_n + \frac{h_n}{2} & x_{n,3} = x_n + \frac{h_n}{2} p_{n,2} & p_{n,3} = f(t_{n,3}, x_{n,3}) \\ t_{n,4} = t_n + h_n & x_{n,4} = x_n + h_n p_{n,3} & p_{n,4} = f(t_{n,4}, x_{n,4}) \end{cases}$$

And finally $t_{n+1} = t_n + h_n$ and $x_{n+1} = x_n + \frac{h_n}{6}(p_{n,1} + 2p_{n,2} + 2p_{n,3} + p_{n,4})$

Proposition 5. If f is continuous then the RK4 method is consistent. If f is $\mathcal{C}^4$ then the RK4 method has order 4.

Theorem 19. If f is continuous and L-Lipschitz in its second variable and the time-steps are bounded then a Runge-Kutta method with q stages is stable with stability constant $S = e^{KT}$ where

$$K = L \sum_{i=1}^{q} b_i \sum_{j=0}^{i-1} L^j h_{\max}^j$$

Corollary 20. Runge-Kutta methods converge if f is $\mathcal{C}^0$ and Lipschitz in x.

Corollary 21. Heun's method converges if f is $\mathcal{C}^0$ and Lipschitz in x. If f is also $\mathcal{C}^2$ then it converges with speed $O(h_{\max}^2)$.

Corollary 22. The RK4 method converges if f is $\mathcal{C}^0$ and Lipschitz in x. If f is also $\mathcal{C}^4$ then it converges with speed $O(h_{\max}^4)$.

A.3 Adaptive Runge-Kutta Methods

Adaptive methods [Söderlind, 2006] estimate the local truncation error of a single Runge–Kutta step and decrease the step size if the error estimate is large (which indicates that the differential equation is not approximated well at that point and more conservative steps need to be taken) and increase it if it is low. This is done by using two interwoven Runge-Kutta methods that have common intermediate stages, meaning that their coefficients $p_{n,i}$ in Definition 7 are the same but the weights b_i in front of them differ. The local error is estimated by the difference between the predictions of the two methods. This way, estimating the error has little or negligible computational cost compared to a step of the higher-order method which already requires the computation of the lower-order method. For example, Heun's method includes Euler's method.

During the integration, the step size is adapted such that the estimated error stays below a chosen threshold. If the estimated error is too high, a step is repeated with a lower step size. If the estimated error is much smaller than the threshold, the step size is increased to save time. This results in an (almost) optimal step size, which saves computation time.

For example, a step of the adaptive Bogacki-Shampine adaptive Runge-Kutta method starting from x_n at time t_n is computed as follows. The gradient evaluation are given by

$$\begin{cases} p_{n,1} &= f(t_n, x_n) \\ p_{n,2} &= f(t_n + \frac{1}{2}h_n, x_n + \frac{1}{2}h_n p_{n,1}) \\ p_{n,3} &= f(t_n + \frac{3}{4}h_n, x_n + \frac{3}{4}h_n p_{n,2}) \\ p_{n,4} &= f(t_n + h_n, x_n + \frac{2}{9}h_n p_{n,1} + \frac{1}{3}h_n p_{n,2} + \frac{4}{9}h_n p_{n,3}) \end{cases} \tag{46}$$

And the predictions for $x(t_{n+1})$ are given by

$$\begin{cases} x_{n+1}^{\text{euler}} & = x_n + h_n p_{n,1} \\ x_{n+1}^{\text{midpoint}} & = x_n + h_n p_{n,2} \\ x_{n+1}^{\text{order3}} & = x_n + \frac{2}{9} h_n p_{n,1} + \frac{1}{3} h_n p_{n,2} + \frac{4}{9} h_n p_{n,3} \\ x_{n+1}^{\text{order2}} & = x_n + \frac{7}{24} h_n p_{n,1} + \frac{1}{4} h_n p_{n,2} + \frac{1}{3} h_n p_{n_3} + \frac{1}{8} h_n p_{n,4} \end{cases} \tag{47}$$

The difference between x_{n+1}^{order3} and x_{n+1}^{order2} is used to adapt the step-size. Different rules exist. They all require the user to fix a tolerance $\varepsilon > 0$. The simplest [Quarteroni et al., 2007] is to simply halve the step-size if the error $\texttt{err} = \|x_{n+1}^{\text{order3}} - x_{n+1}^{\text{order2}}\|$ is larger than ε, to double it if the error is smaller than ε/K, and to keep it as is if the error is between ε and ε/K. Usually $K = 2^{p+1}$, where p is the order of the method. Other rules [Söderlind, 2006] are

$$h_{n+1} = (\frac{\varepsilon}{\texttt{err}})^p h_n$$

and

$$\texttt{err} = \sqrt{\frac{1}{d} \sum_{i=1}^{d} \left(\frac{x_{n+1,i}^{(Order3)} - x_{n+1}^{(Order2)}}{\text{sc}_i} \right)^2}$$

where

$$\text{sc}_i = \varepsilon(1 + \max(|x_{n+1,i}^{(Order3)}|, |x_{n+1,i}^{(Order2)}|)$$

and then

$$h_{\text{new}} = h_n \min(\text{facmax}, \max(\text{facmin}, \alpha(\frac{1}{\texttt{err}})^{\frac{1}{p}})$$

where usually facmax $\in [1.5, 5]$ and $\alpha \in \{0.8, 0.9, 0.25^{1/3}, 0.38^{1/3}\}$. Then, if $\texttt{err} \leq 1$, the step-size h_n is kept as is and h_{n+1} is set to h_{new}. And if $\texttt{err} > 1$, then h_n is set to h_{new} and the step is redone.

B Background on Optimal Transport Theory

B.1 Monge and Kantorovich Problems

We follow [Santambrogio, 2015, Villani, 2008, Villani, 2021] in this overview of optimal transport. Denote $\mathcal{P}(E)$ the space of probability measures on a space E. Given probability measures $\mu \in \mathcal{P}(X)$ and $\nu \in \mathcal{P}(Y)$ and a cost function $c : X \times Y \to [0, \infty]$, the starting point of optimal transport is the following problem

Definition 10. The *Monge problem* is

$$\begin{aligned} \inf_{T:X\to Y} \quad & \int_X c(x, T(x))\, \mathrm{d}\mu(x) \\ \text{subject to} \quad & T_{\#}\mu = \nu \end{aligned} \tag{48}$$

Here $T_{\#}\mu$ is the *push-forward measure* of μ through T defined by the formula $T_{\#}\mu(A) := \mu(T^{-1}(A))\ \forall\ A$ measurable. For this definition and the constraint to make sense, T has to be measurable with respect to the σ-algebras of μ and ν. The Monge problem expresses the problem of moving mass distributed according to μ so that it becomes distributed according to ν, while minimizing the sum (so the integral) of efforts made to transport each point, the effort being encoded in the cost function c, which is usually a distance between points.

An admissible T is called a *transport map* between μ and ν and the objective of the problem is called the *transport cost* of T.

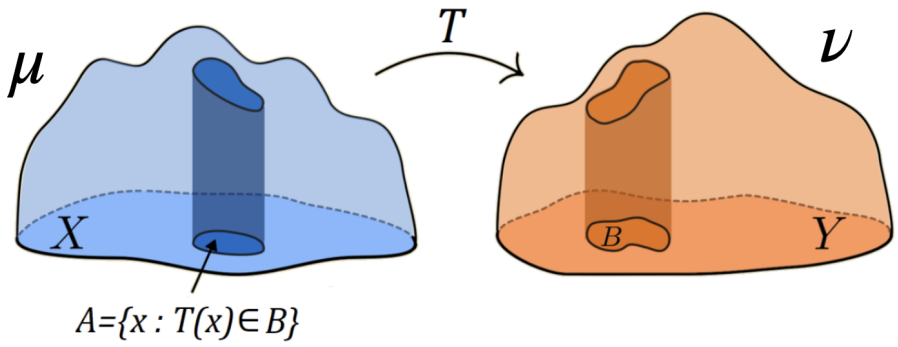

Figure 24: The Monge problem from μ to ν. From [Kolouri et al., 2017].

The Monge problem is difficult because of the constraint on T which is not closed under weak convergence and a T satisfying it need not even exist (e.g. when μ is a Dirac measure and ν is not). A relaxation exists

Definition 11. The *Kantorovitch problem* is

$$\inf_{\gamma \in \Pi(\mu,\nu)} \int_{X\times Y} c \,\mathrm{d}\gamma \tag{49}$$

where $\Pi(\mu,\nu) = \{\gamma \in \mathcal{P}(X\times Y) \mid (\pi_x)_\#\gamma = \mu,\ (\pi_y)_\#\gamma = \nu\}$, π_x is the projection from $X \times Y$ on X and π_y is the projection from $X \times Y$ on Y.

The constraint set $\Pi(\mu,\nu)$ of the Kantorovich problem is never empty as it always contains $\mu \otimes \nu$. This relaxation means that instead of moving the entire mass at x to $T(x)$, $\gamma(A \times B)$ denotes the amount of mass moving from A to B, so that two particles located at $x \in X$ can move to different destination points in Y (see Figure 25 below). An admissible $\gamma \in \Pi(\mu,\nu)$ is called a *transport plan* between μ and ν.

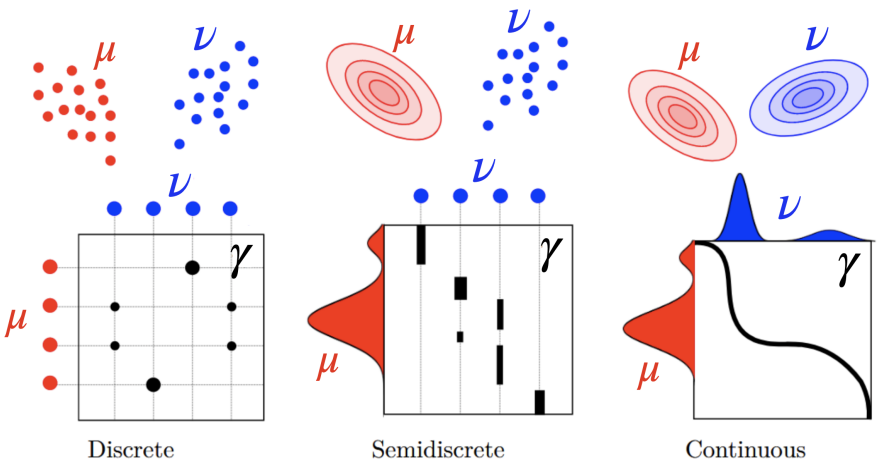

Figure 25: The Kantorovitch problem from μ to ν. From [Peyre and Cuturi, 2019].

The Kantorovich problem has a smaller infimum than the Monge problem, since if $T_\#\mu = \nu$ then $\gamma_T := (\mathtt{id}, T)_\#\mu \in \Pi(\mu,\nu)$ and the transport costs are equal

$$\int_{X\times Y} c \,\mathrm{d}\gamma_T = \int_X c(x, T(x)) \,\mathrm{d}\mu(x) \tag{50}$$

If $\gamma \in \Pi(\mu,\nu)$ is concentrated on a graph in $X \times Y$, i.e. on a set of the form $\{(x, T(x)),\ x \in X\}$, then all the mass in x goes to $T(x)$ and $\gamma = (\mathtt{id}, T)_\#\mu$.

The Kantorovich problem is a relaxation of the Monge problem in the following sense: denote the Kantorovich objective $J(\gamma) := \int c \,\mathrm{d}\gamma$ for $\gamma \in \Pi(\mu,\nu)$. The Monge objective can also be written as a functional K on $\Pi(\mu,\nu)$ defined by $K(\gamma) := J(\gamma)$ if $\gamma = \gamma_T$ for a transport map T and $K(\gamma) := \infty$ otherwise. The Monge problem is equivalent to minimizing K over $\Pi(\mu,\nu)$ because of

(50). Then under some conditions the Kantorovich cost J is the largest lower semi-continuous functional on $\Pi(\mu,\nu)$ that is smaller than the Monge cost K.

For example, if $X = Y = \Omega \subset \mathbb{R}^d$ with Ω compact and c is continuous and μ atomless, then problems (48) and (49) have the same infimum and the set $\{\gamma_T = (\mathtt{id}, T)_{\#}\mu \mid T_{\#}\mu = \nu\}$ is dense (in a weak-* sense) in $\Pi(\mu,\nu)$ so any transport plan can be approximated by transport maps. This allows to prove that J is a relaxation of K.

We have general existence results for the Kantorovich problem. We will prove the following one.

Theorem 23. If X and Y are separable compact metric spaces and c is continuous (or lower semi-continuous), then the Kantorovich problem admits a solution.

Proof. The proof uses the direct method of calculus of variations. First, we know by Riesz representation theorem that, for a separable locally compact metric space E, the space of finite signed Borel measures on E is the dual of $\mathcal{C}_0(E)$ (the set of continuous real valued functions on E that vanish at infinity, which is equal to the set of continous bounded functions $\mathcal{C}_b(E)$ and to the set of continuous functions $\mathcal{C}(E)$ when E is compact). The norm of an element of the dual is the total mass of the measure. $\Pi(\mu,\nu)$ is then included in the dual of $\mathcal{C}(X \times Y)$ and is bounded since all its elements have mass equal to 1. By the Banach-Alaoglu theorem the unit ball of the dual space is weak-* compact in duality with $\mathcal{C}(X \times Y)$.

Denote $J(\gamma) := \int_{X\times Y} c \,\mathrm{d}\gamma$ and l the value of the infimum in (49). Take a minimizing sequence $(\gamma_n)_n$ of problem (49), i.e. a sequence such that $J(\gamma_n) \to l$ and $\gamma_n \in \Pi(\mu,\nu)$. By compacity, we can assume up to sub-sequencing that $\gamma_n \overset{*}{\rightharpoonup} \gamma$. This implies for c continuous that $J(\gamma_n) \to J(\gamma)$ and therefore that $J(\gamma) = l$. It remains to show that $\gamma \in \Pi(\mu,\nu)$. First, γ is clearly a probability measure since by taking the constant function equal to 1 everywhere we have $\gamma(X \times Y) = \int_{X\times Y} 1 \,\mathrm{d}\gamma = \lim \int_{X\times Y} 1 \,\mathrm{d}\gamma_n = 1$. Then take $\varphi \in \mathcal{C}(X)$. We have $\int_{X\times Y} \varphi(x) \,\mathrm{d}\gamma(x,y) = \lim \int_{X\times Y} \varphi(x) \,\mathrm{d}\gamma_n(x,y) = \int \varphi \,\mathrm{d}\mu$, because γ_n is in $\Pi(\mu,\nu)$. Likewise for $\psi \in \mathcal{C}(Y)$, $\int_{X\times Y} \psi(y) \,\mathrm{d}\gamma(x,y) = \int \psi \,\mathrm{d}\nu$.

We used that $(\pi_x)_{\#}\gamma = \mu$ is equivalent to $\int_{X\times Y} \varphi(x) \,\mathrm{d}\gamma(x,y) = \int \varphi \,\mathrm{d}\mu$ for all test functions φ and that $\gamma_n \overset{*}{\rightharpoonup} \gamma$ by definition means $\int_{X\times Y} f \,\mathrm{d}\gamma_n \to \int_{X\times Y} f \,\mathrm{d}\gamma$ for all $f \in \mathcal{C}(X \times Y)$. Finally $\gamma \in \Pi(\mu,\nu)$ and so γ is optimal. □

Existence for the Kantorovich problem can be extended to the case of complete and separable metric spaces (*Polish spaces*) X and Y via Prokhorov's theorem. For the Monge problem, we have the following result which we will need later.

Theorem 24. Consider $X = Y = \Omega \subset \mathbb{R}^d$ with Ω compact and $c(x,y) = h(x-y)$ with h strictly convex. If c is continuous we have the existence of an optimal transport plan γ. If μ is absolutely continuous with respect to the Lebesgue

measure and $\partial\Omega$ is μ-negligible, then the optimal transport plan γ is unique and is of the form $(\mathtt{id}, T)_{\#}\mu$ for a transport map T, meaning that it is constructed from a unique optimal transport map T for the Monge problem.

A final result that we mention is the following, which says that the inverse of the optimal transport map between μ and ν is the optimal transport map from ν to μ,

Theorem 25. If μ and ν are absolutely continuous measures supported respectively on compact subsets X and Y of $\mathbb{R}^d$ with negligible boundaries, and if $c(x, y) = h(x - y)$ with h strictly convex, then there exists a unique couple (T, S) of functions such that the five following points hold

- $T : X \to Y$ and $S : Y \to X$
- $T_{\#}\mu = \nu$ and $S_{\#}\nu = \mu$
- T is optimal for the Monge problem from μ to ν with the cost c
- S is optimal for the Monge problem from ν to μ with the cost c
- $T \circ S \overset{\nu-\text{a.s.}}{=} \mathtt{id}$ and $S \circ T \overset{\mu-\text{a.s.}}{=} \mathtt{id}$

B.2 Duality

The Kantorovitch problem is a linear optimization problem with convex constraints. Duality theory was hence used to study it. We simply state here the duality result, which will be needed to differentiate Wasserstein distances later.

The first step is to rewrite the constraint $\gamma \in \Pi(\mu, \nu)$ as follows

$$\sup_{\varphi,\psi} \int_X \varphi \,\mathrm{d}\mu + \int_Y \psi \,\mathrm{d}\nu - \int_{X\times Y} \varphi(x) + \psi(y) \,\mathrm{d}\gamma(x, y)$$

as this quantity is equal to 0 if $\gamma \in \Pi(\mu, \nu)$ and to ∞ otherwise. The Kantorovitch problem can then be rewritten

$$\inf_{\gamma} \int_{X\times Y} c \,\mathrm{d}\gamma + \sup_{\varphi,\psi} \int_X \varphi \,\mathrm{d}\mu + \int_Y \psi \,\mathrm{d}\nu - \int_{X\times Y} (\varphi(x) + \psi(y)) \,\mathrm{d}\gamma(x, y) \quad (51)$$

Interchanging the infimum and supremum in the above expression, we get

$$\sup_{\varphi,\psi} \int_X \varphi \,\mathrm{d}\mu + \int_Y \psi \,\mathrm{d}\nu + \inf_{\gamma} \int_{X\times Y} (c(x, y) - \varphi(x) - \psi(y)) \,\mathrm{d}\gamma(x, y) \quad (52)$$

The dual of the Kantorovitch problem comes from showing that the maximum of (52) is equal to the minimum of the Kantorovitch problem, and from rewriting $\inf_\gamma \int_{X\times Y}(c(x, y) - \varphi(x) - \psi(y)) \,\mathrm{d}\gamma(x, y)$ as the condition $\varphi(x) + \psi(y) \leq c(x, y)$ first, and then as a property known as *c-concavity*.

Definition 12. Given $\varphi : X \to \bar{\mathbb{R}}$, its *c-transform* (*c-conjugate*) is $\varphi^c : Y \to \bar{\mathbb{R}}$ given by

$$\varphi^c(y) = \inf_{x \in X} c(x, y) - \varphi(x)$$

Similarly, the *$\bar{c}$-transform* of $\psi : Y \to \bar{\mathbb{R}}$ is $\psi^c : X \to \bar{\mathbb{R}}$ given by

$$\psi^{\bar{c}}(x) = \inf_{y \in Y} c(x, y) - \psi(x)$$

Definition 13. A function $\varphi : X \to \bar{\mathbb{R}}$ is said to be *c-concave* if there exists $\psi : Y \to \bar{\mathbb{R}}$ such that $\varphi = \psi^{\bar{c}}$. We denote $c\text{-conc}(X)$ the set of c-concave functions on X.

Theorem 26. If X and Y are separable compact metric spaces and c is uniformly continuous and bounded, then

$$\min_{\gamma \in \Pi(\mu,\nu)} \int_{X \times Y} c \, \mathrm{d}\gamma = \max_{\varphi \in c\text{-conc}(X)} \int_X \varphi \, \mathrm{d}\mu + \int_Y \varphi^c \, \mathrm{d}\nu$$

The functions φ realizing the maximum in the above equation are called the *Kantorovitch potentials* for the transport from μ to ν.

