University of Padova

Department of Physics "Galileo Galilei"

Master Thesis in Physics of Data

Biological Networks as Defense against Adversarial Attacks

Supervisor
Prof. Marco Baiesi
University of Padova

Master Candidate
Andrea Zanola

Student ID
2027588

Academic Year
2021-2022

Quand'ero piccolo, mia madre mi diceva:
"Se farai il soldato, sarai generale; se diventerai monaco, sarai papa".
Ho voluto fare il pittore, e sono diventato Picasso!
— Pablo Picasso

Abstract

In recent years, more and more importance is given to interpretability in the machine learning field. The best known and most famous area in which the interpretability of a neural network is needed is that of cyber-security. The first paper to expose the potential issue is by Szegedy et al. (2014), in ”Intriguing properties of neural networks”, in which it is shown how an image, if altered in the right way, can be completely misclassified by a network trained to classify images.

In this thesis I proposed a new method based on a hybrid network, i.e. half biological and half artificial, in order to develop a neural network that shows adversarial robustness, capable of resisting to many different adversarial attacks. The biological part will be based on the hebbian and anti-hebbian neural dynamics, while the artificial one will be based on specialized neurons and probability.

Contents

Listing of figures

Listing of tables

Listing of acronyms

AI	Artificial Intelligence
XAI	eXplainable Artificial Intelligence
ML	Machine Learning
DL	Deep Learning
ANN	Artificial Neural Network
DNN	Deep Neural Network
BNN	Biological Neural Network
K-H	Krotov and Hopfield
BP	Back-Propagation
pdf	Probability Density Function

1 Introduction

Nowadays in developing and implementing new artificial intelligence algorithms, more and more importance is given by the scientific community to characteristics like robustness and interpretability. Sometimes words like understandability or explainability are used as synonymous, but it will be shown that is not always the case. Indeed the acronym XAI (explainable artificial intelligence) expresses the evidence that standard (not explainable) AI solutions are not perceived as reliable by the users. The main hypothesis is that by building more transparent, interpretable or explainable systems, users will understand better the solution and therefore trust the intelligent agent [7]. However, it is important to notice that the solution to explainable AI is not just "more AI"; it is a human-agent interaction problem and in particular XAI is just one problem inside the bigger framework of human-agent interaction [7]. The lack of interpretability for predictive models can have (and has already had) critical consequences: there have been cases of people incorrectly denied parole, bad decisions were made leading to the release of dangerous criminals, machine learning based pollution models stating that highly polluted air was safe to breathe (see [8] for more).

In this thesis, the main goal is to understand as much as possible what an artificial neural network is learning during training and why such networks are weak in terms of adversarial vulnerability, a concept that will be discussed later. In some sense, the biggest effort is to analyze an algorithm not from the machine point of view, but from the human point of view. Indeed, recognising digits is a task that a child with little or no-effort can do; so why complicated algorithms, trained many hours using GPUs, can be easily fooled by a slightly perturbation of the input, called adversarial example? It is very annoying that humans can do complex tasks so easily, while coding efficiently a neural network can be extremely hard. So there is the hope that developing algorithms based on some human intuitions, could make them more resilient to adversarial attacks. Moreover, even if adversarial vulnerability is only mentioned during the master courses of ML and DL, it is a shame that these powerful algorithms that are able potentially to do crazy things, can be easily fooled as I will show in the next chapters. Finally, a unifying theory of adversarial attack does not exist, meaning that the ML community has not understood yet why ML algorithms are so vulnerable even though they work greatly for many real problems.

In some sense it is like quantum mechanics: many fundamental questions have been not explained, yet the modern technology is based on it and works incredibly well.

2 Human-based Concepts of ML

In this chapter I discuss four of the most frequently used words in the word of XAI: transparency, trust, interpretability and explainability.

2.1 Transparency

As property of a system, transparency addresses how a model works or functions internally, i.e. it is about how much it is possible to understand about a system's inner workings in theory. Transparency is further divided into simulatability (an understanding of the functioning of the model), decomposability (understanding of the individual components) and algorithmic transparency (visibility of the algorithms).

- **Simulatability**: roughly speaking, a model might be called transparent if a person can contemplate the entire model at once. This definition suggests that the model should be a simple one. In order to consider a model fully understood, a human should be able to take the input data together with the parameters of the model and in a reasonable amount of time go through every calculation required and produce a prediction [9].
- **Decomposability**: another notion of transparency might be that, each part of the model i.e. input, parameters and calculations admits an intuitive explanation. This notion of transparency requires that inputs themselves must be individually interpretable [9].
- **Algorithmic transparency**: a final notion of transparency might be applied at the level of the learning algorithm itself. In the case of linear models, one may understand the shape of the error surface and can prove that training will converge to a unique solution, even for new data sets. On the other hand, modern DL-methods lack this sort of algorithmic transparency. While optimization procedures for neural networks are incredibly powerful, it is not clear yet why and how they work [9].

In general, **understandable** models are sometimes called transparent, while incomprehensible models are called black-boxes.

Definition 2.1.1. (**Black-box** algorithms)
With the term **black-box** AI, is identified any artificial intelligence system whose inputs and operations are not visible to the user or another interested party.

In ML, black-box models are created directly from data by an algorithm, meaning that humans, even those who design them, cannot understand how variables are being combined to make predictions. Even if one has a list of the input variables, black-box predictive models can be such complicated functions of the variables that no human can understand how the variables are jointly related to each other to reach a final prediction [10]. What makes a system a "black-box"?

- **Complexity**: in particular this is the case for non-parametric models, i.e. those models in which the number of parameters is huge. An example of these are ANN, where each connection weight is a learnable parameter of the model; typically the system learns their values during the training phase. Because the operation of the neural network depends on the complicated interactions between these values, it is practically impossible, even for small networks, to understand how the network works even if all the parameters are known. This means that even though the algorithm is based on some arbitrary complicated equations written by the user, for example back-propagation, this does not imply that the user is able to understand the final solution.
- **Difficulty of developing explainable solutions**: even if the used AI model supports some level of explainability, it may be difficult to create a user experience for careful yet easily understandable explanations for the users.
- **Risk concerns**: many AI algorithms can be fooled if an attacker carefully designs an input that causes the system to malfunction. In a transparent system, where the attacker knows everything about the algorithm, it is easier to fool the system and come up with strange or unwanted results. Thus, sometimes systems are intentionally designed as black-boxes, in order to decrease the so called adversarial vulnerability.

Definition 2.1.2. (**White-box** algorithms)
With the term **white-box** AI, is identified any artificial intelligence system whose inner logic, workings and programming steps are transparent and therefore its decision making process is interpretable.

The most common examples of white-box models are linear regression models, Bayesian networks and decision trees.

2.2 Trust

With the rise of poorly-understood machine learning models in the field of AI, trust is often cited as a key desirable property of the interaction between any user and the AI agent. Citing a sentence taken from [1]:

"By designing AI that users can and will trust to interact with, AI can be safely implemented in society."

To understand human trust in AI (called Human-AI trust), it is useful to examine how people trust each other.

Definition 2.2.1. (Interpersonal Trust)
If A believes that B will act in A's best interest, and accepts vulnerability to B's actions, then A trusts B.

In particular, interpersonal trust exists to reduce the uncertainty and risk of collaboration by enabling the trustor's ability to anticipate the trustee, where "anticipating" refers to a belief that the trustee will act in the trustor's best interests. Following Hoffman [11]:

"*trust is an attempt to anticipate the impact of behavior under risk.*"

This means that risk is a prerequisite to the existence of human-AI trust. Ideally, the existence of trust can only be verified after verifying the existence of risk. However the ability to anticipate the AI behaviour is a goal, but not necessarily a symptom, of human-AI trust. Indeed anticipating the intended behavior is the user's goal in developing trust, but not necessarily the AI developer's goal.

Another important definition is the concept of contract and contractual trust.

Definition 2.2.2. (Contractual Trust)

Contractual trust is when a trustor has a belief that the trustee will stick to a specific contract.

Generally speaking, a contract refers to any functionality that could be useful, even if it does not imply an increasing of the mere performances. Therefore, model correctness, or commonly called test accuracy, is only one instance of contractual trust. What are other useful contracts? The European Commission has outlined detailed guidelines [12] on what should be required from AI models for them to be trustworthy; each of these requirements can be used to specify a useful contract (see Fig. 2.1). Note that wide trust is built on many contracts, each involving many factors and requiring different evaluation methods.

Definition 2.2.3. (Trustworthy AI)

An AI model is trustworthy to contract C if it is capable of maintaining the contract.

Definition 2.2.4. (Human-AI trust)

If H (human) perceives that M (AI model) is trustworthy to contract C, and accepts vulnerability to M's actions, then H trusts M contractually to C. The objective of H in trusting M is to anticipate that M will maintain C in the presence of uncertainty and consequently, trust does not exist if H does not perceive risk.

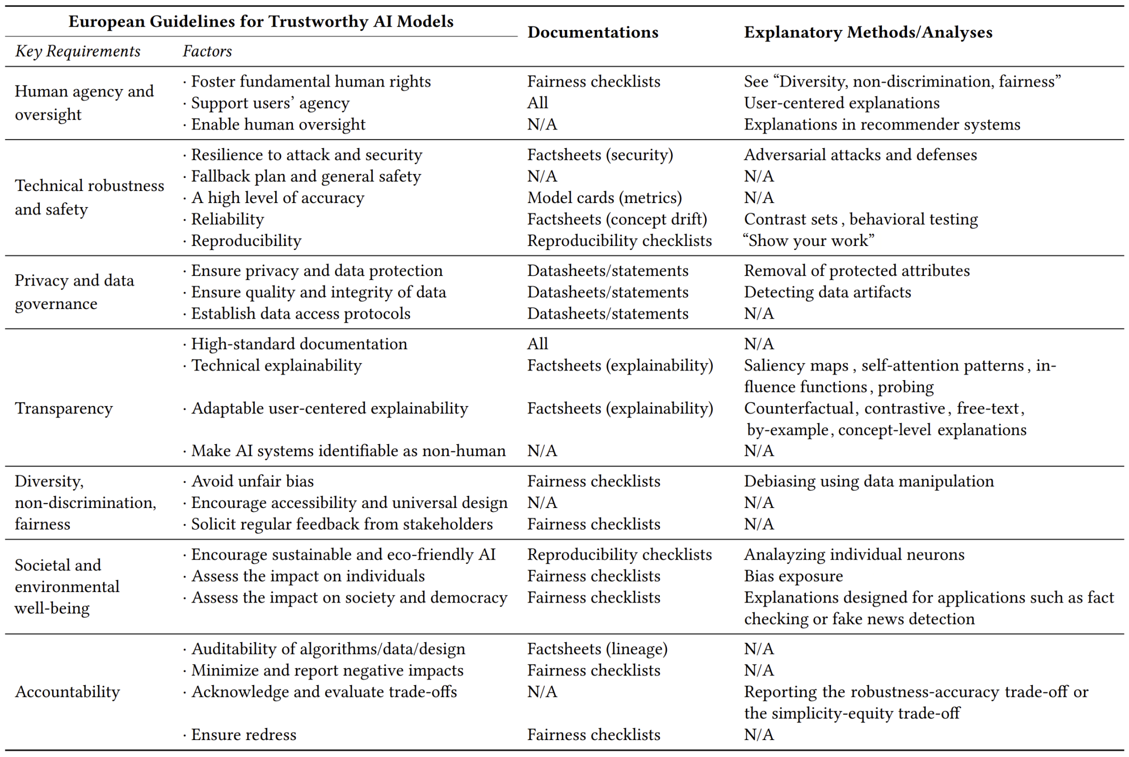
In conclusion, all these definitions are not here just for mathematical fun, but because human-AI trust gives a fundamental lesson, that many students forget when they build ML algorithms: humans can trust AI models only when the risk of failing is perceived. Consequently it is pointless the desire of having an algorithm that never fail, as described by the No-Free Lunch (NFL) theorem.

2.3 Interpretability

In general it is difficult to mathematically define interpretability and moreover there is no agreement within the ML community on the definition of interpretability and the task of interpretation. Here below it is reported two definitions, the first one given by Miller [7] and the second given by Kim et al. [13].

Definition 2.3.1. (Miller)

Interpretability is the degree to which a human can understand the cause of a model's decision.



European Guidelines for Trustworthy AI Models		Documentations	Explanatory Methods/Analyses
Key Requirements	*Factors*		
Human agency and oversight	· Foster fundamental human rights	Fairness checklists	See "Diversity, non-discrimination, fairness"
	· Support users' agency	All	User-centered explanations
	· Enable human oversight	N/A	Explanations in recommender systems
Technical robustness and safety	· Resilience to attack and security	Factsheets (security)	Adversarial attacks and defenses
	· Fallback plan and general safety	N/A	N/A
	· A high level of accuracy	Model cards (metrics)	N/A
	· Reliability	Factsheets (concept drift)	Contrast sets , behavioral testing
	· Reproducibility	Reproducibility checklists	"Show your work"
Privacy and data governance	· Ensure privacy and data protection	Datasheets/statements	Removal of protected attributes
	· Ensure quality and integrity of data	Datasheets/statements	Detecting data artifacts
	· Establish data access protocols	Datasheets/statements	N/A
Transparency	· High-standard documentation	All	N/A
	· Technical explainability	Factsheets (explainability)	Saliency maps , self-attention patterns , influence functions , probing
	· Adaptable user-centered explainability	Factsheets (explainability)	Counterfactual, contrastive , free-text , by-example , concept-level explanations
	· Make AI systems identifiable as non-human	N/A	N/A
Diversity, non-discrimination, fairness	· Avoid unfair bias	Fairness checklists	Debiasing using data manipulation
	· Encourage accessibility and universal design	N/A	N/A
	· Solicit regular feedback from stakeholders	Fairness checklists	N/A
Societal and environmental well-being	· Encourage sustainable and eco-friendly AI	Reproducibility checklists	Analayzing individual neurons
	· Assess the impact on individuals	Fairness checklists	Bias exposure
	· Assess the impact on society and democracy	Fairness checklists	Explanations designed for applications such as fact checking or fake news detection
Accountability	· Auditability of algorithms/data/design	Factsheets (lineage)	N/A
	· Minimize and report negative impacts	Fairness checklists	N/A
	· Acknowledge and evaluate trade-offs	N/A	Reporting the robustness-accuracy trade-off or the simplicity-equity trade-off
	· Ensure redress	Fairness checklists	N/A

Figure 2.1: The European requirements for trustworthy AI, related available documentation, and related explanatory methods or analyses. Table 1 and caption, taken from [1].

Definition 2.3.2. (Kim)
Interpretability is the degree to which a human can consistently predict the model's result.

Even if these two are definitions, they are not in a mathematical sense. So despite the lack of a rigorous definition, a growing body of literature proposes purportedly interpretable algorithms, consequently one could conclude that:

- either the definition of interpretability is universally agreed upon, but no one has bothered to set it in writing [9],
- or the term interpretability is ill-defined [9].

Indeed in the european guidelines reported above, interpretability is not present on the list of key requirements; however exists a long list of desiderata as transparency, robustness, trustworthiness so on and so forth.

2.4 Explainability

Rather than trying to create models that are completely interpretable, there has been a recent explosion of work on explainable AI, where a second separate model, called **posthoc**, is created to explain the first black-box model.

Most of the misunderstandings about explanation come from the term “explanation” itself, which is often used in a misleading way. Indeed, explanatory models do not always attempt to mimic the calculations made by the original one. Even a posthoc model that performs almost identically to a black-box model might use completely different features and thus is not faithful to the computation of the black-box.

A practical example is shown in [8] and here briefly reported. Consider a black-box model for criminal recidivism prediction where the goal is to predict whether someone will be arrested within a certain time after being released from jail. Most recidivism prediction models depend explicitly on age and criminal history, but do not explicitly depend on race. Since criminal history and age are correlated with race in all datasets, a legitimately accurate explanation model could construct a rule such as “This person will be arrested because he is black.” This might be an accurate explanation model since it correctly mimics the predictions of the original model, but it would not be faithful to what the original model actually does internally and moreover not ethic at all.

2.5 Final Remarks

Here I want briefly resume the main take-home message of the first chapter. Many definitions have been introduced, like trust, transparency, interpretability and explainability; other words are just used as synonymous, for example understandability. Although all these words are used with the same meaning in the colloquial language when describing neural networks, it has been shown that while trust and transparency have a particular and specific meaning, the other two are more heuristics. In particular, interpretability is an ill-defined property and the AI community has not yet found a mathematical and unique definition of it. At the same time interpretability is not equal to explainability, that instead is used to explain a black-box algorithm with another approximated model.

3
Theories of Adversarial Attacks

One of the key requirements shown in Fig. 2.1 was technical robustness and the suggested analyses was about adversarial attacks and defenses. This thesis indeed has, as main goal, the study and the development of a ML algorithm that will be adversarial robust, i.e. that cannot be easily fooled by an imperceptible perturbation of the input. Let's start saying that a unified theory of adversarial attacks does not exist, but rather there exist many possible ways to show that a neural network is fragile with respect to adversarial attacks. In particular in this chapter will be presented and quickly described, the main four theories taken in consideration by Li et al. [14] in their paper just to grasp the main ideas behind.

Before starting, let's remember the definition of L_p norm of a vector $\boldsymbol{x} \in \mathbb{R}^n$.

$$||\boldsymbol{x}||_p = \left(|x_1|^p + |x_2|^p + \cdots + |x_n|^p\right)^{\frac{1}{p}}$$

The most famous norms are:

- L_0 (Hamming norm), is not mathematically speaking a norm and can be obtained as $||\boldsymbol{x}||_0 = \lim_{p\to 0} ||\boldsymbol{x}||_p$. It corresponds to the number of non-zero elements of a vector.
- L_1 (Manhattan norm): $||\boldsymbol{x}||_1 = |x_1| + \cdots + |x_n|$
- L_2 (Euclidean norm): $||\boldsymbol{x}||_2 = \sqrt{x_1^2 + \cdots + x_n^2}$
- L_∞ (maximum norm): $||\boldsymbol{x}||_\infty = \max\{x_1, \ldots, x_n\}$

3.1 Low-probability "pockets" in the manifold

At the time of discovery of adversarial attacks, Szegedy et al. [2] interpreted adversarial examples as "blind spots" which belong to low-probability "pockets" in the data manifold. Consider a state-of-the-art deep neural network that performs greatly in the object recognition task; it is expected that such network is robust to small perturbations of its input, because small perturbation cannot change the object category of the image. However, the authors found that applying an imperceptible non-random perturbation to a test image, it is possible to arbitrarily change the network's prediction (see for example Fig. 3.1). These perturbations are found by optimizing the input to maximize the prediction error; the authors called these perturbed examples **adversarial examples**.

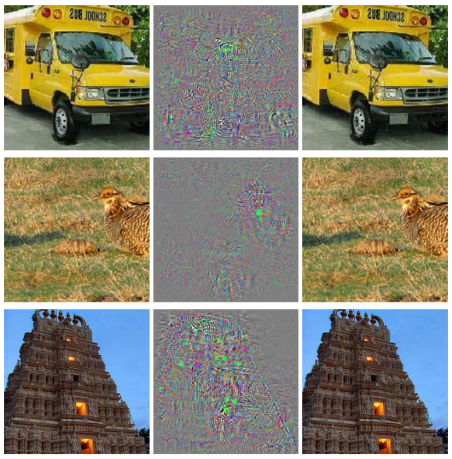


Figure 3.1: Adversarial examples generated for AlexNet. (Left) There are correctly predicted samples, (center) difference between correct images and images predicted incorrectly magnified by 10x (values shifted by 128 and clamped), (right) adversarial examples. All images in the right column are predicted to be an "ostrich, Struthio camelus". Image 5a and caption taken from [2].

