Robustness Analysis of Classifiers Against Out-of-Distribution and Adversarial Inputs

Samy Chali

université
PARIS-SACLAY

Robustness Analysis of Classifiers Against Out-of-Distribution and Adversarial Inputs

Analyse de la robustesse de classifieurs aux données hors distribution et adverses

Thèse de doctorat de l'université Paris-Saclay

École doctorale n° 575 Electrical, Optical, Bio : physics and engineering (EOBE)
Spécialité de doctorat : Sciences de l'Information et de la Communication
Graduate School : Sciences de l'ingénierie et des systèmes.
Référent : Faculté des sciences d'Orsay

Thèse préparée au **Centre de Nanosciences et de Nanotechnologies (Université Paris-Saclay, CNRS)**, sous la direction de **Jacques-Olivier KLEIN**, professeur des universités, le co-encadrement de **Inna KUCHER**, ingénieur-chercheur et du co-encadrement de **Marc DURANTON**, ingénieur-chercheur.

Thèse soutenue à Paris-Saclay, le 28 Mars 2024, par

Samy CHALI

Composition du jury

Membres du jury avec voix délibérative

Damien QUERLIOZ Directeur de recherche, CNRS, Université Paris-Saclay	Président
Maria ZULUAGA Maître de conférence HDR, Eurecom	Rapporteur & Examinatrice
Ronan FABLET Professeur des universités, IMT Atlantique	Rapporteur & Examinateur
Eiji KAWASAKI Chargé de recherche, CEA List, Université Paris-Saclay	Examinateur

THESE DE DOCTORAT
NNT : 2024UPAST012

université PARIS-SACLAY

DOCTORAL SCHOOL
Electrical, Optical, Bio: Physics and Engineering (EOBE)

Titre : Analyse de la robustesse de classifieurs aux données hors distribution et adverses.
Mots clés : Apprentissage automatique, apprentissage profond, modèles génératifs, robustesse, classification, mémoires associatives

Résumé : De nombreux problèmes traités par l'IA sont des problèmes de classification de données d'entrées complexes qui doivent être séparées en différentes classes. Les fonctions transformant l'espace complexe des valeurs d'entrées en un espace plus simple, linéairement séparable, se font soit par apprentissage (réseaux convolutionels profonds), soit par projection dans un espace de haute dimension afin d'obtenir une représentation non-linéaire 'riche' des entrées puis un appariement linaire entre l'espace de haute dimension et les unités de sortie, tels qu'utilisés dans les Support Vector Machines (travaux de Vapnik 1966-1995). L'objectif de la thèse est de réaliser une architecture optimisée, générique dans un domaine d'application donné, permettant de pré-traiter des données afin de les préparer pour une classification en un minimum d'opérations. En outre, cette architecture aura pour but d'augmenter l'autonomie du modèle en lui permettant par exemple d'apprendre en continu, d'être robuste aux données corrompues ou d'identifier des données que le modèle ne pourrait pas traiter.

Title : Robustness Analysis of Classifiers Against Out-of-Distribution and Adversarial Inputs
Keywords : Machine learning, deep learning, generative models, robustness, classification, associative memories

Abstract : Many issues addressed by AI involve the classification of complex input data that needs to be separated into different classes. The functions that transform the complex input values into a simpler, linearly separable space are achieved either through learning (deep convolutional networks) or by projecting into a high-dimensional space to obtain a 'rich' non-linear representation of the inputs, followed by a linear mapping between the high-dimensional space and the output units, as used in Support Vector Machines (Vapnik's work 1966-1995). The thesis aims to create an optimized, generic architecture capable of preprocessing data to prepare them for classification with minimal operations required. Additionally, this architecture aims to enhance the model's autonomy by enabling continuous learning, robustness to corrupted data, and the identification of data that the model cannot process.

Remerciements

Je tiens à remercier à mon équipe encadrante, comprenant Jacques-Olivier Klein, Marc Duranton, Inna Kucher ainsi que Johannes Thiele. Ils ont su me donner les clés pour réussir ma thèse tout en me laissant cette marge de liberté créative qui m'a permis de m'épanouir en tant que chercheur tout au long de ma thèse. Je veux particulièrement remercier Inna qui a assuré la reprise de l'encadrement de ma thèse et a tout de suite su se distinguer comme une encadrante d'exception, engagée et se dépassant tous les jours pour ses thésards. Je veux également mentionner l'aide d'autres personnes qui m'ont donné des conseils avisés pendant ma thèse. Pour cela, j'adresse ma gratitude à Daniel Brunner, Olivier Sentieys, Eiji Kawasaki et à tant d'autres.

J'adresse aussi mes remerciements à mes collègues du LIAE, thésards, ingénieurs, stagiaires ou alternants, qui étaient toujours là pour me conseiller et m'épauler. Nos moments d'échange m'ont particulièrement stimulé et permis une ouverture qui a été cruciale dans mon développement. Les amitiés que nous avons nouées me tiennent à cœur et j'espère leur avoir appporté un soutien à la hauteur de celui qu'ils m'ont donné.

Enfin, mille mercis à ma famille qui a su montrer sa fierté et son soutien tout au long de mon aventure. Ils n'ont jamais douté de moi et mes accomplissements leur ont donné raison. Je suis conscient de leurs efforts et de la patience investis dans mon éducation afin de me fournir les meilleures conditions possibles pour ma réussite. Leur intérêt pour mes travaux était un moteur qui m'a aidé à persévérer. Pour ce soutien inconditionnel, je les remercie du fond du cœur.

Table des matières

Glossary

$\nabla_x f$ or $\frac{\partial f}{\partial x}$	Gradient of the function f with respect to the variable x. Corresponds to the vector formed by the partial derivatives of the function with respect to each components of the variable x.
$\int_{\mathcal{S}} f(x)dx$	Integral of a function f over a set $\mathcal{S}$.
$\sum i = 1^N u_i$	Sum of a sequence of elements u_i, $i = 1...N$.
$\prod_{i=1}^{N} u_i$	Product of a sequence of elements u_i, $i = 1...N$.
$\parallel x \parallel$	Norm of a vector x.
A^T	Transpose of a vector or a matrix A
$\|\mathcal{S}\|$	Cardinal of a set $\mathcal{S}$, i.e;: number of elements in the set.
$\min_x f(x)$	Minimum of a function f with respect to a variable x (over the set of definition of f).
$\max_x f(x)$	Maximum of a function f with respect to a variable x (over the set of definition of f).
$argmin_x f(x)$	Value that minimizes a function f with respect to a variable x (over the set of definition of f).
$argmax_x f(x)$	Value that maximizes a function f with respect to a variable x (over the set of definition of f).
$sup_{x,x\in\mathcal{S}} f(x)$	Supremum value of a function f with respect to a variable x, x being in a set $\mathcal{S}$.
$\mathcal{X}$	Ambient space/input space/feature space, i.e.: space of the input x to a model.
$\mathcal{Y}$	Label space, i.e.: space of the labels y associated to an input with features x.
$x \sim p$	x is sampled/drawn from the distribution parameterized by the density p.
$\mathcal{N}$	Normal or Gaussian distribution.
$\log$	Logarithm function.
e^x or $exp(x)$	Exponential function at an input x.
$det(M)$	Determinant of a matrix M.
$p(x\|y)$	Conditional probability of the variable x knowing the variable y.
$\mathbb{E}_p[X]$	The expectation of a random variable X with respect to the distribution p.
$\mathbb{E}(x\|y)$	Conditional Expectation of the variable x knowing the variable y. Expectation taken with respect to the conditional distribution.
$\mathbb{H}(X)$	Entropy of the random variable X.

$D_{KL}(p\|\|q)$	KL divergence of the distribution p relatively to the distribution q.
$\mathcal{B}_\epsilon(x)$	Ball of radius ϵ centered around a point x. It is often defined relative to a given norm. Corresponds to the set of points such that thy are at a radius ϵ of x.
$\mathcal{L}(x)$	Refers to the log-likelihood of a data point x.
L	Refers to a general loss function of a function.

Résumé étendu en français

Le domaine de l'intelligence artificielle est aujourd'hui largement dominé par le *deep learning* (ou apprentissage profond) où l'on cherche à faire apprendre à un réseaux de neurones, avec des données d'entraînement, la réalisation d'une tâche. L'une de ces tâches, la classification, consiste à assigner une classe (ou label) à une données d'entrée, par exemple une image. Un réseau de neurones a connaissance de l'ensemble des classes qu'il peut assigner à une donnée d'entrée. En dehors de son entraînement (phase d'inférence) il est supposé classer correctement une nouvelle donnée réelle. Cette hypothèse de généralisation se fonde sur la représentativité des données d'entraînement, c'est-à-dire que les données rencontrées auront une distribution de classes et une distribution des données d'entrée similaires aux distributions des données d'entraînement. Cependant, en pratique, la distribution des données peut changer au cours du temps et des données erronées peuvent également être rencontrées par le modèle. Ces données hors-distribution (ou *out-of-distribution*, OOD) peuvent être caractérisées par un changement dans la distribution des données d'entrées ($p(x)$, x représentant les données d'entrées) ou dans la distribution des labels ($p(y)$, y représentant un label). Les données fournies au modèle en inférence peuvent alors être associées à des classes que le modèle n'a pas vues pendant l'entraînement ou des données d'entrée dont la distribution diffère par rapport à l'entraînement, trompant ainsi le modèle.

Lorsque les distributions des données x et des labels y changent, on dit que les nouvelles données sont OOD. Le modèle ne connaît dans ce cas ni les données d'entrée (les attributs ont été modifiées) ni les labels. Bien que la tâche soit toujours de la classification, assigner un label à ces données OOD n'aurait plus de sens. Il pourrait être problématique pour un modèle de traiter ces données sans prendre de précautions supplémentaires. Le modèle devrait alors détecter les éléments qu'il ne peut pas traiter correctement afin de les déléguer rapidement à un humain qui serait capable prendre une décision, par exemple. Cette tâche s'appelle de la détection OOD. Il s'agit d'un type de classification binaire (deux classes), où le modèle assigne un label "ID" (*in-distribution*) à une donnée qui est tirée suivant la distribution d'entraînement ou "OOD" lorsque cette donnée est hors-distribution.

Un réseau de neurones, dans une tâche de classification, associe un label à une donnée en associant une probabilité $p(y|x)$ à chaque label possible. Le label assigné à la donnée fournie en entrée correspond à la classe ayant la plus forte probabilité calculée par le modèle. Un moyen de détecter des données OOD serait alors d'utiliser cette distribution de probabilité, car on s'attendrait intuitivement à ce qu'un modèle incertain de sa prédiction (et donc étant face

à une donnée OOD) assigne une distribution uniforme sur tous les labels, c'est-à-dire que tous les labels seraient équiprobables pour une donnée d'entrée OOD. Cependant, une observation courante en *deep learning* est que les réseaux de neurones ont tendance à montrer un excès de confiance en classification sur des données OOD en associant des probabilités plus élevées sur une classe sur des données OOD que ce qu'il calculerait habituellement sur une donnée ID. D'autres modèles, dits modèles génératifs, fournissent aussi une mesure de vraisemblance des données, permettant d'avoir une mesure de la distribution $p(x)$. Cette mesure pourrait également constituer une bonne manière de détecter des données OOD. Cependant, cette mesure présente également des risques en détection OOD. En effet, des données OOD avec une entropie plus faible que les données ID se voient associer une vraisemblance plus élevée, indiquant donc que pour le modèle, les données OOD sont plus vrai semblables que ses données ID. Ce problème serait possiblement lié à l'entropie des distributions ID et OOD en jeu. Utiliser des estimations des distributions de probabilité pourrait donc poser problème dans le domaine de la détection OOD. D'autres métriques doivent donc être explorées pour pallier à ce problème.

Dans cette thèse, la première partie de mon étude se focalise sur le problème de la détection OOD, définit plus haut, où l'on suppose que les données en inférence peuvent à la fois présenter une déviation de la distribution des labels $p(y)$ ainsi que des attributs (ou *features*) en entrées $p(x)$. J'étudie dans un premier temps une métrique, l'*approximate mass*, qui est une métrique liée à la vraisemblance. Celle-ci quantifie les variations relatives de vraisemblance dans le voisinage d'une donnée et doit être plus élevée pour des données OOD. Cette métrique a été introduite dans le contexte des modèles à énergies, qui sont des modèles permettant l'évaluation de la vraisemblance de données, à une constante de normalisation près. Cette métrique semble fournir de très bon résultats en détection OOD avec des modèles à énergie. Cependant, d'après mes expériences, des modèles génératifs comme des *normalizing flows*, ayant la capacité d'estimer complètement la vraisemblance (sans constante de normalisation), semblent démontrer un comportement inverse aux attentes en assignant des valeurs plus faible d'*approximate mass* aux données OOD et plus élevées aux données ID. Après analyse empirique, ce phénomène ne semble pas tout le temps être présent chez ces modèles. En entraînant un *normalizing flow*, je constate que ce phénomène ne s'observe que vers la fin de l'entraînement mais pas au début. Il y a donc eu une inversion des valeurs d'*approximate mass* au cours de l'entraînement. Ce phénomène est donc lié à un manque de généralisation du modèle, à savoir que le modèle n'arrive pas à s'adapter à de nouvelles données de test. J'ai donc eu l'idée d'incorporer dans l'objectif d'entraînement d'un *normalizing flow* une minimisation de l'*approximate mass* afin de contrôler ses valeurs sur les données

ID. Cette méthode fournit des *normalizing flows* présentant le comportement désiré en détection OOD. Ce modèle a également été testé et a montré des performances supérieures en détection OOD par rapport à des modèles de l'état de l'art. Cependant, il semblerait que le modèle soit plus sensible à des déviations sur la distribution des attributs en entrée x que sur les labels y car les résultats en détection d'anomalie (tâche où $p(x)$ reste stable mais $p(y)$ varie) sont plus mitigés. Cette contribution a également contribué à améliorer la méthodologie d'évaluation de méthodes de détection OOD en apportant plus de rigueur dans les tests. En effet, la littérature en détection OOD présentait des manières de tester les performances de plusieurs façons, sans standard et parfois sans justifications sur les métriques utilisées et sans prendre de précautions sur l'équilibrage entre les données OOD et ID en test. Ces travaux ont mené à une publication dans les proceedings de CVPR 2023, contenant les contributions suivantes :

- Observation de l'évolution de l'*approximate mass* lors de l'entraînement d'un *normalizing flow* ;
- Développement d'un nouvel objectif d'entraînement pour les *normalizing flows*, alternatif au maximum de vraisemblance ;
- Apports méthodologiques pour l'évaluation et la comparaison de modèles en détection OOD ;
- Établissement de résultats dans l'état de l'art en détection OOD ;
- Détermination des types de déviations de distribution (attributs ou labels) pour lesquelles le modèle est le plus sensible.

Dans une deuxième partie, la thèse s'est concentrée sur la défense d'un modèle de classification face à des données adverses, qui sont des données perturbées par un attaquant. Ces perturbations sont imperceptibles par un humain mais permettent de tromper le classifieur en lui faisant associer une autre classe à cette donnée corrompue. Ces attaques sont générées en trouvant des attributs x ayant un faible score pour le modèle et en contraignant ce nouveau point à être dans un rayon proche de l'originale, de sorte à ce que la perturbation soit imperceptible.

La meilleure méthode de défense dans l'état de l'art face à ce genre de corruption est l'*adversarial training* où l'on entraîne un modèle en l'exposant à des données attaquées. Cette méthode présente le désavantage de directement classer les données sans signaler si celle-ci est corrompue. Ceci rend la classification dangereuse, en particulier face à un attaquant réalisant des attaques plus fortes que ce qui a été observé en entraînement. Pour pallier ce problème, une autre solution consisterait à détecter les données adverses puis à les restaurer. Ceci permettrait de savoir en cas d'échec lors de la classification par le modèle par quoi cela aurait pu être causé, à savoir une donnée corrompue. L'étude réalisée sur la détection OOD montre que l'*approximate mass* serait un bon candidat pour détecter les exemples adverses, notamment

grâce à sa sensibilité aux déviation sur la distribution des attributs. En effet, la distribution de données adverses fournie par un attaquant peut être considérée comme une distribution altérée de la distribution des attributs originaux. Ainsi, le problème de détection d'attaques adverses peut donc être considéré comme un problème de détection OOD. De la même manière que pour les données OOD, une donnée est considérée comme adverse lorsqu'elle présente une *approximate mass* élevée.

Afin de calculer une *approximate mass*, il est nécessaire que le classifieur soit équipé d'une mesure de vraisemblance afin de calculer cette approximate mass. Ceci est réalisée en utilisant un modèle DIGLM[1], qui est une architecture basée sur un *normalizing flow* qui pré-traite les données d'entrée en vue d'une classification par un classifieur linéaire par la suite. La classification se réalise sur la représentation fournie par le *normalizing flow* dans son espace latent. Ce type de modèle s'appelle un modèle hybride car il fournit à la fois une mesure de la vraisemblance des données $p(x)$ ainsi qu'une mesure de classification, comme pour un réseau de neurones en tâche de classification, avec $p(y|x)$. Ce modèle peut à la fois classer des données et calculer l'*approximate mass* sur ces données, permettant de détecter si la donnée en entrée est malicieuse. En m'inspirant de mon étude sur la détection OOD, j'ai donc décidé d'intégrer l'*approximate mass* dans l'objectif d'en traînement du DIGLM afin de contraindre le modèle à attribuer des valeurs plus faible d'*approximate mass* aux données saines (données sans corruptions) qu'aux données adverses.

Une fois l'attaque détectée, il est intéressant de restaurer le contenu orignal des données. Ceci peut s'effectuer par un processus appelé *Langevin sampling* (ou échantillonnage de Langevin) qui est une marche aléatoire ayant la propriété de produire des échantillons suivant la distribution des données d'entraînement. Cette propriété permettrait de rapprocher un échantillon adverse de la distribution d'entraînement du modèle, qui est une distribution fiable car connue par le modèle et par les utilisateurs humains. Ce processus, inspiré de la physique, dépend d'une estimation d'une grandeur, le score ($\nabla_x \log p(x)$), dont la norme correspond à l'*approximate mass*. Le score est directement accessible pour un modèle à énergie, un *normalizing flow* ou un DIGLM. Cette procédure a déjà été utilisée dans la littérature en défense aux attaques adverse pour la restauration de données, cependant certains problèmes ont été rencontrés :

- Les échantillons obtenus par *Langevin sampling* doivent être tirés sur un grand nombre d'itérations du processus de Langevin ;
- on observe dans la littérature scientifique que les échantillons obtenus sur une longue chaîne de calcul, bien que théoriquement appartenant à la distribution d'entraînement, ne sont pas ressemblant aux données de cette dernière ;

- aucune approche ne prend en compte le fait que les données adverses sont dans un rayon proche des données originales.

J'ai donc apporté à l'étape du *Langevin sampling* une étape finale de projection sur une boule centrée autour de la donnée adverse. Cette projection se fait sur une boule L_∞ car, à rayon égal, elle englobe toutes les autres boules L_p avec p un entier. Les résultats des expériences réalisées sur le jeu de données MNIST [2] en détection et en classification sont encourageants et démontrent que l'approche

- a un potentiel d'apporter une défense modulable car on peut modifier à tout moment le rayon de projection pour s'adapter à de nouvelles spécifications ;
- permet de créer une défense interprétable, où les données attaquées sont détectées au préalable, permettant de signaler et d'expliquer des manques de performance à un utilisateur humain ;

Les résultats de cette analyse démontrent que les performances en détection constituent une autre preuve de la sensibilité de l'approximate mass aux déviations de la distribution $p(x)$ des attributs. La défense complète quant à elle bénéficie des résultats en détection ainsi qu'en restauration des données et en classification sur les données saines. Enfin, la dernière expérience portait sur la résistance de la défense à une attaque adaptative, c'est-à-dire une attaque visant à déjouer le système de défense d'un classifieur, et qui en l'occurrence consisterait à tromper le classifieur en classification et en détection. Il semblerait, d'après les résultats, qu'il est difficile pour un attaquant de créer une attaque qui soit suffisamment furtive pour ne pas être détectée par le classifieur et à la fois suffisamment puissante pour tromper le classifieur.

Enfin, les idées explorées dans cette thèse peuvent ouvrir de nouvelles perspectives d'études à l'avenir. Notamment, un lien peut être établi entre les méthodes exposées dans ce manuscrit et les mémoires associatives puisque ces modèles permettent de corriger des erreurs dans un code stocké dans sa mémoire. Cette restauration des données corrompues se fait par une méthode itérative se rapprochant d'une donnée saine stockée en mémoire par mesure de proximité. Une donnée corrompue est fournie au modèle, le modèle cherche la donnée la plus similaire et répète cette étape jusqu'à se rapprocher de la donnée la plus similaire possible. Cette méthode itérative se rapproche de la technique présentée précédemment basée sur du *Langevin sampling* en restauration d'attaques adverses. La différence se situe sur le fait que les modèles classiques de mémoires associatives cherchent des données en mémoire ayant été stockées au préalable alors qu'une technique comme le *Langevin sampling* est probabiliste et échantillonne dans ce cas une donnée à partir d'une distribution de probabilité. Cette propriété offre plus de flexibilité étant donné que la mémoire ne repose pas sur le stockage d'un nombre limité d'éléments en mémoire mais ne dépend que de l'expressivité du modèle.

Les modèles à base de mémoire sont fréquemment mobilisés en *deep learning* et apportent souvent des améliorations aux modèles classiques de réseaux de neurones en leur permettant de résoudre de nouvelles tâches et en les rendant plus robustes. Cependant, ces modèles font appel à une mémoire externe, nécessitant des échanges entre ce dernier et le réseau de neurones principal. L'utilisation d'un modèle basé sur des mémoires associatives permettrait de se passer de modules externes, augmentant la complexité du modèle à cause du nombre de paramètres ainsi que les échanges et les calculs entre le module de mémoire et le réseau contrôleur. Considérer l'espace latent d'un modèle probabiliste comme une mémoire permettrait d'améliorer la robustesse du modèle (comme démontré lors de cette thèse) mais aussi de réaliser de nouvelles tâches, comme de l'apprentissage continu (mettre à jour le modèle sur de nouvelles classes sans oublier ce qui a précédemment été appris).

1 - Goal of this work

1.1 . Context of the thesis

The original inspiration for this thesis comes from the observation that some machine learning techniques rely on a high-dimensional representation of their input or their parameters. High-dimensional spaces have unfavorable properties generally in statistical learning because of the so-called "curse of dimensionality" where the amount of data necessary to cover a space increases exponentially with the dimension of the space [3]. The idea can be explained with the illustration in figure 1.1 where the number of points one can fit in a hypercube increases with the dimension. Therefore, fitting data in this space requires a bigger coverage, thus a bigger sample of data.

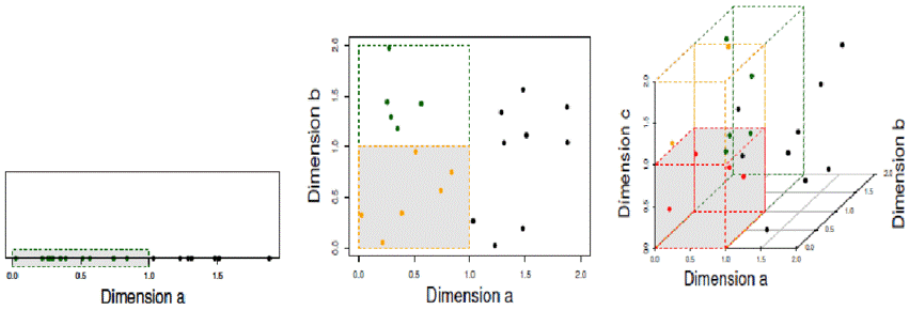


Figure 1.1 – Illustration of the curse of dimensionality from [4]. Adding dimensions adds distance between data points across those dimensions. As dimensions are added, points spread even further making high dimensional data extremely sparse.

Some algorithms, however, use the properties of high-dimensional spaces to discriminate data. It is the case of HDC (High-Dimensional Computing) where the high-dimensional mapping learned by the model is used to cluster data into their respective classes. In the context of HDC, a high number of dimensions is usually defined as being higher than 10,000 in a binary space. The rationale of mapping data to a higher dimensional space is also found in the kernel trick [3] where a kernel function is used to add supplementary dimensions to the input of an SVM (Support Vector Machine) model for example [3] in order to feed more information into the model. An illustration of the kernel trick with SVMs is given in figure 1.2. The approach of HDC is often interpreted as associative memory [5] which is a kind of memory where objects act like addresses as well, as opposed to traditional memory schemes where the address space is separate from the data space. It seems that the clustering effect

associated with the projection of data to a high-dimensional space produces attractors which correspond to the class-centroids of the clusters.

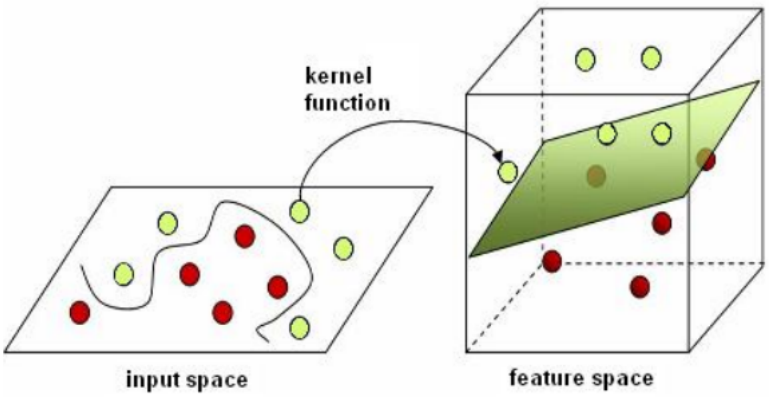


Figure 1.2 – Example of the kernel trick from [6]. The kernel function maps data from a lower dimensional space to a higher-dimensional space making them linearly separable (a hyperplane can be found such that both types of points can be separated).

For more than a decade now, neural networks have started dominating the field of machine learning. Whereas before, the design of feature extractor was made "by hand" with functions such as kernel and learn only the classification part of the model, the modern approach of deep learning indicates that learning the whole processing pipeline of the data yields better results. A trained neural network is composed of a stack of layers each processing the output of the previous layer. It learns a hierarchical representation of the input data as illustrated in figure 1.3 [1], which is a composition of several representations, each focusing on a different level of detail. These representations extracted by the model aggregate more subtle features about the input. Modern deep learning often relies on models with a high number of parameters (overparameterization) which seems to correlate with better results in the problems they are applied in. This overparameterization has been shown to be beneficial to some extent to the learning process and not overfit [7]. The addition of a high number of parameters does not generally add extra dimensions to the input.

I hypothesize that similarly to HDC, the high-dimensional embedding of data might benefit downstream tasks thanks to an emerging property of associative memory. My intuition led me to study the robustness of a family of classifiers in a classification task, more precisely in detecting bad inputs such as out-of-distribution inputs in chapter 3 and adversarial inputs in chapter 4. More precisely, I hypothesize that neural networks may benefit in robustness

1. From https ://pantelis.github.io/cs677/docs/common/lectures/deep-learning-introduction/

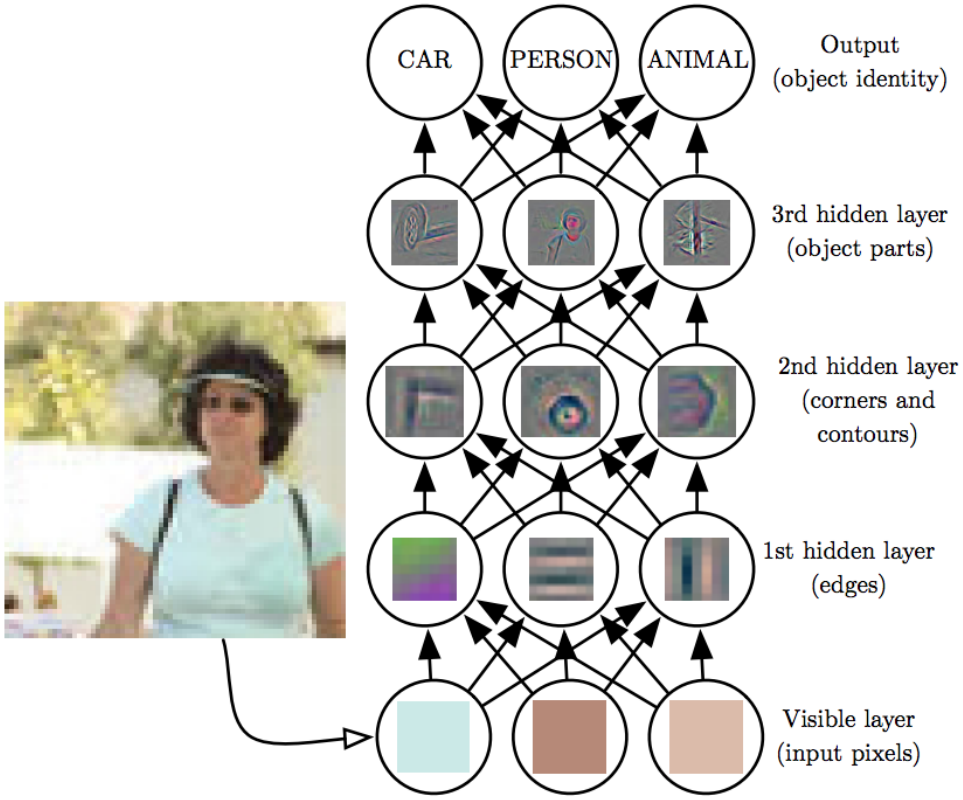


Figure 1.3 – Illustration of the hierarchical representations learned by a deep neural network.

by creating attractors around training data and comparing how close data points are to those learned attractors. Regularizing those attractors may lead to better generalization and robustness from the model. This local smoothing may also allow better interpretability of classifiers.

1.2 . Structure of the document

To introduce my work, I will first start an overview of the domain of artificial intelligence, where I will mainly introduce some notions about deep learning and some models that will be useful for the rest of the thesis. Then, I will start addressing the issue of robustness of classifiers by introducing the concept of out-of-distribution detection along with my contributions to the domain. Next, I will introduce the reader to the field of adversarial machine learning and present my ideas and contributions to the field. Finally, the conclusion will tie all the work introduced in this manuscript and help understanding the full scope of my work as well as the future perspectives for

new developments I have that would bring new contributions.

The structure of the document is displayed in figure 1.4. The reader may follow the natural order of the parts of the document although some other order is possible and would not impede understanding of the manuscript. However, it is crucial for the reader to first read chapter 3 before chapter 4, as some notions laid in the former are important to understand contributions and ideas in the latter, especially about distribution shifts and the approximate mass.

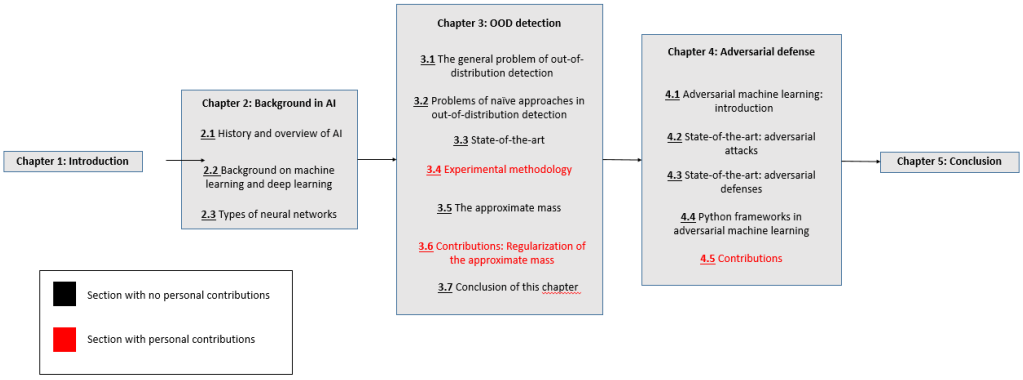


Figure 1.4 – Structure of this thesis. The red sections correspond to the parts where contributions of the thesis are explained in details.

2 - Introduction

2.1 . History and overview of AI

Artificial intelligence (AI) is a domain at the crossroad of computer science and mathematics and other scientific fields such as neuroscience and physics that aims at solving complex problems that have high algorithmic cost or may be loosely formalized or too complicated. Yet some of these problems are still solvable by the human brain. Indeed, some tasks, such as image classification, image segmentation or text generation, deal with such a high number of interconnected variables (a space of pixels in a fixed-size image, a space of vocabulary in a language) that exploring the different configurations of the problem would be prohibitively huge. Furthermore, a problem as loose as "what is this image ? A cat or a dog ?" is too loose to be handled by a program and requires answering sub-questions such as "what is a dog or a cat ?". In this chapter, I make an overview of the field of AI key ideas that will help understand the subject and will be helpful for the rest of the manuscript.

2.1.1 . Symbolic AI and machine learning

The history of modern AI can be traced as far back to 1950 with Alan Turing in his article "Can Machines Think ?" [8]. The ideas in this article mainly dealt with simulating consciousness in a computer but were still precursor of the whole domain.

The two main approaches for AI were statistical learning and symbolic AI. In symbolic AI, a system uses symbols, explicitly embedding human knowledge, and a set of rules in order to perform logical reasoning [9], with the example of the Logic Theorist [10] which was the first system to be similar to AI and which would be able to prove 38 theorems out of 52 in the *Principia Mathematica* of Russel And Whitehead. The interest of symbolic AI comes from its low computational requirements compared to statistical learning as well as its transparency (rules and decisions derived from the system are understandable by a human). The symbolic AI approach is an example of the computationalist school of thought in computational theory of mind which assumes the mind is an automatic formal system [11].

On the other hand, statistical learning or machine learning aims to learn a solution to a problem, by fitting a function, from a dataset. The two fields were seen as contradictory for a long part of the development of AI. Currently, the field of AI is mostly dominated by machine learning, more specifically by deep learning since the evolution of hardware capacity ended up catching up to the computational requirements of such algorithms for modern applications. The power of deep learning lies in its capacity to work on high-dimensional, com-

plex and unstructured data like graphs, images or text [12] [13]. Still, some tasks in machine learning remain dominated by traditional machine learning methods, such as tree-based models on tabular data [14]. Deep learning emerged as a representative of the connectionist school of thought in the computational theory of mind [11].

Symbolic AI and statistical learning may be different in their paradigm but they are not seemingly irreconcilable, as proposed by Marvin Minsky and other important actors of AI [9]. Some modern approaches try to merge the advantages of both connectionism and symbolic AI, such as hyperdimensional computing (HDC) [12] for example. In the next section, I will introduce examples some background in machine learning in section 2.2 before presenting some neural networks architectures that are fundamental to the domain today and that would help understand better the rest of this document.

2.2 . Background on machine learning and deep learning

2.2.1 . The learning problem

Generally, a learning problem is defined as an optimization problem over the set of parameters of the machine learning model considered. More precisely, the goal is for the model to minimize the risk function, which is the expectation of a loss function over a data distribution in order to solve a given task. The end goal is not only for the loss of the model to be low over the given dataset during the "learning" phase but also to generalize well to never-seen data that are drawn from the same data distribution as the training dataset. Theoretically, the problem can be written as in equation 2.1.

$$\min_{\theta} \mathbb{E}_{z\sim p(z)}[L(z,\theta)] = \min_{\theta} \int L(z,\theta) dP(z) \tag{2.1}$$

In particular in an image classification task, where the goal is to identify the associated label to a given image, a common loss defined to train a neural network is the cross-entropy loss $\mathbb{E}_{(x,y)\sim p(x,y)}[-log(f_\theta(x))]$ where f_θ is the classification function outputting the probability of an image x having label y. A problem where we both have an input and its label is called supervised learning, was opposed to unsupervised learning where only the input is available to the model learned. These objectives may however be intractable in a lot of cases, mainly because the integral in equation 2.1 may be too hard to compute or because the set $\mathcal{X}$ of the input data, called the "ambient space", may be too complicated to describe. For example, the set of all natural images of dogs is not formally describable so that an integral can be defined. Therefore, some approximations need to be made in order to compute the risk $\mathbb{E}_{z\sim p(z)}[L(z,\theta)]$, typically through Monte-Carlo estimation. Indeed, the way to approach the evaluation of such integrals is to feed some data from a "training

set" $z_i \sim \mathcal{D}$, then compute the average loss of the model over this training set as per equation 2.2 then minimize it. The term defined in equation 2.2 is called the empirical risk.

$$\mathbb{E}_{z\sim p(z)}[L(z,\theta)] \simeq \frac{1}{|\mathcal{D}|}\sum_{z_i\sim\mathcal{D}} L(z_i,\theta) = \mathcal{R}_{empirical}(\theta,\mathcal{D}) \tag{2.2}$$

Once this new approximation is defined, the optimization problem may still not be easily solvable, as the empirical risk function may be too complex. A general learning framework, most notably in deep learning, uses gradient descent algorithms to solve the problem of equation 2.1 with the approximation defined in equation 2.2. The gradient descent is an algorithm that iteratively updates the value of the parameter of our problem to minimize a function. The algorithm starts from a parameter θ^0 (often random) and successively updates the parameter following equation 2.3, leading to algorithm 1. In equation 2.3, the parameter η_i is the step size of the algorithm, which may or may not be updated every iteration.

$$\theta^{i+1} = \theta^i - \eta_i \nabla_\theta \mathcal{R}_{emp}(\theta^i,\mathcal{D}) \tag{2.3}$$

Algorithm 1 Gradient descent algorithm

$i \leftarrow 0$
$\theta \leftarrow \theta^0$
$\eta \leftarrow \eta^0$
for $i = 1 : N$ **do**
  $\theta \leftarrow \theta - \eta\nabla_\theta \mathcal{R}_{empirical}(\theta,\mathcal{D})$
  Update η
  $i \leftarrow i + 1$
end for

An illustration of the algorithm and its convergence to a minima is given in figure 2.1. The gradient descent algorithm only works for convex functions to find a global minima. With other functions, we have no guarantee that the given result is a global minima, but can be only a local minima. Nonetheless, this optimization technique is the most widely used in deep learning and a number of algorithms derived from it are currently used to train neural networks.

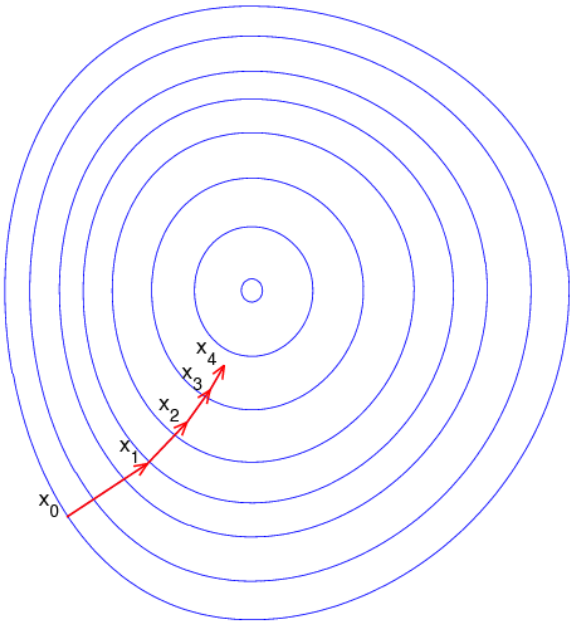


Figure 2.1 – Illustration of the gradient descent procedure. The red arrows represent the successive gradient computed by the algorithm at each step, bringing the parameter closer to a local minima. Credit to Olegalexandrov at English Wikipedia, Public domain, via Wikimedia Commons.