Theorem 27. Under the conditions of Theorem 24, the optimal transport map between μ and ν is given by $T(x) = x - (\nabla h)^{-1}(\nabla \varphi(x))$ where φ is a Kantorovitch potential for the transport from μ to ν.

B.3 The Wasserstein Space

From now on, we take $X = Y = \Omega \subset \mathbb{R}^d$, with Ω complete (i.e. closed), and $c(x, y) = \|x - y\|^p$ for $p \in [1, \infty[$ so that an optimal transport plan for the Kantorovich problem exists (by the extension of Theorem 23 to Polish spaces). Note that if $p > 1$ then c is of the form $c(x, y) = h(x - y)$ with h strictly convex and Theorem 24 stands if Ω is compact and μ absolutely continuous with respect to the Lebesgue measure. Compactness is not necessary if $c(x, y) = \|x - y\|^2$.

A distance between probability measures can be deduced from the Kantorovich problem. For $p \in [1, \infty[$, we restrict ourselves to the set

$$\mathcal{P}_p(\Omega) := \{\mu \in \mathcal{P}(\Omega) \mid \int \|x\|^p \, \mathrm{d}\mu(x) < \infty\} \tag{53}$$

where $\|.\|$ is the euclidean norm. For $\mu, \nu \in \mathcal{P}_p(\Omega)$ we define $W_p(\mu, \nu)$ to be the p-th root of the minimal value of the Kantorovich problem between μ and ν for the cost $c(x, y) = \|x - y\|^p$, i.e.

$$W_p^p(\mu, \nu) := \min_{\gamma \in \Pi(\mu,\nu)} \int_{\Omega \times \Omega} \|x - y\|^p \, \mathrm{d}\gamma(x, y) \tag{54}$$

It can be shown that W_p is a distance over $\mathcal{P}_p(\Omega)$ called the *Wasserstein distance*. One way to prove this is through the *gluing lemma*:

Lemma 1. Let (X_1, μ_1), (X_2, μ_2) and (X_3, μ_3) be Polish probability spaces. Take $\gamma_1 \in \Pi(\mu_1, \mu_2) \subset \mathcal{P}(X_1 \times X_2)$ and $\gamma_2 \in \Pi(\mu_2, \mu_3) \subset \mathcal{P}(X_2 \times X_3)$. Then there exists $\gamma \in \mathcal{P}(X_1 \times X_2 \times X_3)$ such that $(\pi_{1,2})_\# \gamma = \gamma_1$ and $(\pi_{2,3})_\# \gamma = \gamma_2$, where $\pi_{i,j}$ is the projection from $X_1 \times X_2 \times X_3$ on $X_i \times X_j$.

Proposition 6. W_p is a distance over $\mathcal{P}_p(\Omega)$.

Proof. W_p is clearly positive and finite because c is positive and finite, the constraint set is never empty and because of the restriction to $\mathcal{P}_p(\Omega)$. W_p is also clearly symmetric: define $F(x, y) := (y, x)$ then $F_\# \gamma \in \Pi(\nu, \mu)$ if and only if $\gamma \in \Pi(\mu, \nu)$ and the transport costs are the same because c is symmetric.

Taking $\gamma = (\mathtt{id}, \mathtt{id})_\# \mu \in \Pi(\mu, \mu)$ gives $W_p(\mu, \mu) = \int c(x, x) \, \mathrm{d}\mu(x) = 0$. Conversely, $W_p(\mu, \nu) = 0$ implies the existence of $\gamma \in \Pi(\mu, \nu)$ such that $\int c \, \mathrm{d}\gamma = 0$. This means γ has to be concentrated on $c^{-1}(\{0\}) = \{(x, y) \mid x = y\}$. This means that $\gamma = (\mathtt{id}, \mathtt{id})_\# \mu$ and that $\nu = \mathtt{id}_\# \mu = \mu$.

To prove the triangle inequality, consider μ_1, μ_2 and $\mu_3 \in \mathcal{P}_p(\Omega)$. There exists $\gamma_1 \in \Pi(\mu_1, \mu_2)$ an optimal transport plan between μ_1 and μ_2, and $\gamma_2 \in \Pi(\mu_2, \mu_3)$ an optimal transport plan between μ_2 and μ_3. Then by Lemma 1, there exists $\gamma \in \mathcal{P}(\Omega^3)$ with $(\pi_{1,2})_\# \gamma = \gamma_1$ and $(\pi_{2,3})_\# \gamma = \gamma_2$. In particular, this means that $(\pi_{1,3})_\# \gamma \in \Pi(\mu_1, \mu_3)$ because

$$(\pi_{1,3})_\# \gamma(A \times \Omega) = (\pi_1)_\# \gamma(A) = (\pi_1)_\#((\pi_{1,2})_\# \gamma)(A) = (\pi_1)_\# \gamma_1(A) = \mu_1(A)$$

where π_1 denotes the projection on the first coordinate (from Ω^3 or Ω^2 depending on the context) and we used the general formulas $(f \circ g)_\# \gamma = f_\#(g_\# \gamma)$ and $\pi_1 = \pi_1 \circ \pi_{1,2}$.

Likewise $(\pi_{1,3})_\# \gamma(\Omega \times B) = \mu_3(B)$. Since $(\pi_{1,3})_\# \gamma \in \Pi(\mu_1, \mu_3)$ but is not necessarily optimal we can write (denoting $d(x, y) := \|x - y\|$)

$$\begin{aligned}
W_p(\mu_1, \mu_3) &\leq \left(\int d(x, z)^p \, \mathrm{d}(\pi_{1,3})_\# \gamma(x, z) \right)^{\frac{1}{p}} \\
&= \left(\int d(x, z)^p \, \mathrm{d}\gamma(x, y, z) \right)^{\frac{1}{p}} \\
&\leq \left(\int (d(x, y) + d(y, z))^p \, \mathrm{d}\gamma(x, y, z) \right)^{\frac{1}{p}} \\
&\leq \left(\int d(x, y)^p \, \mathrm{d}\gamma(x, y, z) \right)^{\frac{1}{p}} + \left(\int d(y, z)^p \, \mathrm{d}\gamma(x, y, z) \right)^{\frac{1}{p}} \\
&= \left(\int d(x, y)^p \, \mathrm{d}\gamma_1(x, y) \right)^{\frac{1}{p}} + \left(\int d(y, z)^p \, \mathrm{d}\gamma_2(y, z) \right)^{\frac{1}{p}} \\
&= W_p(\mu_1, \mu_2) + W_p(\mu_2, \mu_3)
\end{aligned}$$

where the last equality comes from the optimality of γ_1 and γ_2 and the third inequality comes from the Minkowski inequality $\|f + g\|_p \leq \|f\|_p + \|g\|_p$. □

We denote $\mathbb{W}_p(\Omega)$ the metric space $\mathcal{P}_p(\Omega)$ endowed with the distance W_p, and call it the *Wasserstein space*.

Definition 14. A *curve* w in a metric space (E, d) is a continuous function $w : [0, 1] \to E$. Its *metric derivative* at time t is

$$|w'|(t) := \lim_{h \to 0} \frac{d(w(t+h), w(t))}{|h|} \tag{55}$$

provided the limit exists.

Definition 15. A curve w is *absolutely continuous* if there exists $g \in L^1([0, 1])$ such that

$$d(w(t_0), w(t_1)) \leq \int_{t_0}^{t_1} g(s)\,\mathrm{d}s \quad \forall\ t_0 < t_1 \in [0, 1] \tag{56}$$

We denote $\mathrm{AC}(E)$ the set of absolutely continuous curves valued in E. From the bound in (56) we see that absolutely continuous curves admit a metric derivative at all times.

Definition 16. The *length* of a curve w is

$$\mathrm{Length}(w) := \sup \left\{ \sum_{k=0}^{n-1} d(w(t_k), w(t_{k+1})) \mid 0 = t_0 < .. < t_n = 1, n \geq 1 \right\} \tag{57}$$

From Definition 15 of absolutely continuous curves and Definition 16 of the length of curve, we can see that every absolutely continuous curve w has finite length, as $\mathrm{Length}(w) \leq \int_0^1 g(s)\,\mathrm{d}s < \infty$ for some $g \in L^1([0, 1])$. Actually, we have a formula for the length of an absolutely continuous curve.

Proposition 7. For $w \in \mathrm{AC}(E)$

$$\mathrm{Length}(w) = \int_0^1 |w'|(t)\,\mathrm{d}t \tag{58}$$

It is immediate from the definition of length and the triangle inequality that for any $x, y \in E$ we have $d(x, y) \leq \mathrm{Length}(w)$ for every curve w connecting x to y.

Definition 17. A metric space (E, d) is called a *geodesic space* if $\forall\ x, y \in E$

$$d(x, y) = \min \Big\{ \mathrm{Length}(w) \mid w \in \mathrm{AC}(E),\ w(0) = x,\ w(1) = y \Big\} \tag{59}$$

which means that an absolutely continuous curve must realize the minimum and that its length must be equal to $d(x, y)$. Such a curve is called a *geodesic* between x and y.

Definition 18. A curve w in (E, d) is called a *constant-speed geodesic* between $w(0)$ and $w(1)$ if for all $t, s \in [0, 1]$

$$d(w(t), w(s)) = |t - s|\ d(w(0), w(1)) \tag{60}$$

If we take $g(t) = d(w(0), w(1))$ for all $t \in [0,1]$, it is immediate that a constant-speed geodesic w is absolutely continuous. Its metric derivative is clearly equal to $d(w(0), w(1))$ at all times. This implies via Proposition 7 that its length is $d(w(0), w(1))$ and therefore that it is a geodesic between its endpoints. In fact, we have two other equivalent definitions of constant-speed geodesics.

Proposition 8. Fix $p > 1$ and consider a curve w connecting x to y in a metric space (E, d), then the following three points are equivalent

- w is a constant-speed geodesic
- w is absolutely continuous and $|w'|(t) = d(w(0), w(1))$ for almost every t
- w solves $\min\{\int_0^1 |v'|^p(t)\,\mathrm{d}t \mid v \in \mathrm{AC}(E),\ v(0) = x,\ v(1) = y\}$

Now we prove that $\mathbb{W}_p(\Omega)$ is a geodesic space when Ω is convex.

Theorem 28. $\mathbb{W}_p(\Omega)$ for Ω convex is a geodesic space, i.e. there exists a geodesic curve between every two points in $\mathbb{W}_p(\Omega)$.

More precisely, for $t \in [0,1]$, define $\pi_t : \Omega \times \Omega \to \Omega$ through $\pi_t(x,y) = (1-t)x + ty$. For any $\mu, \nu \in \mathbb{W}_p(\Omega)$, there exists by the previous section an optimal transport plan $\gamma \in \Pi(\mu, \nu)$ between μ and ν for the cost $c(x,y) = \|x - y\|^p$, i.e. γ is optimal for the problem in (54). The curve defined by $\mu_t := (\pi_t)_\# \gamma$ for $t \in [0,1]$ is then a constant-speed geodesic between μ and ν.

Proof. Take $\mu, \nu \in \mathcal{P}_p(\Omega)$ and $\gamma \in \Pi(\mu, \nu)$ optimal for problem (54). Define the curve $(\mu_t)_t$ through $\mu_t := (\pi_t)_\# \gamma$. It is immediate that $\mu_0 = \mu$ and $\mu_1 = \nu$. We start by proving that for every $t < s$, $W_p(\mu_t, \mu_s) \leq |t - s|\ W_p(\mu, \nu)$. First, take $\gamma_t^s := (\pi_t, \pi_s)_\# \gamma$. Then $\gamma_t^s \in \Pi(\mu_t, \mu_s)$ because for every measurable set $A \subset \Omega$ we have

$$\gamma_t^s(\Omega \times A) = \gamma((\pi_t, \pi_s)^{-1}(\Omega \times A)) = \gamma(\pi_s^{-1}(A)) = ((\pi_s)_\# \gamma)(A) = \mu_s(A)$$

and likewise $\gamma_t^s(B \times \Omega) = \mu_t(B)$. Since γ_t^s is an admissible transport plan between μ_t and μ_s but not necessarily an optimal one we can write

$$\begin{aligned} W_p(\mu_t, \mu_s) &\leq \left(\int \|x - y\|^p \,\mathrm{d}\gamma_t^s(x,y) \right)^{\frac{1}{p}} \\ &= \left(\int \|\pi_t(x,y) - \pi_s(x,y)\|^p \,\mathrm{d}\gamma(x,y) \right)^{\frac{1}{p}} \\ &= |t - s| \left(\int \|x - y\|^p \,\mathrm{d}\gamma(x,y) \right)^{\frac{1}{p}} \\ &= |t - s| W_p(\mu, \nu) \end{aligned}$$

The first equality comes from the definition of γ_t^s as a push-forward of γ, i.e from the formula $\int F(x,y)\,\mathrm{d}\gamma_t^s(x,y) = \int F(\pi_t(x,y), \pi_s(x,y))\,\mathrm{d}\gamma(x,y)$ for any function F. The second equality comes from the definitions of $\pi_t(x,y)$ and

$\pi_s(x,y)$ and the relation $\|(1-t)x+ty-(1-s)x-sy\| = |t-s|\|x-y\|$. The third equality comes from the fact that γ is an optimal transport plan between μ and ν.

The inequality $W_p(\mu_t,\mu_s) \leq |t-s|\ W_p(\mu,\nu)$ for $t<s$ is actually an equality, because if it were a strict inequality we could write

$$\begin{aligned} W_p(\mu,\nu) &\leq W_p(\mu,\mu_t) + W_p(\mu_t,\mu_s) + W_p(\mu_s,\nu) \\ &< W_p(\mu,\nu)(t+(s-t)+(1-s)) \\ &= W_p(\mu,\nu) \end{aligned}$$

and so we would have $W_p(\mu,\nu) < W_p(\mu,\nu)$ which is impossible. The equality $W_p(\mu_t,\mu_s) = |t-s|\ W_p(\mu,\nu)$ for all t,s shows that $(\mu_t)_t$ is a constant-speed geodesic between μ and ν according to Definition 18 and concludes the proof that $\mathbb{W}_p(\Omega)$ for Ω convex is a geodesic space. Convexity of Ω is needed so that $\pi_t(\Omega\times\Omega)\subset\Omega$. □

If Ω is convex and $p>1$, then every constant-speed geodesic in $\mathbb{W}_p(\Omega)$ is of the form $((\pi_t)_{\#}\gamma)_t$ for an optimal transport plan γ.

B.4 The Dynamic Formulation of Optimal Transport

Chapter 5 of [Santambrogio, 2015] also shows a link between geodesics in $\mathbb{W}_p(\Omega)$ (so optimal transport plans) and solutions of the *continuity equation*

$$\partial_t\mu_t + \nabla\cdot(\mu_t v_t) = 0,\ \ \mu_0 = \mu \tag{61}$$

where for every $t\in[0,1]$, μ_t is a probability measure on Ω and $v_t:\Omega\to\mathbb{R}^d$ is a vector field. When looking for solutions $(\mu_t,v_t)_t$ to the continuity equation, μ_t does not have to be a density as we need weak solutions where we only integrate against μ_t.

More precisely, we say that $(\mu_t,v_t)_t$ solves the continuity equation in the *weak sense* if for any test function $\psi\in\mathcal{C}^1_c(\bar{\Omega})$, the function $t\mapsto\int\psi\,\mathrm{d}\mu_t$ is absolutely continuous in t and for almost every $t\in[0,1]$ we have

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_\Omega \psi\,\mathrm{d}\mu_t = \int_\Omega \nabla\psi\cdot v_t\,\mathrm{d}\mu_t \tag{62}$$

If μ is Lipschitz continuous in (t,x) and v is Lipschitz continuous in x then being a weak solution is equivalent to being a solution almost everywhere.

If is well known that under some conditions (Ω bounded and v Lipschitz and bounded in x uniformly in t with a flow that remains in Ω), the continuity equation describes the evolution of the density μ_t of particles distributed initially according to $\mu_0=\mu$ and moving according to the velocity field v_t.

More precisely, given a time-dependent vector field v, if points $x \in \Omega$ are distributed according to $\mu_0 = \mu$ and move tangentially to v at all times, i.e satisfy

$$\partial_t \phi_t(x) = v_t(\phi_t(x)),\ \phi_0(x) = x \tag{63}$$

where $\phi_t(x)$ is the position of point x at time t, then their distribution across time $((\phi_t)_{\#}\mu)_t$ solves the continuity equation (61) for the given vector field. This is the unique solution if we restrict ourselves to distributions that are absolutely continuous with respect to the Lebesgue measure. A solution ϕ to (63) is called the *flow* of the differential equation (63) determined by v.

The link with optimal transport comes from the equivalence between absolutely continuous curves in $\mathbb{W}_p(\Omega)$ and solutions of the continuity equation.

Theorem 29. Take $p > 1$ and Ω compact.

- If $(\mu_t)_t$ is an absolutely continuous curve in $\mathbb{W}_p(\Omega)$, then there exists a vector field $v_t \in L^p(\mathbb{R}^d; \mu_t)$ for every $t \in [0,1]$ such that $\partial_t \mu_t + \nabla \cdot (\mu_t v_t) = 0$ is satisfied in a weak sense.
- Conversely, if $(\mu_t)_t$ is a family of measure in $\mathcal{P}_p(\Omega)$ and we have for every $t \in [0,1]$ a vector field $v_t \in L^p(\mathbb{R}^d; \mu_t)$ with $\int_0^1 \|v_t\|_{L^p(\mu_t)}\, \mathrm{d}t < \infty$ such that $\partial_t \mu_t + \nabla \cdot (\mu_t v_t) = 0$ is satisfied in a weak sense, then $(\mu_t)_t$ is an absolutely continuous curve in $\mathbb{W}_p(\Omega)$.
- Furthermore, $(\mu_t)_t$ and $(v_t)_t$ solving the continuity equation together must satisfy $\|v_t\|_{L^p(\mu_t)} = |\mu'|(t)$ for almost every t.

If Ω is convex and compact and $p > 1$, we can combine this result with the fact that $\mathbb{W}_p(\Omega)$ is a geodesic space to obtain another formula for the Wasserstein distance.

Theorem 30. On a convex and compact set Ω, for all $\mu, \nu \in \mathcal{P}_p(\Omega)$ with $p > 1$ we have

$$W_p^p(\mu,\nu) = \min \left\{ \int_0^1 \|v_t\|^p_{L^p(\rho_t)}\, \mathrm{d}t \ \middle|\ \partial_t \rho_t + \nabla \cdot (\rho_t v_t) = 0,\ \rho_0 = \mu,\ \rho_1 = \nu \right\}$$

This is known as the *Benamou-Brenier formula*.

Proof. First, given Definition 17 of geodesic spaces and formula (58) for the length of absolutely continuous curves we can write for any $\mu, \nu \in \mathcal{P}_p(\Omega)$

$$W_p^p(\mu,\nu) = \left(\min \left\{ \int_0^1 |\rho'|(t)\, \mathrm{d}t \ \middle|\ \rho \in \mathrm{AC}(\mathbb{W}_p(\Omega)),\ \rho_0 = \mu,\ \rho_1 = \nu \right\} \right)^p$$

Since we proved the existence of constant-speed geodesics between every two points of $\mathbb{W}_p(\Omega)$, the minimum can be restricted to these geodesics. Given the

second equivalent definition we gave of constant-speed geodesics when $p > 1$ in Proposition 8, we can write

$$W_p^p(\mu,\nu) = \min\left\{ \int_0^1 |\rho'|^p(t)\,\mathrm{d}t \;\;|\;\; \rho \in \mathrm{AC}(\mathbb{W}_p(\Omega)),\ \rho_0 = \mu,\ \rho_1 = \nu \right\}$$

Finally, the equivalence between absolutely continuous curves in $\mathbb{W}_p(\Omega)$ and solutions of the continuity equation allows us to minimize over theses solutions

$$W_p^p(\mu,\nu) = \min\left\{ \int_0^1 \|v_t\|^p_{L^p(\rho_t)}\,\mathrm{d}t \;\;|\;\; \partial_t\rho_t + \nabla\cdot(\rho_t v_t) = 0,\ \rho_0 = \mu,\ \rho_1 = \nu \right\}$$

where we also used the relation $\|v_t\|_{L^p(\mu_t)} = |\rho'|(t)$ between $(\rho_t)_t$ and $(v_t)_t$ that jointly solve the continuity equation. □

In this dynamic formulation of the Kantorovich problem we solve for pairs $(\rho_t, v_t)_t$ satisfying the continuity equation with border conditions $\rho_0 = \mu$ and $\rho_1 = \nu$ that minimize the energy $\int_0^1 \|v_t\|^p_{L^p(\mu_t)}$ which is the integral across time of the integral across points of the norms of the tangent vectors to the movement of points distributed initially according to μ, so another way of expressing the total distance traveled by all points.

