MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

"Adversarial Training with Lookahead"

verfasst von / submitted by

Philip Neuhart, BSc.

angestrebter akademischer Grad / in partial fulfillment of the requirements for the degree of

Master of Science (MSc)

Wien, 2022 / Vienna 2022

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	UA 066 821
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Masterstudium Mathematik
Betreut von / Supervisor:	Univ.-Prof. Dr. Radu Ioan Boţ

Abstract

Deep learning is applied successfully in more and more security-sensitive areas such as autonomous driving or face recognition. Thus, the safety aspect of deep learning algorithms is increasingly getting in the focus of research. One of the most important discoveries in this regard has been that adversaries can deceive state-of-the-art machine learning models by inducing small but deliberate perturbations to inputs (e.g: changing the pixel values of images slightly). The resulting perturbed inputs are called *adversarial examples*. Even though adversarial examples are crafted on a specific model, they were shown to be transferable to other models, i.e., they also fool other models with a high likelihood. This transferability property of adversarial examples poses major security threats to the safe deployment of deep learning algorithms in security-sensitive areas. This gave rise to a new branch of deep learning called *adversarial learning* which also studies potential defenses against *adversarial attacks*. A promising defense strategy, *adversarial training*, replaces the unperturbed inputs in the training phase with adversarial examples in order to strengthen the model's ability to detect adversarial examples.

In this thesis, we will formally introduce adversarial examples, cover the most important adversarial attacks, and discuss established hypotheses regarding the existence of adversarial examples and their transferability property. Furthermore, we will discuss a selected list of defenses and analyze adversarial training from the perspective of robust optimization. In the second part of this thesis, we will introduce and analyze a recently proposed optimization algorithm called Lookahead which has been shown to be able to outperform standard optimization algorithms like SGD or Adam in certain optimization settings. Lastly, we will use Lookahead to adversarially train convolutional neural networks in several numerical experiments and benchmark Lookahead against five other optimization algorithms: SGD, Adam, OGD, ExtraSGD, and ExtraAdam.

Zusammenfassung

Deep Learning wird in immer mehr sicherheitsrelevanten Bereichen wie zum Beispiel für autonomes Fahren oder in der automatischen Gesichtserkennung erfolgreich eingesetzt. Daher rückt der Sicherheitsaspekt von Deep Learning Algorithmen zunehmend in den Fokus der Forschung. Eine der wichtigsten Entdeckungen in diesem Zusammenhang war, dass Angreifer state-of-the-art Machine Learning Modelle, wie z.B. Deep Convolutional Neural Networks, täuschen können, indem sie die zu klassifizierenden Objekte mit kleinen, aber gezielt gesetzten Störungen manipulieren (z.B.: die Pixelwerte eines Bildes leicht abändern). Die daraus resultierenden manipulierten Objekte werden als *Adversarial Examples* bezeichnet. Obwohl Adversarial Examples für ein bestimmtes Modell erstellt werden, sind sie nachweislich auf andere Modelle übertragbar, d.h. sie täuschen auch andere Modelle mit hoher Wahrscheinlichkeit. Diese Übertragbarkeitseigenschaft von Adversarial Examples stellt ein großes Sicherheitsrisiko für den Einsatz von Deep Learning Algorithmen speziell in sicherheitssensiblen Bereichen dar. Im Zuge dieser Entdeckungen entstand ein neuer Unterbereich von Deep Learning, genannt *Adversarial Learning*, welcher sich unter anderem mit potenziellen Verteidigungsstrategien gegenüber Angriffen auf Deep Learning Modelle mit Hilfe von Adversarial Examples, auch *Adversarial Attacks* genannt, befasst. Eine vielversprechende Verteidigungsstrategie, *Adversarial Training* genannt, ersetzt die nicht manipulierten Objekte in der Trainingsphase durch Adversarial Examples, um so die Fähigkeit des Klassifizerungsmodells zu stärken, Adversarial Examples zu erkennen.

In dieser Arbeit werden wir Adversarial Examples formal einführen, die wichtigsten Adversarial Attacks behandeln und etablierte Hypothesen zur Existenz von Adversarial Examples sowie deren Übertragbarkeitseigenschaft diskutieren. Darüber hinaus werden wir eine ausgewählte Liste an Verteidigungsstrategien beschreiben und Adversarial Training aus der Perspektive der robusten Optimierung analysieren. Im zweiten Teil dieser Arbeit werden wir Lookahead, einen kürzlich vorgeschlagenen Optimierungsalgorithmus, vorstellen und analysieren. Es wurde gezeigt, dass Lookahead in der Lage ist, Standard Optimierungsalgorithmen wie SGD oder Adam unter bestimmten Vorraussetzungen zu übertreffen. Schließlich werden wir Lookahead verwenden, um Convolutional Neural Networks in mehreren numerischen Experimenten zu trainieren und die Ergebnisse mit fünf anderen Optimierungsalgorithmen zu vergleichen: SGD, Adam, OGD, ExtraSGD und ExtraAdam.

Acknowledgement

First of all, I would like to express my sincere gratitude to my supervisor Prof. Dr. Radu Ioan Bot for his continued support throughout my academic career. I also want to thank Dr. Axel Böhm for his incredible support and valuable feedback, without which this thesis would not have been possible.

My biggest thanks go to my parents Christian and Lucia Neuhart whose love seems endless and who provided me with everything that is needed for a happy and successful life. In addition, I want to thank my aunt Marion Böck for her continued support over the past years.

Last but not least, I want to thank Marcel Harrer without whom I would not have been able to make it past the first semester.

Contents

Notation

$\mathbb{X} \subseteq \mathbb{R}^n$	input space
n	dimension of input space
$x \in \mathbb{X}$	benign/clean input (e.g. images)
$x' \in \mathbb{X}$	adversarial example/sample
$r \in \mathbb{R}^n$	adversarial perturbation
$\langle x, z \rangle$	inner product between x and z
m	number of classes/labels
$l_c \in \{1, ..., m\}$	correct class/label of input x
$l_t \in \{1, ..., m\}$	target class/label
$\mathscr{D}$	labeled data distribution
$\Theta \subseteq \mathbb{R}^p$	parameter space
θ	parameter vector
$F : \mathbb{X} \times \Theta \to \mathbb{R}^m$	neural network
$Z(x)$	neural network up to the last hidden layer
$F(x)$	output layer
$y \in \mathbb{R}^m$	output vector
$f : \mathbb{X} \to \{1, ..., m\}$	classifier
$\mathcal{L}$	loss function
$\|\cdot\|_p$	ℓ_p/L_p-norm
η	learning rate
ϵ	input variation parameter
x^T	transpose vector
$\mathbb{E}$	Expected value
$\mathbb{V}$	Variance
I	identity matrix

Table 1: Notations. For details, see Sections 1.1 and 2.1.

1 Introduction

Deep neural networks (DNN) are to achieve superhuman performance in computer vision tasks [45, 8]. They are also successfully used for speech recognition [19] and many other applications. However, researchers found that DNNs are vulnerable to attacks. Szegedy et al. [40] first showed that neural networks trained on object classification tasks are susceptible to small perturbations of the input which cause them to misclassify the input. These perturbed inputs are called adversarial examples. Unperturbed inputs are then referred to as clean or benign inputs. Even though adversarial examples are relatively easy to generate, they are hard to find by simply sampling randomly. Today, numerous algorithms for creating adversarial examples exist (see Section 2.2). These algorithms are also referred to as adversarial attacks. An illustration for an adversarial example is provided in Figure 1.1. The image on the left depicting the number '5' is unperturbed

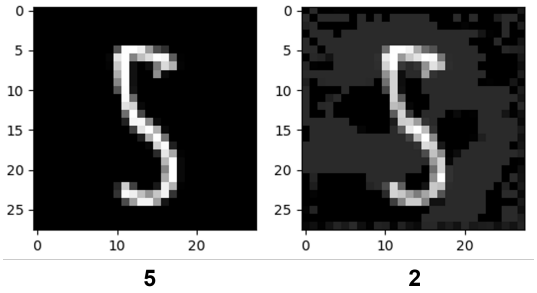


Figure 1.1: The left image depicting a '5' is taken from the MNIST data set[27] and unperturbed while the image on the right-hand side is an adversarial example generated by the projected gradient descent algorithm.

and is classified correctly by a neural network classifier. Using the projected gradient descent attack (see Section 2.2.4), the classifier is misled into predicting the number '2' instead. On the other hand, most humans would still be able to recognize the perturbed version as a '5'.

1.1 Basic Concepts and Notation

This section is dedicated to introducing the basic concepts and notation used for the rest of the thesis.

1.1.1 Supervised Learning

The task of assigning labels $l \in \{1, ..., m\}$, for $m \in \mathbb{N}$ to inputs $x \in \mathbb{X} \subseteq \mathbb{R}^n$, for $n \in \mathbb{N}$, is called *classification*. For example, in the context of a computer vision problem, the inputs x could represent images of animals to be labeled by the classifier. A data set is a finite set of input-label pairs (x, l_c) drawn from a data distribution $\mathscr{D}$ on $\mathbb{X} \times \{1, ..., m\}$. The *learning process* consists of using the training data to find a function

$$\begin{aligned} f : \mathbb{X} &\to \{1, ..., m\} \\ x &\mapsto f(x), \end{aligned}$$

called a *classifier*, among a pool of candidates, called the hypothesis space $\mathcal{H}$, that best solves the classification task. Data that has not been used in the learning process is referred to as unseen data or test data.

For a parametric hypothesis space $\mathcal{H} = \{f_\theta : \mathbb{X} \times \Theta \to \{1, ..., m\} : \theta \in \Theta\}$, where $\Theta \subseteq \mathbb{R}^p$ is called the parameter space, the learning process consists of using a learning algorithm to choose parameters θ. For the rest of this thesis we will assume a parametric hypothesis space.

In the next section, we will formally introduce neural networks which can be used to define a classifier.

1.1.2 Neural Networks

Definition 1. (Nodes and Layers). A node N is a function

$$\begin{aligned} N : \mathbb{R}^n &\to \mathbb{R} \\ x &\mapsto \phi(\langle w, x \rangle + b), \end{aligned}$$

where the parameters $w \in \mathbb{R}^n$ and $b \in \mathbb{R}$ are called the weight and bias of node N, respectively, and $\phi : \mathbb{R} \to \mathbb{R}$ is called an activation function.

Furthermore, we define a hidden layer as a function $L : \mathbb{R}^n \to \mathbb{R}^m$ consisting of m nodes:

$$L(x) = \begin{pmatrix} \phi(N_1(x)) \\ \vdots \\ \phi(N_m(x)) \end{pmatrix},$$

where the same activation function ϕ is applied for each node.

Additionally, we define an output layer as a function

$$\begin{aligned} \psi : \mathbb{R}^m &\to \mathbb{R}^m \\ z &\mapsto \psi(z). \end{aligned}$$

Definition 2. (Neural Networks). A fully connected neural network $F : \mathbb{R}^n \to \mathbb{R}^m$ is defined as a concatenation of $s-1$ hidden layers and a single output layer L_s. More precisely, for $2 \leq j \leq s-1$ the output of node $N_i^{L_j}$ of layer L_j of F, is given by:

$$N_i^{L_j}(x) = \phi\left(\left\langle w_i^{L_j}, L_{j-1}(L_{j-2}(\cdots L_1(x)))\right\rangle + b_i^{L_j}\right),$$

for $i \in \{1, ..., r_{L_j}\}$, and finally, the network's output is given by:

$$\begin{aligned} F_i(x) &= \psi \begin{pmatrix} N_1^{L_{s-1}}(L_{s-2}(\cdots L_1(x))) \\ \vdots \\ N_m^{L_{s-1}}(L_{s-2}(\cdots L_1(x))) \end{pmatrix} \\ &= \psi(L_{s-1}(L_{s-2}(\cdots L_1(x)))), \end{aligned}$$

for $i \in \{1, ..., m\}$.

The network without the output layer is denoted by Z, i.e.,

$$Z(x) = L_{s-1}(L_{s-2}(\cdots L_1(x))),$$

which consequently matches the output of the last hidden layer. In matrix-vector notation, the feed-forward neural network can be written as,

$$F(x) = \psi(\phi(W^{s-1}(\phi(\cdots \phi(W^1 x + b^1))) + b^{s-1})),$$

with $W^j \in \mathbb{R}^{r_{L_j} \times r_{L_{j-1}}}$ and $b^j \in \mathbb{R}^{L_j}$, for $1 \leq j \leq s-1$, where ϕ is applied component-wise.

An illustration is provided in Figure 1.2.

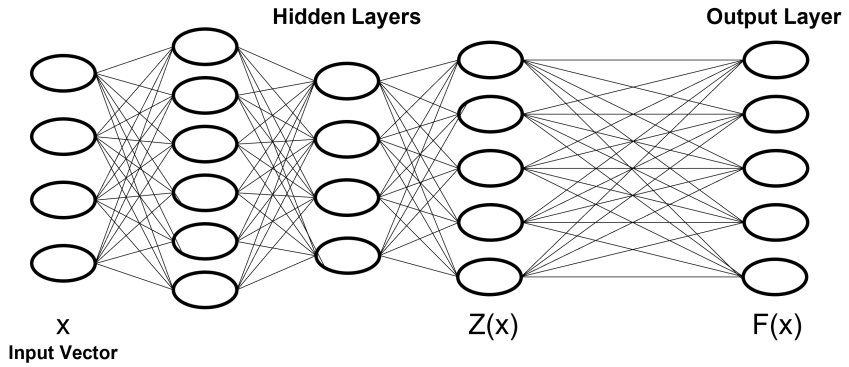


Figure 1.2: Illustration of a typical neural network architecture with three hidden layers.

Definition 3. (Softmax layer). A softmax output layer is a function $\psi : \mathbb{R}^m \to \mathbb{R}^m$ defined by

$$\psi(z) = \left[\frac{e^{z_i}}{\sum_{i=1}^{m} e^{z_i}}\right]_{i=1,...,m}. \tag{1.1.1}$$

In case a neural network is equipped with a softmax output layer, we refer to the components $Z_i(x)$ of the output vector of the last hidden layer as logits.

If not specified otherwise, all neural networks are assumed to have a softmax output layer for the rest of this thesis.

A corresponding classifier can be defined via $f(x) := \arg\max_i F_i(x)$, where $F_i(x)$ is the output of the i-th component F.

1.1.3 Loss function

In order to be able to assess how well a classifier performs on a given test set, we need a way to quantify the classifier's performance.

Definition 4. (Loss function). For a function $g : \mathbb{R}^m \times \{1, ..., m\} \to \mathbb{R}^+$ and a neural network F_θ, we define a *loss function* as a function

$$\begin{aligned} \mathcal{L} : \Theta \times \mathbb{X} \times \{1, ..., m\} &\to \mathbb{R}^+ \\ (\theta, x, l_c) &\mapsto g(F_\theta(x), l_c). \end{aligned}$$

We defined loss functions such that they implicitly depend on the neural network F_θ to implicitly contain the neural network used for the classification task. Therefore, a loss

function $\mathcal{L}$ always corresponds to a neural network F_θ. The loss function $\mathcal{L}$ is used to quantify how *far off* a classifier is from predicting the correct label of a given input x. (1.1.2) provides an example of a loss function commonly used in combination with neural networks. If $\mathcal{L}$ is differentiable in $\theta \in \Theta$, standard optimization algorithms can be used to minimize the loss.

Definition 5. (One-hot encoding). For a label $l \in \{1, ..., m\}$, we define its one-hot encoding as the indicator vector $\hat{l} := (0, .., 0, 1, 0, ..0) \in \mathbb{R}^m$, where the $l - th$ entry is the only non-zero entry.

Definition 6. (Cross entropy). The cross-entropy of two probability distributions $p, q \in \mathbb{R}^m$ is defined by

$$H(p, q) := -\sum_{i=1}^{m} \log(p_i) q_i.$$

Typically, cross-entropy is used as a loss function in combination with neural networks in classification settings. The cross-entropy loss takes probability distributions, i.e., vectors whose entries sum up to 1, as inputs, thus, one-hot encoding of the labels is required. The cross-entropy loss of a model F_θ for a given input x with true label $\hat{l}_c = (0, ..., 1, ..., 0)^T$ is given by

$$\begin{aligned} \mathcal{L} : \Theta \times \mathbb{X} \times \{1, ..., m\} &\to \mathbb{R}^+ \\ (\theta, x, l_c) &\mapsto H(F_\theta(x), \hat{l}_c). \end{aligned} \tag{1.1.2}$$

Plugging in the definitions for cross entropy and one-hot-encoding, the cross entropy loss of (x, l_c) simplifies to $\mathcal{L}(\theta, x, \hat{l}_c) = -\log(F_{\theta, l_c}(x))$, where F_{θ, l_c} is the l_c-th entry of F_θ.

1.1.4 Other Concepts

This section is dedicated to introducing some fundamental definitions used throughout this thesis.

Definition 7. (L-smooth). A function $f : \mathbb{R}^p \to \mathbb{R}$ is called L-smooth if

$$\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|, \forall x, y \in \mathbb{R}^p.$$

Let $f : \mathbb{R}^p \to \mathbb{R}$, and let T be a finite training set drawn from a data distribution $\mathscr{D}$, and let $\mathcal{P}(T)$ be the power set of T. Then, consider the following finite sum minimization problem:

$$\min_\theta \hat{f}(\theta) := \frac{1}{|T|} \sum_{t \in T} f(\theta, t). \tag{1.1.3}$$

Definition 8. (Stochastic Gradient). Let $\mathscr{D}$ be a distribution on the training set T and let $\mathcal{P}(T)$ be the power set of X. For a finite sum minimization problem as defined in (1.1.3), we call an estimator $g : \mathbb{R}^p \times \mathcal{P}(X) \to \mathbb{R}$ of $\hat{f}$ a stochastic gradient.

Moreover, two common assumptions made for the stochastic gradient g are that it is unbiased, i.e., it satisfies, $\forall \theta \in \Theta$,

$$\mathbb{E}_{S\sim\mathscr{D}}[g(\theta, S)] = \nabla \hat{f}(\theta),$$

that g has bounded variance, i.e., for $\sigma \geq 0$,

$$\mathbb{E}_{S\sim\mathscr{D}}\left[\|g(\theta, S) - \nabla \hat{f}(\theta)\|^2\right] \leq \sigma^2.$$

Example. An example of a stochastic gradient is given by

$$g(\theta, \{t\}) = \nabla_\theta f(\theta, t),$$

i.e., the gradient of f evaluated at (θ, t). More generally, in *mini-batching*, we obtain the following stochastic gradient

$$g(\theta, S_n) = \frac{1}{|S_n|} \sum_{t\in S_n} \nabla_\theta f(\theta, t),$$

where S_n is sampled from $\Omega = \{S_n | S_n \subset X, |S_n| = n\}$, for $n \ll |X|$, instead of $\mathcal{P}(X)$.

Two basic concepts of linear algebra, which we will use later, are the following.