2.2.2 . Discussion and extension of the optimization process to deep learning

Some notes can be made from algorithm 1. First, although the risk is formally minimized over the whole dataset $\mathcal{D}$, it is usually preferable to minimize the risk over mini-batches of a few samples of data, drawn from the dataset $\mathcal{D}$. This way, the algorithm can estimate the gradient of the empirical risk over the small batch which accelerates its computation while adding some noise to the estimates of θ which helps generalizing to new data points [15]. This way of performing the gradient descent algorithm is called stochastic gradient descent (SGD) and is the preferred way to train a neural network in deep learning. Some enhancement of this technique, made to accelerate learning, are adaptive methods which update the gradient at each step of training using information about the gradient descent during previous steps, like the Adam optimizer [16] and the RMSprop optimizer [1]. These adaptive optimizers follow equation 2.4. The RMSprop optimizer divides the learning rate during training by a running average of the parameter values in previous steps, updating the weights following equation 2.4 where g_t is an estimate of the running average of the parameter values and v_t is an estimate of the running variance of

1. First described in a course by Geoffrey Hinton, no paper published to this day.

the parameter values, ϵ is a hyperparameter used for numerical stability. The Adam optimizer on the other hand uses equation 2.4 where g_t is an estimate of the running average of the gradient and v_t is an estimate of the running variance of the gradient computed the previous steps.

$$\theta_{t+1} = \theta_t - \eta_t \frac{g_t}{\sqrt{v_t} + \epsilon} \tag{2.4}$$

The steps described in this section, where a model is fed input data from the training dataset in mini-batches then performs the gradient descent algorithm described in algorithm 1 make up an "epoch" of training. It is common to repeat these steps several times, as the model further decreases its loss after every epoch. A typical training of a neural network therefore involves repeating the gradient descent algorithm for every mini-batches of a training dataset $\mathcal{D}_{train}$, making up an epoch, then test the model after the end of the epoch on a test dataset $\mathcal{D}_{test}$, distinct from the training one in order to assess the generalization capability of the model, then repeat these steps for several epochs. A separate validation set $\mathcal{D}_{val}$, also distinct from both the training and the test set, can be made in order to validate the overall performance of the model with different hyperparameters and initial configuration, such as the network architecture, the step size (or learning rate described in the next paragraph).

The step size η_i, also called learning rate in the context of deep learning, generally changes at each iteration in order to make the algorithm converge. Indeed, a constant step size may make the model stuck in a region without ever converging to a single point. Some scheduling functions exist to reduce the step size while training the model. They can either decrease the step size depending on the value of the risk or simply decrease it by a function of the time step. Some examples of learning rate scheduling are cosine annealing [17] and the exponential scheduling where the learning rate decreases exponentially.

Finally, the initialization of the parameters is a very important topic especially concerning the speed of convergence and the stability of the training procedure. Some initialization schemes exist in deep learning that make use of some prior knowledge on the distribution of parameters in the neural network, such as the He initialization [18] or the Xavier initialization [19].

For neural networks trained with algorithm 1, the gradient of the risk is computed by an operation called "backpropagation", which first appeared in [20] but then reemerged in Hinton's work [21] and Lecun's work [22]. In order to update the parameters of the network, the algorithm needs to compute the gradients of the loss with respect to the weights and the biases of each layer. However, this gradient is not directly accessible for intermediary layers as the value of the loss function at the end of the network depends on the value of the last layer, which itself depends on the value at the previous layer and

so on. Because of this composition of functions, the chain rule, introduced in equation 2.5, must be applied in order to get the gradient of the loss with respect to the parameters of one layer. Thus, the idea of the gradient of the loss being "backpropagated" through the layers of the network. Generally, we separate the training of a neural network into two steps : a "forward pass", where the model is fed data in input and outputs its corresponding values in order to compute the loss, followed by a "backward pass" where the model's gradient is backpropagated.

$$\frac{d}{dx}f(g(x)) = \frac{dg(x)}{dx}\frac{df}{dx}(g(x)) \tag{2.5}$$

Explicit formulas may be derived to compute the gradient of a loss with respect to the parameters of any layer. However it is more usual on modern architectures, which can be very deep and make the computation of the gradients much more complex and prone to human error, to use automatic differentiation [23] which is usually implemented in most deep learning frameworks. Automatic differentiation differs from a finite difference calculation which only yield an estimate of the gradient of a function at a given point. Automatic differentiation outputs the exact value of the gradient at the given point. Automatic differentiation instantiates a graph of computation during the forward pass to keep track of every primitive calculations performed during this step. Then it backpropagates the gradient by reading the graph back from the end to the beginning and applying the chain rule in equation 2.5 knowing the derivatives of the primitive operations.

2.2.3 . Training stability, overfitting and regularization

As we have covered in section 2.2.2, training a neural networks can be a complex task as it involves several factors from the choice of the training algorithm, the initialization for the parameters and the different values of the hyperparameters. Some other problems still arise though during training. For example, during the training of a deep neural network, a phenomenon called the "vanishing gradient" may happen, especially with networks using the sigmoid activation function introduced in section 2.3.1. Indeed, the gradient of the sigmoid function, illustrated in figure 2.2 converges to 0 very quickly for values of the input further from the origin. These small values, although not mathematically equal to 0, are rounded to 0 by computers which stalls the training because the backpropagated gradient used in equation 2.3 is equal to 0. The problem is circumvented by the ReLU (Rectified Linear Unit) activation function [24] [25], introduced in section 2.3.2, which is not everywhere differentiable (irregularity at $x = 0$) but shows the behavior desired for an activation function such as exciting the neurons in the next layer if the input of the function is positive and inhibiting (or rather not exciting) the next neurons if its input is negative. The ReLU activation function is illustrated in figure 2.3

along with its derivative, which does not vanish for positive inputs.

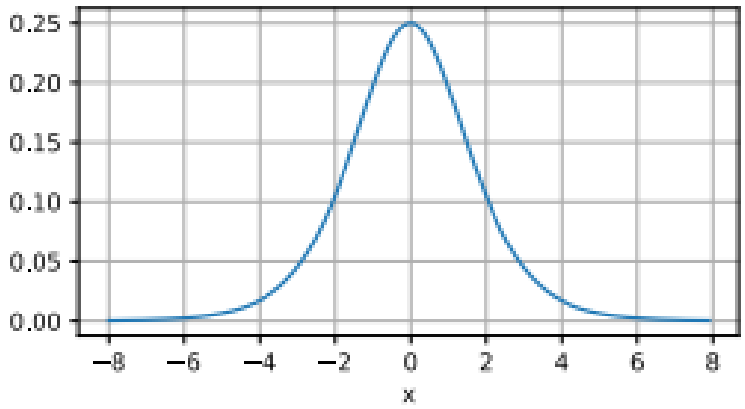
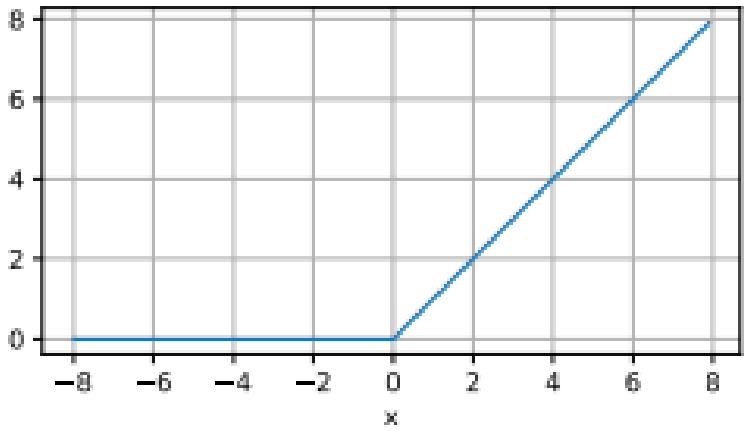


Figure 2.2 – Plot of the gradient of the sigmoid activation function. Taken from the book "Dive into Deep Learning" [26].

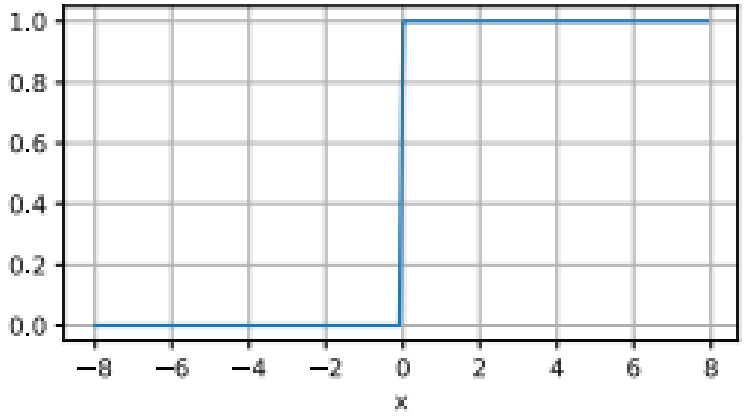
Other methods, such as some architectural changes in the neural networks through the addition of skip layers as introduced in section 2.3.2 has the effect of backpropagating the gradient of a layer to a much earlier one in the network thus mitigating the effect of the vanishing gradient. The initialization is also very important as keeping the weights close to 0 can help keeping the activations of each layer in the linear regime and to avoid getting in the vanishing regime of the sigmoid.

Another problem encountered in training neural networks is the "exploding gradient", as opposed to the vanishing gradient, where the backpropagated gradient takes high magnitude values. This has the effect of assigning too high values for the weights in the network during training, resulting in NaN values (Not a Number). This problem is usually fixed by clipping the gradient values, that is to say by normalizing the magnitude of the gradient to a maximum value. The initialization can also impact the values reached by the gradient similarly to the vanishing gradient problem. The exploding gradient may also be caused by the choice of the ReLU function or the Leaky ReLU [27], illustrated in figure 2.4, as the activation function, the depth of the chosen neural network architecture (deep networks are ore likely to yield exploding gradients), or the optimizer (the Adam optimizer [16] yields more stability than the classic SGD optimizer).

We will see in section 2.3 that the architecture of a neural network includes some inductive biases on the data distribution to model. A prior can also be added to the model by regularizing the model with new terms added to the loss function used in training. The goal of regularization is originally to make the model generalize better to unseen data. The most common example of regularization in machine learning is the L_2 regularization on the weights of



(a) ReLU function.



(b) Derivative of the ReLU function.

Figure 2.3 – Plot of the ReLU activation function and its derivative. Credit to Dive into Deep Learning book [26].

the model, also called Ridge regularization [3] in the context of linear regression or weight decay [26] in the context of deep learning. Weight decay adds a term proportional to the L_2 norm of the weights of the model to the loss function, in order to minimize it, the goal being to make the model more parsimonious. Indeed, according to the principle of Occam's razor [28], a preferred model is one that combines fewer variables for the same overall result. This parsimony makes the model select only the decisive variables of the input to make a decision. Another similar regularization is the Lasso regularization or L_1 regularization [3] which constraints the L_1 norm of the weights of the network, with the effect of bringing some weights to have exact 0 value. Other forms of regularization, such as dropout [26] which randomly drops connections between the neurons of two consecutive layers during training, or data

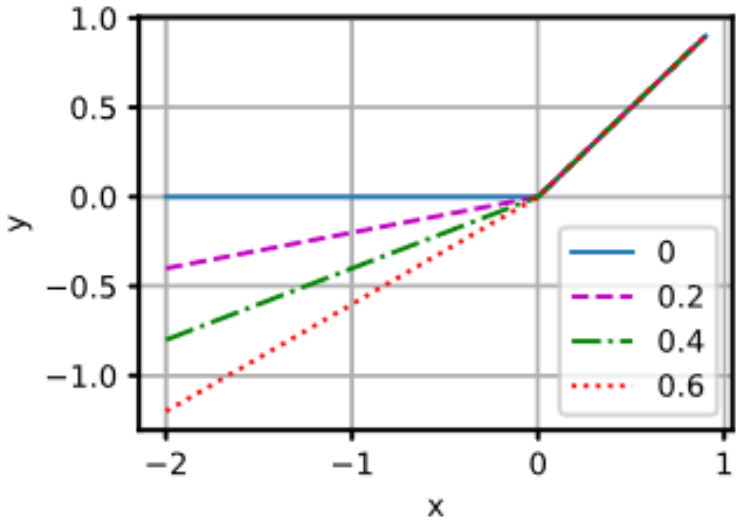


Figure 2.4 – Plot of the leaky ReLU activation function compared to the ReLU function (in blue). Credit to Dive into Deep Learning book [26].

augmentation [26], where the dataset is augmented by transforming the original images in the dataset in several ways (rotating images or changing the color space for example).

As we saw, the goal of regularization is to make the model generalize better. In machine learning, generalization performance is measured through the overfitting of the model. All the regularization techniques introduce in the last paragraph aim at avoiding the overfitting of the model on the training dataset. Overfitting happens when the gap between the model's loss on the training set and on the test set increases during training, as illustrated in figure 2.5.

If the model's loss increases on the test set while it decreases on the train set, it means the model is starting to "memorize" the patterns of the individual data points of the training set instead of learning general pattern that would be transferable from the training set to the test set. Figure 2.6 [2] illustrates the difference between models depending on how well they fit the data points. A line that perfectly fits the training data, the small red and blue points in figure 2.5, like the green line, is not flexible enough which makes generalizing harder. On the other hand, a more regularized and smoother line, like the black line, is more flexible for new data points (thick red and blue points in figure 2.6), although less representative of the training dataset than the green line.

Finally, it also seems to be the case that adding more parameters to the model may increase the overfitting of the model. We call the space of hypothesis $\mathcal{H}$ the space of all possible configuration of the parameters of the model

2. Credit to Chabacano, CC BY-SA 4.0 <https ://creativecommons.org/licenses/by-sa/4.0>, via Wikimedia Commons.

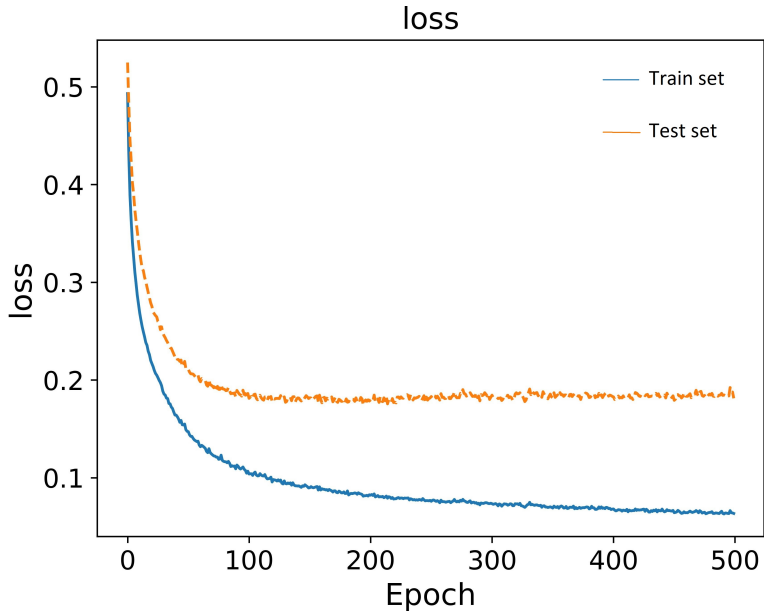


Figure 2.5 – Plots of the train (blue curve) and test (orange curve) loss for a model with respect to the epochs of training. The gap between both losses increases as the training increases, showing that the model begins to overfit and stops generalizing well.

we are fitting on the training dataset. As the capacity of $\mathcal{H}$ increases, observations used to be that the model have poorer generalization performance. However, in the recent years and as computational power increased, it was observed that increasing even more the number of parameters would start to decrease again the overfitting of the model, as explained in [7] by an illustration in figure 2.7.

2.3 . Types of neural network

An artificial neural network, or simply neural network, is a machine learning model that is the composition of successive elementary operations, called layers, parameterized by weights W and biases b and that depend on an activation function σ which is a non-linear function. Similarly to natural neurons, a layer of neurons can either excite of inhibit the neurons in the next layer through the values of the activation function.

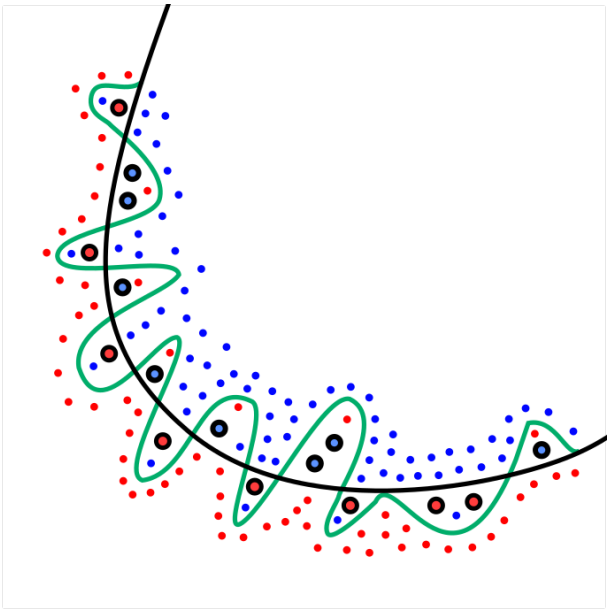


Figure 2.6 – Learning curves of models separating data points in a two-dimensional space (red and blue points). The green line is an overfitted model because it learned to fit the training data perfectly, therefore when a new point arises in the future that happens to be slightly off, the green line would misclassify it. The black line on the other hand is well-fitted and regularized as it separates most of the training data and is flexible enough to adapt to new points on test time.

2.3.1 . Multilayer Perceptron (MLP)

An MLP (MultiLayer Perceptron), also sometimes called a fully-connected neural network, is the most basic type of neural network. They are a type of feed-forward neural network architecture (the input goes into one layer and the output of each layer is the input of the next one) where several layers of fully-connected neurons (each neuron of a layer is connected to every neuron of the next layer) are stacked. The MLP architecture is illustrated in figure 2.8 [3].

Historically, this architecture used a sigmoid activation function $\sigma(x) = \frac{1}{1+e^{-x}}$ whose graph is illustrated in figure 2.9 [4]. The sigmoid activation function was originally chosen as an activation function because it is everywhere differentiable and continuous. It also has the property of continuously approximating a step function. Indeed, if the slope of the function reaches ∞ for inputs near 0, then the graph becomes a Heaviside step function which is a binary function that outputs values in $0, 1$ whose graph is provided in figure

3. Credit to Sky99, CC BY-SA 3.0 <https ://creativecommons.org/licenses/by-sa/3.0>, via Wikimedia Commons.

4. Credit to Dive into Deep Learning book [26].

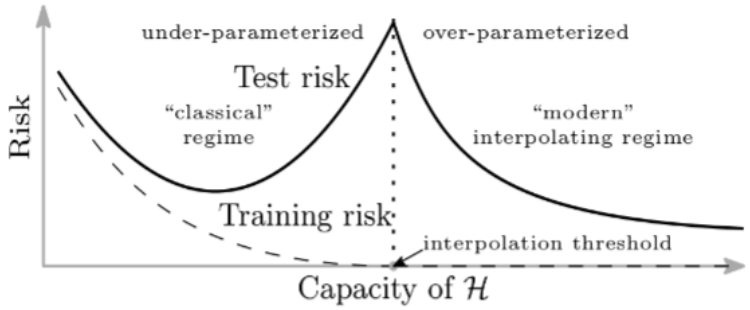


Figure 2.7 – The interpolation regime reported in [7]. The more complex the model, the more sensitive it is to overfitting, until a threshold where the test risk decreases again.

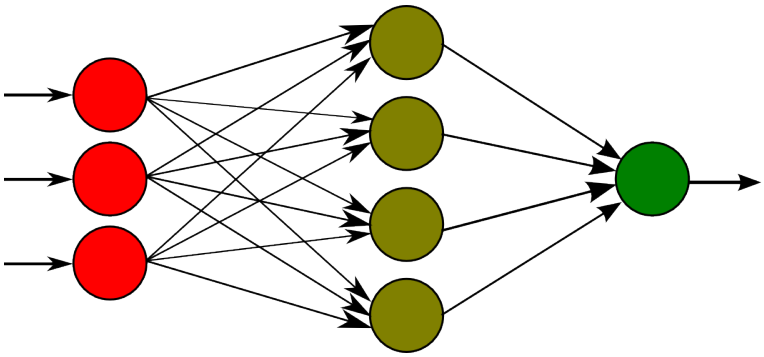


Figure 2.8 – Illustration of the architecture of an MLP. The arrows represent the flow of data within the network, showing why this type of architecture is classified as a feed-forward neural network.

2.10 [5]. This behavior mimics neurons by either inhibiting the next layer (when the activation function is at 0) or exciting it (activation at 1).

MLPs are proved to be universal approximators [29] [30], that is to say for any given function, there exists an MLP that can get arbitrarily close to this given function according to some error function. However, the universal approximation theorems only state the existence of such networks but not their construction, making the development of neural network architectures difficult. Therefore, the choice of the depth and the width of neural networks often rely on heuristics.

The hyperbolic tangent $tanh$ has also been used for the same purpose as the sigmoid activation function. They share a similar shape only differing in their offset with respect to the x-axis. A plot of the $tanh$ function is given in figure 2.11 [6] where it is compared to the sigmoid activation.

5. Credit to Omegatron, CC BY-SA 3.0 <https ://creativecommons.org/licenses/by-sa/3.0>, via Wikimedia Commons.

6. Taken from https ://towardsdatascience.com/activation-functions-neural-

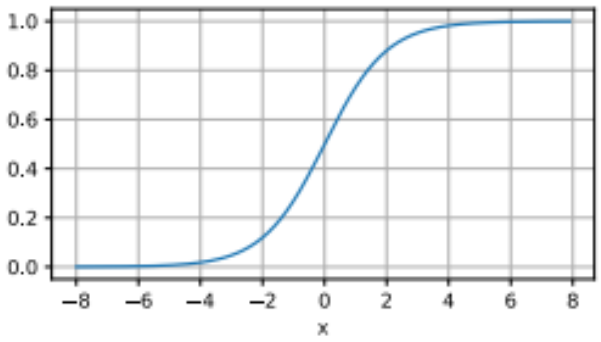


Figure 2.9 – Illustration of the sigmoid activation function.

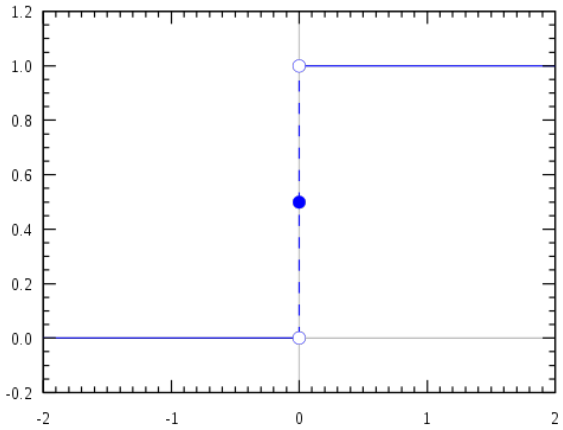


Figure 2.10 – Illustration of the Heaviside step function.

The goal of the activation functions is to add some non-linearity to the network, as it would be equivalent to a piece-wise linear function in their absence. These activation functions are used to simulate the behavior of a biological neuron, with the neuron on one layer firing when its input is positive and inhibiting (with the $tanh$) or not firing (with the sigmoid) when its input is negative.

Each layer of the MLP performs the operation described in equation 2.6 where W is the weight matrix of the layer, b is the bias vector of the layer, σ is the activation function of the layer, and x is the input of the layer. This operation is represented here in vector form but can be written in a similar way for scalar inputs and outputs.

$$output = \sigma(Wx + b) \tag{2.6}$$

In equation 2.6, the transformation inside the activation function σ is a li-

networks-1cbd9f8d91d6.

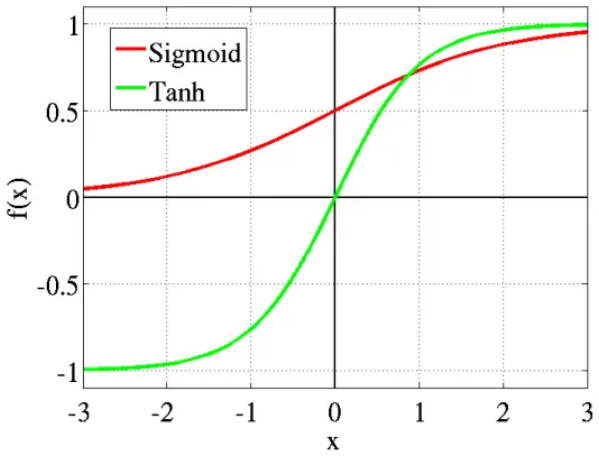


Figure 2.11 – Illustration of the $tanh$ function compared to the sigmoid function.

near transformation with intercept b and slope W. As we saw in section 2.2, those two parameters are learned by the model. A general architecture of neural network has a set of learnable parameters θ. All the following models that we will introduce rely on this paradigm and have a similar shape to the MLP, although the operations performed may differ in order to be more adapted to the data to model.

Finally, the output of the linear transformation of the last layer, called logits,are usually used for a given task. In particular, in classification, this output is fed into a softmax function, described in equation 2.7 for a vector input $x = (x_1, x_2, ..., x_i, ..., x_K)$. This function represents the vector of probability of the input being in class $i \in 1...K$. The exponential functions in this expression are meant to make the input positive. The sum over all the other inputs are used to normalize the softmax so that the function can describe a probability distribution with the logits $x = (x_1, x_2, ..., x_i, ..., x_K)$. Indeed, the sum of all softmax outputs over the input logits $x = (x_1, x_2, ..., x_i, ..., x_K)$ is $\sum_{i=1}^{K} \frac{\exp x_i}{\sum_{j=1}^{K} \exp x_j} = \frac{\sum_{i=1}^{K} \exp x_i}{\sum_{j=1}^{K} \exp x_j} = 1$.

$$softmax(x)_i = \frac{\exp x_i}{\sum_{j=1}^{K} \exp x_j} \tag{2.7}$$

2.3.2 . Convolutional neural networks

A CNN (Convolutional Neural Network) is an architecture that is specifically designed for the images manipulation, although they can also be applied on time series data. The CNN was fundamental in the history of deep learning, as it was the first family of models in deep learning in recent years to beat methods that were dominating the field of computer vision, namely SVMs with explicitly engineered feature transformations, with the AlexNet architecture [31] in 2012 on the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [32]. This model marked the resurgence of CNNs since the 1990s.

A CNN relies on a feed-forward architecture but uses some fundamental operations that are more adapted to the type of data it is made for. The first one is the convolution operation, with a square convolution filter (or kernel) K of size $H \times H$, described with equation 2.8 for an input image I or receptive field, which corresponds to the input of a convolution layer. Note that the dimension of the kernel K (height and width) has to be smaller than the dimension of the input image I. The output of the convolution is called the feature map. The operation defined in 2.8 is only defined for images with a single channel. A channel in an image is a replica of the original image made of just one primary colors, these colors depending on the type of image. For example, an RGB color image will have a red, green and blue channel as in illustration 2.12. A grayscale image has just one channel corresponding to variations of grey in the image for each pixel.

$$[K * I]_{i,j} = \sum_{h=1}^{H} \sum_{w=1}^{H} K_{h,w} \times I_{h+i,w+j} \tag{2.8}$$

Figure 2.13 [7] gives an illustration of the convolution operation. Note that the kernel on one convolution layer "slides" on the input image column by column then row by row, such that it yields several outputs for a single input.

Note that there exist several ways to perform the convolution operation in CNNs (for example, 1×1 convolutions, depthwise convolutions, cross-channel convolutions), but the one introduced here is the most basic one. It is usually applied channel-wise for a multi-channel image such as an RGB image (color image). The kernel itself may be split into several channels and each channel of the kernel is respectively applied to each channel of the image.

The second important operation in CNNs is the pooling operation which is a downsampling operation used to reduce the size of the image, with the most popular type being the max pooling layer. The max pooling operation typically follows a convolution, in order to get the most important output of the feature map and further reduce computation cost on the input image. An

7. Credit to Dive into Deep Learning [26].

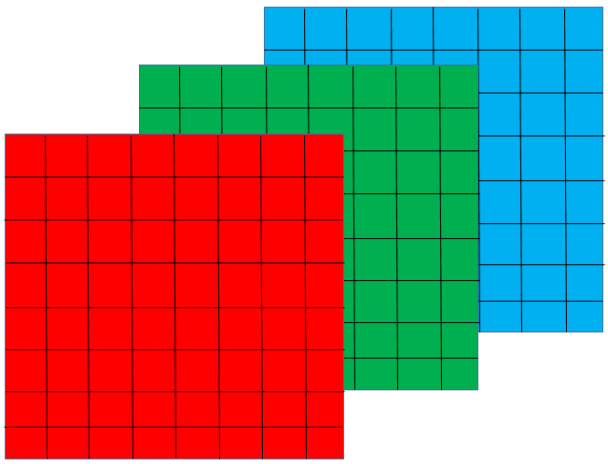


Figure 2.12 – Illustration of channels in an RGB image. Each channel encodes the intensity of its assigned color for each pixel. For example, the red channel assigns high value for pixels where the color red should be high and low values when it should be low. The combination of these three primary colors allows the creation of other colors.

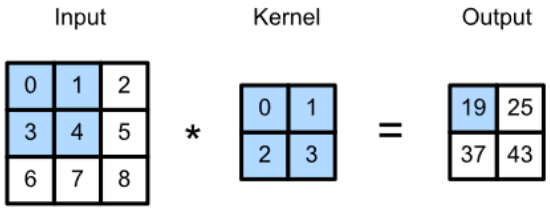


Figure 2.13 – Illustration of the convolution operation.

illustration of the max pooling operation is given in figure 2.14 [8] with a kernel of size $(2, 2)$.

Finally, after successively repeating these operations several times, the flattened output of the last pooling layer is fed into an MLP architecture, described in section 2.3.1.

The activation function used in a CNN architecture also differ form the original MLP as the sigmoid or the $tanh$ functions are placed by a ReLU (Rectified Linear Unit or rectifier) activation function, defined in equation 2.9. The advantages of this function is that it is easier to compute as it only requires a comparison between the input and 0, while the sigmoid and $tanh$ activation functions require the computation of exponential functions and inversions. The second advantage of the ReLU is that it allows to fix the vanishing gradient problem, a phenomenon explained in section 2.2.3. Indeed, the gradient

8. Credit to Aphex34, CC BY-SA 4.0 <https ://creativecommons.org/licenses/by-sa/4.0>, via Wikimedia Commons.

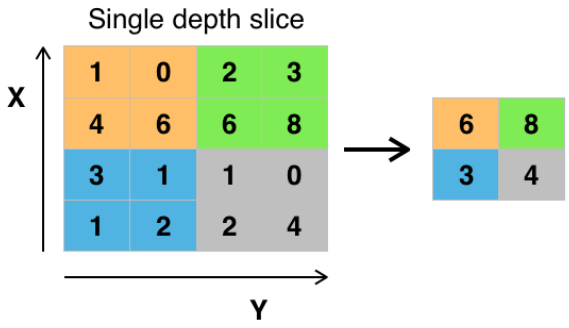


Figure 2.14 – Illustration of the max pooling operation.

of the ReLU function is a Heaviside step function, represented in figure 2.10, which doesn't vanish to 0 for high magnitude inputs. It only outputs a 0 for negative inputs, and 1 for positive inputs. Therefore, the backpropagated gradient will not easily reach 0. The usage of the ReLU is especially important for deeper architectures (more layers stacked) [33].

$$ReLU(x) = max(0, x) \tag{2.9}$$

Some other operations are commonly added to the CNN architecture, such as the batch normalization (batchnorm layer) [34] which have the effect of speeding up training by making the model converge faster. The operation performed by batchnorm layers is illustrated in equation 2.10. In this equation, μ is a mean parameter of the input features of the layer that is computed over the dataset for inference but is calculated over the mini-batch during training, σ^2 is the variance parameter of the input features of the layer on the mini-batch, ϵ is a constant used for numerical stability, γ is the scaling parameter (learned as well) and β is the shifting parameter (learned as well).

$$BN(x) = \gamma \frac{x - \mu}{\sqrt{(\sigma^2 + \epsilon)}} + \beta \tag{2.10}$$

An important contribution of CNNs was the introduction of skip connections which allowed training very deep architectures while avoiding the effect of vanishing or exploding gradients, which was developed in section 2.2.3. The first paper introducing such a mechanism was the Highway network paper [35], which enhanced a CNN architecture with skip connections with gating mechanisms. The skip connection was then further simplified to the skip connection in the ResNet architecture, also called the residual connection (or residual layer) [36]. An abstract illustration of a residual connection is given in figure 2.15 [9]. We see the skip connection sending the output of layer $l - 2$

9. Credit to Jeblad, CC BY-SA 4.0 <https ://creativecommons.org/licenses/by-sa/4.0>, via Wikimedia Commons.

directly to layer l, effectively skipping the transformation in layer l.

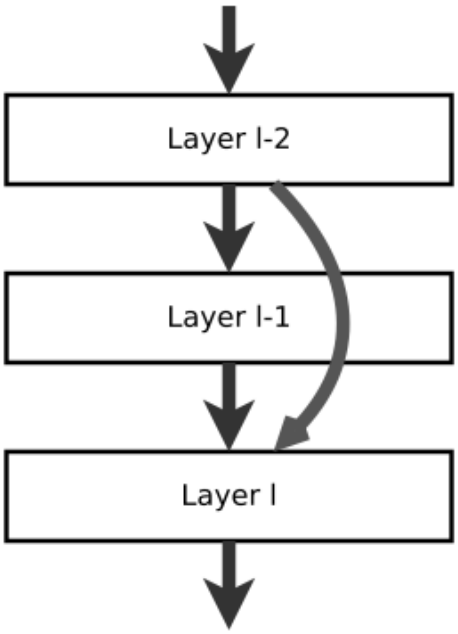


Figure 2.15 – Illustration of the skip connection. layer l-2 directly sends its a copy of its output to layer l, while another copy of this information is also sent to layer l-1.

The transformation performed by a layer of a regular feed-forward neural network is denoted by $y = H(x, W_H)$, where y is the output, x the input, H the non-linear transformation, usually an affine transformation [10] followed by an activation function, parameterized by the weight W_H (omitting the bias for simplicity). The Highway network has an extra mechanism, called a gating mechanism, which adds two non-linear transformations to the traditional operation performed by the usual network : the transform gate T and the carry gate C. Thus, the transformation of a Highway network is turned into equation 2.11. For simplicity, the operation C can be set to be equal to $1 - T$. In this configuration, the two operations clearly appear as erasing information from a give layer to the next one or letting some of the information pass through. A Highway network can smoothly vary its behavior between that of the original transformation H and a simple linear layer which lets all the information pass through with no non-linearity. The transformation described in equation 2.11 forms a Highway block, which can be composed several times.

$$y = H(x, W_H) \times T(x, W_T) + x \times (1 - T(x, W_T)) \tag{2.11}$$

10. A transformation of the form $ax + b$ where a and b are constants and x is the input.

An illustration of the highway block is given in figure 2.16.

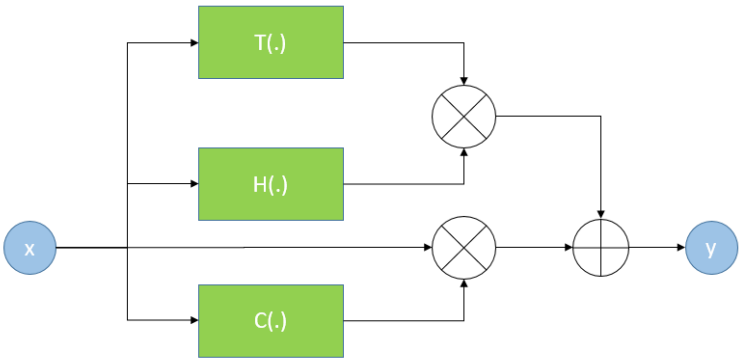


Figure 2.16 – Illustration of the Highway network block. The circled multiplication symbol represents a multiplication operation (element-wise) while the circled sum symbol represents a sum.

2.3.3 . Transformers

Transformers are a widely used family of neural architectures, introduced in 2017 in [37]. Although originally their use was mainly intended for natural language processing, they were quickly adapted to the computer vision domain with the ViT (Vision Transformer) architecture [38]. ViTs in computer vision increased performance several times in different tasks, often beating previous state-of-the-art posed by CNNs. Transformers introduced a drastic change in architecture compared to CNNs and MLPs.

The key idea of a transformer architecture is the attention mechanism [39]. The attention mechanism aims at reproducing the process by which humans focus on objects in images or certain words in a text. The attention vector over an input x is defined in equation 2.12 where q is the query vector, k the key vector, v the value vector, and the function $softmax(\frac{input}{\sqrt{d}})$ is a similarity function, the scaled dot-product. The scalar value d is the dimension of the key vector, used for scaling the softmax dot-product in equation 2.12 for stabilizing the model during training. Keys represent what the model should look for in the input while the queries represent what the model is trying to look for. The value vector corresponds to the input data x.

$$Attention(q, k, v) = softmax(\frac{q \times k^T}{\sqrt{d}}) \times v \tag{2.12}$$

Intuitively, the attention layer computes the similarity between the queries and keys to focus on the parts of the value vector that are the most relevant. Each of these vectors are computed by matrix multiplication of the input data $x : k = W_k x$, $q = W_q x$, $v = W_v x$. The weight matrices W_k, W_q and W_v are the

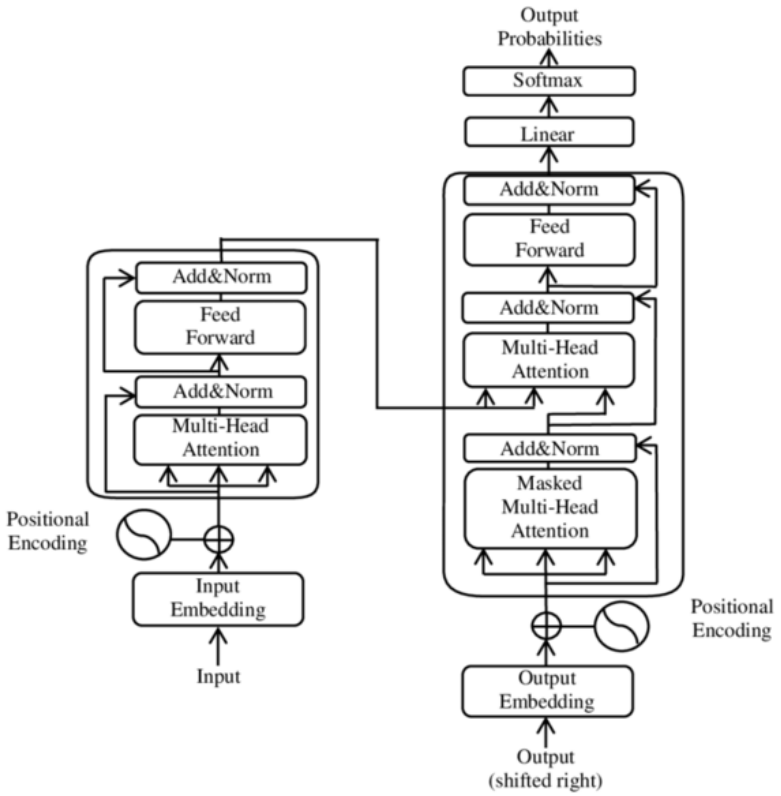


Figure 2.17 – Illustration of the original transformer architecture.

learned parameters of the attention layer. Figure 2.17 [11] shows an illustration of the transformer. The architecture of the model relies on positional embeddings, which indicate to the model the position of each token/element in the input.