From an optimal transport plan γ between μ and ν we can construct the geodesic $((\pi_t)_\#\gamma)_t$ and the associated velocity field v_t for $t \in]0,1]$ without solving the continuity equation. In fact, if an optimal transport map T exists, then $v_t = (T - \mathrm{id}) \circ T_t^{-1}$, where $T_t = (1-t)\mathrm{id} + tT$ and is invertible. Since the continuity equation needs to hold only weakly, mass can still be split between different destinations.

However, we want to solve directly for $(\mu_t, v_t)_t$. As we have mentioned above, if we restrict ourselves to time-dependent vector fields v that are Lipschitz and bounded in x uniformly in t and have a flow ϕ that remains in Ω and to distributions that are absolutely continuous with respect to the Lebesgue measure, then the continuity equation has a unique solution (μ_t) for a given v. The minimization is therefore only over vector fields and we have the problem

$$\begin{aligned} \inf_v \quad & \int_0^1 \|v_t\|^p_{L^p(\mu_t)}\,\mathrm{d}t \\ \text{subject to} \quad & \partial_t\mu_t + \nabla\cdot(\mu_t v_t) = 0,\ \mu_0 = \mu,\ \mu_1 = \nu \end{aligned} \tag{64}$$

Since in our machine learning setting distributions μ and ν are always discrete samples, we prefer to use the constraint (63) on the movement of points:

$$\begin{aligned} \inf_v \quad & \int_0^1 \|v_t\|^p_{L^p((\phi_t)_\#\mu)}\,\mathrm{d}t \\ \text{subject to} \quad & \partial_t\phi_t(x) = v_t(\phi_t(x))\ \forall x \in \Omega,\ \phi_0 = \mathtt{id},\ (\phi_1)_\#\mu = \nu \end{aligned} \tag{65}$$

where we used the fact that $((\phi_t)_{\#}\mu)_t$ with ϕ solution to (63) is a solution to the continuity equation for a given v. Again, since in all generality we don't have uniqueness of ϕ solution to (63), then the optimization is actually over $(v_t, \phi_t)_t$. Since we are going to discretize the problem and use neural networks to approximate the velocity field, we can limit ourselves to uniformly Lipschitz velocity fields v, so that the solution ϕ exists and is unique and the problem as expressed in (65) is well defined.

If we want (63) to hold for all $t \in [0, 1]$, then this formulation is not exactly equivalent to the first dynamic formulation (64) and to the Kantorovich problem, as all the mass in a position x will follow the velocity field to a unique point $y = \phi_1(x)$. We are then back to the Monge problem, but here we look for a velocity field that points distributed according to μ will follow so that they end up distributed according to ν, while minimizing the total energy of this field.

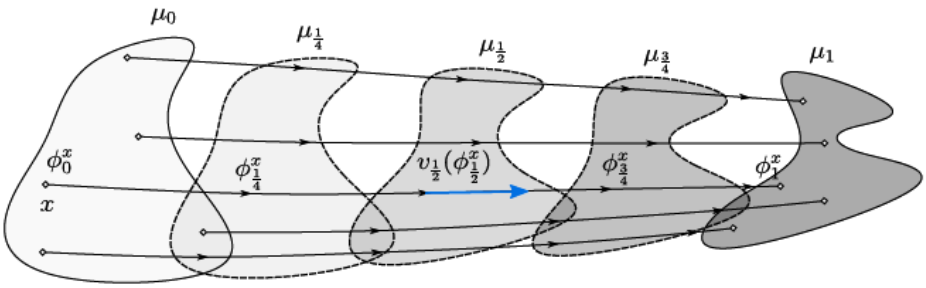

Figure 26: Dynamic transport of μ_0 to μ_1 according to velocity field v (ϕ_t^x is $\phi_t(x)$ in the text). From [de Bézenac et al., 2021].

B.5 Regularity of Optimal Transport Maps

Optimal transport maps have some regularity properties under some further assumptions. In particular, Theorem 6.27 of [Ambrosio et al., 2005] gives a classical almost-everywhere regularity result:

Theorem 31. If $c(x, y) = \|x-y\|^p$ for $p > 1$, and μ and ν have compact supports with $d(\text{supp}(\mu), \text{supp}(\nu)) > 0$, then the optimal transportation map T between μ and ν is $\nu - a.e.$ differentiable and its Jacobian $\nabla T(x)$ has non-negative eigenvalues ν-almost-surely.

More recently, results summarized below, which correspond to Theorems 4.23, 4.24 and Remark 4.25 of [Figalli, 2017], state that the optimal transportation map has one degree of regularity more than the initial transported density:

Theorem 32. Take $c(x, y) = \|x - y\|^p$ for $p > 1$. Suppose there are X, Y, bounded open sets, such that the densities of μ and ν are null in their respective complements and bounded away from zero and infinity over them respectively.

Then, if Y is convex, there exists $\eta > 0$ such that the optimal transport map T

between μ and ν is $C^{0,\eta}$ over X. If Y is not convex, there exist two relatively closed sets A, B in X, Y respectively such that $T \in C^{0,\eta}(X \setminus A, Y \setminus B)$, where A and B are of null Lebesgue measure.

Moreover, if the densities are in $C^{k,\eta}$, then $C^{0,\eta}$ can be replaced by $C^{k+1,\eta}$ in the conclusions above. In particular, if the densities are smooth, then the transport map is a diffeomorphism.

C Background on Gradient Flows

We follow [Santambrogio, 2015, Santambrogio, 2016, Ambrosio et al., 2005, Jordan et al., 1998] for this background on Wasserstein gradient flows.

C.1 Gradient Flows in Euclidean Space

Given $F : \mathbb{R}^d \to \mathbb{R}$ and $x_0 \in \mathbb{R}^d$, a *gradient flow* is a curve x solution of the Cauchy problem

$$\begin{cases} x'(t) = -\nabla F(x(t)) \\ x(0) = x_0 \end{cases} \tag{66}$$

A solution exists and is unique if ∇F is Lipschitz or F is convex. Given $\tau > 0$ and $x_0^\tau = x_0$ define a sequence $(x_k^\tau)_k$ through the *minimizing movement scheme*

$$x_{k+1}^\tau \in \arg\min_{x \in \mathbb{R}^d} \; F(x) + \frac{1}{2\tau}\|x - x_k^\tau\|^2 \tag{67}$$

F lower semi-continous and $F(x) \geq C_1 - C_2\|x\|^2$ guarantees existence of a solution of (67) for τ small enough. F λ-convex meets these conditions and also provides uniqueness of the solution because of strict convexity of the target.

The optimality condition of problem (67) is

$$\frac{x_{k+1}^\tau - x_k^\tau}{\tau} = -\nabla F(x_{k+1}^\tau) \tag{68}$$

which is exactly the formula of the implicit Euler scheme (see Appendix A.1) with step size τ.

We interpret the point x_k^τ as the value of a curve x at time $k\tau$. We then construct a curve x^τ as the piecewise constant interpolation of the points x_k^τ

$$x^\tau(t) = x_k^\tau \text{ for } t \in](k-1)\tau,\ k\tau], k \in \{1, .., K\} \tag{69}$$

We can also construct a curve $\tilde{x}^\tau$ as the affine interpolation of the points x_k^τ

$$\tilde{x}^\tau(t) = x_{k-1}^\tau + (t-(k-1)\tau)\frac{x_k^\tau - x_{k-1}^\tau}{\tau} \text{ for } t \in](k-1)\tau,\ k\tau], k \in \{1, .., K\} \tag{70}$$

If $F(x_0) < \infty$ and $\inf F > -\infty$ then x^τ and $\tilde{x}^\tau$ converge uniformly to the same curve x as τ goes to zero (up to extracting a subsequence). If F is $\mathcal{C}^1$, then the limit curve x is a solution of (66) (i.e. a gradient flow of F). If F is not differentiable then x is solution of the problem defined using the subdifferential of F, i.e. x satisfies $x'(t) \in -\partial F(x(t))$ for almost every t.

If $\lambda > 0$, then the solution to (66) converges exponentially to the unique minimizer of F (which exists by coercivity). So by taking $\tau \to 0$ and $k \to \infty$, we tend towards the minimizer of F.

C.2 Gradient Flows in Metric Spaces

The advantage of the minimizing movement scheme (67) is that it can be adapted to metric spaces. If (E, d) is a metric space, $x_0 \in E$ and $F : E \to \mathbb{R} \cup \{\infty\}$ is lower semi-continuous and bounded from below, we can define a sequence $(x_k^\tau)_k$ with $x_0^\tau = x_0$ and

$$x_{k+1}^\tau \in \arg\min_{x \in E} \; F(x) + \frac{1}{2\tau} d(x, x_k^\tau)^2 \tag{71}$$

If E is compact then a solution to (71) exists. We study the limit as τ goes to 0 of the piecewise constant curve x^τ constructed from these points by (69). In the euclidean setting, we had uniform convergence to a gradient flow of F.

A curve $x : [0, T] \to E$ is called a *minimizing movement* for F and x_0 if there exists a sequence of time steps $(\tau_j)_j$ that goes to 0 such that the sequence of piecewise constant curves $(x^{\tau_j})_j$ uniformly converges to x as $j \to \infty$.

Again, if $F(x_0) < \infty$ and $\inf F > -\infty$ then a minimizing movement exists because we can extract a uniformly converging subsequence of curves from the family of piecewise constant curves $(x^\tau)_\tau$ as $\tau \to 0$.

If we add some structure to (E, d) by assuming it is a geodesic space, we can also construct a metric counterpart of the affine interpolation (70) using geodesics

$$\tilde{x}^\tau(t) = w_{x_{k-1}^\tau, x_k^\tau}\left(\frac{t - (k-1)\tau}{\tau}\right) \text{ for } t \in](k-1)\tau,\ k\tau] \text{ and } k \in \{1, .., K\} \tag{72}$$

where $w_{x_{k-1}^\tau, x_k^\tau}$ is any constant-speed geodesic between x and y. The curve $\tilde{x}^\tau$ is continuous and locally Lipschitz (because geodesics are) and coincides with the piecewise constant curve x^τ at times $k\tau$. If $F(x_0) < \infty$ and $\inf F > -\infty$ then $(\tilde{x}^\tau)_\tau$ is also compact and converges to the same minimizing movement as x^τ when $\tau \to 0$.

However, if x is a minimizing movement for F we cannot say that it is a gradient flow of F, i.e. that it satisfies (66), because $x' = -\nabla F$ does not mean anything in a metric space. We characterize gradient flows in a metric space by finding equivalent conditions to $x' = -\nabla F$ in the euclidean setting that have a metric counterpart. The first such condition is obtained through the Cauchy-Schwartz inequality and is called the *energy dissipation equality* (EDE)

$$F(x(s)) - F(x(t)) = \int_s^t \frac{1}{2}\left(\|x'(r)\|^2 + \|\nabla F(x(r))\|^2\right) \mathrm{d}r \;\forall\; s < t \tag{73}$$

In the metric setting, we use the metric derivative of a curve $|x'|(t)$ and the *descending slope* of a real-valued function

$$|\nabla^- F|(x) := \limsup_{y \to x} \frac{(F(x) - F(y))_+}{d(x, y)} \tag{74}$$

Under some conditions on the descending slope of F (in particular lower semi-continuity) a minimizing movement for F, i.e. the limit of an interpolation of the minimizing movement scheme, satisfies the EDE condition (73).

If F is λ-convex, another condition which is equivalent to $x'(t) \in -\partial F(x(t))$ in the euclidean setting is the *evolution variational inequality* (EVI)

$$\frac{1}{2}\frac{\mathrm{d}}{\mathrm{d}t}\|x(t)-y\|^2 \leq F(y) - F(x(t)) - \frac{\lambda}{2}\|x(t)-y\|^2 \ \forall\ y \in \mathbb{R}^d \tag{75}$$

In the metric setting, we replace the euclidean distance by the distance d and obtain a different definition of gradient flows in metric spaces. The EVI condition leads to uniqueness and stability results for gradient flows and implies the EDE condition, but more assumptions are needed to prove that minimizing movements satisfy it.

C.3 Gradient Flows in the Wasserstein Space

In the geodesic metric space $\mathbb{W}_2(\Omega)$ with Ω compact and convex, for the functional $F : \mathbb{W}_2(\Omega) \to \mathbb{R} \cup \{\infty\}$ lower semi-continuous for the weak convergence of measures in duality with $\mathcal{C}(\Omega)$ and $\rho_0^\tau = \rho_0 \in \mathcal{P}(\Omega)$, the minimizing movement scheme (71) becomes

$$\rho_{k+1}^\tau \in \arg\min_{\rho \in \mathcal{P}_2(\Omega)} \ F(\rho) + \frac{1}{2\tau} W_2^2(\rho, \rho_k^\tau) \tag{76}$$

This problem has a solution because the objective is lower semi-continuous and the minimization is over $\mathcal{P}(\Omega)$ which is compact by Banach-Alaoglu.

We can construct a piecewise constant interpolation between the measures ρ_k^τ as in (69), or a geodesic interpolation as in (72), where we travel along a geodesic between ρ_{k-1}^τ and ρ_k^τ in $\mathbb{W}_2(\Omega)$ (see Theorem 28 in Appendix B.3 for how to construct geodesics in $\mathbb{W}_2(\Omega)$ from optimal transport maps). Again, if $F(x_0) < \infty$ and $\inf F > -\infty$ then both interpolations converge uniformly to a limit curve $(\tilde{\rho}_t)$ as τ goes to zero. If F is also λ-convex along geodesics, then we might be able to prove the EVI condition for the minimizing movement $\tilde{\rho}$.

However, the space $\mathbb{W}_2(\Omega)$ allows us to consider another possible property of the minimizing movement that can also provide uniqueness and stability. We first need to define a notion of differentiation for functionals defined on probability spaces.

Definition 19. Given $F : \mathcal{P}(\Omega) \to \mathbb{R} \cup \{\infty\}$, we say that $\rho \in \mathcal{P}(\Omega)$ is *regular* for F if $F((1-\varepsilon)\rho + \varepsilon\hat{\rho}) < \infty$ for all $\varepsilon \in [0,1]$ and all $\hat{\rho} \in \mathcal{P}(\Omega) \cap L_c^\infty(\Omega)$ (absolutely continuous probability measures with densities in $L_c^\infty(\Omega)$).

Definition 20. Given $F : \mathcal{P}(\Omega) \to \mathbb{R} \cup \{\infty\}$ and $\rho \in \mathcal{P}(\Omega)$ that is regular for F we call the *first variation* of F at ρ and denote $\frac{\delta F}{\delta \rho}(\rho)$, any measurable function, if it exists, such that

$$\frac{\mathrm{d}}{\mathrm{d}\varepsilon} F(\rho + \varepsilon\chi)\Big|_{\varepsilon=0} = \int_\Omega \frac{\delta F}{\delta \rho}(\rho)\,\mathrm{d}\chi \tag{77}$$

for every $\chi = \hat{\rho} - \rho$ with $\hat{\rho} \in \mathcal{P}(\Omega) \cap L_c^\infty(\Omega)$.

Remark 33. If there is a unique Kantorovitch potential φ up to additive constants from μ to ν (see Theorem 26 in Appendix B.2), then $\frac{\delta W_p^p(.,\nu)}{\delta \rho}(\mu) = \varphi$.

We can now say that the minimizing movement $\tilde{\rho}$ may satisfy the following continuity equation:

$$\partial_t \rho_t(x) - \nabla \cdot (\rho_t(x) \nabla (\frac{\delta F}{\delta \rho}(\rho_t))(x)) = 0 \tag{78}$$

And we can then use the two following results. First from [Santambrogio, 2016]:

Proposition 9. If the functional $F : \mathcal{P}(\Omega) \to \mathbb{R} \cup \{\infty\}$ is $\lambda-$geodesically convex and the curves ρ^0 and ρ^1 satisfy the continuity equation (78) then

$$\frac{1}{2} \frac{\mathrm{d}}{\mathrm{d}t} W_2^2(\rho_t^0, \rho_t^1) \leq -\frac{\lambda}{2} W_2^2(\rho_t^0, \rho_t^1)$$

and by Gronwall's inequality

$$W_2^2(\rho_t^0, \rho_t^1) \leq W_2^2(\rho_0^0, \rho_0^1) e^{-\lambda t}$$

So we have uniqueness of the solution to (78) for the same initial condition ρ_0 and stability with exponential convergence as $t \to \infty$ if $\lambda > 0$.

And from [Santambrogio, 2015]:

Proposition 10. Given a functional $F : \mathcal{P}(\Omega) \to \mathbb{R} \cup \{\infty\}$ and a probability measure $\rho^\star$ that is regular for F and that minimizes it, suppose $\frac{\delta F}{\delta \rho}(\rho^\star)$ exists and let $l = \operatorname{ess\,inf} \frac{\delta F}{\delta \rho}(\rho^\star)$ be its essential infimum with respect to the Lebesgue measure.

Then if $\frac{\delta F}{\delta \rho}(\rho^\star) \in \mathcal{C}(\Omega)$ then $\forall x \in \Omega$, $\frac{\delta F}{\delta \rho}(\rho^\star)(x) \geq l$ and $\forall x \in \operatorname{support}(\rho^\star)$, $\frac{\delta F}{\delta \rho}(\rho^\star)(x) = l$.

And if $\frac{\delta F}{\delta \rho}(\rho^\star)$ is only measurable but $\rho^\star$ is absolutely continuous, then for almost every $x \in \Omega$, $\frac{\delta F}{\delta \rho}(\rho^\star)(x) \geq l$ and for almost every $x \in \{\rho^\star > 0\}$, $\frac{\delta F}{\delta \rho}(\rho^\star)(x) = l$.

Therefore if $\rho^\star$ minimizes F while satisfying the conditions of Proposition 10, then the constant curve $\bar{\rho}_t = \rho^\star$ satisfies the continuity equation (78) in the weak sense or almost everywhere as both terms are equal to zero. If the minimizing movement ρ also satisfies the continuity equation (78) then by Proposition 9, $W_2(\tilde{\rho}_t, \rho^\star) \xrightarrow[t \to \infty]{} 0$ exponentially if F is λ-geodesically convex for $\lambda > 0$.

Why does it make sense for $\tilde{\rho}$ to satisfy the continuity equation (78)? By Proposition 10 and Remark 33, the solution ρ_{k+1}^τ to problem (76) satisfies the optimality condition

$$\frac{\delta F}{\delta \rho}(\rho_{k+1}^\tau) + \frac{\varphi}{\tau} = \texttt{const} \tag{79}$$

where φ is the Kantorovich potential associated with the optimal transport S from ρ_{k+1}^τ to ρ_k^τ. By Theorem 27 in Appendix B.2 (with h being the Euclidean norm), $S(x) = x - \nabla\varphi(x)$, we consider the velocity v of particles moving from ρ_k^τ to ρ_{k+1}^τ. According to Appendix B.4, it is given by

$$-v(x) = \frac{S(x) - x}{\tau} = -\frac{\nabla\varphi(x)}{\tau} = \nabla(\frac{\delta F}{\delta\rho}(\rho_{k+1}^\tau))(x) \tag{80}$$

Therefore, as $\tau \to 0$, we can expect to get in $\tilde{\rho}$ a solution to the continuity equation (78), but this has to be proven rigorously for a given F.