Definition 9. (Spectrum of a matrix) Let $A \in \mathbb{C}^{n\times m}$ be a $n \times m$ matrix. The spectrum σ of A is given by

$$\sigma(A) = \{\lambda \in \mathbb{C} : \det(A - \lambda I) = 0\}.$$

Definition 10. (Spectral radius). Let $A \in \mathbb{C}^n \times \mathbb{C}^m$ be a $n \times m$ matrix, the spectral radius ρ of A is defined as

$$\rho(A) = \max_\lambda\{|\lambda| : \lambda \in \sigma(A)\}.$$

2 Adversarial Examples and Attacks

Research has shown that adversarial examples do not only pose security threats for systems operating solely in the digital world but can also be used in the physical world. Kurakin et al. [26] demonstrated that digitally crafted adversarial samples can be used in the physical world as well by simply printing them. The classifier still gets fooled with a high likelihood when it receives pictures of the print-out versions of the original adversarial examples taken by a cell phone camera as inputs. Eykholt et al. [12] demonstrated that it is even possible to generate physical perturbations for physical world objects, such as stickers posted on stop signs, that consistently cause misclassifications. These observations make adversarial examples a very important phenomenon to study since security-sensitive areas such as autonomous driving are heavily dependent on the safe deployment of machine learning algorithms. For example, a self-driving car might not stop on a crossroad if it does not recognize an adversarially perturbed stop sign as such, which could lead to accidents.

The rest of this chapter is organized as follows. In Section 2.1, a formal description of the task of crafting adversarial examples is given. A short introduction to a selected list of adversarial attacks is provided in Section 2.2. In Section 2.3, hypotheses for the existence of adversarial examples are presented and discussed. Furthermore, one of the most important properties of adversarial examples, transferability, is studied in Section 2.4.

2.1 Formal description

To be able to study the phenomenon of adversarial examples, a formal description of the problem is needed. For the rest of Chapter 2, if not specified otherwise, let $f : \mathbb{X} \to \{1, ..., m\}$ be a classifier mapping inputs $x \in \mathbb{X} \subseteq \mathbb{R}^n$ to a discrete set of labels $\{1, ..., m\}$, for $m \in \mathbb{N}$, and let $\mathcal{L} : \mathbb{R}^n \times \{1, ..., m\} \to \mathbb{R}^+$ be the corresponding loss function. Note that in general, both f and $\mathcal{L}$, as defined in Section 1.1, implicitly depend on parameters θ. For convenience, they will be omitted in the rest of Chapter 2, and

can be viewed as fixed. This is justified by the assumption that attacks happen after the training phase of a model.

Definition 11. (Adversarial Attack). Broadly speaking, we call an algorithm $\mathcal{A}$ generating adversarial examples $x + r$ from clean pairs (x, l_c) for a given classifier f, an adversarial attack. More precisely, for a specific classification task, i.e., given hypothesis space $\mathcal{H}$, input space $\mathbb{X}$ and number of classes m, let $f : \mathbb{X} \to \{1, ..., m\}$ be a classifier chosen from $\mathcal{H}$ and let $(x, l_c) \in \mathbb{X} \times \{1, ..., m\}$ be a (clean) input-label pair drawn from a data distribution $\mathscr{D}$ on $\mathbb{X} \times \{1, ..., m\}$. Then, an adversarial attack corresponds to a map

$$\begin{aligned} \mathcal{A} : \mathcal{H} \times \mathbb{X} \times \{1, ..., m\} &\to \mathbb{X} \\ (f, x, l_c) &\mapsto x + r. \end{aligned}$$

In general, adversarial attacks are applicable for a wide range of different classification tasks and are therefore not restricted to a specific setting ($\mathcal{H}$, $\mathbb{X}$, m). Usually, attacks are dependent on multiple parameters as we will see in Section 2.2.

Furthermore, attacks can be either *targeted* or *untargeted*. Targeted attacks aim for a misclassification as a specific label l_t, referred to as *target label*, while untargeted attacks only strive for a misclassification as any label other than the true label $l_c(x)$.

Adversarial attacks usually fit in one of two settings, either the minimal perturbation or the maximal loss setting. If convergence is guaranteed, attacks in the former guarantee a misclassification by the victim model but may induce a large distortion. Conversely, attacks in the latter ensure low distortion of the input but may not produce adversarial examples that actually will be misclassified by the model under attack.

2.1.1 Minimal perturbation

First formalized by Szegedy et al. [40], an adversary in the minimal perturbation setting aims at generating an adversarial example $x + r$ for a given image $x \in \mathbb{X}$, a corresponding label $l_c(x)$ and a target label $l_t \in \{1, ..., m\} \setminus \{l_c(x)\}$ while keeping the perturbation r as small as possible. This corresponds to solving the following optimization problem:

$$\begin{aligned} \min_{r \in \mathbb{R}^n} \quad & \|r\| \\ \text{s.t.} \quad & f(x + r) = l_t \\ & x + r \in \mathbb{X}, \end{aligned} \tag{MP}$$

for $l_t \in \{1, ..., m\}$ and a classifier f. Note that the solution for (MP) is not necessarily unique and depends on the choice of the norm. Popular choices are the ℓ_∞-norm, the ℓ_2-norm and the ℓ_0-norm [15, 6, 33]. However, Sharif et al. [37] show that small distances under ℓ_p-norms ($p = 0, 2, \infty$) between benign inputs and corresponding adversarial examples, in general, do not guarantee perceptual similarity.

The second constraint $x + r \in \mathbb{X}$ ensures that the adversarial example is still a valid input, e.g., the pixel values are within the permitted range. Some authors omit this constraint since in most cases the adversarial distortions are small enough such that the adversarial examples are also in the domain of the function.

In the context of an adversarial attack, it is usually assumed that the classifier correctly predicts the label/class of the clean input x, i.e., $f(x) = l_c$, and that the adversary is only interested in the non-trivial case of $l_t \neq l_c$. Thus, when solving the optimization problem one is essentially looking for the smallest perturbation of the input such that the classifier changes its label prediction from l_c to l_t. However, in some cases one, is satisfied with any prediction other than the true label.

2.1.2 Maximal Loss

In contrast, Goodfellow et al. [15] introduced a different approach. Rather than looking for the smallest distortion of the input that still changes the label prediction, the adversary is striving for the greatest increase in loss induced by the perturbation. To be precise, they need to solve the following optimization problem:

$$\begin{aligned} \max_{r \in \mathbb{R}^n} \quad & \mathcal{L}(x + r, l_c) \\ \text{s.t.} \quad & \|r\| \leq \epsilon, \end{aligned} \tag{ML}$$

where $\mathcal{L}$ is the corresponding loss function and $\epsilon > 0$ is small such that the perturbation is undetectable or at least not too big. Note that the solution to (ML) is again not necessarily unique and is also dependent on the choice of the norm.

2.1.3 Threat Model Taxonomy

In adversarial learning, a threat model represents an attack scenario in a predefined framework. They mainly differ along two dimensions, adversarial capability and adversarial goals. Adversaries can have different levels of access to the target system or knowledge about its architecture. Additionally, different attackers may have different goals, see

Figure 2.1 for an overview. *Confidence reduction* means decreasing the confidence of the

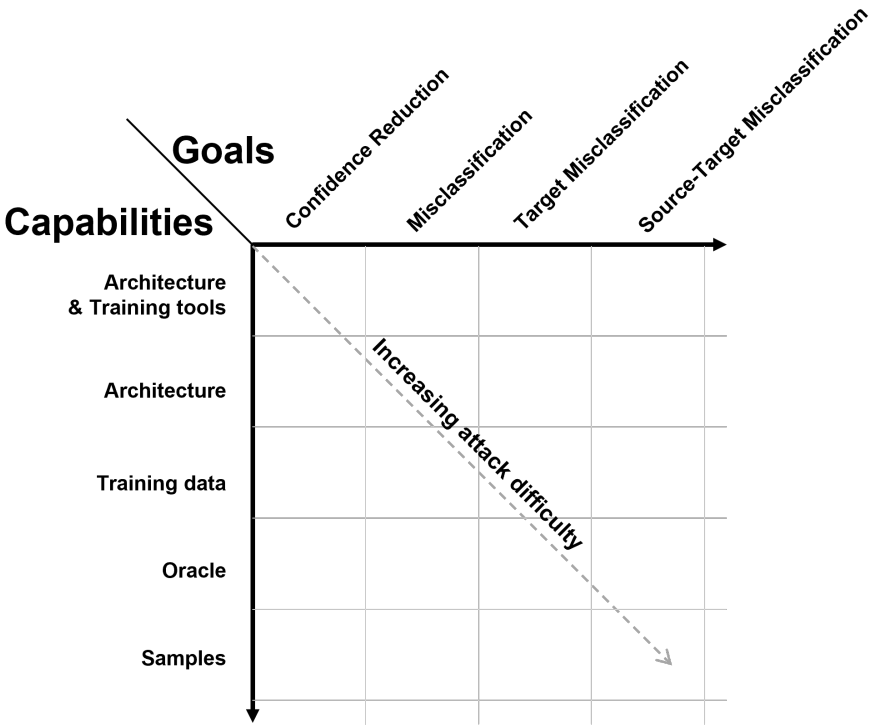


Figure 2.1: Threat model taxonomy: Threat models differ in adversarial capability and adversarial goals [33].

target model when classifying test inputs. On top of that, an adversary could aim for a *misclassification*, forcing the model to classify a given test example as any class other than the true one. Aiming for a *targeted misclassification* is even more ambitious. The adversary is able to produce adversarial examples that force the model to label them as a *specific* target class. However, the best case scenario for the adversary is *source-target misclassification* which would enable the attacker to choose specific inputs from any class and generate adversarial examples which are classified as a target class of choice.

In practice, more powerful attack goals require more knowledge about the victim, i.e., the model under attack. In a *black-box* scenario, the adversary knows nothing about the model's architecture or the data used for training the model. In its weakest form, only input/output pairs are available. That means the attacker is not able to supply their own input to gain insight. In the *oracle* scenario, the adversary has this ability and is thus capable of modifying inputs to observe the differences in output. Another step up would be access to the training data and/or the ability to collect a surrogate training set.

In the *white-box* scenario, the attacker has knowledge about the model architecture. In the case of neural networks, this would include knowledge about the model parameters, activation functions, and the number of nodes in each layer. Additionally, the adversary can possess knowledge about the training method, e.g. the number of epochs or the optimization algorithm used in the training process.

Most attacks today assume that the adversary has full access to the model and knows its architecture. At first glance, this seems very restricting. If there were no black-box attacks, a defender would be able to secure their model by just keeping information about the model secret. However, as described in Section 2.4, a large portion of adversarial examples transfer, i.e. they are able to deceive models on which they were not crafted. This allows an adversary to use a white-box attack in a black-box scenario. The attacker can train their own model and generate adversarial examples on it which are likely to also fool the target model.

2.2 Attacks

Today, already numerous adversarial attacks have been proposed focusing on different objectives and applications. Some are specifically designed for neural networks while others can be used for many different machine learning techniques. There are also attacks that have been initially developed solely for computer vision tasks but mostly can be easily adjusted for a wider scope of applications. A short introduction to a selected list of attacks is provided below. Due to the rapid development of the field, it is difficult to provide an exhaustive list of attacks or defenses. Interested readers are referred to surveys of Yuan et al. [49] and of Akhtar and Mian [1] which provide a more comprehensive list of attacks and defenses.

2.2.1 Overview

The attacks introduced below differ in many ways. In Table 2.1, an overview of these attacks including their most important characteristics is given.

Additionally, the described algorithms may differ in computational cost and efficiency as well as in the quality of output. The quality of an algorithm's output can be measured in terms of the attack's success rate in crafting adversarial examples and/or in the size of the distortions induced by the attack algorithm. Depending on the goal, different attacks should be used. For example, if one wants to test their model against the

strongest possible adversary and is not constrained by computational power or time, the Carlini-Wagner attack is likely best suited for this task since its perturbations of the input have been shown to be very small [5]. Conversely, if computational efficiency is crucial, the fast gradient sign method is a convincing choice since it is not iterative and only requires one gradient evaluation.

Attack	Setting$^{(*)}$	Un-/Targeted	Iterative	Perturbation norm
L-BFGS	M.P.	Targeted	No	ℓ_2
FGSM	M.L.	Untargeted	No	ℓ_∞
PGD	M.L.	Untargeted	Yes	ℓ_∞
Carlini-Wagner	M.P.	Targeted	No	ℓ_2, ℓ_0, ℓ_∞
DeepFool	M.P.	Untargeted	Yes	ℓ_2
JSMA	M.P.	Targeted	Yes	ℓ_0

Table 2.1: Taxonomy of attacks: (*) M.P. = Minimal Perturbation, M.L. = Maximal Loss; Attacks are categorized according to their original version

2.2.2 L-BFGS attack

Szegedy et al. [40] introduced the first algorithm for finding adversarial examples. Instead of trying to directly solve (MP), the authors adapted the optimization problem as follows:

$$\begin{aligned} \min_{r \in \mathbb{R}^n} \quad & c\|r\|_2 + \mathcal{L}(x+r, l_t) \\ \text{s.t.} \quad & x + r \in \mathbb{X}, \end{aligned} \tag{2.2.1}$$

for $c \in \mathbb{R}$, where $\mathcal{L}$ is the loss function. The constraint $f(x+r) = l_t$ (MP) is replaced by adding the loss value $\mathcal{L}(x+r, l_t)$ of the adversarial example $x+r$ with respect to the target label l_t to the objective function. The parameter c then controls a weigh-off between the ℓ_2-norm of the perturbation and the loss of the adversarial example. Too small values for c will result in large distortions which should be avoided since we want them to be imperceptible for humans. Conversely, too large values for c will likely yield perturbations that will not fool the network, rendering them useless. Therefore, a line search is performed to find the minimum $c > 0$ for which the minimizer r of (2.2.1) satisfies $f(x+r) = l_t$. In other words, (2.2.1) is solved repeatedly for different values of c using box constrained L-BFGS, each time updating c with bisection search. Since the loss function is in general non-convex for neural networks, this method yields only an approximation of a solution for (MP).

2.2.3 Fast Gradient Sign Method (FGSM)

Introduced by Goodfellow et al. [15], the fast gradient sign method algorithm has three main differences to the L-BFGS attack algorithm. Firstly, it computes an adversarial example under the ℓ_∞ distance, and secondly focuses on computational efficiency instead of producing very close adversarial examples. Thirdly, in its original version, it is also, in contrast to the L-BFGS attack, an untargeted attack, i.e., the Fast Gradient Sign Method only aims for a misclassification rather than a classification as a specific label or class. The algorithm is motivated by the authors' hypothesis that it is the linear nature of neural networks that is causing adversarial examples (see Section 2.3). The algorithm works by linearizing the objective function in (ML) around the benign input x yielding a simpler optimization problem,

$$\begin{aligned} \max_{r \in \mathbb{R}^n} \quad & \mathcal{L}(x, l_c) + \nabla_x \mathcal{L}(x, l_c)^T r \\ \text{s.t.} \quad & \|r\|_\infty \leq \epsilon, \end{aligned}$$

with an optimal ℓ_∞-norm solution $r = \epsilon \cdot \operatorname{sign}(\nabla_x \mathcal{L}(x, l_c))$. For a benign input x, corresponding label l_c and $\epsilon > 0$ we obtain the fast gradient sign method:

$$x' = x + \epsilon \cdot \operatorname{sign}(\nabla_x \mathcal{L}(x, l_c)). \tag{2.2.2}$$

It is also possible to perform a targeted FGSM attack. For a benign input x, target label l_t with $l_t \neq l_c$ and $\epsilon > 0$, we need to minimize the loss $\mathcal{L}(x, l_t)$ instead of maximizing $\mathcal{L}(x, l_c)$. We now have

$$x' = x - \epsilon \cdot \operatorname{sign}(\nabla_x \mathcal{L}(x, l_t)). \tag{2.2.3}$$

Variants of the FGSM attack that use other ℓ_p-norms instead of the ℓ_∞-norm have also been proposed. For $p \neq \infty$ in an untargeted setting, the algorithm computes

$$x' = x + \epsilon \cdot \frac{\nabla_x \mathcal{L}(x, l_c)}{\|\nabla_x \mathcal{L}(x, l_c)\|_p}.$$

The attack is then referred to as the fast gradient method (FGM).

2.2.4 Projected Gradient Descent (PGD)

The projected gradient descent (PGD) attack introduced by Kurakin et al. [26] as the Basic Iterative Method can be seen as an extension of the fast gradient (sign) Method. For each iteration, a FGM step is performed and the intermediate output is then clipped

to the ϵ-neighborhood around the clean input x. See Algorithm 1 for the pseudocode. Note that both the FGM step and the clipping depend on the norm. In its original version, the PGD attack was proposed with the ℓ_∞-norm, although other norms like the ℓ_2-norm are also possible. A targeted version of this attack follows directly from the targeted version of the fast gradient sign Method described in (2.2.3). Moreover, Kurakin et al. also introduced a special targeted version of the PGD attack which they referred to as the Iterative Least-Likely Class Method. This approach is designed for neural networks and uses, for a clean input x, for each iteration a targeted FGM step with target label $l_{LL} = \arg\min_{l_t} p(l_t|x)$, i.e., the least likely class or label according to the network's prediction.

Algorithm 1 PGD attack [26]

Require: input x with label l_c,
  input variation parameter $\epsilon > 0$,
  step size $\alpha > 0$,
  number of iterations $i_{max} \in \mathbb{N}$
Output: adversarial example x'.
  $x_0 \leftarrow x$
  $i \leftarrow 0$
  $\mathcal{L} \leftarrow$ Cross Entropy Loss
  for $i < i_{max}$ **do**
    $\tilde{x}_{i+1} = x_i + \alpha \cdot \text{sign}(\nabla_x \mathcal{L}(x_i, l_c))$
    $x_{i+1} \leftarrow Clip_{x_0,\epsilon}(\tilde{x}_{i+1})$ ▷ Clip $\tilde{x}_{i+1}$ to ϵ-nbh around the clean input
    $i \leftarrow i + 1$
  end for

2.2.5 Carlini-Wagner

For this attack let's assume a neural network F with softmax output layer and a corresponding classifier f, i.e., $f(x) = \arg\max_{i=1,\dots,m} F(x) = \arg\max_{i=1,\dots,m} \text{softmax}(Z(x))$, where $Z(x) = z$ is the input of the softmax function and referred to as logits.

Originally developed by Carlini and Wagner [6] to overcome the distillation defense technique (see Section 3.1), the Carlini-Wagner-ℓ_2 attack takes a different approach than the FGSM or L-BFGS attack. Note there also exist ℓ_0 and ℓ_∞ variants of the attack which are beyond the scope of this thesis. Trying to avoid the highly non-linear constraint $f(x + r) = l_t$ in (MP), which makes it hard for optimization algorithms to solve (MP)

directly, Carlini et al. introduced an objective function

$$g(x') := \max(\max\{Z(x')_i : i \neq l_t\} - Z(x')_{l_t}, -\kappa),$$

for $\kappa > 0$, such that g satisfies:

$$g(x + r) \leq 0 \iff f(x + r) = l_t.$$

The objective function can be interpreted as a kind of loss function since it is positive as long as $x + r$ is not classified as l_t and only smaller or equal to 0 once the goal is reached. The confidence with which the adversarial example produced by the algorithm gets misclassified can be controlled by adjusting the confidence parameter κ, although its value is usually set to 0. Additionally, the input data is scaled to be within $[0, 1]^n$ to allow for an elegant change of variables. This allows for a reformulation of (MP):

$$\begin{aligned} \min_{r \in \mathbb{R}^n} \quad & \|r\|_2 + c \cdot g(x + r) \\ \text{s.t.} \quad & x + r \in [0, 1]^n. \end{aligned} \tag{2.2.4}$$

The two optimization problems are equivalent in the sense that there exists a $c > 0$ such that an optimal solution to (2.2.4) is also an optimal solution to (MP). Carlini et al. report that, empirically, the best way to choose the constant c is to look via a modified binary search for the smallest value of c for which the resulting solution x^* satisfies $g(x^*) \leq 0$. To circumvent the issues arising in the optimization process from clipping all the coordinates to be within the box, Carlini and Wagner introduced a change of variables

$$r_i = \frac{1}{2}(\tanh(w_i) + 1) - x_i.$$

Because of $-1 \leq \tanh(w_i) \leq 1$, it follows automatically that $0 \leq x_i + r_i \leq 1$, which ensures the validity of the output. Finally we obtain

$$\min_{w \in \mathbb{R}^n} \|\frac{1}{2}(\tanh(w) + 1) - x\|_2^2 + c \cdot g(\frac{1}{2}(\tanh(w) + 1)).$$

This optimization problem can now be solved efficiently by standard optimization algorithms like Adam [22].