Some adaptation of the original transformer can be used to process images instead of text data. The ViT architecture slices the input image into patches that are flattened then fed into the encoder architecture shown in figure 2.18 [12] to obtain a global representation of the image.

The activation function of the MLP component of the ViT architecture is a Gaussian error linear unit GeLU, a smoother version of the ReLU described in section 2.3.2. The model also applies dropout to the output of each layer of the MLP.

Now that I essentially introduced some important general purpose architectures, I will now move on to another category of models, based on these previous models for the most part. In the next sections of this chapter, I will

11. Credit to Yuening Jia, CC BY-SA 3.0 <https ://creativecommons.org/licenses/by-sa/3.0>, via Wikimedia Commons.
12. Credit to Dive into Deep Learning [26].

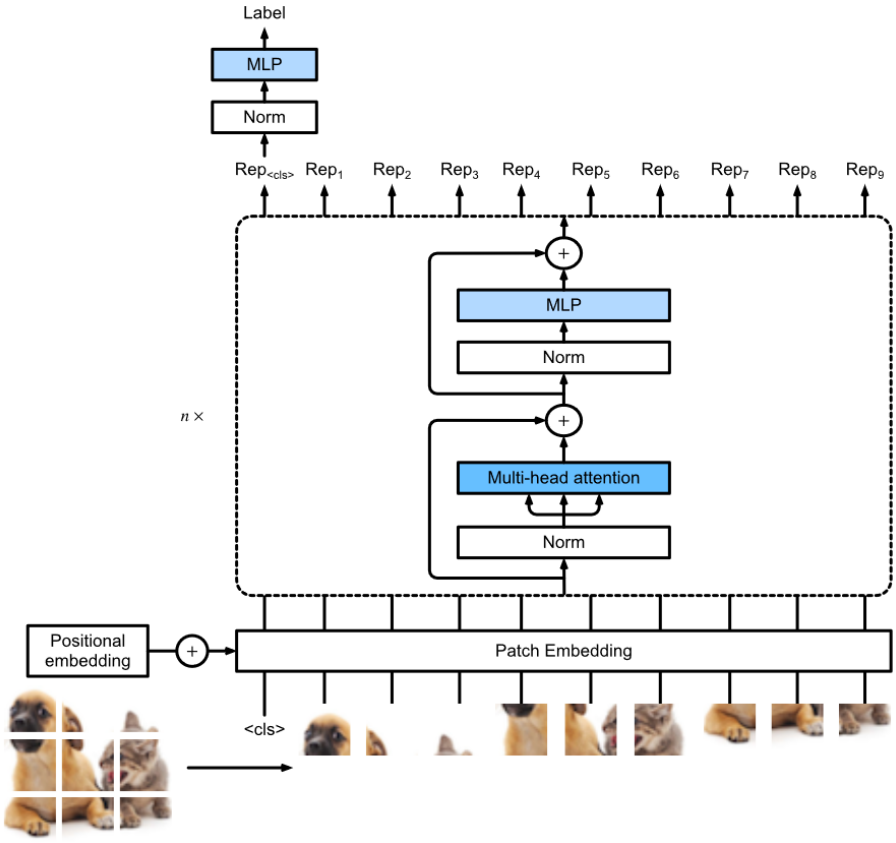


Figure 2.18 – Illustration of the ViT architecture for classification.

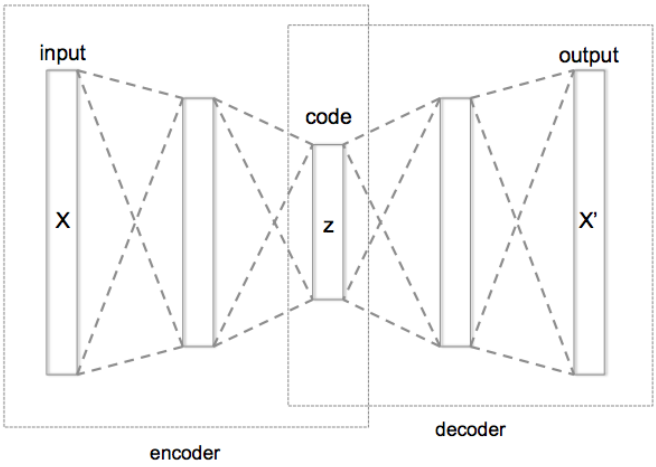


Figure 2.19 – Illustration of the autoencoder architecture.

introduce some important generative models. Generative models aim at modeling the distribution of the features of the data $p(x)$, where x represents the input (for instance, an image). The aim is to ultimately generate data following this same distribution or to model the input. Data generation differs from data classification as the goal in this task is to model the distribution of labels of data conditioned by the features $p(y|x)$ where y is a label assigned to the data point, selected from a set of classes.

2.3.4 . Variational autoencoder (VAE)

Variational autoencoders (VAE) [40], introduced in 2014, are a type of generative model that allows data generation, denoising and representation and compression. The VAE is an adaptation of the autoencoder architecture

An autoencoder (AE) [41] is an architecture that consists in an encoder, which compresses the input data x into a latent code z (a compressed representation) and a decoder which decompresses this latent code to reproduce the original input. Both the encoder and the decoder can be an MLP or a CNN. An illustration of the autoencoder architecture is given in figure 2.19 [13]. Autoencoder are not inherently designed for sampling data points from the training distribution.

In VAEs, the latent space is not deterministic but probabilistic and parameterized to follow a Gaussian distribution parameterized by a latent mean vector z_{mean} and a latent log-variance $z_{log-variance}$. Sampling from the latent space then simply amounts to sampling following the multivariate Normal dis-

13. Credit to Chervinskii, CC BY-SA 4.0 <https ://creativecommons.org/licenses/by-sa/4.0>, via Wikimedia Commons.

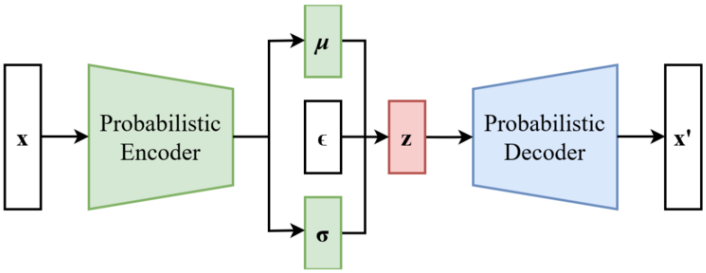


Figure 2.20 – Illustration of the variational autoencoder (VAE) architecture.

tribution with the parameters z_{mean} and
$z_{log-variance}$. The decoder then performs similarly to vanilla AE by mapping the latent code z back to the original space of the input into a vector x' as close as possible to the original point x.

Autoencoders are generally trained with a reconstruction loss, which can be for example the L_2 reconstruction loss defined in equation 2.13 where x is the original input sample from the training dataset and x' is the reconstructed sample as defined in the previous paragraphs.

$$L_{reconstruction}(x, x') = \| x - x' \|_2^2 \tag{2.13}$$

The encoder network in VAEs is represented by a posterior distribution $q_\phi(z|x)$ called the approximate posterior. This distribution is given by a neural network parameterized by the vector ϕ and assigns a latent code z from a given input x. Sampling a latent element z is done by first sampling from the multivariate Gaussian distribution $\epsilon \sim \mathcal{N}(0, I)$ then scaling and shifting this distribution with z_{mean} and $z_{log-variance}$ with the formula in equation 2.14. This allows the parameters $z_{log-variance}$ and z_{mean} to stay differentiable and not be attached to the random variable ϵ. This step is commonly called the reparameterization trick. We note the prior over the latent space $p(z)$.

$$z = z_{log-variance} \times \epsilon + z_{mean} \tag{2.14}$$

The decoder network maps the latent samples z to x', a reconstructed version of the original input x by sampling from the distribution $p_\theta(x|z)$. This distribution is called the likelihood. An illustration of the VAE architecture, slightly different from the autoencoder architecture in figure 2.19, is given in figure 2.20 [14].

In the case of VAEs, the reconstruction loss defined in equation 2.13 is jointly optimized with an extra term, the ELBO (Evidence Lower Bound) loss

14. Credit to EugenioTL, CC BY-SA 4.0 <https ://creativecommons.org/licenses/by-sa/4.0>, via Wikimedia Commons.

which is a Kullback-Leibler (KL) divergence between the distribution of the sampled latent variables and a prior distribution, usually a Gaussian distribution. The KL divergence is defined in equation 2.15. As its name suggests, the KL divergence is a measure of divergence between two distributions.

$$D_{KL}(p||q) = \int_{x \in \mathcal{X}} p(x) \log(\frac{p(x)}{q(x)}) dx \tag{2.15}$$

The ELBO encourages certain properties in the latent space and is equivalent to maximizing the log-likelihood of input data $p_\theta(x)$ while minimizing the divergence between the true posterior distribution of the code z knowing the input x, $p_\theta(z|x)$, with its approximation $q_\phi(z|x)$ (the encoder). The approximation $q_\phi(z|x)$ is important as $p_\theta(z|x)$ is intractable as we only have access to $p_\theta(x|z)$, hence the qualification of this autoencoder as "variational". The ELBO is defined in equation 2.16. Both the reconstruction loss and the ELBO are optimized during training with respect to the parameters θ and ϕ.

$$L_{ELBO}(x, \theta, \phi) = \mathbb{E}_{z \sim q_\phi(.|x)}(\frac{q_\phi(z|x)}{p_\theta(z|x)}) = -\log p_\theta(x) + D_{KL}(q_\phi(.|x)||p_\theta(.|x)) \tag{2.16}$$

2.3.5 . Normalizing flows

Similarly to VAEs, a normalizing flow [42] [43] maps an input x to a latent space z, also called base space [44] to sample data from the distribution of the input noted $p(x)$ and compute its exact values. Its architecture is similar to the VAE although the dimension of the latent space is the same as the ambient space (the input space). It relies on the change of variable formula from probability theory, written in equation 2.17 where $p_X(x)$ is the distribution of the input data, $p_Z(z)$ is the distribution of the latent space and f is a diffeomorphism (invertible and differentiable transformation) of inverse f^{-1} mapping values of the random variable X to the random variable Z : $f(x) = z$.

$$p_X(x) = p_Z(f(x)) \times |det(\frac{df}{dx})| \tag{2.17}$$

In the formulation of normalizing flows, the distribution p_Z is generally fixed to a Gaussian distribution of mean 0 and unit variance, similarly to the VAE, although the mean and variance parameters can be learned as well. The invertible transformation, which we note f_θ, is the key component of the normalizing flow model. It has to be both invertible and differentiable in order for the change of variable theorem to be applicable, expressive enough to successfully transform the variable $x \sim p_X(x)$ into the variable $z \sim p_Z(z)$, and the determinant of its Jacobian (the term $|det(\frac{df}{dx})|$ in equation 2.17) must be tractable for fast inference and training. The transformation may be expressed as the composition of several simpler transformations whose Jacobians

have tractable determinant. Composing successively several of these transformations may progressively make the transformation more expressive as the number of transformations increases. The determinant of a composition of functions $f_1 ... f_n$ is given in equation 2.18, where each function f_{i+1} receives as input the output of $z_i = f_i(z_{i-1})$, with $z_0 = x$.

$$det(\frac{d(f_1 \circ f_2 \circ ... \circ f_n)(x)}{dx}) = det(\frac{df_1(x)}{dx}) \times det(\frac{df_2(z_1)}{dx}) \times ... \times det(\frac{df_n(z_{n-1})}{dx}) \tag{2.18}$$

These formulas in equations 2.17 and 2.18 are often manipulated in logarithmic form in order to transform products into sums, thus making computations easier.

Once the transformation f_θ and the base transformation p_Z are defined, the normalizing flow is learned by minimizing the KL divergence between the estimated log-likelihood $\log p_\theta$ and the true log-likelihood $\log p_X$, written in equation 2.19.

$$D_{KL}(p(x)||p_\theta(x)) = \mathbb{E}_{x \sim p_X}(\log p_\theta(x)) - \mathbb{E}_{x \sim p_X}(\log p_X(x)) \tag{2.19}$$

Maximizing the KL divergence is equivalent to maximizing the first term, $\mathcal{E}_{x \sim p_X}(p_{theta}(x))$, which is equivalent to a maximum likelihood objective. However this objective is intractable, therefore it is replaced by an approximate objective by Monte-Carlo computation, defined in equation 2.20 where the x_i are true samples drawn from the distribution p_X.

$$\mathbb{E}_{x \sim p_X}(\log p_\theta(x)) \approx \frac{1}{N} \sum_{i=1}^{N} \log p_\theta(x_i) \tag{2.20}$$

The training objective of a normalizing flow is therefore defined in expression 2.21.

$$\max_\theta \sum_{i=1}^{N} \log p_\theta(x_i) \tag{2.21}$$

An illustration of the general scheme of a normalizing flow is given in figure 2.21[15]. In this figure, we can see that an input (represented by a signal) is successively transformed by each function composing the flow until it is transformed into a base signal with a simpler distribution such as a Gaussian distribution.

Some relaxations can be given however on the bijectivity of the transformation f_θ [45]. A broader conception of the normalizing flow family can be

15. Credit to janosh, MIT <http ://opensource.org/licenses/mit-license.php>, via Wikimedia Commons.

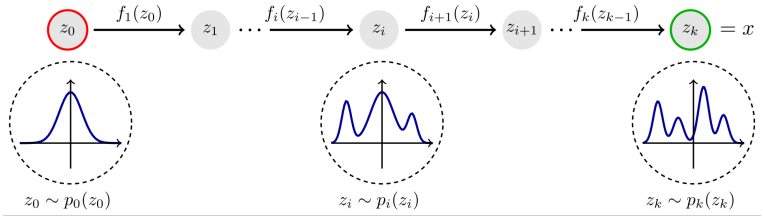


Figure 2.21 – Illustration of the principle of a normalizing flow.

described to turn neural architectures using fully-connected layers, convolutional layers and some activation functions into a normalizing flow [46].

From this general description, several models can be defined. They generally use neural networks combined in some invertible operations in order to obtain expressive and differentiable transformations f_θ while staying invertible. We will now see some normalizing flows architectures, especially those that will be important for the rest of the manuscript.

The RealNVP (Non-Volume Preserving) model [47] is a normalizing flow model with an invertible transformation defined by the elementary transformation in equation 2.22. This transformation is defined over an input x, split into two parts x_{id} and x_{transf} such that $x = \begin{pmatrix} x_{id} \\ x_{transf} \end{pmatrix}$. In equation 2.22, the $\odot$ operator represents the Hadamard or element-wise product. This type of elementary transformation is called a coupling layer.

$$\begin{pmatrix} y_{id} \\ y_{transf} \end{pmatrix} = f_\theta(\begin{pmatrix} x_{id} \\ x_{transf} \end{pmatrix}) = \begin{pmatrix} x_{id} \\ x_{transf} \odot e^{s_\theta(x_{id})} + t_\theta(x_{id}) \end{pmatrix} \tag{2.22}$$

The transformation in equation 2.22 can be reversed like in equation 2.23. Its Jacobian determinant is given in equation 2.24 where L is the number of coupling layers in the model.

$$\begin{pmatrix} x_{id} \\ x_{transf} \end{pmatrix} = f_\theta^{-1}(\begin{pmatrix} y_{id} \\ y_{transf} \end{pmatrix}) = \begin{pmatrix} y_{id} \\ (y_{transf} - t_\theta(y_{id}) \odot e^{-s_\theta(y_{id})} \end{pmatrix} \tag{2.23}$$

$$det(\frac{df_\theta}{dx}) = \prod_{l=1}^{L} e^{s_\theta(z_{l-1})} \tag{2.24}$$

The transformation defined by the RealNVP model (in equation 2.22) makes use of two neural networks s_θ and t_θ, which are called the st-network (scaling and translation networks). The choice of these networks does not influence the invertibility of the transformation although they are generally chosen to be differentiable [16]. Generally, the s and t networks are chosen to be identi-

16. Operators such as the rounding operator, used in quantization, is non-differentiable although some approximation can be made.

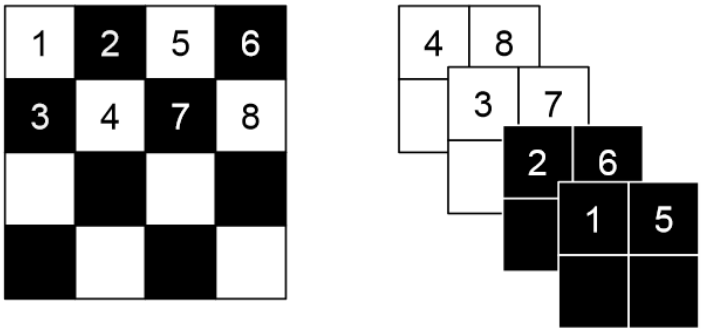


Figure 2.22 – Illustration of the squeezing operation from the original paper [47]. On the left the checkerboard mask, on the right the channel-wise mask.

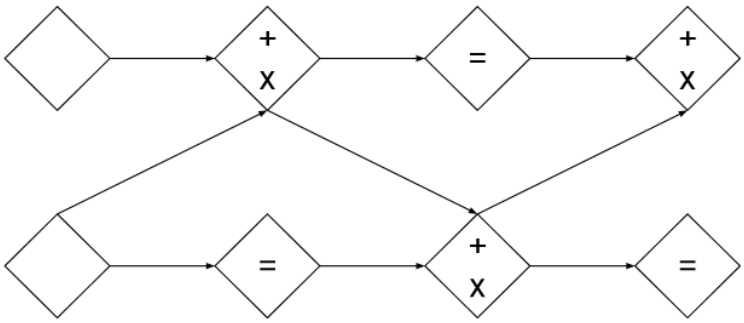


Figure 2.23 – Illustration of the RealNVP model from the original paper [47]. On the left, the two blank squares represent the input split into two parts. They are transformed following the rest of the illustration. the "=" sign signifies that the input of the block is left unchanged.

cal. Finally, the coupling layers are generally stacked, by alternating at each layer the part of the input which will be transformed and which will remain identical.

In order to capture information at different scales in the input, the RealNVP model introduces a multi-scale architecture, where at each scale the model combines three coupling layers with alternating checkerboard masks, then a squeezing operation where the height and width of the input is halved while quadrupling the channel dimension, before finally applying another set of three coupling layers with alternating channel-wise masks this time. Figure 2.22 illustrates the two types of masks, namely checkerboard and channel-wise masks, used in the multi-scale architecture as well as the squeezing operation where an $s \times s \times c$ image is turned into an $\frac{s}{2} \times \frac{s}{2} \times 4c$ image.

An illustration of the RealNVP architecture is given in figure 2.23.

This model is the one I chose to use in my experiments in the next chap-

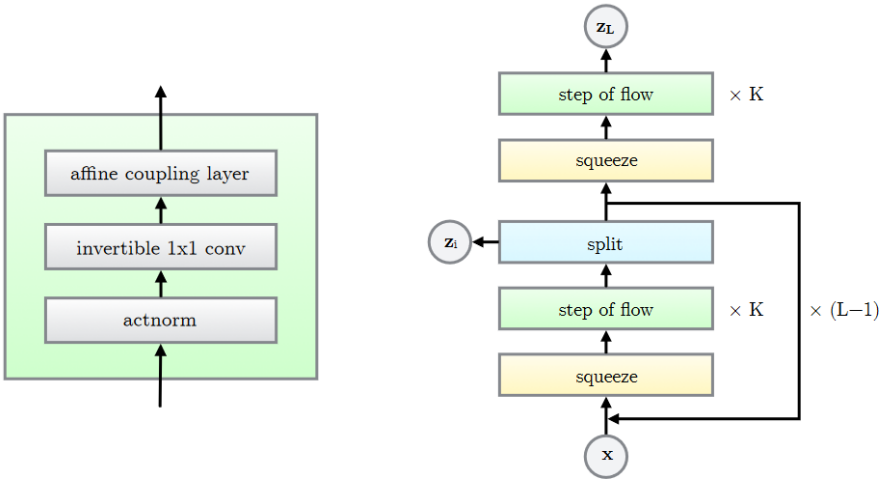


Figure 2.24 – Illustration of the Glow model from the original paper [49].

ters as it gives enough stability during training to train a large model, as well as enough representation power to model input data. Furthermore, previous papers like [48] use this model in their experiments, therefore it would make a fair comparison.

The Glow model (Generative flow) [49] is an evolution of the RealNVP model that has a coupling layer made of an actnorm layer (normalization over the activations, the inputs) followed by an invertible 1×1 convolution layer then an affine coupling layer defined in equation 2.22. The Glow architecture simplifies however the RealNVP architecture by removing the checkerboard mask operation and keeping only the channel-wise one.

A complete representation of the architecture is given in figure 2.24.

This model is important as it shows an evolution of the already existing RealNVP architecture but is hard to train as the model often explode during training.

It is also possible to formulate normalizing flows by a continuous dynamic instead of a composition of functions [50], with the transformation defined in equation 2.25 where $f(z_t, t)$ is a continuous map depending on a time parameter t, z_T is the equivalent of the latent variable in discrete composition flows (defined above) and z_0 the input data point x.

$$z_T = F(z_0) = z_0 + \int_0^T f(z_t, t)dt \tag{2.25}$$

Finally, normalizing flows can be used outside of unsupervised learning for generative modeling as hybrid models, such as with the DIGLM (Deep Invertible Generalized Linear Model) [1], used for supervised learning in classification tasks. The DIGLM uses a normalizing flow as a backbone and adds to it a generalized linear model [51] that takes as input the values of the latent space

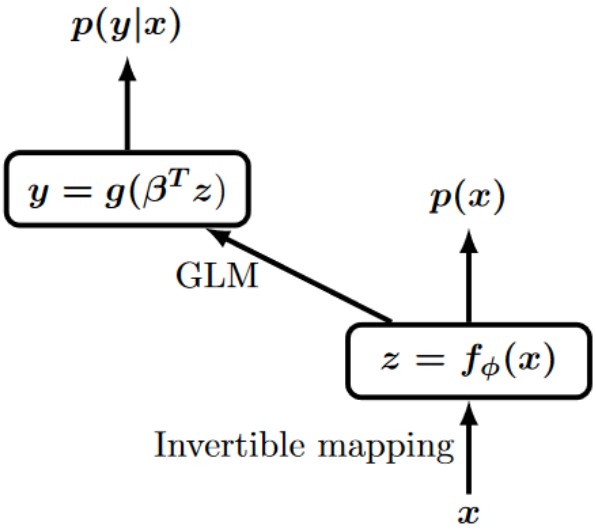


Figure 2.25 – Illustration of the DIGLM model from the original paper [1].

to output softmax probabilities for classification. A generalized linear model is a simple linear regression with a "link function". A general linear model can be described with equation 2.26 where y is the label, z the latent variable from the normalizing flow, g^{-1} the link function and β the parameter of the model. I used this model in my experiments in chapter 4.

$$\mathbb{E}(y|z) = g^{-1}(\beta z) \tag{2.26}$$

Therefore, the model can output both the probability distribution $p(x)$ (input distribution) through the generative model part and the probability $p(y|x)$ (label distribution conditionally to the input) through the linear model and the softmax function. An illustration of the model is given in figure 2.25.

2.3.6 . Score-based models

Score-based models are another family of generative models that focus on estimating the gradient of the log-likelihood of the distribution we wish to sample from, that is to say $\nabla_x \log p(x)$ [52] [53] instead of estimating the log-likelihood directly or via a surrogate. This model is important as it helped define later on denoising diffusion models, defined in section 2.3.7. They are also important in this manuscript as the underlying principles of score-based models are used in chapter 4.

The gradient $\nabla_x \log p(x)$ is coined the "score" and can be directly used for data generation, typically with Langevin sampling. It is a sampling technique similar to a gradient ascent described by the dynamic in equation 2.27 where η is a fixed parameter and $z_t \sim \mathcal{N}(0, I)$ is a random Gaussian noise. The dynamic can be initialized with a random point x_0. It models a system evolving

when subjected to deterministic and random forces, such as the dynamics of grains of pollen put atop of a water surface, where the pollen interacts with water molecules (Brownian motion, mathematically described by Albert Einstein in [54]).

$$x_{t+1} = x_t + \frac{\eta}{2}\nabla_x \log p(x) + \sqrt{\eta} z_t \tag{2.27}$$

Langevin sampling has some properties that makes sampling with this technique interesting, among which the fact that asymptotically (when the number of time steps of the Langevin dynamic reaches infinity), the samples generated by this process follow the distribution $p(x)$. Also, having direct access to the distribution $p(x)$ can be hard, as some methods only give an estimation up to a multiplicative constant, such as with energy-based models [55]. For example, in RBM (Restricted Boltzmann Machines) [56] [57], the joint probability of an observation x and the hidden state of the model h is given by equation 2.28 where $E(x, h)$ is an energy function evaluated at the points x and h and Z is a normalization constant such that the sum over all possible states (x, h) of the probability distribution is 1.

$$p(x, h) = \frac{e^{-E(x,h)}}{Z} \tag{2.28}$$

This constant Z can be intractable, making the computation of $p(x, h)$ hard. However, the score of the model $\nabla_x \log p(x, h)$ is easy to compute as long as the energy function can be differentiated since $\nabla_x \log p(x, h)$
$= -\nabla_x \log E(x, h)$.

Score-based generative models are reckoned mainly for their applications in diffusion models which are generative models that successfully beat GANs (described in section 2.3.8) in the recent years especially in the domain of image generation where they produce highly realistic data samples.

2.3.7 . Diffusion models

Diffusion models [58] are models based on statistical physics where the aim is to learn a diffusion process which is a process that progressively adds noise to an image. The noise repeatedly added to the input is a Gaussian noise $\mathcal{N}(0, I)$ making the portion of the space of naturally occurring images (we wish to model) progressively "diffuse" to the rest of the space of images (even noisy ones). This final distribution, the equilibrium distribution, is indistinguishable from Gaussian noise. Therefore, a model that can cancel this diffusion process can be used to sample from the original space of images of interest. This reverse diffusion process relies on score matching methods.

Denoising diffusion probabilistic models [59] are especially regarded as the method that greatly improved on existing diffusion models. The forward diffusion process, which progressively adds noise to the image x_0 at each time

step t producing a new noisy version x_t, is defined with the dynamic in equation 2.29 where $\beta_1 ... \beta_T$ are fixed parameters in $]0, 1[$ and $z_t \sim \mathcal{N}(0, I)$.

$$x_{t+1} = \sqrt{1 - \beta_t} x_t + \sqrt{\beta_t} z_t \tag{2.29}$$

At step t, the distribution of the image x_t conditionally to x_0 is $x_t | x_0 \sim \mathcal{N}(\sqrt{\bar{\alpha}_t} x_0, \sqrt{1 - \bar{\alpha}_t} x_0)$ where $\bar{\alpha}_t = \prod_{i=1}^{t} (1 - \beta_i)$, which stays a normal distribution. Then, the backward diffusion process, which seeks to cancel the forward diffusion process described above, estimates the mean and covariance parameters of the Gaussian such that the forward diffusion can be approximately cancelled by $x_{t-1} \sim \mathcal{N}(\mu_\theta(x_t, t), \Sigma_\theta(x_t, t))$. Here, $\mu_\theta(x_t, t)$ and $\Sigma_\theta(x_t, t)$ are the mean and covariance parameters given by a neural network parameterized by θ and given the noisy input x_t and the time step t as inputs. This yields a backward process defined by $p_\theta(x_T) = \mathcal{N}(x_T; 0, I)$ and $p_\theta(x_{t-1} | x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t))$.

Knowing these two processes, the goal of the DDPM is to train a neural network with parameters θ such that $p_\theta(x_0)$ is as close as possible to the true distribution of the original sample which we note $p(x_0)$. This is done through variational inference, similarly to the training mode in VAEs in section 2.3.4. The loss function used to train a diffusion model is described in equation 2.30 where $x_{1:T}$ is the sequence of elements x_t for t varying between 1 ans T (the length of the diffusion sequence). This loss term translates the discrepancy between the original diffusion process $p(x_{1:T} | x_0)$.

$$L(\theta, (x_t)_{t=0:T}) = \mathbb{E}_{x_{0:T} \sim p(x)} (\log(p_\theta(x_{0:T})) - \log(p(x_{1:T} | x_0))) \tag{2.30}$$

This loss is optimized by stochastic gradient descent, described in section 2.2.2. This loss can be further simplified for training on a dataset with the expression in equation 2.31.

$$L(\theta, (x_t)_{t=0:T}) = -\sum_{t=1}^{T} \mathbb{E}_{x_{t-1}, x_t \sim p} \log p_\theta(x_{t-1} | x_t) \tag{2.31}$$

DDPM and score-based generative models are equivalent [60] and one can see that both methods are in fact closely related, especially since both methods use a stochastic iterative sampling method.

An illustration of the principle of the DDPM is given in figure 2.26.

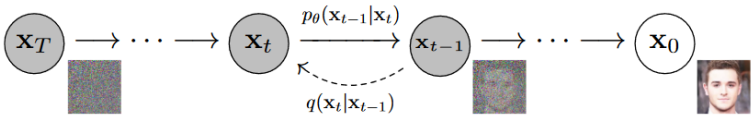


Figure 2.26 – Illustration of the DDPM from the original paper [59]. The diffusion process q iteratively adds noise to the image on the right to yield a Gaussian random noise on the left at the end of its diffusion. The denoising diffusion process p iteratively inverts this process to yield an image close to the one from the original space.

2.3.8 . Generative adversarial networks (GAN)

Generative adversarial networks or GANs [61] are a type of generative model that are based on two neural networks that compete against each other in a zero-sum game in order to be trained. One of them is called the generator, whose goal is to generate data as close as possible to the original data distribution, the other is the discriminator whose goal is to discriminate images from the original distribution and and images from the generator. Hence the name "adversarial".

The objective used to train both networks is the loss described in equation 2.32. In this equation, G refers to the generator network, D to the discriminator and p to the reference distribution the network G aims to imitate. The label y assigned to the input x by the discriminator is equal to 1 if x is real or 0 if it is fake. This loss is supposed to be minimized by the generator and maximized by the discriminator, leading to the min-max game $\min_G \max_D L(G, D, p)$.

$$L(G, D, p) = \mathbb{E}_{x\sim p, y\sim D(x)}(\log y) + \mathbb{E}_{x\sim G, y\sim D(x)}(\log 1 - y) \tag{2.32}$$

As opposed to previously introduced generative models, GANs do not explicitly train the likelihood of the model or try to optimize some proxy of the probability distribution. Instead, they use a min-max optimization objective to indirectly train a generator to fool a discriminator, which is also trained to distinguish fake data from real data. This approach is hoped to yield good samples.

In practice, the generator takes as input some noise from a known fixed random distribution one can sample from (for example, Gaussian) and composes it with the generator network : sample $z \sim p_Z$ and generate the new samples with $G(z)$.

Several issues arise when training GANs, among which the convergence of training which may be unstable [62] leading to no solution for the optimization objective defined above, and mode collapse [61] [63] where a GAN trained on diverse instances in a training set fails to catch all the modes of the distribution and may only generate images from one mode (one class of the dataset).

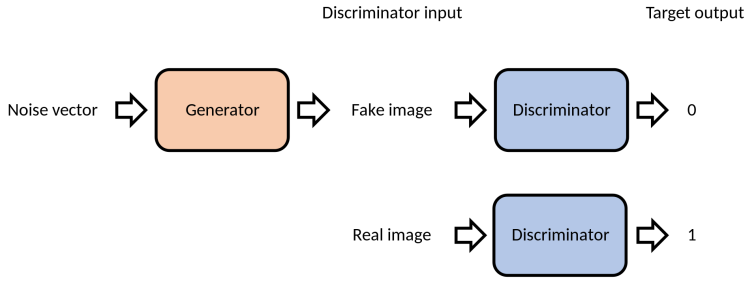


Figure 2.27 – Illustration of the GAN model with the discriminator on the right and the generator on the left.

I provide in figure 2.27 [17] an illustration of a GAN model.

2.4 . Memory-augmented models and associative memories

In this last section, I will introduce a field of machine learning that uses memory structures as a basis or as an enhancement of machine learning architectures. The interest of such models is that they provide a way that may help to understand the construction of modes around training data in probabilistic models like normalizing flows, previously introduced. These models based on memory also have easy interpretations and can be used to make models more robust as some of them can be used in reconstructing corrupted inputs by finding a pattern in their memory that is the closest to a given corrupted input.

The models introduced in this last section will not be used in the rest of the manuscript in scientific contributions. However, they will be used in the conclusion as a way to open perspectives for future research. They will give insights on how the thesis was conceived. Chapters 3 and 4 cover different aspects of the problem of robustness. Chapter 4 in particular covers an idea to restore corrupted input data that was partly inspired by the mechanism of pattern restoration in hyperdimensional computing, which will be introduced in section 2.4.1.

These memory-based models can be traced back to Hopfield networks, which is a model we will introduce in section 2.4.2. The concept of adding memory to a model also gained more importance in the recent years through several works wee will cover in section 2.4.3. In this chapter, we separate two types of memories, one being the address-based memory and the other one being the content-addressable memory (CAM). The address-based memory is the traditional memory type we are used to manipulating in a computer for example where objects and numbers are stored at a given address. In this

17. Credit to Mtanti, CC BY-SA 4.0 <https ://creativecommons.org/licenses/by-sa/4.0>, via Wikimedia Commons.

case, the address space (the set of possible addresses) is distinct from the input space (the set of values that can be processed). Content-addressable memory or associative memory [5] on the other hand is a type of memory where stored objects can also act as addresses themselves. In associative memories, addressing an element given an address is an iterative process where at each step a similarity measure is used to find the nearest element of the address. The most similar element then becomes the new address that is used to find another close element. The algorithm converges once the iterative process stabilizes, that is to say once the addressed element is equal to the address. Associative memory is often used in applications where the goal is to restore a corrupted or partially masked input. Restoring the input is equivalent to finding the most similar pattern, given the input query.

2.4.1 . Hyperdimensional computing

Hyperdimensional computing (HDC) [64] [65] is a paradigm of computation where objects such as numbers, categories or features, are represented as high-dimensional vectors called hypervectors. These vectors are often randomly drawn from a binary space with a high dimension (generally above 10,000 dimensions) and each of them represent a given symbol.

The idea behind the use of a high-dimensional space for representing objects is that any object can be associated with any given random vector. These vectors being high-dimensional and random, they can be expected to be distant from each other on average. This allows for separation of the vectors, and thus the underlying object being represented. This step of assigning a random vector to a given object is called the symbolic encoding step. The use of randomness in assigning a vector to an object allows the vector representation to be approximately orthogonal between each other as a random binary vector in a high-dimensional space is sparse and is therefore likely to have 0s in places where another randomly drawn vector has 1s. This enables the representation of distinct objects as orthogonal vectors, the goal being to have similar objects having similar representations and dissimilar objects having dissimilar representations. The interest of embedding these objects in a vector space comes from the fact that these objects can then be manipulated following some algebraic rules to combine them with each other.

There are two fundamental operations described in HDC. The first one, the binding operation, combines vectors between them in order to form a new vector with a meaning being the result of the composition of the underlying concepts of the composed vectors. For example, combining the words "King - Man + Woman" results in the concept "Queen". An illustration of such an embedding can be found in illustration 2.28 [18].

18. Credit to Singerep, CC BY-SA 4.0 <https ://creativecommons.org/licenses/by-sa/4.0>, via Wikimedia Commons.

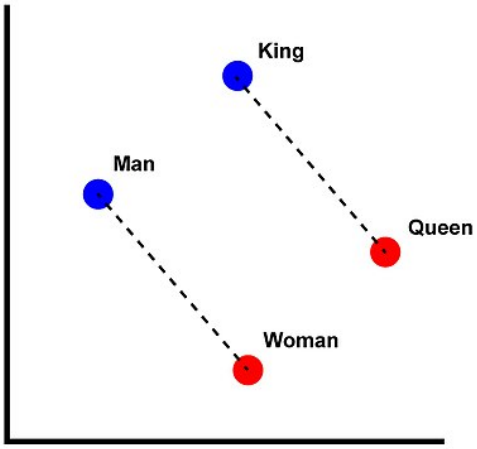


Figure 2.28 – Illustration of embeddings of words, similar to a Word2Vec embedding [66]. Vectors, representing the semantic meaning of words such as "King" or "Woman" can be combined to form new concepts.

The second operation in HDC is superposition. This operation represents the simultaneous occurrence of two concepts. For example, superposing the vectors for "Queen" and "King" would result in a vector that represents the union of both concepts. Superposition captures mostly the co-occurrence of objects while the binding operation captures the relation between these objects.

Hypervectors are often interpreted as associative memories as it is possible, given a query hypervector, to find the closest hypervector representing through similarity search. The query vector may have no particular meaning but would be close to one of the embeddings. This interpretation of hypervectors enables the implementation of error correcting codes [67]. They are also associated with the vector symbolic architecture (VSA) [65] which is a framework where objects are represented in a distributed way, like in HDC, making the representation more robust by distributing information across several components of the representation.

HDC and VSA approaches are methods that are seen as bridging the gap between symbolic and connectionist paradigms. Indeed, their use of symbols to represent objects in order to manipulate them in a semantically meaning-

ful way makes them obviously symbolic AI approaches. Furthermore, the way HDC represents information in a distributed way makes it an approach that is also close to how neural networks distribute information in their representation across different layers, composed of activation patterns. HDC also takes its roots in how brains perform computations [67] by working with associativity and composition of symbols on a distributed way, making the brain fault tolerant.

The associative memory property of the HDC paradigm also manages to create a memory module that is implicit in the architecture and the representation of the represented objects. This is opposed to models we will see in section 2.4.3 that use an external dedicated memory module to store information about dependence between data points.

HDC is close in principle to the Sparse Distributed Memory, or SDM, model [68] which is a long-term memory model that works as an associative memory/CAM. It is very similar to HDC as it uses high-dimensional binary addresses to efficiently store and retrieve data patterns. HDC can be seen as a broader framework than SDM, with the difference that SDM requires sparse high-dimensional vectors to store patterns.

2.4.2 . Hopfield networks

A Hopfield network [69] is a type of neural network inspired by physics. It is made of binary neurons/units connected with each other where the connections are parameterized by a symmetric weight matrix W which represent the strength of the connection between every pairs of units. An illustration of a Hopfield network is given in figure 2.29 [19].