D Background on Module-wise Training

The typical setting of (sequential) module-wise training for minimizing a loss L, is, given a dataset $\mathcal{D}$, to solve one after the other, for $1 \leq k \leq K$, Problems

$$(T_k, F_k) \in \arg\min_{T,F} \sum_{x \in \mathcal{D}} L(F, T(G_{k-1}(x))$$

where $G_k = T_k \circ ... \circ T_1$ for $1 \leq k \leq K$ and $G_0 = \texttt{id}$. Here, T_k is the module (one or many layers) and it receives the output of module T_{k-1}, and F_k is an auxiliary classifier that processes the outputs of T_k so the loss can be computed. The final network trained this way is $F_K \circ G_K$, but we can stop at any depth k and use $F_k \circ G_k$ if it performs better.

In fact, module-wise training suffers often from a well-documented *stagnation problem* observed in [Marquez et al., 2018, Belilovsky et al., 2019, Wang et al., 2021, Pyeon et al., 2021], whereby greedy early modules overfit and learn more discriminative features than end-to-end training, and deeper modules don't improve the test accuracy significantly, or even degrade it.

We present below some recent methods that aim at alleviating this problem and improving the accuracy of module-wise training, either by adding other terms to the loss used to train the modules, or simply through architectural considerations.

D.1 Additional Loss Terms

InfoPro [Wang et al., 2021]. InfoPro start by verifying that vanilla module-wise training outperforms end-to-end training in the first modules, but then destroys too much information between the learned features and both the inputs and the labels and stagnates and gets overtaken by end-to-end training in the later modules. See Figure 27 below.

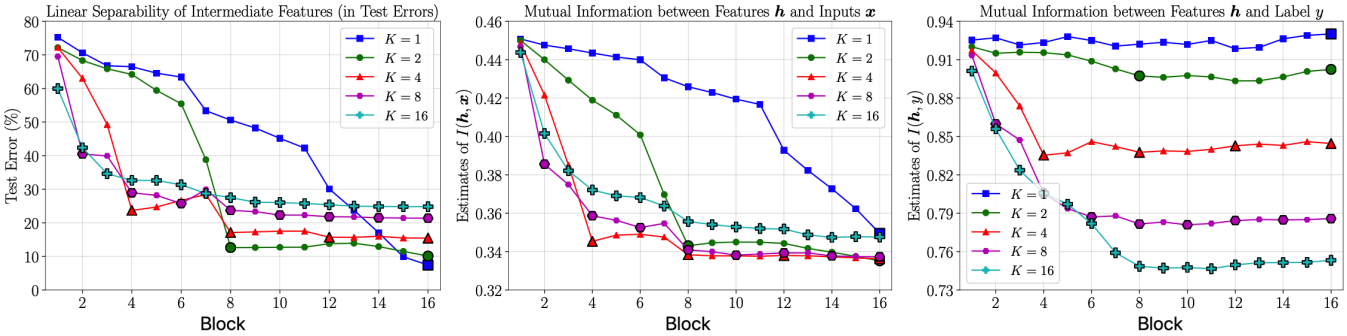

Figure 27: Linear separability (left), mutual information with input (center), and mutual information with output (right) of features learned with module-wise training with K modules. K=1 is end-to-end training. From [Wang et al., 2021].

They then propose to train each module to maximize the mutual information between the learned features h and both the input x and its label y. This means minimizing

$$-\lambda_1 I(h, x) - \lambda_2 I(h, y)$$

where I is the mutual information given by $I(a, b) = H(b) - H(b|a)$ where H is the entropy [Shannon, 1948]. The mutual information $I(h, y)$ with the label is approximated using an auxiliary classifier with the cross-entropy loss or a contrastive loss inspired by [Chen et al., 2020b, Khosla et al., 2020, He et al., 2020]. The mutual information $I(h, x)$ with the input is approximated using an auxiliary decoder that tries to reconstruct x from h with a reconstruction loss that is binary cross-entropy.

In [Pathak et al., 2022], this method is slightly modified in that the auxiliary decoder now tries to reconstruct the previous feature (i.e the input to the module) and not the initial input x to the network.

PredSim [Nøkland and Eidnes, 2019]. PredSim adds an auxiliary similarity matching loss inspired by similarity losses used in neuroscience [Patel et al., 2022]. The similarity matching loss is the distance between matrices whose entries are meant to measure the similarity between inputs in a mini-batch. The target for such a matrix is then the similarity matrix constructed from the labels of the inputs, which contains 1 at position (i, j) if inputs i and j have the same label, and 0 otherwise. More precisely, given a matrix A made up of stacked vectors a_i, we denote $S(A)$ the *adjusted cosine similarity matrix* or *correlation matrix*, whose entries are given by

$$S(A)_{i,j} = S(A)_{j,i} = \frac{\tilde{a}_i^\intercal \tilde{a}_j}{\|\tilde{a}_i\|_2 \|\tilde{a}_j\|_2}$$

where we mean-center vector a_i into vector $\tilde{a}_i$. Given a mini-batch, we denote Y the one-hot encoded matrix of its labels, and H the matrix of the stacked embeddings h_i of the elements of the mini-batch after the current module. The similarity matching loss is then $\|S(\texttt{Conv}(H) - S(Y)\|_F^2$, meaning that the embeddings go through a convolutional auxiliary layer to reduce their dimension. This loss is added to the cross-entropy classification loss for the training of every module.

D.2 Architectural Approaches

DGL [Belilovsky et al., 2020]. DGL, and previously [Belilovsky et al., 2019], simply consider the best auxiliary architectures for module-wise training. They find that deeper convolutional auxiliary classifiers lead to the best performances. However, these auxiliary classifiers are quite large which reduces the computational savings we desire from module-wise training. We, like InfoPro [Wang et al., 2021], limit ourselves to one convolutional layer. DGL then introduce an auxiliary architecture made up of staged resolution reduction phase before the

linear classification layer, which they find offers a good trade-off between size and performance.

Sedona [Pyeon et al., 2021]. Sedona design light auxiliary classifiers inspired by the architecture of MobileNetV2 [Sandler et al., 2018] that relies on depth-wise and point-wise convolutions and inverted residual blocks. They also introduce an architecture search phase that chooses the best auxiliary classifier architecture from a pool of candidates, and where exactly to insert them in the network. This search phase however leads to a deep early module which reduces the memory savings of module-wise training.

E Background on Adversarial Attacks

E.1 Adversarial Attacks

We present here how some popular adversarial attacks on neural networks that solve an image classification task work. We refer to [Aldahdooh et al., 2022, Li et al., 2022] for more details.

Below, L is the classification loss (usually cross-entropy), x is a clean image, y is its label and d its dimension as a vector, $\tilde{x}$ is the adversarial image found from x, $\delta = x - \tilde{x}$ is the perturbation, and $\epsilon > 0$ is the maximal allowed perturbation size in a given L_p norm. For targeted attacks, $t \neq y$ is the target label the attack wants the network to predict for $\tilde{x}$. We denote the pre-softmax outputs of the network Z, while f denotes the softmax outputs.

E.1.1 White-box Attacks

White-box attacks are attacks where the adversary has access to the network's architecture, weights and gradients. We present below three gradient-based attacks (FGM, BIM and PGD), one optimization-based attack (CW), which is one of the strongest and most widely used attacks, and one boundary-based attack (DF).

FGM [Goodfellow et al., 2015b]. The Fast Gradient Method is a simple early attack. It finds the update direction that maximizes the loss L at each pixel of image x and takes one step in that direction. So

$$\tilde{x} = x + \epsilon \text{ sign}(\nabla_x L(x, y)) \text{ such that } \tilde{x} \in [0, 1]^d$$

This way $\|\tilde{x} - x\|_\infty \leq \epsilon$.

BIM [Kurakin et al., 2017]. The Basic Iterative Method iterates the FGM attack k times. So for $0 \leq i < k$ and $\tilde{x}_0 = x$,

$$\tilde{x}_{i+1} = \tilde{x}_i + \alpha \text{ sign}(\nabla_x L(\tilde{x}_i, y)) \text{ such that } \tilde{x}_{i+1} \in [0, 1]^d$$

where $\alpha = \epsilon/k$ and $\tilde{x} = \tilde{x}_k$

PGD [Madry et al., 2018]. Projected Gradient Descent is similar to BIM but starts from a random perturbation in an L_p ball around x. Auto-PGD [Croce and Hein, 2020b] adds a momentum to the gradient step and adapts the step size α across iterations.

CW [Carlini and Wagner, 2017b]. The Carlini-Wagner attack is one of the strongest attacks available. It considers a loss

$$g(x) = \max(\max_{j \neq t}(Z_j(x)) - Z_t(x), -c)$$

where $c > 0$ is a confidence parameter. Minimizing g finds an image with a high softmax output the target t. So the attack then solves the optimization problem

$$\min_\delta \|\delta\|_p + c\ g(x + \delta) \text{ such that } \tilde{x} = x + \delta \in [0, 1]^d$$

The attack also transforms the problem from a box-constrained one to an unconstrained one by rewriting the perturbation as $\delta = \frac{1}{2}(\tanh(w) + 1) - x$.

DF [Moosavi-Dezfooli et al., 2016]. The DeepFool attack approximates the decision regions of a neural network by polyhedrons, taking inspiration from the case of an affine classifier. The attack then accumulates perturbations that are the shortest projections to the boundary of the polyhedron of the correct label. First the attack computes $w_j = \nabla f_j(x) - \nabla f_y(x)$ and $d_j = f_j(x) - f_y(x)$ for all labels j that are different from the ground truth label y. It then finds the closest boundary thus

$$k = \arg\min_{j \neq y} \frac{|d_j|}{\|w_j\|_2}$$

The perturbation added to x is then

$$\delta = \frac{|d_k|}{\|w_k\|_2^2} w_k$$

This process is repeated starting from $x + \delta$ until the predicted label changes.

Among other white-box attack we mention JSMA (Jacobian Saliency Map Attack) [Papernot et al., 2016b], UAP (Universal Adversarial Perturbation) [Moosavi-Dezfooli et al., 2017b], FA (Feature Adversary) [Sabour et al., 2016], ATN (Adversarial Transformation Networks) [Baluja and Fischer, 2017] and FAB (Fast Adaptive Boundary) [Croce and Hein, 2020a].

E.1.2 Black-box Attacks

Black-box attacks don't have any knowledge about the the network and can only query it. We present below the substitute model family of black-box attacks, a random search attack and an optimization-based attack.

Substitute model attacks [Papernot et al., 2016a, Papernot et al., 2017, Hu and Tan, 2022]. These attacks simply train a network (called a substitute model) that imitates the predictions of the targeted network. The attacker then uses a white-box attack to find an adversarial sample for the substitute model and feeds this sample to the targeted network.

SA [Andriushchenko et al., 2020]. The Square Attack selects via random search local square-shaped ϵ-bounded perturbations at random positions to solve the optimization problem

$$\min_{\tilde{x}} \left(Z_y(\tilde{x}) - \max_{j \neq y} Z_j(\tilde{x}) \cdot Z_y(\tilde{x}) \right) \text{ such that } \tilde{x} \in [0,1]^d \text{ and } \|\tilde{x} - x\|_p \leq \epsilon$$

The attack has an L_2 variant and an L_∞ variant.

ZOO [Chen et al., 2017b]. The Zeroth Order Optimization attack estimates the gradients of the network f using finite differences. Indeed

$$\frac{\mathrm{d}f(x)}{\mathrm{d}x_i} \simeq \frac{f(x + he_i) - f(x - he_i)}{2h}$$

and

$$\frac{\mathrm{d}^2 f(x)}{\mathrm{d}^2 x_i} \simeq \frac{f(x + he_i) - 2f(x) + f(x - he_i)}{h^2}$$

where $h > 0$ is small and e_i is the i-th standard basis vector. In one variant of the attack, these estimations are used to perform stochastic coordinate descent optimization methods [Bertsekas, 2016] to minimize a similar target to that of the Carlini-Wagner attack:

$$g(x) = \max(\max_{j \neq t}(\log f_i(x)) - \log f_t(x), -c)$$

Among other black-box attacks we mention Pixel Attacks [Su et al., 2019, Kotyan and Vargas, 2020], ST (spatial transformation) [Engstrom et al., 2019], HSJ (Hop-Skip-Jump) [Chen et al., 2020a] and BA (Boundary Attack) [Brendel et al., 2018].

E.2 Adversarial Detectors

We describe here how some recent adversarial detectors operate. Generally, an adversarial detector extracts statistics from the activations of an input going through a network, before training a classier on these statistics.

E.2.1 Statistical Approaches

Mahalanobis [Lee et al., 2018]. The Mahalanobis detector uses Mahalanobis distances between the activations and a Gaussian fitted to them during network training as detection features. The means of the Gaussian distributions are fitted for each class c thus

$$\hat{\mu}_{c,m} = \frac{1}{N_c} \sum_{x \in \mathcal{D}_c} h_m(x)$$

where $\mathcal{D}_c$ is the set of samples x with class c, $N_c = \#\mathcal{D}_c$ is its cardinality, and h_m is the embedding function at depth m. The covariance matrix is the same for all classes and is fitted thus

$$\hat{\Sigma}_m = \frac{1}{N} \sum_{(x,y) \in \mathcal{D}} (h_m(x) - \hat{\mu}_{y,m})(h_m(x) - \hat{\mu}_{y,m})^\intercal$$

where $\mathcal{D}$ is the dataset of sample-label pairs, and $N = \#\mathcal{D}$ is the number of samples. The Mahalanobis distances to these Gaussians for an input x are then for every class c

$$\hat{M}_{c,m}(x) = -(h_m(x) - \hat{\mu}_{c,m})^\intercal \hat{\Sigma}^{-1} (h_m(x) - \hat{\mu}_{c,m})$$

The confidence scores for an input x to be classified as adversarial (or out-of-distribution) or not are then

$$\hat{S}_m(x) = \max_c -\hat{M}_{c,m}(x)$$

In practice, the detector (a logistic regression) is trained on Mahalanobis scores $\hat{S}_m(x)$ at five different depths m.

[Ren et al., 2021] propose an improvement over the Mahalanobis distance for out-of-distribution detection called RMD (Relative Mahalanobis Distance). They fit another Gaussian to all training samples, regardless of class. So the estimated mean is now

$$\tilde{\mu}_m = \frac{1}{N} \sum_{(x,y)\in\mathcal{D}} h_m(x)$$

And the estimated covariance is now

$$\tilde{\Sigma}_m = \frac{1}{N} \sum_{(x,y)\in\mathcal{D}} (h_m(x) - \tilde{\mu}_m)(h_m(x) - \tilde{\mu}_m)^\intercal$$

Then they compute the Mahalanobis distance to this Gaussian

$$\tilde{M}_m(x) = -(h_m(x) - \tilde{\mu}_m)^\intercal \tilde{\Sigma}^{-1} (h_m(x) - \tilde{\mu}_m)$$

And define the relative Mahalanobis distance as follows

$$\mathrm{RMD}_{c,m}(x) = \hat{M}_{c,m}(x) - \tilde{M}_m(x)$$

The confidence scores used as features for detection are now

$$\tilde{S}_m(x) = \max_c -\mathrm{RMD}_{c,m}(x)$$

LID [Ma et al., 2018]. Given a point x sampled from a probability distribution P, and a batch $B = \{x_i\}_{i=1}^k$ of points also sampled i.i.d from P, the MLE estimator of the *local intrinsic dimensionality* (LID) of P at point x is given by

$$\hat{L}(x, B) = -\left(\frac{1}{k} \sum_{i=1}^{k} \log \frac{r_i(x)}{r_k(x)} \right)^{-1}$$

where $r_i(x)$ is the distance from x to its i-th nearest neighbor in B. For adversarial detection, the features used are $\hat{L}(a, B)$ where a is an activation at a certain depth from an input (clean or adversarial) going through the network, and B is always a batch of activations of clean samples.

E.2.2 Computer Vision Approaches

NSS [Kherchouche et al., 2020]. The natural scene statistics (NSS) detector uses image quality assessment statistics that detect distortions to detect adversarial samples. It extracts 18 of these statistics from the images themselves and not from their embeddings. We describe here how the first two statistics are extracted.

The mean subtracted contrast normalized (MSCN) coefficients [Mittal et al., 2012] of an image I are given by

$$\hat{I}(i,j) = \frac{I(i,j) - \mu(i,j)}{\phi(i,j) + c}$$

where $c > 0$ is a small constant, (i,j) is a pixel position, the local mean μ is

$$\mu(i,j) = \sum_{k=-3}^{3} \sum_{l=-3}^{3} w(k,l) I(i+k, j+l)$$

and the local standard deviation ϕ is

$$\phi(i,j) = \sqrt{\sum_{k=-3}^{3} \sum_{l=-3}^{3} w(k,l)(I(i+k,j+l) - \mu(i,j))^2}$$

where w is a circularly-symmetric Gaussian weighting function. A generalized Gaussian distribution (GGD) is then fitted to $\hat{I}$ and its shape parameter and variance are used as the first two features for detection.

Fourier detectors [Harder et al., 2021]. The discrete Fourier transform (DFT) in 2 dimensions of an image $I \in [0,1]^{d \times d}$ is given by

$$\mathcal{F}(I)(l,m) = \sum_{j,k=0}^{d} I(j,k) \exp -2\pi i \frac{lj + km}{d}$$

for $l, k \in \{0, .., d-1\}$. The magnitude of the Fourier coefficients is given by

$$|\mathcal{F}(I)(l,m)| = \sqrt{\mathrm{Re}^2(\mathcal{F}(I)(l,m)) + \mathrm{Im}^2(\mathcal{F}(I)(l,m))}$$

And the phase is given by

$$\phi(\mathcal{F}(I)(l,m)) = \arctan \frac{\mathrm{Re}(\mathcal{F}(I)(l,m))}{\mathrm{Im}(\mathcal{F}(I)(l,m))}$$

[Harder et al., 2021] propose two detectors. This first is called MFS (magnitude Fourier spectrum) and uses the magnitudes of the Fourier coefficients as detection features. The second is called PFS (phase Fourier spectrum) and uses the phases of the Fourier coefficients as detection features. They find it is better to extract these features from the embeddings of the images at certain layers than from the images themselves. But given the high dimensionality of these embeddings, they have to heuristically select a subset of layers to consider.

F Additional Experiments from Section 3

F.1 Implementation Details

These are further implementation details for the experiments in Section 3.5. Orthogonal initialization with gain 0.01 is used for ResNet models. Kaiming initialization is used for ResNeXt models. SGD is used for training. The momentum is always set to 0.9 and weight decay to 0.0001. For ResNet models, the learning rate is 0.01 and is divided by 5 at epochs 120, 160 and 200 (when the training goes that far). For ResNeXt models, the learning rate is 0.1 and is divided by 10 at epochs 150, 225 and 250. Batch size is 128 for all experiments. Architectures of ResNet [He et al., 2016a] and ResNeXt [Xie et al., 2017] blocks are standard and exactly as in the references. The ResNets used are single representation ResNets (i.e. containing one residual stage) with 9 blocks. The ResNeXt architecture used is the ResNeXt-50-32×4d from [Xie et al., 2017].

F.2 Transport Visualization

If we pretrain an autoencoder on MNIST, we can use its encoder as the encoder of the ResNet and freeze it during training. This makes it possible to visualize the transport of the data by decoding, via the pretrained decoder, the output of each residual block (see Figure 28 below).