What is the reason behind this unexpected behaviour? Generally speaking, the result in output from a neural network is a highly non-linear function of its input. When it is trained with the cross-entropy loss (using the softmax activation function), it represents a conditional distribution of the label given the input (and the training set presented so far). It has been argued [15] that the deep stack of non-linear layers in between the input and the output unit of a neural network is a way for the model to encode a non-local generalization prior over the input space. Yoshua Bengio explained very well the concept of non-local generalization in a Quora post [16]. With this term it is indicated an algorithm that is able to give good generalizations even for inputs that are far from those it has seen during training. It should be able to generalize to new combinations of the underlying concepts that explain the data. Nearest-neighbor methods, like decision trees or SVM, can only generalize in some neighborhood around the training examples, in a way that is simple, typically linear interpolation or extrapolation. Because the number of possible configurations of the underlying concepts that explain the data is exponentially large, this kind of generalization is good but not sufficient. Non-local generalization refers to the ability to generalize to a huge space of possible configurations of the underlying causes of the data, potentially very far from the observed data, going beyond linear combinations of training examples that have been seen in the neighborhood of the given input. In other words, it is assumed that is possible for the output unit to assign presumably non infinitesimal probabilities to regions of the input space that contain no training examples in their vicinity. Such regions (i.e. images) can represent, for instance, the same objects from different viewpoints, which are relatively far in pixel space, but which share both the label and the statistical structure of the original inputs. Obviously local-generalization near training examples works as expected, because it was argued that DNN has the most general ability of non-local generalization.

This imply that for a small enough radius $\eta = ||\boldsymbol{\eta}|| < \varepsilon$, where $\varepsilon > 0$ in the vicinity of a given input $\boldsymbol{x}$, an example $\boldsymbol{x} + \boldsymbol{\eta}$ will get assigned a high probability of the correct class by the model. Note that $\boldsymbol{\eta}$ is a signal with the same structure of $\boldsymbol{x}$ (e.g. an image) whose norm η, under some norm L_p, is less then ε. This kind of smoothness prior is typically valid for computer vision problems. With the term "smoothness" the authors refer to the fact that, thinking about the space of all the possible realizable images, starting from an arbitrary image classified with label l, one can continuously and slightly perturb the original image and yet the classifier assigns to the new image again the label l. The main result obtained by the authors is that for DNN, the smoothness assumption that underlies many kernel methods does not hold. In some sense, what the authors describe is a way to traverse the manifold represented by the network in an efficient way (by optimization) and finding adversarial examples in the input space. The adversarial examples represent then low-probability regions in the input space, which are hard to efficiently find by simply randomly sampling the input around a given example. Consequently this means that if one change random pixels in an image, probably he would not get an adversarial example and the image is correctly classified.

3.1.1 Gaussian Noise Attack

How to mathematically prove, that random noise is not effective in creating adversarial attacks? Suppose to perturb an input image $\boldsymbol{x}$ with i.i.d Gaussian noise, then the adversarial perturbation will be

$$\tilde{\boldsymbol{x}} = \boldsymbol{x} + \sigma \boldsymbol{r} \quad \boldsymbol{r} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}). \tag{3.1}$$

where σ is the standard deviation of the gaussian pdf. Now suppose to give in input to the network an image $\boldsymbol{x}$ near a linear boundary between two regions, classified with two different labels. As mentioned above, an adversarial perturbation can be written as $\tilde{\boldsymbol{x}} = \boldsymbol{x} + \boldsymbol{\eta}$. Now let's call $\boldsymbol{\eta}$ the direction perpendicular to the decision boundary such that, if one move $\boldsymbol{x}$ according to it, then $\tilde{\boldsymbol{x}}$ will be misclassified. Following this reasoning, there is a probability of $\frac{1}{2}$ to be misclassified, given a random perturbation $\boldsymbol{r}$. So the mathematical formulation can be written as

$$\boldsymbol{\eta}^T \cdot \boldsymbol{r} > 0 \tag{3.2}$$

i.e. the angle between these two vectors should be acute, in order to cause a misclassification. The problem of such argument is that in a high-dimensional space, the probability of $\boldsymbol{\eta}^T \cdot \boldsymbol{r} > 0$ is diminishing very quickly as the dimensionality of $\boldsymbol{r}$ grows; this is a well-known phenomenon called curse of dimensionality. One can easily find the probability that Eq. (3.2) happens, that is described by the following equation.

$$\mathbb{P}\left[\boldsymbol{\eta}^T \cdot \boldsymbol{r} \geq \delta\right] \leq \frac{||\boldsymbol{\eta}||}{\delta\sqrt{2\pi}} e^{-d^2 \frac{\delta^2}{2||\boldsymbol{\eta}^2||}} \tag{3.3}$$

Therefore, as $d \to \infty$ it becomes increasingly more difficult for i.i.d. Gaussian noise to succeed in attacking. For an image with dimensions 28×28 pixels, d is equal to 784.

3.2 Linearity of the model

One of the most widely accepted reasons for the origin of adversarial examples, is the linearity of the model in high-dimensional spaces, described by Goodfellow et al. in [17]. Note that the paper claims that linearity of the model in high-dimensional spaces seems to be a **sufficient condition** to develop adversarial attacks. How the authors explained the existence of adversarial examples for linear models? In many problems, the precision of the input is limited: for example, digital images often use only 8 bits per pixel so they discard all information below $\frac{1}{2^8-1} = \frac{1}{255}$ of the dynamic range. Because the precision of input data is limited, the classifier will not respond differently to an input $\boldsymbol{x}$ compared to an adversarial input $\tilde{\boldsymbol{x}} = \boldsymbol{x} + \boldsymbol{\eta}$, if every element of the perturbation $\boldsymbol{\eta}$ is smaller than the value of the less-significant bit. Formally, for problems with well-separated classes, it is expected that the classifier assigns the same class to $\boldsymbol{x}$ and $\tilde{\boldsymbol{x}}$ so long as $||\boldsymbol{\eta}||_\infty < \varepsilon$, where ε is small enough to be discarded by the sensor or data storage apparatus associated with the problem. Consider now the dot-product between a weight vector $\boldsymbol{w}$ and an adversarial example $\tilde{\boldsymbol{x}}$.

$$\boldsymbol{w}^T \cdot \tilde{\boldsymbol{x}} = \boldsymbol{w}^T \cdot \boldsymbol{x} + \boldsymbol{w}^T \cdot \boldsymbol{\eta} \tag{3.4}$$

The adversarial perturbation causes the activation to grow by $\boldsymbol{w}^T \cdot \boldsymbol{\eta}$. So one can maximize this increase subject to the max norm constraint on $\boldsymbol{\eta}$ (i.e. $||\boldsymbol{\eta}||_\infty < \varepsilon$) by assigning $\boldsymbol{\eta} = \varepsilon \cdot \text{sign}(\boldsymbol{w})$. If $\boldsymbol{w}$ has n dimensions and the average magnitude of an element of the weight vector is m, then the activation will grow by εmn. Since $||\boldsymbol{\eta}||_\infty$ does not grow with the dimensionality of the problem but the change in activation caused by perturbation by $\boldsymbol{\eta}$ can grow linearly with n, then for high-dimensional problems, one can make many infinitesimal changes to the input that add up to one large change to the output. In the MNIST example indeed, where the images are 28×28 the input space is $[0, 255]^{784}$.

3.3 Non robust features

In a recent work, Ilyas et al. in [18] shown how adversarial examples carry non-robust features, that indeed turns out to be a new perspective on the phenomenon of adversarial attacks. In contrast to the previous works, the authors interpreted adversarial vulnerability as a fundamental consequence of the dominant supervised learning paradigm. Recall that usually, one train a classifier to solely maximize the distributional accuracy. Consequently, classifiers tend to use any available information to do so, even those that look incomprehensible to humans. After all, the presence of "a tail" or "ears", that are features relevant for us as humans, is no more natural to a classifier than any other equally predictive feature. Finally, this perspective establishes adversarial vulnerability as a **human-centric phenomenon** since, from the standard supervised learning point of view, non-robust features can be as important as robust ones. This point of view also suggests that approaches aiming to enhance the interpretability of a given model forcing the use of "human-like" features for its explanation, actually hide features that are meaningful and predictive to standard models. What the authors mean with robust features?

Definition 3.3.1. (Feature)
A feature $\mathcal{F}$ is a function mapping the input space χ to the real numbers.

$$\mathcal{F} : \chi \to \mathbb{R} \tag{3.5}$$

Note that in non-mathematical terms, the word feature is also used to indicates some particular details in an image, useful for the task taken in consideration. As an example, in recognising cats from dogs, an example of useful features in the image are the dimensions of the animal, the texture of the skin and the relative dimension between the eyes and the skull etc.

Definition 3.3.2. (ρ-useful features)
For a given training set $\mathcal{D}$, a feature $\mathcal{F}$ is ρ-useful ($\rho > 0$), if it is correlated with the true label in expectation, that is if

$$\mathbb{E}_{(\boldsymbol{x},l)\sim\mathcal{D}}\left[l\mathcal{F}(\boldsymbol{x})\right] \geq \rho \tag{3.6}$$

where $(\boldsymbol{x}, l)$ indicates a training example, defined by $\boldsymbol{x}$ (e.g. an image) and its label l. For simplicity assume a binary classification task, i.e. $l = \pm 1$.

Crucially, a linear classifier trained on ρ-useful features can attain non-trivial performance.

Definition 3.3.3. (γ-robustly useful feature)
Suppose $\mathcal{F}$ to be a ρ-useful feature, then $\mathcal{F}$ is called robust feature, if under an adversarial perturbation (for some specified set of valid perturbations $\mathcal{B}_{\boldsymbol{x}}$), $\mathcal{F}$ remains γ-useful. Formally, if

$$\mathbb{E}_{(\boldsymbol{x},l)\sim\mathcal{D}}\left[\inf_{\boldsymbol{\eta}\in\mathcal{B}_{\boldsymbol{x}}} l\mathcal{F}(\boldsymbol{x}+\boldsymbol{\eta})\right] \geq \gamma. \tag{3.7}$$

Definition 3.3.4. (Useful, non robust feature)
A useful, non-robust feature is a feature which is ρ-useful for some ρ bounded away from zero, but is not a γ-robust feature for any $\gamma \geq 0$. These features help with classification in the standard setting, but may hinder accuracy in the adversarial setting.with the label can be flipped.

Within this theoretical framework, a binary classifier C is composed by a set of features $F = \{\mathcal{F}\}$, a weight vector $\boldsymbol{w}_f$ and a scalar bias b. For a given input $\boldsymbol{x}$, the simplest binary classifier predicts the label as follow

$$C(\boldsymbol{x}) = \operatorname{sgn}\left(b + \sum_{\mathcal{F}\in F} \boldsymbol{w}_f \cdot \mathcal{F}(\boldsymbol{x})\right).$$

So in this case, there is one single neuron connected to all other neurons in a previous layer through $\boldsymbol{w}_f$; but what there is in input to these neurons? These hidden neurons, received a highly non-linear function of the input, in particular they receive the input image convoluted with all the feature maps of the previous layers. In this terms, one can explain Eq. (3.6) in the following way: assuming that the network has all positive weights and neglecting the bias, the classifier C will give a label $+1$ only if the features $\mathcal{F}(\boldsymbol{x})$ are positive. So if in expectation the target label has the same sign of the feature, then the classifier is working correctly; consequently the feature is useful for the classification task.

The central premise of this proposed framework is that there exist both robust and non-robust features; how to provide evidence in support of this hypothesis? On one hand, the authors built a "robustified" dataset, consisting of samples that primarily contain robust features. Concretely, they created a training set (semantically similar to the original) on which standard training yields good robust accuracy on the original, unmodified test set. Moreover they were are also able to construct a training dataset for which the inputs are nearly identical to the originals, but all appear incorrectly labeled (adversarial attacks); for more please see [18]. This demonstrates that robustness can arise by removing certain features from the dataset (as, overall, the new dataset contains less information about the original training set). Moreover, it provides evidence that adversarial vulnerability is caused by non-robust features and is not inherently tied to the standard training framework. This indicates that natural models use non-robust features to make predictions, even in the presence of robust features.

3.4 Separation of $\mathbb{R}^n$ by hyperplanes.

Adi Shamir et al. in their work [3] proposed a quite intuitive possible explanation for the origin of adversarial attacks, based on geometric arguments. The analysis starts by considering the simplest case in which the mapping from inputs to outputs is determined by a sum of hyperplanes. Note that this setting differs from linear separators for multiclass classification, where the maximum over several linear classifiers is used to determine the classification. Instead, here there are m given hyperplanes of the form

$$\sum_{j=1}^{n} a_{ij}x_j + b_i \quad \text{for } i = \{1, \dots, m\}$$

which split $\mathbb{R}^n$ into many cells; in general one can assume that the hyperplanes are in general positions. If one denotes by A the $m \times n$ matrix whose entries are the a_{ij} coefficients and by B the column vector whose entries are the constants b_i, then each cell in the partition is defined by a particular vector S of m "$\pm$" signs, and consists of all the points $\boldsymbol{x} \in \mathbb{R}^n$ for which $A\boldsymbol{x} + B$ is a column vector of m numbers whose signs are as specified in S. The maximal possible number of cells has been shown to be $\sum_{i=0}^{n} \binom{m}{i}$ [3]. The predictor associates labels y (which are typically object classes such as "horse" or "car") to some of the cells. The union of the cells labeled by y can be arbitrarily shaped (in particular it need not be connected, convex or hole-free). The predictor classifies any input $\boldsymbol{x}$ by using the label of the cell in which it is located. Note that even a small number of hyperplanes (e.g. 20) suffices to split $\mathbb{R}^n$ into a huge number of cells and thus can be potentially used to recognize a large number of object classes (up to 2^{20}). A pictorial representation of this separation of the space is given in Fig. 3.2.

The goal of the authors was to study the neighborhood structure of such a partition of $\mathbb{R}^n$ under the L_0 distance function (the case with L_2 distance is still an open problem). The authors moreover were able to shown that given any two cells and a particular point $\boldsymbol{x}$ which is located in the first cell, the smallest possible number k of coordinates in $\boldsymbol{x}$ that one have to change in order to reach some point $\boldsymbol{z}$ which is located in the second cell, for moderately sized m is surprisingly small. Based on this theory, the authors developed an algorithm able to create adversarial attacks and then they applied it to the MNIST dataset. They were able also to find the $m = 11$ pixels having the largest

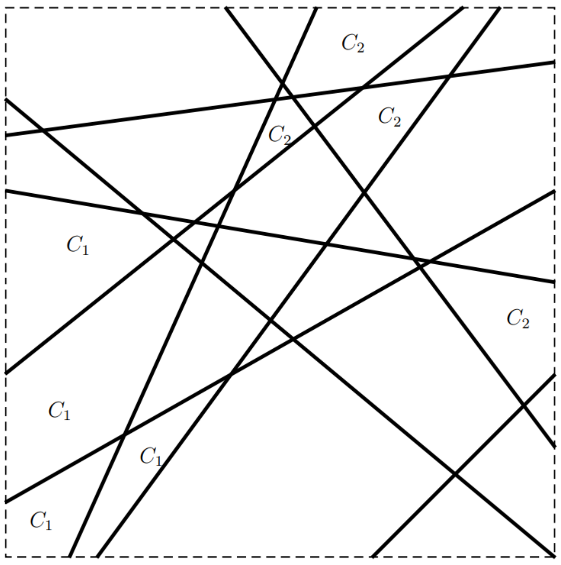


Figure 3.2: Separation of $\mathbb{R}^n$ by hyperplanes arrangement. Image 3 and caption taken from [3].

standard deviation among the images in the training set and perturbing the same 11 pixels, with different intensity, they were able to change the prediction of a digit labeled as 7 to that of any other digit.

3.5 Final Remarks

Here I want briefly resume the main take-home message from the second chapter, that will be useful later. From the original work done by Szegedy et al. (2014), an increasing number of publications have been made in the field of adversarial attacks, making it one of the currently most active fields of research. The original and fundamental observation, that was named as ”adversarial perturbation”, is that one can cleverly change the input by applying an imperceptible modification and a state-of-the-art DL architecture will be easily fooled. The first explanation for this phenomenon is based on the idea that adversarial examples represent low-probability regions in the manifold, left unexplored by the training. One year later, Goodfellow et al. give a simple demonstration of how a linear model can develop adversarial vulnerability if the input has a sufficient high-dimensionality, that in the case of images is easy to have. It is worth to remark that these two theories are not in contradiction with each other, but by the fact that both gives a meaningful explanation, surely we cannot talk of a unifying theory. Even though these two are the main accepted theories, many other theories and observations have been made through the years. One of the most famous is based on the concept of robust features; even though one can define the concept of robust features, it is important to remark that in this framework adversarial vulnerability is seen as a human-centric phenomenon. Features that for a human are naturally the most obvious to catch, could be different respect those that a neural network will learn during the training; this will be an important point later.

4
Adversarial Attacks

Although a unifying theory of adversarial attacks does not exist, computer scientists and machine learning engineers during the years have developed a full zoology of algorithms and techniques in order to create adversarial attacks. On the other hand researchers are also interested on adversarial defenses, namely techniques in order to improve the robustness of the AI algorithms. In this chapter I will describe the most famous and important adversarial attacks on the market, while adversarial defences are not here discussed since the final goal of the thesis. Let's start with some general definitions; very broadly speaking, adversarial attacks can be divided into two categories.

Definition 4.0.1. (Untargeted Attack)
An untargeted (or indiscriminate) attack aims to cause the DNN to return an incorrect result, such as a misclassification.

An example of this would be to avoid facial detection; so long as an image is not positively identified as a specific person, the actual DNN output is not important.

Definition 4.0.2. (Targeted Attack)
A targeted attack aims to generate a specific output from the DNN processing.

For example, to cause a digit to be classified as an other one. From now on, in order to explain the algorithms behind the construction of the adversarial attacks, one has to keep in mind that the reference theories kept in consideration are, generally speaking, the original ones i.e. [2] and [17].

One way of visualizing all the possible images that could be given in input to the DNN, would be to place each one at its own position in this high-dimensional input space. In such space, one dimension represents one pixel, meaning that for MNIST images there are 784 dimensions, with a total number of possible different images equal to $256^{784} \approx 10^{1888}$, that is much larger compared to the Avogadro's number. A pictorial and over-simplified representation of the landscape of such high-dimensional space, in relation with the multi-classification task, could

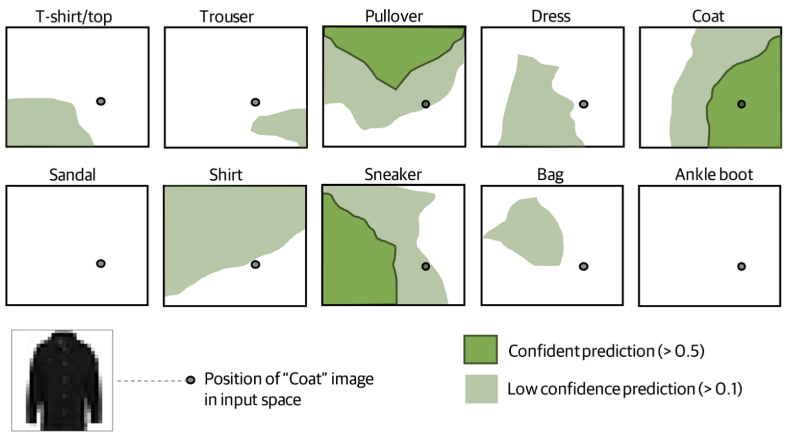

T-shirt/top
Trouser
Pullover
Dress
Coat
Sandal
Shirt
Sneaker
Bag
Ankle boot
Position of "Coat" image in input space
Confident prediction (> 0.5)
Low confidence prediction (> 0.1)

Figure 4.1: A model's prediction landscape for each classification. Image 5-2 and caption taken from [4].

be that one shown in Fig. 4.1. Although it should be clear that this high-dimensional space is the space of all possible images, maybe it is not visually clear which images this space could contain. In Fig. 4.2 are reported images from both ImageNet and Fashion-MNIST.