"Training" a Hopfield network is a process that allows the storage of patterns in the network's weight matrix. The weight matrix is set to the value in equation 2.33, where N is the number of units in the network, P the number of patterns to store and V_i^p is the i^{th} component of the p^{th} pattern.

$$W_{i,j} = \frac{1}{N} \sum_{p=1}^{P} V_i^p V_j^p \tag{2.33}$$

An energy function is associated with the network. The energy function is defined in equation 2.34 where s_i is the state of the i^{th} unit, $W_{i,j}$ the element of the weight matrix W representing the strength between units i and j.

$$E = -\frac{1}{2} \sum_{(i,j),i \neq j} W_{i,j} s_i s_j - \sum_i \theta_i s_i \tag{2.34}$$

This energy function is updated through the update rule given in equation 2.35 when a change in the state s_i of the unit i of the network occurs. The

19. Credit to Zawersh, CC BY-SA 3.0 <https ://creativecommons.org/licenses/by-sa/3.0>, via Wikimedia Commons.

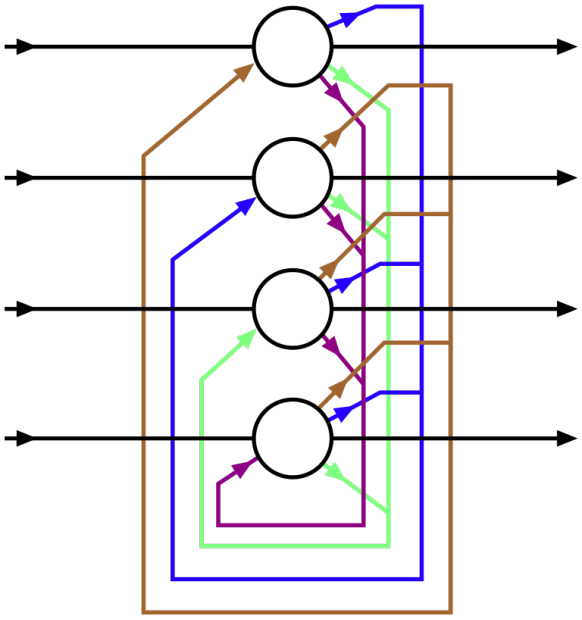


Figure 2.29 – Illustration of a Hopfield network.

aim for the network is to minimize this energy function. A low energy function is associated with a stable state of the overall network and therefore corresponds to a stored pattern in the memory of the network.

$$\Delta E = -\Delta s_i \sum_{(i,j),i \neq j} W_{i,j} s_j \tag{2.35}$$

The state of the units of the network are updated according to the state of each unit and the connection weights through an update rule inspired by Hebbian learning [70]. This update rule is shown in equation 2.36 where θ_i is a threshold value associated with the i^{th} neuron. This update allows the network to converge to a new state associated with a new energy, corresponding to a stored pattern.

$$s_i = \begin{cases} 1 & \text{if } \sum_j W_{i,j} s_j \geq \theta_j \\ -1 & \text{otherwise} \end{cases} \tag{2.36}$$

Hopfield networks are mostly used for storing patterns, thus forming a content-addressable memory. A pattern can be retrieved in the memory of the network by updating the state of the network following equation 2.36 until the energy defined in equation 2.34 is minimized, corresponding to a stable state. Hopfield networks can also be used in combinatorial optimization problems, such as the travelling salesman problem, where, given a list of cities

and the distances between each pair of cities, find the shortest route that visits each city exactly once and returns to the starting point.

Hopfield networks have recently gained more popularity, especially since some equivalence can be found between them and attention layers [71]. This new model relies on an extension of Hopfield networks to a continuous case with dense associative memories, also called modern Hopfield networks [72] [73]. These extensions of Hopfield networks use a new energy function in order to increase the storage capacity of the model. The general form of this energy function can be found in equation 2.37 where N is the number of patterns to store, x is the patterns to store and ξ is the state of the network, written in vector form for convenience. The function F, called an interaction function, can take several form but is assigned to a polynomial in [72] and an exponential in [73].

$$E = -\sum_{i}^{N} F(x_i^T \xi) \tag{2.37}$$

The work in [71] showing how a Hopfield model can be made equivalent to an attention layer led to the development of Hopfield layers, which can be used in existing models for applications in multiple instance learning, set-based and permutation invariant learning, and associative learning.

2.4.3 . Memory-augmented neural models

In this section, we will cover some memory-augmented models that rely on an external memory module in order to enhance the capacity of neural architectures and enable dynamic storage and retrieval of information during inference for example.

Neural Turing Machine

The Neural Turing Machine or NTM [74] is a model equipped with a controller, which can be a feed-forward neural network, introduced in section 2.3.1 or an LSTM network [75], that processes inputs to interact with an external memory, which is a large storage matrix that can be read from and written to by the controller through differentiable read and write operations. Figure 2.30 illustrates the NTM architecture. The architecture is similar to a Turing Machine because of the read and write heads as well as a Von Neumann architecture for its external memory module that the controller (similar to an arithmetic and logic unit) interacts with.

Each component of the NTM being differentiable, it can be trained through backpropagation. The training of the model minimizes the difference between the NTM's predicted outputs and the ground truth in order to optimize the memory matrix as well as make the controller better at manipulating the memory.

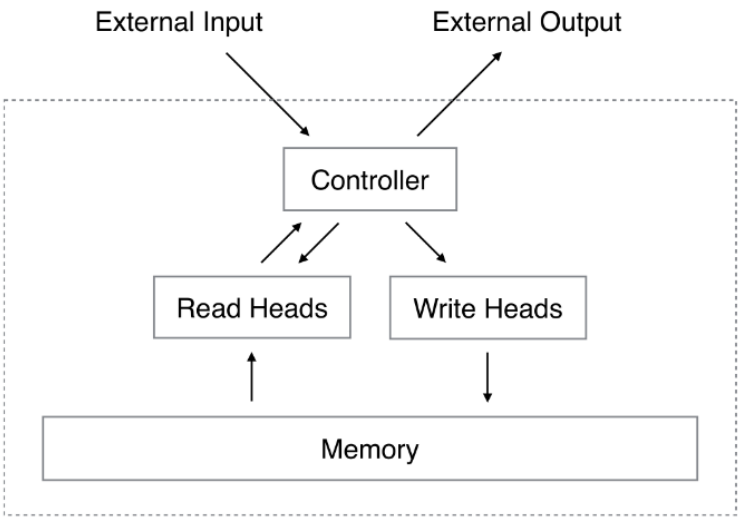


Figure 2.30 – Illustration of the NTM architecture from the original paper [74].

This model is especially interesting in tasks where the model needs to use algorithmic reasoning or dynamic data processing. It can be applied to several tasks such as natural language processing, computer vision or or sequence processing. The memory module allows the model to memorize patterns in the data distribution in order to better inform its decisions. Thanks to the dynamic memory module the NTM is adaptable during inference : the model can store information in the matrix during inference as part of the processing of the inputs to retrieve them later, memory having an important role in reasoning in the human brain. The model can also perform well on noisy or masked inputs thanks to the possibility of finding similar patterns in memory.

Kanerva machine

The Kanerva machine [76] is a model based on the Kanerva's SDM model, mentioned in section 2.4.1. It is a model that is similar in principle to the NTM as it works with an external memory, but this time for data generation. It is composed of three parts : the generative model, the reading inference model (to read from the memory), and the writing inference model (to write to the memory). The generative model generates data X using information provided by a memory matrix M which is a global latent variable that captures statistics about the dataset $x_1...x_T$, and local latent variables y_t and z_t that capture information about x_t, with the joint distribution provided in equation 2.38. In this equation, θ represents the parameters of the model, x_t represents an instance of the dataset, composed of T samples and y_t and z_t are the latent variable corresponding to the observed variables x_t.

$$p_\theta(X, Y, Z|M) = \prod_{t=1}^{T} p_\theta(x_t, y_t, z_t|M) = \prod_{t=1}^{T} p_\theta(x_t|z_t)p_\theta(z_t|y_t, M)p_\theta(y_t) \quad (2.38)$$

The generative model is a VAE model (introduced in section 2.3.4) with a prior $p(z)$ on the latent code z and a conditional distribution $p_\theta(x|z)$. The inference part of the model is the approximate posterior $q_\phi(z|x)$. The training objective of the Kanerva machine here is to maximize the mutual information between the memory matrix M and the input data X to store.

The memory M is a random matrix under which the distribution X, Y, Z is independent between each samples of the episode. The addresses are stored in an address matrix A, optimized through backpropagation as well. The addressing latent variable y_t corresponds to an addressing variable with prior $p_\theta(y_t)$, an isotropic Gaussian distribution. This latent variable is projected with a mapping f (an MLP) to yield a key vector $b_t = f(y_t)$ which allows to compute a weight vector $w_t = b_t^T A$.

The code z_t is a representation of x_t that allows to generate samples with the distribution $p_\theta(x_t|z_t)$. The distribution of z_t itself depends both on the memory matrix M and the latent variable y_t through the weight w_t.

Sampling from the model requires first sampling the memory M once, then for each data samples, sampling latent variables y_t and z_t, and then sampling x_t.

The reading inference model is a conditional distribution, defined in equation 2.39 where ϕ represents the parameters of the model. For a given input x_t in the dataset, it provides the corresponding latent variables y_t and z_t by reading from the given memory matrix M. Kanerva's SDM uses an iterative reading mechanism to read samples from the memory, which iteratively feeds back into the model the previous sample. Kanerva proved that this dynamic decreases error when the initial error is within a certain range, thus creating a stored memory asymptotically. The authors here propose a Gibbs-like sampling which also iteratively reconstructs data through a feedback loop. This process relies the computation of $q_\phi(y_t|x_t, M)$ for addressing, which can be costly during training. It is however known in coding theory that such an intractable posterior can be approximated efficiently with loopy belief-propagation.

$$q_\phi(Y, Z|M, X) = \prod_{t=1}^{T} q_\phi(y_t, z_t|M, x_t) = \prod_{t=1}^{T} q_\phi(z_t|x_t, y_t, M)q_\phi(y_t|x_t) \quad (2.39)$$

The writing inference model has the role of finding a trade-off that allows it to update the memory given some input X while maintaining old information. The writing mechanism is based on Bayes' rule where $p(M|X)$ is propor-

tional to the likelihood of the observed data $p(X|M)$ times the prior distribution of memory $p(M)$ (a Gaussian distribution). The likelihood of the observed data $p(X|M)$ is computed using the generative model, which is a hierarchical conditional generative model that generates samples from the memory vectors.

Differentiable Neural Computer

The DNC (Differentiable Neural Computer) [77]. The DNC is an extension of the NTM with memory attention mechanisms that control where the memory is stored, and temporal attention that records the order of events. This makes the DNC be more robust and abstract than an NTM. The DNC model is similar to the Von Neumann architecture and is Turing complete. An illustration if the DNC is provided in figure 2.31[20].

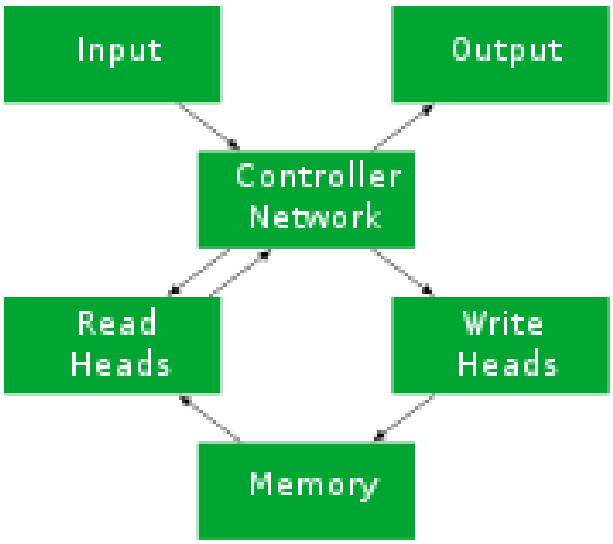


Figure 2.31 – Illustration of the DNC architecture.

The DNC differs from the NTM neural memory frameworks as the memory can be selectively written to and read from by an iterative modification of memory content thanks to an attention mechanism. The DNC uses a differentiable attention mechanism to define weightings over locations memory matrix M. The weightings represent the degree to which each location is involved in the read or write operation. The read vector r over M is a weighted sum of the matrix rows and the reading weights w_r. The write operation on the other hand uses a weighting w_w to erase with a vector e then add v to the matrix. Each of these operation are performed by a read and a write heads.

20. Credit to Kjerish, CC BY-SA 4.0 <https ://creativecommons.org/licenses/by-sa/4.0>, via Wikimedia Commons.

The heads use 3 forms of differentiable attention :

1. The first one performs content lookup where a key vector is created by the controller and is compared to the content of each location with a similarity measure allowing to perform associative recall with the read head or modify vectors in memory with the write head.
2. The second attention mechanism records transitions between consecutively written locations in a temporal link matrix L where each element $L_{i,j}$ of the matrix gets closer to one if i is the next location written after j, and 0 otherwise. For every write vector w, $L \times w$ smoothly shifts focus forward to the location written after those emphasized in w. $L^T w$ shifts the focus backward on the other hand thus giving the ability to recover sequences in the order in which they were written.
3. The third mechanism allocates memory for writing. The usage of a location is a number between 0 and 1 and the weighting picks out unused locations to deliver to the write head. The usage increases after a write operation and decreases after a read. The allocation mechanism is independent of the size of the memory, allowing to increase the size of the memory without needing to retrain the DNC.

In other words, the content lookup forms an associative memory data structure, the temporal links allows for sequential retrieval of inputs similarly to a human "free-recall", and the memory allocation provides unused locations to the writing head. An illustration of the whole model with each subcomponents and memory modules can be found in figure 2.32.

The DNC is capable of performing some reasoning tasks by combining supporting facts and make deductions from them in a text format. It The model also performs well on reasoning tasks on graphs, such as finding the shortest path in an underground network. It can also apply logical planning tasks on a block puzzle game where it was observed that the DNC was able to plan several steps ahead before acting upon it.

This section shows some possibilities of how memory can enhance the capacity of a model. The idea can be traced back to the origins of AI and can be found even in recent works [78]. In particular, recent models make use of external memory modules in order to increase the processing power of neural models. However, making a shift towards associative memory modules may help prevent a bottleneck in those neural models similar to a Von Neumann bottleneck both in algorithmic complexity (need to store a huge memory matrix) and in channel capacity (how much information can be transmitted from the memory back to the main neural controller).

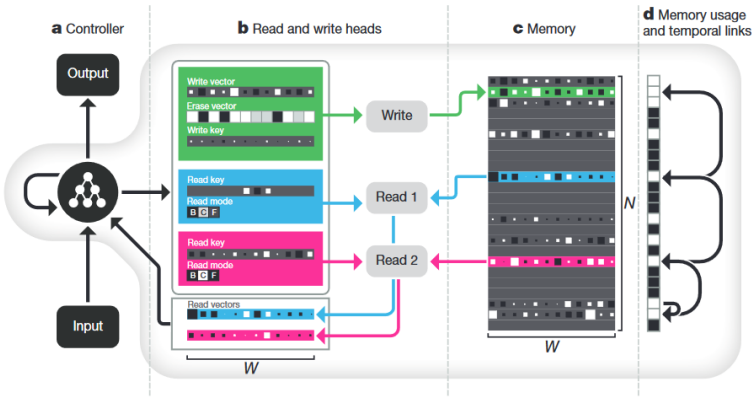


Figure 2.32 – Illustration of the DNC architecture with each memory module from the original paper [77]. In **a**, a recurrent neural network, the controller network, receives inputs to produce two outputs, ne of them parameterizing he the write head the read heads in **b, c**. The write head outputs a write (box area) and an erase vector (shading) that are used to edit the memory matrix. A write key is spots previously written locations to edit and can contribute to defining a weighting that selectively focuses the write operation over the rows, or locations, in the memory matrix. The read heads can use gates called read modes to switch between content lookup using a read key ('C') and reading out locations either forwards ('F') or backwards ('B') in the order they were written. In **d**, the locations which have been used so far are recorded with a usage vector, and a temporal link matrix records the order in which locations were written (the order is represented with the directed arrows.

3 - Out-of-distribution detection

3.1 . The general problem of out-of-distribution detection

When a model is deployed after being trained on curated training dataset and tested on the test set, problems may arise when the model is confronted to data that differ from the training distribution. A vision system should either be able to filter out or hand out this new piece of unusual data to a human to process it. We call "closed-world assumption" [79] the setting where a machine learning model is trained without awareness of data outside drawn from the same distribution as the training distribution. I note this distribution $p(x, y)$ where x are the features and y is the label of the data point. The aim is to make models more reliable and robust as well as to be able to interpret more easily failures in their detection when they are caused by an outlier. In a real open-world scenario where any data can arise, out-of-distribution (OOD) samples should however be handled with caution. Data points, when they are not drawn from $p(x, y)$ are called OOD data as opposed to in-distribution (ID) data drawn from $p(x, y)$. Indeed, models faced with data belonging to another class than those it was trained on cannot assign a label to it as it would not make sense. In this section, I will introduce the general problem of OOD detection as defined in [79]. OOD detection aims at detecting this data that does not belong to the underlying distribution the model is tuned for in order to filter it out before the true classification process.

We usually consider OOD data to be drawn from a distribution which is shifted from the in-distribution. This distribution shift can either be caused by a semantic shift or a covariate shift. A semantic shift or label shift is a shift in the label distribution $p(y)$ but $p(x|y)$ stays the same. This shift causes the nature of the problem to change. For example, a new class that was not seen in the training set appears may appear on inference (when the model is deployed). A covariate shift is a shift in the features of the input data $p(x)$ while $p(y|x)$ stays the same. This shift does not change the problem but rather the distribution of the features the model was trained on and therefore is mostly related to the generalization capabilities of the model (can it classify data that was taken in different conditions ?). These distribution shifts are illustrated in figure 3.1[1]. Other types of shifts exist, such as the concept shift where the definition of labels changes over time. For example, a definition may change in the fashion industry where the problem is to determine what garments pass as trendy, or in mental health screening where the definition of mental illness

1. Illustration of the original data and the covariate shift are taken from the book "Dive into Deep Learning" [26]. Illustration of the label shift is taken from the CIFAR-10 dataset [80].

may also evolve. The problem of OOD detection arises whenever one cannot guarantee that the data distribution is stationary or that input data may be taken from sources that are not related to the problem the model is trained on [26].

(a) Original data the model was trained on

(b) Covariate shift from the original data

(c) Label shift from the original data

Figure 3.1 – Illustration of semantic and covariate distribution shifts.

Depending on the type of distribution shift considered in the underlying problem, the task may be called differently. The paper [79] created a framework called "generalized out-of-distribution detection" in order to classify problems that relied on the open-world assumption. The problems covered in the survey in [79] are mainly focused on semantic shifts. The lack of understanding in the relations between those problems (open-set recognition, outlier detection, anomaly detection, novelty detection) often brings confusion by mixing datasets and terms.

In the next sections of this chapter, the reader can refer to [79] when studying a specific problem in the generalized OOD detection framework. Specifically, "OOD detection" refers to a problem where the ID and OOD dataset differ both from covariate and semantic shift. "Class anomaly detection" or "anomaly detection" is when both datasets differ only semantically.

3.2 . Problems of naive approaches in out-of-distribution detection

In deep learning, in a classification task, a model outputs a probability distribution for any given input thanks to the softmax function described in chapter 2. This softmax distribution $\{\sigma_k\}_{k\in\{1...K\}}$ corresponds to the probability of the input being classified as class $k \in \{1...K\}$. The input is labeled with the class of the highest softmax probability.

Intuitively, one should be able to think of this probability output as the confidence of the model in its classification. For example, if the assigned label to an input x by a model is label k with probability $\sigma_k = 0.6$, then intuitively one would think the model is confident at 60% that the label of x is k. Therefore, using the softmax distribution seems like an obvious solution to problems in the generalized OOD detection framework, as an OOD output should have a softmax distribution "close" to a uniform distribution $\mathcal{U}(\llbracket 1, K \rrbracket)$.

However, several problems arise when using the softmax distribution, which have been reported in the literature on uncertainty in deep learning [81]. It has been noted for example that classifiers can be overconfident in their classification, even when they are wrong (which is often observed especially in adversarial machine learning) [81] or when the input does not correspond at all to the distribution they were trained on or even unrecognizable [82] [83] [84]. This phenomenon of bad softmax predictions may be explained by the softmax function being a smooth approximation of the indicator function which tends to give a spiky distribution instead of a uniform one over the class set. This problem of misalignment between the actual uncertainty of the neural network and its predicted softmax probability distribution is called a "calibration" problem [81], i.e. : ensuring the model's predicted uncertainty (its output softmax distribution) is aligned with its measured accuracy. A well-calibrated model has a softmax distribution similar to what is represented in figure 3.2 [2]. We can see that, a JEM model (Joint Energy-based Model) [85] (right histogram) is well calibrated as its level of accuracy is close to its measured accuracy, as opposed to the baseline model on the left (the same architecture as the JEM model without an energy-based model training). The ECE (Expected Calibration Error) [86] of the well-calibrated model is lower than the ECE of the naive model.

Therefore, the softmax output of a classifier for OOD detection should be used with caution in order to avoid side effects from the overconfidence of the model. Some techniques though do utilize the softmax output to detect OOD inputs. For example, a common baseline is to use the maximum softmax output of the model ($max_k\{\sigma_k\}_{k\in\{1...K\}}$ and classify as OOD any input data whose maximum softmax is lower than a given threshold. More sophis-

2. Taken from [85].

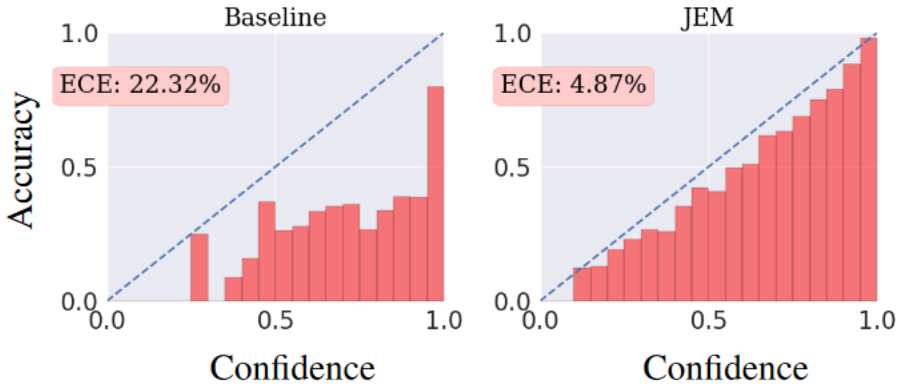


Figure 3.2 – Histogram of the accuracy level of the model with respect to its probability of classification. The x-axis represents the value of the log-likelihood, the y-axis the number of occurrence in the dataset for each bin of value (group of values). A well-calibrated model is expected to have a confidence close to its accuracy level, therefore an ideal model has a histogram such that the accuracy is equal to the confidence.

ticated techniques using the divergence (Kullback-Leibler divergence [87] or Bregman divergence [88]) between the model's softmax output and a uniform distribution exist as well. The rationale behind these divergence-based approaches is that the more uncertain a well-calibrated model is, the closer its softmax distribution should be to a uniform distribution (no single class should be more likely than another). These techniques however would only work correctly if the model is already well-calibrated.

On the other hand, some other methods relying on unsupervised learning can be thought about, such as techniques relying on the likelihood of the input computed by a generative model. Indeed, the likelihood measure $p(x)$ seems like a good candidate for OOD detection as it directly translates how likely, or how "in the distribution", a piece of data is compared to what was seen during training (assumed to be in the in-distribution). However, problems also arise when using the likelihood on OOD detection. Particularly, it was previously observed that training a likelihood model on some in-distribution and testing it on an out-distribution that is less complex, the assigned likelihood (or log-likelihood) is smaller for the in-distribution than for the out-distribution [89]. For example, if the likelihood model is a normalizing flow, the in-distribution the CIFAR-10 dataset [80] and the out-distribution the SVHN dataset [90], we observe a consistently lower log-likelihood on the in-distribution than on the out-distribution. This phenomenon is illustrated in figure 3.3.

We see that using a likelihood measure to filter out OOD data is not a reliable method. Directly using the probability a model yields as an output seems not to work as efficiently as expected. Other methods and metrics exist

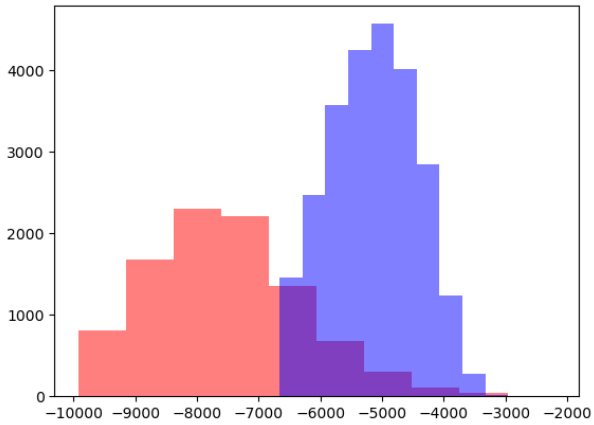


Figure 3.3 – Log-likelihood distribution of a RealNVP model [47] trained on CIFAR-10 (red) and tested on SVHN (blue). On the x-axis, the values of the log-likelihood, on the y-axis the number of occurrences of each bins (group of values) of log-likelihood. The log-likelihood is on average smaller for the in-distribution than for the out-distribution.

to circumvent the problems linked to using "naive" metrics, which will be covered in this chapter.

3.3 . State-of-the-art

As explained in the previous section, the problem of OOD detection cannot be covered by naïve methods relying on the probability distribution given by a model. In this section, I will introduce several methods of the state-of-the-art in OOD detection. The methods introduced here will be used later in this chapter to evaluate my method. During the bibliography phase, several methods recurrently appeared in the literature in OOD detection. The methods described in this section, which are also the ones used in the evaluations, are state-of-the-art methods that were that appeared in most OOD detection works as a comparison baseline.

Two big families of models were noticeable at the time of the bibliography, mostly output-based methods and gradient methods. The output-based methods base their approach on the output of the model to detect deviation in the input from the learned in-distribution. Gradient methods use information provided by the gradient of some output of the network (for example the output of the penultimate layer of the neural network) with respect to the network parameters to assess the deviation of the input from the in-distribution.

3.3.1 . ODIN

The ODIN method [91] (**O**ut-of-**DI**stribution detector for **N**eural networks) relies on a model trained on the in-distribution (for a classification task), without adding any modification to it. The method first adds noise to the input x, thus yielding a preprocessed data point $\tilde{x}$, according to equation 3.7. This preprocessing is inspired by adversarial attacks, more specifically the FGSM attack [92]. The softmax score used here is defined in equation 3.2 and corresponds to the temperature scaled softmax distribution with temperature parameter T.

$$\tilde{x} = x - \epsilon sign(-\nabla_x \log(S_{\tilde{y}}(x, T))) \text{ where } S_{\tilde{y}}(x, T) = \max_i S_i(x, T) \tag{3.1}$$

Then the preprocessed input is fed into the network to calculate the calibrated softmax score $S(\tilde{x}, T)$ which corresponds to the softmax distribution returned by the model scaled by a temperature parameter T as specified in equation 3.2.

$$S_i(x, T) = \frac{exp(f_i(x)/T)}{\sum_{j=1}^{N} \exp(f_j(x)/T)} \tag{3.2}$$

Finally, this calibrated softmax is compared to a threshold $delta$ in order to classify as ID or OOD the input according to the following decision rule based on the function g defined in 3.3 : if $g(x, T, \delta, \epsilon)$ is equal to 1 the input is ID, if it is equal to 0 it is OOD.

$$g(x, T, \delta, \epsilon) = \begin{cases} 0 & \text{if } S_{\tilde{y}}(\tilde{x}, T) \leq \delta \\ 1 & \text{else} \end{cases} \tag{3.3}$$

In this model, the parameters T, δ, ϵ are chosen such that the true positive rate is at 95% on a validation set composed in part of OOD and ID data.

3.3.2 . Generalized ODIN

The ODIN method was further developed to lead to Generalized ODIN [93] (abbreviated as G-ODIN) which does not rely on OOD data to search for hyperparameters. Although the temperature parameter T in the softmax equation 3.2 is originally tuned with OOD data, a later work [94] suggested that setting it to a high value was enough, the gain being saturated after $T = 1000$, which the authors of [93] follow. Furthermore, in order to tackle the limitations of the uncertainty on the softmax distribution exposed in section 3.2, the authors consider the output probability distribution of the network differently, this time by including in the variable conditioned on the domain d of data. Implicitly, other methods condition the distribution $p(y|x)$ on $d = d_{in}$ where d_{in} is the in-distribution domain, which is based on the closed-world assumption

as defined in section 3.1. The new output distribution in G-ODIN is changed for the conditional probability distribution defined in equation 3.4.

$$p(y|d_{in}, x) = \frac{p(y, d_{in}|x)}{p(d_{in}|x)} \tag{3.4}$$

For an input x p_{out}, both probabilities (called "decomposed confidence scores") $p(y, d_{in}|x)$ and $p(d_{in}|x)$ are expected to be low. Depending on their respective values however, the ratio in equation 3.4 may be arbitrarily high, which explains how softmax classifiers can be overconfident.

In order to learn $p(y|x, d_{in})$ without any OOD data, the authors use the prior knowledge on the distribution given by equation 3.4 by writing the logits $f_i(x)$ for class i as a ratio of two functions given by equation 3.5. This ratio is then normalized by the softmax function to get a class probability. Training $p(y|x, d_{in})$ with a cross-entropy loss means it can be minimized either by minimizing $h_i(x)$ or maximizing $g(x)$. This prior encourages the function $h_i(x)$ to behave closer to $p(y, d_{in}|x)$ and $g(x)$ to behave like $p(d_{in}|x)$.

$$f_i(x) = \frac{h_i(x)}{g(x)} \tag{3.5}$$

The function g uses the features $f^p(x)$ from the penultimate layer of neural networks as in equation 3.6. In this equation, BN is the batch normalization operator. This function can be seen as a learned temperature scaling function while providing a probabilistic view for its effect. The function h_i may be chosen among several similarity measures (the authors tested the inner-product, the negative euclidean distance and the cosine similarity), but the cosine similarity yields the best results.

$$g(x) = \sigma(BN(w_g f^p(x) + b_g)) \tag{3.6}$$

The preprocessing step provided in equation 3.7 is also slightly modified in order to remove the dependence on the tuning of the ϵ parameter on OOD data and remove the log function, as shown in equation 3.7. In G-ODIN, the ϵ value in equation 3.7 is computed by maximizing the score $S(x)$ on the preprocessed in-distribution validation set D_{in}^{val}, solving the optimization problem 3.8 with a grid search on 6 values. It is argued that an ϵ parameter that makes the score $S(x)$ sufficiently large is enough to discriminate OOD and ID data.

$$\tilde{x} = x - \epsilon sign(-\nabla_x(S(x)) \tag{3.7}$$

$$\epsilon^* = arg\max_{\epsilon} \sum_{x \in D_{in}^{val}} S(\hat{x}) \tag{3.8}$$

An illustration of the Generalized ODIN method is provided in figure 3.4 [3].

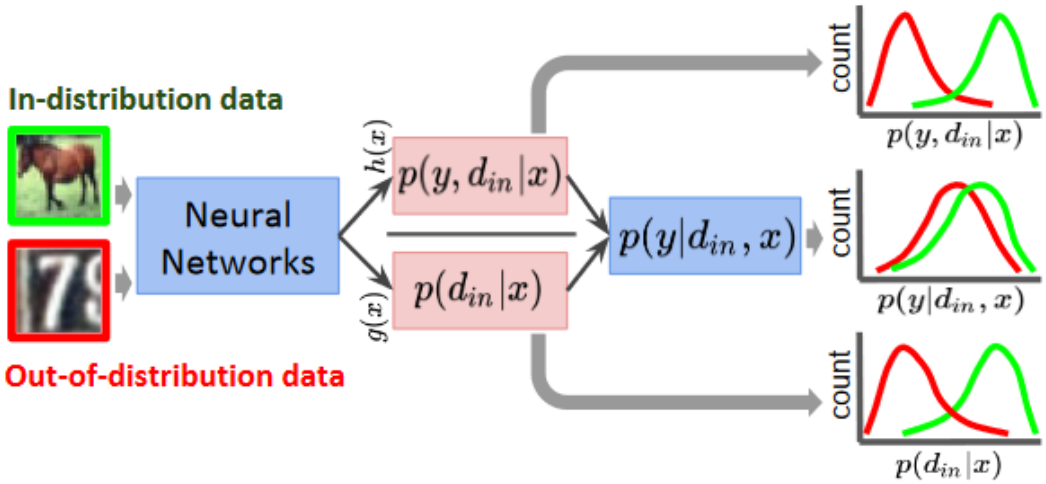


Figure 3.4 – Illustration of the Generalized ODIN method.

3.3.3 . Mahalanobis-distance

For a softmax classifier expressed in equation 3.9 where $f(x)$ is the output of the penultimate layer of the neural network, the authors in [95] show how to turn it into a generative model by assuming a Gaussian class-conditional distribution with a mutual covariance for every class with equation 3.10.

$$p(y = c|x) = \frac{\exp(w_c f(x) + b_c)}{\sum_{c'=1} \exp(w_{c'} f(x) + b_{c'})} \tag{3.9}$$

$$p(f(x)|y = c) = \mathcal{N}(f(x)|\mu_c, \Sigma) \tag{3.10}$$

The parameters of the Gaussian in definition 3.10 are estimated with the estimators 3.11 where N_c is the number of training samples with in class c and N the total number of samples.

$$\begin{aligned} \hat{\mu_c} &= \frac{1}{N_c} \sum_{i, y_i = c} f(x_i)\,, \\ \Sigma &= \frac{1}{N} \sum_{c} \sum_{i, y_i = c} (f(x_i) - \hat{\mu_c})(f(x_i) - \hat{\mu_c})^T \end{aligned} \tag{3.11}$$

In [95], the Mahalanobis distance-based confidence score, defined in 3.12, uses the class-conditional Gaussian distribution. The metric in 3.12 corresponds to measuring the log probability density of the tested sample.

3. From the original paper [93]

$$M(x) = \max_c -(f(x) - \hat{\mu_c})^T \hat{\Sigma}^{-1} (f(x) - \hat{\mu_c}) \tag{3.12}$$

The Mahalanobis distance is argued to be more effective than the Euclidean distance in various tasks, the latter being used in previous works by other authors [96] [97]. The Mahalanobis distance takes into account the covariance structure of the data, which is important for detecting out-of-distribution samples. The Mahalanobis distance is a measure of the distance between a point and a distribution, making this method able to capture the correlations between features and adjust the distance metric accordingly, which leads to better performance in detecting out-of-distribution samples compared to other distance metrics such as Euclidean distance. An illustration of the difference between the Euclidean and the Mahalanobis distance can be found in figure 3.5 [4].

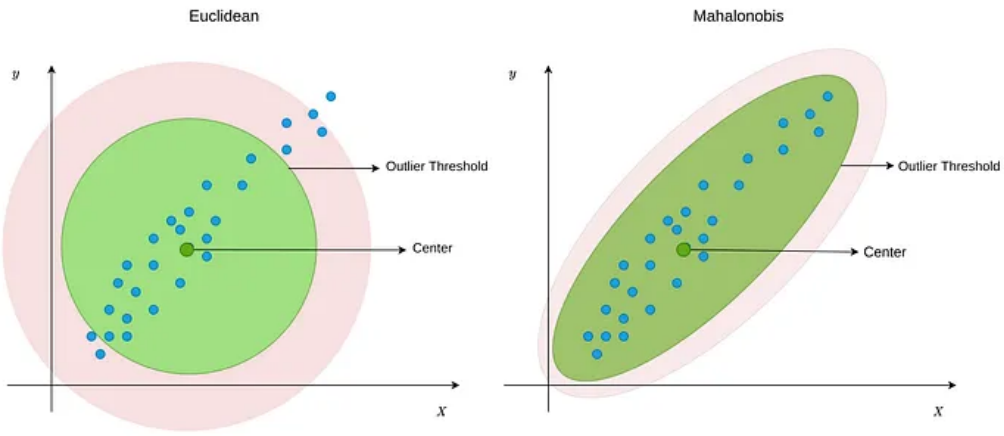


Figure 3.5 – Illustration of the Euclidean and the Mahalanobis distance. We can see that the Mahalanobis distance takes into account the covariance structure of the data by considering the scattering direction of the points.

In order to make ID and OOD data more separable, the model also adds noise to the input as part of a preprocessing step following equation 3.13 where $\hat{c}$ is the closest class. Here the noise aims at increasing the confidence score, as opposed to ODIN [94] as exposed in paragraph 3.3.1.

4. From https ://towardsdatascience.com/multivariate-outlier-detection-in-python-e946cfc843b3.

$$\begin{aligned}\hat{x} &= x + \epsilon sign(\nabla_x(M(x)) \\ &= x - \epsilon sign(\nabla_x(f(x) - \hat{\mu_c})^T \hat{\Sigma}^{-1}(f(x) - \hat{\mu_c}))\end{aligned} \tag{3.13}$$

Finally, the performance of in OOD detection are further improved by combining the confidence scores from the final features as well as other lower level features in the DNN. Each layer has its own covariance matrix $\hat{\Sigma}_l$ and class means $\hat{\mu}_{l,c}$. The scores are then aggregated by the weighted average formula in equation 3.14 where α_l parameter is the weight of the l^{th} layer. The parameter α_l is found by logistic regression detector using validation samples.

$$M_{aggregated}(x) = \sum_l \alpha_l M_l(x) \tag{3.14}$$

3.3.4 . Energy-based

In [98], the authors show it is possible to design an energy function from a softmax classifier that has the same shape as in equation 3.2 with equation 3.15.

$$E(x, f) = -T \log \sum_i^K exp(\frac{f_i(x)}{T}) \tag{3.15}$$

The authors of this paper propose to detect OOD samples with this energy function in order to mitigate the problems in the softmax confidence score exposed in section 3.2. Naturally, from this energy score, one can derive a density $p(x)$ through equation 3.16. The denominator Z is intractable, however its absence does not influence the performance in OOD detection as higher probability of occurrence is interpreted as a lower energy level in equation 3.16. Indeed, the rationale behind the use of an energy score for OOD detection is that it is theoretically aligned with the probability density of the inputs and is less susceptible to the overconfidence issue that can occur with softmax confidence scores. Additionally, the energy score can be flexibly used as a scoring function for any pre-trained neural classifier.

$$p(x) = \frac{exp(\frac{f(x)}{T})}{\int_x exp(\frac{f(x)}{T})} = \frac{exp(\frac{f(x)}{T})}{Z} \tag{3.16}$$

Therefore, the authors of this paper propose a decision rule directly based on the energy function, which is presented in equation 3.17. If $G(x, \tau, f) = 0$ then the sample is OOD, if $G(x, \tau, f) = 1$ the sample is ID. In 3.17, τ is defined as the energy threshold which is the threshold that correctly classifies the highest fraction of in-distribution inputs with detector G.

$$G(x, \tau, f) = \begin{cases} 0 & \text{if } -E(x, f) \leq \tau \\ 1 & \text{else} \end{cases} \tag{3.17}$$

The energy metric allows to perform OOD detection without having to compute the normalization constant Z in equation 3.16. It acts as a replacement of the softmax confidence score for a trained neural network as seen in the final decomposition in 3.19 (when temperature $T = 1$).

$$\max_y p(y|x) = \max_y \frac{exp(f_y(x))}{\sum_i exp(f_i(x))} = \frac{1}{\sum_i exp(f_i(x) - f_{y_{max}}(x))} \tag{3.18}$$

$$\log \max_y p(y|x) = E(x, f(x) - f_{y_{max}}(x)) = E(x, f(x)) - f_{y_{max}}(x) \tag{3.19}$$

Equation 3.19 shows that the softmax confidence score is a version of the energy score biased by the $f_{y_{max}}(x)$ term which is the highest softmax output value. The authors argue that working in the original logit space (the energy score) rather than the shifted logit space (the softmax score) gives more useful information for each sample.