The Carlini-Wagner attack finds adversarial examples with lower distortions than most other attacks but is computationally expensive and thus slower than others. Let F be a neural network, $x \in \mathbb{X}$ an input, and $\epsilon > 0$. To show that it is possible to determine

the existence of an adversarial example within a distance ϵ from an input x, Katz et al. [21] encoded F and the constraints regarding ϵ as a set of linear equations and ReLU constraints, and then used the Reluplex algorithm, an extension of the simplex algorithm [9], a solver for linear programs, to attempt to find an adversarial example within a distance ϵ. Reluplex either returns a valid adversarial example or responds that there exists no adversarial example in the ϵ-ball around x. The smallest ϵ such that an adversarial example exists can be approximated up to a desired precision by applying bisection search.

Carlini et al. [5] used Reluplex to generate (up to a desired precision) provably minimally-distorted adversarial examples, i.e., examples that are solutions of (MP). In an experiment on the MNIST data set, they showed that the adversarial examples generated by the Carlini-Wagner attack are on average within 11.6% of optimal when using the ℓ_∞-norm, i.e., the distance from the benign input to the minimally-distorted example was 11.6% lower on average than to the adversarial example found by the Carlini Wagner attack.

2.2.6 DeepFool

The DeepFool method, introduced by Moosavi-Dezfooli et al. [29], is an iterative procedure. It is aiming to solve the following optimization problem for a clean input x with corresponding true label l_c in the ℓ_2-norm,

$$\begin{aligned} \min_{r\in\mathbb{R}^n} \quad & \|r\|_2 \\ \text{s.t.} \quad & f(x+r) \neq l_c, \end{aligned} \tag{2.2.5}$$

for a classifier $f(x) = \underset{i=1,\dots,m}{\arg\max} F_i(x)$ mapping inputs $x \in \mathbb{R}^n$ to a set of labels $\{1, ..., m\}$ and a function $F : \mathbb{R}^n \to \mathbb{R}^m$ determined by the machine learning technique in use. Note that the DeepFool attack is therefore untargeted. Moreover, we can substitute F for f in (2.2.5) and obtain an equivalent formulation of the problem:

$$\begin{aligned} \min_{r\in\mathbb{R}^n} \quad & \|r\|_2 \\ \text{s.t.} \quad & \exists l_t : F_{l_t}(x+r) \geq F_{l_c}(x+r). \end{aligned}$$

To cause a misclassification by the classifier, we need to distort x such that the generated adversarial example crosses one of the zero level sets $\Gamma_i := \{x : F_i(x) - F_{l_c}(x) = 0\}$. Since solving this optimization problem is hard to solve for optimization algorithms due to the high non-linearity of the constraint, the authors proposed an approximating

iterative procedure instead. Motivated by affine linear classifiers, the algorithm linearizes each F_i around x_j, i.e., $F_i(x_j) + \nabla F_i(x_j)^T r$, at each iteration j and projects x_j onto the closest linearized level set $\hat{\Gamma}_i^j := \{x : F_i(x_j) + \nabla F_i(x_j)^T x - F_{l_c}(x_j) - \nabla F_{l_c}(x_j)^T x = 0\}$. See Figure 2.2 for an illustration for x_0. The full method is given in Algorithm 2.

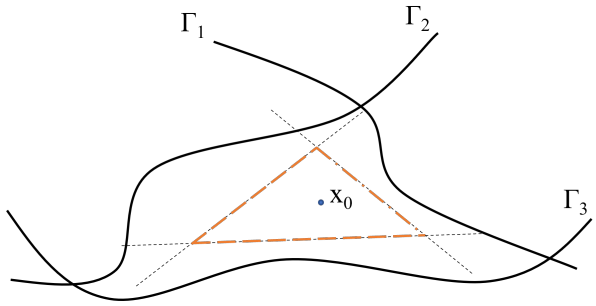


Figure 2.2: For $m = 4$, let x_0 be an input belonging to class 4 and let $\Gamma_i := \{x : F_i(x) - F_4(x) = 0\}$ be the zero level sets. The corresponding linearized level sets are shown in dashed lines and the boundaries of the polyhedron $P_0 = \bigcap_{i=1}^{4} \{x : F_i(x_0) + \nabla F_i(x_0)^T x - F_4(x_0) - \nabla F_4(x_0)^T x \leq 0\}$ in orange.

Adversarial examples generated by DeepFool are widely considered good approximations of the minimal perturbation [6, 50, 28].

Algorithm 2 DeepFool algorithm [29]

Require: input x with label l_c, classifier $f = \arg\max_{i=1,\ldots,m} F_i$
Output: Perturbation r.

$x_0 \leftarrow x$
$i \leftarrow 0$
while $f(x_i) = l_c$ **do**
  for $l \neq l_c$ **do**
    $\omega_l \leftarrow \nabla F_l(x_i) - \nabla F_{l_c}(x_i)$ ▷ $\hat{\Gamma}_l^i = \{x : \Delta_l + \omega_l^T x = 0\}$
    $\Delta_l \leftarrow F_l(x_i) - F_{l_c}(x_i)$
  end for
  $\hat{l} \leftarrow \arg\min_{l \neq l_c} \frac{|\Delta_l|}{\|\omega_l\|_2}$ ▷ Select entry with closest linearized level set $\hat{\Gamma}_{\hat{l}}^i$
  $r_i \leftarrow \frac{|\Delta_{\hat{l}}|}{\|\omega_{\hat{l}}\|_2^2} \omega_{\hat{l}}$ ▷ Project onto $\hat{\Gamma}_{\hat{l}}^i$
  $x_{i+1} \leftarrow x_i + r_i$
  $i \leftarrow i + 1$
end while
return $r = \sum_i r_i$

2.2.7 Jacobian-based Saliency Map Attack (JSMA)

The Jacobian-based Saliency Map Attack (JSMA) introduced by Papernot et al. [33] is an iterative algorithm for finding approximations to solutions of (MP). The algorithm focuses on keeping the number of perturbed input features as small as possible while still deceiving the model under attack. Unlike the L-BFGS method or the Carlini-Wagner attack, JSMA is not using the gradient of the loss. For each iteration, instead of computing the gradient of the network's associated loss function for finding descent directions, it computes the Jacobian $J_F(x)$ of the model F itself, where $F : \mathbb{R}^n \to \mathbb{R}^m$ is a neural network mapping inputs x to output probability vectors y. The Jacobian, also referred to as Forward Derivative by the authors, is then used to construct a map $S_{x,l} : \{1, ..., n\} \to \mathbb{R}$,

$$S_{x,l_t}(i) := \begin{cases} 0 & \frac{\partial F_{l_t}}{\partial x_i}(x) < 0 \text{ or } \sum_{j \neq l_t} \frac{\partial F_j}{\partial x_i}(x) > 0 \\ \left(\frac{\partial F_{l_t}}{\partial x_i}(x)\right) \cdot \left|\sum_{j \neq l_t} \frac{\partial F_j}{\partial x_i}(x)\right| & otherwise, \end{cases} \tag{2.2.6}$$

where l_t is the target label. Inspired by Simonyan et al. [38], who used maps $S_{x,l_t}(i)$ as visualization tools (see Figure 2.3), Papernot et al. refer to them $S_{x,l_t}(i)$ as *saliency* maps. High values of $S_{x,l_t}(i)$ indicate that increasing x_i will either increase F_{l_t}, i.e., the

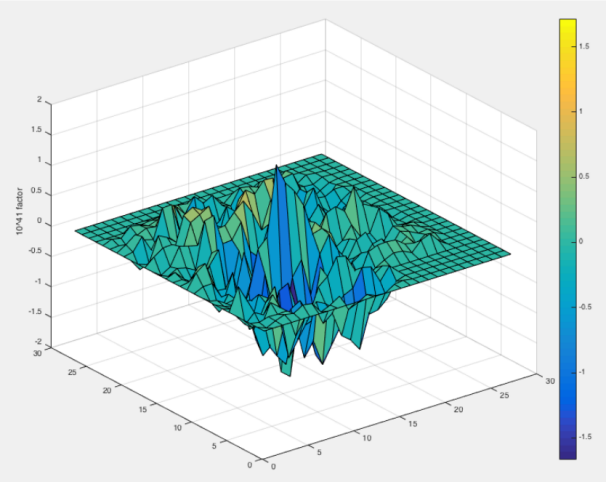


Figure 2.3: Saliency map of a 784-dimensional input to the LeNet architecture [33]. Input features with large absolute values signalize high sensitivity to perturbations.

likelihood of target label l_t, or decrease the likelihood of the other labels, or both.

Remark. Other saliency maps are possible, e.g.,

$$\hat{S}_{x,l_t}(i) := \begin{cases} 0 & \frac{\partial F_{l_t}}{\partial x_i}(x) > 0 \text{ or } \sum_{j \neq l_t} \frac{\partial F_j}{\partial x_i}(x) < 0 \\ \left|\frac{\partial F_{l_t}}{\partial x_i}(x)\right| \cdot \left(\sum_{j \neq l_t} \frac{\partial F_j}{\partial x_i}(x)\right) & \textit{otherwise.} \end{cases} \tag{2.2.7}$$

For this choice, high values of $\hat{S}_{x,l_t}(i)$ indicate input features that an adversary should decrease to achieve misclassification.

In each step, after the computation of the saliency map S_{x',l_t}, where $x' = x + r$ is the current perturbed input, x' is modified by a parameter η along the input feature $i_{max} = \arg\max_i S_{x',l_t}(i)$. The algorithm stops once the target label l_t is predicted, i.e., $l_t = \arg\max_j F_j(x)$ or the maximum distortion, specified by the maximum distortion parameter ϵ, is reached. The maximum distortion parameter ϵ limits the number of features changed. The full method is provided in Algorithm 3. It is worth noting that other variants are possible, e.g., instead of specifying a maximum distortion parameter one could constrain the number of total iterations. Alternatively, one could also use a different saliency map.

Algorithm 3 Jacobian-based saliency map attack (JSMA) [33]

Require: input x with target label l_t
  classifier $f = \arg\max_{i=1,\dots,m} F_i$
  $\eta > 0$
  $\epsilon > 0$
Output: Adversarial sample x'.
  $x' \leftarrow x$
  $r \leftarrow 0$
  while $f(x') \neq l_t$ and $\|r\|_0 < \epsilon$ **do**
    Compute $J_F(x)$
    Compute S_{x,l_t} ▷ saliency map as defined in (2.2.6)
    $i_{max} \leftarrow \arg\max_i S_{x,l_t}(i)$
    Modify $x'_{i_{max}}$ by η
    $r \leftarrow x' - x$
  end while

2.3 Existence of Adversarial Examples

The question why adversarial examples exist for neural networks at all generated a lot of interest and many different possible explanations have been proposed. In this section, we

are going to discuss the most popular hypotheses regarding the existence of adversarial examples. But before that, we need to introduce some new concepts.

Definition 12. Let $f : \mathbb{X} \to \{1, ..., m\}$ be a classifier and let $D_{train}, D_{test} \subset \mathbb{X} \times \{1, ..., m\}$ be the training and test set drawn from a data distribution $\mathscr{D}$.

We say that f *generalizes*, if it, after being trained on D_{train}, scores adequate accuracy on unseen data, e.g., D_{test}, relative to its accuracy on D_{train}. Here, adequate means that the classifier does not suffer from a significant loss in accuracy on the test data compared to its performance on the training data.

For $\epsilon > 0$ and $p \in \mathbb{N} \cup \{\infty\}$, define $B_\epsilon(x) := \{z : \|x - z\|_p \leq \epsilon\}$. We say that f shows ϵ-*local generalization* around an input $x \in \mathbb{X}$ with label $l_c(x)$, if, for $\epsilon > 0$,

$$\underset{r \in B_\epsilon(x)}{\forall} f(x) = l_c(x).$$

In other words, f generalizes ϵ-locally, if for $\epsilon > 0$, f does not change its prediction if x is perturbed with any $r \in \mathbb{R}^n$ satisfying $\|r\|_p < \epsilon$.

While neural networks have been shown to generalize well in most cases, Szegedy et al. found that neural networks do not show ϵ-local generalization around most inputs x for relatively large values of $\epsilon > 0$. Before that neural networks were believed to generally exhibit ϵ-local generalization for reasonably large values of ϵ. The belief in this kind of robustness prior was founded on the empirical evidence that small perturbations typically had no effect on the model predictions.

2.3.1 Linearity Hypothesis

Goodfellow et al. [15] showed that even primitive linear models are susceptible to adversarial examples suggesting that it is not the high non-linearity of models or overfitting that is explaining the existence of these examples. These findings gave rise to the now popular linearity hypothesis. According to Goodfellow et al. [15], it is their extreme linear behaviour as a function of their input that renders neural networks susceptible to adversarial attacks.

For $w \in \mathbb{R}^n$ and $b \in \mathbb{R}$, a separating hyperplane h in $\mathbb{R}^n$ is defined as

$$\begin{aligned} h : \mathbb{R}^n &\to \mathbb{R} \\ x &\mapsto w^T x + b, \end{aligned}$$

and let

$$f : \mathbb{R}^n \to \{-1, 1\}$$
$$x \mapsto \text{sign}(h(x))$$

be a corresponding two-class linear classifier. An illustration for the case of $n = 2$ is given in Figure 2.4. If we perturb a clean input $x \in \mathbb{R}^2$, with $h(x) = c < 0$, with $r \in \mathbb{R}^2$, we can always cause a misclassification as long as $w^T r > -c$. The smallest possible perturbation measured under the ℓ_2-norm that is sufficient to deceive the classifier is found in direction w, i.e. the gradient of $h(x)$.

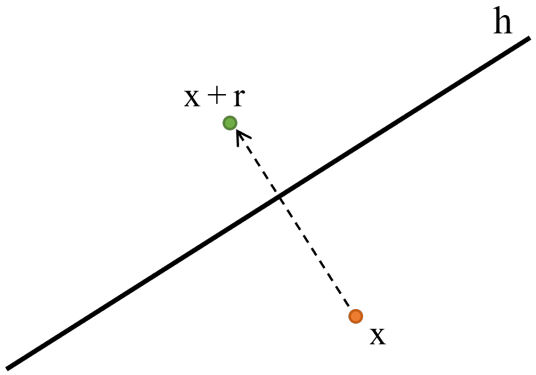


Figure 2.4: Illustration of an adversarial example $x + r$ for a linear classifier $f(x) = \text{sgn}(h(x))$, where $h(x) = w^T x + b$ is a separating hyperplane.

For a neural network F with corresponding loss function $\mathcal{L}$, let $x \in \mathbb{R}^n$ be an input with label $l_c(x)$. Warde-Farley et al. [43] argue that if the loss function $\mathcal{L}(x, l_c(x))$ of F is increasing in a roughly linear fashion in a direction d, the linear behaviour alone is responsible for the existence of adversarial examples. For $d^T r$ large enough, the adversarial example $x + r$ will be misclassified. For a threshold $\kappa \in \mathbb{R}$, the hyperplane $d^T r = \kappa$ divides the input space $\mathbb{R}^n$ into two half-spaces. The linearity hypothesis predicts that most points that are on the same side of the hyperplane as x will be correctly classified. Conversely, nearly all points on the opposite side of the hyperplane are misclassified by the model, according to the hypothesis.

In their work, Warde-Farley et al. [43] provide strong evidence for the existence of half-spaces, mentioned above, using visualizations of two-dimensional subspaces of the input space (see Figure 2.5). They referred to the plots as *church window plots* due to their resemblance to stained glass windows. The plot shows a two-dimensional cross-section of the classification function, in the vicinity of a test example. The cross-section is defined by an adversarial direction specified by the fast gradient sign method and a random

direction. Interested readers are referred to [43] for additional plots of cross-sections exploring two random orthogonal directions and cross-sections defined by an adversarial example and a direction defined by the component of the gradient that is orthogonal to the first direction. Both additional plots support the claim that the linearity hypothesis holds for deep neural networks.

2.3.2 Feature Hypothesis

Ilyas et al. [20] argued that the linearity hypothesis alone does not fully explain the existence of adversarial examples. Since classifiers are usually trained to maximize accuracy, they are incentivized to use any pattern in the training data to do so, even ones that are imperceptible to humans. This suggests that the existence of adversarial examples could be strongly tied to properties of the training set.

In order to be able to further explore this hypothesis, we need to introduce some definitions. Ilyas et al. considered the case of binary classification, where a classifier $f : \mathbb{X} \to \{1, -1\}$ is trained on input-label pairs $(x, l_c) \in \mathbb{X} \times \{1, -1\}$ drawn from a distribution $\mathscr{D}$.

Definition 13. [20]. A *feature* is a function $g : \mathbb{X} \to \mathbb{R}$ mapping from the input space to the real numbers.

The set of all features is denoted by $\mathscr{F} = \{g : \mathbb{X} \to \mathbb{R}\}$. We can now introduce the concept of useful and (non-)robust features:

Definition 14. [20]. Let $\mathbb{E}_{(x,l_c)\sim\mathscr{D}}[g(x)] = 0$ and $\mathbb{E}_{(x,l_c)\sim\mathscr{D}}[g(x)^2] = 1$, then g is called

- a ρ-useful feature (ρ>0) if

$$\mathbb{E}_{(x,l_c)\sim\mathscr{D}}[l_c \cdot g(x)] \geq \rho,$$

  i.e., it is correlated with the true label in expectation. The largest ρ for which g is ρ-useful is denoted by $\rho_{\mathscr{D}}(g)$. Note that if g is negatively correlated, $-g$ is useful.

- a γ-robustly useful feature if g is ρ-useful for some $\rho > 0$ and

$$\mathbb{E}_{(x,l_c)\sim\mathscr{D}}\left[\inf_{r\in S(x)} l_c \cdot g(x+r)\right] \geq \gamma,$$

  i.e., the feature remains γ-useful even under adversarial perturbation (for some set of valid perturbations S).