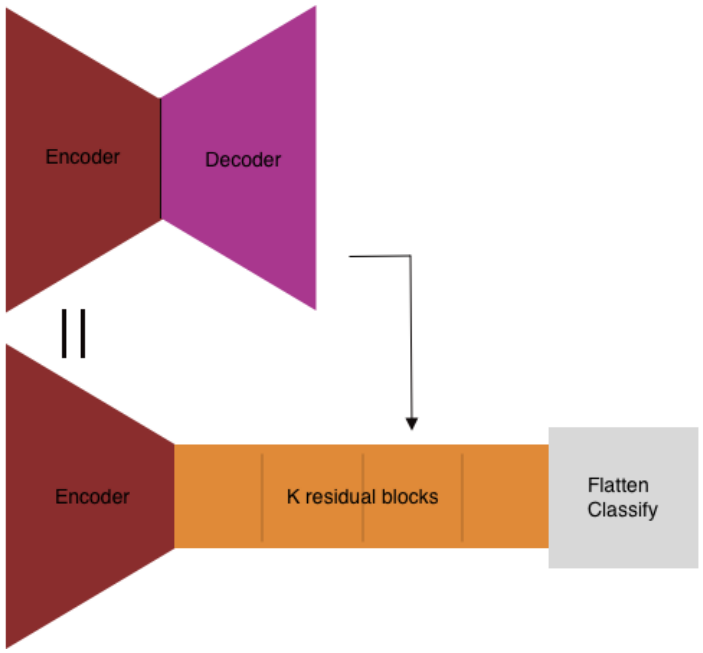

Figure 28: Decoding the internal representations of a network.

We show this below on MNIST. In Figure 29, we see the decodings of a vanilla ResNet trained to achieve 99.4% test accuracy.

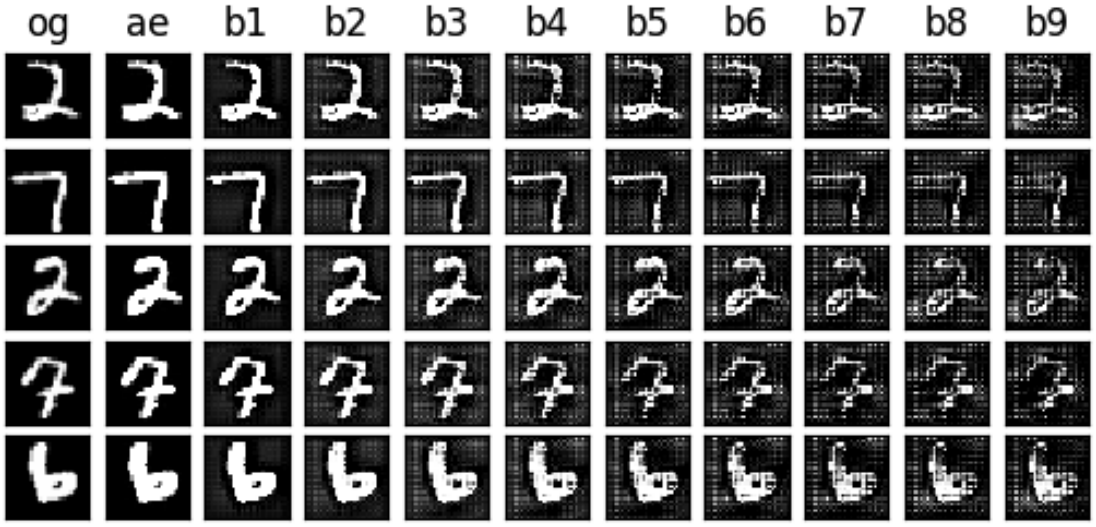

Figure 29: Decodings of the internal representations (outputs of the ResBlocks) after training a ResNet-9 on MNIST (og: original image, ae: encoding, b1: output of block 1...)

We add the transport cost with λ_0=5, τ=1 and s=5. The accuracy barely drops (99.3%) and we see in Figure 30 that the decodings change much less.

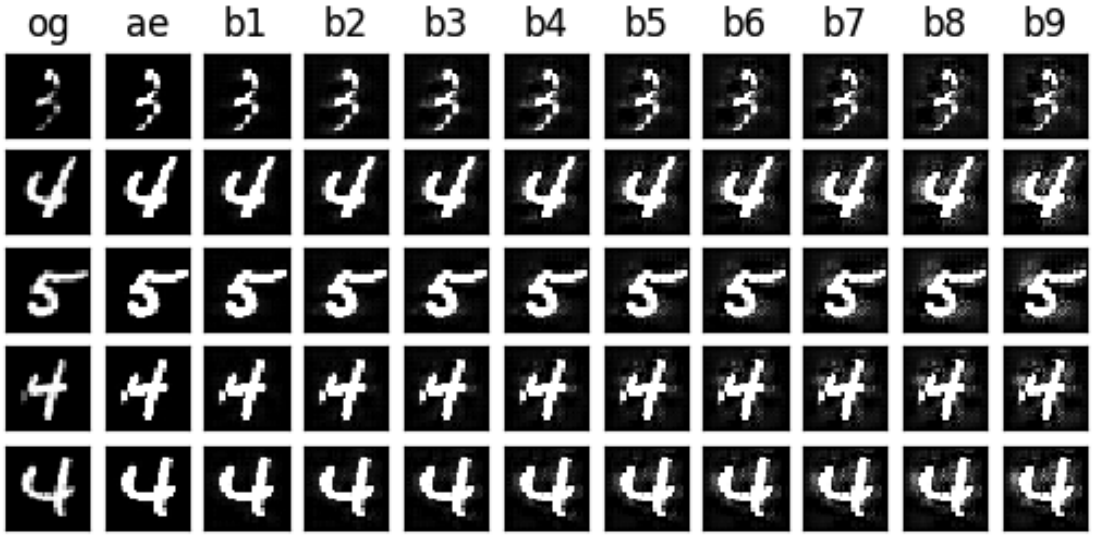

Figure 30: Decodings of the internal representations (outputs of the ResBlocks) after training a LAP-ResNet-9 on MNIST (og: original image, ae: encoding, b1: output of block 1...)

We do the same thing on CIFAR10, first with a vanilla ResNet-9 in Figure 31.

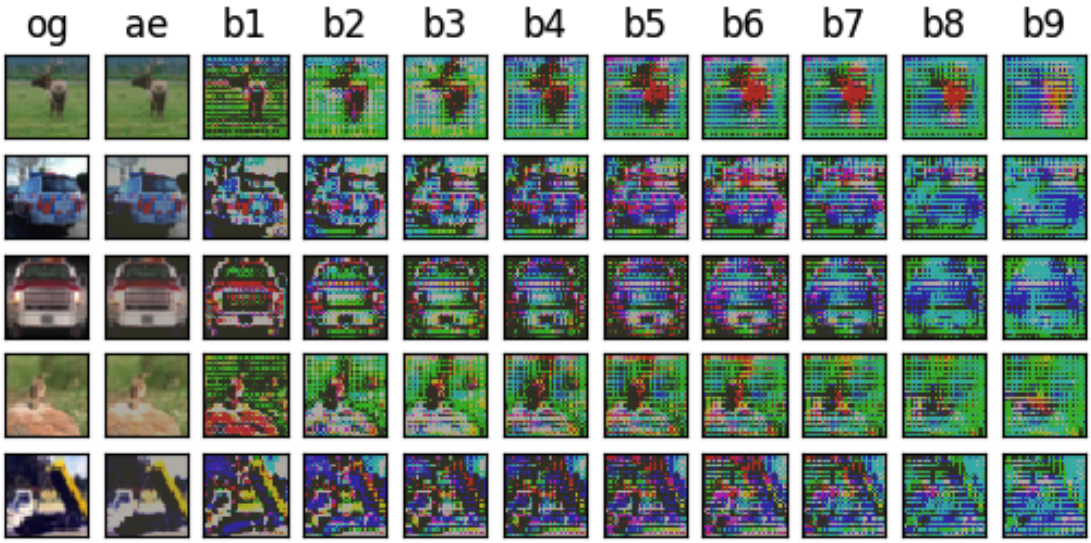

Figure 31: Decodings of the internal representations (outputs of the ResBlocks) after training a ResNet-9 on CIFAR10 (og: original image, ae: encoding, b1: output of block 1...)

Then with LAP training with λ_0=10, τ=0.1 and s=50 in Figure 32 below.

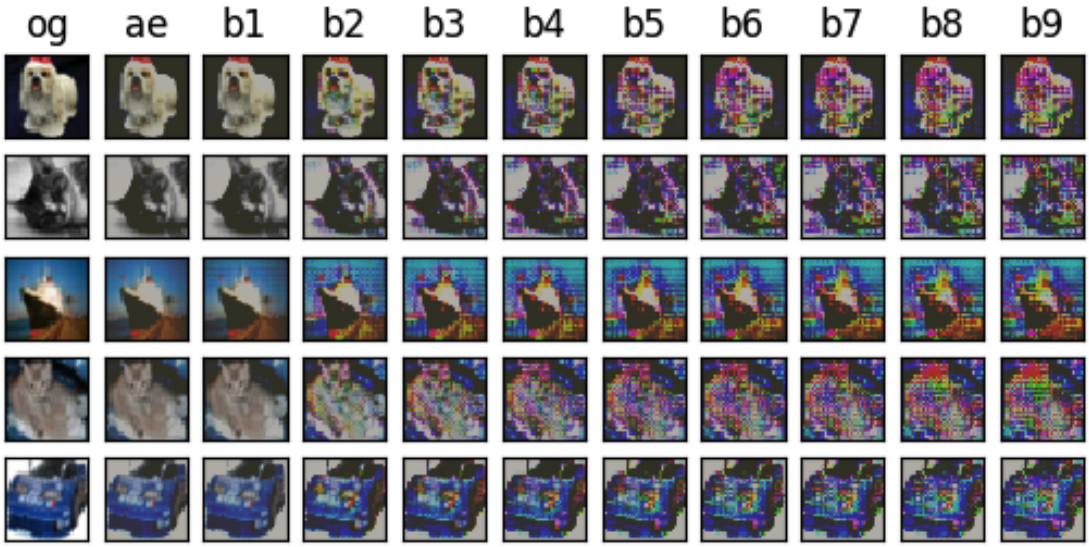

Figure 32: Decodings of the internal representations (outputs of the ResBlocks) after training a LAP-ResNet-9 on CIFAR10 (og: original image, ae: encoding, b1: output of block 1...)

Finally, on CIFAR100, the decodings without regularization are in Figure 33.

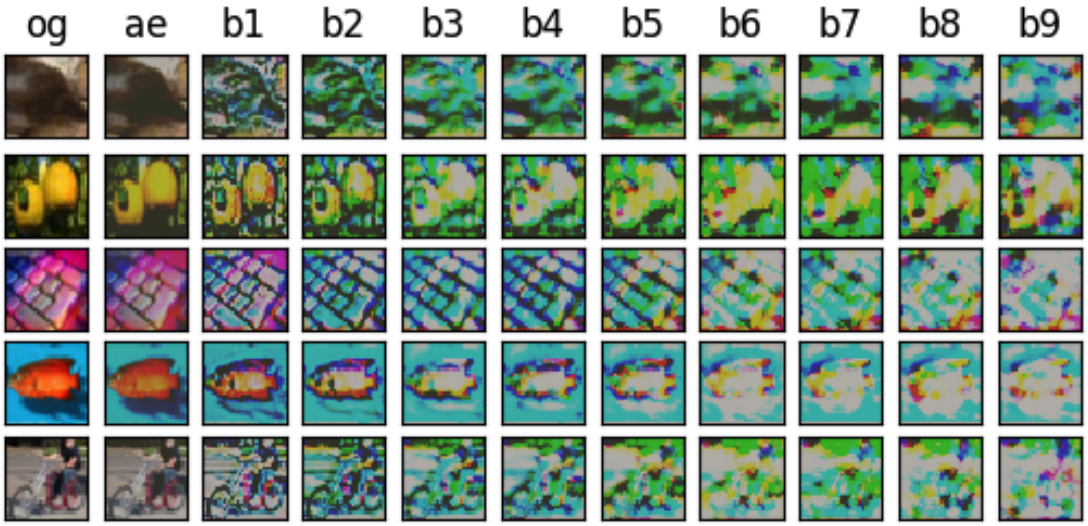

Figure 33: Decodings of the internal representations (outputs of the ResBlocks) after training a ResNet-9 on CIFAR100 (og: original image, ae: encoding, b1: output of block 1...)

With the transport regularization, the decodings are in Figure 34.

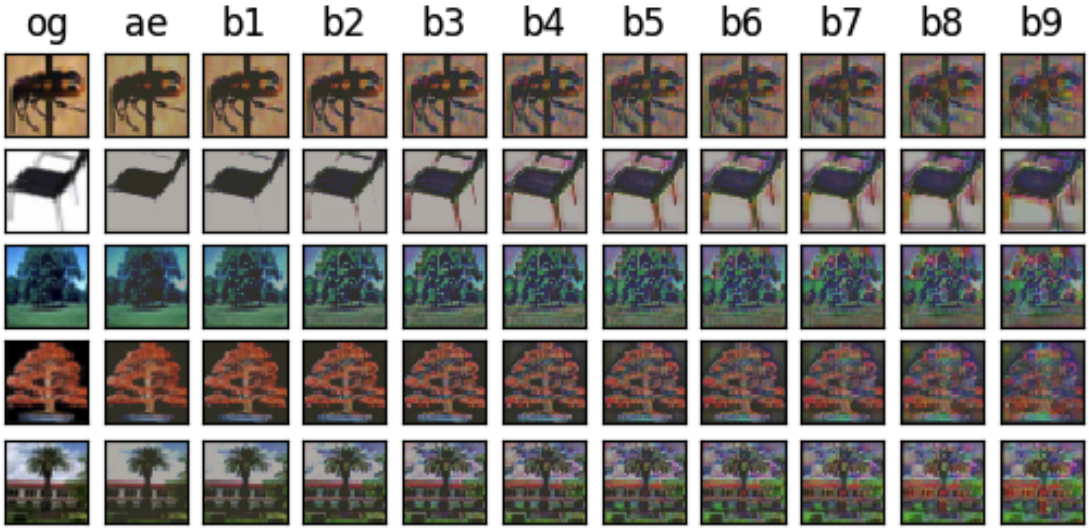

Figure 34: Decodings of the internal representations (outputs of the ResBlocks) after training a LAP-ResNet-9 on CIFAR100 (og: original image, ae: encoding, b1: output of block 1...)

F.3 Experiments with Fixed λ

In this section, we show some additional experimental results with a model where, instead of using an adaptive optimization algorithm, we simply take the transport cost as a regularizer, thus giving us a minimization objective:

$$\mathcal{L}(\theta) + \lambda \;\; \mathcal{C}(\theta)$$

This is an easier and more straightforward method, simply considering a relaxed constraint in the optimization problem (18). Aside from the advantage of simpler implementation, it allows for easier fine-tuning of the regularization hyper-parameter which is useful when the datasets and networks are big. The adaptivity is lost but this still leads to better test accuracy than the non-regularized networks. Results on the same tasks as in Section 3.5 are below.

Table 16: Average highest test accuracy and 95% confidence interval of ResNeXt50 over 10 instances on CIFAR100 with training sets of different sizes (in %)

Train size	ResNeXt	LAP-ResNeXt50	ResNeXt50 with $\lambda = 0.01$
50 000	72.97 ± 1.18	**76.11** ± 0.78	75.96 ± 1.04
25 000	62.55 ± 2.37	**64.11** ± 1.85	64.10 ± 1.74
12 500	45.90 ± 2.74	**48.23** ± 1.84	47.77 ± 1.85

Table 17: Average highest test accuracy and 95% confidence interval of ResNet-9 over 10 instances on CIFAR100 with training sets of different sizes (in %)

Train size	ResNet	LAP-ResNet	ResNet with $\lambda \in \{0.05, 0.2\}$
50 000	72.32 ± 0.24	72.43 ± 0.18	**72.62** ± 0.21
25 000	64.34 ± 0.23	64.34 ± 0.23	**64.76** ± 0.24
10 000	49.27 ± 0.42	**50.57** ± 0.23	50.46 ± 0.27
5 000	34.74 ± 0.84	37.97 ± 0.30	**38.44** ± 0.45
1 000	15.66 ± 0.43	**16.42** ± 0.32	16.03 ± 0.49

Table 18: Average highest test accuracy and 95% confidence interval of ResNet-9 over 20 instances on CIFAR10 with training sets of different sizes (in %)

Train size	ResNet	LAP-ResNet	ResNet with $\lambda = 0.2$
50 000	91.49 ± 0.09	**91.94** ± 0.10	91.36 ± 0.08
30 000	88.61 ± 0.14	**89.41** ± 0.09	88.50 ± 0.11
20 000	85.73 ± 0.14	**86.74** ± 0.13	85.82 ± 0.11
10 000	79.25 ± 0.25	**80.90** ± 0.16	80.15 ± 0.13
5 000	70.32 ± 0.32	**72.58** ± 0.12	72.03 ± 0.31
4 000	67.80 ± 0.26	**70.12** ± 0.30	69.64 ± 0.30
1 000	49.22 ± 0.52	**51.14** ± 0.45	50.38 ± 0.45
500	41.55 ± 0.41	**42.92** ± 0.37	42.30 ± 0.43
100	26.98 ± 0.99	25.34 ± 0.76	**27.53** ± 0.94

F.4 Impact on Weight Distribution of Classifier

LAP training moves the weights of the classification layers away from zero: compare Figure 35 for a vanilla network and Figure 36 for a LAP network.

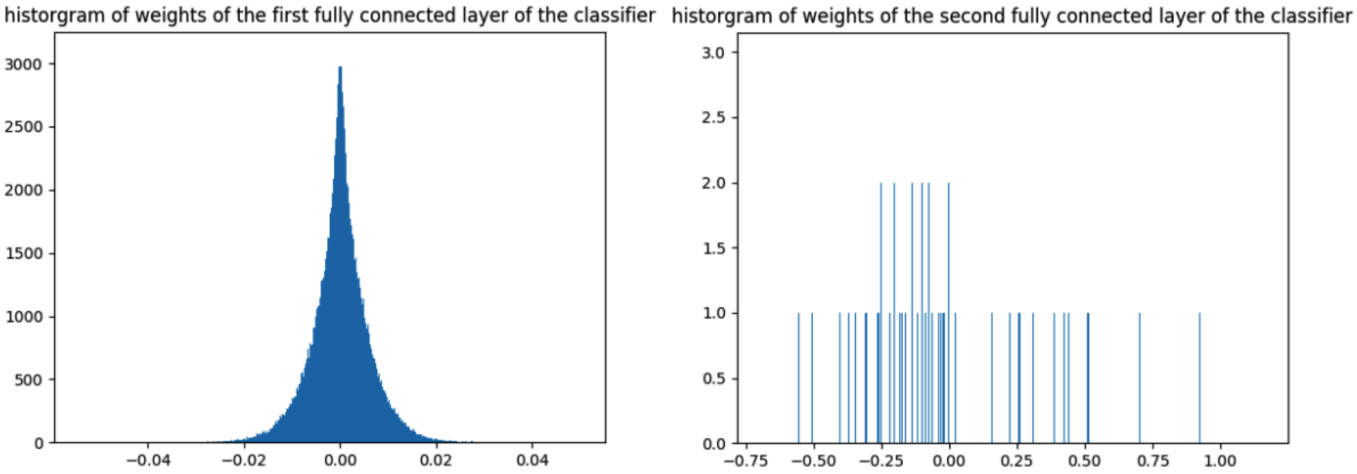

Figure 35: Weight distribution of the two fully connected layers of the classification head of a vanilla ResNet-9 trained on CIFAR10.