However, despite the energy score being already showing good results compared to the softmax confidence score in OOD detection, the authors propose to enlarge the gap between the in-distribution and the out-distribution. To this end, the authors propose an energy bounded learning objective 3.20 involving a regularization loss L_{energy} defined in 3.21 to fine-tune the energy function to explicitly expand the energy gap by assigning lower energy to ID data and higher energy to OOD data. In equation 3.21, the terms m_{in} and m_{out} are margin hyperparameters that are used to penalized in two separate hinge square loss terms in-distribution samples that produce higher energy levels than m_{in} and out-of-distribution samples that produce energy values lower than m_{out}. The model is thus constrained to keep its energy function within the segment $[m_{in}, m_{out}]$. These hyperparameters are free to choose and not found with a grid search on a validation set for example.

$$\min_\theta \mathbb{E}_{(x,y)\sim\mathcal{D}_{in}^{val}}[-\log F_y(x)] + \lambda L_{energy} \tag{3.20}$$

$$\begin{aligned} L_{energy} =& \mathbb{E}_{(x,y)\sim\mathcal{D}_{in}^{train}}[\max(0, E(x_{in}) - m_{in}]^2 + \\ & \mathbb{E}_{x_{out}\sim\mathcal{D}_{out}^{train}}[\max(0, m_{out} - E(x_{out})]^2 \end{aligned} \tag{3.21}$$

One important thing to highlight from this OOD detection method is that the training of the model relies on OOD detection like in paragraph 3.3.1 although in this case the ODIN model relied on a validation out-distribution dataset in order to find optimal hyperparameters while here the OOD dataset directly intervenes in finding the model's parameters in equation 3.21.

An illustration of the principle of the energy-based OOD detection method is given in figure 3.6 [5].

5. Taken from the original paper [98]

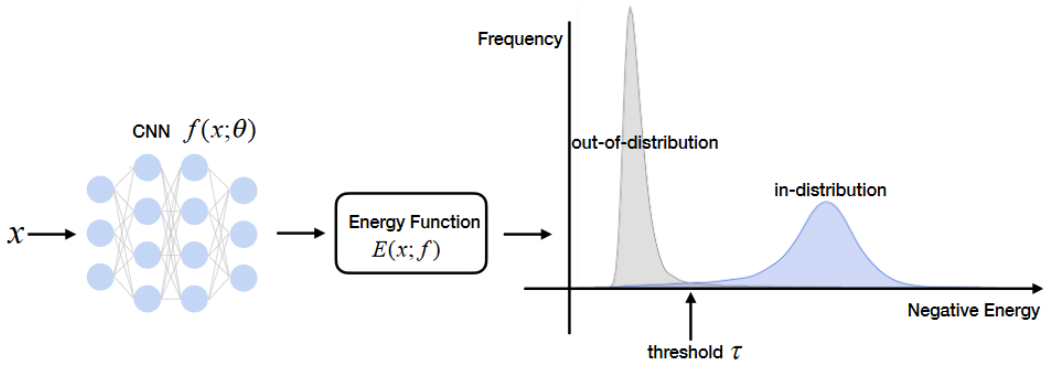


Figure 3.6 – Illustration of the energy-based OOD detection : an energy function is derived from the output of a CNN and yields good OOD detection results.

3.3.5 . GradNorm

The GradNorm method [99] is a gradient-based OOD detection technique that can be directly used on a pre-trained classifier. From the classifier, one can compute the KL divergence between the softmax distribution and a reference uniform distribution over the labels $u = [\frac{1}{K}, \frac{1}{K}, ..., \frac{1}{K}]$, as described in section 3.2. For a softmax distribution parameterized by a temperature parameter T (similarly to paragraph 3.3.1), this KL divergence term is written in equation 3.22 where $H(u)$ is the entropy of the uniform random variable u (constant).

$$D_{KL}(u||softmax(f(x))) = -\frac{1}{K}\sum_{i=1}^{K}\log\frac{exp(f_i(x)/T)}{\sum_{j=1}^{K}exp(f_j(x)/T)} - H(u) \quad (3.22)$$

The idea behind GradNorm is to detect OOD samples from the gradient with respect to the model's parameters of the KL divergence in equation 3.22, as written in equation 3.23. In equation 3.23, w is a component of the model's parameters θ and $\mathcal{L}_{CE}(f(x), i)$ the cross-entropy loss of the model f from label i. Naturally , the entropy term $H(u)$ has a gradient of 0.

$$\frac{\partial D_{KL}(u||softmax(f(x)))}{\partial w} = -\frac{1}{K}\sum_{i=1}^{K}\frac{\partial \mathcal{L}_{CE}(f(x), i)}{\partial w} \quad (3.23)$$

Equation 3.23 shows that taking the gradient of the KL divergence is equivalent to averaging the derivative of the categorical cross-entropy loss across all labels, therefore simplifying the computation of the GradNorm score $S(x)$. The score used to detect OOD samples is derived from this gradient term by taking its p-norm, as exposed in equation 3.24 and by taking the gradient with

respect to all the parameters of the model θ, which correspond to the parameters w concatenated into a single vector. However, it is also shown that using only the last layer for taking the gradient is sufficiently informative. Moreover, an L_1 norm is shown to be the most effective.

$$S(x) = \| \frac{\partial D_{KL}(u||softmax(f(x)))}{\partial \theta} \|_p \tag{3.24}$$

Similarly to previous paragraphs, a decision rule is derived from this term in 3.25 where γ is a threshold parameter.

$$g(x) = \begin{cases} out & \text{if } S(x) \leq \gamma \\ in & \text{else} \end{cases} \tag{3.25}$$

The gradient of the KL divergence is argued to be more powerful for OOD detection than the direct use of the KL divergence. Furthermore, GradNorm is label-agnostic (does not require OOD data) and it captures uncertainty across all labels which empirically shows better results than using only the dominant label (maximum softmax value in the output).

Finally, a mathematical study of this method shows that the GradNorm metric captures joint information between the feature and the output. Indeed, it can be shown that the score $S(x)$ is proportional to the product of a variable U, which is solely dependent on the input features, and a variable V which depends on the output of the model. Multiplying those two terms results in a supposedly stronger separability between ID and OOD data.

An illustration of the difference between the input space and the gradient space can be observed in figure 3.7 [6].

3.3.6 . GradCon

Another model based on a gradient metric is GradCon (for **Grad**ient **Con**straint) [100]. As opposed to previous methods introduced in this part, the GradCon model was not introduced nor tested in the context of OOD detection. Rather, this model was introduced in the context of class anomaly detection, where one class in a dataset is the ID class while the remaining classes are OOD (further details in section 3.6.7). Furthermore, it relies on an unsupervised generative model instead of a classifier trained in supervised learning context.

Autoencoders such as VAEs (Variational Autoencoders) or AAEs (Adversarial Autoencoders) encode inputs to their latent space which is constrained to follow a normal distribution generally. Deviations of the input's representation from the normal distribution characterizes abnormalities. The GradCon model adds a directional gradient constraint during the training of an autoencoder to further discriminate ID and OOD data. This constraint forces ID data

6. From the original paper [99].

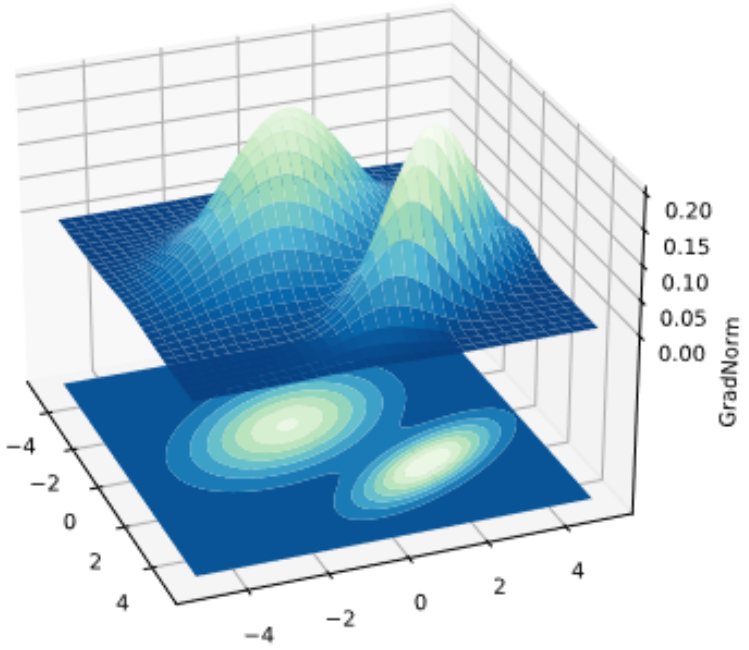


Figure 3.7 – Illustration of a two-dimensional input space. Input data is depicted in the xy-plane. Gradient norm for each input is depicted in the z dimension. The magnitude of gradients is higher for ID data (light green) than for OOD data (deep blue).

to have gradients aligned with each other. The gradient of OOD data on the other hand will not be aligned with the other gradients.

The gradient constraint is added with a regularization term, $\mathcal{L}_{grad}$ to the loss of the autoencoder, J. The cosine similarity between the gradient of each layer i at iteration k and the average of the training gradients of the same layer i until iteration $k-1$ is computed. The regularization term takes the form in equation 3.26. The average term is explicitly written in equation 3.27.

$$\mathcal{L}_{grad} = -\mathbb{E}_i[cosSIM(\frac{\partial \mathcal{J}^{k-1}}{\partial \phi_{i_{avg}}}, \frac{\partial \mathcal{L}^k}{\partial \phi_i})] \tag{3.26}$$

$$\frac{\partial \mathcal{J}^{k-1}}{\partial \phi_{i_{avg}}} = \frac{1}{k-1}\sum_{t=1}^{k-1}\frac{\partial \mathcal{J}^t}{\partial \phi_i} \tag{3.27}$$

The loss J of the autoencoder is defined as in equation 3.28 where $\mathcal{L}_{grad}$ is introduced in equation 3.26, $\mathcal{L}$ is the reconstruction error ensuring the sampled output of the autoencoder is similar to its input, and Ω is the latent loss ensuring the latent space of the autoencoder follows a Gaussian distribution. Their respective definition depends on the autoencoder architecture. The hyperparameter α is a weight defining the importance of the gradient loss 3.26.

This hyperparameter is set sufficiently small to ensure gradients actively explore the optimal weights until the reconstruction error and the latent loss become small enough. Only the decoder layers are constrained while leaving the encoder layers unconstrained.

$$J = \mathcal{L} + \Omega + \alpha \mathcal{L}_{grad} \tag{3.28}$$

An anomaly score is defined through the combination of the loss terms by the term $\mathcal{L}_{grad} + \beta\mathcal{L}$ with a β parameter different than the α weight used during training in equation 3.28. The gradient loss is found to be often more effective than the reconstruction error in anomaly detection. The two loss terms in the anomaly detection metric are balanced by setting $\beta = 4\alpha$.

This anomaly detection method is mostly used in the context of class anomaly detection (further explanations in section 3.6.7) and anomalous condition detection. This last task mostly detects covariate shifts due to conditions on the acquisition of data differing from the original dataset (blurry lens, rain...). As a personal note, this task may be unnatural in the context of OOD detection as it raises the question of what an OOD detection model should filter out as these out-distributions are technically OOD but should not be filtered out and instead should be classified by a model. This mostly enters in the context of generalization.

3.3.7 . OCGAN

OCGAN (One-Class GAN) [101] is a model made specifically for the task of one-class novelty detection where the goal is to determine if a query example is from the same class the model is trained on (equivalent to class anomaly detection). The idea behind OCGAN is to make the model learn a latent representation that would only make it possible for the model to reconstruct images that are similar to the ID class. Thus, the latent representation of an OOD sample (from a novel class) would get a worse reconstruction loss than inputs from the ID class. This intuition is illustrated in figure 3.8.

The OCGAN method uses four components : a denoising autoencoder, a latent discriminator, a visual discriminator and a classifier.

The denoising autoencoder learns a representation for the given class by minimizing the distance between the input and the output of the network with the loss denoted in equation 3.29, where n is a Gaussian noise $\mathcal{N}(0, 0.2)$. The latent space has a lower dimension than the input space in order to retain only essential information for reconstruction. The support of the latent space is bounded by introducing a $tanh$ activation in the output layer of the encoder to ensure the output is in $(-1, 1)^d$, where d is the dimension of the latent space.

$$l_{MSE} = \| \ x - De(En(x + n)) \ \|_2^2 \tag{3.29}$$

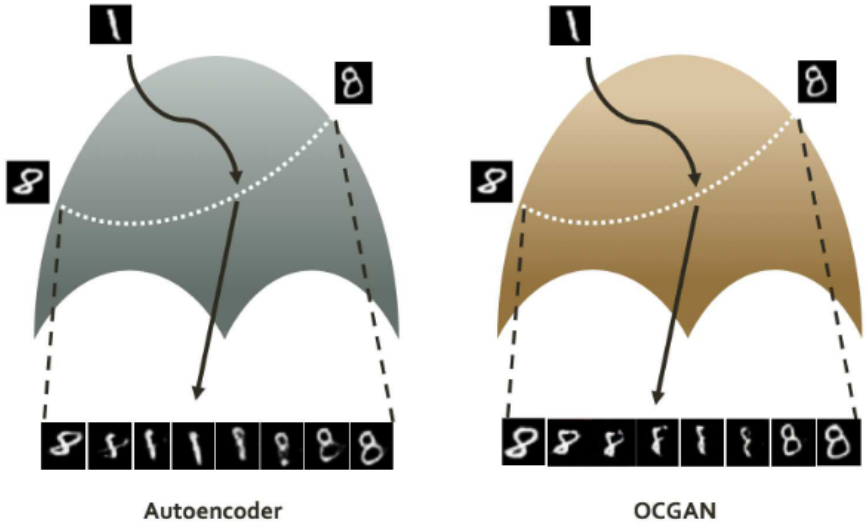


Figure 3.8 – Illustration of the idea behind OCGAN : on the left the latent space of a vanilla autoencoder and on the right the latent space of the OCGAN model. Both models have ID class "8" in the MNIST dataset. However, given a certain path in the latent space, the vanilla model may be able to give a representation that yields an image close to a "1". On the other hand, the OCGAN model still outputs an "8" from the latent representation of an input "1". Credit to the original paper [101].

The latent discriminator D_l forces the latent representation of ID class examples to be evenly distributed across the latent space. It is trained to differentiate between latent representations of real images of the given class and samples drawn from a uniform distribution $\mathbb{U}[(-1,1)^d]$ with the loss defined in equation 3.29. In equation 3.30, p_x is the distribution of the ID class. The loss l_{latent} is similar to the training objective of a GAN [61], hence the term OCGAN. Indeed, a GAN is trained by making a generator network fool a discriminator network to classify generated images as natural. Here, the rationale is similar in that the encoder is trained to generated latent code that is similar to a uniform distribution to fool the latent discriminator. The encoder and the latent discriminator are jointly trained with a min-max objective $\max_E n \min_{D_l} l_{latent}$.

$$l_{latent} = -(\mathbb{E}_{s\sim\mathbb{U}[(-1,1)^d]}[\log D_l(s)] + \mathbb{E}_{x\sim p_x}[\log 1 - D_l(En(x+n))]) \quad (3.30)$$

The visual discriminator D_v enforces the constraint that all images generated from the latent space are similar to the ID class without any negative class available during training. The visual discriminator is trained to differentiate between images of the given class and images generated from random latent samples using the decoder. The training procedure here is also similar to that of a GAN [61] as the goal is also for the generator (the decoder and the

latent space structure) to fool the visual discriminator. The adversarial loss is defined in equation 3.31. The visual discriminator is learned jointly with the autoencoder with the min-max objective $\max_{De} \min_{D_v} l_{visual}$.

$$l_{visual} = -(\mathbb{E}_{s \sim \mathbb{U}[(-1,1)^d]}[\log D_v(De(s))] + \mathbb{E}_{x \sim p_l}[\log 1 - D_v(x)]) \tag{3.31}$$

However, this training procedure still outputs samples that do not look like the ID class because sampling from the whole space is impossible. To tackle this issue, the authors decide to actively seek out regions in the latent space that produce negative samples. These samples are called informative-negative samples. They are used to train the decoder so that it learns to produce good quality samples even in these regions. These samples are found by using a classifier to assess the quality of image generated from the samples. The loss of the classifier, the cross-entropy loss, is backpropagated to the latent space. The backpropagated gradient is then used to take a small step in the direction of the gradient to move to a new point in the latent space where the classifier is confident about the generated image. The classifier does not take part to the min-max games in the GAN training, therefore it is not required to balance it.

The discriminators are trained by minimizing the loss $l_{visual} + l_{latent}$, the autoencoder is trained with informative negative examples and latent projections (real samples injected with noise) of the given class using the loss $10l_{MSE} + l_{visual} + l_{latent}$, with a larger empirical term given to l_{MSE} to get good reconstruction results.

It is assumed the novelty detection strategy uses the reconstruction loss l_{MSE} defined in equation 3.29 to discriminate novel/OOD samples from ID samples, although this was not explicitly written by the authors in their paper.

3.4 . Experimental methodology

Some efforts have been made in the past few years by the OOD detection community to standardize and make testing and comparison easier for new methods of OOD detection. In the experiments in section 3.6.6 and 3.6.7, I encountered several issues when trying to reuse results from previous state-of-the-art models to compare my approach against them in the benchmarks. In this section, I will cover some new points of methodology I introduced in order to make the comparisons more fair between models. These points include the choice of the datasets, the choice of the models and their number of parameters, the balance between classes and the choice of metrics.

3.4.1 . Definition of the problem and the choice of dataset

The first problem I noticed was the lack of standardization in the definition of the problem of OOD detection. This was covered in section 3.1 by a previous study [79] which defines in clear terms the differences between each sub-problem in OOD detection. In my experiments, I followed this standardization by choosing datasets accordingly. In OOD detection, I chose datasets that presented no label overlap (no common labels between ID and OOD datasets) while ensuring that the visual features of the images were not too close to each other from a qualitative point of view. As such, I did not use the CIFAR-100 dataset [80] in the experimentations when I trained a model on CIFAR-10 [80] for example, as opposed to other papers in the domain. In anomalous class detection, I used the standard approach which involved selecting a class in a dataset as ID and the other classes as OOD.

3.4.2 . Evaluation measures and experimental precautions

The second problem encountered was methodological. I studied several implementations in order to use them on new datasets (that would better fit the problem setting defined in [79] and in section 3.1) would misuse conventional metrics. Usually, when dealing with a binary classification problem, a model that outputs a raw score needs to turn this score into a binary label (0 or 1) by comparing the score to a threshold : if the score is above the threshold, the label is 1, else it is 0. The most common metrics used in OOD detection are generally non-thresholded, that is to say metrics they directly use the raw score given by the model as an output instead of a binary value (0 or 1). The most common non-thresholded metric is the AUROC [102] (Area Under the Receiver-Operating Characterstic curve). An illustration of the ROC curve is given in figure 3.9 [7]. It represents the probability that the model assigns a higher score to the positive class A than to the negative class B. The higher the value, the better the results (the highest value is 1 and the lowest value is 0). However, this metric is sensitive to class imbalance : in a binary classification problem, if a class A is more represented than class B and the model has better classification accuracy on A than B, then the AUROC will be higher. Therefore, it is important to balance appropriately the ID and OOD classes in the test datasets used in experiments in order for the AUROC to stay interpretable. This was however not always respected in some experimentation implementation nor was it mentioned in their respective papers.

Along with the AUROC, other metrics can complement the classification results of the model. For instance, I decided to add the AUPR in the experiments, at first because it has the property of being robust to class imbalance, therefore complementing the AUROC [103]. It is also a standard metric in the

7. Credit to Wikimedia Commons (Roc curve.svg by cmglee and MartinThoma, licensed under CC BY-SA 4.0).

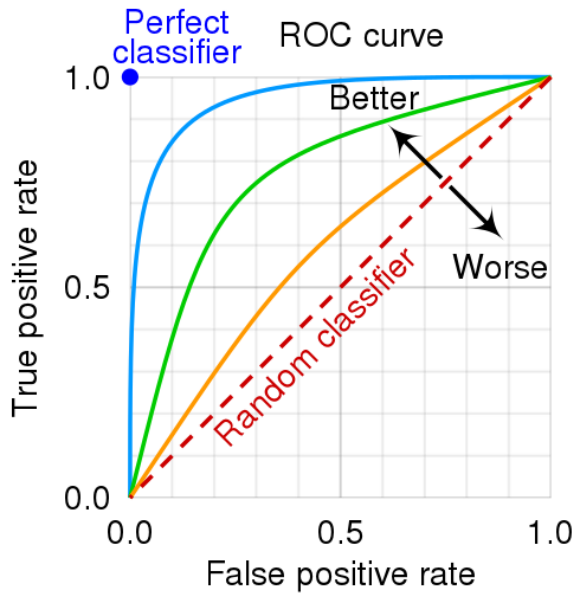


Figure 3.9 – Illustration of the ROC (Receiver-Operating Characterstic) curve. The AUROC is the area under this curve. The diagonal line represents a random classifier : in binary classification with balanced classes, this amounts to flipping a fair coin and classify data according to the coin flip. Any curve above the random line is a better classifier. Any curve below that line is not necessarily a bad classifier but may be due to label inversion (in binary classification). The ideal classifier shows a True Positive Rate of 1 for any level of False Positive Rate.

OOD detection literature. While the AUROC is mostly sensitive to classification the overall accuracy, the AUPR is mostly sensitive to the classification accuracy of the positive class. An illustration of the precision and the recall metrics is given in figure 3.10 [8]. The AUPR is the area under the curve of the precision-recall plot. The higher the value, the better the results (the highest value is 1 and the lowest value is 0).

Finally, the TNR@95%TPR (True Negative Rate at 95% True Positive Rate) which is simply a point on the ROC curve which gives a better understanding of the classification performance of the model, efficiently complementing the AUROC metric. As the AUROC is the area under the ROC curve, in the absence of an explicit ROC plot, one may need some extra information in the form of certain data points. Particularly, knowing the values of the TPR at a high FPR value can be very informative. It is to be noted though that this metric is also sensitive to potential class imbalance like the AUROC.

8. Credit to Wikimedia Commons (Precisionrecall.svg by Walber, licensed under CC BY-SA 4.0).

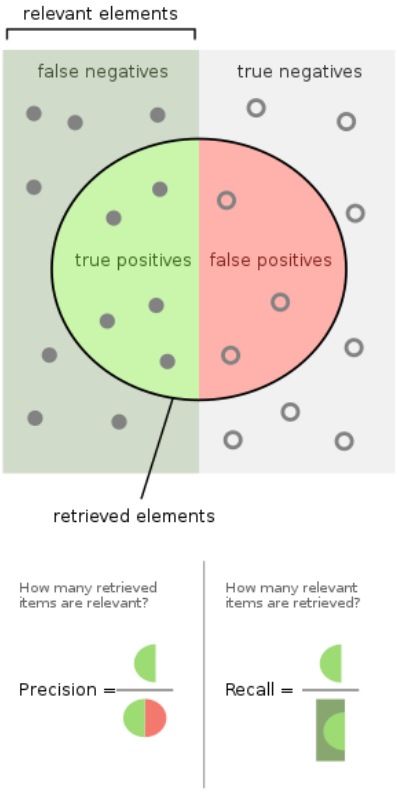


Figure 3.10 – Illustration of the precision and the recall metrics.

3.4.3 . Frameworks

Efforts to standardize evaluation in OOD detection also lead to the creation of frameworks to efficiently compare OOD detection approaches against standard or state-of-the-art methods. These frameworks centralize into one library the most used models in the literature in benchmarks and the most common datasets in this task with code to quickly compare implement an OOD detection experiment with a new model.

An example of such frameworks include openOOD [104] [9]. This framework reproduces methods within the Generalized Out-of-Distribution Detection methodology developed in section 3.1, aiming to make a fair comparison across methods that were initially developed for various tasks such as anomaly detection, novelty detection, open set recognition, and out-of-distribution detection.

The Pytorch-OOD [105] repository is also an interesting initiative with the intent to make the development and experimentation of OOD detection al-

9. With the associated Github repository at https ://github.com/Jingkang50/OpenOOD.

gorithms quicker. It is a PyTorch-based library that gives access to several modular, tested, and well-documented implementations of OOD detection methods. This library is designed to provide users with a unified interface, pre-trained models, utility functions, and benchmark datasets for OOD detection.

These frameworks however did not come to my knowledge during my experimentations and therefore I did not rely on them.

3.5 . The approximate mass

Alternatively to the metrics introduced in section 3.3, other metrics can be found in the literature on OOD detection. In [85], the approximate mass, defined in equation 3.32 is the L_2-norm of the gradient with respect to the input x of the log-likelihood given by a model parameterized by parameters θ, $\mathcal{L}(x, \theta)$.

$$AM = \| \frac{\partial \mathcal{L}(x;\theta)}{\partial x} \| \tag{3.32}$$

As exposed in section 3.2, directly using the log-likelihood $\mathcal{L}$ in OOD detection exposes the model to some risks as the measure might sometimes assign larger log-likelihood values to OOD data than to ID data. The approximate mass is another derived from the log-likelihood that seem to be well suited for OOD detection in the paper where it is introduced [85].

The approximate mass is expected to be higher for OOD data than for ID data for an energy model in [85] but seems to behave the other way around on tractable likelihood models such as normalizing flows. I reproduced this behavior observed on a normalizing flow (RealNVP model [47]) with the same training setting as in [48]. I trained the model on CIFAR-10 as the ID class and test it both on CIFAR-10 [80] (ID) and SVHN [90] (OOD).

Figure 3.11 shows the distribution of the approximate mass computed at the end of the training of the normalizing flow. We can observe that the model is indeed able to discriminate OOD and ID data but does so in a way that might not be robust. Indeed, the intuition behind the definition of the approximate mass given in equation 3.32 shows that the larger the approximate mass is for ID data, the more the log-likelihood changes, which indicates a fragile system. A slight variation in the input may lead to high variations around ID data while intuitively, for a well-trained model that generalizes well, the log-likelihood should not change much for close samples. This experimental confirmation raises question about whether this observed behavior on tractable likelihood model may not be caused by a lack of generalization capacity by the model, in other words overfitting.

The approximate mass still presents interesting properties as the metric should be exactly the same for energy-based models an exact likelihood mo-

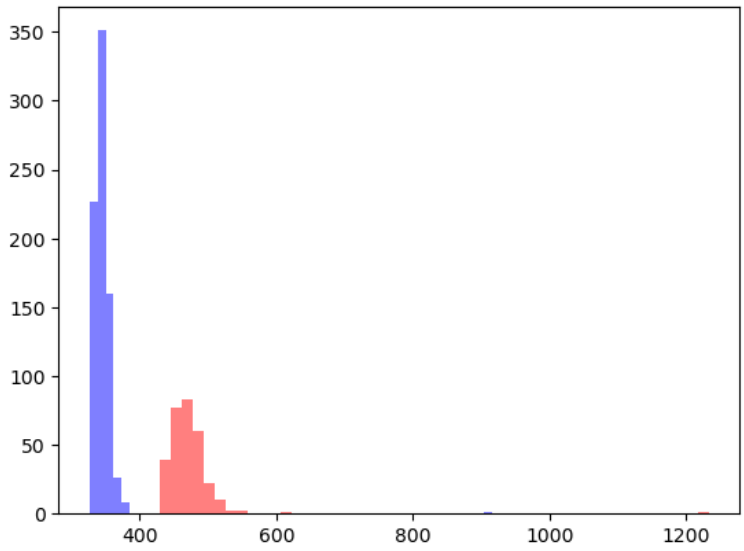


Figure 3.11 – Approximate mass distribution for a RealNVP model trained on CIFAR10 at the end of training (epoch 100). The x-axis represents the value of the approximate mass, the y-axis the number of occurrence in the dataset for each bin of value (group of values). In red, approximate mass values on ID data (CIFAR10), in blue on OOD data (SVHN). The red values should be smaller than the blue values.

dels. Indeed, in equation 3.33 we can find the likelihood formula given by an energy-based model where $E(x, \theta)$ is the energy function of the model and Z its normalization (intractable). Taking the gradient of the log-likelihood of such a model cancels the constant Z leading to equation 3.34. In this energy-based interpretation of the probability learned by the model, the approximate mass may have a direct physical interpretation as a conservative force derived from some energy level $E(x, \theta)$.

$$p(x, \theta) = \frac{e^{-E(x,\theta)}}{Z} \tag{3.33}$$

$$AM = \| \nabla_x \mathcal{L}(x; \theta) \| = \| \nabla_x E(x; \theta) \| \tag{3.34}$$

Furthermore, recent advances in generative models showed the importance of metrics based on the gradient of the log-likelihood. More specifically, denoising diffusion models [59] are models that rely on the estimation of the gradient of the log-likelihood of the noise in the noising process in order to generate an image through a diffusion process. These models imply an interpretation of the gradient of the log-likelihood as being equivalent to the noise level estimated in the image. The model then uses a Langevin sampling procedure to move the image in a direction that minimizes this noise (the diffusion

process) to generate a realistic image.

3.6 . Contributions : Regularization of the approximate mass

3.6.1 . First contribution : regularizing the likelihood with the approximate mass

My first approach for fixing the issue of OOD detection with normalizing flows involved regularizing the log-likelihood computed by a normalizing flow with the approximate mass. Indeed, I noticed that in the literature on normalizing flows, sampling is accomplished by temperature annealing on the probability distribution [49], therefore sampling from $(p_\theta(x))^{T^2}$ instead of $p_\theta(x)$, as illustrated in figure 3.12. This method was argued to bring data points closer to the learnt distribution modes.

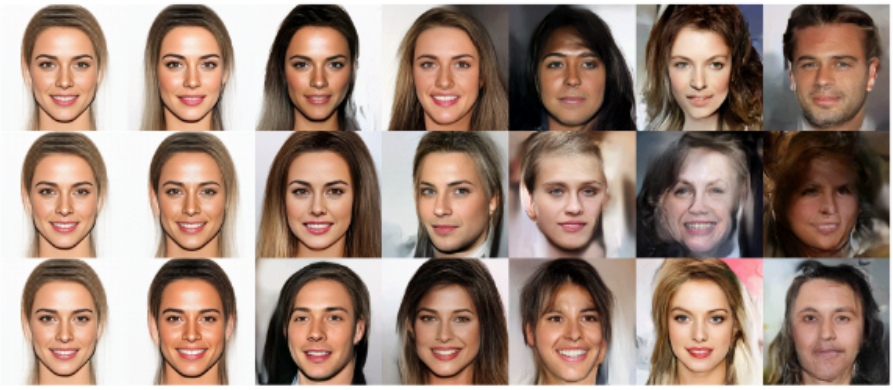
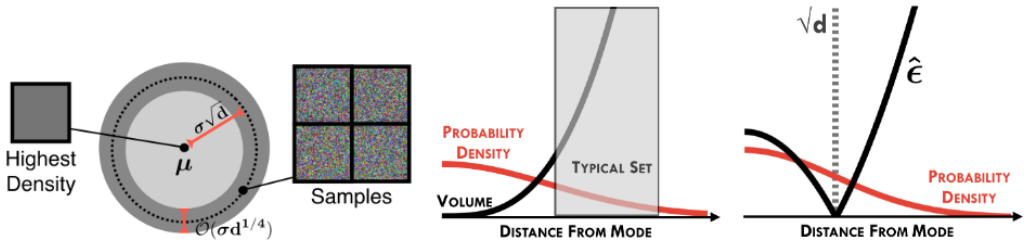


Figure 3.12 – Illustration of the effect of temperature annealing when sampling images. The higher the temperature parameter (close to 1, moving to the images on the right) the more diverse but less qualitative they will be ; the lower the temperature (close to 0, moving to the images on the left), the less diverse but the more qualitative.

Furthermore, in [106], OOD data is detected with a typicality test. In this paper, the authors argue that data in a training set is sampled from the typical set which is a set distinct from the set of highest probability density, as illustrated in figure 3.13.

This assumption is interpreted in [85] where they introduce the approximate mass $\| \frac{\partial \mathcal{L}(x;\theta)}{\partial x} \|$ as a measure of "in-distributionality". Intuitively, they interpret the typical set hypothesis [106] (TSH) as meaning that OOD regions may show higher likelihood values but they also show sharper variations around samples. The approximate mass translates this fact by measuring the gradient of the log-likelihood of data. In their paper, this measure is argued to yield better results than the log-likelihood with energy-based models. The rationale behind these techniques made us hypothesize that if the log-likelihood



(a) Gaussian Example (b) Illustration (c) Simulation

Figure 3.13 – Illustration of the typical set with a Gaussian distribution. The typical set is the annulus located at a distance proportional to $\sqrt{d}$ from the mean of the distribution, while the highest probability set should intuitively include the mean.

learned by the model varies sharply but still has some peaks in OOD regions, a regularization which "flattens" the peaks in the OOD regions to increase the likelihood in ID regions might benefit for likelihood estimation in the context of OOD detection.

Indeed, we know the learned density is constrained by construction to be normalized ($\int_{\mathcal{X}} p(x)dx = 1$) and positive ($p(x)$>0 for all $x \in \mathcal{X}$). Therefore, increasing the likelihood locally in ID regions while smoothing variations around them so that the neighboring points are assigned higher likelihood should move the probability mass over the input space. This intuition is similar to a Wasserstein regularization where we aim to "move" some probability distribution from some parts of the space to place it somewhere else in the same space.

Increasing the likelihood can be accomplished by a regular maximum likelihood training and smoothing values by adding a regularization term proportional to the approximate mass. The idea is to push likelihood values high in ID regions and, thanks the normalization and positivity constraints, push likelihood values down in OOD regions. Therefore, I introduced a new loss in equation 3.35 to train a normalizing flow that would encode both constraints.

$$-\mathcal{L}(x;\theta) \rightarrow -\mathcal{L}(x;\theta) + \alpha \parallel \frac{\partial \mathcal{L}(x;\theta)}{\partial x} \parallel \tag{3.35}$$

In equation 3.35, the term $\mathcal{L}(x;\theta)$ is the log-likelihood of the data point x computed by the model parameterized by θ. In order for to effectively move probability mass around the space $\mathcal{X}$, the loss should be formulated in terms of $p(x;\theta)$ rather than $\mathcal{L}(x;\theta) = \log p(x;\theta)$ as the negative log-likelihood does not fulfill the normalization constraint. However, the typical negative log--likelihood values returned by a normalizing flow by the end of its training were of the order of 1000, which means that the model would suffer from an underflow problem if the exponential was directly computed. Instead, I kept the objective described in equation 3.35 as it already encompasses all the

properties we need for my objective function. Indeed, by breaking down the approximate mass term in equation 3.36, we get :

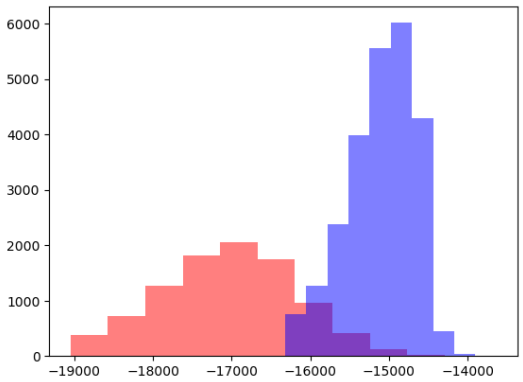
$$\| \frac{\partial \log p(x;\theta)}{\partial x} \| = \frac{1}{p(x;\theta)} \| \frac{\partial p(x;\theta)}{\partial x} \| \tag{3.36}$$

In other words, minimizing the approximate mass alone is equivalent to maximizing the likelihood and minimizing the term $\| \frac{\partial p(x;\theta)}{\partial x} \|$. Intuitively, this term can be interpreted as a measure of the amount of probability density around an input x as measured by the likelihood with parameter θ. However, in order to control the trade-off between maximum likelihood and likelihood smoothing, we need to work with both terms and introduce the parameter α to control the emphasis of the smoothing term in comparison to the likelihood term in the loss function.

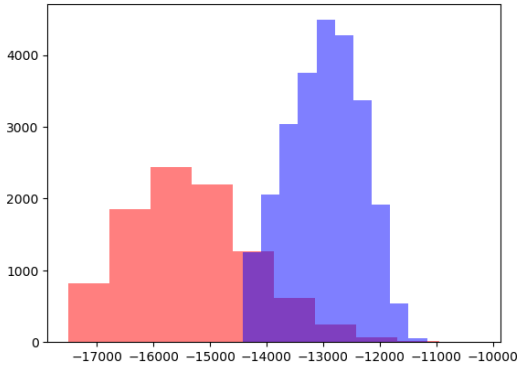
In order to test this intuition, I trained a normalizing flow with the new loss function. The training setting as well as the architecture of the model were the same as in [48] : I trained a RealNVP model [47] with 3 scales, 8 blocks in the st-network, a learning rate of 10^{-4}, batch size of 32, weight decay of 5.10^{-5} for 100 epochs on CIFAR-10 [80]. On FashionMNIST [107], the model is a 2 scale-RealNVP model with 6 blocks in the st-network, a learning rate of 5.10^{-5}, batch size 32 trained for 80 epochs.

Results on likelihood regularization are illustrated in figure 3.14. The histograms depict the distribution of the log-likelihood assigned by the model on the test set on ID data (in red, CIFAR-10) and on OOD data (in blue, SVHN). As we can see, the log-likelihood of the ID data assigned by the penalized RealNVP model is still lower than the log-likelihood of the OOD data. The overlap also indicates that discrimination of both classes stays a hard task.

Conclusion from this part show that penalizing the gradient of the log-likelihood may not be enough to constraint the log-likelihood of the model, possibly due to the dimensionality of the problem or the optimization problem which may not translate the previously built intuition. Indeed, working on the log-likelihood might lead indirectly to constraints on the likelihood but the log scaling might hurt performance. Therefore, in the next section I try another way of exploiting this intuition based on a method that was introduced in the scientific literature.



(a) End of the first epoch



(b) End of training

Figure 3.14 – Evolution of the log-likelihood with a RealNVP model trained on CIFAR-10 (red) and tested on SVHN (blue) with my regularization of the log-likelihood with the approximate mass. On the x-axis, the values of the log-likelihood, on the y-axis the number of occurrences of each bins (group of values) of log-likelihood.

3.6.2 . Second idea : regularizing the likelihood with a VAT-inspired loss

In [108], the authors introduce the notion of virtual adversarial training (VAT). This method is inspired by the original concept of adversarial training. Adversarial training is a supervised learning technique which requires the label of both the sane and the perturbed samples in the training set to train the model on both natural and adversarial examples in order to make it more robust. The idea of VAT allows to generalize adversarial training to a self-supervised training setting, where we may not have the label of all data instances. In order to train on unlabeled data, the model assigns to the data instance its most likely label (assign a "virtual label"), similarly to label propagation [109]. I will further describe adversarial training in a subsequent chapter 4.

The authors of [108] develop a regularization term that aims at isotropically smoothing the softmax output probability distribution learned by a classifier. This term, the LDS (Local Distribution Smoothness), is added to the cross entropy or negative log-likelihood loss and is proportional to the Kullback-Leibler divergence (defined in 3.37) between the learned softmax distribution and the same distribution taking perturbed inputs, as specified in equation 3.38.

$$D_{KL}(p||q) = \int_{x \in \mathcal{X}} p(x) \log(\frac{p(x)}{q(x)}) dx \tag{3.37}$$

$$LDS(x, \theta) = D_{KL}(p(y|x, \theta)||p(y|x + r_{vadv}, \theta)) \tag{3.38}$$

In equation 3.38, the term $r_{v_{adv}}$ is computed by approximating the direction of the highest perturbation of the KL divergence with the formula in equation 3.39.

$$r_{vadv} \approx \frac{g}{\| g \|_2} \tag{3.39}$$

The formula in equation 3.39 is the result of a finite difference approximation with step size ξ and a one-step power iteration computation initialized with a random unit vector d. The finite difference approximation allows to circumvent the computational complexity of computing the exact gradient of the KL divergence. The power iteration allows to compute the largest eigenvalue of the Hessian of the KL divergence $\nabla_r^2 D_{KL}(p(y|x, \theta)||p(y|x + r_{vadv}, \theta))$, which corresponds to the perturbation $r_{v_{adv}}$. This correspondence to an eigenvalue comes from a second order Taylor approximation of the KL divergence.