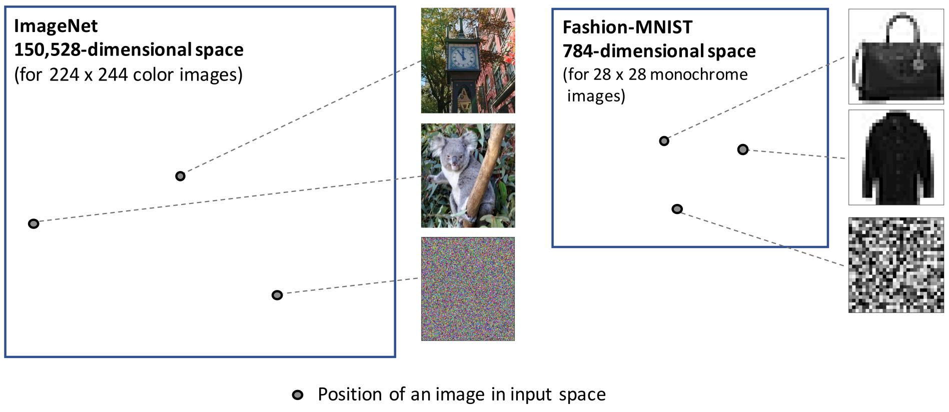

ImageNet
150,528-dimensional space
(for 224 x 244 color images)
Fashion-MNIST
784-dimensional space
(for 28 x 28 monochrome images)
Position of an image in input space

Figure 4.2: Input spaces outrageously simplified in two dimensions (obviously not in scale). Image 5-1 and caption taken from [4].

Let's begin by considering the challenge of changing many pixels very slightly to a Fashion-MNIST image, to result in a misclassification. Changing a selection of pixels in the image will shift it through the input space to another location, moving it across the landscapes depicted in Fig. 4.1. The image must be changed sufficiently so that its position within the input space is no longer within the "Coat" classification area. If this is a targeted attack, there is the additional constraint that the image has to be moved into an area of the input space defined by the corresponding target label. A possible useful representation is given in Fig. 4.3. So now should start to be clear that generation of adversarial perturbation comes down to the challenge of which pixels will cause the most change away from the correct classification, and possibly toward a target classification. This is not strange at all, indeed it is what has been done by Shamir et al. in [3], i.e. they were able to find the 11 pixels with highest

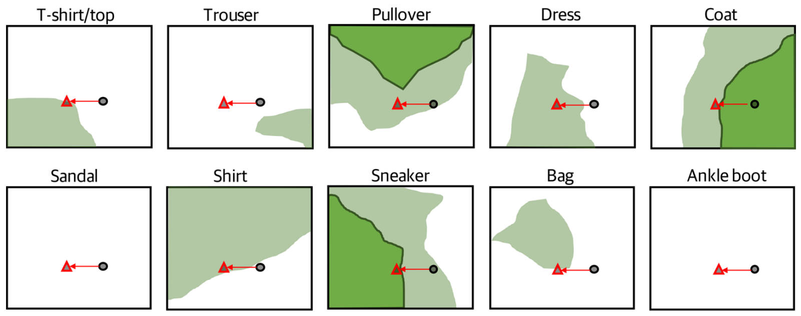


Figure 4.3: Untargeted attack-moving outside the "Coat" classification area of the input space. Image 5-8 and caption taken from [4].

variance. Remember that any perturbation required must be minimized so that it is insignificant to the human eye. In other words, ideally the perturbation is likely to be the minimum change to the image required to move it just outside the "Coat" classification boundary or just inside the target classification boundary. There are several approaches that one may take to create an adversarial attack, each assuming a different level of knowledge of the DNN architecture. These methods can be categorized based on the attacker level of access to the model.

Definition 4.0.3. (**White-box** attack)
A white-box attack exploits complete knowledge of the DNN model to create adversarial input, i.e. the attacker knows the entire set of weights and biases of the DNN.

Definition 4.0.4. (**Limited Black-box** attack)
A limited black-box attack refines adversarial input based on an output generated from the model or from the system in which it resides. For example, the output might be simply the label of a final classification.

Definition 4.0.5. (**Score-based Black-box** attack)
These methods refine adversarial input based on the raw predictions (scores) returned from the DNN. They require access to more detailed responses than a limited black-box attack, but do not require access to the model algorithm as a white-box attack.

Based on these definition, it is obvious how white-box attacks are more powerful respect black-box ones; so let's now focus on white-box attacks.

4.1 White-Box attacks

Let's denote with $\mathcal{L}\big(f(\boldsymbol{x}, \Theta); l\big)$, the loss function associated to a DNN, indicated by f, with a set of weights and biases known and fixed, denoted as Θ. Let's denote a given input image $\boldsymbol{x} \in \mathbb{R}^m$ and the corresponding target label l, where $(\boldsymbol{x}, l) \sim \mathcal{D}$ are extracted from the training set $\mathcal{D}$.
The notation $\mathcal{B}_{(\boldsymbol{x},\varepsilon)}$, indicates an ε-ball around an example $\boldsymbol{x}$ defined respect some specified norm L_p.
Up to now the word adversarial attack was used many times, but what is its rigorous definition?

Definition 4.1.1. (Targeted Adversarial Attack)

Let $\boldsymbol{x}$ be a data point belonging to class C_l (associated with the label l) and define a target class $C_{\tilde{l}}$ (associated to $\tilde{l}$). A (targeted) adversarial attack is a mapping $\mathcal{A} : \mathbb{R}^m \to \mathbb{R}^m$ such that the perturbed data is misclassified as $C_{\tilde{l}}$.

$$\tilde{\boldsymbol{x}} = \mathcal{A}(\boldsymbol{x}) \tag{4.1}$$

Among many types of adversarial attacks, the most commonly used is the additive one, i.e. $\mathcal{A}$ is a linear operator that adds a perturbation to the original sample. Note that these are also the types of attacks mentioned by the theories [2], [17] described before.

Definition 4.1.2. (Additive Targeted Adversarial Attack)

Let $\boldsymbol{x}$ be a data point belong to class C_l and define a target class $C_{\tilde{l}}$. An additive (targeted) adversarial attack is an addition of a perturbation $\boldsymbol{\eta}$ such that the perturbed data is misclassified as $C_{\tilde{l}}$.

$$\tilde{\boldsymbol{x}} = \boldsymbol{x} + \boldsymbol{\eta} \tag{4.2}$$

Finally, two other definitions useful for describing how adversarial attack are created, are reported below.

Definition 4.1.3. (Minimum Norm Attack)

The minimum norm attack finds a perturbed data $\tilde{\boldsymbol{x}}$ by solving the optimization

$$\min_{\tilde{\boldsymbol{x}}} ||\tilde{\boldsymbol{x}} - \boldsymbol{x}|| : \quad f(\boldsymbol{x}, \Theta) = l \neq \tilde{l} = f(\tilde{\boldsymbol{x}}, \Theta) \tag{4.3}$$

An alternative to the minimum norm attack is the maximum allowable attack.

Definition 4.1.4. (Maximum Allowable Attack)

The maximum allowable attack finds a perturbed data $\tilde{\boldsymbol{x}}$ by solving the optimization

$$\min_{\boldsymbol{\eta} \in \mathcal{B}_{(\boldsymbol{x}, \varepsilon)}} \mathcal{L}\big(f(\boldsymbol{x} + \boldsymbol{\eta}, \Theta); \tilde{l}\,\big) \tag{4.4}$$

where $\varepsilon > 0$ denotes the magnitude of the attack.

Now let's describe some of the most famous algorithms used for the construction of adversarial attacks.

4.1.1 L-BFGS Attack

The first algorithm created to generate adversarial attacks was developed in the paper written by Szegedy et al. [2]. The main idea behind is try to solve the **box-constrained** optimization problem, formulated in Alg. 4.1. Note that the last requirement in Alg. 4.1, implies that every pixel of the image has been previously normalized in the interval $[0, 1]$; informally, $\boldsymbol{x}+\boldsymbol{\eta}$ is the closest image to $\boldsymbol{x}$ classified as $\tilde{l}$ by f. The minimizer $\boldsymbol{\eta}$ might not be unique (i.e. there could be different perturbations that one can apply), so the considered one is arbitrarily chosen by $D(\boldsymbol{x}, \tilde{l}\,)$ also called "minimum distortion" function. Obviously this task is non-trivial only if $f(\boldsymbol{x}) = l \neq \tilde{l}$, indeed one can choose the minimum perturbation $\boldsymbol{\eta} = 0$ and the task in Eq. (4.5) is trivially solved. In general, the exact

Algorithm 4.1 Box-constrained optimization

Minimize the perturbation $||\boldsymbol{\eta}||_2$, subject to the following requirements:

$$\begin{cases} f(\boldsymbol{x}+\boldsymbol{\eta}) = \tilde{l} \\ \boldsymbol{x}+\boldsymbol{\eta} \in [0,1]^m \end{cases} \tag{4.5}$$

Algorithm 4.2 Box-constrained L-BFGS optimization

Find an approximation of $D(\boldsymbol{x}, \tilde{l})$ by performing line-search to find the minimum $c > 0$ for which the minimizer $\boldsymbol{\eta}$ of the following problem satisfies the two requirements shown in Alg. 4.1. In particular the minimization is not anymore over $||\boldsymbol{\eta}||_2$ but over

$$c||\boldsymbol{\eta}||_2 + \mathcal{L}\big(f(\boldsymbol{x}+\boldsymbol{\eta}, \Theta); \tilde{l}\,\big) \tag{4.6}$$

computation of $D(\boldsymbol{x}, \tilde{l}\,)$ is a hard problem, so the authors approximate it by using a modified version of Alg. 4.1, called box-constrained L-BFGS (Limited-memory Broyden–Fletcher–Goldfarb–Shanno), described in Alg. 4.2.

This penalty function method would yield the exact solution for $D(\boldsymbol{x}, \tilde{l}\,)$ in the case of convex losses, however neural networks are non-convex in general, so one end up with an approximation [2]. Note that the linear search is performed to find the constant $c > 0$ that yields an adversarial example of minimum distance [19].

4.1.2 FGSM Attack

In 2015, Goodfellow et al. in [17] proved a simple algorithm, the Fast Gradient Sign Method (FGSM), to be effective in generating adversarial examples. The FGSM has two key differences respect the L-BFGS method: first, it is optimized for the L_∞ norm (and not L_2) and second, it was designed primarily to be fast instead of producing very accurate adversarial examples [19]. The FGSM algorithm calculates the direction in the input space which, at the location of the image, appears to be the fastest route to a misclassification. This direction is calculated using gradient descent and a cost function similar to that for training a network. A concrete explanation is to simply think of the direction calculation as being an indirect measure of the steepness of the contours in the prediction landscape at the location of the image. The adversarial direction is roughly estimated, so each input value (i.e. pixel value) is assigned to be:

- $+1$: indicating that this input value would be best being increased to cause a misclassification.
- -1: indicating that this input value would be best being decreased to cause a misclassification.

With the direction established, FGSM then applies a small perturbation, denoted as ε, to every pixel, adding the perturbation if the adversarial direction is positive or subtracting the perturbation otherwise. The reason of the success of FGSM is all behind the assumption that the steepness of the slope in a particular direction will be maintained; mathematically, the function that the model represents exhibits linear behavior. Previous to FGSM, it was assumed that DNN algorithms comprised more complex nonlinear gradients, which one could imagine as a rough landscapes composed by hills and valleys. This linearity occurs because the optimization step during training will always favor the simplest model, i.e. the simplest gradients.

Let's now give a mathematical formulation of the FGSM attack. In order to calculate the direction, let's apply first the gradient of the loss function $\mathcal{L}$ and secondly the sign operation, i.e. $\operatorname{sign}\big(\nabla_{\boldsymbol{x}}\mathcal{L}\big(f(\boldsymbol{x},\Theta);y\big)\big)$; finally multiplying it by a small value ε, one obtain the adversarial perturbation.

Algorithm 4.3 FGSM

The adversarial attack $\tilde{\boldsymbol{x}}$ generated by the FGSM algorithm is:

$$\tilde{\boldsymbol{x}} = \boldsymbol{x} + \varepsilon \times \operatorname{sign}\big[\nabla_{\boldsymbol{x}}\mathcal{L}\big(f(\boldsymbol{x},\Theta);l\big)\big] \tag{4.7}$$

Note that as mentioned before, the typical resolution of an image is 8 bit, so ε must be grater then $1/255$, in order to have $\tilde{\boldsymbol{x}} \neq \boldsymbol{x}$.

4.1.3 IFGSM Attack (or BIM Attack)

In 2017, Kurakin et al. in [20] suggest a very simple improvement to FGSM. The idea is to apply FGSM multiple times with a small step size α and clip pixel values of intermediate results after each step to ensure that they are in an ε-neighbourhood of the original image.

Algorithm 4.4 IFGSM

The adversarial attack $\tilde{\boldsymbol{x}}$ generated by the Iterative-FGSM algorithm is

$$\begin{cases} \tilde{\boldsymbol{x}}_0 = \boldsymbol{x} \\ \vdots \\ \tilde{\boldsymbol{x}}_{N+1} = \operatorname{Clip}_{\boldsymbol{x},\varepsilon}\big\{\tilde{\boldsymbol{x}}_N + \alpha \cdot \operatorname{sign}\big(\nabla_{\boldsymbol{x}}\mathcal{L}\big(f(\boldsymbol{x},\Theta);l\big)\big)\big\} \end{cases} \tag{4.8}$$

where the function $\operatorname{Clip}_{\boldsymbol{x},\varepsilon}\{\boldsymbol{x}'\}$ performs per-pixel clipping of the image $\boldsymbol{x}'$, so the result will be in L_∞ ε-neighbourhood of the source image $\boldsymbol{x}$. The exact Clip function, for a general RGB image is:

$$\operatorname{Clip}_{\boldsymbol{x},\varepsilon}(\tilde{\boldsymbol{x}})(x,y,z) = \min\big\{255, \boldsymbol{x}(x,y,z) + \varepsilon, \max\{0, \boldsymbol{x}(x,y,z) - \varepsilon, \tilde{\boldsymbol{x}}(x,y,z)\}\big\} \tag{4.9}$$

The number of iterations was chosen heuristically by the authors to be $\min(\varepsilon + 4, 1.25\varepsilon)$; indeed it is sufficient for the adversarial example to reach the edge of the ε max-norm ball but restricted enough to keep the computational cost of experiments low.

4.1.4 PGD Attack

The Projected Gradient Descent (PGD) attack is essentially the same as IFGSM attack, where the only difference is that PGD initializes the example to a random point in the ball of interest (decided by the L_∞ norm) and does random restarts, while IFGSM initializes to the original point $\boldsymbol{x}$.

Algorithm 4.5 PGD

The adversarial attack $\tilde{\boldsymbol{x}}$ generated by the PGD algorithm is

$$\begin{cases} \tilde{\boldsymbol{x}}_0 = \boldsymbol{x} + \mathcal{U}(-\varepsilon, +\varepsilon) \\ \vdots \\ \tilde{\boldsymbol{x}}_{N+1} = \mathrm{Clip}_{\boldsymbol{x},\varepsilon}\left\{\tilde{\boldsymbol{x}}_N + \alpha \cdot \mathrm{sign}\left(\nabla_{\tilde{\boldsymbol{x}}_N} \mathcal{L}\left(f(\tilde{\boldsymbol{x}}_N, \Theta); l\right)\right)\right\} \end{cases} \tag{4.10}$$

where $\mathcal{U}$ is a uniform distribution, acting in d dimensions.

4.1.5 RFGSM Attack

This method developed by Tramèr et al. in [21] is a slightly modified version of the original FGSM, indeed the acronym stays for Random-step FGSM. This algorithm can be seen as a single-step variant of the general PGD algorithm.

Algorithm 4.6 RFGSM

The adversarial attack $\tilde{\boldsymbol{x}}$ generated by the Random-FGSM algorithm is

$$\tilde{\boldsymbol{x}} = \boldsymbol{x} + \alpha \cdot \mathrm{sign}\left(\mathcal{N}(0^d, 1^d)\right) + (\varepsilon - \alpha) \cdot \mathrm{sign}\left(\nabla_{\boldsymbol{x}} \mathcal{L}\left(f(\boldsymbol{x} + \alpha \cdot \mathrm{sign}\left(\mathcal{N}(0^d, 1^d)\right), \Theta); l\right)\right) \tag{4.11}$$

where $\mathcal{N}$ represents the noise, i.e. the small random perturbation. Note that α must be less then ε.

5
Biological Neural Networks

Let's stop for a moment the discussion about the artificial neural networks in order to introduce the concept of biological neural networks. The main properties of a BNN will be useful later on in developing the proposed algorithm that will be adversarial robust. In this chapter it is given a quick overview of biological neural network, focusing on the most important aspects relevant for the purposes of this thesis.

5.1 Neurobiology

Even though a complete comprehension of the human's brain is not achieved yet, lot of studies have been done since the pioneering work of Hodgkin and Huxley. Let's focus on one single neuron, trying to understand better which approximations are needed to make its modelling efficient, nevertheless keeping it as simple as possible. A simple schema of a neuron is shown in Fig. 5.1.

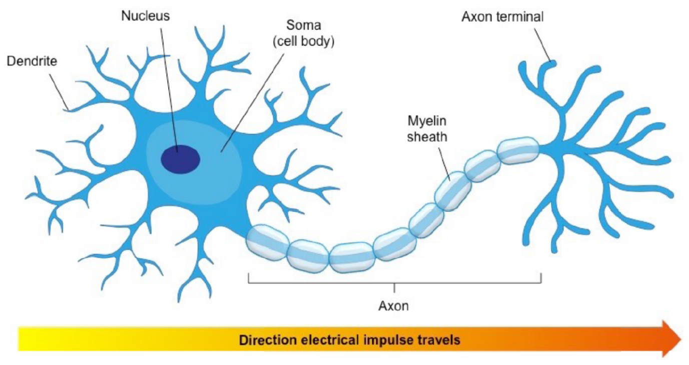


Figure 5.1: Schematic representation of a neuron.