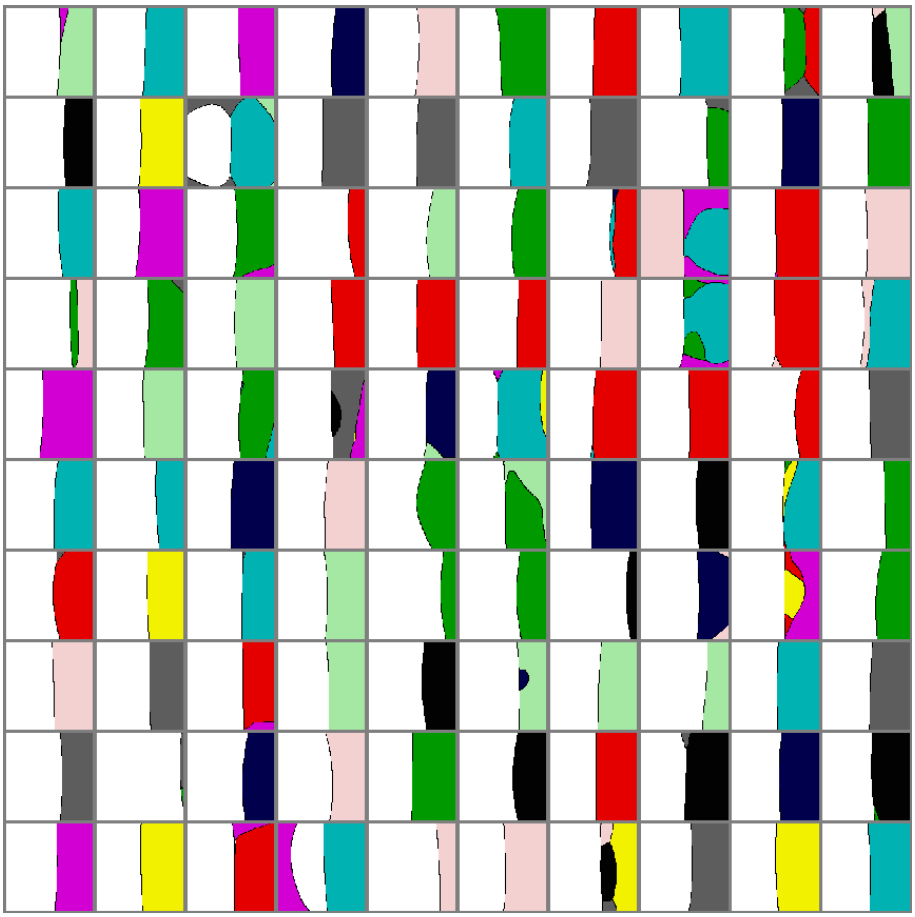


Figure 2.5: Church window plots applied to a convolutional neural network F, trained on the CIFAR-10 data set. Each subplot in the 10x10 grid depicts a church window plot for a different test example. For every test example, each class is assigned a color with white always being the colour of the true class of the example. Each pixel, specified by the coordinates (h, v), is coloured according to the class output $f(x + hu^{(1)} + vu^{(2)})$, where $u^{(1)}$ and $u^{(2)}$ are orthogonal unit vectors spanning a 2-dimensional subspace of $\mathbb{R}^n$. The first unit vector $u^{(1)}$ is generated with the fast gradient sign method while $u^{(2)}$ is a random direction orthogonal to $u^{(1)}$. Both the horizontal h and the vertical coordinate v range from $-\epsilon$ to ϵ ($\epsilon = 0.25$), positioning the benign inputs x at the center of each square. The decision boundaries have mostly roughly linear shapes dividing the input space in half-spaces of correct and incorrect classifications. The benign inputs are usually in the correct region but not far from the decision boundary [43].

- a useful non-robust feature if g is ρ-useful for some $\rho > 0$ but is not a γ-robust feature for any $\gamma \geq 0$.

Remark. The definitions stated above can be straightforwardly generalized to multi-class classification. Let $\mathscr{D}_{l_c}$ denote the distribution of input-label pairs fixed label l_c. In this setting, a feature g is called ρ-useful if

$$\exists_{l_c, l_c' \in \{1,\dots,m\}} \mathbb{E}_{(x,l_c)\sim\mathscr{D}_{l_c}}[g(x)] \geq \rho \wedge \mathbb{E}_{(x,l_c')\sim\mathscr{D}_{l_c'}}[-g(x)] \geq \rho.$$

The definition for (non-)robust features is adapted analogously.

In the framework of [20], a binary classifier f can be written as follows.

Definition 15. [20]. Let $G_f \subseteq \mathscr{F}$ be a finite set of functions $g_{\tilde{\theta}} : \mathbb{X} \to \mathbb{R}$, where $\tilde{\theta} \in \mathbb{R}^{p-|G_f|-1}$ for $p \in \mathbb{N}$. Furthermore, let $w \in \mathbb{R}^{|G_f|}$ be a weight vector, and $b \in \mathbb{R}$ a bias. A binary classifier with parameters $(\tilde{\theta}, w, b) \in \mathbb{R}^p$ is defined as

$$\begin{aligned} f : \mathbb{R}^n &\to \{-1, 1\} \\ x &\mapsto \operatorname{sign}\left(\sum_{g_{\tilde{\theta}} \in G_f} w_{g_{\tilde{\theta}}} \cdot g_{\tilde{\theta}}(x) + b\right). \end{aligned}$$

Once the learning phase is completed, the model parameters stay constant and we say that f has learned the features $g_{\tilde{\theta}} \in G_f$. Note that for a neural network $F : \mathbb{R}^n \to \mathbb{R}$ with associated classifier $f(x) = \operatorname{sign}(F(x))$, the finite set of features G_f learned by f corresponds to the output of the penultimate layer. For example, a given binary classifier f with parameters $(\tilde{\theta}, w, b) \in \mathbb{R}^p$ and a data set of cat images, cat ears can be encoded as a feature g_{ear} which is learned by f during the learning process. This encoding is both dependent on the data set (which influences the calibration of the parameters $\hat{\theta}$) and the model architecture itself. After all, a human perceives ears differently than an artificial neural network. On the other hand, the ear of a cat in a high-resolution coloured image looks different from a low-resolution black-and-white copy. Since we will only use trained models, the dependence of $g_{\tilde{\theta}}$ on $\tilde{\theta}$ will be omitted in the following.

Ilyas et al. found that most standard machine learning data sets admit highly predictive but unrecognizable features. They provided empirical evidence that supports their (feature) hypothesis. They created a robust version of the CIFAR-10 data set by removing non-robust features. To be precise, Ilyas et al. extracted the features learned by an adversarially trained (Section 3.3) neural network classifier f and used them as an

approximation to robust features. They then created a data set serving as a proxy for a distribution $\hat{\mathscr{D}}_R$ that satisfies

$$\mathbb{E}_{(x,l_c)\sim\hat{\mathscr{D}}_R}[l_c \cdot g(x)] = \begin{cases} \mathbb{E}_{(x,l_c)\sim\mathscr{D}}[l_c \cdot g(x)] & g \in G_f \\ 0 & otherwise, \end{cases} \tag{2.3.1}$$

where G_f corresponds to the set of features learned by f. (2.3.1) ensures that only the features extracted from the original adversarially trained remain useful in the modified robust distribution. Interested readers are referred to [20] for more details about the creation of the robust data set. Ilyas et al. also created a second training set for which they used a standard non-robust model instead of an adversarially trained model. The corresponding distribution is denoted by $\hat{\mathscr{D}}_{NR}$.

Finally, Ilyas et al. [20] tested four different model setups, on both standard inputs and adversarial examples. The first model served as a baseline as it was trained on the standard CIFAR-10 data set without adversarial training. The second model was adversarially trained on the standard data set, and the third and fourth model were trained on the robust and non-robust data set respectively, using the standard training procedure. See Figure 2.6 for the results of their experiment. The model trained on the robust data set clearly shows non-trivial adversarial robustness while the model trained on the non-robust data set shows even less robustness than the baseline model.

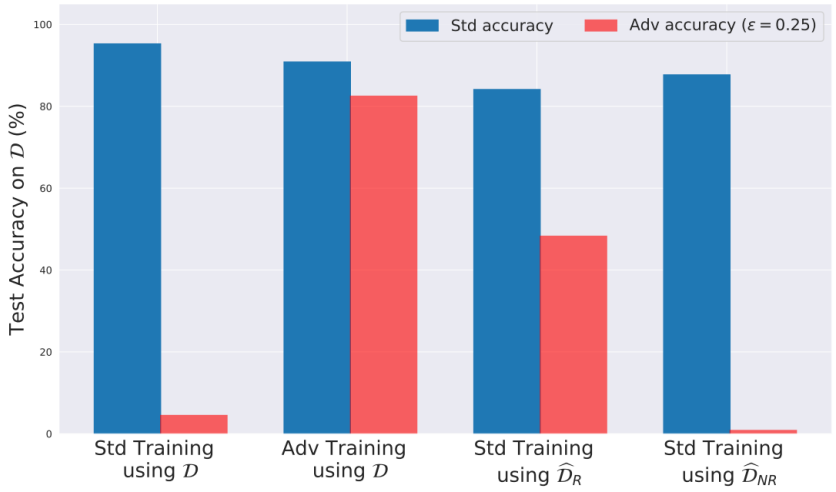


Figure 2.6: Standard and adversarial accuracy on the CIFAR-10 test set of models trained with (i) standard training on the standard CIFAR-10 training set (on $\mathscr{D}$), (ii) adversarial training (on $\mathscr{D}$), (iii) standard training using the robust data set (on $\hat{\mathscr{D}}_R$), (iv) standard training on the non-robust data set (on $\hat{\mathscr{D}}_{NR}$) [20].

2.4 Transferability

Adversarial examples are generated from an input x using a given network f and an attack algorithm. One of their most important properties is their transferability, i.e., it has been shown that there is a high probability that the same adversarial example gets also misclassified by a different model f' [40, 41]. To be precise, Szegedy et al. showed that adversarial examples generalize across models with different hyper-parameters and even across models trained on disjoint training sets. This means that f' can have a different architecture or can be trained on a disjoint training set but adversarial examples crafted on f are still likely to deceive f'.

This phenomenon poses a threat to the secure deployment of machine learning models as it enables black-box attacks. Adversaries can train their own local model on a disjoint training set and still expect to be able to fool the network under attack. For example, if the goal is to attack a neural network f classifying images of cats and dogs, the attacker can use their own model without knowing the architecture of f and train it on their own training set of cat and dog images [32, 33]. Although the success rate of crafting adversarial examples decreases considerably when a different model is used, it is still possible to fool the victim networks in most cases.

Intra-technique transferability refers to the phenomenon where models are susceptible to attacks using adversarial examples crafted on a different model which was trained using the same machine learning technique. Papernot et al. [31] showed that adversarial attacks are applicable not only for deep neural networks but for a variety of machine learning techniques such as logistic regression (LR), k-Nearest Neighbor (kNN), support-vector machines (SVMs) or decision trees (DT). They demonstrated intra-technique transferability for all classifiers mentioned above. For example, logistic regression Classifiers showed high misclassification rates on adversarial samples crafted by other LR classifiers, whereas decision trees seem to be more robust to this approach (see [31] Figure 2 for an overview). The authors also demonstrated that most machine learning techniques are vulnerable to attacks using adversarial samples generated for other machine learning techniques. For example, decision trees were easily fooled by adversarial examples crafted for any other mentioned technique. Deep Neural Networks and kNN classifiers demonstrated more robustness in this setting (see [31] Figure 3 for an overview).

According to Tramer et al. [41], adversarial examples span a subspace of high dimensionality. The authors used the Gradient Aligned Adversarial Subspace (GAAS) technique to estimate the dimensionality of the subspace under a first-order approximation of

the loss function. The technique finds for an input x orthogonal perturbations r with, $\|r\|_2 \leq \epsilon$, that result in a significant increase in loss, i.e., $\mathcal{L}(x+r, l_t) \geq \mathcal{L}(x, l_t) + \gamma$, for a $\gamma > 0$ and target label l_t. To be precise, the goal is to find a maximal set of orthogonal perturbations/directions $r_1, r_2, ..., r_k$ satisfying $\|r_i\|_2 \leq \epsilon$ and

$$\langle r_i, \nabla_x \mathcal{L}(x, y) \rangle \geq \gamma. \tag{2.4.1}$$

The restriction of the angle between r_i and $\nabla_x \mathcal{L}(x, y)$ in (2.4.1) ensures that the increase in loss is large enough. See Figure 2.7 for an illustration of GAAS. In an experiment where

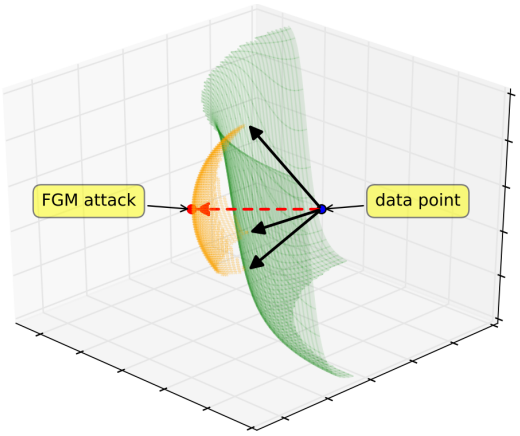


Figure 2.7: Illustration of the Gradient Aligned Adversarial Subspace (GAAS). The red arrow represents a successful FGM attack as it crosses the decision boundary (green). The black arrows are orthogonal perturbations that form a subspace of potential adversarial samples [41].

they trained two fully connected neural networks f_{src} and f_{target} on the MNIST data set [27], Tramer et al. found that for $\|r\|_2 \leq 5$ on average 44.28 orthogonal perturbations existed for f_{src} of which 24.87 directions transferred to model f_{target}. By randomly sampling in the spanned spaces, they were able to fool the source model f_{src} in 99% of cases, and model f_{target} in 89% of cases, suggesting that the perturbations span a dense subspace of adversarial examples. However, Madry et al. [28] observed in experiments that there exist adversarial perturbations r with a negative inner product with the gradient of the input, therefore violating (2.4.1), that successfully fool the model. This suggests that the adversarial subspace view proposed by Tramer et al. is not fully capturing the phenomenon.

Furthermore, Tramer et al. argue that the similarity of the decision boundaries of source and target models, i.e., small inter-boundary distance, could be the cause for the existence

of these large transferable adversarial subspaces. The underlying argument behind this hypothesis is that if compared to the distance to the decision boundary, the distance between the decision boundaries along the direction of a perturbation is very small, adversarial examples crafted from the source model f_{src} that cross the decision boundary of f_{src}, are also very likely to cross the decision boundary of the target model f_{target}. See Figure 2.8 for an illustration. For a given test example of class 1, the inter-boundary distance in the direction of an adversarial example is very small. It is thus very likely that the sample not only crosses the decision boundary of model 1 (source model) but also that of model 2 (target model), in other words, the adversarial sample transfers. For a more detailed formal description, the reader is referred to [41]. Tramer et al. observed

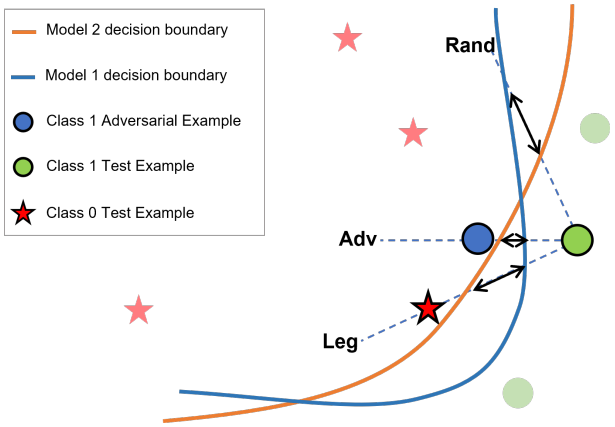


Figure 2.8: Illustration of three different directions. Adv: line in direction of an adversarial example. Leg: line in direction of the closest test point of class 0. Rand: line in a random direction. The black double-ended arrows represent the inter-boundary distance of the two models. Model 1 and model 2 represent the source and target model respectively.

in experiments that the distance to the decision boundary is mostly smaller than the inter-boundary distance of source and target model corroborating the intuition described above. Interestingly, the analysis of the experiments also showed that the minimum distance to the decision boundary was larger in random directions than in direction of test samples from other classes. This could serve as an explanation for the observation that random noise perturbations usually do not change the class prediction by the model.

3 Adversarial Training and other Defenses

Considering the effectiveness of adversarial attacks, countermeasures have to be developed in order to be able to safely deploy neural networks in security-sensitive areas, e.g., in self-driving cars. Many defenses to adversarial attacks have been proposed which at first offered promising results but were later broken by new or adapted attacks. The search for better attacks and defenses created an arms race that is still ongoing. So far, the most promising defense is *adversarial training*. But before focusing entirely on this defense strategy for the rest of the thesis, a short introduction to a selected list of other defense strategies is provided below.

3.1 Defensive Distillation

Distillation was initially introduced as a training procedure that aimed to decrease the computational complexity of neural network architectures [2, 18]. It works by transferring knowledge from a, in terms of number of layers and nodes, larger neural network to a smaller network, which is then called a distilled model.

Defensive distillation suggested by Papernot et al. [34] as a defense technique against adversarial attacks is a variant of distillation. Most adversarial attacks try to calculate a sensitivity estimate for each input dimension which is then used to generate a perturbation. For example, the JSMA method (see Section 2.2.7) directly computes the Jacobian $J_F(x)$ of the network F with respect to its input $x \in \mathbb{R}^n$ in each iteration, thus obtaining the gradient $\nabla_x F_i$ for each output dimension i. The gradients are used for computing sensitivity estimates along each input dimension. Finally, the algorithm modifies the input along the most sensitive dimensions. Defensive Distillation aims at preventing attacks from making use of high gradient amplitudes. For this reason, Papernot et al proposed an adaptation of the softmax output function (1.1.1) for neural networks:

Definition 16. For $z \in \mathbb{R}^m$, define the softmax output function at temperature $T \in \mathbb{R}$

as

$$\psi_T(z) = \left[\frac{e^{\frac{z_i}{T}}}{\sum_{i=1}^{m} e^{\frac{z_i}{T}}} \right]_{i=1,\dots,m}. \tag{3.1.1}$$

The new parameter $T \in \mathbb{R}$ is called *temperature* or *distillation temperature*. Even though defensive distillation is a procedure that is performed during the training phase, where model parameters θ are not fixed, the dependence of F on θ will be omitted in the following for the sake of notation clarity. Let F be a neural network and let $Z(x) \in \mathbb{R}^m$ denote the output of the penultimate layer. If F has a softmax output layer at temperature T, it takes the following form:

$$F(x) = \psi(Z(x)) = \left[\frac{e^{Z_i(x)/T}}{\sum_{i=1}^{m} e^{Z_i(x)/T}} \right]_{i=1,\dots,m}.$$

For each input x the neural network outputs a conditional distribution $F(x) = p(\cdot|x)$ over all labels $i \in \{1, ..., m\}$,i.e., $F_i(x) = p(i|x)$ represents the probability assigned by F that x has label i. Higher values of T force the network to output a more ambiguous distribution, i.e., assigning relatively large probabilities for each class. For $T \to \infty$, one obtains a uniform distribution. The reason for introducing T is that more ambiguous distributions make it harder for an adversary to obtain a good sensitivity estimate as described above. A model's small sensitivity to input variations is encoded in its Jacobian $J_F(x)$. Substituting $g(x) = \sum_{i=1}^{m} e^{\frac{Z_i(x)}{T}}$, we obtain for the (i, j)-th component of the Jacobian of a neural network $F : \mathbb{R}^n \to \mathbb{R}^m$ at temperature T:

$$\begin{aligned}
\frac{\partial F_i(x)}{\partial x_j} &= \frac{\partial}{\partial x_j} \left(\frac{e^{\frac{Z_i(x)}{T}}}{\sum_{l=1}^{m} e^{\frac{Z_l(x)}{T}}} \right) \\
&= \frac{1}{g^2(x)} \left(\frac{\partial e^{\frac{Z_i(x)}{T}}}{\partial x_j} g(x) - e^{\frac{Z_i(x)}{T}} \frac{\partial g(x)}{\partial x_j} \right) \\
&= \frac{1}{g^2(x)} \frac{e^{\frac{Z_i(x)}{T}}}{T} \left(\sum_{l=1}^{m} \frac{\partial Z_i(x)}{\partial x_j} e^{Z_l(x)/T} - \sum_{l=1}^{m} \frac{\partial Z_l(x)}{\partial x_j} e^{Z_l(x)/T} \right) \\
&= \frac{1}{T} \frac{e^{\frac{Z_i(x)}{T}}}{g^2(x)} \left(\sum_{l=1}^{m} \left(\frac{\partial Z_i(x)}{\partial x_j} - \frac{\partial Z_l(x)}{\partial x_j} \right) e^{Z_l(x)/T} \right).
\end{aligned} \tag{3.1.2}$$

Note that the partial derivatives are inversely proportional to the temperature of the model and the logits Z_i are scaled by T before being exponentiated. It becomes clear from (3.1.2) that an increasing temperature is reducing gradient amplitudes for each output dimension and thus decreasing model sensitivity to small input variations.