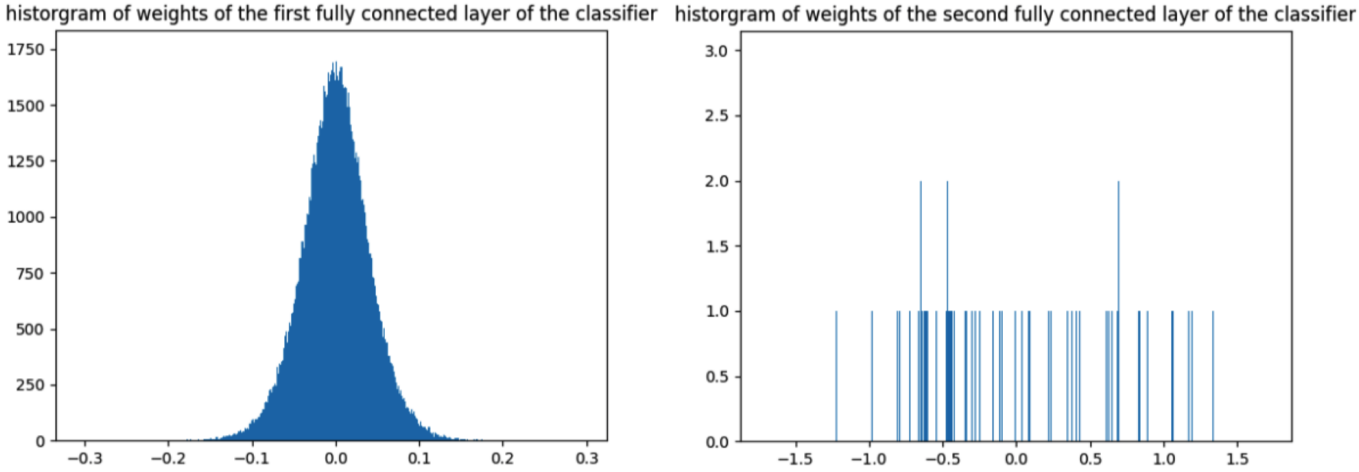

Figure 36: Weight distribution of the two fully connected layers of the classification head of a LAP-ResNet-9 trained on CIFAR10.

F.5 Additional Experiments in 2 Dimensions

F.5.1 Comparison to Exact Wasserstein Distances

In 2 dimensions we can compute the exact W_2 distance between successive point clouds in the network before and after each block, and compare it, in Figure 37 below, to the displacement that the block causes. This confirms that residual networks are indeed already close to being optimal transport maps, and that LAP training encourages this bias.

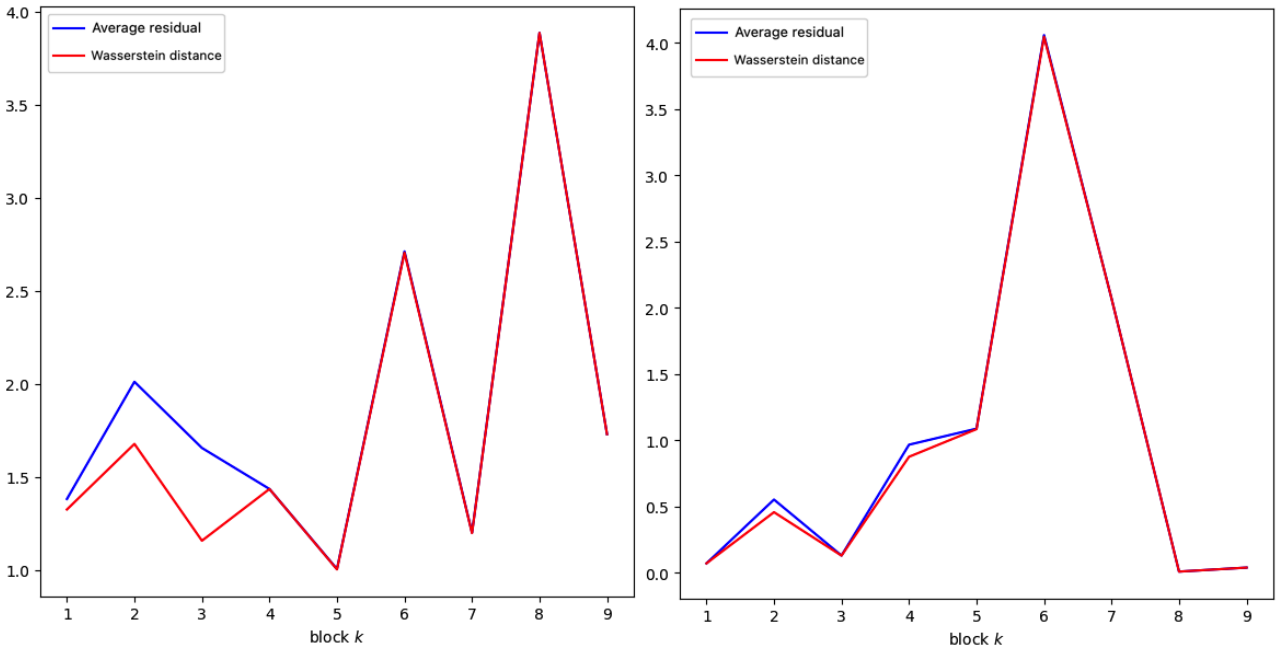

Figure 37: Average residual function norm and W_2 distance between successive point clouds in a ResNet-9 trained on the circles test set. Vanilla training on the left and LAP training on the right.

F.5.2 Comparison to Batch Normalization

We compare our method with batch normalization. We find that the two cooperate well to improve test accuracy on the same 2-dimensional task as in Section 3.2 when the model is too small (1 block, Table 19), too big (100 blocks, Table 20), poorly initialized ($\mathcal{N}(0, 5)$ initialization, Table 22) and when the data set is small (50 points, Table 21). LAP-ResNets use $\lambda_0 = 0.1$, $\tau = 0.1$ and $s = 5$.

We are interested in cases where adding the transport cost to the loss helps with the generalization of the network. The advantage of Uzawa's algorithm is that fixing $\lambda_0 = 0.1$, $\tau = 0.1$ and $s = 5$ works well in most cases. A fixed λ requires more tinkering, but when a good value is found, it often performs

better than Uzawa's algorithm with those values, and finding better values for Uzawa requires a grid search over three parameters instead of just one. Since the results are similar for both the circles and the moons dataset, we report below the results on the circles dataset only.

A first case where adding $\mathcal{C}$ to the loss helps is when the network is small. If the network has only two blocks, then it takes on average around 73 epochs to achieve 100% test accuracy and keep that performance for three consecutive epochs (on average over 100 instances of the problem, an instance being a randomly generated circles dataset and a random initialization of the model's weights). If we add batch normalization to the residual blocks as in Figure 1, then the average number of epochs is around 71. But if we add the transport cost $\mathcal{C}$ with fixed weight $\lambda = 0.01$, then the average number of epochs to achieve the same performance drops to 54. Adding both batch normalization and the transport cost at the same time does not help, as the average number of epochs goes up to 64.

If the network has only one block (Table 19), then it never reaches 100% test accuracy. The highest test accuracy it reaches after 100 epochs is on average 76.6%. With batch normalization, this figure drops to 75.4%, but with Uzawa's algorithm, it goes up to 82.1%. With both batch normalization and Uzawa, the average best accuracy is even higher at 84.6%.

Table 19: Average test accuracy and 95% confidence interval over 100 instances on the circles 2D dataset with 1000 points and 1 block (in %)

	No batch normalization	Batch normalization
ResNet	76.6 ± 3.5	75.4 ± 3.1
ResNet with $\lambda = 0.005$	76.5 ± 3.5	75.6, ± 3.3
LAP-ResNet	82.1 ± 2.6	**84.6** ± 3.1

A second case is when the network is too big (Table 20). The network with 100 blocks and no batch normalization has an average test performance of only 89.1%. As expected, adding batch normalization helps greatly as the average test accuracy rises to 99.4%. This performance is improved slightly by adding a transport cost via Uzawa's algorithm and reaches 99.8%. Note that adding the transport cost without batch normalization hinders the test performance in this case.

Table 20: Average test accuracy and 95% confidence interval over 100 instances on the circles 2D dataset with 1000 points and 100 blocks (in %)

	No batch normalization	Batch normalization
ResNet	89.1 ± 0.9	99.4 ± 0.4
ResNet with $\lambda = 0.09$	69.7 ± 4.0	99.5 ± 0.6
LAP-ResNet	75.7 ± 2.9	**99.8** ± 0.2

A third instance of the transport cost improving generalization is when we use a big initialization (Table 22). The network has 9 blocks and we initialize the weights and biases of the fully connected layers by drawing from a centered normal distribution with standard deviation equal to 5. Here, batch normalization is essential for the network to recover from this initialization and get close to 100% test accuracy inside 100 epochs. If we further add a transport cost with $\lambda = 0.04$, the average highest test accuracy improves from 97.8% to 99.8%. If we add a transport cost via Uzawa's algorithm instead of a fixed weight, we get an average test accuracy of 99.6%. An example of the behavior we want to avoid with a big initialization is in Figure 39 below. With batch normalization (and the same random seeds), we get the behavior in Figure 40. If we also add $\lambda = 0.04$, we gain further control as seen in Figure 41.

Table 21: Average test accuracy and 95% confidence interval over 100 instances on the circles 2D dataset with 50 points and 9 blocks (in %)

	No batch normalization	Batch normalization
ResNet	88.2 ± 2.3	92.9 ± 2.0
ResNet with $\lambda = 0.04$	93.5 ± 2.1	94.4 ± 2
LAP-ResNet	95.8 ± 1.8	**96.0** ± 1.3

A final case is when the training set is small (Table 21). Here we consider a dataset containing 50 points (40 train and 10 test) and a network with 9 blocks. The test performance of this network is on average 88.2%. We can get to 92.7% average test accuracy with batch normalization but using Uzawa's algorithm and no batch normalization yields the best average performance at 95.7%. Adding batch normalization does not change this.

Table 22: Average test accuracy and 95% confidence interval over 100 instances on the circles 2D dataset with a $\mathcal{N}(0, 5)$ initialization (in %)

	No batch normalization	Batch normalization
ResNet	90.2 ± 1.3	98.0 ± 0.8
ResNet with $\lambda = 0.04$	89.7 ± 1.5	**99.7** ± 0.2
LAP-ResNet	79.1 ± 3.9	99.4 ± 0.4

F.5.3 Additional Transport Visualizations in 2 Dimensions

We provide here further visualizations of transport dynamics in 2 dimensions inside a ResNet-9, as in Section 3.2.2, but now showing point clouds after every block. In Figure 38, we show the normal behavior of a network trained with transport regularization (to be compared with the first row in Figure 12 in Section 3.2.2). The three following figures show how transport regularization and batch normalization help control the behavior of a badly initialized network.

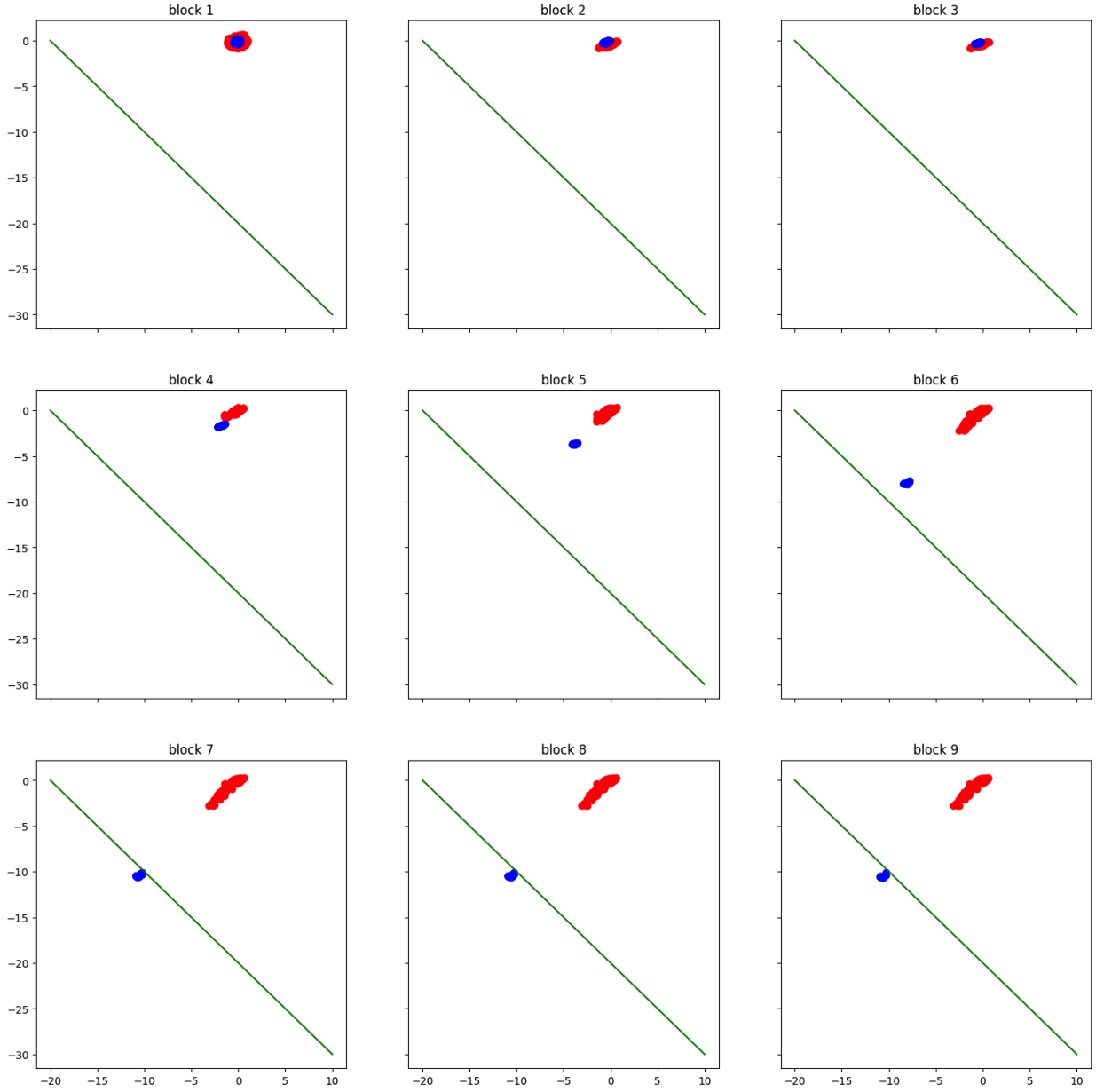

Figure 38: Circles test set after each block after training with transport regularization with weight $\lambda = 0.07$.

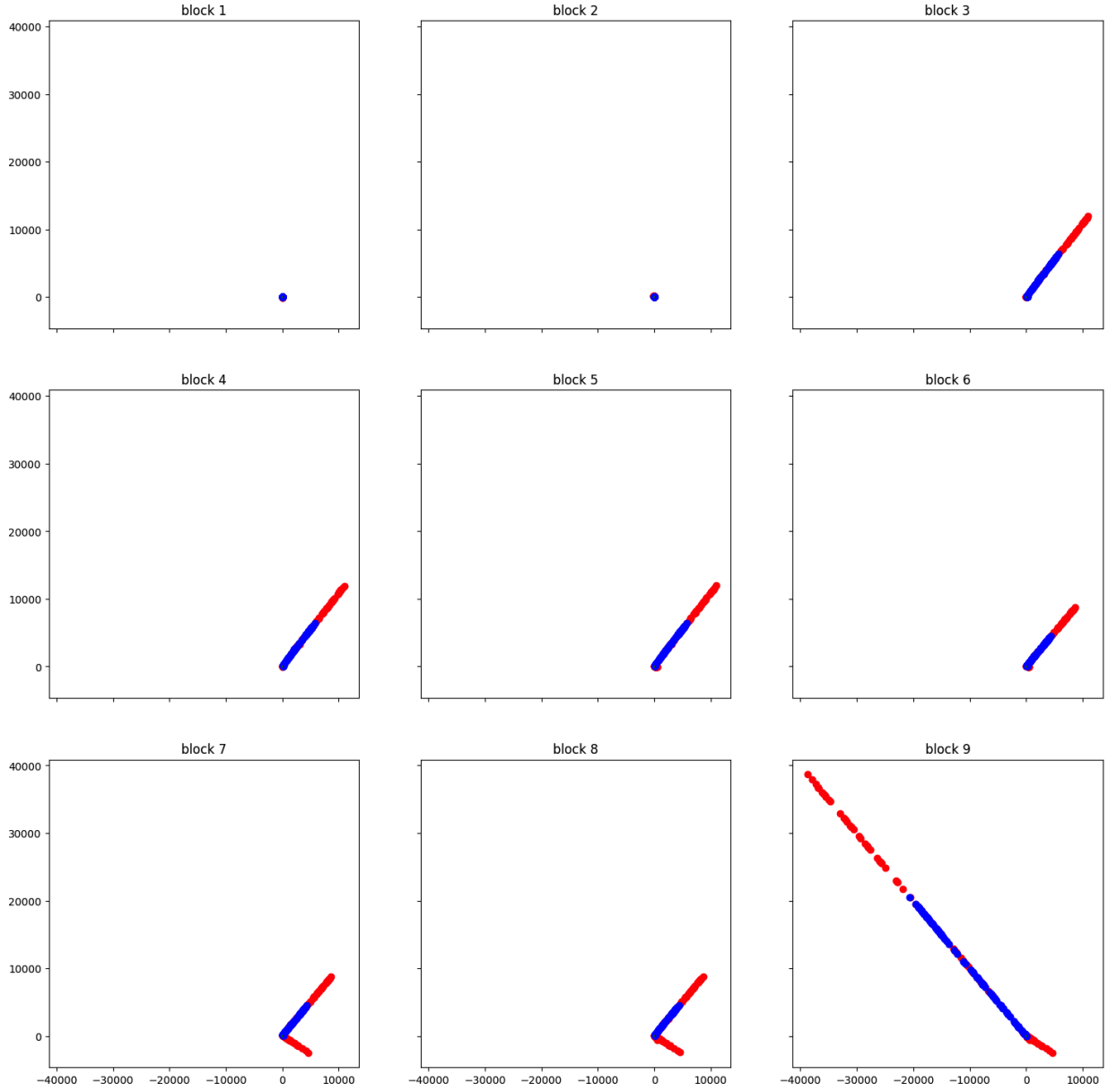

Figure 39: Circles test set after each block after training with $\mathcal{N}(0, 5)$ initialization.

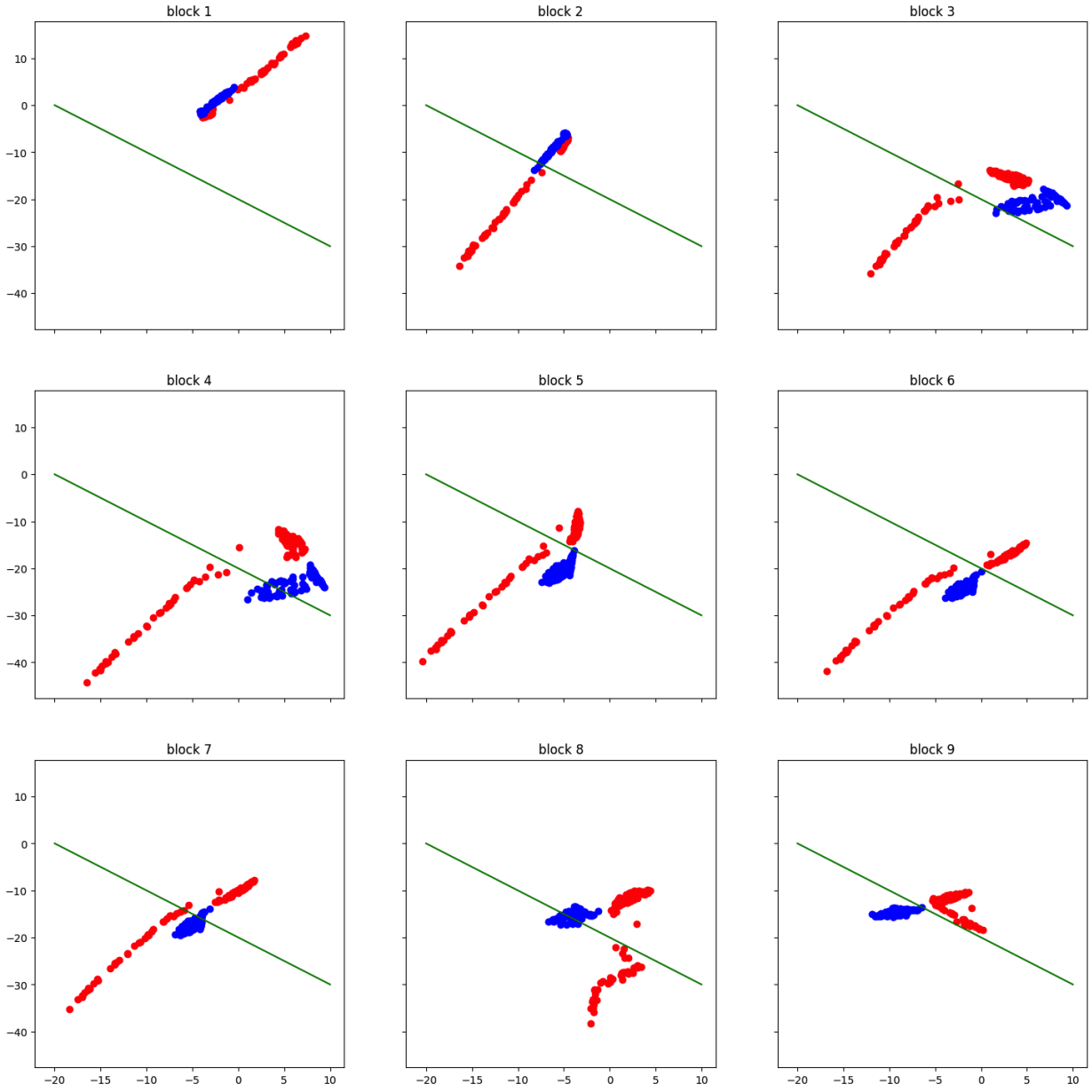

Figure 40: Circles test set after each block after training with $\mathcal{N}(0, 5)$ initialization and batch normalization.