It essentially consists of a single cell, whose body is named soma, which contains a nucleus. The ramifications near the soma are called dendrites, allowing for receiving signals from other neurons and propagating them to the soma; axons instead are the ramifications that transmits the signal towards other neurons. Finally, synapses are either chemical or physical junctions placed in the axon terminal and allow for signal transmission to other neurons' dendrites. Every neuron has many dendrites, but a single axon. Note that the propagation of signals is usually asymmetric, i.e. directional; indeed, it does not occur that receptors send neurotransmitters back to the axon, through the synapse. This is an important point that will be discussed later. The electric propagation of the signal through the axon is accomplished through channels on the cell membrane, that can pump in/out ions, generating an electric potential difference. A signal begins by depolarizing the first part of the axon (closest to the soma), by opening Na+ channels. Initially, membrane potential is at rest, with a voltage of $V_i = -70\text{mV}$. Shortly after, also the K+ channels open, and counterbalance the potential difference, thus decreasing back the membrane potential. For small voltages increases, the K+ current exceeds the Na+ current and voltage returns to its resting value, hence stopping the signal propagation. However, if the critical threshold is surpassed ($V_{th} = -55\text{mV}$), the Na+ ions dominate, and the process explodes triggering the propagation: the Na+ influx spreads to the closest regions, while regions that have already "fired" close, allowing the K+ mechanism to re-balance, thus coming back to the initial resting potential $V_f = -70\text{mV}$.

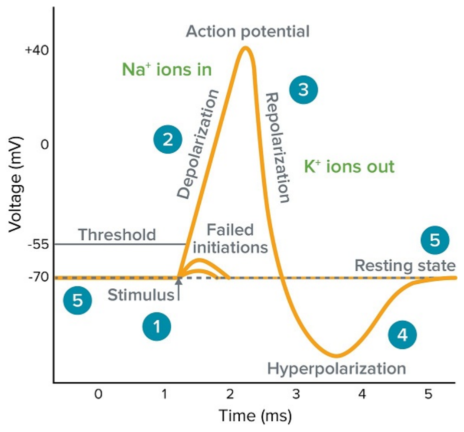


Figure 5.2: Action potential diagram of a neuron.

In order to make the description more quantitative let's define some quantities of interest.

Definition 5.1.1. (Neural response function)

The neural response function of a neuron after a stimulus S_t is defined as

$$\rho(t) = \sum_{i=1}^{n} \delta(t - t_i) \tag{5.1}$$

where t_i is the time of the i-th spike; $\rho(t)$ is then the sum of a neuron' spike train.

The most natural way to define a firing rate is counting the number of spikes in a specified interval of time, however this definition is very sensitive to the chosen Δt.

Definition 5.1.2. (Spike-time dependent firing rate)
The spike-time dependent firing rate is defined as

$$r(t) = \frac{1}{\Delta t} \int_{t}^{t+\Delta t} \langle \rho(\tau) \rangle d\tau \tag{5.2}$$

where the average $\langle \cdot \rangle$ is over different trials.

Note that the choice of Δt is crucial for a good definition of $r(t)$ because if $\Delta t \gg 1$ one will always find a spike arriving from the neuron, while if $\Delta t \ll 1$ one will never find a neuron spiking. So in general $\rho(t)$ is a noisy quantitative, that is why the average is done over different trials.

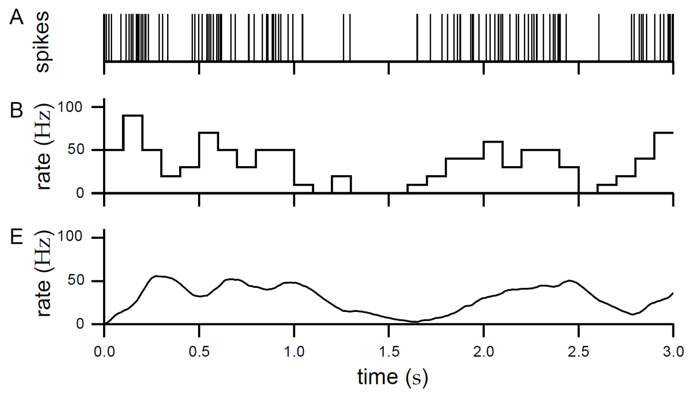


Figure 5.3: (A) An example of spike train, (B) discrete time firing rate obtained by binning time and counting spikes, while (E) approximate firing rate using a window function. Image taken from [5].

5.2 Firing Rate Model

Based on this description McCulloch and Pitts in [22] proposed a model of a neuron years later called **McCulloc-Pitts neuron**; its schematic representation is shown in Fig. 5.4.

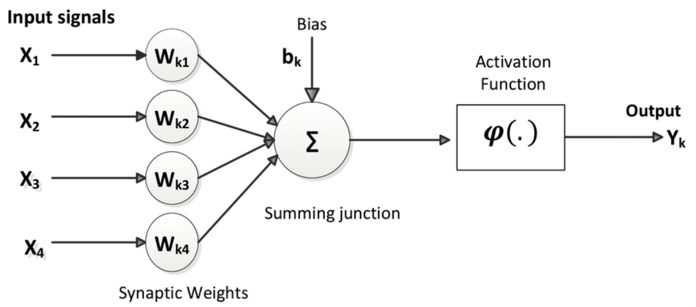


Figure 5.4: McCulloch-Pitts' neuron schematic representation.

One can see that exists a precise and natural correspondence between a biological neuron and this artificial/-computational model of it. The input signals are called **presynaptic inputs** and correspond to the signals taken in input by the dendrites of the neuron. The summing junction corresponds to the soma of the neuron where the information is processed, while the activation function mimics the activation threshold in the action-potential mechanism. Finally the set of weights $\boldsymbol{w}_k$, mimics the fact that different synapses have different "strength"; in particular if $w_{ki} > 0$ the synapse is called excitatory, while if $w_{ki} < 0$ the synapse is called inhibitory. Functionally, synaptic strength is defined as the average amount of current or voltage excursion produced in the postsynaptic neuron by an action potential in the presynaptic neuron [5]. Thanks to the heuristic Dale's law, if a neuron is excitatory, then all its synapses are excitatory and vice-versa.

Suppose now to have N inputs and let's denote the input as $\boldsymbol{v}(t) = \{v_1(t), \ldots, v_N(t)\}$; let's denote also $I_s(t)$ the total **presynaptic** current and $i_b(t)$ the input current coming from one specific neuron b. In general one can state that

$$i_b(t) = w_b k_b(t) \quad \text{where} \int_0^{\infty} k_b(t)dt = 1 \tag{5.3}$$

where $k_b(t)$ is called synaptic kernel and describes the synaptic conductance of the link. The total current $I_s(t)$ is then defined as

$$I_s(t) = \sum_{b=1}^{N} i_b = \sum_{b=1}^{N} w_b \underbrace{\int_{-\infty}^{t} k_s(t-\tau)\rho_b(\tau)d\tau}_{=k_b(t)} = \sum_{b=1}^{N} w_b \int_{-\infty}^{t} k_s(t-\tau)v_b(\tau)d\tau \tag{5.4}$$

where$\rho_b(t)$ is the neural response of the presynaptic neuron b, that under some conditions is equivalent to the firing rate of the neuron $v_b(t)$ (that is actually what we mean with the word input). k_s instead is a kernel function supposed to be

$$k_s(t) = \frac{1}{\tau_r} e^{-\frac{t}{\tau_r}}$$

where τ_r is a time constant of the neuron; it describes the time course of the synaptic current in response to a presynaptic spike arriving at time $t = 0$. Then, after some calculations, one can obtain from Eq. (5.4) the following result

$$\tau_r \frac{dI_s}{dt} = \boldsymbol{w} \cdot \boldsymbol{v}(t) - I_s(t). \tag{5.5}$$

At stationarity then $I_s^* = \boldsymbol{w} \cdot \boldsymbol{v}^*$. Let's call $\boldsymbol{\psi}$ the activation function mimicking the action-potential threshold, then the **postsynaptic** firing-rate of the neuron at stationarity is $h_\infty = \psi(I_s^*)$. Finally, if one take in consideration the capacitance effect of the neuron, the final equations of the firing rate model are

$$\begin{cases} \tau_r \frac{dI_s}{dt} = \boldsymbol{w} \cdot \boldsymbol{v}(t) - I_s(t) \\ \tau_h \frac{dh}{dt} = \psi(I_s(t)) - h(t). \end{cases} \tag{5.6}$$

From the biological point of view, this model is a kind of resource-consumer model, very well known in ecology, where the output h is the specie abundance, while the inputs $\boldsymbol{v}$ are the resources. So in the quasi-stationary approximation, one have $\tau_h \gg \tau_r$,meaning that the resources have a faster dynamic then the consumer. This inequality will be also found later.

5.3 Plasticity

The McCulloch-Pitts' model is a very useful computational model that is very well suited for a simple description of a biological neuron. However up to now, the weights associated to the synapses of the neuron are supposed to be fixed, quenched. The real brain however is plastic, a human indeed can "rewire" it with habits and new experiences. The simplest form of plasticity is that neural pathways that are used a lot strengthen; this is inspired by the famous Hebb's principle [23].

Principle 5.3.1. (Hebb's Principle)

When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased.

From this principle comes out the so called Hebb's conjecture, that is a mathematical formulation of the previous statement.

$$\tau_w \frac{d\boldsymbol{w}}{dt} = h\boldsymbol{v} = \psi(\boldsymbol{w} \cdot \boldsymbol{v})\boldsymbol{v} \tag{5.7}$$

This means that if a neuron has an output $h > 0$ and there is a input $v_b > 0$ then the connection w_b should be increased; notice also that Hebb's rule has introduced a feedback loop over $\boldsymbol{w}$. Although this is the most famous form of plasticity, there are several problems associated to this formulation, for example w_b in such a way can grow unbounded. There are indeed two missing ingredients in this naive formulation.

- **Competition**: there is a finite amount of resources to grow the synapses. This imply that some connections have to be cut, while other can grow.
- **Synapses weakening**: known also as anti-Hebbian rule, if a synapse is not useful, its strength should be decreased and eventually removed through competition.

Since the Hebb's rule, many other plasticity rules have been formulated; one indeed can introduce a threshold θ_h determining the level of postsynaptic activity needed in order to increase the synaptic connection.

$$\tau_w \frac{d\boldsymbol{w}}{dt} = (h - \theta_h)\boldsymbol{v} \tag{5.8}$$

Typically $\theta_h = \langle h \rangle$ is taken as the average postsynaptic activity; if $h - \theta_h > 0$ there is **long term potentiation** (LTP) where the value of $\boldsymbol{w}$ increase, otherwise there is long term depression (LTD). Another version of plasticity rule is that one introducing competition between the presynapses, through the so called **subtractive normalization rule**.

$$\tau_w \frac{d\boldsymbol{w}}{dt} = h\boldsymbol{v} - \frac{1 \cdot \boldsymbol{w}}{N} 1 \tag{5.9}$$

Note the presence of the term $1 \cdot \boldsymbol{w} = \sum_b w_b = const$, meaning that resources available for the strengthening of the synapses are limited.

5.4 Back-Propagation

Once more artificial neurons are linked together, one can create the so called artificial neural networks. The reason behind the effectiveness of these architectures in solving every day problems is all about the learning rule used to train them, i.e. the back-propagation algorithm. Since the pioneering work of Hinton et al. in [24], BP was lately implemented in TensorFlow and PyTorch in order to train DNN effectively. The reason behind why this algorithm is so effective in training DNN, is because the only objective is the minimization of the loss function. The minimization is indeed done exclusively, by changing the weights of the network initialized at random at the beginning. Even though the BP algorithm is the most known and used in training ANN, one question could arise: if an ANN is biologically inspired by the brain, is back-propagation biologically inspired by some learning processes happening in the brain? The answer that many neuroscientists support is no and moreover they argue that BP cannot happen in the brain. There are several reasons behind this claim and here below there are the most commonly ones.

- As described in the previous section, neuronal plasticity, i.e. the ability of changing the strength of the synapses, happens always between two neurons. This means that changing the strength of the synapses is a **local procedure**. BP instead is not a local algorithm, because is based on the chain rule, meaning that in order to change a connection w_{ij} it requires the knowledge of all the previous connections up to the output layer. A complete description of the BP algorithm can be found everywhere on the literature; here instead it is briefly reported a justification of the previous statement. The starting idea of BP is indeed the chain rule; let's enumerate the layers of an ANN with the index k, where $k = 0$ is in the input layer, while $k = N_{layers}$ is the output layer.

  $$\frac{\partial \mathcal{L}}{\partial w^k_{ij}} = \underbrace{\frac{\partial \mathcal{L}}{\partial a^k_j}}_{(*)} \underbrace{\frac{\partial a^k_j}{\partial w^k_{ij}}}_{(\#)} \quad \text{where } a^k_i = \sum_{j=0}^{r_{k-1}} w^k_{ji} o^{k-1}_j = \sum_{j=0}^{r_{k-1}} w^k_{ji} \psi(a^{k-1}_j)$$

  Note that with reference to a layer k, o^{k-1}_j is the output of neuron j in the $k-1$ layer, that is the input for a neuron in layer k. a^k_i instead is the dot product between input and matrix of weights of layer k; let's call ψ a general activation function that changes based on the layer. The second term can be written as

  $$(\#)\ \frac{\partial a^k_j}{\partial w^k_{ij}} = \frac{\partial}{\partial w^k_{ij}} \left(\sum_{l=0}^{r^{k-1}} w^k_{lj} o^{k-1}_l \right) = o^{k-1}_i$$

  while the first, after some calculations, can be written as

  $$(*)\ \delta^k_j \equiv \frac{\partial \mathcal{L}}{\partial a^k_j} = \psi'(a^k_j) \sum_{l=1}^{r^{k+1}} w^{k+1}_{jl} \delta^{k+1}_l .$$

  In conclusion one can easily see that, if one want to calculate the error derivative respect w_{ij} in layer k, he needs the knowledge of all the weights w_{jl} in layer $k+1$.

- Another observation moved against supervised learning in general, is that it is not how humans learn. For example, animals require lot of sensory experience to tune the early visual system into an adult system. This experience is believed to be predominantly observational, with few or no labels, so that there is no explicit task to solve [6]. This type of learning is said to be unsupervised. So if in the brain there is no obvious source of labels, why the learning paradigm should use labels in its loss function? This is an observation against the learning paradigm but indirectly against BP, because BP can be used only if one know the target values (also called as ground-truth).

- Moreover, since BP is based on the propagation of the errors, these errors $e_{ij} \in \mathbb{R}$ are real values. However as seen in the neurobiology section, neurons propagate all-or-none spikes and indeed the neural response, described by Eq. (5.1), is given by a sum of δ-train. This suggest that brain does not propagate real value signals but spike-train. It is important to not confuse the neural spikes with the voltage of the action-potential, that instead could be a continuous variable.
- Artificial neurons moreover send two different types of signals: in the forward pass the activity, in the backward pass the error derivative. This is indeed anomalous from the biological point of view.
- Another interesting point is that in BP artificial neurons allow forward and backward pass. In reality neurons do not have a symmetrical reciprocal connections. As studied before, the signal moves from the soma to the axon but not vice-versa, excluded some particular cases (e.g. endocannabinoids).

The problem is that, BP works extremely well for really tough practical problems, as one can see training large DNN. In conclusion, up to now, a sure answer to the question "Is BP implemented in brain?", is not yet found, even though there is a strong evidence of why BP could not be performed in the brain. Consequently, the literature and the current research is looking toward algorithms that are in some way more biologically plausible. In general it is quite difficult to create algorithms that have a solution for every point mentioned above, this is why scientists are focused in developing algorithms that takes in consideration at least some of them.

5.5 Krotov and Hopfield's algorithm

A fundamental starting point of this thesis is the work done by Krotov and Hopfield in [6]. This paper published in PNAS 2019, is an example of how the current research is focused in finding and developing algorithms, a little bit more biological w.r.t standard algorithms trained with BP. Their observation is quite simple: suppose to have a 3-layers ANN as shown in Fig. 5.5, here BP is not a local algorithm, because changing the connections blue-yellow requires knowing the connections blue-green. So they decided to split the network in two parts: the first yellow-blue will be trained in a biological-way, while the second one, blue-green, using BP. In this way, BP is a local algorithm, because applied only between two consequent layers.

The first part, as I just said, is trained in a biological way, meaning that the learning dynamics of the weights of the network, is a neuronal dynamics inspired by neurobiology principles. This means that the authors have engineered a specific plasticity rule in order to train the network. Moreover, since the learning dynamics is based on the firing-rate model and on plasticity, there is no loss function; consequently there is no space for labels, meaning that the first matrix of weight is trained in an unsupervised way.

5.5.1 Algorithm Description

It has been known, since the pioneering work of Hubel and Wiesel, that many neurons in the visual cortex are tuned to detect certain elementary patterns of activity in the visual stimulus. A relevant part of the neurobiology

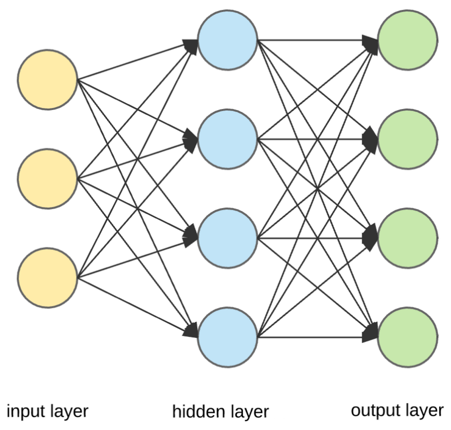


Figure 5.5: Example of a 3-layers ANN.

literature is dedicated to the study of biologically plausible mechanisms for development of orientation selectivity in the visual cortex; in particular the work of Bienenstock, Cooper and Munro (BCM) [25] is particularly important. The idea of BCM theory is that for a random sequence of input patterns a synapse is learning to differentiate between those stimuli that excite the postsynaptic neuron strongly and those stimuli that excite that neuron weakly. Learned BCM feature detectors cannot, however, be simply used as the first layer of an ANN so that the entire network is competitive to a network of the same size trained with BP end-to-end [6]. In other words, BCM is a theory of the development of the pattern selectivity of a single cell, i.e. there is a temporal competition between patterns seeking to drive a single neuron. This competition is controlled by the dynamics of an adjustable threshold parameter.

In the algorithm proposed by K-H instead, the neurons compete with each other for patterns: the competition is between neurons, not between patterns. When one neuron becomes tuned to some pattern of inputs, the within-layer lateral inhibition keeps other neurons far from becoming selective to that same pattern. As first equation, let's describe the example-presentation dynamic, i.e. how the postsynaptic activation change based on the input given to the network; this, by the way, looks similar to the h-equation of the firing rate model described in Eq. (5.6).

$$\tau_r \frac{dh_\mu}{dt} = \psi(I_s^\mu(t)) - h_\mu(t) = I_s^\mu(t) - w_{inh} \sum_{\nu \neq \mu} \varphi(h_\nu) - h_\mu(t) \tag{5.10}$$

Here φ is an activation function, generally chosen as $\varphi(h) = \text{ReLU}(h) = \max\{0, h\}$, while ψ in this case is a linear function of I_s^μ plus a subtracting term. The parameter w_{inh} is describing the strength of the within-layer lateral inhibition between output neurons and its value is set so that in the final state, only a small fraction of hidden units have positive activity. Finally notice that lateral inhibition between neurons within a layer makes the network not strictly feedforward. A schematic representation of what done so far is shown in Fig. 5.6.