In the first step of defensive distillation, an initial network is trained on the training data at a temperature T. The output distributions are then used as (one-hot encoded) labels to train a second network again at temperature T, which is then referred to as the distilled network. Figure 3.1 gives an overview of the procedure. After finishing the training process, the temperature of the softmax output layer of the distilled network is set back to 1. Papernot et al. reported that defensive distillation is effective against the

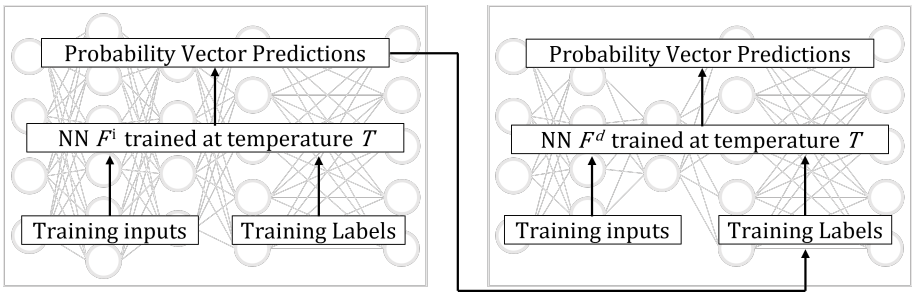


Figure 3.1: Illustration of Defensive Distillation [34]. In the first step, the initial larger neural network (NN) F^i is trained on the labeled training data at Temperature T. The probability vector predictions, i.e., the output of F^i, are then used as labels paired with the initial training inputs for training the distilled neural network F^d. In contrast to standard distillation, usually the same architecture is used for both models.

Jacobian-based saliency map attack and the fast gradient sign method [34, 30]. However, Carlini et al. [6] showed that the Carlini-Wagner attack (see Section 2.2.5) is an effective counter against defensive distillation.

3.2 Feature Squeezing

Feature squeezing, originally proposed by Xu et al. [48], is in contrast to defensive distillation or adversarial training, a reactive defense, i.e., it is deployed after the training phase and aims at detecting adversarial examples. The authors proposed two simple methods for squeezing input features: decreasing the color depth of each pixel and reducing the differences between individual pixels via spatial smoothing. For black-and-white images, Xu et al. used a binary filter as an example for the color depth method. The pixel range is split in half and the pixels of each feature of an image is set to either 0 (black) or 1 (white) depending on which half the original pixel value is in. For multi-channel, i.e. coloured, images, the color depth of each pixel is reduced, e.g., from 8-bit depth (per RGB channel) to 4-bits on the CIFAR-10 [25] data set. The

second method, spatial smoothing, can be divided into two subgroups, local and non-local smoothing. Both were originally designed to reduce image noise but they differ in their strategies. Local smoothing replaces each pixel with the mean of the pixels in a predefined vicinity of it. Non-local means, representing non-local smoothing, is not restricted to a close neighborhood. For each pixel in a given image, the method replaces it with the mean over all pixels weighted by a similarity measure. An illustration of the method as proposed in [48] is provided in Figure 3.2. Copies of the original model with inserted

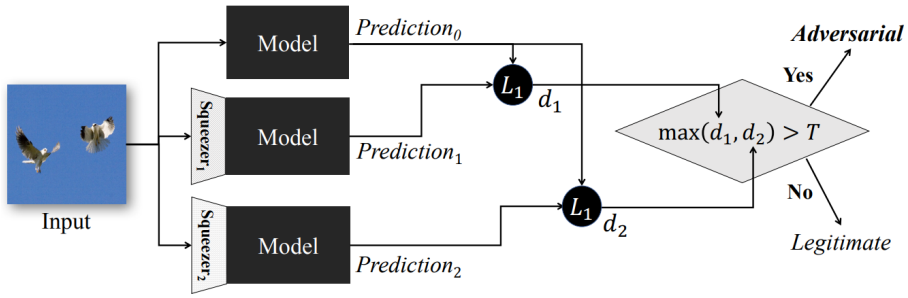


Figure 3.2: Illustration of feature squeezing: The classification model is supported by two copies of the original model with a feature squeezing method inserted before them, in detecting adversarial examples. All three models are evaluated on the input. If the difference d_i in one of the predictions between the original model and one of the squeezer models under the L_1-norm exceeds a threshold, the input is labeled adversarial [48].

feature squeezers are referred to as squeezer models in the following. After feeding the inputs to both the original model and the squeezer models, the output prediction of the classifier is compared with the output of each squeezer model. If one of the output prediction differences under the L_1-norm exceed a threshold $T > 0$, the input is labeled adversarial and consequently rejected.

Since feature squeezing is not performed during training, the technique is compatible with defenses like adversarial training. Xu et al. [48] argued that feature squeezing alone may not be sufficient to provide protection against adversarial attacks but could easily be used in combination with other defenses due to its low computational cost.

3.3 Adversarial Training

Adversarial training is a defense strategy in which the defender incorporates adversarial examples into the learning process of the model by either re-training the model on the

adversarial examples or training the model solely on them in the first place. Thus, adversarial training is also, like defensive distillation, a proactive defense, i.e., the model is robustified against adversarial attacks during the training phase through adaptations of the learning process.

In their work, Madry et al. propose an optimization view on adversarial robustness, i.e., a model's robustness against adversarial attacks [28]. Let $(x, l_c) \in \mathbb{R}^n \times \{1, ..., m\}$ be an input/label pair drawn from a data distribution $\mathscr{D}$. Instead of assuming a standard empirical risk minimization (ERM) setting for the defender, the authors introduce the following saddle point formulation,

$$\min_{\theta} \mathbb{E}_{(x,l_c)\sim\mathscr{D}} \left[\max_{r \in S} \mathcal{L}(\theta, x + r, l_c) \right], \tag{SP}$$

for a set of allowed perturbations $S \subseteq \mathbb{R}^n$. This formulation allows for a unifying perspective on standard ERM and adversarial training. Solving this optimization problem for a given neural network F with parameters $\theta \in \mathbb{R}^p$ and corresponding loss $\mathcal{L}$ would result in an optimally robustified model against adversarial attacks constrained to S.

At first the saddle point problem in (SP) looks intractable due to the highly non-concave inner maximization problem. However, Madry et al. provided evidence that it is indeed tractable by looking at the loss landscape of different models on the MNIST and CIFAR-10 data sets. Their experiments showed that the loss of local maxima of the inner maximization problem obtained through projected gradient descent (PGD) are well-concentrated (see Figure 3.3), despite the distances between local maxima being distributed close to the expected distance in the ℓ_∞-ball, i.e., $S = \{r : \|r\|_\infty \leq \epsilon\}$. In other words, the experiment showed that the loss values of the local maxima are without any outliers and very well concentrated while the local maxima themselves are distributed more evenly in the input space. Not surprisingly, the loss values are considerably lower for adversarially trained networks.

The high concentration of loss values of local maxima obtained through PGD gives rise to the conjecture that training against the PGD adversary (see Section 2.2.4) is providing robustness against all first-order adversaries, i.e., adversaries who use only gradients of the loss function with respect to the input. Moreover, Madry et al. argued that since most optimization problems in machine learning are solved with first-order methods, first-order attacks are universal in the current state of deep learning. Therefore, models trained against PGD adversaries should be robust against *all* other current attacks. Indeed, additional experiments on the MNIST and CIFAR-10 data set confirmed the

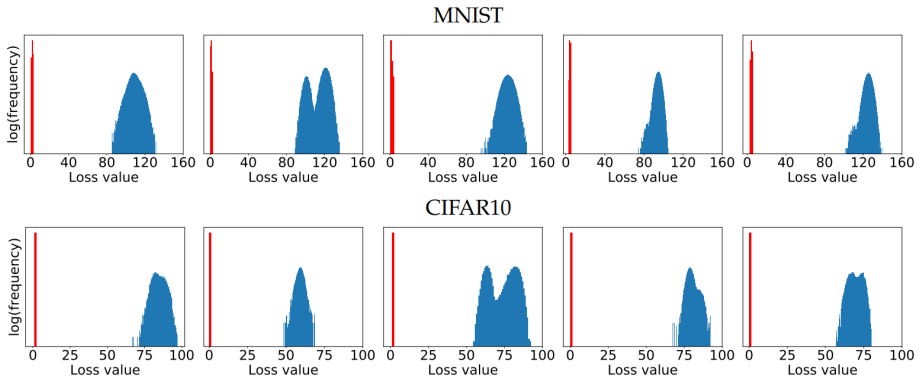


Figure 3.3: For both data sets, each of the five plots shows two histograms of the loss of local maxima obtained by repeatedly starting projected gradient descent (PGD) from 10^5 uniformly distributed random points in the ℓ_∞-ball around an example from the MNIST and CIFAR-10 evaluation sets. The blue histogram corresponds to a normally trained neural network and the red histogram to an adversarially trained network [28].

authors' hypothesis. PGD-trained neural networks showed significant robustness against the FGSM, PGD, and even the stronger Carlini-Wagner attack. Carlini et al. [5] also concluded that adversarial training as described above is effective.

3.3.1 Model Capacity

Another observation Madry et al. made was that model capacity plays a key factor in adversarial robustness. Small capacity models often failed completely at adversarial training, resulting in a classifier that always predicts a fixed class. Conversely, as model capacity increased, the value of (SP) decreased, i.e., the model was better able to fit the adversarial examples.

4 Robust Optimization

4.1 Introduction to Robust Optimization

This section is based on [3].

Robust Optimization (RO) is a field of optimization theory that aims at producing uncertainty-immune solutions to optimization problems with uncertain data. We will use linear programs (LP) to further explain the scope of RO. In RO, an uncertain LP is given by a set of LPs,

$$\{\min_x\{c^T x : Ax \leq b\} : (c, A, b) \in U\}, \tag{4.1.1}$$

where the set U represents the uncertainty set. In this setting, we do not have complete knowledge about the optimization problem at hand, but rather have to deal with uncertainty around the problem parameters (A, c, b) induced by U. According to [3], what it means to solve (4.1.1) is determined by three implicit assumptions regarding the underlying "decision-making environment":

i) *All entries in the decision vector x should get specific numerical values as a result of solving the problem before the actual data "reveals itself".*

ii) *The decision maker is fully responsible for consequences of the decisions to be made when, and only when, the actual data is within the prespecified uncertainty set U.*

iii) *The decision maker cannot tolerate violations of constraints when the data is in U.*

These assumptions arise from a worst-case oriented point of view. For example, a solution x to (4.1.1) must not violate the constraint $Ax \leq b$, for any $(c, A, b) \in U$. This observation leads to the question: What constitutes a solution to an uncertain problem in robust optimization? The best possible robust feasible solution, i.e., a solution satisfying (i)-(iii), is given by

$$\min_x \left\{ \sup_{(c,A,b)\in U} c^T x : Ax \leq b, \forall (c, A, b) \in U \right\},$$

or equivalently by

$$\min_{x,t} \left\{ t : c^T x \leq t : Ax \leq b, \forall (c, A, b) \in U \right\}.$$

In stochastic optimization, the uncertain data (c, A, b) obeys a probability distribution P which is at least in part known in advance. In its simplest form, we have full knowledge of P. In this case, we obtain the following optimization problem,

$$\min_{x,t} \left\{ t : \text{Prob}_{(c,A,b)\sim P}\{c^T x \leq t \ \& \ Ax \leq b\} \geq 1 - \epsilon \right\}, \tag{4.1.2}$$

where $\epsilon \ll 1$ is a given tolerance. In case P is only partially known, i.e., P is restricted to a family $\mathcal{P}$ of probability distributions on the space of the data, we have instead,

$$\min_{x,t} \left\{ t : \text{Prob}_{(c,A,b)\sim P}\{c^T x \leq t \ \& \ Ax \leq b\} \geq 1 - \epsilon, \forall P \in \mathcal{P} \right\}. \tag{4.1.3}$$

Even though these problem formulations of robust optimization and stochastic optimization look very similar, they have significant differences in regards to the assumption made in each setting. SO is less conservative, in other words, violations of assumption (iii) are permitted for a solution of (4.1.2) or (4.1.3). Additionally, a central assumption in stochastic optimization is that the uncertain data obeys a probability distribution which can be (at least partially) identified and used for computation.

RO has found use in many different fields such as decision theory [3] or control theory [11, 44] but we will mainly focus on its application in machine learning.

4.1.1 Robust Optimization in Adversarial Training

We already introduced adversarial training in Section 3.3 as a defense strategy against adversarial attacks. In security-sensitive areas, one usually seeks security guarantees for systems against potential attacks. In the optimal case, an immunity guarantee for a system against *any* attack can be given. To obtain such a guarantee, the defender essentially solves the following constraint satisfaction problem

$$\begin{aligned} &\text{find } \theta \in \Theta \\ &\text{s.t. } f_\theta(x + r) = f_\theta(x) \ , \ x + r \in \mathbb{X}, \quad \forall (x, l_c) \sim \mathscr{D}, \forall r \in S, \end{aligned} \tag{4.1.4}$$

for an input space $\mathbb{X}$, a classifier f_θ and a set of permitted perturbations S. A solution of (4.1.4) would guarantee that no attack, satisfying the restrictions imposed by S, could fool the classifier. Unfortunately, there need not even exist a solution to (4.1.4). But

even if this was the case, the problem turns out to be intractable in practice, since we do not know the underlying data distribution $\mathscr{D}$ but have only access to a finite training set X drawn from $\mathscr{D}$. This observation requires a reformulation of the problem.

For the above-mentioned reasons, the model cannot be expected to be immune to attacks targeting any $(x, l_c) \sim \mathscr{D}$, if only finite amounts of training data are available. A less ambitious but far more attainable goal is to try to minimize how *far off* the model is on average from the correct prediction when classifying inputs $x + r$, for **any** $r \in S$. In this setting, we have a collection of minimization problems,

$$\left\{ \min_{\theta} \left\{ \mathbb{E}_{(x,l_c)\sim\mathscr{D}} \mathcal{L}(\theta, x + r, l_c) \text{ s.t. } x + r \in \mathbb{X} \right\} : r \in S \right\},$$

with a common structure varying in a given set S of allowed adversarial perturbations. The constraint $f_\theta(x + r) = f_\theta(x)$ in (4.1.4) is removed, and instead, we minimize the expected value of the corresponding loss function $\mathcal{L}$. This allows us to approach this problem from the perspective of robust optimization and the best possible feasible solution is given by the solution of

$$\min_{\theta} \left\{ \sup_{r\in S} \mathbb{E}_{(x,l_c)\sim\mathscr{D}} [\mathcal{L}(\theta, x + r, l_c)] \text{ s.t. } x + r \in \mathbb{X}, \right\}. \tag{4.1.5}$$

Another implicit assumption that is often made is that the adversarial perturbations r are small enough that the validity of the adversarial example, i.e., $x + r \in \mathbb{X}$, is ensured for any $x \in \mathbb{X}$. This recovers the saddle point formulation (SP) of adversarial training introduced by Madry et al. [28].

For the scope of adversarial learning, it is assumed that two agents, the defender and the adversary, are participating in an arms raise of attacks and defenses. Provided that the defender is using adversarial training as a defense technique, their goal is to find, for a given model f_θ, optimal weights $\bar{\theta}$ that minimize (4.1.5). On the other hand, the adversary tries to find the best attack algorithm such that the defenses deployed by the defender are broken, i.e., generating perturbations r such that (4.1.5) is maximized. We can therefore treat this process as a two-player zero-sum game, also called a minimax game, which we are going to introduce in the next section.

4.2 Two-player Minimax Games

Definition 17. (Two-player minimax game). Let $f : \Theta \times \Phi \to \mathbb{R}$ be called a value function. In a two-player minimax game or zero-sum game, two players compete against each other, where one player aims at minimizing f, while the other aims at maximizing it. To be precise:

$$\min_{\theta \in \Theta} \max_{\varphi \in \Phi} f(\theta, \varphi).$$

In the case of adversarial training, we have two players, a defender and an adversary. The defender tries to minimize the expected value of the loss, while the adversary tries to maximize it in order to fool the model under attack. More precisely, for a model parameter space Θ and a set of permitted perturbations S, we obtain a two-player minimax game,

$$\min_{\theta \in \Theta} \max_{r \in S} \mathbb{E}_{(x,l_c)\sim\mathscr{D}} \left[\mathcal{L}(\theta, x + r, l_c)\right].$$

In the following section, we are going to describe optimization algorithms specifically designed for minimax games which were also used for the experiments presented in Chapter 6.

4.3 Algorithms

4.3.1 Gradient Descent Ascent

The gradient descent ascent (GDA) algorithm is a straightforward extension of the gradient descent algorithm to minimax games. For a two-player minimax game as described in Definition 17, the update rule of GDA is given by

$$\begin{aligned} \theta^{(t+1)} &= \theta^{(t)} - \eta \nabla_\theta f(\theta^{(t)}, \varphi^{(t)}) \\ \varphi^{(t+1)} &= \varphi^{(t)} + \eta \nabla_\varphi f(\theta^{(t)}, \varphi^{(t)}), \end{aligned}$$

for a learning rate $\eta > 0$, if the iterates of players are updated *simultaneously*. If they are updated *alternatingly*, we have

$$\begin{aligned} \theta^{(t+1)} &= \theta^{(t)} - \eta \nabla_\theta f(\theta^{(t)}, \varphi^{(t)}) \\ \varphi^{(t+1)} &= \varphi^{(t)} + \eta \nabla_\varphi f(\theta^{(t+1)}, \varphi^{(t)}). \end{aligned}$$

For the rest of this section, we are going to assume simultaneous updates. In order to study the convergence performance of GDA in the minimax setting, we will investigate

its behavior on a very simple form of minimax games called bilinear zero-sum games.

Definition 18. (Bilinear zero-sum games [52]). Bilinear zero-sum games can be formulated as the following minmax-problem:

$$\min_x \max_y x^T A y + b^T x + c^T y,$$

for $b \in \mathbb{R}^p$, $c \in \mathbb{R}^n$ and $A \in \mathbb{R}^{p \times n}$ invertible.