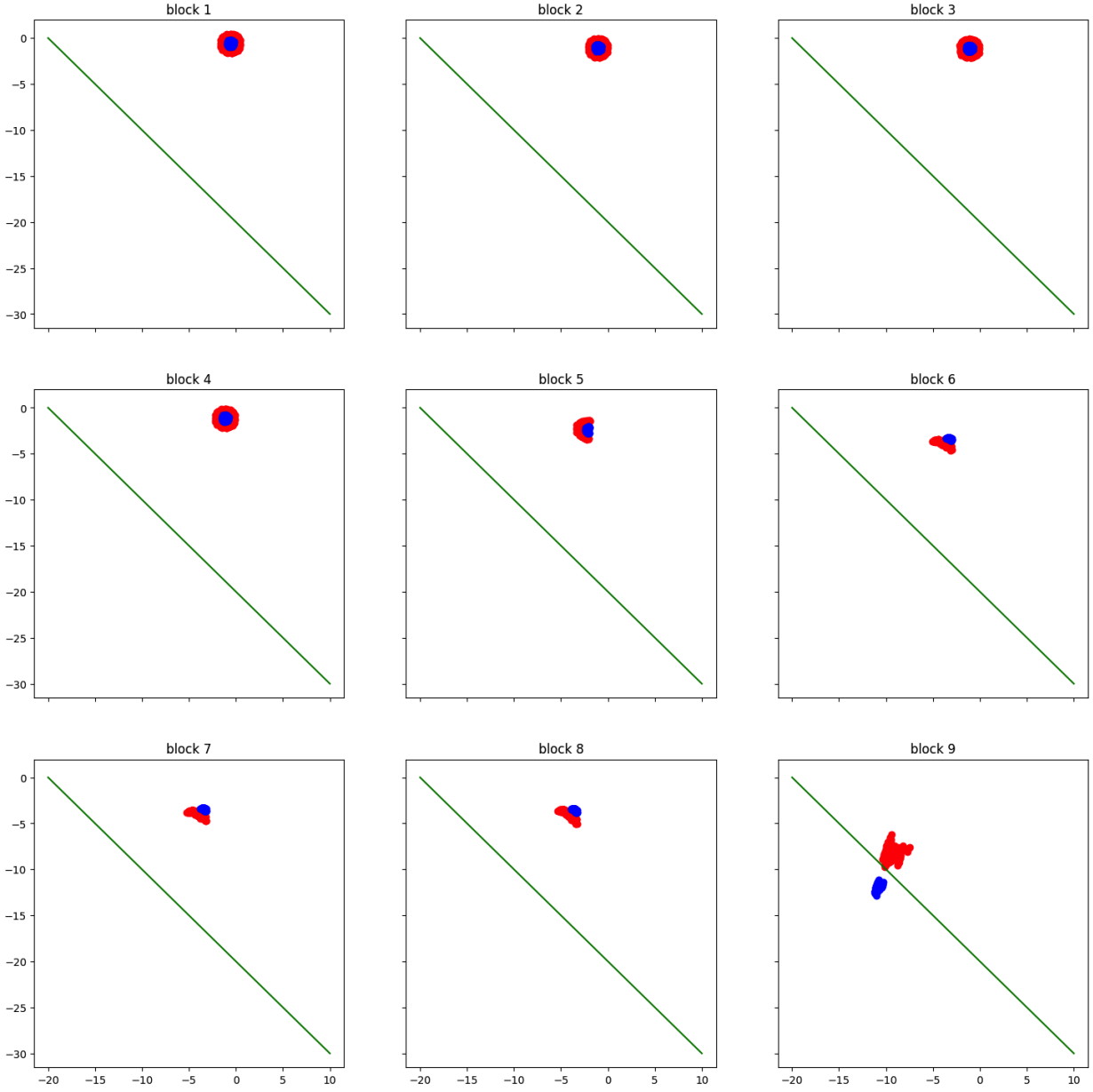

Figure 41: Circles test set after each block after training with $\mathcal{N}(0, 5)$ initialization, batch normalization and transport regularization with fixed weight $\lambda = 0.04$.

G Additional Experiments from Section 4

G.1 Implementation Details

We use standard data augmentation and standard implementations for VGG-19, ResNet-50, ResNet-101, ResNet-110, ResNet-152 and Swin-Tiny Transformer (the same as for the other methods). We use NVIDIA Tesla V100 16GB GPUs for the experiments.

Training a Resnet-152 on TinyImageNet in Table 4 takes about 36 hours. Training a Resnet-152 on CIFAR100 in Table 4 takes about 11 hours. Training a ResNet-110 on STL10 in Table 6 takes about 3 hours. Training a Swin-Tiny Transformer in Table 7 take between 40 minutes and 1 hour.

For sequential and multi-lap sequential training, we use SGD with a learning rate of 0.007. With the exception of the Swin Transformer in Table 7, we use SGD for parallel training with learning rate of 0.003 in all Tables but Table 6 where the learning rate is 0.002. For the Swin Transformer in Table 7, we use the AdamW optimizer with a learning rate of 0.007 and a CosineLR scheduler.

For end-to-end training we use a learning rate of 0.1 that is divided by five at epochs 120, 160 and 200. Momentum is always 0.9. For parallel and end-to-end training, we train for 300 epochs. For sequential and multi-lap sequential training, the number of epochs varies per module (see Section 4.5.5).

For experiments in Section 4.5.1, we use a batch size of 256, orthogonal initialization [Saxe et al., 2014] with a gain of 0.1, label smoothing of 0.1 and weight decay of 0.0002. Only for Table 6, the batch size changes to 64.

For experiments in Section 4.5.5, we use a batch size of 128, orthogonal initialization with a gain of 0.05, no label smoothing and weight decay of 0.0001.

For the experiments on TinyImageNet in Table 4, we use $\tau = 500000$ for the first two modules and then double it for the last two modules for TRGL.

For the experiments on CIFAR100 in Table 5, we use $\lambda_{k,1} = 1, h = 1$ and $s = 50$ for TRGL.

For the experiments on STL10 in Table 6, we use $\tau = 0.5$ and double it at the midpoint, expect for the first line where $\tau = 50$.

G.2 Additional Experiments

Table 23: Test accuracy of 2-7 ResNet on CIFAR100 with parallel TRGL and VanGL, compared to DDG and FR from [Huo et al., 2018a] that also split their networks in 2 module-wise-parallel-trained modules.

Parallel VanGL (ours)	Parallel TRGL (ours)	DDG ResNet-101	FR ResNet-101
76.26	**76.96**	75.75	76.90

Table 24: Test accuracy of parallel TRGL and VanGL with 2 modules, compared to DGL and DDG from [Belilovsky et al., 2020] and InfoPro from [Wang et al., 2021] that also split a ResNet-110 in 2 module-wise-parallel-trained modules, on CIFAR 10.

Parallel VanGL (ours)	Parallel TRGL (ours)	DGL	DDG	InfoPro
94.01 ± .17	**94.05** ± .18	93.50	93.41	93.58

Table 25: Average last block test accuracy and 95% confidence interval of 20-1 ResNet (32 filters, fixed encoder, same classifier) over 20/50 runs on MNIST with different train sizes and parallel (Par) TRGL and VanGL, compared to E2E.

Train size	Par VanGL	Par TRGL	E2E
60000	99.07 ± .04	**99.08** ± .04	99.30 ± .03
30000	98.90 ± .05	**98.93** ± .06	99.22 ± .03
12000	98.52 ± .06	**98.59** ± .06	98.96 ± .06
6000	98.05 ± .09	**98.16** ± .07	98.62 ± .06
1500	96.34 ± .12	**96.91** ± .07	97.19 ± .08
1200	95.80 ± .12	**96.58** ± .09	96.88 ± .09
600	91.35 ± .99	**95.16** ± .15	95.30 ± .17
300	89.81 ± .73	**92.86** ± .24	92.87 ± .28
150	81.84 ± 1.22	**87.48** ± .42	87.82 ± .59

Table 26: Average last block test accuracy and 95% confidence interval of 20-1 ResNet (100 filters, fixed encoder, same classifier) over 10 runs on CIFAR10 with different train sizes and parallel (Par) TRGL and VanGL, compared to E2E.

Train size	Par VanGL	Par TRGL	E2E
50000	85.98 ± .28	**86.02** ± .26	93.11 ± .19
25000	80.94 ± .25	**81.09** ± .32	89.10 ± .29
10000	72.49 ± .46	**73.01** ± .31	80.52 ± .46
5000	62.31 ± .54	**64.06** ± .57	69.44 ± .88
500	38.61 ± .47	**41.44** ± .44	40.40 ± .60

Figures 18 and 42 suggest that our regularization helps most when looking at the accuracy of the last block. We confirm this by including in Table 27 the accuracy achieved by the best block for the same experiment as in Table 11 and we notice a more important improvement from the regularization in the accuracy of the last block than in the accuracy of the best block. We also observe that with the regularization the difference between the accuracy of the last block and that of the best block is smaller than without the regularization. We further confirm this through the following experiment. As the network gets deeper (50 blocks trained for 10 epochs each sequentially), we expect training it block-wise to become more difficult, and indeed the improvement from the regularization is slightly larger than usual when looking at the accuracy of the last block for both sequential training methods (Table 28). We also include in Table 29 results from block-wise training of ResNeXt-50-32×4d [Xie et al., 2017], which turns out to be difficult to train block-wise.

Table 27: Average highest test accuracy and 95% confidence interval of 10-1 ResNet over 10 runs on CIFAR10 with different train sizes and sequential (Seq) TRGL and VanGL, compared to E2E.

Train	Seq VanGL	Seq TRGL	E2E
50000	88.14 ± .14	**88.34** ± .22	91.88 ± .18
25000	84.15 ± .17	**84.46** ± .22	88.75 ± .27
10000	76.62 ± .40	**77.47** ± .35	82.61 ± .35
5000	69.60 ± .43	**70.22** ± .50	73.93 ± .67
1000	51.59 ± .91	**52.06** ± .71	50.63 ± .98

Table 28: Average last block test accuracy and 95% confidence interval of 50-1 ResNet over 10 runs on CIFAR100 with sequential (Seq) and multi-lap sequential (MLS) TRGL and VanGL, compared to E2E.

Seq VanGL	Seq TRGL	MLS VanGL	MLS TRGL	E2E
63.40 ± .46	**63.86** ± .56	62.59 ± .64	**63.24** ± .50	63.34 ± 2.41

Table 29: Average test accuracy and 95% confidence interval of ResNeXt50 over 10 runs on CIFAR100 with sequential (Seq), multi-lap sequential (MLS) and parallel (Par) TRGL and VanGL. End-to-end training in this setting achieves an accuracy of 72.97 ± 1.18.

Seq VanGL	Seq TRGL	MLS VanGL	MLS TRGL	Par VanGL	Par TRGL
52.29 ± .53	**52.42** ± .65	52.59 ± .63	**52.84** ± .65	57.86 ± .49	**57.93** ± .51

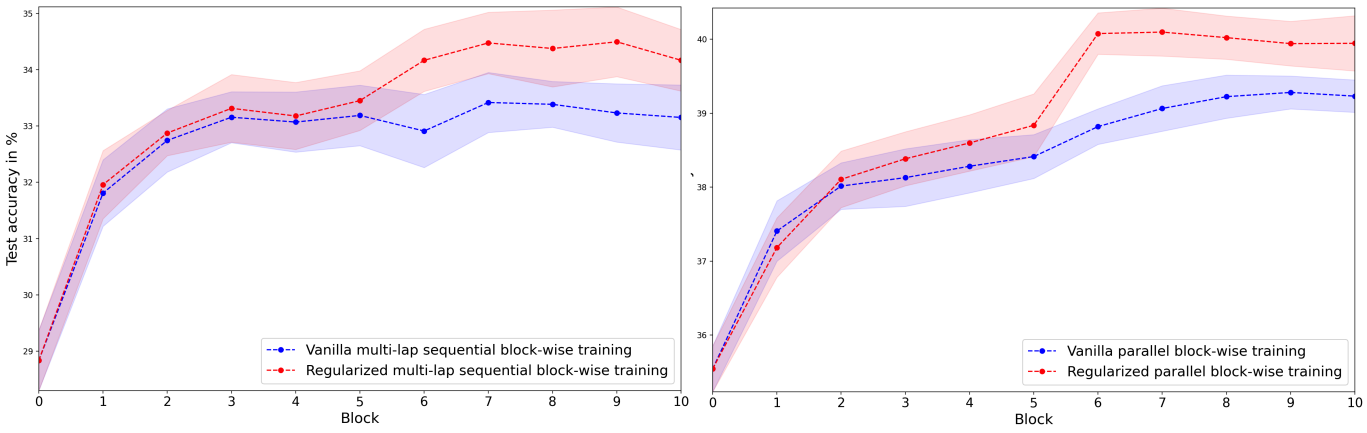

Figure 42: Test accuracy after each block of 10-1 ResNet averaged over 10 runs with 95% confidence intervals. Left: multi-lap sequential vanilla (VanGL, in blue) and regularized (TRGL, in red) block-wise training on 10% of the CIFAR100 training set. Right: parallel vanilla (VanGL, in blue) and regularized (TRGL, in red) block-wise training on 10% of CIFAR100 training set.

G.3 Memory Usage

In Table 30 below, we look at the accuracy of a ResNet110 split into K modules that are equally heavy instead of equally deep, which consumes even less memory (see Table 9 in Section 4.5.3). The module-wise training methods are then called VanGL*, TRGL* and InfoPro* as in [Wang et al., 2021]. We notice a slight degradation in the performance of module-wise training compared to Table 6, but the benefit of the regularization increases, as we now gain almost two percentage points of accuracy when using 4 modules, and still outperform end-to-end training.

Table 30: Test accuracy of parallel (Par) TRGL* with K modules (average and 95% confidence interval over 5 runs) on STL10, compared to InfoPro* and E2E training from Table 3 in [Wang et al., 2021]

Architecture	K	Par VanGL*	Par TRGL* (ours)	InfoPro*	E2E
ResNet-110	2	79.05 ± 1.33	**79.47** ± 1.36	79.05 ± 0.57	77.73 ± 1.61
	4	77.14 ± 1.23	**78.94** ± 1.13	78.78 ± 0.72	77.73 ± 1.61

In Table 31 below, we report the memory saved (as a percentage of the memory required for end-to-end training) for the four networks trained module-wise in parallel on TinyImageNet with 4 modules in Table 4, and the Swin-Tiny Transformer [Liu et al., 2021] trained on CIFAR10 with a batch size of 1024, also split into 4 modules trained in parallel, where the regularization increases the test accuracy from 90.96 to 93.2% (average of 5 runs).

Table 31: Memory savings on TinyImageNet (4 first rows) and CIFAR10 (last row) of networks split into 4 modules trained in parallel with a batch size of 256 (4 first rows) and 1024 (last row), as a percentage of the weight of end-to-end training.

Architecture	Par VanGL	Par TRGL
VGG-19	27%	21%
ResNet-50	26%	20%
ResNet-101	24%	11%
ResNet-152	21%	8%
Swin-Tiny	32%	30%

Note that methods DDG [Huo et al., 2018b] and FR [Huo et al., 2018a], being delayed gradient methods and not module-wise training methods, do no save memory (they actually increase memory usage, see FR [Huo et al., 2018a]). Sedona [Pyeon et al., 2021] also does not claim to save memory, as their first module (the heaviest) is deeper than the others, but rather to speed up computation. Finally, DGL [Belilovsky et al., 2020] is architecture-wise essentially identical to VanGL and consumes the same amount of memory.

G.4 Sensitivity to Hyperparameter τ

We show in Figure 43 below that TRGL still performs better than VanGL (in the same setting as in Table 6 in Section 4.5.1, i.e. using a ResNet110 trained on STL10 with 16 modules trained in parallel) for values of τ from 0.03 to 100 and is still roughly equivalent to it for values up to 5000.

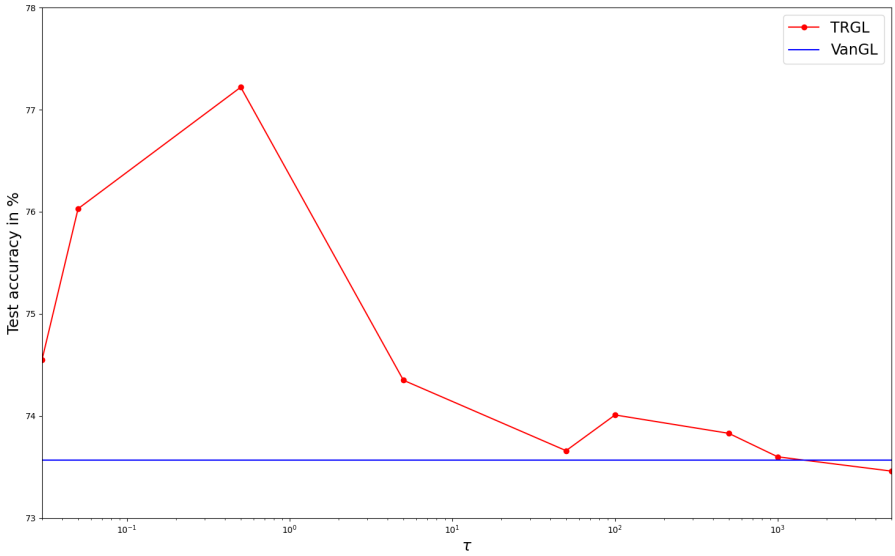

Figure 43: Average test accuracy over 5 runs of parallel TRGL using a ResNet110 on STL10 with 16 modules with different values of τ (in red), and of VanGL (blue line).

H Additional Experiments from Section 5

In the tables below, VAN corresponds to detectors trained on a vanilla network, RCE to detectors trained on a network trained with reverse cross entropy and LAP to detectors trained on a transport-regularized network. 'MH' refers to the Mahalanobis detector and 'TR' refers to our detector.

H.1 Implementation Details

For ResNeXt50 ([Xie et al., 2017]) on CIFAR100 ([Krizhevsky, 2009]), we train for 300 epochs using SGD with a learning rate of 0.1 (divided by ten at epochs 150, 225 and 250), Kaiming initialization, a batch size of 128 and weight decay of 0.0001. For RCE training, the only changes are that the learning rate is 0.05 and the initialization is orthogonal with a gain of 0.05.

For ResNet110 ([He et al., 2016b]) on CIFAR10 ([Krizhevsky, 2009]), we train for 300 epochs using SGD with a learning rate of 0.1 (divided by ten at epochs 150, 225 and 250), orthogonal initialization with a gain of 0.05, a batch size of 256, weight decay of 0.0001 and gradient clipping at 5. For RCE training, the only change is that we don't use gradient clipping.