Once the hidden neurons reach a steady state solution $h_\mu(t) = h_\mu^*$, then the learning phase can start. So in the first phase an input image (here a MNIST image) is shown to the network, while in the second phase the weights connections of the network are modified. The equation describing the engineered plasticity rule found by the

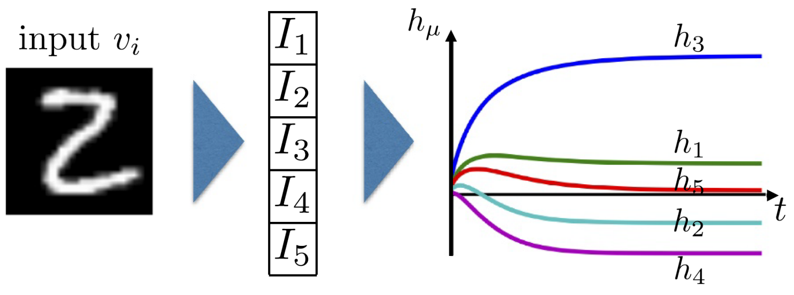


Figure 5.6: Inputs v_i are converted to a set of input currents I_s^μ. These currents define the dynamics that lead to the steady-state activation of the hidden units. Image 2 and caption taken from [6].

authors is the following,

$$\tau_w \frac{dw_{\mu i}}{dt} = \Phi(h_\mu)\left[R^p v_i - \left(\sum_{k=1}^{N} w_{\mu k} v_k\right) w_{\mu i}\right] = \underbrace{\Phi(h_\mu) R^p v_i}_{(*)} - \underbrace{\Phi(h_\mu)\left(\sum_{k=1}^{N} w_{\mu k} v_k\right) w_{\mu i}}_{(\#)} \tag{5.11}$$

where $\Phi(h)$ is a non differentiable activation function defined as

$$\Phi(h) = \begin{cases} 0 & \text{if } h < 0 \\ -\Delta & \text{if } 0 \leq h < h_* \\ 1 & \text{otherwise.} \end{cases} \tag{5.12}$$

The first term $(*)$ of the plasticity rule is the product between a function of the postsynaptic activity $\Phi(h_\mu)$ and the presynaptic activity of neuron v_i; thus, it is an example of Hebbian-like plasticity (remember Eq. (5.7)). In particular this function Φ has both positive and negative values, resulting in both Hebbian and anti-Hebbian learning (i.e. synapses weakening). The second term $(\#)$ instead ensures that the vector of weights between an hidden neuron and the input neurons, converges to a vector of norm equal to R (usually set equal to 1), i.e. it describes a **homeostatic constraint**. This plasticity rule is an extension of the famous Oja's rule [26], here reported for one neuron (the second index μ is missing)

$$\Delta w_i = \gamma \eta(t)\left[v_i(t) - \eta(t) w_i(t)\right] = \gamma\left(\sum_{k=1}^{N} w_k v_k\right)\left[v_i - \left(\sum_{k=1}^{N} w_k v_k\right) w_i\right] \tag{5.13}$$

where γ is called plasticity coefficient. Taking the inverse of γ and going to the continuum limit one retrieve Eq. (5.11); notice that in the Oja's equation the activation functions are linear.

$$\Phi(h) = \Phi\big(\psi(\boldsymbol{w} \cdot \boldsymbol{v})\big) = \boldsymbol{w} \cdot \boldsymbol{v}$$

Once the first matrix of weights have been learned, one can fix these values and learn the second matrix of weights using back-propagation.

5.5.2 Results

The MNIST data set, that contains 70000 images, was randomly split by the authors into a 50000 examples training set, 10000 examples validation set, that was used for tuning the hyper-parameters and 10000 examples test set. The performance of this biologically trained network was compared with the performance of a feedforward network of the same size $(784 - 2000 - 10)$ trained end-to-end using the Adam optimizer starting from random weights. The results are shown in Fig. 5.7.

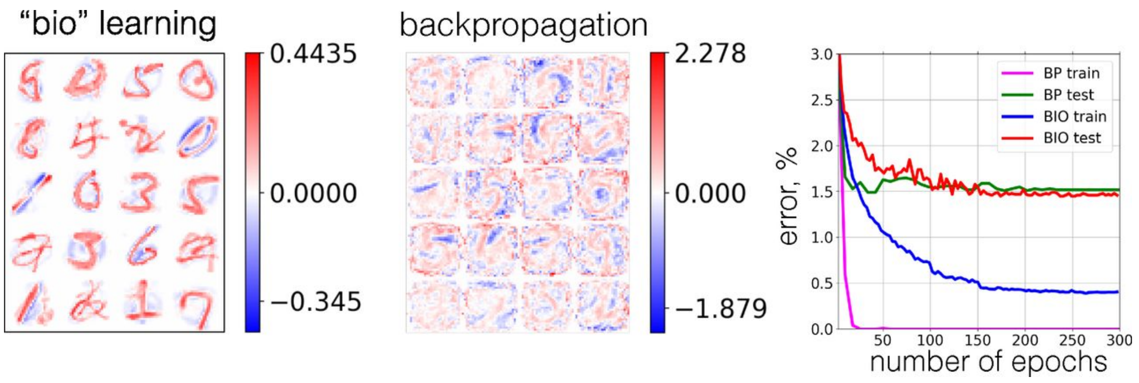


Figure 5.7: (Left) The weights learned by the biological network. Twenty randomly chosen feature detectors of 2,000 are shown. (Center) The weights learned by the network trained end-to-end with BP. Twenty randomly chosen feature detectors of 2,000 are shown. (Right) Error rate on the training and test sets as training progresses for both networks. Image 3 and caption taken from [6].

Let's give a brief discussion about the results.

- First of all, it is impressive how the feature maps obtained from the BNN are meaningful, encoding digits in each of them. The feature maps obtained from BP instead are more noisy and not meaningful at all for a human-being.
- However, even if the feature maps are quite different, the test accuracy in both the networks are quite the same. This is not a coincidence at all, remember one of the theories behind adversarial attacks: neural networks can learn both robust and non-robust features and yet, the classifier can correctly pursue the classification task.
- Note also how BP training is much faster w.r.t the bio-training with a final training accuracy worse respect the BP counter-part; yet the test accuracy is similar as said before, with a similar convergence time.

6 Proposed Algorithm

Now that all the ingredients have been introduced, it is time to build an algorithm that exhibits adversarial defence. The fundamental idea is given by looking at the feature maps obtained from the bio-learning in Fig. 5.7. If a network learns what for a human are the correct things to learn, how is it possible that this architecture is vulnerable to adversarial attacks? Indeed humans are not vulnerable to adversarial attacks. In some sense if the first layer of the ANN, that by the way is the most important for the classification task, has feature maps that humans can understand, then this first layer is transparent in the sense of decomposability. This means that each feature map of the model admits an intuitive explanation, in this case each feature map represents a digit prototype. Note also that in this way humans develop trust with respect to the AI algorithm, where the contractual trust is based on the transparency contract. In order to investigate about the adversarial robustness of the Krotov and Hopfield's algorithm, let's implement it first and then check how it behaves. This by the way will help the user to stipulate another contract, in particular technical robustness and safety, based on the study of adversarial robustness. As said before, a broader trust on the AI algorithm is based on more then one contract.

6.1 Implementation of Krotov and Hopfield's algorithm

In this section it will be shown the implementation of the bio-algorithm proposed by K-H. In the paper they discussed two versions of it: one naive and slow and another more advanced and faster. Here will be discussed only the naive version, for the other please see [6]. Remember that the algorithm proposed by the authors is divided in two phases since the chosen ANN has two matrices of weights. In the first phase the matrix of weights between input and hidden layers (called here W^1) is learned with a biological inspired algorithm, while in the second phase the second matrix of weights (W^2) between hidden and output layers is learned with back-propagation. The structure of the ANN is composed by $N_i = 784$ input neurons, $N_h = 361$ hidden neurons and $N_o = 10$ output neurons. The first phase is constituted by an iteration of two steps: presentation dynamics and learning dynamics. For the first step of the first phase, the code is implementing Eq. (5.10). A simple Euler method was used in order to solve the ODE. Since solving the ODE for every image is computational demanding, using a fast ODE-solver is fundamental; note that no other existing method is faster compared to the Euler's one. In the following equations, it is shown how to adapt Eq. (5.10) in order to implement it in python.

$$h_\mu(t+1) = h_\mu(t) + \frac{dh_\mu}{dt}dt = h_\mu(t) + \left[I_{s,\mu}(t) - w_{inh}\sum_{\nu\neq\mu}\varphi(h_\nu) - h_\mu(t)\right]\frac{dt}{\tau_r}$$

$$\boldsymbol{h}(t+1) = \boldsymbol{h}(t)\left(1-\frac{dt}{\tau_r}\right) + \left[\boldsymbol{I}_s(t) - w_{inh}\underbrace{\mathbb{I}_m \cdot \max\{\boldsymbol{h},0\}}_{\equiv S}\right]\frac{dt}{\tau_r}$$

$\mathbb{I}_m$ is a useful matrix that implements the operation $\sum_{\nu\neq\mu}$.

$$S = \underbrace{\begin{bmatrix} 0 & 1 & \dots & 1 \\ 1 & 0 & 1 & \vdots \\ \vdots & 1 & \ddots & 1 \\ 1 & \dots & 1 & 0 \end{bmatrix}}_{=\mathbb{I}_m} \begin{bmatrix} \varphi(h_1) \\ \vdots \\ \varphi(h_n) \end{bmatrix} = \begin{bmatrix} \varphi(h_2)+\varphi(h_3)+\dots+\varphi(h_n) \\ \varphi(h_1)+\varphi(h_3)+\dots+\varphi(h_n) \\ \vdots \\ \varphi(h_1)+\dots+\varphi(h_{n-1}) \end{bmatrix}$$

The code solving the ODE of the first part is reported below.

```
tau_r = 0.01
winh = 0.9
neurons_hidden = 361
neurons_input = 748

tmax1, tmax2 = 0.05, 0.3
dt1, dt2 = 0.0001, 0.001
t1 = np.arange(0,tmax1,dt1)/tau_r
t2 = np.arange(tmax1,tmax2,dt2)/tau_r
lt1, lt2 = len(t1),len(t2)
tt = np.concatenate([t1, t2])
lt = len(tt)
Im = np.ones(neurons_hidden) - np.identity(neurons_hidden)

def hsteady(h_steady,Is):
  h = np.zeros((neurons_hidden,lt))
  h[:,0] = h_steady
  dt_tau1 = dt1/tau_r
  dt_tau2 = dt2/tau_r
```

```
for i in range(1,lt1):
  S = np.matmul(Im, np.maximum(h[:,i-1],0))
  h[:,i] = h[:,i-1]*(1-dt_tau1) + (Is - winh*S)*dt_tau1
for i in range(lt1,lt2+lt1):
  S = np.matmul(Im, np.maximum(h[:,i-1],0))
  h[:,i] = h[:,i-1]*(1-dt_tau2) + (Is - winh*S)*dt_tau2

return h[:,-1], h
```

Note that here the real physical parameter is w_{inh}, all the other parameters like t_{max} and dt are important in order to obtain a good solution in a reasonable amount of computational time. In particular the authors state that w_{inh} should be chosen such that at stationarity only a small fraction of neurons have a positive activation value; for example with $w_{inh} = 0.9$, there is 1.94% of active neurons. Also the value of τ_r is not important, the only requirement is that $\tau_r \ll \tau_w$, i.e. the input dynamic is much faster compared to the learning dynamic. Surprisingly this inequality is similar to that mentioned when the firing rate model was introduced. The input image is normalized between $[0, 1]$ and the weights between input and hidden neurons are extracted at random from a gaussian pdf $\mathcal{N}(0, 1)$. Note also there are two dt because at the beginning the simulation-steps have to be done carefully in order to avoid strange simulation artifacts, while in reaching stationarity the time-steps could be longer; this helps speeding up the computation. Indeed the price to pay in using the Euler method, that is very fast and simple, is to use a small dt since the precision of the method goes as $O(dt^2)$. The dynamic of the postsynaptic neurons and their steady-state activation are shown in Fig. 6.1.

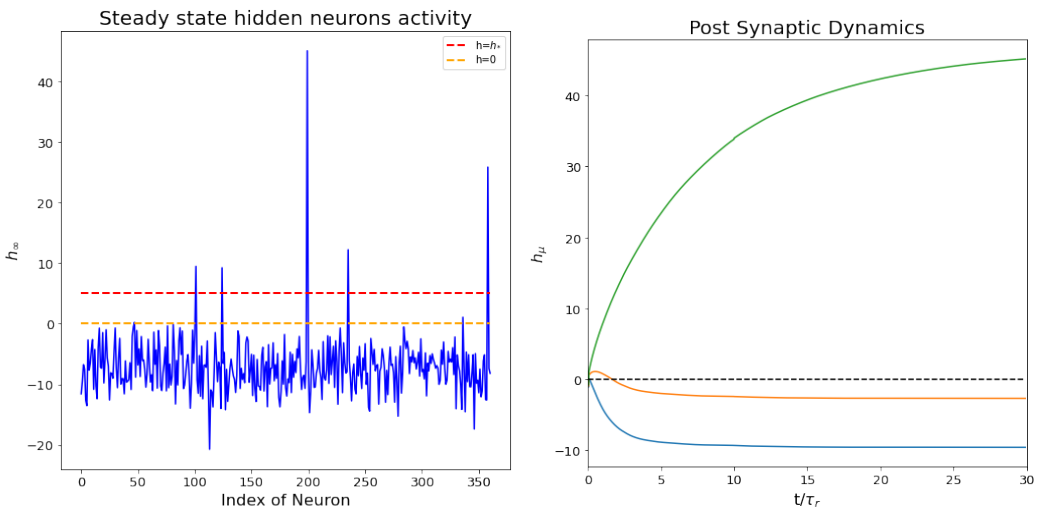


Figure 6.1: (Left) Steady state activation h_∞ of the hidden neurons. The dashed orange is the value $h_\infty = 0$, while in red the value $h_\infty = h^*$, the free parameter of the activation function. (Right) The presentation dynamic for three random neurons; the absolute time t was normalized w.r.t the characteristic time scale of the presentation dynamic τ_r.

Summing up, in the first step one give in input to the network an image and wait until the hidden neurons reach a steady state h_∞; this is what the authors mean with input presentation dynamics. Let's now move on to the second step; the activation function in equ. 5.12 is implemented as follow.

```
def g(h_steady, h_star, delta):
  gh = np.ones((len(h_steady),1))*delta
  gh[h_steady >= h_star] = 1
  gh[h_steady < 0] = 0
  return gh

delta = -0.01
```

Again a simple Euler method is used to solve the ODE, where notice that here the physical parameters are Δ, h^* and τ_w. Again, the code is implementing EQ. (5.11), so some arrangements are needed in order to implement it in python.

$$
\begin{aligned}
W^1(t+1) &= W^1(t) + \frac{dW^1}{dt}dt = W^1(t) + \Phi(\boldsymbol{h})\boldsymbol{v}^T\frac{dt}{\tau_w} - \Phi(\boldsymbol{h})\left[\left(\sum_{k=1}^{N} w^1_{\mu k} v_k\right) w^1_{\mu i}\right]\frac{dt}{\tau_w} \\
&= W^1(t) + \Phi(\boldsymbol{h})\boldsymbol{v}^T\frac{dt}{\tau_w} - \Phi(\boldsymbol{h})\left[\underbrace{W^1\boldsymbol{v}}_{\equiv \boldsymbol{I}_w} * w^1_{\mu i}\right]\frac{dt}{\tau_w} = W^1(t) + \Phi(\boldsymbol{h})\boldsymbol{v}^T\frac{dt}{\tau_w} - \underbrace{[\Phi(\boldsymbol{h}) \odot \boldsymbol{I}_w]\mathbb{I}^T}_{\equiv T} \odot W^1\frac{dt}{\tau_w}
\end{aligned}
$$

In the equation above W^1 is the matrix of weights whose elements are $w^1_{\mu i}$, while $\odot$ indicates per-element multiplication. All the steps above are done because the ODE in Eq. (5.11) is formulated by the authors in terms of single weight $w_{\mu i}$, but from the coding point of view is good practice using vectorial calculus. Note that i is an index running over the input neurons, while μ over the hidden ones; consequently T is a $n \times n$ matrix, while W^1 is a $n \times m$ matrix, where $n = 361$ and $m = 784$, i.e. respectively the number of hidden and input neurons. The use of $\odot$ is a useful mathematical notation since the use of per-element multiplication is naturally implemented in python (and in particular in NumPy) with the keyboard symbol $(*)$.

$$
T \odot W^1 = [\Phi(\boldsymbol{h}) \odot \boldsymbol{I}_w]\mathbb{I}^T \odot W^1 = \begin{bmatrix} \Phi(h_1) \\ \Phi(h_2) \\ \vdots \\ \Phi(h_\mu) \\ \vdots \\ \Phi(h_n) \end{bmatrix} \odot \begin{bmatrix} I_{w,1} \\ I_{w,2} \\ \vdots \\ I_{w,k} \\ \vdots \\ I_{w,n} \end{bmatrix} \mathbb{I}^T \odot W^1 = \begin{bmatrix} \Phi(h_1)I_{w,1} \\ \Phi(h_2)I_{w,2} \\ \vdots \\ \Phi(h_\mu)I_{w,k} \\ \vdots \\ \Phi(h_n)I_{w,n} \end{bmatrix} \begin{bmatrix} 1 & \dots & 1 \end{bmatrix} \odot W^1
$$

$$
= \underbrace{\begin{bmatrix} \Phi(h_2)I_{w,2} & \dots & \Phi(h_2)I_{w,2} \\ \vdots & \dots & \vdots \\ \Phi(h_\mu)I_{w,k} & \dots & \Phi(h_\mu) \\ \vdots & \dots & \vdots \\ \Phi(h_n)I_{w,n} & \dots & \Phi(h_n)I_{w,n} \end{bmatrix}}_{n \times n} \odot \begin{bmatrix} w^1_{11} & \dots & w^1_{1m} \\ \vdots & \dots & \vdots \\ w^1_{n1} & \dots & w^1_{nm} \end{bmatrix} = \begin{bmatrix} \Phi(h_1)I_{w,1}w^1_{11} & \dots & \Phi(h_1)I_{w,1}w^1_{1m} \\ \vdots & \dots & \vdots \\ \Phi(h_n)I_{w,n}w^1_{n1} & \dots & \Phi(h_n)I_{w,n}w^1_{nm} \end{bmatrix}
$$

The code solving the ODE of the second step is reported below.

```
tau_w = 5
tmax3 = 2.5
dt3 = 0.02
t3 = np.arange(0,tmax3,dt3)/tau_w
lt3 = len(t3)

def W_update_quick(ghp,vp,Wp):
  Wnew = np.zeros((neurons_hidden,neurons_input))
  dt_tau3 = dt3/tau_w
  ghv = (ghp*vp.T)*dt_tau3

  for j in range(1,lt3):
    Iw = np.matmul(Wp,vp)
    T = np.matmul(ghp*Iw,np.ones((1,neurons_input)))
    Wp = Wp + ghv - T*Wp*dt_tau3

  return Wp
```

The result of the learning dynamics is shown in Fig. 6.2.