Where $g = \nabla_r D_{KL}(p(y|x, \theta)||p(y|x+r, \theta))$ at $r = \xi d$ with d a random unit vector.

As the LDS regularization term is applicable in self-supervised training, one could also extend it to an unsupervised training setting. This idea joins the intuition we worked with in section 3.6.1 where I attempted to smooth the likelihood function directly with an approximate mass penalization during training. As the idea failed to yield the expected results, that is to say reducing the likelihood of OOD data and increasing that of ID data, I decided to try it out again with the VAT approach. This time, I added to the loss function of the model a modified LDS term, written in equation 3.40.

$$-\mathcal{L}(x; \theta) \rightarrow -\mathcal{L}(x; \theta) - \alpha D_{KL}(p(x, \theta)||p(x + r_{vadv}, \theta)) \tag{3.40}$$

Where

$$D_{KL}(p(x, \theta)||p(x + r, \theta)) = \int_{x \in \mathcal{X}} p(x, \theta) \log(\frac{p(x, \theta)}{p(x + r, \theta)}) dx \tag{3.41}$$

In my case, in order to compute the KL divergence, we would need to have access to the exact value of $p_\theta(x)$ for a sample x drawn from the target distribution $p(x)$. However, the normalizing flow gives by default access to a log-likelihood because it eases computations, making a maximum likelihood training simply optimizing a sum of log-probabilities instead of a product of probabilities. Typical values of negative log-likelihood reach the order of magnitude of 10,000 which means that computing $p_\theta(x)$, which requires taking the exponential of $-\log p_\theta(x)$, would be rounded to 0 in a computer. This underflow makes evaluation of the loss, and thus optimization, impossible by direct computation of the KL divergence. Instead, I tried to approximate the quantity $p_\theta(x)$ with power series. Indeed, we can rewrite $p_\theta(x)$ as in equation 3.42.

$$p_\theta(x) = \exp \log p_\theta(x) = \sum_{n=0}^{\infty} \frac{\log p_\theta(x)^n}{n!} \tag{3.42}$$

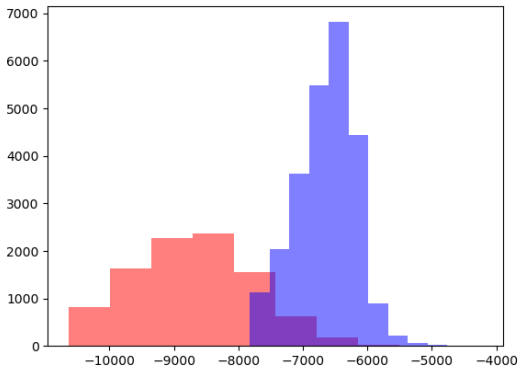
This power series can be estimated with a truncation (stopping the sum at a fixed index). However, this is a biased estimation of the exponential and there is little control one can have over its accuracy, the only choice to improve being to increase the number of iterations. However, if the terms in the sum do not decrease fast enough, choosing the right stopping index can add errors to the loss. Instead, I chose to use a statistical estimator of this power series, the Russian roulette estimator [110], in order to have an unbiased estimate of the series and some control over its variance. The Russian roulette estimator works by truncating the sum at a random index : at each iteration a Bernoulli sample $s \sim \mathcal{B}(p)$ of probability $p \in [0, 1[$ is drawn, if the outcome is 0 then the sum stops, otherwise it continues, as detailed in algorithm 2 where z_i is the i^{th} element of the series.

Algorithm 2 Russian Roulette Algorithm

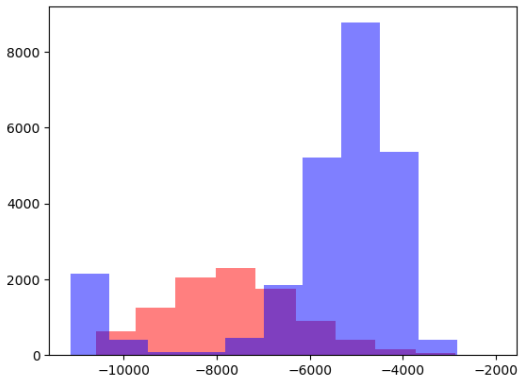
$SUM \leftarrow z_0$
$i \leftarrow 0$
$STOP \leftarrow 0$
while do $STOP = 0$
  $SUM \leftarrow SUM + \frac{z_i}{1-p}$
  $i \leftarrow i + 1$
  $STOP \sim \mathcal{B}(p)$
end while

This method of computing the penalization term was also not conclusive as the computation of the power series displayed a high variance, always showing too high values (in absolute scale) of the KL divergence term. Furthermore, although the model could learn, the log-likelihood showed weird beha-

vior, far from what I expected to happen (displacement of the in-distribution to higher values of log-likelihood than out-distribution) but also from the regular behavior. The log-likelihood histograms in figure 3.15 show that the model's log-likelihood can evolve in an erratic way during training, due to the high variance of the Russian roulette estimator. This makes this loss term particularly hard to deal with.



(a) End of the first epoch



(b) End of training

Figure 3.15 – An example of the evolution of the log-likelihood with a RealNVP model trained on CIFAR-10 (red) and tested on SVHN (blue) with the D_{KL} penalization. On the x-axis, the values of the log-likelihood, on the y-axis the number of occurrences of each bins (group of values) of log-likelihood.

The biggest issue with approximating the KL divergence term in the loss defined in equation 3.41 comes from the intractability of the integral. A Monte-Carlo approximation would be suitable in that case by turning the integral in equation 3.41 into equation 3.43. In this equation, one would draw the N samples from $p_\theta(x)$ with the sampling procedure provided by any normalizing flow. This approximation would however split training into two steps :

1. first, compute and make a gradient descent step to lower the negative

log-likelihood,

2. then, use the newly updated likelihood function to sample data from and calculate equation 3.43 over.

This would be very costly for a suitable approximation of the integral (which requires a lot of samples) as it would require $N + 1$ forward passes (1 for the likelihood computation and N for the samples in the KL divergence Monte-Carlo approximation) and N inverse passes, along with two separate back-propagations per batch. Furthermore, the number of parameters needed to make this algorithm work increases its complexity too much, between the α parameter in the loss, the ξ parameter in the finite difference step, the number of steps in the power iteration algorithm, the p parameter in the Bernoulli random variable in case we perform a Russian roulette estimation of the power series or the parameter N if we perform the D_{KL} estimation with the Monte-Carlo algorithm exposed above.

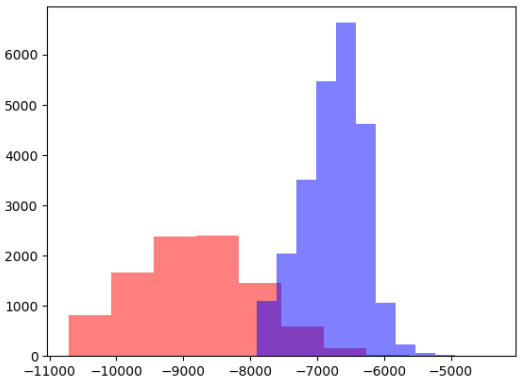
$$D_{KL}(p(x,\theta)||p(x+r,\theta)) \approx \sum_{i=1}^{N} \log p_\theta(x) - \log p_\theta(x+r) \tag{3.43}$$

Visual results of the experiments training the normalizing flow model (RealNVP with similar parameters and hyperparameters as in section 3.6.1) are represented in figure 3.16). I trained a model with my new loss with a penalty parameter $\alpha = 2$ for 100 epochs.

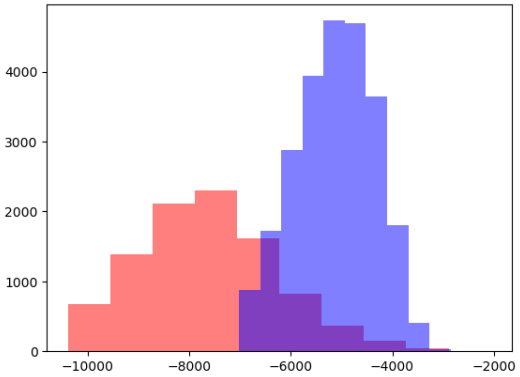
The histograms in figure 3.16 show that the log-likelihood the model assigns to the in-distribution still seems to be smaller than the log-likelihood of the out-distribution.

From these results in section 3.6.1 and 3.6.2, I concluded that regularizing the log-likelihood of a normalizing flow in order to introduce a bias in the learned distribution that would shift the assigned values between the in-distribution and out-distribution may not work. This is possibly due to the dimensionality of the problem as we are working on a probability space with values that can be very low because we are working on $p(x)$ instead of $p(y|x)$. The complexity of the problem in this case is huge since the number of dimensions of $\mathcal{X}$ (the feature space) is itself huge (working on image data). Meanwhile, the dimension of $\mathcal{Y}$ (the label set) is relatively small making manipulations on the probability density $p(y|x)$ more practical.

However, other elements in the scientific literature seem to challenge the fundamental limitations of likelihood-based OOD detection, as we will see in section 3.6.3.



(a) End of the first epoch



(b) End of training

Figure 3.16 – Evolution of the log-likelihood with a RealNVP model trained on CIFAR-10 (red) and tested on SVHN (blue) with the D_{KL} penalization. On the x-axis, the values of the log-likelihood, on the y-axis the number of occurrences of each bins (group of values) of log-likelihood.

3.6.3 . Entropic issues with the likelihood for OOD detection : the likelihood is not appropriate

In section 3.6, I presented several ideas I had to regularize the log-likelihood of the models in order to make OOD detection more practical and avoid limitations observed in the scientific literature as introduced in section 3.2. However, what if fundamentally the likelihood, as a metric for OOD detection, was not suitable for this task? The paper [111] questions this metric which intuitively seems natural, in the context of OOD detection, showing that the log-likelihood estimated by any model is fundamentally limited by a constant term, independent of the model's parameters, the entropy.

Equation 3.44 shows a decomposition of the log-likelihood of a model on average. Note that a perfect model has a KL divergence term $D_{KL}(p_\theta||p)$ equal to 0, naturally making the average log-likelihood equal to the entropy of the

random variable (matching the definition of the entropy). This equation indicates that fundamentally, the log-likelihood of any model, as perfect as it can be, will be influenced by the entropy of the source data. In equation 3.45, we replaced the distribution $p(x)$ we take the expectation over (in equation 3.44) by an out-distribution $q(x)$. This shows that a complex distribution $p(x)$, which shows a higher entropy $\mathbb{H}(p)$ than a less complex distribution $q(x)$, may be assigned on average a lower log-likelihood.

$$\mathbb{E}_{x\sim p(x)}(\log p_\theta(x)) = \mathbb{H}(p) - D_{KL}(p||p_\theta) \tag{3.44}$$

$$\mathbb{E}_{x\sim q(x)}(\log p_\theta(x)) = \mathbb{H}(q) - D_{KL}(q||p_\theta) \tag{3.45}$$

This mathematical explanation matches with empirical observations where less complex images (SVHN [90] or MNIST [2]) in comparison to the in-distribution (CIFAR-10 [80] or FashionMNIST [107]) are assigned lower likelihood. This mathematical formulation also explains why methods like likelihood ratios seem to work for OOD detection. In those methods, the entropy of the in-distribution or out-distribution is cancelled out by the entropy of an alternative distribution (which we take the ratio with). Equation 3.46 shows this phenomenon with a difference of log-likelihoods which is equivalent to a ratio of likelihoods. The density r_ϕ is the density of an alternative model called a *reference model* which is carefully chosen depending on the task (model of the background, or image compression technique). Equation 3.46 clearly shows that a likelihood ratio test depends only on the KL divergence between the reference distribution r_ϕ and the distribution being tested ($q(x)$ or $p(x)$).

$$\mathbb{E}_{x\sim q(x)}(\log p_\theta(x)) - \mathbb{E}_{x\sim q(x)}(\log r_\phi(x)) = D_{KL}(q||r_\phi) \tag{3.46}$$

All these elements hint that the idea of OOD detection with log-likelihood is fundamentally limited by a natural phenomena, which is the entropy as a fundamental upper-bound for the log-likelihood. One could alternatively think about training or tuning the model in order for the KL divergence to compensate this entropy term. Using divergence terms to detect OOD data would be a good choice as explained in section 3.2 where we covered a few models using such terms. However, based on 3.3.5 and 3.3.6, it seems that OOD detection techniques based on the *gradient space* surpass those relying on *feature space*. Alternative methods relying on the gradient of a certain metric with respect to the model's parameters seemed to be successful. Therefore, I oriented the thesis using alternative metrics in order to make models more robust and not rely on the model's likelihood directly for OOD detection.

3.6.4 . Observation on the behavior of the approximate mass

The approximate mass 3.36 is a gradient-space based metric, although not taken over the model's parameters. It is natural to test out the performance of this metric on OOD detection with flow-based models. Authors in [85] report that this metric works better for OOD detection than the likelihood of their energy-based model. However, it is also reported that this metric seems not to work appropriately with tractable likelihood models (e.g. : flow-based models) as the values of approximate mass seem to adopt the opposite behavior than energy-based models. This behavior is not desirable as ideally, the model should not have higher likelihood variations around ID data than OOD data.

I tested these claims by training a RealNVP model [47] on the CIFAR10 [80] and FashionMNIST [107] datasets following the same configuration (number of epochs, batch size, model architecture, optimizer etc.) as in [48]. What I actually observed during the experiments was more complex than the behavior reported in [85]. Indeed, by the end of the training, the approximate mass showed the problems mentioned above. However, this behavior only appeared in the last epochs and was not apparent at the beginning of training. This seems to indicate that the problem does not come from the family of model but rather the type of training, that is to say maximum likelihood training. It is therefore possible to perform OOD detection with normalizing flow in a suitable way, instead of assuming that approximate mass values for ID data will be greater than approximate mass values for OOD data. All one needs is to control the variations of the log-likelihood around input data in order for the approximate mass not to take too high values compared to OOD data.

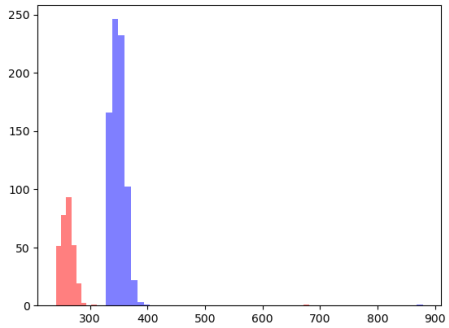
3.6.5 . Fixing the overfitting of the approximate mass

A previous paper [112] performed double backpropagation of the model's loss with respect to the model's inputs to control its variations. The typical training objective introduced in this paper includes a loss and its gradient with respect to the inputs, as represented in equation 3.47. This objective is argued to be better for the generalization capabilities of models and to avoid sensitivity by reducing variations of the loss around input data points.

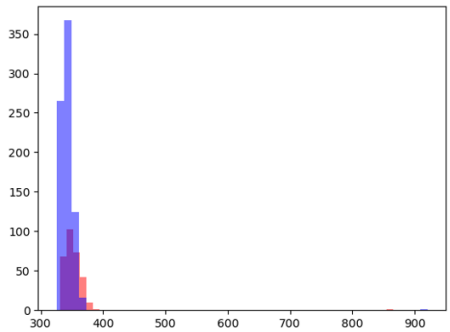
$$L(x;\theta) = l(x;\theta) + \alpha \parallel \nabla_x l(x;\theta) \parallel \tag{3.47}$$

My training objective 3.35 in section 3.6.1 follows the same equation as in 3.47. We can also observe that during training, the approximate mass does not evolve in the same way as in the vanilla trained model. The likelihood does not seem to overfit around ID data points and does not assign higher approximate mass values for ID data than for OOD data, matching the behavior observed on energy-based models in [85] 3.17.

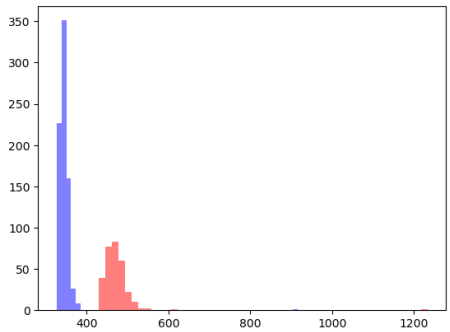
My training objective bears some resemblance to other similar training



(a) Epoch 60



(b) Epoch 85



(c) Epoch 100

Figure 3.17 – Approximate mass distribution for a RealNVP model trained on CIFAR10 at three training epochs (epochs 60, 85 and 100). On the x-axis, the values of the approximate mass, on the y-axis the number of occurrences of each bins (group of values) of approximate mass. In red, approximate mass values on ID data (CIFAR10), in blue on OOD data (SVHN). The approximate mass should be like in epoch 60 for OOD detection.

objectives such as GradCon [100] and GradNorm [99]. These two training methods involve computing the gradient of a loss term with respect to model's parameters. On inference time, the gradient metric part corresponds to the OOD detection metric. In my case, in section 3.6.1 I first tried to use the first loss term (the log-likelihood) as an OOD detection metric. However now, similarly to [100] and [99], the second term in the optimization objective is my OOD detection metric.

This family of objective functions is however problematic because of the computational complexity involved in the double backpropagation operation. Indeed, two backpropagation operations per training iteration can be extremely costly. Some mathematical study already analyzed the family of double backpropagation loss functions [113] and found a way to reduce calculations by a third for Frobenius-norm penalties on Jacobian (i.e. : first-order derivative penalizations). Future works on my method could exploit the results derived in [113] for the specific case of unsupervised normalizing flows training (which rely on a specific architecture that may not be covered by the mathematical analysis in this paper).

Intuitively, this regularization technique can be seen as a generalization of data augmentation. Indeed, a first-order penalty is expected to isotropically smooth the loss around input data. Data augmentation on the other hand aims at minimizing the loss function on a finite amount of data sampled by modifying data from the training set (by adding noise, rotating, flipping...). As the first-order constraint penalizes the model to smooth the loss isotropically around each training data point, this has the effect of bringing loss values closer to each other for close data points.

Further investigations can also test few-shot learning capabilities [114] (or more generally, meta-learning [115]) of this training approach in an unsupervised context. In few-shot learning, the goal is for a model to learn a new class from few instances of this task. Several methods exist to tackle this issue [116], some of them including memory-augmented neural networks [117] which have the ability to store information about new instances within their internal memory module to reuse later in inference. Some other methods, prototypical networks [118], use "prototypes" and a similarity measure to decide if a new instance on test time belongs to one of those new classes. A prototype is a class centroid of the latent representation of data instances of the same class. When a neural network learns to classify instances, the latent representation of these classes can be separated from the latent space, which can be illustrated with a t-SNE [119] visualization (example in figure 3.18). The idea behind protypical networks is that the new class instances will have their own clusters during training, as such on inference one can discriminate data instances by assigning to an undetermined input the class of the "nearest" prototype (a class-cluster centroid). In other words, on inference, the model :

1. takes the distance between the input latent representation and the learned prototypes with a similarity measure (e.g. : Mahalanobis distance, cosine similarity);
2. assigns to the input the class of the closest prototype/cluster

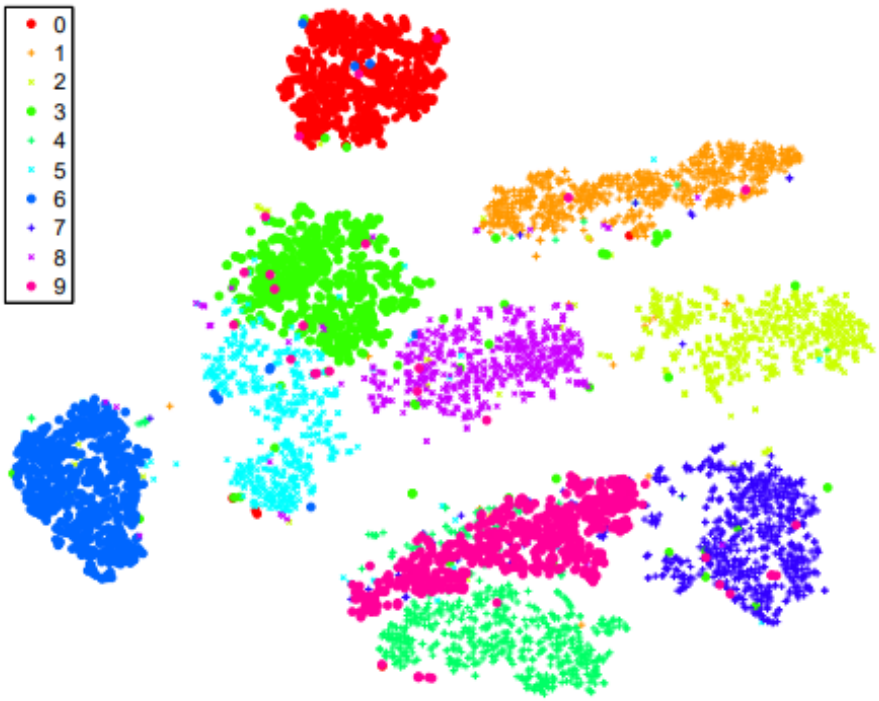


Figure 3.18 – Illustration of a projection with the t-SNE algorithm for a neural network trained on the MNIST dataset. We can clearly see clusters formed for each class, indicating that the model efficiently separated the input space. Credit to the original paper on t-SNE [119].

In the training procedure, smoothing the loss around input data is expected to naturally yield homogeneous clusters, where each data point of a given class is expected to lie in a hypersphere centered around the prototype. I expect this method to give a latent representation for new classes that would be more exploitable by the model on inference as the natural similarity measure, the log-likelihood, is smoothed out for latent points near the learned centroid. In order to work, the new training method should include the log-likelihood augmented by the label information, for example by penalizing the log-posterior of a DIGLM [1] model.

Aside from those two point which are left for future work of my approach, we focus now on the results in OOD detection, testing two different settings. Each setting corresponds to different configurations of distribution shifts. We want to ensure first that the model classifies data correctly between OOD and

ID and also whether the method is sensitive to specific distribution shifts, as introduced in section 3.1.

3.6.6 . Results on OOD detection

My paper [120] [10] exposes my results in OOD detection,as defined in section 3.1, that is when ID and OOD data show both in covariate and semantic shifts.

I test my approach in a classic OOD detection setting with the CIFAR-10 [80] and FashionMNIST [107] datasets as in-distribution sets. For the models trained on FashionMNIST, I use the MNIST[2] KMINST [121] and EMNIST [122] datasets as OOD distributions while the models trained on CIFAR-10 are tested against the SVHN [90], DTD [123], GTSRB [124], Places365 [125] and iNaturalist [126] (split into "Animalia" and "Plantae" parts) datasets as out-of-distribution sets. I chose these datasets as they show no overlapping in their labels.

Common metrics in OOD detection, that I chose to use as well in my study include AUROC (area under the ROC curve), AUPR (area under the precision-recall curve) and TNR at 95% TPR (True Negative Rate at a fixed level of 95% True Positive Rate).

I compare my results to state-of-the-art methods that are commonly used in OOD detection benchmarks : Generalized ODIN [93], an extension of the original ODIN [91], Mahalanobis [95] and Energy-based OOD detection [98]. The ODIN family of methods rely on adversarial perturbations and smoothing the output distribution. The Mahalanobis uses Gaussian discriminant analysis on the feature space to detect OOD samples. The energy-based method uses an energy-based interpretation (different from Joint Energy-based Model [85]). The energy-based model also requires the use of OOD data to train its upper energy bound. The energy-based model is trained with both CIFAR-10/FashionMNIST as ID data and SVHN/MNIST as OOD data.

The models used here are RealNVP models trained on the ID datasets. For grayscale images (FashionMNIST and MNIST), the model is a RealNVP model with 4 blocks and 2 scales (≈10 million parameters) and for color pictures (CIFAR-10 and SVHN) it is a RealNVP model with 6 blocks and 3 scales (≈60 million parameters).

Those methods are initialized with a similar number of parameters, which changes with the dimension of data. I kept the same models as the ones trained in section 3.6.1 and make the other models in the benchmark match this number of parameters.

Finally, the number of OOD and ID data is equal in order to have meaningful AUROC and TNR measures as the ROC curve is sensitive to class im-

10. Titled "*Improving Normalizing Flows With the Approximate Mass for Out-of-Distribution Detection*", accepted at the Generative Computer Vision workshop of CVPR 2023 and available in the proceedings of the conference.

balance [103]. This attention I brought on the class balance and the number of parameters is rarely done in the field of OOD detection. If this balance is not present, the models are not strictly comparable with each other and the values of AUROC can not be interpreted easily.

Results of the experiments can be found in table 3.1, where I report, for each model and pairs of datasets, measured scores for each metric.

Table 3.1 shows state-of-the-art performance for the penalized model. The average AUROC is of 97.0% on the CIFAR-10 benchmarks, 98.7% on the FashionMNIST benchmarks and a global average AUROC of 97.6% on all datasets. On the other hand, the average AUROC is of 62.4% for Generalized ODIN, 83.8% for Mahalanobis and 84.0% for Energy (or 79.44% when not taking into account MNIST and SVHN as these datasets are used during training to tune the energy bounds).

The results show state-of-the-art performance on OOD detection. The approximate mass regularization performs well when there is both a covariate shift and a semantic shift between the in-distribution and the out-distributions. The results surpass by 16% on average the AUROC of the best baseline, Energy (with its OOD datasets included). Furthermore, the model shows a more con-sistent behavior than the baselines as my penalized training yields good performance on all datasets in the benchmark while other models are more irregular in their classification performance depending on the OOD dataset. Finally, a gap can be observed between TNR@95%TPR values and the AUROC values for the other models in the benchmark. I investigated this discrepancy by plotting their respective ROC curve where we can observe that lowering the value of the TPR increases the TNR (for example, TNR@95%TPR is greater than TNR@99%TPR but lower than TNR@90%TPR).

These excellent results in OOD detection add further understanding on the approximate mass metric developed in section 3.5. Indeed, recalling equation 3.36, we see that the approximate mass can be interpreted as the local rate of change of the likelihood around an input x, relatively to the value of the likelihood at this point. In other words, the approximate mass weighs the variations of likelihood by the value of the likelihood itself. We observed in section 3.6.1 that the log-likelihood (and therefore the likelihood) distribution does not seem to change its distribution between OOD and ID data in figure 3.14 even after adding the penalization term. Therefore, the approximate mass regularization did not change the relative distribution of likelihood values between ID and OOD data. However, the approximate mass distribution between ID and OOD data was affected by the regularization term. This means that the "concentration term" $\| \frac{\partial p(x;\theta)}{\partial x} \|$ had an impact during training. On the one hand, the concentration term of the approximate mass should indeed be

Model	Training dataset	OOD dataset	AUROC	AUPR	TNR @95%TPR
G-ODIN	CIFAR-10	SVHN	0.810	0.804	0.355
Mahalanobis	CIFAR-10	SVHN	0.936	0.926	0.758
Energy	CIFAR-10	SVHN	**0.999**	**1.0**	**1.0**
Approximate mass	CIFAR-10	SVHN	0.969	0.969	**1.0**
G-ODIN	CIFAR-10	iNaturalist	0.581	0.552	0.124
Mahalanobis	CIFAR-10	iNaturalist	0.745	0.715	0.290
Energy	CIFAR-10	iNaturalist	0.647	0.902	0.872
Approximate mass	CIFAR-10	iNaturalist	**0.968**	**0.906**	**0.994**
G-ODIN	CIFAR-10	iNaturalist (plants)	0.581	0.552	0.124
Mahalanobis	CIFAR-10	iNaturalist (plants)	0.731	0.706	0.166
Energy	CIFAR-10	iNaturalist (plants)	0.587	**0.881**	0.077
Approximate mass	CIFAR-10	iNaturalist (plants)	**0.968**	0.799	**0.991**
G-ODIN	CIFAR-10	DTD	0.882	0.887	0.53
Mahalanobis	CIFAR-10	DTD	0.857	0.788	0.576
Energy	CIFAR-10	DTD	0.734	0.912	0.341
Approximate mass	CIFAR-10	DTD	**0.978**	**0.979**	**1.0**
G-ODIN	CIFAR-10	Places365	0.616	0.577	0.157
Mahalanobis	CIFAR-10	Places365	0.522	0.518	0.08
Energy	CIFAR-10	Places365	0.753	**0.937**	0.177
Approximate mass	CIFAR-10	Places365	**0.968**	0.936	**0.978**
G-ODIN	CIFAR-10	GTSRB	0.411	0.450	0.072
Mahalanobis	CIFAR-10	GTSRB	0.764	0.681	0.356
Energy	CIFAR-10	GTSRB	0.890	**0.975**	0.582
Approximate mass	CIFAR-10	GTSRB	**0.970**	0.970	**1.0**
G-ODIN	FashionMNIST	MNIST	0.535	0. 611	0.008
Mahalanobis	FashionMNIST	MNIST	0.995	0.995	0.994
Energy	FashionMNIST	MNIST	**1.0**	**1.0**	**1.0**
Approximate mass	FashionMNIST	MNIST	0.994	0.971	**1.0**
G-ODIN	FashionMNIST	EMNIST	0.870	0.885	0.425
Mahalanobis	FashionMNIST	EMNIST	**0.995**	**0.995**	0.982
Energy	FashionMNIST	EMNIST	0.990	0.998	0.955
Approximate mass	FashionMNIST	EMNIST	0.969	0.969	**1.0**
G-ODIN	FashionMNIST	KMNIST	0.328	0.391	0.019
Mahalanobis	FashionMNIST	KMNIST	0.990	0.990	0.948
Energy	FashionMNIST	KMNIST	0.981	0.995	0.917
Approximate mass	FashionMNIST	KMNIST	**1.0**	**0.996**	**1.0**

Table 3.1 – Results on OOD classification for different state-of-the-art models. The energy model is trained with the CIFAR-10 (resp. FashionMNIST) as ID data and SVHN (resp. MNIST) datasets as OOD data.

low for inputs x that are seen (or close to data seen during training) by the model as variations are controlled by the regularization term around this input. On the other hand, unseen data or rather data that does not seem to follow the training distribution have high variations as the model did not explore this part of the input space. As such, despite a larger average value of likelihood on the OOD dataset compared to the ID dataset, the values of the concentration term weigh in by taking even higher values. On the other hand the ID dataset may have smaller values of likelihood but since the model partially explored the ID data space, it assigns to these inputs a smaller concentration term. Overall, this behavior explains why the approximate mass is almost consistently lower for ID data than for OOD data.

Due to the nature of the OOD detection problem which includes two separate types of distribution shifts, I now aim to find out what distribution shift affects performance in OOD detection more. In the next part, I will test out whether the approximate mass is suitable for semantic shift by experimenting how this metric performs in anomaly detection.

3.6.7 . Results on class anomaly detection

In class anomaly detection (or anomaly detection or anomalous class detection), the goal is to train the model on a single class in a dataset (corresponding to its in-distribution) while the remaining classes of the dataset are treated as the out-distribution. For example, if we trained a model on the class "1" of the MNIST dataset, the ID data on inference time would be data drawn from the "1" class in the MNIST distribution while all the other classes would be the OOD distribution.

The methodology is to split a dataset in an ID set made of only one class of the dataset while the other classes are OOD. In order to evaluate the model, it is trained on a single class of the MNIST (or CIFAR-10) training data then tested on a split of the test set between the ID class and anomalous classes (the remaining classes).

I compare my approach to two state-of-the-art models in the domain : OCGAN and GradCon. OCGAN [101] is based on the features extracted in the latent space by a GAN [61]. GradCon [100] is based on a gradient metric with respect to the model's features (instead of the input) of the reconstruction loss of a VAE [40]. The compared models are the same as in the previous study in section 3.6.6 with the number of parameters kept the same for their respective settings (grey-level images or RGB images).

Similarly to [100], I use the MNIST [2] and CIFAR-10 [80] datasets to test performance in an anomalous class setting.

The reported metric is the AUROC, as is usually done in this setting, in table 3.2 and table 3.3.

Model	0	1	2	3	4	5	6	7	8	9
GradCon	0.995	**0.999**	**0.952**	**0.973**	0.969	0.977	**0.994**	0.979	0.919	**0.973**
OCGAN	**0.998**	**0.999**	0.942	0.963	**0.975**	**0.980**	0.991	**0.981**	0.939	**0.981**
Approximate mass	0.969	0.064	0.629	0.924	0.796	0.946	0.977	0.682	**0.969**	0.572

Table 3.2 – AUROC results on anomalous class detection on MNIST for different models.

Model	Airplane	Automobile	Bird	Cat	Deer	Dog	Frog	Horse	Ship	Truck
GradCon	0.760	0.598	0.648	0.586	**0.733**	0.603	0.684	0.567	0.784	**0.678**
OCGAN	0.757	0.531	0.640	0.620	0.723	0.620	0.723	0.575	0.820	0.554
Approximate mass	**0.855**	**0.605**	**0.756**	**0.835**	0.574	**0.801**	**0.797**	**0.862**	**0.871**	0.656

Table 3.3 – AUROC results on anomalous class detection on CIFAR-10 for different models.

The average measured AUROC is of 75.2% on MNIST and 76.1% on CIFAR-10 with my penalized RealNVP approach. The results reported in [100] are an average AUROC of 97.3% on MNIST and 66.4% CIFAR-10 for GradCon, and 97.5% on MNIST and 65.7% CIFAR-10 for OCGAN. This experiment shows that the approximate mass penalized model is sensitive to the semantic of data to some extent as it is able to detect label shifts. The gap in performance between the MNIST and CIFAR-10 datasets is much lower than the one measured for OCGAN or GradCon, consistently with results in section 3.6.6. However, the AUROC is distributed unevenly across classes in the datasets, some classes being seemingly less distinguishable by the model than others (e.g. : class 1 in the MNIST dataset in Table 3.2).

3.7 . Conclusion of this chapter

In this chapter, we saw how the use of likelihood-based metrics in OOD detection has limitations. I attempted several methods of circumventing the issue of the likelihood in OOD detection :

1. I first attempted to use the approximate mass in an way to fix this issue by adding constraints on the likelihood that would make it smoother, expecting the likelihood to increase in the neighborhood of ID data at the cost of the likelihood on OOD data neighborhood, thus decreasing on these subsets. I used the approximate mass as a regularizer of the smoothness of the likelihood function. This approach did not seem to yield the expected results.
2. Then, I built on the same intuition a regularizer inspired by the VAT method, that also did not yield expected results.

3. The seemingly impossibility to regularize the likelihood in a way that would make it suitable in OOD detection might be due to the complexity of the space we are working on, the feature space $\mathcal{X}$ instead of the label space $\mathcal{Y}$ which is countable and finite.

After more research in the literature on OOD detection, it turned out that the likelihood was no suited for OOD detection. Further investigation of the approximate mass led to the following contributions :

1. The creation a new methodological standard to test OOD detection algorithms.
2. The discovery of a property of the approximate mass during training. This metric seems to behave in an inconvenient way during training, ending up overfitting.
3. The introduction of a new method of regularizing normalizing flows which controls this metric to behave correctly.
4. The observation of the limits of this metric : it performs excellently on OOD detection but had more mitigated results on anomaly detection, suggesting that the metric is more suitable for covariate shift detection than semantic shift detection.

The approximate mass seems to be related on the local smoothness of the likelihood relatively to the value of the likelihood. This allows to enrich the likelihood metric with a measure of the regularity of the model on certain parts of the image space. Therefore, if a model hasn't seen a certain part of the image space (i.e. : OOD data), it will be less regular on this part thus assigning a higher approximate mass.

In order to test performance of the approximate mass on covariate shift, we will follow this study in the next chapter by looking at the behavior of the model in adversarial attack detection. as we will see in the next chapter, the inputs generated by adversarial attacks can be seen as a certain type covariate shift.

Future works on my method exposed in this chapter could exploit the results derived in [113] for the specific case of unsupervised normalizing flows training (which rely on a specific architecture that may not be covered by the mathematical analysis in this paper). The method could also be extended to supervised learning where the new training method should include the log-likelihood augmented by the label information, for example by penalizing the log-posterior $\log p(y|x)$ of a DIGLM [1] model, which is a model that will be covered in the next chapter. Finally, some mathematical analysis would make the understanding of the approximate mass easier, notably through the lens of information geometry as the minimization of the approximate mass might

be equivalent to finding a geodesic [11], or measure theory by assimilating the approximate mass to a Wasserstein distance.

11. Shortest path in a general space.

4 - Adversarial defense

Chapter 3 introduced a background on OOD detection. As a reminder, an OOD input is an input that comes from outside the target distribution one wishes to model (the same distribution as the training dataset of the model). The domain of adversarial machine learning emerged in the last few years and may be interpreted as a form of OOD input. This section introduces the domain of adversarial machine learning by giving some background on the domain by presenting common attacks and defense mechanisms, problems in the domain as well as my own contributions.

4.1 . Adversarial machine learning : introduction

The domain of adversarial machine learning is a branch of machine learning that is concerned with studying the robustness of machine learning algorithms, specifically deep learning models, against inputs that have been added with noise specifically crafted to perturb the model. Indeed, some observations were made [127] that it is possible to tailor perturbations to images in order to fool the model into misclassifying it. The perturbation is often designed in such a way that the difference between the original image and the perturbed image is unnoticeable for the human eye. The misclassification may often result in overconfidence from the model, in other words the model assigns a high softmax probability to the wrong label. An illustration of an adversarial attack is represented in figure 4.1. In this figure, we can see that even though the original model successfully classifies the image with the right label, "panda", adding a noise with small magnitude results in a misclassification as the model classifies the perturbed image as a "gibbon" with an even higher confidence (99.3% instead of the original 57.7%) and despite the difference between both images being barely noticeable for the human eye.

The adversarially perturbed input is modified only on its features, making adversarial attacks a type of covariate shift. The shift is due to noise added to the image, making the problem of classification of adversarial images a problem of generalization. Indeed, the adversary mainly exploits the lack of generalization of the model as a way to craft its noise.

Some criticism may be made about adversarial machine learning, such as whether the perturbations added to the images can really be computed in a real-time environment, whether the model's parameters may be accessed by the attacker or whether the perturbation may be injected into the image. It is critical to understand that as the domain of deep learning quickly moves towards the extensive use of "foundation models" [130] [131], which are self-

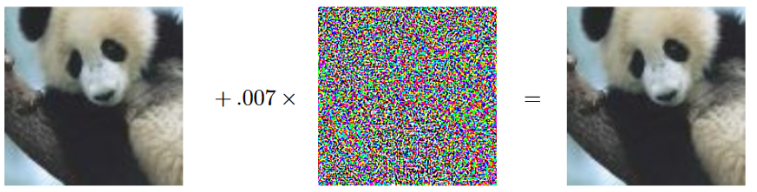


Figure 4.1 – Illustration of an adversarial attack performed on a GoogLeNet network [128] trained on the ImageNet dataset [129]. A small imperceptible noise was added with the FGSM attack [92]. It resulted in the model misclassifying the image with more confidence (higher softmax score) than its classification results on the original image. From the FGSM paper [92].

supervised often open-sourced models trained on a vast amount of data in order to be used later on in downstream tasks [132] [133], and knowing that adversarial perturbations can be transferable [134], it is essential to make deep learning models more robust. Indeed, knowing that these models can be accessed by anyone can makes the discovery of potential backdoors [135] [136] and attacks on models derived from them more likely in deployed models. The domain also remains important as it allows one to quickly find images the model does not generalize on, instead of exhaustively finding noise that efficiently perturbs the model.

4.2 . State-of-the-art : adversarial attacks

The state-of-the-art in adversarial machine learning starts with adversarial attacks. In the adversarial machine learning setting, a threat model must first be established. The threat model is dependent on what information the adversary has access to. Threat models are mostly divided into the "black box" threat model and the "white box" threat model [137]. Another information level called "gray box" also exists but is rarely seen in the literature.