As it turns out, GDA does not necessarily converge for bilinear games. For example, let $b = 0$, $c = 0$, $A = I$, then the update rule of GDA becomes

$$\begin{pmatrix} \theta^{(t+1)} \\ \varphi^{(t+1)} \end{pmatrix} = \begin{pmatrix} I_{p\times p} & -\eta I_{n\times n} \\ \eta I_{n\times n} & I_{p\times p} \end{pmatrix} \begin{pmatrix} \theta^{(t)} \\ \varphi^{(t)} \end{pmatrix}.$$

Thus, the following identity holds for the norms of the iterates

$$\|\omega^{(t+1)}\|^2 = (1 + \eta^2)\|\omega^{(t)}\|^2,$$

showing that, for $t \to \infty$, GDA diverges for all choices of $\eta > 0$. This observation motivates the optimization algorithms introduced below.

4.3.2 Extragradient

Originally proposed by Korpelevich [24], the *extragradient* method is designed to overcome the convergence issues GDA is facing for minimax games. Extragradient performs an extrapolation step before updating the weights via GDA. More precisely, we have

$$\text{Extrapolation}: \begin{cases} \theta^{(t+\frac{1}{2})} = \theta^{(t)} - \eta \nabla_\theta f(\theta^{(t)}, \varphi^{(t)}) \\ \varphi^{(t+\frac{1}{2})} = \varphi^{(t)} + \eta \nabla_\theta f(\theta^{(t)}, \varphi^{(t)}) \end{cases}$$

$$\text{Update}: \begin{cases} \theta^{(t+\frac{1}{2})} = \theta^{(t)} - \eta \nabla_\theta f(\theta^{(t+\frac{1}{2})}, \varphi^{(t+\frac{1}{2})}) \\ \varphi^{(t+\frac{1}{2})} = \varphi^{(t)} + \eta \nabla_\theta f(\theta^{(t+\frac{1}{2})}, \varphi^{(t+\frac{1}{2})}). \end{cases}$$

Korpelevich showed that for minimax games, as defined in Section 4.2, with the additional assumption that the objective function $f(\theta, \phi)$ is convex in θ and convex in ϕ, and the partial derivatives of f are Lipschitz continuous, extragradient converges to a saddle point. In case we use SGD instead of gradient descent in the extrapolation and update step, we analogously refer to the resulting method as *ExtraSGD*.

Similarly to extragradient, *ExtraAdam*, proposed by Gidel et al. [13], performs an extrapolation step before updating its weights using Adam [22].

$$\text{Extrapolation}: \begin{cases} \theta^{(t+\frac{1}{2})} = \theta^{(t)} - \eta \frac{\hat{m}_t^\theta}{\sqrt{\hat{v}_t^\theta} + \epsilon} \\ \varphi^{(t+\frac{1}{2})} = \varphi^{(t)} + \eta \frac{\hat{m}_t^\varphi}{\sqrt{\hat{v}_t^\varphi} + \epsilon} \end{cases}$$

$$\text{Update}: \begin{cases} \theta^{(t+\frac{1}{2})} = \theta^{(t)} - \eta \frac{\hat{m}^\theta_{t+\frac{1}{2}}}{\sqrt{\hat{v}^\theta_{t+\frac{1}{2}}} + \epsilon} \\ \varphi^{(t+\frac{1}{2})} = \varphi^{(t)} + \eta \frac{\hat{m}^\varphi_{t+\frac{1}{2}}}{\sqrt{\hat{v}^\varphi_{t+\frac{1}{2}}} + \epsilon}, \end{cases}$$

where $\hat{m}_t^\theta, \hat{m}_t^\varphi$ and $\hat{v}_t^\theta, \hat{v}_t^\varphi$ are the bias-corrected estimates of the first and second moment for both iterates, respectively.

4.3.3 OGDA

Optimistic Gradient Descent Ascent (OGDA), proposed by Rakhlin et al. [35], is an adaptation of the optimistic gradient descent (OGD) algorithm for zero-sum games. In contrast to ExtraSGD or ExtraAdam, OGDA requires only one gradient evaluation per step, while storing the gradient of the previous iteration. The update rules for the iterates are as follows:

$$\begin{cases} \theta^{(t+1)} = \theta^{(t)} - \eta \left(2\nabla_\theta f(\theta^{(t)}, \varphi^{(t)}) + \nabla_\theta(\theta^{(t-1)}, \varphi^{(t-1)}) \right) \\ \varphi^{(t+1)} = \varphi^{(t)} - \eta \left(2\nabla_\varphi f(\theta^{(t)}, \varphi^{(t)}) + \nabla_\varphi(\theta^{(t-1)}, \varphi^{(t-1)}) \right) \end{cases}$$

Optimistic Gradient Descent Ascent also convergences for bilinear zero-sum games [10].

In the next chapter, we are going to discuss an optimization algorithm introduced by Zhang et al. [53], referred to as Lookahead, which has been shown to improve performance over standard optimization algorithms like Adam [22] or stochastic gradient descent (SGD) in training generative adversarial networks (GANs) [14, 7], which also involve solving a minmax-problem.

5 Lookahead Optimizer

The Lookahead optimizer proposed by Zhang et al. [53] is an optimization algorithm that has been shown to be able to outperform standard optimization algorithms like SGD or Adam in machine learning tasks. Intuitively, the optimizer is using a set of weights obtained from another optimization algorithm, called the fast weights, to 'look ahead' k steps and then taking a more conservative step in that direction, yielding a second set of weights, called the 'slow' weights. In its inner loop, i.e., when looking ahead, any standard optimizer, including SGD and Adam, can be used. Furthermore, Zhang et al. reported that the Lookahead algorithm is less dependent on optimal hyperparameters and therefore requires less hyperparameter tuning in comparison to other optimization algorithms used in the domain of machine learning.

5.1 Algorithm

This section is based on [53].

Formally, Lookahead (LA) maintains two sets of weights, the fast weights θ and the slow weights ϕ. In the inner loop, a specified optimization algorithm $\mathcal{A}$ updates the fast weights θ k times. After that, the slow weights ϕ are updated in the direction of the fast weights through interpolation. In the next iteration, the fast weights are reset to the current slow weights' value. The learning rate of the slow weights is denoted by α, while the learning rate of the fast weights, i.e., the learning rate of the inner optimizer, is denoted by γ. The synchronization parameter k is also referred to as Lookahead-steps (LA-steps) parameter. The pseudocode is provided in Algorithm 4.

For an illustration of the trajectories of both fast and slow weights, see Figure 5.1. In this experiment, SGD displays a rather counterproductive cycling behavior around the minimum. On the other hand, LA overcomes this issue by updating the slow weights with a convex combination of the old slow weights and the latest fast weights which pushes the slow weight trajectory much more in the direction of the minimum.

Algorithm 4 Lookahead Optimizer [53]

Require: initial parameters ϕ_0, objective function L
  LA-steps k, slow weights step size α, inner optimizer $\mathcal{A}$
  for $t = 1, 2, ...$ **do**
    Synchronize parameters $\theta^{(t,0)} \leftarrow \phi^{(t-1)}$
    for i=1,2,...,k **do**
      sample minibatch of data $d \sim \mathscr{D}$
      $\theta^{(t,i)} \leftarrow \theta^{(t,i-1)} + A(L, \theta^{(t,i-1)}, d)$
    end for
    Perform outer update $\phi^{(t)} \leftarrow \phi^{(t-1)} + \alpha(\theta^{(t,k)} - \phi^{(t-1)})$
  end for
Output: parameters ϕ

The fast weights are updated k times within each inner loop according to the following update rule for a given optimizer $\mathcal{A}$, an objective function L and a current minibatch of training examples d:

$$\theta^{(t,i+1)} = \theta^{(t,i)} + \mathcal{A}(L, \theta^{(t,i-1)}, d).$$

Note that the slow weight trajectory can be seen as an exponential moving average (EMA) of the final fast weights of each inner loop. After each inner loop, the slow weights are updated as follows:

$$\begin{aligned}\phi^{(t+1)} &= \phi^{(t)} + \alpha(\theta^{(t,k)} - \phi^{(t)}) \\ &= \alpha[\theta^{(t,k)} + (1-\alpha)\theta^{(t-1,k)} + ... + (1-\alpha)^{t-1}\theta^{(0,k)}] + (1-\alpha)^t\phi_0,\end{aligned}$$

for initial slow weights $\phi^{(0)} = \phi_0$.

5.1.1 Computational Complexity

The computational complexity of LA is only marginally higher than that of its inner optimizer $\mathcal{A}$. After k updates with $\mathcal{A}$, a single update step of the outer weights is performed. In total, LA requires $\mathcal{O}(\frac{k+1}{k})$-times the number of operations of the inner optimizer.

5.1.2 Lookahead-Minmax

We can straightforwardly adapt LA for two-player minimax games (see Algorithm 5). For the sake of clarity in notation, we will denote the slow weights as ξ_θ and ξ_φ for both players, respectively.

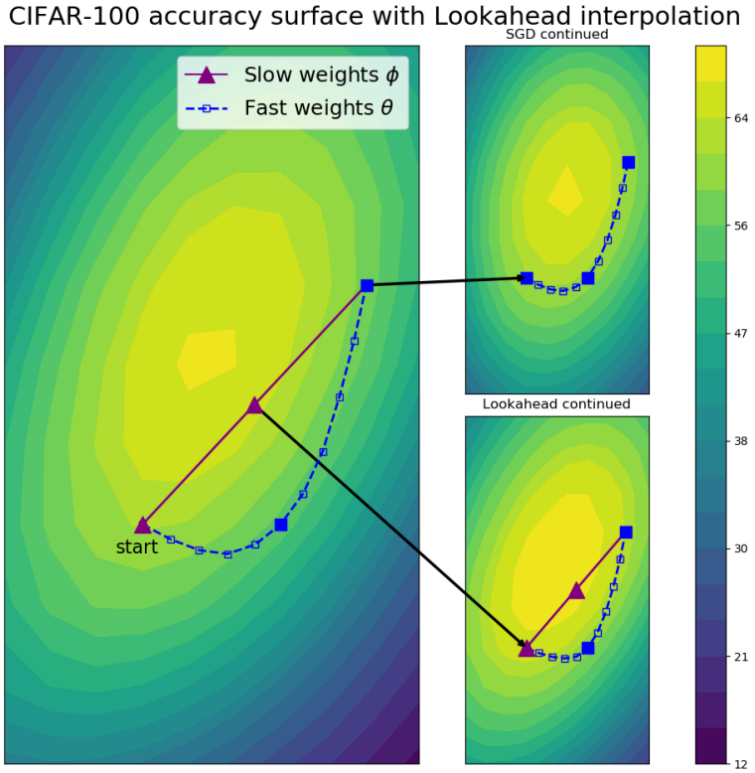


Figure 5.1: Illustration of Lookahead: CIFAR-100 accuracy surface for a standard deep neural network trained with the LA optimizer ($k = 10$) equipped with SGD for its inner loop. The weights are projected on the plane defined by the first, middle, and last fast weight in the inner loop. The trajectory of fast and slow weights are respectively shown in blue and purple. Points that lie on the plane are drawn solid. The upper right-hand figure shows a continuation of the fast weight trajectory without the interpolation step. The lower right-hand figure shows a full second LA iteration [53].

5.2 Convergence Statements

5.2.1 Noisy Quadratic Convergence

For $\mathbb{X} \subseteq R^n$, let $F_\theta : \Theta \times \mathbb{X} \to \mathbb{R}^m$ be a neural network parametrized by $\theta \in \Theta \subseteq \mathbb{R}^p$, with a corresponding loss function $\mathcal{L} : \Theta \times \mathbb{X} \times \{1, ..., m\} \to \mathbb{R}^+$. The input-label pairs $(x, l_c) \in \mathbb{X} \times \{1, ..., m\}$ are drawn i.i.d from a data distribution $\mathscr{D}$. In empirical risk minimization (ERM), the objective is to minimize the expected loss $\mathbb{E}_{(x,l_c)\sim\mathscr{D}}[\mathcal{L}(\theta, x, l_c)]$. Each sample pair (x, l_c) contributes a per-sample loss $\mathcal{L}(\theta, x, l_c)$ to the expected loss $\mathbb{E}_{(x,l_c)\sim\mathscr{D}}[\mathcal{L}(\theta, x, l_c)]$.

Algorithm 5 LA-Minmax [7]

Require: Initial weights $\xi_0^\theta, \xi_0^\varphi$, objective function f
LA steps k, slow weights step size α, inner optimizer $\mathcal{A}$
$\theta^{(0,0)}, \varphi^{(0,0)} \leftarrow \xi_0^\theta, \xi_0^\varphi$
for $t = 1, 2, ..., T-1$ **do**
Synchronize parameters $\theta^{(t,0)} \leftarrow \xi_\theta^{(t-1)}$
Synchronize parameters $\theta^{(t,0)} \leftarrow \xi_\varphi^{(t-1)}$
for i=1,2,...,k **do**
sample minibatch of data $d \sim \mathscr{D}$
$\theta^{(t,i)} \leftarrow \theta^{(t,i-1)} + A(f, \theta^{(t,i-1)}, d)$
$\varphi^{(t,i)} \leftarrow \varphi^{(t,i-1)} - A(f, \varphi^{(t,i-1)}, d)$
end for
Perform outer update:
$\xi^{(t)} \leftarrow \xi_\theta^{(t-1)} + \alpha(\theta^{(t,k)} - \xi_\theta^{(t-1)})$
$\xi^{(t)} \leftarrow \xi_\varphi^{(t-1)} + \alpha(\varphi^{(t,k)} - \xi_\varphi^{(t-1)})$
end for
Output: parameters $\xi_\theta^{(T)}, \xi_\varphi^{(T)}$

Zhang et al. [53] used a noisy quadratic model as a proxy for empirical risk minimization to explore the convergence behavior of LA with SGD as its inner optimizer in a non-deterministic setting. To be precise, the authors used the following model,

$$\hat{\mathcal{L}}(\theta) = \frac{1}{2}(\theta - c)^T A(\theta - c), \tag{5.2.1}$$

for $\theta, c \in \mathbb{R}^p$, $A \in \mathbb{R}^{p \times p}$ positive semi-definite, and $c \sim \mathcal{N}(\theta^*, \Sigma)$ drawn from a normal distribution with mean value $\theta^* \in \mathbb{R}^p$ and covariance matrix $\Sigma \in \mathbb{R}^{p \times p}$. This model serves as a proxy for neural network optimization in a stochastic setting. It is assumed that the per-sample loss function $\mathcal{L}(\theta, x, l_c)$ is smooth and can be locally approximated by $\hat{\mathcal{L}}(\theta)$ (see [36] for further discussion). Instead of sampling input-label pairs from $\mathscr{D}$, random vectors c are drawn i.i.d. from a normal distribution. Note that in general A is dependent on x but for simplicity, A is required to be identical for all samples. For a positive semi-definite matrix A the per-sample optimum is attained at $\theta = c$ and it is assumed that the per-sample optima are normally distributed.

The iterates of the SGD algorithm and the linear interpolation step performed in the LA algorithm are both invariant to translations and rotations in the parameter space. That means that the convergence behavior of SGD and LA is not affected by translations or rotations (for details see [39, Proposition 6.1]). Therefore, without loss of generality, the mean value of c is assumed to be zero, i.e., $\theta^* = 0$, as well as A being diagonal.

Furthermore, [53] required Σ to be diagonal which is a non-trivial assumption but common, see [46, 51]. The diagonal elements of A and Σ are denoted by a_i and σ_i respectively.

We now take the expectation over c instead of (x, l_c) and the objective becomes:

$$\min_{\theta \in \Theta} \mathbb{E}_{c \sim \mathcal{N}(0,\Sigma)}[\hat{\mathcal{L}}(\theta)]. \tag{5.2.2}$$

Using an iterative optimization algorithm like SGD or LA to solve (5.2.2), a new random vector $c^{(t)}$ is introduced at each iteration t, which is independent of the random vectors $c^{(j)}$, for $0 \leq j \leq t-1$, of the previous iterations. Since for SGD, the iterates $\theta^{(t)}$ are a linear combination of the initial weights $\theta^{(0)} = \theta_0 \in \mathbb{R}^p$ and the random vectors $c^{(j)}$, for $0 \leq j \leq t-1$, the following holds

$$\mathrm{Cov}(\theta^{(t)}, c^{(t)}) = 0, \tag{5.2.3}$$

i.e., the iterate $\theta^{(t)}$ and the random vector $c^{(t)}$ are uncorrelated. Furthermore, we obtain for the expected loss at iterate $\theta^{(t)}$:

$$\begin{aligned}
\mathbb{E}_{c^{(t)} \sim \mathcal{N}(0,\Sigma)}[\hat{\mathcal{L}}(\theta^{(t)})] &= \frac{1}{2}\mathbb{E}_{c^{(t)} \sim \mathcal{N}(0,\Sigma)}\left[\sum_i a_i(\theta_i^{(t)} - c_i^{(t)})^2\right] \\
&= \frac{1}{2}\sum_i a_i \mathbb{E}_{c^{(t)} \sim \mathcal{N}(0,\Sigma)}\left[(\theta_i^{(t)} - c_i^{(t)})^2\right] \\
&= \frac{1}{2}\sum_i a_i \left(\mathbb{E}[(\theta_i^{(t)})^2] - 2\underbrace{\mathbb{E}[\theta_i^{(t)} c_i^{(t)}]}_{=0} + \mathbb{E}[(c_i^{(t)})^2]\right) \\
&= \frac{1}{2}\sum_i a_i \left(\mathbb{E}[\theta_i^{(t)}]^2 + \mathbb{V}[\theta_i^{(t)}] + \sigma_i^2\right).
\end{aligned} \tag{5.2.4}$$

Before we can proceed to the main result of this section we need to prove the following Lemma.

Lemma 5.2.1. *Let $\phi^{(t)}$ and $\theta^{(t,k)}$ denote the LA slow weights and fast weights, respectively, after the k-th inner SGD update at the t-th (slow weights) iteration used for minimizing the noisy quadratic function $\hat{\mathcal{L}}$ defined in* (5.2.1)*. Then, the following holds:*

$$\mathrm{Cov}\left(\phi^{(t)}, \theta^{(t,k)}\right) = (I - \gamma A)^k \mathbb{V}\left[\phi^{(t)}\right].$$

Proof. This proof is based on [53, Lemma 1].