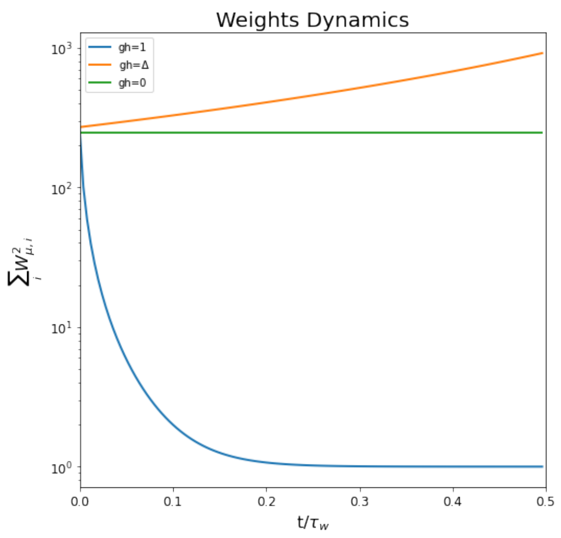


Figure 6.2: Convergence of the weights to a unit sphere for three random neurons, each one in a different learning regime. The absolute time t was normalized w.r.t the characteristic time scale of the learning dynamic τ_w.

In Fig. 6.2 three random hidden neurons are taken and all their connections to the input neurons are squared and then summed; again, μ is an index running over the hidden neurons, while i is running over the input neurons. So one can see that if a hidden neuron has a steady state activation $h_\infty > h^*$ (so $\Phi(h_\infty) = 1$) there is Hebbian learning (blue line) and the weights converged to a sphere of radius 1 (remember the radius R^p in Eq. (5.11) set to 1). If there is no learning (green line) the weights will stay constant and this happens in the case $h_\infty < 0$ (so $\Phi(h_\infty) = 0$). Finally, in the anti-Hebbian regime the sum of weights-squared tends to diverge. The divergence in the anti-Hebbian regime is not a bug and it is something expected by the authors themselves (for all the details see pp. 7725 of [6]). The only thing to mention is that this divergence is not a problem, because the probability to be in the anti-Hebbian regime is low. So this divergence does not grow too much and in the next step the weights or will stay constant or they will converge to the unit sphere. The stability depends also by the simulation time: as one can see from Fig. 6.2, the weights converge faster then divergence and t_{max} is set such that the convergence is reached for the blue line. In this way, the weights in the Hebbian regime converge, while those in the anti-Hebbian regime are not able to grow in an unbounded way. However setting a too high Δ or lowering too much the competition w_{inh} could make the algorithm fail.

Now the first and the second steps can iteratively applied for many samples. For a quick implementation let's start giving in input to the network 600 images, i.e. 1% of the entire test set (test plus validation set). Every digit class has 60 images inside the set of images presented to the network; in such a way the network should learn an equal representation for each digit, i.e. it is expected an equal number of feature maps. The code implementing the full training is reported below.

```
h_steady = np.zeros(neurons_output)
np.random.seed(2347)
sigma = 0.2
W_first = sigma*np.random.rand(neurons_output,neurons_input)
W = W_first

max_samples = 600
np.random.seed(2347)
sample_indexs = np.random.randint(0,dataset_size-1,max_samples)

t = time.time()
for sample_index in tqdm(sample_indexs):

  image = train_dataset[sample_index][0]
  v = torch.flatten(image).unsqueeze(dim=1).numpy()
  I = np.matmul(W,v).squeeze()
  h_steady = hsteady_quick(h_steady,I)

  gh = g(h_steady, h_star,delta)
  W = W_update_quick(gh,v,W)
```

The feature maps learned during this short training are shown in Fig. 6.3. Since 361 hidden neurons are considered in this architecture, there are 361 feature maps, each of them composed by 784 input neurons. Graphically these feature maps are arranged in pictures of 28×28 (784) pixels, in a grid of 19×19 (361) squares, for a better visualization.

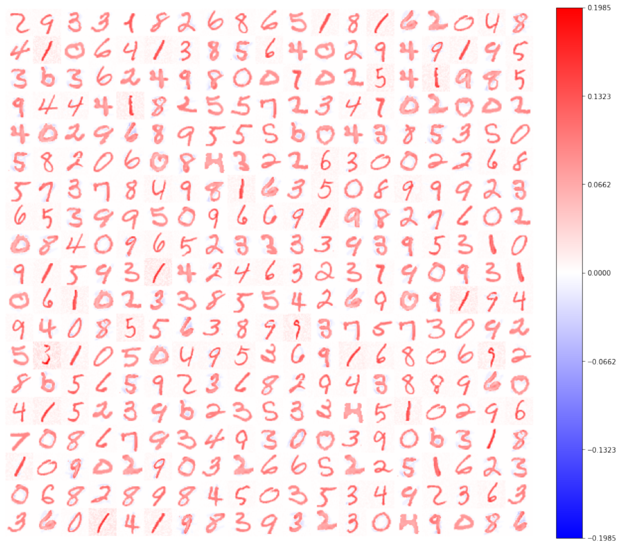
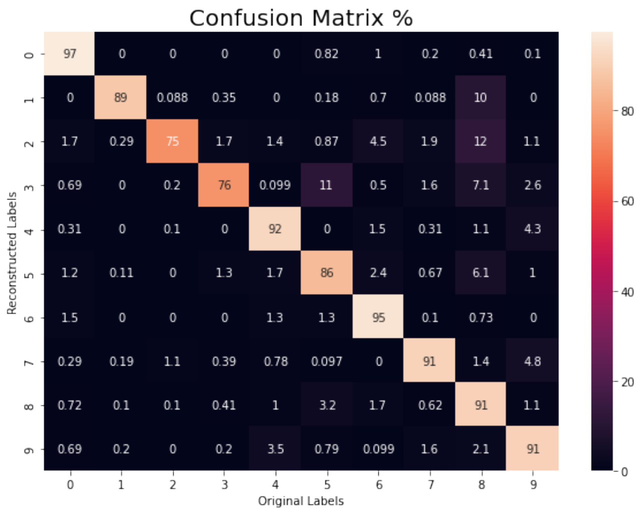


Figure 6.3: Feature maps learned during the bio-training. These feature maps, as anticipated before, can be understood by humans because are digit prototypes. Red pixels means positive weights, while blue pixels means negative ones.

From now on, all the ideas and results shown in the next pages come from personal work and cannot be found in the paper [6]. After the biological training, the unsupervised phase is finished, so freezing the first set of weights just found, one can build a PyTorch ANN with two layers and training the second one with the full MNIST dataset using BP. Using the library skorch [27], with a k-fold technique (5 folds) for a maximum of 15 epochs, using Adam as optimizer with a learning rate of 0.005, one can obtain without fine-tuning the parameters, a final test accuracy of 88.18%. The activation function is linear in the input layer, ReLU in the hidden layer, while for the last layer the soft-max function was used. The resulting confusion matrix is shown in Fig. 6.4.



	0	1	2	3	4	5	6	7	8	9
0	97	0	0	0	0	0.82	1	0.2	0.41	0.1
1	0	89	0.088	0.35	0	0.18	0.7	0.088	10	0
2	1.7	0.29	75	1.7	1.4	0.87	4.5	1.9	12	1.1
3	0.69	0	0.2	76	0.099	11	0.5	1.6	7.1	2.6
4	0.31	0	0.1	0	92	0	1.5	0.31	1.1	4.3
5	1.2	0.11	0	1.3	1.7	86	2.4	0.67	6.1	1
6	1.5	0	0	0	1.3	1.3	95	0.1	0.73	0
7	0.29	0.19	1.1	0.39	0.78	0.097	0	91	1.4	4.8
8	0.72	0.1	0.1	0.41	1	3.2	1.7	0.62	91	1.1
9	0.69	0.2	0	0.2	3.5	0.79	0.099	1.6	2.1	91

Figure 6.4: Confusion matrix, obtained from the evaluation of the network to the test set.

6.1.1 Understanding the Parameters

Let's see now what happen if the hyper-parameters of the bio-algorithm change, starting from w_{inh}. This part is not mentioned in the paper, but it is necessary in order to fully grasp the logic behind.

Remember that w_{inh} regulates the strength of the lateral competition between hidden neurons. Neurons indeed are competing with each other for being active to a given input pattern, so if w_{inh} is large, the competition is so strong that if one neuron becomes more active respect other, than all the other neurons are suppressed. This means that their steady state activation h_∞ will be less then zero.

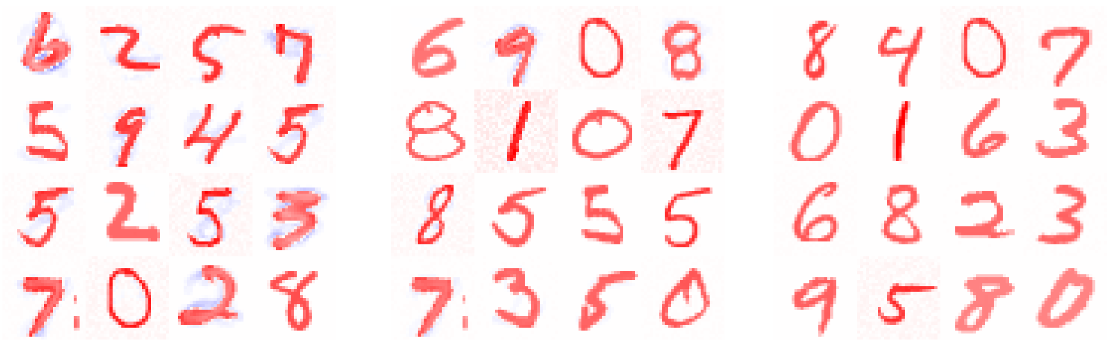


Figure 6.5: Some feature maps obtained setting different w_{inh} keeping $\Delta = -0.01$ and $h^* = 3$. Left, $w_{inh} = 0.9$, center $w_{inh} = 1$, right $w_{inh} = 1.3$.

One can see in Fig. 6.5 that even though the digits presented, the order of the images and the random initialization of the weights are the same in all the three cases, the feature maps embedded are different. This means that the value of w_{inh} in relation with the random initialization can change the order of the feature maps embedded. Note also that if one increase w_{inh} the negative weights (colored in blue) start to disappear. Moreover some of the feature maps shown above has some red and white random pixels behind the digit; obviously this is a unwanted situation, because they contribute to the activation of the neuron. These pixels are random weights that have been initialized at the beginning and that the learning dynamics was not able to change. A simple solution is just to give a smaller random initialization, with variance less then one (e.g. $\sigma = 0.2$) and give more time to the learning and presentation dynamics.

Let's now see the impact of the Δ parameter; Δ and h^* are both important in determining the impact of the anti-Hebbian regime. Remember that Δ is the value of the activation function Φ of the hidden neuron in the anti-Hebbian regime. So if an input and hidden neurons are not both active, then grater is Δ more their connection will be decreased. Indeed as one can see in Fig. 6.6 increasing Δ gives broader regions with a darker blue (for a comparison between red and blues weights see 5.7). Increasing h^* instead means increasing the region where the anti-Hebbian regime is active, meaning a major presence of negative weights. Indeed anti-Hebbian regime means negative value of the activation function, so even though anti-Hebbian regime does not imply negative weights, the presence of blue pixels surely implies the presence of neurons in anti-Hebbian regime. However it is not easy to understand the role of h^* and indeed for $h^* = 5$ there are less negative weights compared to the case $h^* = 3$. The results are shown in Fig. 6.7.

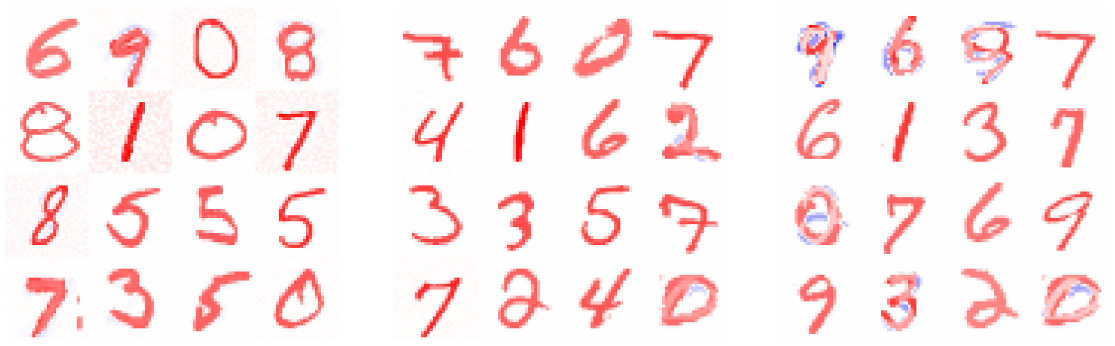


Figure 6.6: Some feature maps obtained setting different Δ keeping $w_{inh} = 1$ and $h^* = 3$. Left, $\Delta = -0.01$, center $\Delta = -0.05$, right $\Delta = -0.1$.

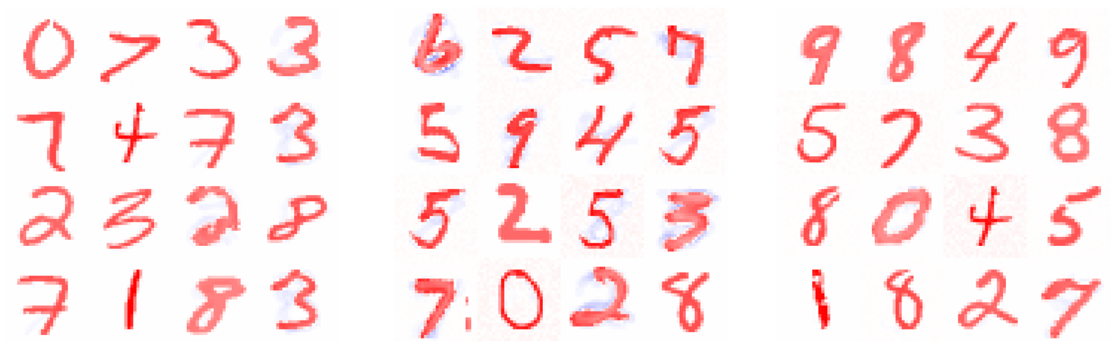


Figure 6.7: Some feature maps obtained setting different h^* keeping $w_{inh} = 0.9$ and $\Delta = -0.01$. Left, $h^* = 1$, center $h^* = 3$, right $h^* = 5$.

6.1.2 Adversarial Vulnerability

From the previous section it turns out that using this bizarre type of training, one can achieve easily 88% of accuracy on the test set, without spending too much time on fine-tuning the biological and the PyTorch's parameters. Now it is time to test the adversarial vulnerability of this architecture. The FGSM attack seen in chapter 4 can be easily implemented using PyTorch, where the code can be found in the section "Tutorials" in their web page [28]. If one test the proposed architecture against adversarial attacks, what turns out is that the network can be easily fooled by FGSM, as can be seen in Fig. 6.8. The test accuracy under attack drops linearly up to $\varepsilon = 0.15$ and then reaches some few percent ($\approx 3\%$) with $\varepsilon = 0.3$. The adversarial attacks can also be visualized respect the various ε; more in detailed, an example with $\varepsilon = 0.15$ can be seen in Fig. 6.9 .

Unfortunately, even though the ANN is embedding "human-level" features, i.e. digits prototypes in its feature maps, it is still vulnerable to adversarial attacks like the FGSM ones. If one look at the adversarial examples shown in Fig. 6.8 however, it could see that these new images are just the original one, plus some gray-noise around the digit. So it is quite annoying that such perturbations fool the network, because from a human point of view, these images are clearly digits that can be easily classified.

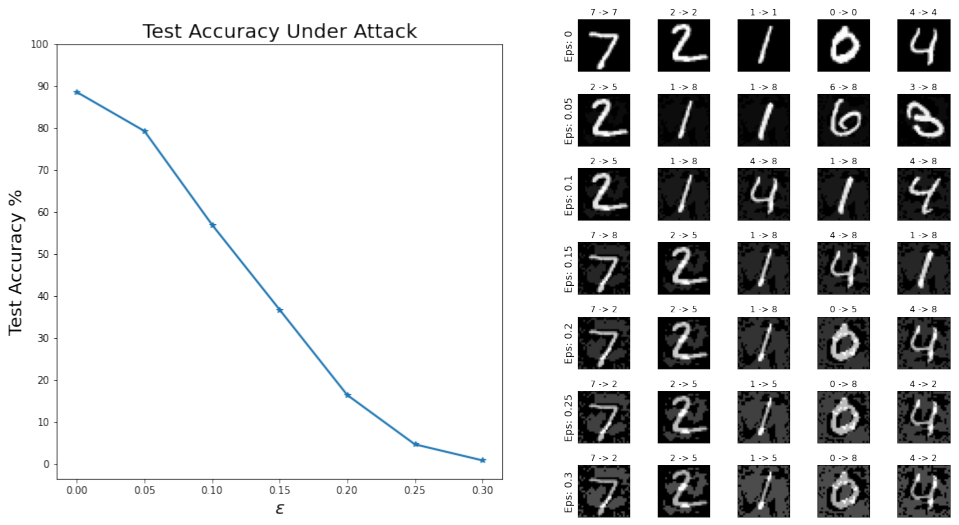


Figure 6.8: (Left) Test accuracy under attack, varying ε, the strength of the attack. (Right) Examples of adversarial attacks, for different ε. In each image's title, there is the original label, then an arrow and then the (wrong) label given by the network.

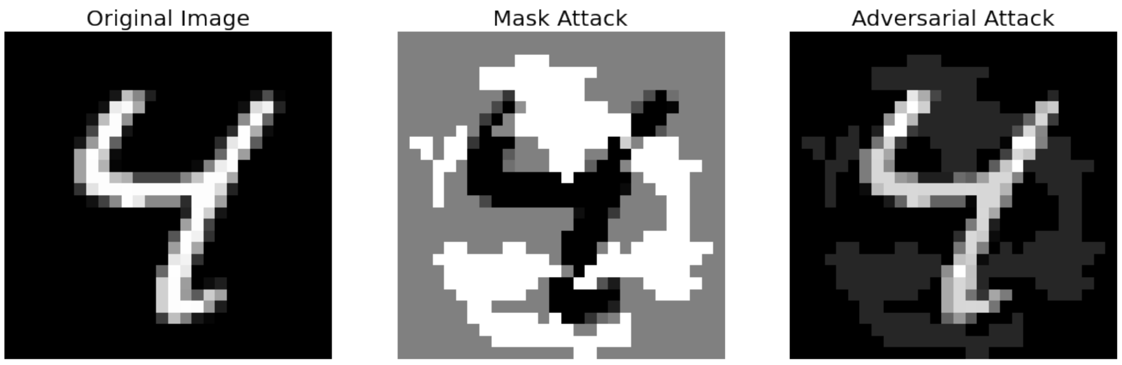


Figure 6.9: (Left) Original image taken from the MNIST dataset. (Center) The adversarial perturbation generated from the FGSM attack. (Right) Result of an additive attack, i.e. summing the original image and the mask attack together. Remember that black pixels have value 0, while white pixels have value 255.

6.2 Probabilistic Layer

As described above, an ANN built and trained in such a way does not displays adversarial vulnerability. Since the first layer has learned the right things, i.e. feature maps equal to digit prototypes, let's now focus on the second layer. Now the matrix of weights of the first layer is fixed and an image is given in input: what will happen to the activation of the hidden neurons? Let's start giving in input the digit 0; the input image is shown in Fig. 6.10. The image is then convoluted with all the feature maps and the result summed in order to give a single scalar number for each hidden neuron. Doing this one can evaluate the activation profile; the result is shown in Fig. 6.11.