In the black box threat model, the adversary does not have access to the parameters of the model. It is not always clear though if the model has access to the network's architecture, loss function or generally to the model's learning algorithm. Generally speaking, implementations have no knowledge about the training of the classifier and simply assume a classic training with a cross-entropy loss. The adversary usually has the possibility to query the network and get the output, however some restrictions can be given on the nature of the output, whether the model has access to the label output (the decision of the model or a one-hot, binary encoding of the label output) or the raw output (logits or softmax output).

In the white box threat model on the other hand, the adversary has access to the parameters of the model and potentially other information on the network. More accurately, the attacker not only get access to the architecture and the weights of the network but also has the possibility to query it, get its output and make computations on the model such as gradient computations.

Information level between both threat models falls in the gray box threat model [137]. In this threat model, knowledge on the classifier is limited to the structure of the target model.

This section covers the state-of-the-art methods in adversarial attacks, mainly the ones that I used in my experiments. Efforts to standardize evaluations in adversarial machine learning in the previous years have lead to the development of several repositories to make testing of new defense algorithms on previous attacks easier. I settled on the AutoAttack [138] [1] as it evaluates models by using a set of four attacks to reliably evaluate robustness, selected to represent a variety of threat levels from black-box attacks to white-box attacks with various loss functions in their objective. The attacks are also implemented in a way that most hyperparameters of the attacks are fixed therefore no tuning is required to test a new classifier, thus making evaluations homogeneous. This Python library implements the APGD-CE [138] (Automatic PGD attack [138] with Cross-Entropy loss), APGD-DLR [138] (Automatic PGD attack [138] with Difference of Logits Ratio loss), Square Attack [139] and FAB [140] attacks, which I will introduce in the next paragraphs.

The Automatic PGD attack (APGD) is a variant of the PGD (Projected Gradient Descent) attack with momentum [141] [142] and was introduced in the AutoAttack paper [138] in order to fix issues in the PGD attack and let as few hyperparameters free as possible. The PGD attack is a white-box attack that uses the projected gradient descent optimization technique. In equation 4.1, we see the iteration process starting from a benign input x^0 that is progressively transformed to an adversarial example by taking a step of size η in the direction of the gradient of the loss function L and then projecting the result in the ϵ-ball centered around the original benign input x^0. An illustration of the PGD attack is provided in figure 4.2 [2].

$$x^{k+1} = \mathcal{P}_{\mathcal{B}_\epsilon(x^0)}(x^k + \eta^k \nabla_x L(x^k)) \tag{4.1}$$

APGD aims at tackling the suboptimal step size η which does not guarantee convergence and performance, its lack of sensitivity to budget (the number of iterations does not translate into better attack results), and the blindness of the attack to the trend of the evolution of the loss during the search of the perturbation. It separates the iteration process into an exploration step to

1. https ://github.com/fra31/auto-attack/tree/master
2. Taken from https ://towardsdatascience.com/know-your-enemy-7f7c5038bdf3

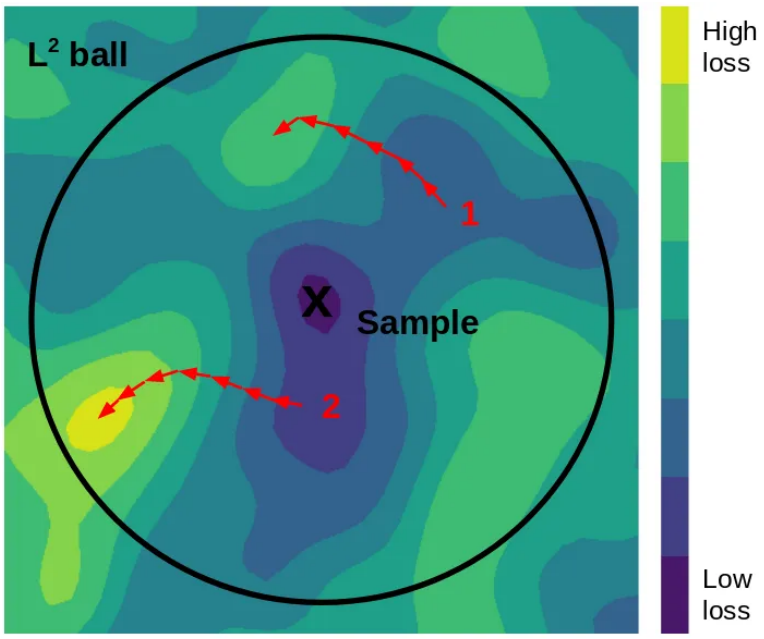


Figure 4.2 – Illustration of the PGD attack. The sample x in the middle of the circle is the clean sample. The first red arrow converges to a region of low loss within the circle while the second arrow converges to a high-loss region inside the same circle, thus producing samples that are mislabeled by the model.

search for a good set of initial points, then an exploitation step where the attacker maximizes the knowledge accumulated so far. The transition between both phases is done by progressively reducing the step size. APGD follows the original PGD algorithm but adds a momentum term. For every step k, the algorithm updates its terms following equation 4.2. The step η^k is updated by halving its value at some fixed checkpoints. The parameter ρ represents the fraction of the total update step we want the algorithm to successfully increase the loss.

$$\begin{aligned} x_{temp}^{k+1} &= \mathcal{P}_{\mathcal{B}_\epsilon(x^0)}(x^k + \eta^k \nabla L(x^k)) \\ x^{k+1} &= \mathcal{P}_{\mathcal{B}_\epsilon(x^0)}(x^k + \rho(_{temp}^{k+1} - x^k) + (1-\rho)(x^k - x^{k-1})) \end{aligned} \tag{4.2}$$

In equation 4.2 is typically the cross-entropy loss (CE) as it is assumed to be the most common training objective for a neural network trained on a classification task. However, in the context of an attacker searching for a perturbation through optimization techniques, the gradient of the cross-entropy $\nabla_x CE(x, y)$ with y the label of the model reaches 0 when the model's softmax output reaches 1 for the true label (and subsequently 0 for the wrong ones). The authors of [138] propose the DLR (Difference of Logits Ratio Loss), defined in equation 4.3, as a new loss where π is the ordering of the components of

the logits z (activation before the softmax) in decreasing order. This loss is still in $[0, 1]$ even when x is correctly classified.

$$DLR(x, y) = -\frac{z_y - max_{i \neq y} z_i}{z_{\pi_1} - z_{\pi_2}} \tag{4.3}$$

The DLR loss can be modified in the APGD-DLR objective to make it a targeted attack as shown in equation 4.4. This objective aims at getting $z_t > z_y$ and ensure the loss is not constant.

$$Targeted - DLR(x,) = -\frac{z_y - z_t}{z_{\pi_1} - \frac{(z_{\pi_3} + z_{\pi_4})}{2}} \tag{4.4}$$

The FAB attack [139], or Fast Adaptive Boundary attack, produces minimally distorted adversarial examples with respect to any L_p-norm, $p \in 1, 2 \inf$ for a given point x^0 initially correctly classified by the classifier f as c. This attack is close in principle to DeepFool [143] but has a higher quality as it has no incentive to find an adversarial perturbation close to the point x^0. Indeed, the DeepFool attack has an objective to minimize the Euclidean distance between the original sample and the adversarial sample. The FAB attack finds the point that is closest to the decision boundary between the true class and the target class of the attack while respecting the box constraint of the attack, that is to say the distance between the original point x^0 and the corrupted point x^k has to be lower than a threshold δ. It linearizes the classifier f at the point x^k at iteration k to compute the box-constrained projections of x^k respectively to x^0 onto the approximated decision hyperplane then makes a convex combination of these projections depending on the distance from x^k and x^0 to the decision hyperplane. Finally, the attack makes an extrapolation step to find the adversarial sample. The iteration process to generate adversarial samples is written in equation 4.5 where C is the box constraint $\| x^0 - x^i \| < \delta$, π_s is the decision hyperplane separating the true label l and the target label c of the attack, $\alpha \in [0, 1]$ is the convex combination hyperparameter.

$$x^{i+1} = (1 - \alpha) proj_p(x^i, \pi_s, C) + \alpha proj_p(x^i, \pi_s, C) \tag{4.5}$$

Finally, the Square attack [140] is a black box attack that uses random search to optimize the loss function within a given radius of attack. Random search is a greedy optimization method that randomly samples a value in the given set and selects this value if it minimizes/maximizes the loss compared to the previous value. For the Square attack however, sampled images $\hat{x}$ are constructed such that they remain on the boundary of the L_p-norm ball at every iteration before projecting them onto the segment $[0, 1]^d$. This way, the changes are localized in the image so that at each step only a small fraction of neighboring pixels shaped into squares is modified. The side length of the

square decreases at each iteration following a fixed schedule. Then, the algorithm picks a new perturbation δ randomly and adds it to the current sample $\hat{x}$. The updated sample is kept only if it decreases the loss. The algorithm stops when an adversarial example is found.

4.3 . State-of-the-art : adversarial defenses

The most efficient state-of-the-art method in adversarial defense is adversarial training [92]. This mode of training uses adversarial samples to train the model in order to make the model generalize on these instances. However, the lack of "awareness" coming with this technique, due to the fact that the model still classifies data irrespective of its adversarial nature, wouldn't allow raising alarms in a deployed system or understanding failures of the classifier. Furthermore, some evidence from previous works [144] seem to show that increasing robust accuracy through adversarial training may also lead to a decrease in clean accuracy. It can be proved that there is an underlying trade-off between generalization for better robustness and generalization for better accuracy. This is argued in [144] to be due to the differing nature of the features that robust classifiers learn compared to those a standard classifier learn. Therefore, detection algorithms started being explored in the recent works.

4.3.1 . Adversarial detection

Adversarial detection algorithms are able to detect whether an input is clean or perturbed by adversarial noise and therefore filter out the latter ones. However, some benchmarks on adversarial detection methods [145] showed that papers on detection algorithms may conflate their results and it can be showed that a detector with radius of detection ϵ is mathematically equivalent to a classifier of radius of robustness 2ϵ. Therefore, this work implies that efforts in adversarial detection may be vain or replaced by stronger algorithms. However, the paper also proves that detectors can be transformed into classifiers. However, their theorems only state the existence and how to construct such a classifier but isn't specific about the metrics to use or the actual sampling method to use.

Some state-of-the-art works in adversarial detection include multiLID [146], MetaAdvDet [147] and Maximum Mean Discrepancy (MMD)-based test [148].

The multiLID method [146] uses an approximation of the LID (Local Intrinsic Dimensionality) metric [149] [150]. The LID locally measures the properties of data distributions by assessing the growth rates of the cumulative distribution function within this neighborhood. The multiLID computes a vector of LID values, the $multiLID$ vector, instead of computing an aggregated value (such as the mean of this vector). Its i^{th} component is given in equation 4.6

where $d_i(x)$ is the euclidean distance between the representation by a carefully chosen neural network of a sample x and its i^th nearest-neighbor. The collection $d_i(x)_{i\in[1,k]}$ is sorted with $d_1 < ... < d_k$ where k is a hyperparameter.

$$multiLID(x)_i = -\log(\frac{d_i(x)}{d_k(x)}) \tag{4.6}$$

This vector is then fed into a random forest model to decide whether the input data x is benign or an adversarial example. A random forest is a method that combines multiple decision trees to improve the accuracy and robustness of the model.

The MetaAdvDet [147] is based on meta-learning and aims at adapting to new attacks based on experience accumulated with previous state-of-the-art attacks. It uses a network $\mathcal{T}$ that focuses on learning the new attack, considered as a task in the meta-learning framework, and a network $\mathcal{M}$, the master network, which learns a general strategy over all the tasks from the accumulated gradient in $\mathcal{T}$. The learning task relies on a dataset sampled from various attacks, each attack corresponding to a task of the meta-learning problem. The meta-learner experiences several scenarios thanks to the large amount of tasks/attacks allowing adaptation to new attacks rapidly. The training dataset is organized in tasks, each task being split into a support set to learn the basic capability of detecting old attacks and a query set to act as new attacks. The networks $\mathcal{T}$ and $\mathcal{M}$ output a probability that the input sample is benign or adversarial. The algorithm tests a sample similarly to a few-shot testing procedure [151] by fine-tuning the test model with few-shot examples.

The MMD (Maximum Mean Discrepancy) test [152], which is usually applied to detect distributional discrepancy between two datasets, uses a test statistic, the MMD. The equation of this metric is written in equation 4.7, where f is a function that can be expressed in the RKHS (Restricted Kernel Hilbert Space) $\mathcal{H}_k$ with kernel k, $\mathbb{P}$ and $\mathbb{Q}$ are the two distribution for which we wish to test the equality, X and Y are the random variables corresponding respectively to these distributions and $\mu_{\mathbb{P}} = \mathbb{E}(k(.,X))$ and $\mu_{\mathbb{Q}} = \mathbb{E}(k(.,Y))$ are kernel mean embeddings of $\mathbb{P}$ and $\mathbb{Q}$ respectively.

$$\begin{aligned} \text{MMD}(\mathbb{P},\mathbb{Q};\mathcal{H}_k) &= \sup_{f\in\mathcal{H}_k, \|f\|_{\mathcal{H}_k}\leq 1} |\mathbb{E}(f(X)) - \mathbb{E}(f(Y))| \\ &= \| \mu_{\mathbb{P}} - \mu_{\mathbb{Q}} \|_{\mathcal{H}_k} \end{aligned} \tag{4.7}$$

It was demonstrated that the MMD test is aware of differences between adversarial and benign samples in [148], despite previous claims showing that adversarial examples may not be easily detected [153]. The authors of [148] contradict the study in [153] by fixing three caveats to increase the power of the test :

1. The test power, which is the detection rate of adversarial attacks, of the MMD test is dependent on the form of the kernel. The Gaussian kernel looks at data uniformly instead of focusing on key areas. A deep, semantic-aware kernel would be more suitable by adding expressivity. The MMD test with this kernel is called SAMMD (semantic-aware MMD).
2. The test power is also very dependent on the optimization of the kernel bandwidth. This parameter of the deep kernel can be optimized by maximizing the approximate test power which is computed by asymptotic analysis of the SAMMD when detecting adversarial attacks.
3. Finally, the MMD test relies on the input data being independent and identically distributed (IID), which may not be the case for adversarial attacks. The wild bootstrap [154] is a technique used to resample the value of SAMMD with the optimized kernel in order to get correct p-values similarly to an IID scenario as the non-IID scenario makes the test meaningless. Adversarial data may show dependence when the adversary attacks the data used for training the model (such as in adversarial training) or when an adversary attacks an instance many times to generate several adversarial samples. The disadvantage of this method is that it cannot be deployed in a real-time setting but instead needs a dataset to be collected and run the test against sane data to assess whether the new dataset (and not individual points inside the dataset) is corrupted.

However, simply filtering out adversarial data samples may not be sufficient to make a model robust and autonomous once deployed. If the model "knows" what clean data look like, one may expect from it to restore adversarial samples to a clean state that would be useful for the classifier.

4.3.2 . Adversarial restoration

In recent years, some methods aiming to restore the original information from corrupt samples have emerged. Some of these methods use Langevin dynamics, introduced in section 2.3.6, to find a new sample from an original corrupted adversarial sample so that a downstream classifier would process the restored input correctly. From the Langevin dynamic described by the discretized stochastic differential equation in equation 4.8, a sample from the distribution of $p(x)$ can be found. In equation 4.8, the term x_t represents the value of the sample at iteration t, η is a step size in the gradient descent, z_t is a Gaussian noise with 0 mean and unit variance. Langevin sampling has some properties that make it useful in adversarial machine learning as the Markov chain yields a sample from the learned distribution asymptotically.

$$x_{t+1} = x_t + \frac{\eta}{2}\nabla_x \log p(x) + \sqrt{\eta} z_t \tag{4.8}$$

In [155], the authors introduce a way to reconstruct original samples from

an adversarial image with an energy-based model (EBM). Energy-based models [55] are models that rely on the optimization of an energy metric $E(x, y; \theta)$ where x is the feature vector of the input data, y is the label and θ the parameters of the model. The training of an EBM aims at minimizing the energy value associated to an input x and its corresponding label y. A long-run energy-based Langevin sampling, as performed in [155], creates unrealistic images which may make classification irrelevant. On the other hand, short-run Langevin sampling is not enough to converge to a good sample for classification. The authors in [155] therefore train the model with two different optimizers in two phases : first the Adam optimizer [16] which allows the model to sample realistic images on short-run Langevin dynamics, and then the SGD (Stochastic Gradient Descent) optimizer to update the parameters of the EBM so that long-run samples align with short-run ones. This effectively results in more realistic samples, making classification by the downstream classifier more reasonable in long-run Langevin sampling, although the results are qualitatively different from the initial short-run.

Another method relying on Langevin sampling to recover original samples from adversarial examples is introduced in [156]. A denoising autoencoder (DAE) [157] minimizes the reconstruction error on input data corrupted with Gaussian noise $\nu\,\mathcal{N}(0, \sigma^2 I)$ defined in equation 4.9 where p' denotes the empirical estimation of the target distribution p.

$$\mathbb{E}_{p'(x)p'(\nu)}(\parallel r(x+\nu) - r(x) \parallel) \tag{4.9}$$

It was proved in some earlier work [158] that, under the condition that $r(x) = x + o(1)$, then $r(x) - x = \sigma^2 \nabla_x \log p(x) + o(\sigma^2)$ when σ^2 converges to 0. Therefore, a DAE can be used to estimate the gradient of the log-likelihood of input data. Using this estimate, similarly to the previous paragraph, the authors use the MALA (Metropolis-Adjusted Langevin Algorithm) to sample from the distribution $p(x)$ with the same iteration described in equation 4.8. The authors choose to train a supervised denoising autoencoder (sDAE) [159] [160] by minimizing the loss defined in 4.10 made of a reconstruction error term and a cross-entropy term J.

$$\mathbb{E}_{p'(x,y)p'(\nu)}(\parallel r(x+\nu) - x \parallel^2 + 2\sigma^2 J(r(x+\nu), y)) \tag{4.10}$$

Then, the minimizer of equation 4.10 satisfies equation 4.11 where $\tilde{p}(y|x) = y^T \hat{y}(x)$ and $p(\tilde{x}, y) = \tilde{p}(y|x)p(x)$.

$$r(x) - x = \sigma^2 \mathbb{E}_{p(y|x)}(\nabla_x \log \tilde{p}(x, y)) + O(\sigma^3) \tag{4.11}$$

Finally, the authors add further contribution to the reconstruction of adversarial samples by altering the MALA reconstruction procedure defined by

equation 4.8 and introduce the MALADE procedure (MALA with sDAE for Defense). In order to circumvent issues arising with MALA which may drive the input to high-density regions that are not labeled correctly, the MALADE algorithm drives samples into high-density regions of the conditional training distribution $p(x|y)$ instead of the prior $p(x)$. Therefore, equation 4.8 is replaced by the dynamic defined by equation 4.12. An illustration of the MALADE algorithm is given in figure 4.3.

$$x_{t+1} = x_t + \eta \mathbb{E}_{p(y|x_t)}(\nabla_x \log p(x_t|y)) + \kappa \tag{4.12}$$

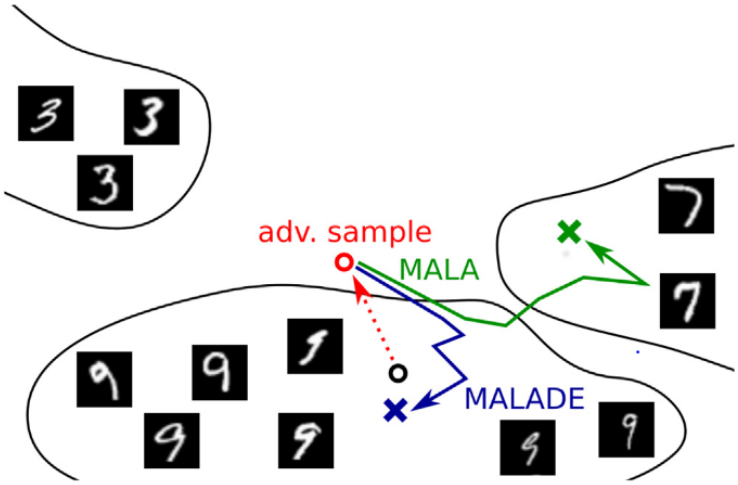


Figure 4.3 – Illustration of the MALADE algorithm from [156].

However, the intuition that the MALADE procedure being guided by the posterior $p(x|y)$ brings samples closer to the true class cluster may be erroneous. Indeed, this idea fails to take into account that the attacker may use a targeted attack which would waste any attempt to use the label information given by the classifier.

4.4 . Contributions

This section covers my contribution in adversarial defense, based on the state-of-the-art developed in the previous sections. My approach of adversarial defense relies on making a hybrid classifier, introduced in section 2.3.5, more robust to adversarial attacks. This system would detect incoming attacks, thanks to the detection metric introduced in chapter 3, the approximate mass. The generative part of the hybrid model would be biased towards assigning higher approximate mass values to clean samples by reusing the regularization term introduced in section 3. Once a sample has been identified as

adversarial, the model purifies it, similarly to approaches introduced in section 4.3. Once the sample is cleansed, it is fed again to the system in order to be classified.

The advantage of my method relies on the fact that there is no need to generate multiple adversarial samples during training and that it is attack agnostic. Indeed, in adversarial training, generalization over adversarial samples is done by training the model on adversarial samples generated through a particular attack. However, training on a weaker but faster attack such as FGSM [92] may improve the model's robustness to adversarial samples on FGSM attacks but PGD attacks may still be able to beat the model's accuracy [161]. Furthermore, the defense can be adapted to stronger attack radius without a need for retraining the model, as we will see in section 4.4.2. Finally, it corresponds to an implementation of the detector-classifier model introduced theoretically in the paper [145].

4.4.1 . Detecting adversarial samples

Adversarial samples detection can be seen similarly to the OOD detection framework developed in chapter 3. This idea relies on the assumption that clean and adversarial samples are not drawn from the same distribution. Indeed, the adversarial distribution is a distribution similar to the clean distribution, only differing by covariate shift (see chapter 3 for a definition).

First, the model is fed the data as input then computes the approximate mass of this sample. The value is then compared to a threshold computed on a validation set containing both clean and adversarial samples. If the approximate mass is smaller than the given threshold, the sample is considered clean and is directly classified. Otherwise, it is filtered out as adversarial and restored, as will be explained in section 4.4.2.

Algorithm 3 details the steps of the detection model, if it used alone without any classification step afterward. The goal here is a binary classification, similar to OOD detection where the model classifies input x_0 as adversarial or clean data, given a threshold τ computed on a validation set.

Algorithm 3 Adversarial detection with the approximate mass algorithm

if $\| \nabla_x \log p_\theta(x_0) \| < \tau$ **then**
  $REJECT = True$
else
  $REJECT = False$
end if

The advantage of the adversarial detection approach is that even when the input is cleansed and fed back into the model for classification, the model can still raise an alarm, independently of the classification result. That way, if

the input was successfully detected as adversarial, the model may be able to delegate its classification to a human supervisor or add some uncertainty to the prediction.

The threshold τ, as mentioned above, is calculated on a validation set, separate from the train and test sets as defined in section 2.2.2. This validation set is augmented with some adversarial data, making up the dataset of half of clean data and half of adversarial data. The threshold is computed as the one that yields the highest accuracy to separate clean data from adversarial data. It is computed following algorithm 4 with a validation dataset $\mathcal{V}$.

Algorithm 4 Compute Optimal Threshold

Input Validation set with true labels : $\{(x_i, y_i)\}$
Output Optimal threshold : τ

best_accuracy $\leftarrow 0$
best_threshold $\leftarrow 0$

for threshold $\leftarrow 0.55$ ... 1 **by** 0.01 **do** correct_predictions $\leftarrow 0$

for $i \leftarrow 1$ ... n **do** predicted_label $\leftarrow (\| \nabla_x \log p_\theta(x_0) \| < \tau)$

if predicted_label $=$ true_label$_i$ **then** correct_predictions $\leftarrow$ correct_predictions $+ 1$

end if accuracy $\leftarrow \frac{\text{correct\_predictions}}{n}$

if accuracy $>$ best_accuracy **then** best_accuracy $\leftarrow$ accuracy $\tau \leftarrow$ threshold

end if

end for
return τ

In order for the model to assign as low as possible an approximate mass to benign data samples, I add the regularization introduced in section 3.6. Similarly to what was done in OOD detection, I expect the approximate mass to have lower values on sane data samples that way and consistently assign a higher approximate mass on adversarial data. The training loss of the DIGLM model is defined in equation 4.13 where the hyperparameter α represents the trade-off between the maximum likelihood objective $min_\theta - \log p_\theta(x)$ and the new objective $min_\theta \nabla_x \log p_\theta(x)$.

$$L(x, \theta) = -\log p_\theta(y|x) + \beta[-\log p_\theta(x) + \alpha \nabla_x \log p_\theta(x)] \tag{4.13}$$

4.4.2 . Projecting reconstructed samples onto the neighborhood of the attacked sample

As introduced in section 4.3, adding a cleaning step after the detection step helps getting better adversarial classification results. Using a method such as the Langevin sampling process has the advantage of making the adversarial sample converge to the training distribution of the model when the length of the Markov chain increases, or at least what is modeled by the distribution p_θ computed by the normalizing flow in the hybrid model. Therefore, a cleaning technique like Langevin sampling acts more like a resampling process that aims at bringing adversarial samples closer to the original distribution p (or rather an estimation of it through p_θ).

However, the papers [162] and [155] show that although short run Langevin sampling may produce samples that are close to the original distribution, longer runs produce samples that do not make sense and that, paradoxically, do not look like the original distribution. A fix is to change the optimizer during training in order to yield samples that are close to the original distribution. During the first training epochs of the model, the optimizer is the Adam optimizer to then be changed to a simple SGD optimizer (regular stochastic gradient descent). Some other papers try to augment the power of the Langevin sampling, such as with the MALADE procedure [156] introduced in section 4.3. As a reminder, instead of sampling from the distribution $p(x)$, they sample from $p(x|y)$ supposedly in order not to avoid converging to the wrong class domain. However, this approach specifically relies on the evaluation of the label y during Langevin sampling, which already shifts the distribution. Furthermore, attacks may be targeted, which means that the attacker can assign any desired label to the data sample. That way, an attacker may want to give a label that is far away from the original label domain.

In my approach however, I take into account the prior that the attacker has a specified radius of attack when forging malicious inputs. This radius should be such that the perturbation would be imperceptible for a human. This is illustrated in figure 4.4. Knowing the maximum desirable perturbation is therefore a good piece of information to give the model in order for it to get closer to the original clean data sample. In order to do that, I add a projection step at the end of the Langevin sampling on the ball for radius ϵ_{spec} (the specified radius of attack). This projection step has two advantages :

1. bringing the Langevin sample closer to the original clean sample ;
2. avoiding the caveats reported in [162] and [155] where the Langevin sampling procedure yields outputs that do not look like the original distribution for long iterations.

In order to make the projection more effective, the final sample of the Langevin process is projected onto an L_∞-ball. Indeed, for a given radius ϵ_{spec},

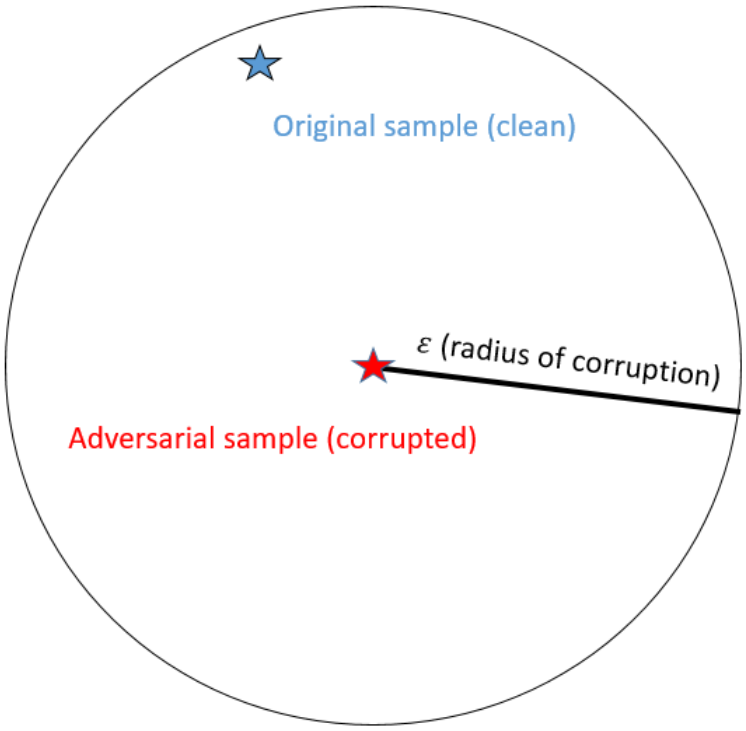


Figure 4.4 – Illustration of the information a defender may have against an adversary : the original sample is in a ball of radius ϵ centered around the adversarial sample.

the L_∞-ball with this radius centered around the adversarial sample x_{adv} has the largest volume out of all the L_p-balls, as illustrated in figure 4.5 [3].

Furthermore, the L_∞-norm is commonly used in a lot of attacks along with the L_2-norm to search for the optimal perturbation. Thus a defense that matches the norm used by the attacker, or at least an upper bound of it, is more likely to successfully locate the original sample given that prior information.

3. Credit to Quartl, CC BY-SA 3.0 <https ://creativecommons.org/licenses/by-sa/3.0>, via Wikimedia Commons.

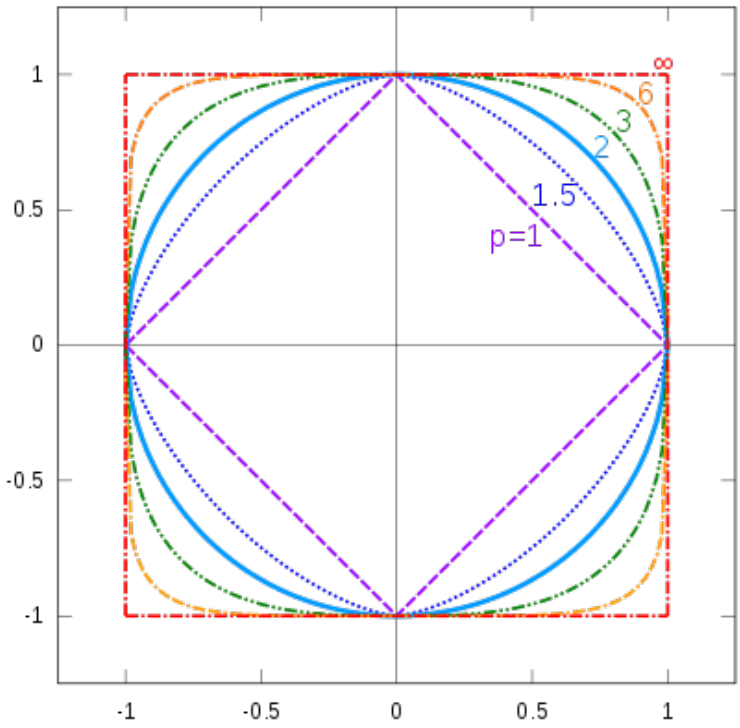


Figure 4.5 – Illustration of different L_p-balls and the L_∞-ball in 2 dimensions. The x-axis represents the first coordinate of a vector in $\mathbb{R}^2$, the y-axis the second coordinate. The L_∞-ball surrounds every other ball therefore if an attacker perturbs the input relatively to this norm, the perturbation may be outside of every other L_p-balls.

4.5 . Experimental methodology

Testing defenses in adversarial machine learning is also a complex task similarly to the evaluation of OOD detection models. In this section, I will introduce the methodology I use in order to evaluate my model. I will cover here the choice of datasets, attacks and libraries, and metrics.

4.5.1 . Choice of datasets

Datasets used in adversarial machine learning are common across benchmarks. In section 4.6, the dataset I will be focusing on in order to evaluate my approach will be the MNIST dataset [2]. Further experimentations on other datasets will be conducted but the MNIST dataset allows to provide a proof of concept to demonstrate how the method fares in different settings.

These datasets show the advantage of covering different semantics (labels) and different types of images (gray scale and color images).

I decide to split the test set into a validation and a new test set, where the

validation set is made of 20% of the total test set, randomly sampled. The rest of the test set is then shortened to 1000 samples in order to accelerate testing.

4.5.2 . Evaluation metrics

The metrics used in the experiments differ between the given task. In detection, I chose the same metrics as the ones introduced in chapter 3, section 3.4. These metrics are the AUROC, the AUPR, the TNR@95%TPR (True Negative Rate at 95% True Positive Rate). They are further described in chapter 3.

However, as opposed to the previous chapter, I also add the accuracy of the model in the detection task as it will be important in order to later evaluate the full defense (detection, cleansing, classification). This accuracy is computed as the fraction of adversarial samples correctly classified as corrupted. Further explanations are given in section 4.6.1. In essence, the model first computes a threshold of approximate mass on a validation set, then classifies as adversarial any data whose approximate mass is above this threshold and classifies as clean any data with a lower approximate mass than the threshold.

4.5.3 . Python frameworks

As mentioned in section 4.2, the choice of library that implemented adversarial attacks settled on the AutoAttack library. However, it is important to note that many other Python libraries, compatible with the Pytorch library, were created in the recent years to make implementation of adversarial attacks easier. Examples include Torchattacks [163] [4], cleverhans [164] [5], and ART (Adversarial Robustness Toolbox) [165] [6] which implement widely used attacks in the domain.

The attacks implemented in several of these adversarial evaluation libraries include the PGD attack [141] [142] and the Carlini-Wagner (CW) attack [166]. Indeed, both attacks are known for being robust against gradient obfuscation methods [167]. These methods are defense mechanisms that prevent the computation of adversarial attacks by making gradients in the model hard or impossible to compute through the use of randomness or non-differentiable operators (such as rounding operators). A wider diversity of attacks, including black box attacks which are insensitive to gradient obfuscation, allows the library to make fairer comparisons.

The choice of hyperparameters may also be detrimental to a fair evaluation. For example, the number of steps in the PGD attack [141] is an important hyperparameter but some papers may set it to a value that is too low to actually produce relevant samples for the evaluation. This is why the AutoAttack library sets to a default value most hyperparameters, apart from the radius of

4. https ://github.com/Harry24k/adversarial-attacks-pytorch
5. https ://github.com/cleverhans-lab/cleverhans
6. https ://github.com/Trusted-AI/adversarial-robustness-toolbox

attack ϵ, and even implements attacks that require as few hyperaparameters as possible (such as the APGD attacks [138], described in section 4.2).

Furthermore, the AutoAttack library is associated with the RobustBench [168] [7] benchmark which is a website registering defenses evaluated on the AutoAttack library. It references the results of these defenses with respect to the dataset used for the benchmark (CIFAR-10 [80], CIFAR-100 [80], MNIST [2]...), the distance to measure perturbation radius (L_∞ or L_2 distance), and the radius of attack ϵ.

4.6 . Experiments

In this section, I report experimental results on 3 experiments to test out the capacity of the defense method in detection, classification and with the full defense (detection + classification).

These evaluations are performed for DIGLM models (described in chapter 2) trained with an approximate mass penalization ($\alpha = 2$ penalization factor in the loss defined in equation 4.13, to take a similar value as what work in chapter 3), and on a simple model trained with maximum likelihood ($\alpha = 0$, referred to as the "naive" model in this section). The β value in equation 4.13 is set to 0.1 in order to give more importance to the classification accuracy of the model so that the clean accuracy [8] is as high as possible.

Adversarial attacks are generated with the APGD-CE attack, described in section 4.2, with a radius of attack of $\epsilon = \frac{8}{255} \sim 0.032$ which corresponds to the highest radius of perturbation on this dataset in benchmarks.

4.6.1 . Experiment : results in adversarial detection

The first experiment aims at demonstrating the power of the detection module of my defense mechanism. In order to do that, the detection is evaluated similarly to what was done in section 3.6.6. The test dataset is composed of half of clean data and half of adversarially generated data, with one attack mentioned in the column "Attack". The test is ran for all attacks in the AutoAttack library. The metrics used are the AUROC, the AUPR as well as the TNR at 95% TPR (refer to section 3.6.6 for further details on these metrics). The accuracy is also added as a thresholded metric, where the threshold of classification τ is computed as explained in section 4.5 on a validation set composed of a set of a subset of the test set along with the same samples attacked.

The results can be found in table 4.1. One may observe that on the MNIST dataset, the detection of adversarial attacks with a penalized model is on par with what we observed in OOD detection. The non-thresholded metrics (AU-

7. https ://robustbench.github.io/
8. Accuracy on clean samples.

ROC, AUPR, TNR@95%TPR) show that the model clearly detects adversarial samples. Furthermore, the accuracy indicates that the threshold computed on the validation set by the model is good enough to detect the samples with high accuracy. On the other hand, it can be deduced from the AUROC value that the naive model seems to invert the labels of the adversarial and the clean, which also explains the low TNR value. The accuracy of 50% shows that the model didn't manage to compute a threshold that would classify data correctly because of the inverted labels assigned by the model on the validation set.

α	Dataset	Attack	AUROC	AUPR	TNR@95%TPR	Accuracy
2	MNIST	APGD-CE	1.0	1.0	1.0	0.922
0	MNIST	APGD-CE	0.056	0.318	0	0.5

Table 4.1 – Results on detection of adversarial attacks for the penalized version ($\alpha = 2$) and the naive version ($\alpha = 0$).

4.6.2 . Experiment : results in classification

The second experiment is concerned with the classification accuracy of the classifier when it is given an adversarial input. The goal of this experiment is to clean an adversarial sample and measure the accuracy of the module once the cleaning step is done. In order to do that, the experiment is conducted by building a test set of adversarial samples. The model cleans them with the Langevin sampling procedure then feeds it back into the classifier to assign a label to it. The length of the chain is of 10^6 steps (a long chain is necessary to converge to a good sample), the step size is of 10^{-3}. The Langevin dynamic may however not need to be run for such a long time for every samples. Therefore, it can be stopped sooner with a stopping criterion, by calculating a reference approximate mass such that the Langevin sampling provides the highest accuracy on a given validation set, built similarly to the detection experiment in section 4.6.1.

The reported metric is the accuracy in table 4.2. Note that the accuracy on the adversarial sample (before restoration) is very close to 0%. The results indicate here that the Langevin restoration method with projection yields excellent results on the penalized model and manages to increase the accuracy for either model, although the increase is much more significant on the penalized model. We can deduce here that the Langevin sampling procedure can be helped by the penalization to provide a smoother likelihood landscape locally around adversarial data which may make the Gradient ascent less susceptible from being stuck in high-loss regions.

α	Dataset	Attack	Accuracy
2	MNIST	APGD-CE	0.481
0	MNIST	APGD-CE	0.107

Table 4.2 – Results on classification of adversarial samples after Langevin cleaning for the penalized version ($\alpha = 2$) and the naive version ($\alpha = 0$).

4.6.3 . Experiment : results for the full defense

In this experiment, the performance of the overall defense is measured. The model is given a test set of half clean samples and another half of adversarial samples. Each sample is first filtered by the detection module then if the sample is judged as adversarial by the detection module it is cleansed and fed back again for classification.

Results of the experiment can be found in table 4.3. The results show that the model with approximate mass penalization benefits both from a better detection rate (from the detection experiment) and a better accuracy after recovery on adversarial samples. This suggests that the full defense benefits from both the good results on detection and those on Langevin restoration.