Using (5.2.3), we compute recursively for $1 \leq l \leq k$,

$$\begin{aligned}
&\mathrm{Cov}\left(\theta^{(t,l-1)}, \theta^{(t,l)}\right) = \mathrm{Cov}\left(\theta^{(t,l-1)}, (I-\gamma A)\theta^{(t,l-1)} + \gamma A c^{(l-1)}\right) \\
&= \mathbb{E}\left[\theta^{(t,l-1)}\left((I-\gamma A)\theta^{(t,l-1)} + \gamma A c^{(l-1)}\right)^T\right] - \mathbb{E}\left[\theta^{(t,l-1)}\right]\mathbb{E}\left[(I-\gamma A)\theta^{(t,l-1)}\right]^T \\
&= \mathbb{E}\left[\theta^{(t,l-1)}(\theta^{(t,l-1)})^T(I-\gamma A)^T\right] + \underbrace{\mathbb{E}\left[\theta^{(t,l-1)}(c^{(l-1)})^T\right]}_{=0}\gamma A \\
&- (I-\gamma A)\mathbb{E}\left[\theta^{(t,l-1)}\right]\mathbb{E}\left[\theta^{(t,l-1)}\right]^T \\
&= \mathbb{E}\left[(I-\gamma A)\theta^{(t,l-1)}(\theta^{(t,l-1)})^T\right] - (I-\gamma A)\mathbb{E}\left[\theta^{(t,l-1)}\right]\mathbb{E}\left[\theta^{(t,l-1)}\right]^T \\
&= (I-\gamma A)\left(\mathbb{E}\left[\theta^{(t,l-1)}(\theta^{(t,l-1)})^T\right] - \mathbb{E}\left[\theta^{(t,l-1)}\right]\mathbb{E}\left[\theta^{(t,l-1)}\right]^T\right) \\
&= (I-\gamma A)\mathbb{V}\left[\theta^{(t,l-1)}\right].
\end{aligned}$$

Finally, we use that $\theta^{(t,0)} = \phi^{(t)}$ and obtain

$$\mathrm{Cov}\left(\phi^{(t)}, \theta^{(t,k)}\right) = (I-\gamma A)^k \mathbb{V}\left[\phi^{(t)}\right].$$

□

For SGD and LA with SGD as its inner optimizer, Zhang et al. [53] proved the following result which we can then use afterwards for a comparison of the convergence behaviour of SGD and LA in the noisy quadratic setting introduced above.

Proposition 5.2.2. *([53, Proposition 2]). For $L = \max_i a_i$, where a_i are the diagonal entries of A in (5.2.1), let $0 < \gamma < 2/L$ be the learning rate of both SGD and LA. In the noisy quadratic model, the expected value of the iterates of SGD and LA (with inner optimizer SGD) converges to 0 and the variances of the iterates converge to the following fixed points:*

$$V^*_{SGD} := \lim_{t\to\infty} \mathbb{V}[\theta^{(t)}] = \frac{\gamma^2 A^2 \Sigma}{I - (I-\gamma A)^2}$$

$$V^*_{LA} := \lim_{t\to\infty} \mathbb{V}[\phi^{(t)}] = \frac{\alpha^2(I-(I-\gamma A)^{2k}}{\alpha^2(I-(I-\gamma A)^{2k}) + 2\alpha(1-\alpha)(I-(I-\gamma A)^k} V^*_{SGD}.$$

Proof. This proof is based on [53].

The update rule of SGD is given by

$$\theta^{(t+1)} = \theta^{(t)} - \gamma \nabla_\theta(\hat{\mathcal{L}}(\theta^{(t)})) = \theta^{(t)} - \gamma A(\theta^{(t)} - c^{(t)}),$$

with $\theta^{(0)} = \theta_0 \in \mathbb{R}^p$. Treating the weights $\theta^{(t)}$ as random variables, we take the expectation over $c^{(t)}$ and obtain

$$\begin{aligned}\mathbb{E}\left[\theta^{(t+1)}\right] &= (I - \gamma A)\mathbb{E}\left[\theta^{(t)}\right] \\ &= (I - \gamma A)^{t+1}\mathbb{E}[\theta_0] \\ &= (I - \gamma A)^{t+1}\theta_0.\end{aligned} \tag{5.2.5}$$

Since $|(1 - \gamma a_i)|^{t+1} < 1$, $\forall i$, we have:

$$\lim_{t\to\infty} \mathbb{E}\left[\theta^{(t+1)}\right] = \lim_{t\to\infty} (I - \gamma A)^{t+1}\theta_0 = 0.$$

Using (5.2.3), we obtain for the variance,

$$\begin{aligned}\mathbb{V}\left[\theta^{(t+1)}\right] &= \mathbb{V}\left[(I - \gamma A)\theta^{(t)} + \gamma A c^{(t)}\right] \\ &= (I - \gamma A)^2\mathbb{V}\left[\theta^{(t)}\right] + \gamma^2 A^2 \mathbb{V}\left[c^{(t)}\right] + 2(I - \gamma A)\gamma A \underbrace{\mathrm{Cov}(\theta^{(t)}, c^{(t)})}_{=0} \\ &= (I - \gamma A)^2\mathbb{V}\left[\theta^{(t)}\right] + \gamma^2 A^2 \Sigma.\end{aligned} \tag{5.2.6}$$

Due to $|(1 - \gamma a_i)|^{t+1} < 1$, $\forall i$, (5.2.6) is a contraction map, and by Banach's fixed point theorem, it follows that (5.2.6) has a unique fixed point, which can be be obtained directly by solving

$$\begin{aligned}V^*_{SGD} &= (I - \gamma A)^2 V^*_{SGD} + \gamma^2 A^2 \Sigma \\ \implies V^*_{SGD} &= \frac{\gamma^2 A^2 \Sigma}{I - (I - \gamma A)^2}.\end{aligned}$$

Now, we can proceed to prove the corresponding statements for LA. Let $\phi^{(t)}$ denote the slow weights of LA at iteration t and let α be the slow weights step size parameter.

We make use of our computations in (5.2.5) and, for an initial condition $\phi^{(0)} = \phi_0 \in \mathbb{R}^p$, obtain for the expected value of the slow weights at iteration $t + 1$:

$$\begin{aligned}\mathbb{E}\left[\phi^{(t+1)}\right] &= (1 - \alpha)\mathbb{E}\left[\phi^{(t)}\right] + \alpha\mathbb{E}\left[\theta^{(t,k)}\right] \\ &= (1 - \alpha)\mathbb{E}\left[\phi^{(t)}\right] + \alpha(I - \gamma A)^k\mathbb{E}\left[\phi^{(t)}\right] \\ &= \left(1 - \alpha + \alpha(I - \gamma A)^k\right)\mathbb{E}\left[\phi^{(t)}\right] \\ \implies \lim_{t\to\infty} \mathbb{E}\left[\phi^{(t+1)}\right] &= \lim_{t\to\infty} \left(1 - \alpha + \alpha(I - \gamma A)^k\right)^{t+1} \phi_0 = 0.\end{aligned}$$

where $\theta^{(t,k)}$ represents the fast weights after the k-th inner update at the t-th (slow

weights) iteration. For the variance, we can write

$$\mathbb{V}\left[\phi^{(t+1)}\right] = (1-\alpha)^2\mathbb{V}\left[\phi^{(t)}\right] + \alpha^2\mathbb{V}\left[\theta^{(t,k)}\right] + 2\alpha(1-\alpha)\,\mathrm{Cov}\left(\phi^{(t)}, \theta^{(t,k)}\right).$$

By Lemma 5.2.1, we obtain

$$\mathbb{V}\left[\phi^{(t+1)}\right] = (1-\alpha)^2\mathbb{V}\left[\phi^{(t)}\right] + \alpha^2\mathbb{V}\left[\theta^{(t,k)}\right] + 2\alpha(1-\alpha)(I-\gamma A)^k\mathbb{V}\left[\phi^{(t)}\right].$$

Simplifying the equation and substituting the SGD variance formula from (5.2.6) yields,

$$\mathbb{V}\left[\phi^{(t+1)}\right] = \left(1-\alpha+\alpha(I-\gamma A)^k\right)^2\mathbb{V}\left[\phi^{(t)}\right] + \alpha^2\sum_{i=0}^{k-1}(I-\gamma A)^{2i}\gamma^2A^2\Sigma. \tag{5.2.7}$$

Using the identity, $\sum_{i=0}^{k-1} x^i = (1-x^k)/(1-x)$, we, analogously to the SGD case, compute the fixed point of (5.2.7):

$$\begin{aligned}
V_{LA}^* &= \left(1-\alpha+\alpha(I-\gamma A)^k\right)^2 V_{LA}^* + \alpha^2\sum_{i=0}^{k-1}(I-\gamma A)^{2i}\gamma^2A^2\Sigma \\
\implies V_{LA}^* &= \frac{\alpha^2\sum_{i=0}^{k-1}(I-\gamma A)^{2i}}{I-\left(1-\alpha+\alpha(I-\gamma A)^k\right)^2}\gamma^2A^2\Sigma \\
\implies V_{LA}^* &= \frac{\alpha^2(I-(I-\gamma A)^{2k})}{I-\left(1-\alpha+\alpha(I-\gamma A)^k\right)^2}\frac{\gamma^2A^2\Sigma}{I-(I-A)^2} \\
\implies V_{LA}^* &= \frac{\alpha^2(I-(I-\gamma A)^{2k}}{\alpha^2(I-(I-\gamma A)^{2k}) + 2\alpha(1-\alpha)(I-(I-\gamma A)^k}V_{SGD}^*.
\end{aligned} \tag{5.2.8}$$

□

For $\alpha < 1$, the fraction term in the final line of (5.2.8) is smaller than 1. Thus, for the same learning rate γ, LA has a strictly smaller variance fixed point than that of SGD. In (5.2.4), we decomposed the expected loss of the iterates $\theta^{(t)}$ into a linear combination of the expected value and the variance of $\theta^{(t)}$ and the entries σ_i of the covariance matrix Σ of the random vectors $c^{(t)}$. Now, if we insert our results from Proposition 5.2.2 into (5.2.2), we obtain for $t \to \infty$, that LA converges to a smaller expected loss than SGD for the same learning rate. Since we assume a constant learning rate γ, we cannot expect the variance of the iterates to converge to zero.

Additionally, Zhang et al. [53] also compared the convergence rates of LA and SGD in experiments. The authors specified the eigenvalues of A and set $\Sigma = -A$ and computed the expected loss given in (5.2.2) for learning rates in the range $(0, 1)$ for SGD and LA

with $\alpha \in (0, 1]$ and $k = 5$ at time $T = 1000$. By computing the variance fixed point for each learning rate under each optimizer, Zhang et al. were able to compute the optimal loss and plotted the difference between the expected loss at T and the final loss, as a function of the final loss, see Figure 5.2. This experiment allows us to compare the

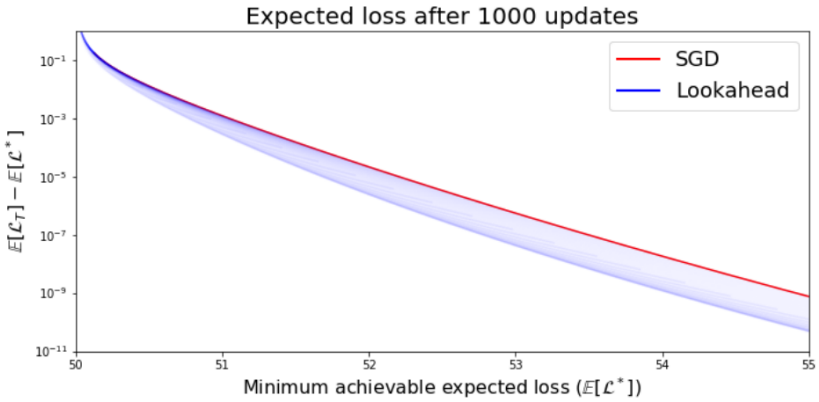


Figure 5.2: Comparison of expected optimization progress between SGD and LA for $k = 5$ on the noisy quadratic model defined in (5.2.1). Each vertical slice compares the convergence of optimizers with the same optimal loss values. For LA, convergence rates for 100 evenly spaced α values in the range $(0, 1]$ are overlaid [53].

convergence performance of SGD and LA optimization settings which converge to the same solution. As shown in Figure 5.2, LA outperformed SGD across a broad range of α values.

5.2.2 Non-Convex Convergence

In the following we are going to analyze the convergence behavior of LA for possibly non-convex objective functions. The analysis is mainly based on [7, 42].

LA as presented in Algorithm 4 with SGD as its inner optimizer involves two nested loops but it can also be written in equivalent form with a single loop. In the single loop case, the fast weights are updated at every iteration l and the slow weights are updated only when $l + 1 \mod k = 0$. Let $\theta^{(l)}$ and $\phi^{(l)}$ denote the fast weights and slow weights respectively after l updates and define the parameter matrix $Q_l = (\theta^{(l)}, \phi^{(l)}) \in \mathbb{R}^{p \times 2}$ and the stochastic gradient matrix $G_l = (g(\theta^{(l)}, S_l), 0) \in \mathbb{R}^{p \times 2}$, where $S_l \subseteq X$ is the subset

drawn in iteration l. Furthermore, define the model synchronization matrix $P_l \in \mathbb{R}^{2\times 2}$ as

$$P_l = \begin{cases} \mu(1,1), & (l+1) \mod k = 0 \\ I, & otherwise, \end{cases}$$

where $\mu = (\alpha, 1-\alpha)^T \in \mathbb{R}^2$ and $\alpha \in [0,1]$ is the slow weights learning rate. Wang et al. [42] introduced a **single loop** update rule for LA:

$$Q_{l+1} = (Q_l - \gamma_l G_l) P_l, \tag{SL}$$

where γ_l is the learning rate of the fast weights at iteration l. Since $(1,1)\mu = 1$ and $I\mu = \mu$, we have $P_l\mu = \mu$. Therefore, if we multiply both sides of (SL) by μ, we obtain

$$\begin{aligned} Q_{l+1}\mu &= (Q_l - \gamma_l G_l) P_l \mu \\ &= Q_l\mu - \alpha\gamma_l g(\theta^{(l)}, S_l) \end{aligned} \tag{5.2.9}$$

Now, define the sequence $\psi^{(l)} := Q_l\mu = \alpha\theta^{(l)} + (1-\alpha)\phi^{(l)}$, $\forall l \geq 0$ and rewrite (5.2.9):

$$\psi^{(l+1)} = \psi^{(l)} - \alpha\gamma_l g(\theta^{(l)}, S_l). \tag{5.2.10}$$

Observe, that for $l \geq 0$:

$$\psi^{(lk)} - \theta^{(lk)} = (1-\alpha)(\phi^{(lk)} - \theta^{(lk)}) = 0$$

$$\psi^{(lk)} - \phi^{(lk)} = \alpha(\theta^{(lk)} - \phi^{(lk)}) = 0.$$

After every k steps, the slow weights, the fast weights and the sequence $\psi^{(l)}$ are equal to each other. Thus, if $\psi^{(l)}$ converges to a stationary point, then $\theta^{(lk)}$ and $\phi^{(lk)}$ converge to it as well. This observation motivates the statements below.

Recall the definition of an unbiased stochastic gradient with bounded variance from Definition 8. We are now ready to prove two theorems about the convergence behavior of LA given in [42].

Theorem 5.2.3. *(Convergence of LA with diminishing learning rate [42]). Suppose the LA-weights are initialized as $\theta^{(0)} = \phi^{(0)} = \psi^{(0)}$, the loss function $\hat{\mathcal{L}}$ is L-smooth and g is an unbiased stochastic gradient of $\hat{\mathcal{L}}$ with bounded variance. If the fast learning rate is*

constant within each inner loop, i.e., $\forall t \geq 0, r \in \{0, ..., k-1\}$,

$$\gamma_{tk+r} = \gamma_{tk}, \tag{5.2.11}$$

and satisfies

$$\sum_{t=0}^{\infty} \gamma_{tk} = \infty, \sum_{t=0}^{\infty} \gamma_{tk}^2 < \infty, \sum_{t=0}^{\infty} \gamma_{tk}^3 < \infty, \tag{5.2.12}$$

$$\alpha\gamma_{tk}L + (1-\alpha)^2\gamma_{tk}^2 L^2 k(k-1) \leq 1, \quad \forall t \geq 0, \tag{5.2.13}$$

then

$$\liminf_{l\to\infty} \mathbb{E}\left[\|\nabla\hat{\mathcal{L}}(\psi^{(l)})\|^2\right] = 0$$

Proof. This proof is based on [42].

Since $\hat{\mathcal{L}}$ is L-smooth, we can apply Lemma 7.1.2(see Appendix) for $v = \psi^{(l+1)}$ and $x = \psi^{(l)}$:

$$\hat{\mathcal{L}}(\psi^{(l+1)}) - \hat{\mathcal{L}}(\psi^{(l)}) \leq \langle\nabla\hat{\mathcal{L}}(\psi^{(l)}), \psi^{(l+1)} - \psi^{(l)}\rangle + \frac{L}{2}\|\psi^{(l)} - \psi^{(l+1)}\|^2.$$

Substituting the update rule given in (5.2.10), we have,

$$\hat{\mathcal{L}}(\psi^{(l+1)}) - \hat{\mathcal{L}}(\psi^{(l)}) \leq -\alpha\gamma_l\langle\nabla\hat{\mathcal{L}}(\psi^{(l)}), g(\theta^{(l)}, S_l)\rangle + \frac{\alpha^2\gamma_l^2 L}{2}\|g(\theta^{(l)}, S_l)\|^2. \tag{5.2.14}$$

In each iteration l a subset S_l of the training set is drawn i.i.d. from $\mathscr{D}$, creating stochastic noise in g. Let $\mathbb{E}_l[\cdot]$ denote the conditional expected value $\mathbb{E}_{S_l\sim\mathscr{D}}[\cdot|\mathcal{F}_l]$, where $\mathcal{F}_l$ is the σ-Algebra of $\psi^{(l)}$ and $\theta^{(l)}$ generated by the stochasticity of g up until iteration l. We then have for the first term on the RHS of (5.2.14),

$$\begin{aligned}
&\mathbb{E}_l\left[\langle\nabla\mathcal{L}(\psi^{(l)}), g(\theta^{(l)}, S_l)\rangle\right] = \langle\nabla\mathcal{L}(\psi^{(l)}), \nabla\mathcal{L}(\theta^{(l)})\rangle \\
=&\frac{1}{2}(\|\nabla\mathcal{L}(\psi^{(l)})\|^2 + \|\nabla\mathcal{L}(\theta^{(l)})\|^2 - \|\nabla\mathcal{L}(\psi^{(l)}) - \nabla\mathcal{L}(\theta^{(l)})\|^2) \\
\geq&\frac{1}{2}(\|\nabla\mathcal{L}(\psi^{(l)})\|^2 + \|\nabla\mathcal{L}(\theta^{(l)})\|^2 - L^2\|\psi^{(l)} - \theta^{(l)}\|^2) \\
=&\frac{1}{2}(\|\nabla\mathcal{L}(\psi^{(l)})\|^2 + \|\nabla\mathcal{L}(\theta^{(l)})\|^2 - (1-\alpha)^2L^2\|\theta^{(l)} - \phi^{(l)}\|^2),
\end{aligned}$$

where the definition of $\psi^{(l)}$ has been used in the last step. For the second term on the

RHS of (5.2.14), using that g is unbiased and has bounded variance, we find that

$$\mathbb{E}_l \left[\| \ g(\theta^{(l)}, S_l)\|^2 \right] \leq \|\nabla \hat{\mathcal{L}}(\theta^{(l)})\|^2 + \sigma^2.$$

Plugging our estimates back into (5.2.14) and taking the total expectation, we obtain

$$\begin{aligned} \mathbb{E}\left[\hat{\mathcal{L}}(\psi^{(l+1)})\right] - \mathbb{E}\left[\hat{\mathcal{L}}(\psi^{(l)})\right] &\leq \frac{\alpha\gamma_l}{2}\mathbb{E}\left[\|\nabla\hat{\mathcal{L}}(\psi^{(l)})\|^2\right] + \frac{\alpha^2\gamma_l^2 L\sigma^2}{2} \\ - \frac{\alpha\gamma_l}{2}(1-\alpha\gamma_l L)\mathbb{E}\left[\|\hat{\mathcal{L}}(\theta^{(l)})\|^2\right] &+ \frac{\alpha\gamma_l(1-\alpha)^2L^2}{2}\mathbb{E}\left[\|\theta^{(l)} - \phi^{(l)}\|^2\right]. \end{aligned} \tag{5.2.15}$$