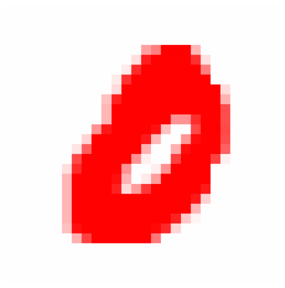


Figure 6.10: Example of digit 0 sampled from the test set and given in input to the network.

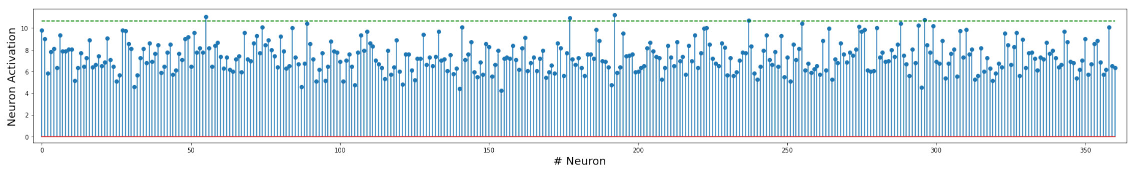


Figure 6.11: Hidden neurons activation. In red the baseline equal to the value 0, while the green dashed line corresponds to the 95% of the maximum value.

One can see that some neurons are more active then other, so is a natural question asking which feature maps these neurons are embedding in the first layer; the answer is shown in Fig. 6.12.

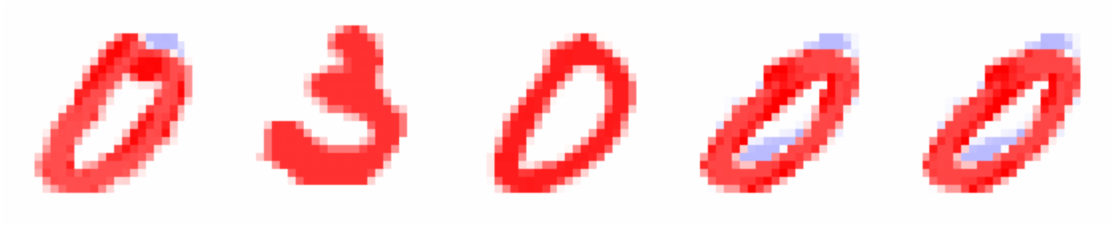


Figure 6.12: Feature maps of the five most active hidden neurons given the input digit shown in Fig. 6.10. These five neurons are those with an activation grater then 95% the maximum value.

As expected these neurons, which during training learned the digit prototype 0, will be more active when a 0 is presented in input. Looking at the feature maps, one can see that is present also a digit prototype of 3, obviously different from 0, meaning that this hidden neuron is very active to the input 0, even if it is a different digit compared to its feature map. This could happen when a particular input image has many pixels in common with the feature map; this fact is shown in Fig. 6.13 with a different example.

As underlined with the green line in Fig. 6.11, there exists an elite of few neurons super-active for that particular 0 presented in input. This elite of few neurons is chosen setting a very strict threshold in the activation value, keeping only the top 5% most active neurons. In some sense these neurons are ”super-specialized” in recognising this type of digit; this threshold will be denoted as $\lambda = 0.95$. From this observation, it is obvious that the final layer, used for the classification, should be based on this concept of ”super-active” neurons. The idea is formulated as follow: the first layer learns digit prototypes, the second layer assigns labels based on the plausibility of the most

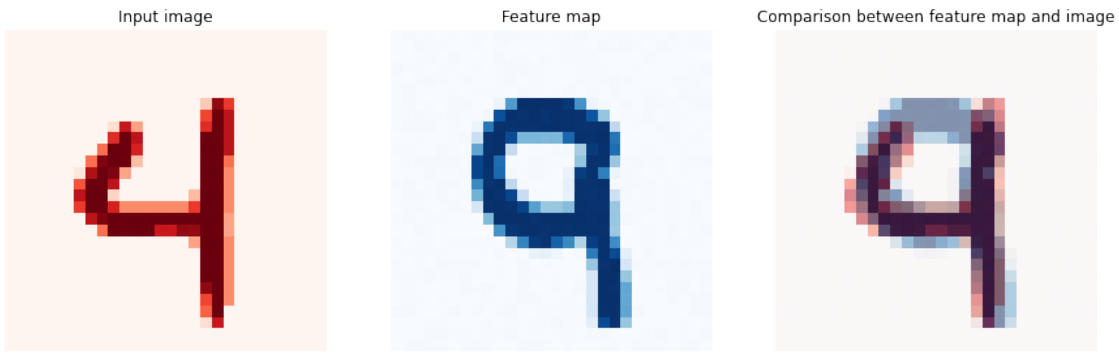


Figure 6.13: Another example where the feature map of the most active neuron is not equal to the input image; however these two digits share many pixels locations. The colors used are just for visualization purposes.

active hidden neurons. Let's assume for a moment, that each neuron is sensitive to only one type of digit, then one could just group together these neurons and they will be useful to detect the corresponding digit. The problem is that hand-written digits are very different one from the other, because of the calligraphy, so one neuron could not be sensitive to every picture of the same digit. Moreover as seen above with two examples, it is clear why this assumption does not hold.

Note that the learning algorithm of the second layer was BP in the original work by Krotov and Hopfield, but here substituted with the proposed algorithm i.e. the objective of this thesis. Therefore the learning algorithm proposed in this thesis for the second layer is constructed as follow:

1. Give in input to the network an image.
2. The first matrix of weights W^1 is fixed and learned using the bio-algorithm of K-H.
3. Perform matrix multiplication between input and matrix weight. Doing this one can calculate the activation values of the hidden neurons.
4. Check for the most active hidden neurons, let's say the top 5%.
5. The result of the activation functions of the hidden neurons, will give 1 for the most active ones 0 otherwise. This could be achieved using a steep sigmoid activation function in the hidden layer. Note that ideally one should use a Heaviside function that however is not differentiable.
6. The connections from these super-active neurons to the one corresponding to the label of the image, are increased by one while all the other are left the same. The connections are all initialized to zero at the beginning.
7. Repeat iteratively the steps above for all the images in the training set. In this phase the second matrix of weights W^2 is learned.
8. Once all images have been presented, the connections are normalized respect the number of times the hidden neurons were active. These weights are then interpretable as probabilities that a particular output neuron is active when a given hidden neuron is active. Obviously the probability that a hidden neuron is assigned to any label has to be one.

For clarity, let's give an illustrated example of the aforementioned algorithm (see Fig. 6.14).

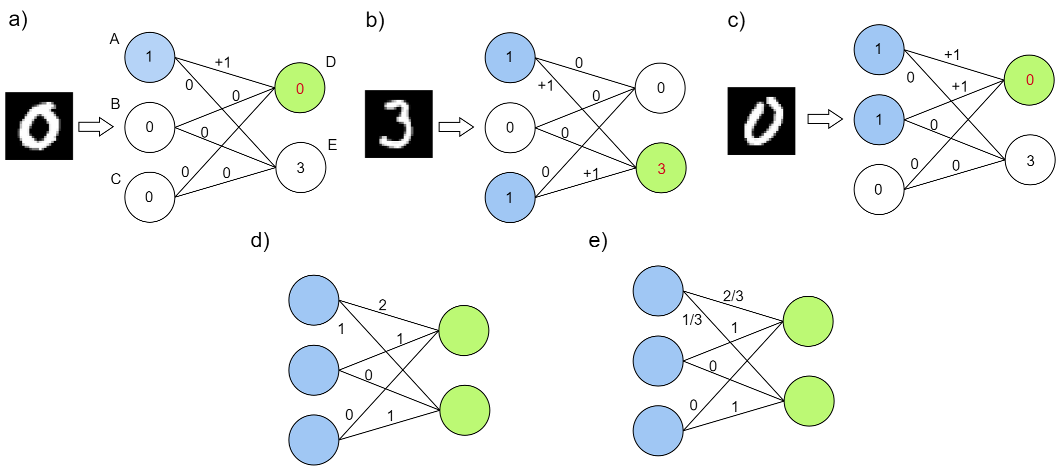


Figure 6.14: An illustrated example of the learning algorithm. In the networks shown here, with reference to Fig. 5.5, hidden neurons are colored in blue, while output neurons in green. In a)-b)-c) there are three examples of "super-active" neurons, in d) there is the set of counts for each connections and finally in e) there is the set of weights obtained from the normalization of the previous counts.

Let's call the three hidden neurons (A, B, C) and the output ones (D, E); at the beginning the weights connections are all initialized to zero. Then an image of a 0 is presented in input (a); here only neuron A is very active, so its connection shared with neuron D should be increased. A new image of a 3 is presented in input, so new neurons are super-active, in this case the A and C (b). Now one can use the entire training-set in order to train the second layer (c). At the end (d) one have the matrix of counts obtained from the previous steps. Once the matrix of counts is known it can be normalized such that the sum of the weights of the out-coming links of a hidden neuron is equal to one (e).

From a mathematical point of view, the strength connection between a hidden neuron A and a output neuron D is determined as follow

$$w_{AD} = \frac{\#\text{ times } A \text{ and } D \text{ are active}}{\#\text{ times } A \text{ is active}}. \tag{6.1}$$

This is one of the most important idea of the thesis: indeed one, after some manipulations, can show that this definition of weights has a probabilistic interpretation.

$$w_{AD} = \frac{\#\text{ times } A \text{ and } D \text{ are active}}{\#\text{ images presented in input}} \cdot \frac{\#\text{ images presented in input}}{\#\text{ times } A \text{ is active}} \equiv \frac{\mathbb{P}(A \cap D)}{\mathbb{P}(A)} = \mathbb{P}(D|A)$$

In point a) of Fig. 6.14 when A is active, the reference label is 0, i.e. should be active neuron D. Increasing the counter between A and D means increasing the joint probability of having A and D active i.e. $\mathbb{P}(A \cap D)$. Since the real interest is in estimating $\mathbb{P}(D)$ one can calculate it using the rules of probability as

$$\mathbb{P}(D) = \sum_i \mathbb{P}(D|i)\mathbb{P}(i) \quad \text{where } i \in \{A, B, \dots\} \text{ hidden neurons.}$$

The network then gives in output the probability for each label, meaning that no soft-max activation is needed in principle. Finally looking for the maximum of the various probabilities one can choose the final label. So the overall training of the network is done in the following way:

- Only 600 images from the 50000 images of the test set are presented to the first layer of the network. Here digit prototypes are learned via bio-learning, with the algorithm proposed by Krotov and Hopfield.
- Then all the 50000 images are presented in input, with the matrix of the first layer fixed. Here the connection weights of the second layer are learned with the algorithm proposed in this thesis.
- Finally the test set is presented in input to the network and the test accuracy of the network is evaluated. In particular there is no need of soft-max activation function, since the algorithm is able to calculate the probability of the resulting label. However the use of soft-max can enlarge small differences between probabilities and so it can increase few percent the test accuracy.

Finally, evaluating the test accuracy, one can easily achieve 86.38%; remember that with BP a similar accuracy was achieved (88.18%). The confusion matrix obtained is shown in Fig. 6.15.

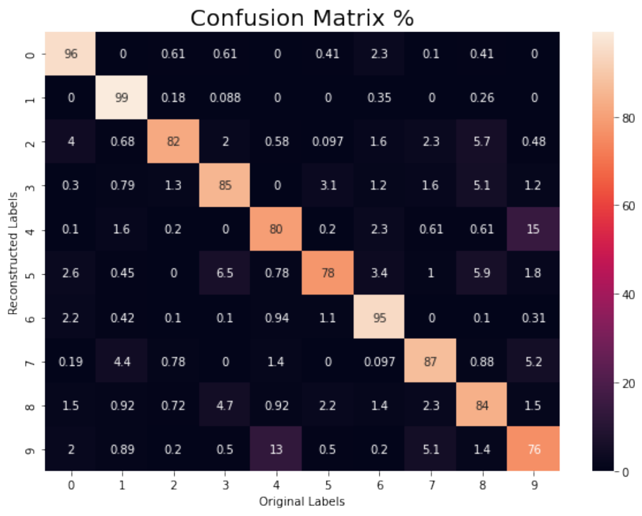


Figure 6.15: Confusion matrix, obtained from the evaluation of the network to the test set using the proposed learning algorithm.

So after the full training of the second layer, one can interpret the results and make a consistency check. Let's consider the neuron number 300; one can easily see that the vector of counts between that neuron and the output neurons has a huge spike in the label 2. Indeed if one check the feature map of that neuron, the digit prototype is 2 as expected (see Fig. 6.16).

Moreover one can inspect the feature maps of the trained network; for the first layer is quite simple evaluating the feature maps since they are just a reshape version of the rows of the first matrix of weights. But how evaluate the feature maps of the second layer, or in other words, what the output neurons see respect the input? It is well known in ML literature how to do it; let's call again W^1 the matrix of weights of the first layer, while W^2 the matrix of the second one. The network will have 10 feature maps seen from the output neurons, each of them composed by 784 weights.

$$F_{ik} = \sum_{j=1}^{N_h=361} W^2_{ij} * W^1_{jk} \quad \text{where } k \in \{1,\ldots,784\}, i \in \{1,\ldots,10\} \tag{6.2}$$

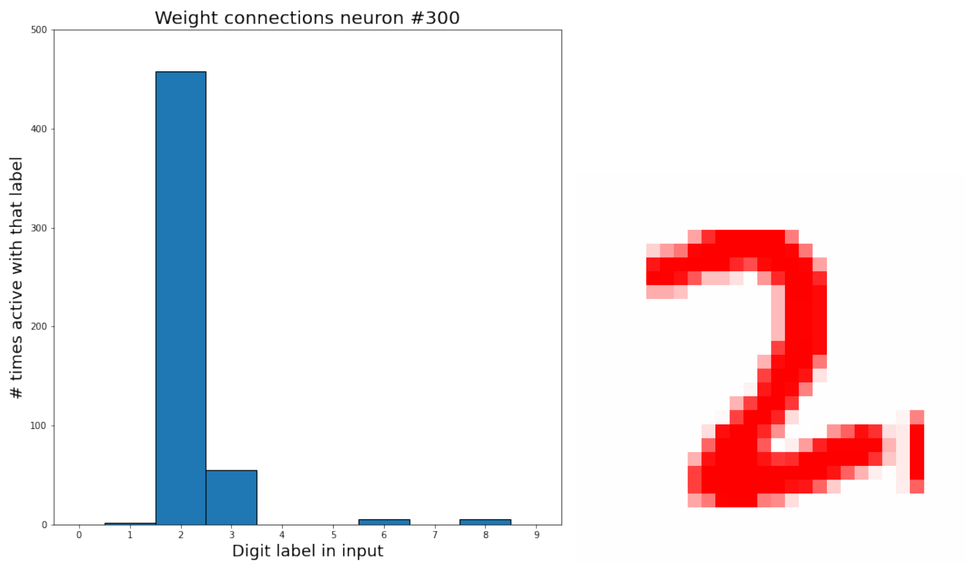


Figure 6.16: (Left) Visualization of the counts for neuron number 300. (Right) Feature map embedded in the first layer for this neuron.

The resulting feature maps for the second layer are shown in Fig. 6.17.

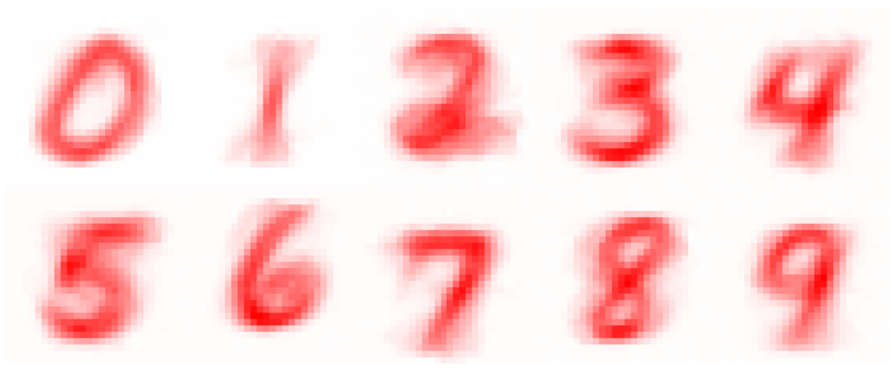


Figure 6.17: Feature maps of the second layer obtained from an ANN trained with the proposed algorithm. Each output neuron ”sees” its corresponding digit through the weights of the network.

Even though this result seems obvious, let’s check what feature maps an ANN, trained end-to-end with BP, are embedding in its second layer (see Fig. 6.18).

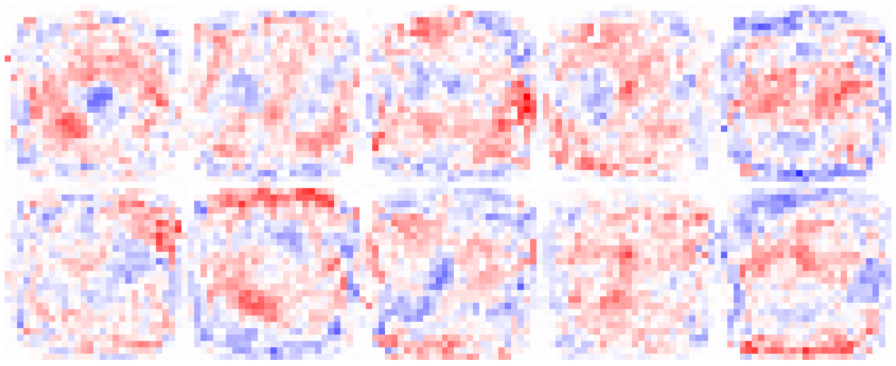


Figure 6.18: Feature maps of the second layer of an ANN trained only with BP.

The difference between Fig. 6.17 and 6.18 is enormous, so now should start to be clear why transparency is a fundamental requirement. It has no-sense stating that the neural network "has learned" these chaotic feature maps, no human ever will give trust to such architecture and so no one will use it for important and potentially critical and/or dangerous decisions. However the network has learned something since the test accuracy of such network is incredibly high, about 97%. Again, the ANN has learned non-robust features that are not understandable by humans.

6.2.1 Tuning of the Network

The training procedure proposed in this thesis gives a final result of 86.38% in the test accuracy. As said many times, no hyper-parameters were fine tuned, the values are chosen based on some suggestions that could be found in the original paper [6] and some heuristics. The hyper-parameters and their values used up to now are the following:

- $\Delta = -0.05$: the strength of the anti-Hebbian learning.
- $w_{inh} = 1.1$: the strength of the within lateral competition between hidden neurons.
- $h^* = 5$: a parameter describing the extension of the anti-Hebbian regime.
- $\sigma = 0.2$: standard deviation of the gaussian pdf used for the weights initialization in the first layer.
- $\tau_w = 5$: once τ_r and $t^1_{max} + t^2_{max}$ are chosen such that the hidden neurons reach a steady state solution, τ_w with t^3_{max} are the only left parameters. But t^3_{max} is set in order to avoid the divergence discussed above.
- $\lambda = 0.95$: the threshold used in the hidden layer to establish the "super-active" neurons.
- $N_h = 361$: the number of hidden neurons.
- $r_{seed} = 2347$: the random seed determining both the weight initialization and which images are presented to the bio-algorithm.
- Even if it is not a parameter, the composition of the N_{images} could be important. Up to now, every digit has the same number of images inside the small dataset composed by N_{images} presented during the first phase.