For WideResNet ([Zagoruyko and Komodakis, 2016]) on TinyImageNet, we train for 300 epochs using SGD with a learning rate of 0.1 (divided by ten at epochs 150, 225 and 250), orthogonal initialization with a gain of 0.1, a batch size of 114 and weight decay of 0.0001.

For the magnitude parameter of the Mahalanobis detector, we try all the values tried in their paper for the magnitude and we report the best results.

H.2 Detection of Seen Attacks

Table 32: Average adversarial detection accuracy of seen attacks and standard deviation over 5 runs using ResNet110 on CIFAR10.

	Attack							
Detector	FGM	APGD	BIM	DF	CW	AA	HSJ	BA
VAN TR	97.14 ± 0.59	94.10 ± 0.41	97.54 ± 0.52	**99.98** ± 0.49	**98.04** ± 0.47	88.88 ± 1.34	**99.94** ± 0.11	96.56 ± 0.56
RCE TR	95.96 ± 0.54	95.32 ± 0.80	96.16 ± 0.61	99.92 ± 0.10	89.36 ± 0.53			
LAP TR	**98.70** ± 0.28	**97.50** ± 0.37	**99.28** ± 0.30	99.84 ± 0.11	97.96 ± 0.37	**94.08** ± 0.80	99.92 ± 0.14	**97.02** ± 0.18
VAN MH	87.78 ± 4.20	82.08 ± 4.00	86.78 ± 4.70	91.50 ± 3.17	85.58 ± 2.60	80.46 ± 2.24	85.50 ± 1.75	80.20 ± 2.11
RCE MH	93.00 ± 0.60	87.88 ± 0.49	92.30 ± 1.03	94.98 ± 0.40	83.38 ± 0.49			
LAP MH	95.64 ± 0.62	90.70 ± 0.72	95.38 ± 0.54	96.70 ± 0.69	93.36 ± 0.61	89.96 ± 0.88	94.56 ± 0.36	89.62 ± 0.43
VAN NS	94.56 ± 0.67	94.28 ± 0.54	95.04 ± 0.52	99.78 ± 0.10	93.86 ± 9.22	88.78 ± 1.41	99.68 ± 0.10	92.10 ± 0.46

Table 33: Average adversarial detection accuracy of seen attacks and standard deviation over 5 runs using ResNeXt50 on CIFAR100.

	Attack					
Detector	FGM	PGD	BIM	AA	DF	CW
VAN TR	97.26 ± 0.53	96.04 ± 0.48	98.02 ± 0.28	84.90 ± 0.74	**99.80** ± 0.15	**97.04** ± 0.88
RCE TR	97.44 ± 0.43	96.58 ± 0.14	97.78 ± 0.22	50.10 ± 0.09	99.04 ± 0.14	92.52 ± 0.34
LAP TR	**98.32** ± 0.28	**97.84** ± 0.54	**98.92** ± 0.14	**87.56** ± 0.64	99.58 ± 0.18	97.80 ± 0.18
VAN MH	95.82 ± 0.49	93.94 ± 0.48	96.06 ± 0.64	83.90 ± 0.69	97.30 ± 0.45	95.38 ± 0.56
RCE MH	96.46 ± 0.43	94.74 ± 0.44	96.58 ± 0.60	50.10 ± 0.09	97.64 ± 0.44	88.36 ± 0.62
LAP MH	96.82 ± 0.36	94.60 ± 0.70	97.76 ± 0.50	86.58 ± 0.54	97.12 ± 0.28	96.42 ± 0.42
VAN NS	94.70 ± 0.67	94.18 ± 0.97	94.72 ± 0.65	84.82 ± 0.76	99.56 ± 0.21	90.72 ± 1.39

Table 34: Average adversarial detection accuracy of seen attacks and standard deviation over 5 runs using WideResNet on TinyImageNet.

	Attack			
Detector	FGM	APGD	BIM	AA
VAN TR	**95.36** ± 0.36	**95.22** ± 0.50	**95.26** ± 0.46	**81.38** ± 0.38
LAP TR	95.14 ± 0.49	95.20 ± 0.71	95.12 ± 0.67	81.24 ± 0.53
VAN MH	81.06 ± 1.10	79.66 ± 1.00	81.20 ± 1.31	78.40 ± 0.72
LAP MH	85.26 ± 0.96	85.10 ± 0.60	82.46 ± 1.62	78.40 ± 1.04
VAN NS	94.90 ± 0.70	94.86 ± 0.93	95.00 ± 0.56	81.32 ± 0.22

H.3 Detection of Unseen Attacks

Table 35: Average adversarial detection accuracy of unseen attacks after training on FGM and standard deviation over 5 runs using ResNet110 on CIFAR10.

	Attack						
Detector	APGD	BIM	AA	DF	CW	HSJ	BA
VAN TR	89.3 ± 1.6	96.0 ± 0.7	**85.1** ± 1.1	**91.0** ± 0.9	**93.2** ± 1.0	**93.0** ± 0.9	**90.9** ± 0.6
RCE TR	91.8 ± 1.1	93.6 ± 1.1	50.0 ± 0.1	63.4 ± 1.1	60.5 ± 0.9	63.9 ± 1.0	52.5 ± 0.5
LAP TR	**92.8** ± 0.5	**98.8** ± 0.4	84.2 ± 0.5	75.5 ± 1.2	75.2 ± 1.0	76.8 ± 0.6	75.0 ± 0.4
VAN MH	77.3 ± 4.7	77.2 ± 4.8	72.1 ± 3.1	80.1 ± 3.4	79.9 ± 3.7	79.7 ± 3.0	79.3 ± 3.0
RCE MH	81.5 ± 0.6	82.6 ± 1.3	50.0 ± 0.1	81.2 ± 0.7	76.0 ± 0.9	81.6 ± 0.9	68.5 ± 1.2
LAP MH	87.9 ± 0.8	84.9 ± 0.4	81.9 ± 1.2	81.6 ± 0.7	81.5 ± 0.6	81.5 ± 0.3	81.4 ± 0.8
VAN NS	92.1 ± 0.5	93.9 ± 0.4	51.8 ± 0.6	51.4 ± 0.58	50.84 ± 1.1	52.1 ± 0.7	59.9 ± 5.4

Table 36: Average adversarial detection accuracy of unseen attacks after training on FGM and standard deviation over 5 runs using ResNeXt50 on CIFAR100.

	Attack						
Detector	APGD	BIM	AA	DF	CW	HSJ	BA
VAN T	91.9 ± 0.8	95.0 ± 0.5	73.3 ± 1.0	**85.2** ± 0.6	**78.2** ± 1.0	**85.0** ± 0.4	**92.1** ± 4.8
RCE T	87.7 ± 0.5	95.1 ± 0.9	50.0 ± 0.1	72.3 ± 0.4	61.9 ± 0.5	72.4 ± 0.4	57.7 ± 0.4
LAP T	89.3 ± 0.8	**97.7** ± 0.3	74.0 ± 1.3	76.0 ± 1.0	74.7 ± 1.1	78.1 ± 3.4	71.9 ± 3.9
VAN M	90.9 ± 0.8	93.2 ± 0.3	73.1 ± 0.6	82.7 ± 0.9	76.4 ± 0.7	82.8 ± 1.2	84.5 ± 2.0
RCE M	82.0 ± 0.7	88.6 ± 0.8	50.0 ± 0.1	74.1 ± 0.8	63.0 ± 0.6	74.6 ± 0.3	63.2 ± 0.9
LAP M	86.7 ± 0.9	93.9 ± 0.4	**80.0** ± 0.6	79.4 ± 1.6	80.9 ± 2.0	80.5 ± 3.7	78.2 ± 2.0
VAN NS	**92.16** ± 0.41	93.88 ± 0.97	51.32 ± 0.41	51.62 ± 0.53	51.02 ± 0.43	52.04 ± 0.74	57.9 ± 7.52

Table 37: Average adversarial detection accuracy of unseen attacks after training on FGM and standard deviation over 5 runs using WideResNet on TinyImageNet.

	Attack				
Detector	APGD	BIM	AA	DF	CW
VAN TR	93.26 ± 0.60	94.66 ± 0.49	**77.04** ± 0.74	**90.62** ± 0.60	91.42 ± 1.06
LAP TR	93.48 ± 0.72	**94.80** ± 0.56	76.58 ± 0.48	90.12 ± 0.55	**91.52** ± 0.89
VAN MH	76.96 ± 0.94	77.02 ± 1.08	60.36 ± 0.62	73.18 ± 0.59	75.52 ± 0.82
LAP MH	77.96 ± 0.49	78.00 ± 0.77	61.96 ± 0.89	73.98 ± 1.12	76.22 ± 0.83
VAN NS	**94.06** ± 0.61	94.62 ± 0.64	72.82 ± 1.98	71.96 ± 4.03	65.60 ± 2.20

H.4 Detection Rate of Successful Adversarial Samples

As in [Lee et al., 2018], we might be only concerned with detecting adversarial samples that successfully fool the network and that are created from clean samples that are correctly classified. We find that the detection rate of successful adversarial samples is always very high and close to 100% on our detector. The results are in Table 38 for seen attacks and in Table 39 for unseen attacks.

Table 38: Average detection rate of successful adversarial samples from seen attacks over 5 runs on Network/LAP-Network.

		Attack						
Network/Data	Det	FGM	APGD	BIM	AA	DF	CW	HSJ
ResNet110 CIFAR10	TR	97.7/**98.6**	99.3/**99.4**	98.3/**99.6**	**100**/**100**	**100**/99.9	98.6/98.7	**100**/99.9
	MH	88.3/93.9	85.9/86.8	88.1/93.8	88.3/95.4	93.8/98.2	83.5/93.9	82.3/95.1
	NS	95.9	95.9	96.4	99.9	99.9	**99.9**	99.7
ResNeXt50 CIFAR100	TR	98.2/**98.6**	97.1/**97.9**	98.6/**99.2**	**100**/**100**	**99.9**/99.4	99.9/99.6	
	MH	96.7/97.1	95.8/92.7	96.6/98.0	98.8/98.7	97.6/97.8	98.1/98.1	
	NS	95.0	94.8	94.6	99.9	99.3	**100**	
WideResNet TinyImageNet	TR	95.4/**95.9**	**96.7**/96.4	95.0/**96.1**	**100**/**100**			
	MH	84.0/85.0	82.7/84.8	85.8/86.2	95.6/96.5			
	NS	94.4	94.5	94.3	99.9			

Table 39: Average detection rate of successful adversarial samples from unseen attacks after training on FGM over 5 runs.

		Attack						
Network/Data	Detector	APGD	BIM	AA	DF	CW	HSJ	BA
ResNet110 CIFAR10	TR	**96.94**	**98.68**	**98.54**	**93.42**	**92.22**	**93.22**	**93.38**
	MH	79.80	78.66	81.10	79.96	78.96	78.14	79.42
	NS	93.02	94.24	10.22	8.12	8.76	9.60	25.50
ResNeXt50 CIFAR100	TR	**98.76**	**98.56**	**74.06**	**75.14**	**72.90**	**75.42**	**91.04**
	MH	90.86	93.38	73.74	73.64	72.34	73.66	76.44
	NS	91.8	93.8	8.32	8.06	7.38	8.94	21.9
WideResNet TinyImageNet	TR	**100.0**	**100.0**	**91.06**	**95.00**	**96.00**		
	MH	79.16	86.42	71.70	68.34	76.66		
	NS	93.20	93.98	54.5	50.80	51.96		

H.5 False Positive Rate

We report here the false positive rate on seen (Table 40) and unseen (Table 41) attacks.

Table 40: Average false positive rate of seen attacks over 5 runs on Network/LAP-Network.

		Attack						
Network/Data	Det	FGM	APGD	BIM	AA	DF	CW	HSJ
ResNet110 CIFAR10	TR	3.3/**1.5**	6.3/**2**	2.7/**0.8**	1.9/**1.5**	**0.1**/0.2	2.6/2.8	**0.1**/**0.1**
	MH	13.9/3.5	18.7/4.7	13.6/3.1	13.3/6.4	10.2/4.6	12.4/7.4	11.9/5.8
	NS	5.4	4.9	4.8	2.9	0.4	**2.3**	0.3
ResNeXt50 CIFAR100	TR	3.4/**1.9**	4.6/**2.3**	2.5/**1.9**	4.0/**4.1**	**0.2**/0.3	**0.3**/0.4	
	MH	5.2/3.3	6.3/3.6	4.6/3.3	13.7/10.8	2.9/2.7	2.5/2.2	
	NS	5.0	5.0	4.8	5.1	**0.2**	2.8	
WideResNet TinyImageNet	TR	**3.7**/5.2	5.3/4.9	**3.6**/4.6	**6.8**/7.0			
	MH	18.1/16.3	17.5/16.9	18.0/15.9	14.8/15.3			
	NS	4.5	**4.5**	4.2	7.4			

Table 41: Average false positive rate of unseen attacks after training on FGM over 5 runs.

		Attack						
Network/Data	Detector	APGD	BIM	AA	DF	CW	HSJ	BA
ResNet110 CIFAR10	TR	**6.70**	**6.02**	**8.06**	**6.34**	**5.64**	**6.70**	**6.70**
	MH	21.36	21.46	23.36	18.48	18.86	18.72	19.10
ResNeXt50 CIFAR100	TR	8.96	7.10	**7.82**	**3.74**	**3.74**	**4.86**	**5.74**
	MH	**7.28**	**7.08**	7.90	7.78	7.68	7.56	7.90
WideResNet TinyImageNet	TR	**9.44**	**9.44**	**9.48**	**9.44**	**9.44**		
	MH	14.48	14.54	14.48	14.54	14.82		

H.6 AUROC

We report in Table 42 the AUROC of seen attacks, and in Table 43 the AUROC of unseen attacks. Note that the AUROC is computed on the class-agnostic random forest detector, not on the ensemble of the class-agnostic and the class-conditional detectors.

The AUROC is the area under the receiver operating characteristic curve (or ROC curve). The ROC curve plots the true positive rate (TPR) against the false positive rate (FPR), at different threshold values from 0 to 1.

For our random forest classifiers, it is the percentage of decision trees in the forest that detect an adversarial sample (or an out-of-distribution sample) that is compared to the threshold. If it is higher than the threshold, than the input is declared adversarial (or out-of-distribution). If it is lower than the threshold, then the input is declared clean.

Table 42: Average AUROC of seen attacks over 5 runs on Network/LAP-Network.

		Attack			
Network/Data	Det	AA	DF	CW	HSJ
ResNet110	TR	94.91/**99.95**	**99.94**/99.92	98.77/**99.96**	**99.95**/99.94
CIFAR10	MH	81.94/94.17	89.60/94.17	88.33/94.31	86.01/93.45
ResNeXt50	TR	**99.86**/99.70	**99.84**/99.58	**99.85**/99.61	
CIFAR100	MH	87.13/94.32	86.88/91.82	86.33/91.36	
WideResNet	TR	82.58/**82.95**			
TinyImageNet	MH	70.85/71.36			

Table 43: Average AUROC of unseen attacks over 5 runs on Network/LAP-Network.

		Attack			
Network/Data	Detector	AA	DF	CW	HSJ
ResNet110	TR	**82.70**	**90.09**	**88.84**	**87.30**
CIFAR10	MH	56.46	62.25	61.59	55.89
ResNeXt50	TR	**71.53**	**72.61**	**71.89**	**74.85**
CIFAR100	MH	56.60	56.00	56.06	57.07
WideResNet	TR	**70.37**	**81.06**	**80.22**	
TinyImageNet	MH	51.61	51.72	51.57	

H.7 Detection of Out-of-Distribution Samples

We report below the AUROC (Table 44 for ResNet110 on CIFAR10 and Table 46 for ResNeXt50 on CIFAR100) and the FPR at a fixed TPR of 95% (Tables 45 for ResNet110 on CIFAR10 and 47 for ResNeXt50 on CIFAR100) for the out-of-distribution detection experiments in Section 5.4.4.

Table 44: Average OOD detection AUROC and standard deviation over 5 runs using ResNet110 trained on CIFAR10.

	OOD Experiment 1		OOD Experiment 2	
Detector	CIFAR100 (seen)	SVHN (unseen)	CW-CIFAR10 (seen)	SVHN (unseen)
VAN TR	99.64 ± 0.13	98.74 ± 0.47	**99.32** ± 0.14	**96.29** ± 0.71
RCE TR	99.61 ± 0.09	99.01 ± 0.25	96.53 ± 0.58	86.34 ± 3.98
LAP TR	**99.73** ± 0.09	**99.43** ± 0.28	99.31 ± 0.07	96.12 ± 0.59
VAN MH	92.74 ± 1.50	97.00 ± 0.80	88.38 ± 2.94	88.04 ± 4.03
RCE MH	97.97 ± 0.25	96.26 ± 0.40	88.28 ± 2.22	79.53 ± 3.78
LAP MH	98.06 ± 0.38	96.31 ± 0.75	95.22 ± 0.90	86.54 ± 3.60

Table 45: Average OOD detection FPR at 95% TPR and standard deviation over 5 runs using ResNet110 trained on CIFAR10.

	OOD Experiment 1		OOD Experiment 2	
Detector	CIFAR100 (seen)	SVHN (unseen)	CW-CIFAR10 (seen)	SVHN (unseen)
VAN TR	1.16 ± 0.66	2.68 ± 1.05	2.71 ± 1.01	6.68 ± 0.98
RCE TR	1.16 ± 0.41	1.94 ± 0.86	19.2 ± 2.35	24.98 ± 5.43
LAP TR	**1.02** ± 0.44	**1.54** ± 0.56	**2.70** ± 0.64	**5.68** ± 1.12
VAN MH	36.42 ± 4.29	15.66 ± 1.69	49.46 ± 4.80	36.78 ± 6.97
RCE MH	7.34 ± 0.90	20.30 ± 4.45	41.94 ± 6.44	52.72 ± 7.71
LAP MH	6.98 ± 2.35	17.56 ± 5.76	23.06 ± 6.29	58.40 ± 14.42

Table 46: Average OOD detection AUROC and standard deviation over 5 runs using ResNeXt50 trained on CIFAR100.

	OOD Experiment 1		OOD Experiment 2	
Detector	AA-CIFAR100 (seen)	SVHN (unseen)	CW-CIFAR100 (seen)	SVHN (unseen)
VAN TR	94.96 ± 0.38	78.77 ± 1.15	99.00 ± 0.13	95.17 ± 0.31
RCE TR	50.12 ± 0.05	50.30 ± 8.13	87.50 ± 1.65	79.99 ± 3.62
LAP TR	**96.45** ± 0.08	76.28 ± 0.73	**99.16** ± 0.28	94.84 ± 1.63
VAN MH	93.32 ± 0.41	85.02 ± 0.97	98.24 ± 0.32	93.07 ± 0.33
RCE MH	50.15 ± 0.08	58.04 ± 3.56	86.73 ± 2.03	77.42 ± 3.38
LAP MH	94.87 ± 0.24	**92.76** ± 0.39	97.84 ± 0.20	**95.92** ± 0.64

Table 47: Average OOD detection FPR at 95% TPR and standard deviation over 5 runs using ResNeXt50 trained on CIFAR100.

	OOD Experiment 1		OOD Experiment 2	
Detector	AA-CIFAR100 (seen)	SVHN (unseen)	CW-CIFAR100 (seen)	SVHN (unseen)
VAN TR	27.68 ± 1.84	32.66 ± 1.41	5.42 ± 0.82	**8.90** ± 1.20
RCE TR	96.12 ± 0.96	94.72 ± 3.10	44.12 ± 3.31	45.68 ± 2.01
LAP TR	**21.62** ± 0.30	**29.47** ± 0.59	**4.90** ± 0.92	10.52 ± 3.61
VAN MH	29.75 ± 1.09	39.06 ± 0.96	8.02 ± 0.81	16.38 ± 0.82
RCE MH	95.28 ± 0.27	91.42 ± 1.26	45.12 ± 3.76	48.45 ± 4.55
LAP MH	24.74 ± 0.69	34.54 ± 1.16	8.10 ± 0.46	9.80 ± 1.21