α	Dataset	Attack	Accuracy
2	MNIST	APGD-CE	0.608
0	MNIST	APGD-CE	0.329

Table 4.3 – Results on detection and classification of adversarial attacks and clean samples after detection and cleansing for the penalized version ($\alpha = 2$) and the naive version ($\alpha = 0$).

4.6.4 . Experiment : results of the full defense with an adaptive attack

Finally, this experiment aims at testing the defense on an adaptive adversarial attacks, following the recommendations in [169]. An adaptive attack is an attack that is specifically aimed at breaking a given defense method. In order to build a suitable adaptive attack for this defense method, an approximate mass term is included in the loss so that the objective of the attacker is at the same time to fool the model while minimizing the approximate mass of the sample in order to make it undetectable.

To realize this attack, the APGD-CE attack from the AutoAttack library is modified to include the approximate mass of the model to the loss.

The results of the experiment can be found in table 4.4. The accuracy of both models here benefit from a huge boost compared to the previous experiment. This might be evidence that the objective of finding a sample that

is hard to detect as adversarial is contradictory to the objective of finding a sample that is hard to find, thus producing bad quality adversarial samples.

α	Dataset	Attack	Accuracy
2	MNIST	APGD adaptive	0.619
0	MNIST	APGD adaptive	0.331

Table 4.4 – Results on on the full defense against an adaptive attack for the penalized version ($\alpha = 2$) and the naive version ($\alpha = 0$).

4.7 . Conclusion of this chapter

This chapter introduced two uses of the score $\nabla_x \log p(x)$ in adversarial defense. The contributions of this section are as follow :

1. I confirmed the sensitivity of the approximate mass to adversarial attacks and therefore to covariate shifts.
2. Results in adversarial detection with the approximate mass show that the metric is particularly suitable for detecting adversarial samples.
3. I also demonstrated the use of Langevin sampling methods in adversarial sample restoration with the addition of a projection step.
4. I found that producing hard to detect adversarial samples may be contradictory with the objective of finding a good adversarial samples that fools the classifier.

The rationale behind the use of the approximate mass in adversarial detection is that, from the results in chapter 3, the approximate mass being sensitive to OOD samples (covariate and semantic shifts) but not as much to semantic shifts, the model might perform well on covariate shifts.

On the other hand, the use of Langevin sampling to restore corrupted data is not new and has been explored in many different ways in the scientific literature, not only in adversarial defense but also in denoising diffusion models [59] for example. The advantage of Langevin sampling is that it samples data that asymptotically belongs to the distribution used for the sampling. However, the estimation $\log p_\theta(x)$ of the log-likelihood $\log p(x)$ is flawed. Therefore, sampling from p_θ would not guarantee having a sample from p. Nonetheless, the classifier is only familiar with training data, that is to say the very same data used to estimate p_θ. In other words, it is sufficient to sample from the estimate p_θ in order to have a satisfying sample for the classifier. My contribution came by noticing that most methods using Langevin sampling for image reconstruction suffer from a problem that the sample is not realistic or close

to the original sample. Some fixes have been introduced but none of them uses the *a priori* information that the corrupted sample is in the vicinity of the original sample, or at least a sample that would be handled correctly by the classifier.

Future research based on these ideas may study the implementation of a Langevin sampling procedure in the latent space. Indeed, the algorithm may be simplified by avoiding to compute many forward passes to compute the log-likelihood of input data and backpropagation passes to compute the score $\nabla_x \log p_\theta(x)$. Instead, performing the Langevin sampling in the latent space, by mapping the input once to the latent space and then computing only the gradient of the log-likelihood in the latent space might drastically reduce computational costs. Indeed, the latent space is, in general, parameterized by a normal distribution, therefore the gradient of the log-likelihood (the score) is easily tractable as an analytic form can be derived for the score, therefore sparing computations with a backpropagation algorithm.

More experiments will be conducted on other datasets, specifically the CIFAR-10 dataset [80] and the CIFAR-100 dataset [80] to cover other data distributions (different labels spaces and features).

Further analysis will also be performed on the integration of the defense technique introduced here with adversarial training to see how the defense described in this chapter further fits with other defense techniques as one would intuitively think that the combination of several defenses benefits the model even more. The idea would be to train the model with the approximate mass penalization as well as adversarial training where the model is fed adversarial data. Also, a comparison between the model and other defense models will be useful, especially in detection, as the experiments shown here can be seen as proof of concept of the defense method.

5 - Conclusion and future perspectives

The problem of the robustness of neural networks can be defined in several ways. It is mostly dependent on the type of distribution shift on the input data we aim to characterize. The problem of robustness in classifiers can be assimilated to error detection and error correction in coding theory. These tasks amount to finding errors in a code sent in a noisy canal and then restore the original code from the deteriorated one. In this manuscript, we covered two subfields of AI that relate to these issues, namely OOD detection and adversarial machine learning. In the case of OOD detection, the problem was assimilated in this thesis to error detection while adversarial machine learning was tackled both from the point of view of error detection and correction. Both tasks are crucial in order to deploy models in an autonomous way.

In this thesis, I studied the use of generative models, specifically normalizing flows, because of their flexibility as they can be used in an unsupervised and a supervised context, to solve these problems of robustness. Intuitively, normalizing flows were a good choice as their tractable log-likelihood would make a good measure for detecting and restoring data. However, we saw that, because of entropic issues associated with the likelihood, this metric might not be adapted in every cases. Instead, I decided to cover the tasks described above with a new metric, called the approximate mass, argued to present better behavior in OOD detection. However, my experiments showed that the approximate mass behaves correctly at the beginning of training but ends up overfitting by the end of training on tractable likelihood models. This issue means that OOD detection cannot be performed the same way as for another model like an energy-based model. In order to fix this issue, I changed the training of normalizing flows by adding to their training objective a term proportional to the approximate mass. This training objective successfully makes the model adopt a consistent behavior during training and avoids this overfitting. Moreover, the results obtained in my benchmarks show that the approximate mass is well adapted for OOD detection, even beating state-of-the-art models on standard benchmarks. I tested my approach with a new methodological approach that took into account the balance in data classes, the metrics to measure the quality of OOD detection as well as the number of parameters of the models I compared my model to. This level of detail is rarely given attention in previous paper in the domain. I further showed that this metric had some limitations on other typed of distributional shifts than the types encountered in OOD detection. More specifically, the approximate mass might be adapted to mostly detect covariate shifts instead of semantic shifts.

In the context of adversarial machine learning, I decided to use a DIGLM model as a classifier, which is an architecture that relies on a normalizing flow as a feature extractor. The idea this time was to integrate both error detection and correction in one model. Therefore, I decided to explore the use of the approximate mass in the detection of adversarial attacks and their restoration. This part also aimed at testing the hypothesis that the approximate mass might be mostly sensitive to covariate shifts. My experiments showed that the approximate mass allows the detection of such samples. Furthermore, the use of the approximate mass can be extended to restore adversarial data through Langevin sampling. Some a priori information can also be added to the Langevin sampling procedure by adding a projection step which gives a better aspect to the restored images than with naive Langevin sampling.

These results show that the approximate mass is a versatile metric as it allows for the detection and the restoration of data. The score, which is the gradient of the log-likelihood $\nabla_x \log p(x)$, is a quantity that may embed a lot of information about the data distribution. It may be related to other information theoretic and information geometric quantities such as the Fisher information [1].

Future works on the double backpropagation loss introduced in chapter 3 may improve computational efficiency by drawing ideas from [113]. The approximate mass as a regularization term may also be extended to supervised learning where the log-likelihood is replaced by the log-posterior $\log p(y|x)$ of a DIGLM [1] model. Some theoretical improvements may be found in analyzing the approximate mass with information geometry, measure theory by assimilating the approximate mass to a Wasserstein distance or even differential equations.

Future research based on the approach introduced in chapter 4 may explore the Langevin sampling procedure in the latent space to simplify many forward and backward passes. Instead, performing the Langevin sampling in the latent space, by mapping the input once to the latent space and then computing only the gradient of the log-likelihood in the latent space might drastically reduce computational costs. Indeed, the latent space is, in general, parameterized by a normal distribution, therefore the gradient of the log-likelihood (the score) is easily tractable as an analytic form can be derived for the score, therefore sparing computations with a backpropagation algorithm. Some other experiments will be performed to assess the improvements added by the method introduced in 4, such as an ablation study on the addition of the projection step after the Langevin sampling, the integration of the method with adversarial training. Finally, some method to distinguish between OOD data and adversarial data with the approximate mass should be studied

1. Defined by the gradient $\nabla_\theta \log p_\theta(x)$. It measures the amount of information that an input (modeled by a random variable) carries about the parameter θ.

in order for the model to be completely autonomous.

The works in chapter 4 can be extended to the idea of memory-augmented models introduced in section 2.4. The Langevin restoration technique introduced in chapter 4 can be seen as an associative memory process here the model uses a probabilistic model to find a similar pattern to the input. The approximate mass detection on the other hand is an error detection mechanism. Thus the contributions covered in chapters 3 and 4 perform a robust encoding of a data distribution (instead of individual patterns) to implement a robust error detection and correction system.

Furthermore, performing memory queries directly in the latent space of a model may increase the capacity of such models and add some desirable properties such as associativity which may help bridge the gap between symbolic AI and deep learning. This would allow designing neural models in a more interpretable way, for example by adding compositionality to models, where the symbols manipulated by the model. Moving the external memory module to the "computing unit" (the equivalent of the neural controller) is inspired by current trends in embedded AI where some embedded systems now move computations directly inside the memory (in-memory computing) instead of relying on the traditional Von Neumann architecture.

Finally, modern debates around the opposition between energy-based models and contrastive learning approaches [2] seem to oppose both approaches. Here, the use of likelihood-based models may help bridging the gap between both methods as the likelihood can be assimilated to an energy measure with the relation $E = -\log p(x) - \log Z$. Furthermore, the likelihood and the approximate mass are natural metrics for measuring the similarity between two objects.

2. A training method where two modalities (e.g.. : an image and a text associated to the image) are used to train a latent space. If both modalities are related (positive pair), their latent representation are brought closer in the representation space. If they are not related (negative pair), their distance is maximized.

Bibliographie

[1] Eric Nalisnick, Akihiro Matsukawa, Yee Whye Teh, Dilan Gorur, and Balaji Lakshminarayanan. Hybrid Models with Deep and Invertible Features. *arXiv :1902.02767 [cs, stat]*, May 2019.

[2] Li Deng. The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]. *IEEE Signal Processing Magazine*, 29(6) :141–142, November 2012.

[3] Sami Tibshirani and Harry Friedman. Valerie and Patrick Hastie.

[4] Lance Parsons, Ehtesham Haque, and Huan Liu. Subspace clustering for high dimensional data : A review. *ACM SIGKDD Explorations Newsletter*, 6(1) :90–105, June 2004.

[5] G. Palm. On associative memory. *Biological Cybernetics*, 36(1) :19–31, February 1980.

[6] Gokmen Zararsiz, Ferhan Elmali, and Ahmet Ozturk. Bagging Support Vector Machines for Leukemia Classification. 9(6), 2012.

[7] Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine learning practice and the bias-variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32) :15849–15854, August 2019.

[8] A. M. TURING. I.—COMPUTING MACHINERY AND INTELLIGENCE. *Mind*, LIX(236) :433–460, October 1950.

[9] Yoshihiro Maruyama. Symbolic and Statistical Theories of Cognition : Towards Integrated Artificial Intelligence. In Loek Cleophas and Mieke Massink, editors, *Software Engineering and Formal Methods. SEFM 2020 Collocated Workshops*, volume 12524, pages 129–146. Springer International Publishing, Cham, 2021.

[10] A. Newell and H. Simon. The logic theory machine–A complex information processing system. *IRE Transactions on Information Theory*, 2(3) :61–79, September 1956.

[11] Brian P. McLaughlin. Computationalism, Connectionism, and the Philosophy of Mind. In Luciano Floridi, editor, *The Blackwell Guide to the Philosophy of Computing and Information*, pages 135–151. Wiley, 1 edition, January 2004.

[12] Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric Deep Learning : Grids, Groups, Graphs, Geodesics, and Gauges, May 2021.

[13] Henry W. Lin, Max Tegmark, and David Rolnick. Why does deep and cheap learning work so well? *Journal of Statistical Physics*, 168(6) :1223–1247, September 2017.

[14] Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. Why do tree-based models still outperform deep learning on typical tabular data?

[15] Dominic Masters and Carlo Luschi. Revisiting Small Batch Training for Deep Neural Networks, April 2018.

[16] Diederik P. Kingma and Jimmy Ba. Adam : A Method for Stochastic Optimization, January 2017.

[17] Ilya Loshchilov and Frank Hutter. SGDR : Stochastic Gradient Descent with Warm Restarts, May 2017.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving Deep into Rectifiers : Surpassing Human-Level Performance on ImageNet Classification. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034, Santiago, Chile, December 2015. IEEE.

[19] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, March 2010.

[20] Seppo Linnainmaa. Taylor expansion of the accumulated rounding error. *BIT*, 16(2) :146–160, June 1976.

[21] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088) :533–536, October 1986.

[22] Yann A. LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Efficient BackProp. In Grégoire Montavon, Geneviève B. Orr, and Klaus-Robert Müller, editors, *Neural Networks : Tricks of the Trade : Second Edition*, Lecture Notes in Computer Science, pages 9–48. Springer, Berlin, Heidelberg, 2012.

[23] Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic Differentiation in Machine Learning : A Survey. *Journal of Machine Learning Research*, 18(153) :1–43, 2018.

[24] Kunihiko Fukushima. Cognitron : A self-organizing multilayered neural network. *Biological Cybernetics*, 20(3) :121–136, September 1975.

[25] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International*

Conference on International Conference on Machine Learning, ICML'10, pages 807–814, Madison, WI, USA, June 2010. Omnipress.

[26] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. Dive into Deep Learning, February 2023.

[27] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier Nonlinearities Improve Neural Network Acoustic Models.

[28] Marco Gori, Alessandro Betti, and Stefano Melacci. Chapter 2 - Learning principles. In Marco Gori, Alessandro Betti, and Stefano Melacci, editors, *Machine Learning (Second Edition)*, pages 53–111. Morgan Kaufmann, January 2024.

[29] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5) :359–366, January 1989.

[30] Balázs Csáji. *Approximation with Artificial Neural Networks*. PhD thesis, June 2001.

[31] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[32] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet : A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, June 2009.

[33] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep Sparse Rectifier Neural Networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 315–323. JMLR Workshop and Conference Proceedings, June 2011.

[34] Sergey Ioffe and Christian Szegedy. Batch Normalization : Accelerating Deep Network Training by Reducing Internal Covariate Shift, March 2015.

[35] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Training Very Deep Networks, November 2015.

[36] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[37] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[38] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An Image is Worth 16x16 Words : Transformers for Image Recognition at Scale. In *International Conference on Learning Representations*, October 2020.

[39] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate, May 2016.

[40] Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes, May 2014.

[41] Nathalie Japkowicz, Stephen José Hanson, and Mark A. Gluck. Nonlinear Autoassociation Is Not Equivalent to PCA. *Neural Computation*, 12(3) :531–545, March 2000.

[42] Esteban G. Tabak and Eric Vanden-Eijnden. Density estimation by dual ascent of the log-likelihood. *Communications in Mathematical Sciences*, 8(1) :217–233, 2010.

[43] Danilo Jimenez Rezende and Shakir Mohamed. Variational Inference with Normalizing Flows.

[44] George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing Flows for Probabilistic Modeling and Inference. *arXiv :1912.02762 [cs, stat]*, April 2021.

[45] Keegan Kelly, Lorena Piedras, Sukrit Rao, and David Roth. Variations and Relaxations of Normalizing Flows, September 2023.

[46] Bálint Máté, Samuel Klein, Tobias Golling, and François Fleuret. Flowification : Everything is a normalizing flow. In *Advances in Neural Information Processing Systems*, May 2022.

[47] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using Real NVP. *arXiv :1605.08803 [cs, stat]*, February 2017.

[48] Polina Kirichenko, Pavel Izmailov, and Andrew Gordon Wilson. Why Normalizing Flows Fail to Detect Out-of-Distribution Data. *arXiv :2006.08545 [cs, stat]*, June 2020.

[49] Diederik P. Kingma and Prafulla Dhariwal. Glow : Generative Flow with Invertible 1x1 Convolutions. *arXiv :1807.03039 [cs, stat]*, July 2018.

[50] Will Grathwohl, Ricky T Q Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. FFJORD : FREE-FORM CONTINUOUS DYNAMICS FOR SCALABLE REVERSIBLE GENERATIVE MODELS. 2019.

[51] J A Nelder and R W M Wedderburn. Generalized Linear Models. 2022.

[52] Yang Song and Stefano Ermon. Generative Modeling by Estimating Gradients of the Data Distribution. *arXiv :1907.05600 [cs, stat]*, October 2020.

[53] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. SCORE-BASED GENERATIVE MODELING THROUGH STOCHASTIC DIFFERENTIAL EQUATIONS. 2021.

[54] Albert Einstein - On the Movement of Small Particles Suspended in. https ://einstein.academicwebsite.com/publications/7-on-the-movement-of-small-particles-suspended-in-stationary-liquids-required-by-the-molecular-kinetic-theory-of-heat.

[55] Yee Whye Teh, Max Welling, Simon Osindero, and Geoffrey E. Hinton. Energy-Based Models for Sparse Overcomplete Representations. *Journal of Machine Learning Research*, 4(Dec) :1235–1260, 2003.

[56] Paul Smolensky. Information Processing in Dynamical Systems : Foundations of Harmony Theory.

[57] G. E. Hinton and R. R. Salakhutdinov. Reducing the Dimensionality of Data with Neural Networks. *Science*, 313(5786) :504–507, July 2006.

[58] Jascha Sohl-Dickstein, Eric A Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep Unsupervised Learning using Nonequilibrium Thermodynamics.

[59] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising Diffusion Probabilistic Models. In *Advances in Neural Information Processing Systems*, volume 33, pages 6840–6851. Curran Associates, Inc., 2020.

[60] Calvin Luo. Understanding Diffusion Models : A Unified Perspective, August 2022.

[61] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

[62] Farzan Farnia and Asuman Ozdaglar. Do GANs always have Nash equilibria ? In *Proceedings of the 37th International Conference on Machine Learning*, pages 3029–3039. PMLR, November 2020.

[63] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen, and Xi Chen. Improved Techniques for Training GANs. In *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[64] Pentti Kanerva. Hyperdimensional Computing : An Introduction to Computing in Distributed Representation with High-Dimensional Random Vectors. *Cognitive Computation*, 1(2) :139–159, June 2009.

[65] Denis Kleyko, Dmitri Rachkovskij, Evgeny Osipov, and Abbas Rahimi. A Survey on Hyperdimensional Computing aka Vector Symbolic Architectures, Part I : Models and Data Transformations. *ACM Computing Surveys*, May 2022.

[66] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space, September 2013.

[67] Tony F. Wu, Haitong Li, Ping-Chen Huang, Abbas Rahimi, Gage Hills, Bryce Hodson, William Hwang, Jan M. Rabaey, H.-S. Philip Wong, Max M. Shulaker, and Subhasish Mitra. Hyperdimensional Computing Exploiting Carbon Nanotube FETs, Resistive RAM, and Their Monolithic 3D Integration. *IEEE Journal of Solid-State Circuits*, 53(11) :3183–3196, November 2018.

[68] Sparse Distributed Memory. https ://mitpress.mit.edu/9780262514699/sparse-distributed-memory/.

[69] J J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8) :2554–2558, April 1982.

[70] R. G. Morris. D.O. Hebb : The Organization of Behavior, Wiley : New York ; 1949. *Brain Research Bulletin*, 50(5-6) :437, 1999.

[71] Hubert Ramsauer, Bernhard Schäfl, Johannes Lehner, Philipp Seidl, Michael Widrich, Thomas Adler, Lukas Gruber, Markus Holzleitner, Milena Pavlović, Geir Kjetil Sandve, Victor Greiff, David Kreil, Michael Kopp, Günter Klambauer, Johannes Brandstetter, and Sepp Hochreiter. Hopfield Networks is All You Need, April 2021.

[72] Dmitry Krotov and John J. Hopfield. Dense Associative Memory for Pattern Recognition. In *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[73] On a Model of Associative Memory with Huge Storage Capacity | SpringerLink. https ://link.springer.com/article/10.1007/s10955-017-1806-y.

[74] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural Turing Machines, December 2014.

[75] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-term Memory. *Neural computation*, 9 :1735–80, December 1997.

[76] Yan Wu, Greg Wayne, Alex Graves, and Timothy Lillicrap. THE KANERVA MACHINE : A GENERATIVE DISTRIBUTED MEMORY. 2018.

[77] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou,

Adrià Puigdomènech Badia, Karl Moritz Hermann, Yori Zwols, Georg Ostrovski, Adam Cain, Helen King, Christopher Summerfield, Phil Blunsom, Koray Kavukcuoglu, and Demis Hassabis. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626) :471–476, October 2016.

[78] Timothée Darcet, Maxime Oquab, Julien Mairal, and Piotr Bojanowski. Vision Transformers Need Registers, September 2023.

[79] Jingkang Yang, Kaiyang Zhou, Yixuan Li, and Ziwei Liu. Generalized Out-of-Distribution Detection : A Survey. *arXiv :2110.11334 [cs]*, October 2021.

[80] Alex Krizhevsky. Learning Multiple Layers of Features from Tiny Images. page 60.

[81] Jakob Gawlikowski, Cedrique Rovile Njieutcheu Tassi, Mohsin Ali, Jongseok Lee, Matthias Humt, Jianxiang Feng, Anna Kruspe, Rudolph Triebel, Peter Jung, Ribana Roscher, Muhammad Shahzad, Wen Yang, Richard Bamler, and Xiao Xiang Zhu. A Survey of Uncertainty in Deep Neural Networks. *arXiv :2107.03342 [cs, stat]*, January 2022.

[82] Dan Hendrycks and Kevin Gimpel. A Baseline for Detecting Misclassified and Out-of-Distribution Examples in Neural Networks. In *International Conference on Learning Representations*, November 2016.

[83] Joan Bruna, Christian Szegedy, Ilya Sutskever, Ian Goodfellow, Wojciech Zaremba, Rob Fergus, and Dumitru Erhan. Intriguing properties of neural networks. December 2013.

[84] Anh Nguyen, Jason Yosinski, and Jeff Clune. 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, Boston, MA, USA, June 2015. IEEE.

[85] Will Grathwohl, Kuan-Chieh Wang, Jörn-Henrik Jacobsen, David Duvenaud, Mohammad Norouzi, and Kevin Swersky. Your Classifier is Secretly an Energy Based Model and You Should Treat it Like One. *arXiv :1912.03263 [cs, stat]*, September 2020.

[86] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On Calibration of Modern Neural Networks. In *Proceedings of the 34th International Conference on Machine Learning*, pages 1321–1330. PMLR, July 2017.

[87] Yufeng Zhang, Jialu Pan, Wanwei Liu, Zhenbang Chen, Ji Wang, Zhiming Liu, Kenli Li, and Hongmei Wei. Kullback-Leibler Divergence-Based Out-of-Distribution Detection with Flow-Based Generative Models, March 2023.

[88] Xixi Liu, Yaroslava Lochman, and Christopher Zach. GEN : Pushing the Limits of Softmax-Based Out-of-Distribution Detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 23946–23955, 2023.

[89] Eric Nalisnick, Akihiro Matsukawa, Yee Whye Teh, Dilan Gorur, and Balaji Lakshminarayanan. Do Deep Generative Models Know What They Don't Know? *arXiv :1810.09136 [cs, stat]*, February 2019.

[90] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Ng. Reading Digits in Natural Images with Unsupervised Feature Learning. *NIPS*, January 2011.

[91] Shiyu Liang, Yixuan Li, and R. Srikant. Enhancing The Reliability of Out-of-distribution Image Detection in Neural Networks. *arXiv :1706.02690 [cs, stat]*, August 2020.

[92] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and Harnessing Adversarial Examples, March 2015.

[93] Yen-Chang Hsu, Yilin Shen, Hongxia Jin, and Zsolt Kira. Generalized ODIN : Detecting Out-of-Distribution Image Without Learning From Out-of-Distribution Data. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10948–10957, Seattle, WA, USA, June 2020. IEEE.

[94] Shiyu Liang, R Srikant, and Yixuan Li. ENHANCING THE RELIABILITY OF OUT-OF-DISTRIBUTION IMAGE DETECTION IN NEURAL NETWORKS. 2018.

[95] Kimin Lee, Kibok Lee, Honglak Lee, and Jinwoo Shin. A Simple Unified Framework for Detecting Out-of-Distribution Samples and Adversarial Attacks. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[96] Reuben Feinman, Ryan R. Curtin, Saurabh Shintre, and Andrew B. Gardner. Detecting Adversarial Samples from Artifacts, November 2017.

[97] Xingjun Ma, Bo Li, Yisen Wang, Sarah M. Erfani, Sudanthi Wijewickrema, Grant Schoenebeck, Dawn Song, Michael E. Houle, and James Bailey. Characterizing Adversarial Subspaces Using Local Intrinsic Dimensionality, March 2018.

[98] Weitang Liu, Xiaoyun Wang, John Owens, and Yixuan Li. Energy-based Out-of-distribution Detection. In *Advances in Neural Information Processing Systems*, volume 33, pages 21464–21475. Curran Associates, Inc., 2020.

[99] Rui Huang, Andrew Geng, and Yixuan Li. On the Importance of Gradients for Detecting Distributional Shifts in the Wild. In *Advances in*

Neural Information Processing Systems, volume 34, pages 677–689. Curran Associates, Inc., 2021.

[100] Gukyeong Kwon, Mohit Prabhushankar, Dogancan Temel, and Ghassan AlRegib. Backpropagated Gradient Representations for Anomaly Detection. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision – ECCV 2020*, volume 12366, pages 206–226. Springer International Publishing, Cham, 2020.

[101] Pramuditha Perera, Ramesh Nallapati, and Bing Xiang. OCGAN : One-Class Novelty Detection Using GANs With Constrained Latent Representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2898–2906, 2019.

[102] Andrew P. Bradley. The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30(7) :1145–1159, July 1997.

[103] Takaya Saito and Marc Rehmsmeier. The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets. *PLoS ONE*, 10(3) :e0118432, March 2015.

[104] Jingkang Yang, Pengyun Wang, Dejian Zou, Zitang Zhou, Kunyuan Ding, Wenxuan Peng, Haoqi Wang, Guangyao Chen, Bo Li, Yiyou Sun, Xuefeng Du, Kaiyang Zhou, Wayne Zhang, Dan Hendrycks, Yixuan Li, and Ziwei Liu. OpenOOD : Benchmarking Generalized Out-of-Distribution Detection. *Advances in Neural Information Processing Systems*, 35 :32598–32611, December 2022.

[105] Konstantin Kirchheim, Marco Filax, and Frank Ortmeier. PyTorch-OOD : A Library for Out-of-Distribution Detection Based on PyTorch. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4351–4360, 2022.

[106] Eric Nalisnick, Akihiro Matsukawa, Yee Whye Teh, and Balaji Lakshminarayanan. Detecting Out-of-Distribution Inputs to Deep Generative Models Using Typicality. page 15.

[107] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST : A Novel Image Dataset for Benchmarking Machine Learning Algorithms. *arXiv :1708.07747 [cs, stat]*, September 2017.

[108] Takeru Miyato, Shin-Ichi Maeda, Masanori Koyama, and Shin Ishii. Virtual Adversarial Training : A Regularization Method for Supervised and Semi-Supervised Learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(8) :1979–1993, August 2019.

[109] Xiaojin Zhu and Zoubin Ghahramani. Learning from Labeled and Unlabeled Data with Label Propagation.

[110] Anne-Marie Lyne, Mark Girolami, Yves Atchadé, Heiko Strathmann, and Daniel Simpson. On Russian Roulette Estimates for Bayesian Inference with Doubly-Intractable Likelihoods. *Statistical Science*, 30(4) :443–467, November 2015.

[111] Anthony L. Caterini and Gabriel Loaiza-Ganem. Entropic Issues in Likelihood-Based OOD Detection. In *I (Still) Can't Believe It's Not Better ! Workshop at NeurIPS 2021*, pages 21–26. PMLR, February 2022.

[112] H. Drucker and Y. Le Cun. Improving generalization performance using double backpropagation. *IEEE Transactions on Neural Networks*, 3(6) :991–997, November 1992.

[113] Christian Etmann. A Closer Look at Double Backpropagation, June 2019.

[114] Yisheng Song, Ting Wang, Puyu Cai, Subrota K. Mondal, and Jyoti Prakash Sahoo. A Comprehensive Survey of Few-shot Learning : Evolution, Applications, Challenges, and Opportunities. *ACM Computing Surveys*, 55(13s) :271 :1–271 :40, July 2023.

[115] Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. Meta-Learning in Neural Networks : A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(09) :5149–5169, September 2022.

[116] Kai He, Nan Pu, Mingrui Lao, and Michael S. Lew. Few-shot and meta-learning methods for image understanding : A survey. *International Journal of Multimedia Information Retrieval*, 12(2) :14, June 2023.

[117] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-Learning with Memory-Augmented Neural Networks. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 1842–1850. PMLR, June 2016.

[118] Jake Snell, Kevin Swersky, and Richard S. Zemel. Prototypical Networks for Few-shot Learning, June 2017.

[119] Laurens van der Maaten and Geoffrey Hinton. Visualizing Data using t-SNE. *Journal of Machine Learning Research*, 9(86) :2579–2605, 2008.

[120] Samy Chali, Inna Kucher, Marc Duranton, and Jacques-Olivier Klein. Improving Normalizing Flows With the Approximate Mass for Out-of-Distribution Detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 750–758, 2023.

[121] Tarin Clanuwat, Mikel Bober-Irizar, Asanobu Kitamoto, Alex Lamb, Kazuaki Yamamoto, and David Ha. Deep Learning for Classical Japanese Literature, 9999.

[122] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and André van Schaik. EMNIST : An extension of MNIST to handwritten letters, March 2017.

[123] Mircea Cimpoi, Subhransu Maji, Iasonas Kokkinos, Sammy Mohamed, and Andrea Vedaldi. Describing Textures in the Wild. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3606–3613, June 2014.

[124] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. The German Traffic Sign Recognition Benchmark : A multi-class classification competition. In *The 2011 International Joint Conference on Neural Networks*, pages 1453–1460, July 2011.

[125] Bolei Zhou, Agata Lapedriza, Aditya Khosla, Aude Oliva, and Antonio Torralba. Places : A 10 Million Image Database for Scene Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(6) :1452–1464, June 2018.

[126] Grant Van Horn, Oisin Mac Aodha, Yang Song, Yin Cui, Chen Sun, Alex Shepard, Hartwig Adam, Pietro Perona, and Serge Belongie. The iNaturalist Species Classification and Detection Dataset. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8769–8778, Salt Lake City, UT, June 2018. IEEE.

[127] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks, February 2014.

[128] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going Deeper With Convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[129] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3) :211–252, December 2015.

[130] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. January 2013.

[131] Yang Yuan. On the Power of Foundation Models. In *Proceedings of the 40th International Conference on Machine Learning*, pages 40519–40530. PMLR, July 2023.

[132] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-Resolution Image Synthesis With Latent Diffusion Models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10684–10695, 2022.

[133] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D. Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish

Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners. *Advances in Neural Information Processing Systems*, 33 :1877–1901, 2020.

[134] Futa Waseda, Sosuke Nishikawa, Trung-Nghia Le, Huy H. Nguyen, and Isao Echizen. Closer Look at the Transferability of Adversarial Examples : How They Fool Different Models Differently. In *2023 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 1360–1368, Waikoloa, HI, USA, January 2023. IEEE.

[135] Shafi Goldwasser, Michael P. Kim, Vinod Vaikuntanathan, and Or Zamir. Planting Undetectable Backdoors in Machine Learning Models : [Extended Abstract]. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 931–942, October 2022.

[136] Bolun Wang. Improving and Securing Machine Learning Systems. 2018.

[137] Kui Ren, Tianhang Zheng, Zhan Qin, and Xue Liu. Adversarial Attacks and Defenses in Deep Learning. *Engineering*, 6(3) :346–360, March 2020.

[138] Francesco Croce and Matthias Hein. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. In *Proceedings of the 37th International Conference on Machine Learning*, pages 2206–2216. PMLR, November 2020.

[139] Francesco Croce and Matthias Hein. Minimally distorted Adversarial Examples with a Fast Adaptive Boundary Attack. In *Proceedings of the 37th International Conference on Machine Learning*, pages 2196–2205. PMLR, November 2020.

[140] Maksym Andriushchenko, Francesco Croce, Nicolas Flammarion, and Matthias Hein. Square Attack : A Query-Efficient Black-Box Adversarial Attack via Random Search. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision – ECCV 2020*, Lecture Notes in Computer Science, pages 484–501, Cham, 2020. Springer International Publishing.

[141] Jiakai Wang. Adversarial Examples in Physical World. In *Twenty-Ninth International Joint Conference on Artificial Intelligence*, volume 5, pages 4925–4926, August 2021.

[142] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards Deep Learning Models Resistant to Adversarial Attacks, September 2019.

[143] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. DeepFool : A Simple and Accurate Method to Fool Deep Neural Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2574–2582, 2016.

[144] Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. Robustness May Be at Odds with Accuracy. In *International Conference on Learning Representations*, September 2018.

[145] Florian Tramer. Detecting Adversarial Examples Is (Nearly) As Hard As Classifying Them. In *Proceedings of the 39th International Conference on Machine Learning*, pages 21692–21702. PMLR, June 2022.

[146] Peter Lorenz, Margret Keuper, and Janis Keuper. Unfolding Local Growth Rate Estimates for (Almost) Perfect Adversarial Detection, December 2022.

[147] Chen Ma, Chenxu Zhao, Hailin Shi, Li Chen, Junhai Yong, and Dan Zeng. MetaAdvDet : Towards Robust Detection of Evolving Adversarial Attacks. In *Proceedings of the 27th ACM International Conference on Multimedia*, MM '19, pages 692–701, New York, NY, USA, October 2019. Association for Computing Machinery.

[148] Ruize Gao, Feng Liu, Jingfeng Zhang, Bo Han, Tongliang Liu, Gang Niu, and Masashi Sugiyama. Maximum Mean Discrepancy Test is Aware of Adversarial Attacks. In *Proceedings of the 38th International Conference on Machine Learning*, pages 3564–3575. PMLR, July 2021.

[149] Laurent Amsaleg, Oussama Chelly, Teddy Furon, Stéphane Girard, Michael E. Houle, Ken-ichi Kawarabayashi, and Michael Nett. Estimating Local Intrinsic Dimensionality. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, pages 29–38, New York, NY, USA, August 2015. Association for Computing Machinery.

[150] Michael E. Houle. Local Intrinsic Dimensionality I : An Extreme-Value-Theoretic Foundation for Similarity Applications. In Christian Beecks, Felix Borutta, Peer Kröger, and Thomas Seidl, editors, *Similarity Search and Applications*, Lecture Notes in Computer Science, pages 64–79, Cham, 2017. Springer International Publishing.

[151] Sachin Ravi and Hugo Larochelle. Optimization as a Model for Few-Shot Learning. In *International Conference on Learning Representations*, November 2016.

[152] Arthur Gretton, Karsten M. Borgwardt, Malte J. Rasch, Bernhard Schölkopf, and Alexander Smola. A Kernel Two-Sample Test. *Journal of Machine Learning Research*, 13(25) :723–773, 2012.

[153] Nicholas Carlini and David Wagner. Adversarial Examples Are Not Easily Detected : Bypassing Ten Detection Methods. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, AISec '17, pages 3–14, New York, NY, USA, November 2017. Association for Computing Machinery.

[154] C. F. J. Wu. Jackknife, Bootstrap and Other Resampling Methods in Regression Analysis. *The Annals of Statistics*, 14(4) :1261–1295, December 1986.

[155] Mitch Hill, Jonathan Mitchell, and Song-Chun Zhu. STOCHASTIC SECURITY : ADVERSARIAL DEFENSE USING LONG-RUN DYNAMICS OF ENERGY-BASED MODELS. 2021.

[156] Vignesh Srinivasan, Csaba Rohrer, Arturo Marban, Klaus-Robert Müller, Wojciech Samek, and Shinichi Nakajima. Robustifying models against adversarial attacks by Langevin dynamics. *Neural Networks*, 137 :1–17, May 2021.

[157] Yoshua Bengio, Li Yao, Guillaume Alain, and Pascal Vincent. Generalized Denoising Auto-Encoders as Generative Models. In *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013.

[158] Guillaume Alain and Yoshua Bengio. What Regularized Auto-Encoders Learn from the Data-Generating Distribution.

[159] Jason Lee, Elman Mansimov, and Kyunghyun Cho. Deterministic Non-Autoregressive Neural Sequence Modeling by Iterative Refinement. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1173–1182, Brussels, Belgium, October 2018. Association for Computational Linguistics.

[160] Eric P. Lehman, Rahul G. Krishnan, Xiaopeng Zhao, Roger G. Mark, and Li-wei H. Lehman. Representation Learning Approaches to Detect False Arrhythmia Alarms from ECG Dynamics. In *Proceedings of the 3rd Machine Learning for Healthcare Conference*, pages 571–586. PMLR, November 2018.

[161] Tianjin Huang, Vlado Menkovski, Yulong Pei, and Mykola Pechenizkiy. Bridging the Performance Gap between FGSM and PGD Adversarial Training, October 2022.

[162] Erik Nijkamp, Mitch Hill, Tian Han, Song-Chun Zhu, and Ying Nian Wu. On the Anatomy of MCMC-Based Maximum Likelihood Learning of Energy-Based Models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04) :5272–5280, April 2020.

[163] Hoki Kim. Torchattacks : A PyTorch Repository for Adversarial Attacks, February 2021.

[164] Nicolas Papernot, Fartash Faghri, Nicholas Carlini, Ian Goodfellow, Reuben Feinman, Alexey Kurakin, Cihang Xie, Yash Sharma, Tom Brown, Aurko Roy, Alexander Matyasko, Vahid Behzadan, Karen Hambardzumyan, Zhishuai Zhang, Yi-Lin Juang, Zhi Li, Ryan Sheatsley, Abhibhav Garg, Jonathan Uesato, Willi Gierke, Yinpeng Dong, David Berthelot, Paul Hendricks, Jonas Rauber, Rujun Long, and Patrick McDaniel. Technical Report on the CleverHans v2.1.0 Adversarial Examples Library, June 2018.

[165] Maria-Irina Nicolae, Mathieu Sinn, Minh Ngoc Tran, Beat Buesser, Ambrish Rawat, Martin Wistuba, Valentina Zantedeschi, Nathalie Baracaldo, Bryant Chen, Heiko Ludwig, Ian M. Molloy, and Ben Edwards. Adversarial Robustness Toolbox v1.0.0, November 2019.

[166] Nicholas Carlini and David Wagner. Towards Evaluating the Robustness of Neural Networks, March 2017.

[167] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated Gradients Give a False Sense of Security : Circumventing Defenses to Adversarial Examples, July 2018.

[168] Francesco Croce, Maksym Andriushchenko, Vikash Sehwag, Edoardo Debenedetti, Nicolas Flammarion, Mung Chiang, Prateek Mittal, and Matthias Hein. RobustBench : A standardized adversarial robustness benchmark, October 2021.

[169] Nicholas Carlini, Anish Athalye, Nicolas Papernot, Wieland Brendel, Jonas Rauber, Dimitris Tsipras, Ian Goodfellow, Aleksander Madry, and Alexey Kurakin. On Evaluating Adversarial Robustness, February 2019.