The number of images presented to the bio-algorithm ($N_{images} = 600$) is changed w.r.t the number of hidden neurons, such that the ratio is kept near $2 : 3$, i.e. every 2 hidden neurons, 3 images are presented in input. Since there are 7 parameters, considering 4 possible values for each of them and considering that each training requires $\approx$ 8 minutes, one need $4^7 \times 8\text{min} \approx 91$ days to perform the grid search exactly; obviously this is infeasible. Starting from the default values, one can keep all the values except one and see what happen to the test accuracy: obviously this is a compromise, a better solution could be random search. The results are reported in Table 6.1.

Δ	w_{inh}	σ	h^*	λ	τ_w	N_h - N_{images}
-0.01	0.9	0.1	1	0.91	1	169 - 280
-0.03	0.95	0.2	5	0.93	3	361 - 600
-0.05	1.1	0.3	7	0.95	5	484 - 800
-0.07	1.3	0.4	10	0.97	10	625 - 1030
Test accuracy %						
88.01	82.36	73.61	86.14	87.93	Div.	84.11
87.58	84.69	88.76	88.23	88.27	88.46	88.23
88.23	88.23	88.24	88.36	88.23	88.23	89.96
87.64	88.76	88.14	88.08	87.70	89.12	89.98

Table 6.1: (Top) Values tried during the fine-tuning phase of the architecture. (Bottom) The corresponding test accuracy for each value above; every time a parameter changes all the other are fixed to default values described before. Div. means that the algorithm displays numeric divergence.

Moreover one can also study how the training time scales with the number of hidden neurons and the number of images presented (see Table 6.2).

$N_{images} \| N_h = 484$	Training time $[s]$	$N_h \| N_{images} = 600$	Training time $[s]$
280	199.8	169	136.9
600	424.9	361	304.8
800	568.2	484	424.9
1030	732.2	625	584.3

Table 6.2: (Left) Training time vs. the number of images presented in input (the number of hidden neurons is fixed to $N_h = 484$). (Right) Training time vs the number of hidden neurons (the number of input images is fixed to $N_{images} = 600$).

While the ratio between training time vs N_{images} is approximately constant (≈ 0.71) suggesting a linear relation, the ratio between training time and N_h seems to increase. This is enough to show (at least for the studies made) that the computational time scales linearly with the training examples; however no other studies will be done in this direction.

Finally, let's check if the test accuracy could increase if the images in input are not equally-partitioned into the 10 digit classes. Looking to the confusion matrix shown in Fig. 6.15, the idea is to give in input more images of those digits that are more misclassified, for example $3 - 5$ and$4 - 9$. The reason behind why these digits are confused is because they shear many pixels positions, as shown in Fig. 6.13. In order to find a statistical evidence, let's try to train the network with 5 different random seeds for two pdf of number of digit samples (see Table 6.3). These two proposed pdf are the uniform (i.e. what done up to now) defined by a set of weights $[0.1, \dots, 0.1]$ and a custom pdf, composed by the following set weights $[0.05, 0.05, 0.075, 0.15, 0.15, 0.15, 0.05, 0.05, 0.125, 0.15]$. In a such a way the digit 0 will have less images ($600 \times 0.05 = 30$) compared to the uniform case ($600 \times 0.1 = 60$).

r_{seed}	Test accuracy %, custom pdf	Test accuracy %, random pdf
2347	90.16	90.02
43782	89.65	90.04
7653	89.59	89.56
57829	89.75	89.97
9267	89.94	89.36
24738	89.60	89.56

Table 6.3: Test accuracy, for 6 random seed, for both random and uniform pdf.

So one can calculate the mean and the standard deviation of the mean using the t-student distribution for the custom and uniform pdfs, obtaining $89.9 \pm 0.1\%$ and $89.7 \pm 0.1\%$ respectively. The z-test gives a result of 0.56, so the two test accuracy are statistically equal; no other studies will be done in this direction.

With the tuned parameters, I can finally say that the test accuracy of the trained architecture is $89.9 \pm 0.1\%$, showing an increase of $+3.5\%$ respect the non fine-tuned version.

6.2.2 Adversarial Vulnerability

Up to now, improvements have been done in increasing the test accuracy, but what about adversarial vulnerability? It turns out that using again the FGSM attack, now the network is totally resilient (up to a certain point) to these attacks; in particular the test accuracy under attack decrease in a very slightly way (see Fig. 6.19).

Even if the evaluation of the test accuracy of a neural network is usually done with an ε that is a small fraction, typically from a minimum of $1/255$ up to 0.3, one can surely increase ε. Obviously the network will perform worse, but the point is that the perturbation applied to the image is so strong that is not anymore an adversarial attack (see Fig. 6.20). Remember, from the theory of adversarial attacks, that a perturbation is considered as an attack if it is imperceptible to human-eye. Note also that in Fig. 6.8 the minimum value of the test accuracy under attack was $\approx 3\%$, i.e. the attacker is always able to create a perturbation that leads the classifier to a misclassification. Here instead the minimum value for $\varepsilon \geq 1$ is equal to 50.3%, i.e. the success of the attack is random, sometimes the attacker achieves its goal some other not. Even if the value of test accuracy is low, this is an incredibly powerful form of defense, because the network is able to reduce the attacker to a random attacker.

Moreover, using the library torchattacks [29], one can import and implement all the other possible types of adversarial attacks. The adversarial vulnerability of the model was tested using all the adversarial attacks described in chapter 4 and some variants of them, in particular: FGSM, BIM, RFGSM, PGD, PGDL2, FFGSM e TPGD. Of course there are many other, but these are the main attacks usually implemented in bench-marking platforms. In Table 6.4 it is reported the benchmark performed; the values used in the algorithms are mainly the default set from the library.

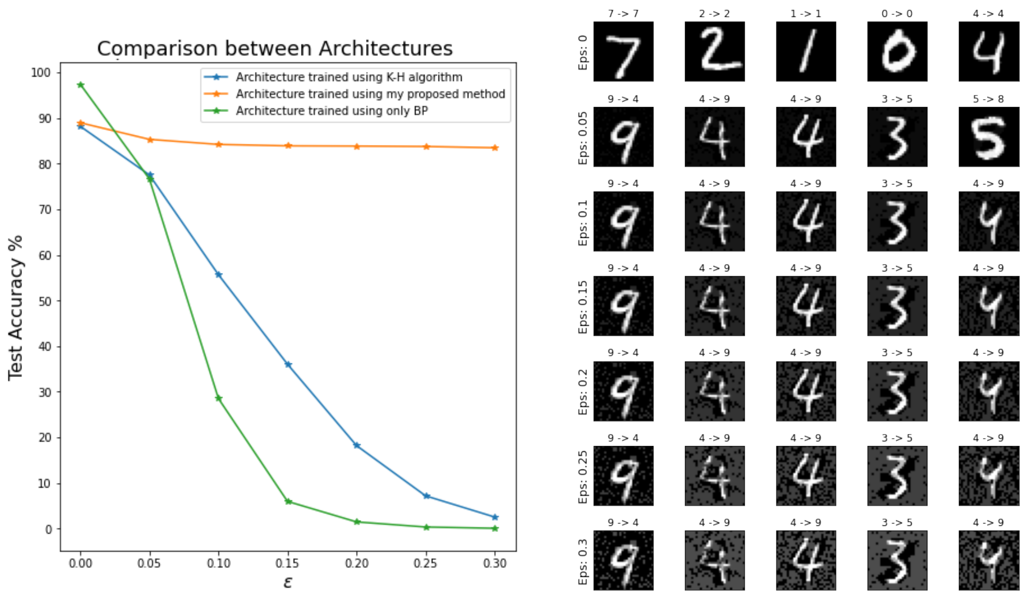


Figure 6.19: (Left) Test accuracy under attack for the proposed algorithm, varying ε, the strength of the attack, compared with other architectures. In blue the performances of the architecture proposed by K-H, in green the performances of an ANN trained only with BP while in orange the performances of the proposed algorithm. (Right) Examples of adversarial attacks, for different ε for the proposed algorithm. In each image's title, there is the original label, then an arrow and then the (wrong) label given by the network.

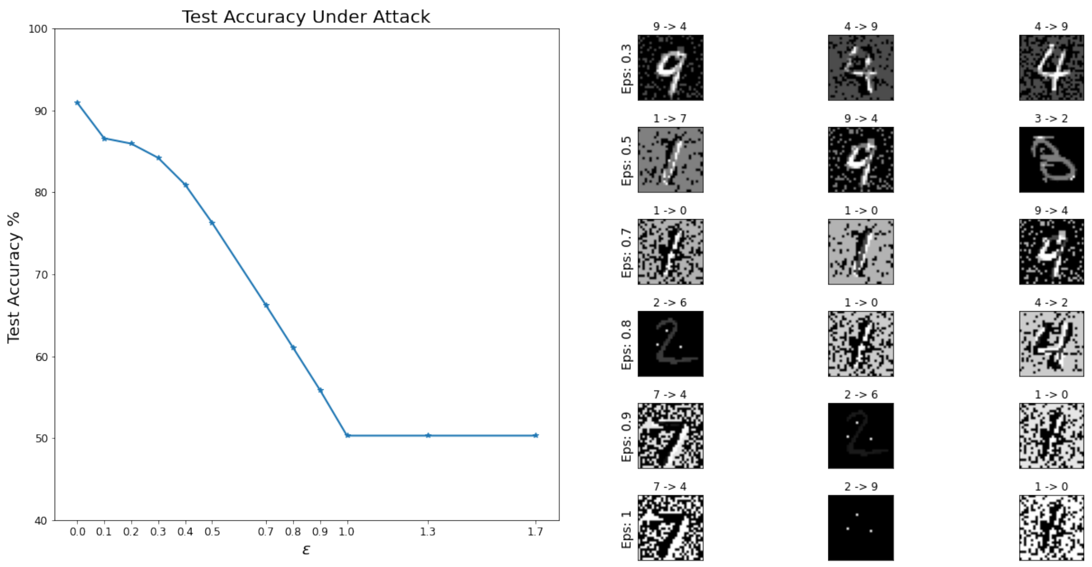


Figure 6.20: (Left) Test accuracy under attack for the proposed algorithm, varying ε, the strength of the attack, for a larger range of ε. (Right) Examples of adversarial attacks, for different ε. In each image's title, there is the original label, then an arrow and then the (wrong) label given by the network.

Adv. Attack	ε	α	Steps	Metric	Notes	Robust Accuracy
FGSM	8/255	/	/	L_∞	/	86.23%
BIM	8/255	2/255	10	L_∞	/	84.52%
RFGSM	8/255	2/255	10	L_∞	/	84.48%
PGD	8/255	2/255	10	L_∞	Random restart	84.40%
FFGSM	8/255	10/255	/	L_∞	/	86.32%
TPGD	8/255	2/255	10	L_∞	/	88.52%
PGDL2	1	0.2	40	L_2	Random restart	77.13%

Table 6.4: Benchmark of the network trained with the proposed algorithm, respect many different adversarial attacks.

In conclusion considering a trained network that displays a test accuracy of 90.96%, the test accuracy under an FGSM attack with strength $\varepsilon = 0.1$ is 86.60%, while for an $\varepsilon = 0.3$ is 84.22%.

There exist groups of research that benchmark systematically the various algorithm for adversarial defense published through the years; an example is shown in the following web page [30]. Here are published the results of many authors, that use much more complicated algorithm than mine and consequently they achieve better results. However if one measure the difference between the test accuracy and the test accuracy under attack, my algorithm performs equally or better in the MNIST data set.

7 Conclusion

I will briefly resume the thesis, highlighting the principal concepts described in it. First of all it has been discussed about the meaning of words like, trustiness, robustness, transparency etc. They are not fancy words used in the colloquial language, but they are indeed key requirements decided by the european guidelines for trustworthy AI. If one is able to built an AI algorithm keeping in mind these properties, he is increasing the human-AI trust on it. In the thesis has been also described what are adversarial attacks: they are signals, typically images, constructed in order to fool the network. The way in which these attacks are built reflects the main theory taken in consideration; as said many times, a fundamental theory of adversarial attacks does not exist, yet there are many possible explanation for them.

Then, in order to understand how an artificial neuron behaves, it has been quickly described the functioning of a real neuron and the concept of plasticity. Since the Hebb's original formulation, many other plasticity rules has been proposed, yet they shear the common idea that synapses strengthen happens between two near-by neurons. So if artificial neural networks are inspired by the brain, is back-propagation inspired by some learning paradigm happening in the brain? The answer, for the moment, is no, because for example BP is not a local algorithm, so let's try to avoid back-propagation.

The starting point for developing an algorithm that is robust, transparent and does not use BP, is the work done by Krotov and Hopfield. Their algorithm is effective in learning digit-prototypes in the first layer of the ANN. Doing so, the feature maps embedded in the matrix of weights are transparent to the user; moreover they are learned in an unsupervised way, that is a much more difficult task compared to supervised learning. Even if the intuition of using a BNN in the first layer is right, their algorithm does not show adversarial robustness; the second layer then is trained with the algorithm proposed by me. The main idea is to establish a matrix of weights, where its elements can be interpreted as transitional probabilities; this allow to calculate the output probabilities of the corresponding labels. For calculating this matrix of weights the concept of "super-active" neurons was created.

There are several pros and cons about the proposed training algorithm. The pros are related for sure to the transparency of the network, the simplicity of the training algorithm and the fact that is adversarial robust against many different types of adversarial attacks. The first layer not only is trained in an unsupervised way, but is trained with only 800 images. The cons are the training time, typically 10 minutes and the test accuracy that is roughly 90%. However also in the results of K-H it is shown how BP requires 6 times less epochs compared to their algorithm.

In conclusion, why this thesis should be of interest for the scientific community? So first of all, up to my knowledge no papers has been publicized about K-H's algorithm since the original one of 2019. Moreover up to my knowledge no one in the literature tried to test the adversarial robustness of the network proposed by K-H and no one has tried to solve the problem of adversarial defence using biological neural networks. Moreover, I think that this thesis can be of potential interest for future works. There are many things that could be improved: first of all, try to build deep biological neural networks. It could turns out that hierarchical stack of biological layers is beneficial both for robustness and accuracy. Surely, try to implement the fast version of the algorithm proposed by Krotov and Hopfield giving in input the whole MNIST training set; doing so there is the hope that the network becomes competitive respect networks trained only with BP. Third aspect, try to modify the neuronal dynamic, hoping to achieve better transparency in the algorithm sense or better accuracy.

As final conclusion, I would say that this thesis could be taken as an example of how an effective algorithm could be developed, starting from the human point of view. Indeed, in developing this algorithm, my north star was the question:

"Why I am capable to recognise digit and my network not?"

I believe that robust and transparent algorithms are the future of machine learning, in order to effectively use them for high-dangerous roles in society.

References

[1] A. Jacovi, A. Marasović, T. Miller, and Y. Goldberg, "Formalizing trust in artificial intelligence: Prerequisites, causes and goals of human trust in ai," *FAccT '21: Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, p. 624–635, 2021.

[2] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv*, 2014.

[3] A. Shamir, I. Safran, E. Ronen, and O. Dunkelman, "A simple explanation for the existence of adversarial examples with small hamming distance," *arXiv*, 2019.

[4] K. Warr, *Strengthening Deep Neural Networks: Making AI Less Susceptible to Adversarial Trickery*. O'Reilly, 2019.

[5] L. Abbott and P. Dayan, *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*, 2005.

[6] D. Krotov and J. Hopfield, "Unsupervised learning by competing hidden units," *PNAS*, vol. 116, pp. 7723–7731, 2019.

[7] T. Miller, "Explanation in artificial intelligence: Insights from the social sciences," *Artificial Intelligence*, vol. 167, pp. 1–38, 2019.

[8] C. Rudin, "Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead," *Nature Machine Intelligence*, vol. 1, pp. 206–215, 2019.

[9] Z. C. Lipton, "Mythos the of model interpretability: In machine learning, the concept of interpretability is both important and slippery." *Queue*, vol. 16, pp. 31–57, 2018.

[10] C. R. J. Radin, "Why are we using black box models in ai when we don't need to? a lesson from an explainable ai competition," *Harvard Data Science Review*, 2019.

[11] R. R. Hoffman, "A taxonomy of emergent trusting in the human–machine relationship," 2017.

[12] Guidelines. [Online]. Available: https : / / digital-strategy . ec . europa . eu / en / library / ethics-guidelines-trustworthy-ai

[13] K. Been, R. Khanna, and O. O. Koyejo, "Examples are not enough, learn to criticize! criticism for interpretability." *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pp. 2288–2296, 2016.

[14] H. Li, Y. Fan, F. Ganz, A. Yezzi, and P. Barnaghi, "Verifying the causes of adversarial examples," *25th International Conference on Pattern Recognition (ICPR)*, 2020.

[15] Y. Bengio, "Learning deep architectures for ai." *Foundations and Trends® in Machine Learning*, vol. 2, pp. 1–127, 2009.

[16] quora. [Online]. Available: https : / / www . quora . com / What-does-it-mean-for-an-algorithm-say-deep-learning-algorithms-to-generalize-non-locally

[17] I. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv*, 2015.

[18] A. Ilyas, S. Santurkar, D. Tsipras, L. Engstrom, B. Tran, and A. Madry, "Adversarial examples are not bugs, they are features," *33rd Conference on Neural Information Processing Systems*, vol. 32, 2019.

[19] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," *arXiv*, 2016.

[20] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial examples in the physical word," *arXiv*, 2017.

[21] F. Tramèr, A. Kurakin, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel, "Ensemble adversarial training: Attacks and defenses," *arXiv*, 2017.

[22] W. McCulloch and W.Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, pp. 115–133, 1943.

[23] D. Hebb, *The organization of behavior; a neuropsychological theory*, 1949.

[24] D. Rumelhart, G. Hinton, and R. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, 1986.

[25] E. Bienenstock, L. Cooper, and P. Munro, "Theory for the development of neuron selectivity: orientation specificity and binocular interaction in visual cortex." *Journal of Neuroscience*, vol. 2, pp. 32–48, 1982.

[26] E. Oja, "A simplified neuron model as a principal component analyzer," *Journal of Mathematical Biology*, vol. 15, pp. 267–273, 1982.

[27] skorch. [Online]. Available: https://skorch.readthedocs.io/en/stable/

[28] PyTorch. [Online]. Available: https://pytorch.org/tutorials/beginner/fgsm_tutorial.html

[29] torchattacks. [Online]. Available: https://adversarial-attacks-pytorch.readthedocs.io/en/latest/attacks.html

[30] robustml. [Online]. Available: https://www.robust-ml.org/defenses/

Acknowledgments

Vorrei ringraziare il mio relatore, prof. Marco Baiesi che, oltre ad essere stato un ottimo professore, ha sempre ammirato le mie intuizioni da fisico e le ha sempre supportate nel loro sviluppo. Lo ringrazio per tutto il tempo dedicatomi e tutte le gentilezze avute nei miei confronti. Lo ringrazio infine per avermi incoraggiato a partecipare al dottorato di ricerca facendo sì di non avere rimorsi in futuro.

Anche se indirettamente, vorrei ringraziare il prof. Samir Simon Suweis che attraverso il progetto per l'esame di Quantiative Life Science, mi ha permesso di venire a conoscenza del lavoro svolto da Krotov e Hopfield e avere quindi il punto di partenza per questa tesi.

Finally, I would like to thank prof. Byers Jefferson MC Culloch, not only for an illuminating discussion on adversarial attacks in Venice, but also for a sentence said during the course of Information Theory, whose concept can be summarized as follow:
"If from your machine learning algorithm could depend the life of a person, you really should be careful in what you are doing."