Without loss of generality, we write the iteration index l as $l = tk + r$, for the outer loop iteration index $t \geq 0$ and inner loop index $0 \leq r < k$. Since we assumed in (5.2.11) that $\gamma_{tk+r} = \gamma_{tk}$, $\forall t \geq 0, 0 \leq r < k$, i.e., the learning rate stays constant within each inner loop, we can sum both sides of (5.2.15) from $l = tk$ to $l = tk + (k-1)$ and obtain an upper bound for the resulting telescoping sum:

$$\begin{aligned} \mathbb{E}\left[\hat{\mathcal{L}}(\psi^{((t+1)k)})\right] - \mathbb{E}\left[\hat{\mathcal{L}}(\psi^{(tk)})\right] \leq &- \frac{\alpha\gamma_{tk}}{2}\sum_{r=0}^{k-1}\mathbb{E}\left[\left\|\nabla\hat{\mathcal{L}}(\psi^{(tk+r)})\right\|^2\right] + \frac{\alpha^2\gamma_{tk}^2 L\sigma^2 k}{2} \\ &- \frac{\alpha\gamma_{tk}}{2}(1-\alpha\gamma_{tk}L)\sum_{r=0}^{k-1}\mathbb{E}\left[\left\|\nabla\hat{\mathcal{L}}(\theta^{(tk+r)})\right\|^2\right] \\ &+ \frac{\alpha\gamma_{tk}(1-\alpha)^2L^2}{2}\sum_{r=0}^{k-1}\mathbb{E}\left[\left\|\theta^{(tk+r)} - \phi^{(tk+r)}\right\|^2\right] \end{aligned} \tag{5.2.16}$$

Recalling that $\theta^{(tk)} = \phi^{(tk)} = \phi^{(tk+r)}, \forall t \geq 0, r < k$ and repeatedly using the identity,

$\left\|\sum_{i=1}^n x_i\right\|^2 \leq n\sum_{i=1}^n \|x_i\|^2$, allows us to bound the last term in (5.2.16) :

$$\begin{aligned}
&\mathbb{E}\left[\left\|\theta^{(tk+r)} - \phi^{(tk+r)}\right\|^2\right] = \mathbb{E}\left[\left\|\theta^{(tk+r)} - \theta^{(tk)}\right\|^2\right] \\
\overset{SGD}{=}\ &\mathbb{E}\left[\left\|\sum_{j=tk}^{tk+r-1} \gamma_{tk} g(\theta^{(j)}, S_j)\right\|^2\right] = \gamma_{tk}^2 \mathbb{E}\left[\left\|\sum_{j=tk}^{tk+r-1} g(\theta^{(j)}, S_j)\right\|^2\right] \\
\leq &\gamma_{tk}^2 2\mathbb{E}\left[\left\|\sum_{j=tk}^{tk+r-1} (g(\theta^{(j)}, S_j) - \nabla\hat{\mathcal{L}}(\theta^{(j)}))\right\|^2 + \left\|\sum_{j=tk}^{tk+r-1} \nabla\hat{\mathcal{L}}(\theta^{(j)})\right\|^2\right] \\
\leq &\gamma_{tk}^2 \left(2r \sum_{j=tk}^{tk+r-1} \mathbb{E}\left[\left\|(g(\theta^{(j)}, S_j) - \nabla\hat{\mathcal{L}}(\theta^{(j)}))\right\|^2\right] + \mathbb{E}\left[\left\|\sum_{j=tk}^{tk+r-1} \nabla\hat{\mathcal{L}}(\theta^{(j)})\right\|^2\right]\right) \\
\leq &\gamma_{tk}^2 \left(2\sigma^2 r + 2\mathbb{E}\left[\left\|\sum_{j=tk}^{tk+r-1} \nabla\hat{\mathcal{L}}(\theta^{(j)})\right\|^2\right]\right) \\
\leq &\gamma_{tk}^2 \left(2\sigma^2 r + 2r \sum_{j=tk}^{tk+r-1} \mathbb{E}\left[\left\|\nabla\hat{\mathcal{L}}(\theta^{(j)})\right\|^2\right]\right),
\end{aligned}$$

where for the penultimate step the boundedness of the stochastic gradient g was used. Summing from $r = 0$ to $r = k - 1$ yields,

$$\begin{aligned}
&\sum_{r=0}^{k-1} \mathbb{E}\left[\left\|\theta^{(tk+r)} - \phi^{(tk+r)}\right\|^2\right] \\
\leq &\sigma^2\gamma_{tk}^2(k-1)k + 2\gamma_{tk}^2 \sum_{r=0}^{k-1} r \sum_{j=tk}^{tk+r-1} \mathbb{E}\left[\left\|\nabla\hat{\mathcal{L}}(\theta^{(j)})\right\|^2\right] \\
= &\sigma^2\gamma_{tk}^2(k-1)k + 2\gamma_{tk}^2 \sum_{r=0}^{k-2} \left(\mathbb{E}\left[\left\|\nabla\hat{\mathcal{L}}(\theta^{(tk+r)})\right\|^2\right] \sum_{s=r+1}^{k-1} s\right) \\
= &\sigma^2\gamma_{tk}^2(k-1)k + \gamma_{tk}^2 \sum_{r=0}^{k-2} \left(\mathbb{E}\left[\left\|\nabla\hat{\mathcal{L}}(\theta^{(tk+r)})\right\|^2\right] (k+r)(k-1-r)\right).
\end{aligned}$$

Note that $(k+r)(k-1-r)$ has its maximum when $r=0$. Therefore, we have

$$\begin{aligned}
&\sum_{r=0}^{k-1} \mathbb{E}\left[\left\|\theta^{(tk+r)} - \phi^{(tk+r)}\right\|^2\right] \\
&\leq \sigma^2 \gamma_{tk}^2 (k-1)k + \gamma_{tk}^2 (k-1)k \sum_{r=0}^{k-2} \mathbb{E}\left[\left\|\nabla \hat{\mathcal{L}}(\theta^{(tk+r)})\right\|^2\right] \\
&\leq \gamma_{tk}^2 (k-1)k \left[\sigma^2 + \sum_{r=0}^{k-1} \mathbb{E}\left[\left\|\nabla \hat{\mathcal{L}}(\theta^{(tk+r)})\right\|^2\right]\right].
\end{aligned}$$

Now we can substitute our upper bound of the last term of (5.2.16) back into (5.2.16) and obtain

$$\begin{aligned}
&\mathbb{E}\left[\hat{\mathcal{L}}(\psi^{((t+1)k)})\right] - \mathbb{E}\left[\hat{\mathcal{L}}(\psi^{(tk)})\right] \\
&\leq -\frac{\alpha\gamma_{tk}}{2} \sum_{r=0}^{k-1} \mathbb{E}\left[\left\|\nabla \hat{\mathcal{L}}(\psi^{tk+r})\right\|^2\right] + \frac{\alpha^2 \gamma_{tk}^2 L \sigma^2 k}{2} \\
&+ \frac{(1-\alpha)^2 \alpha \gamma_{tk}^3 L^2 \sigma^2 k(k-1)}{2} - \frac{\alpha\gamma_{tk} C}{2} \sum_{r=0}^{k-1} \mathbb{E}\left[\left\|\nabla \hat{\mathcal{L}}(\theta^{(tk+l)})\right\|^2\right],
\end{aligned}$$

where $C = 1 - \alpha\gamma_{tk} L - (1-\alpha)^2 \gamma_{tk}^2 L^2 k(k-1)$. Based on the assumption made in (5.2.13), C is non-negative, i.e., $C \geq 0$, and thus,

$$\begin{aligned}
\mathbb{E}\left[\hat{\mathcal{L}}(\psi^{((t+1)k)})\right] - \mathbb{E}\left[\hat{\mathcal{L}}(\psi^{(tk)})\right] \leq &-\frac{\alpha\gamma_{tk}}{2} \sum_{r=0}^{k-1} \mathbb{E}\left[\left\|\nabla \hat{\mathcal{L}}(\psi^{(tk+1)})\right\|^2\right] \\
&+ \frac{\alpha^2 \gamma_{tk}^2 L \sigma^2 k}{2} + \frac{(1-\alpha)^2 \alpha \gamma_{tk}^3 L^2 \sigma^2 k(k-1)}{2}.
\end{aligned}$$

Summing on both sides from $t=0$ to $t=T-1$ yields the following upper estimate for the resulting telescoping sum:

$$\begin{aligned}
\mathbb{E}\left[\hat{\mathcal{L}}(\psi^{(Tk)}) - \hat{\mathcal{L}}(\psi^{(0)})\right] \leq &-\frac{1}{2} \sum_{t=0}^{T-1} \alpha\gamma_{tk} \sum_{r=0}^{k-1} \mathbb{E}\left[\left\|\nabla \hat{\mathcal{L}}(\psi^{(tk+r)})\right\|^2\right] \\
&+ \frac{\alpha^2 L \sigma^2 k}{2} \sum_{t=0}^{T-1} \gamma_{tk}^2 + \frac{(1-\alpha)^2 \alpha L^2 \sigma^2 (k-1)k}{2} \sum_{t=0}^{T-1} \gamma_{tk}^3.
\end{aligned}$$

After rearranging, we have

$$\begin{aligned}
&\frac{1}{Y_T}\sum_{t=0}^{T-1}\gamma_{tk}\frac{1}{k}\sum_{r=0}^{k-1}\mathbb{E}\left[\left\|\nabla\hat{\mathcal{L}}(\psi^{(tk+r)})\right\|^2\right] \\
\leq&\frac{2\mathbb{E}\left[\hat{\mathcal{L}}(\psi^{(0)})-\hat{\mathcal{L}}(\psi^{(Tk)})\right]}{\alpha k Y_T}+\alpha L\sigma^2\frac{\sum_{t=0}^{T-1}\gamma_{tk}^2}{Y_T} \\
&+(1-\alpha)^2L^2\sigma^2(k-1)\frac{\sum_{t=0}^{T-1}\gamma_{tk}^3}{Y_T} \\
\leq&\frac{2(\hat{\mathcal{L}}(\psi^{(0)})-\hat{\mathcal{L}}_{\inf})}{\alpha k Y_T}+\alpha L\sigma^2\frac{\sum_{t=0}^{T-1}\gamma_{tk}^2}{Y_T}+(1-\alpha)^2L^2\sigma^2(k-1)\frac{\sum_{t=0}^{T-1}\gamma_{tk}^3}{Y_T},
\end{aligned} \tag{5.2.17}$$

where $Y_T=\sum_{t=0}^{T-1}\gamma_{tk}$ and $\hat{\mathcal{L}}_{\inf}=\inf_{\theta\in\mathbb{R}^p}\hat{\mathcal{L}}(\theta)$. Recall our assumption made in (5.2.12). Thus, taking the limit $T\to\infty$ on both sides concludes the proof as the RHS tends to 0. □

Theorem 5.2.4. *(Convergence of LA with fixed learning rate [42]). Suppose the LA-weights are initialized as $\theta^{(0)}=\phi^{(0)}=\psi^{(0)}$, the loss function $\hat{\mathcal{L}}$ is L-smooth and g is a stochastic gradient of $\hat{\mathcal{L}}$ with bounded variance. If the fast learning rate is constant within each inner loop, i.e., $\forall t\geq 0$, $r\in\{0,1,...,k-1\}$,*

$$\gamma_{tk+r}=\gamma,$$

and satisfies

$$\alpha\gamma L+(1-\alpha)^2\gamma^2L^2k(k-1)\leq 1,$$

then, for $K\in\mathbb{N}$:

$$\begin{aligned}
\frac{1}{K}\sum_{l=0}^{K-1}\mathbb{E}\left[\|\nabla\hat{\mathcal{L}}(\psi^{(l)})\|^2\right]\leq&\frac{2(\hat{\mathcal{L}}(\psi^{(0)})-\hat{\mathcal{L}}_{inf})}{\alpha\gamma K}+\alpha\gamma L\sigma^2 \\
&+(1-\alpha)^2L^2\gamma^2\sigma^2(k-1),
\end{aligned}$$

where $\hat{\mathcal{L}}_{\inf}=\inf_{\theta\in\mathbb{R}^p}\hat{\mathcal{L}}(\theta)$.

Moreover, if the learning rate is set to $\gamma = \frac{1}{\sqrt{K}}$, we obtain:

$$\begin{aligned}\frac{1}{K}\sum_{l=0}^{K-1}\mathbb{E}\left[\|\nabla\hat{\mathcal{L}}(\psi^{(l)})\|^2\right] \leq & \frac{2(\hat{\mathcal{L}}(\psi^{(0)}) - \hat{\mathcal{L}}_{inf}) + \alpha^2 L\sigma^2}{\alpha\sqrt{K}} \\ & + \frac{(1-\alpha)^2 L^2\sigma^2(k-1)}{K} \\ & = \mathcal{O}\left(\frac{1}{\sqrt{K}}\right).\end{aligned}$$

Proof. This proof is based on [42].

Continuing the proof of Theorem 5.2.3, we fix the fast learning rate, i.e., $\gamma_{tk} = \gamma, \forall t \geq 0$. This changes the upper estimate of (5.2.17) as follows,

$$\begin{aligned}&\frac{1}{kT}\sum_{t=0}^{T-1}\sum_{l=0}^{k-1}\mathbb{E}\left[\left\|\nabla\hat{\mathcal{L}}(\psi^{(tk+r)})\right\|^2\right] = \frac{1}{K}\sum_{l=0}^{K-1}\mathbb{E}\left[\left\|\nabla\hat{\mathcal{L}}(\psi^{(l)})\right\|^2\right] \\ \leq & \frac{2\left(\hat{\mathcal{L}}(\psi^{(0)}) - \hat{\mathcal{L}}_{\text{inf}}\right)}{\alpha\gamma K} + \alpha\gamma L\sigma^2 + (1-\alpha)^2\gamma^2 L^2\sigma^2(k-1),\end{aligned}$$

where $K = Tk$ is the total number of iterations. Finally, setting $\gamma = 1/\sqrt{K}$ yields,

$$\frac{1}{K}\sum_{l=0}^{K-1}\mathbb{E}\left[\left\|\nabla\hat{\mathcal{L}}(\psi^{(l)})\right\|^2\right] \leq \frac{2\left[\hat{\mathcal{L}}(\psi^{(0)}) - \hat{\mathcal{L}}_{\text{inf}}\right] + \alpha^2 L\sigma^2}{\alpha\sqrt{K}} + \frac{(1-\alpha)^2 L^2\sigma^2(k-1)}{K}.$$

□

For the proof of the next theorem we need the following result.

Lemma 5.2.5. *([23], Lemma 17). For any parameters $r_0, b, e, d \geq 0$ there exists constant step size $\gamma \leq \frac{1}{d}$ such that*

$$\frac{r_0}{\gamma(T+1)} + b\gamma + e\gamma^2 \leq 2\left(\frac{br_0}{T+1}\right)^{\frac{1}{2}} + 2e^{\frac{1}{3}}\left(\frac{r_0}{T+1}\right)^{\frac{2}{3}} + \frac{dr_0}{T+1}$$

Proof. For a proof refer to [23]. □

Building on the work of [42], Chavdarova et al. [7] improved the convergence guarantees for LA given in Theorem 5.2.4.

Theorem 5.2.6. *(Improved Convergence of LA with fixed learning rate [7]). Suppose the LA-weights are initialized as $\theta^{(0)} = \phi^{(0)} = \psi^{(0)}$, the loss function $\hat{\mathcal{L}}$ is L-smooth and*

g is a stochastic gradient $\hat{\mathcal{L}}$ of with bounded variance. If the fast learning rate is constant within each inner loop, i.e., $\forall t \geq 0$, $r \in \{0, 1, ..., k-1\}$,

$$\gamma_{tk+r} = \gamma,$$

and satisfies

$$\alpha\gamma L \leq \frac{1}{2}, \quad (1-\alpha)^2\gamma^2 L^2 k(k-1) \leq \frac{1}{2}, \tag{5.2.18}$$

then LA satisfies $\mathbb{E}\|\nabla\mathcal{L}(\omega_{out})\|^2 \leq \epsilon$ after at most

$$\mathcal{O}\left(\frac{\sigma^2}{\epsilon^2} + \frac{1}{\epsilon} + \frac{1-\alpha}{\alpha}\left(\frac{\sigma\sqrt{k-1}}{\epsilon^{\frac{3}{2}}} + \frac{k}{\epsilon}\right)\right)$$

iterations, where ω_{out} denotes a uniformly at random chosen iterate amongst the set $\{\omega^{(0)}, ..., \omega^{(K-1)}\}$ of LA.

Proof. This proof is based on [7].

By (5.2.18), we get that

$$\alpha L\gamma + (1-\alpha)^2\gamma^2 L^2 k(k-1) \leq 1.$$

Therefore, all assumptions of Theorem 5.2.4 are met. We can directly improve the convergence rate,

$$\mathcal{O}\left(\frac{2(\hat{\mathcal{L}}(\psi^{(0)}) - \hat{\mathcal{L}}_{inf})}{\alpha\gamma K} + \alpha\gamma L\sigma^2 + (1-\alpha)^2 L^2\gamma^2\sigma^2(k-1)\right), \tag{5.2.19}$$

given in Theorem 5.2.4, by minimizing the rate in (5.2.19) for γ while respecting the constraint stated in (5.2.18).

Let $\delta_0 = 2(\hat{\mathcal{L}}(\psi^{(0)}) - \hat{\mathcal{L}}_{inf})$. Applying Lemma 5.2.5, for $K = T + 1$, with $r_0 = \frac{\delta_0}{\alpha}$, $b = \alpha L\sigma^2$, $e = (1-\alpha)^2 L^2\sigma^2(k-1)$, and $d = \max\{2\alpha L, \sqrt{2}(1-\alpha)Lk\}$, we have: $\exists\gamma \leq \frac{1}{d}$:

$$\begin{aligned}
&\frac{\delta_0}{\alpha\gamma K} + \alpha L\sigma^2\gamma + (1-\alpha)^2 L^2\sigma^2(k-1)\gamma^2 \\
&\leq 2\left(\frac{\alpha L\sigma^2\delta_0}{\alpha K}\right)^{\frac{1}{2}} + 2\left((1-\alpha)^2 L^2\sigma^2(k-1)\right)^{\frac{1}{3}}\left(\frac{\delta_0}{\alpha K}\right)^{\frac{2}{3}} \\
&+ \frac{d\delta_0}{\alpha K}.
\end{aligned} \tag{5.2.20}$$

We consider each term of the RHS of (5.2.20) separately. For the first term, we have,

$$2\left(\frac{\alpha L\sigma^2\delta_0}{\alpha K}\right)^{\frac{1}{2}} \leq \epsilon \iff c_1\frac{\sigma^2}{\epsilon^2} \leq K,$$

for $c_1 = 4L\delta_0$. Similarly, for the second term we obtain,

$$2\left((1-\alpha)^2L^2\sigma^2(k-1)\right)^{\frac{1}{3}}\left(\frac{\delta_0}{\alpha K}\right)^{\frac{2}{3}} \leq \epsilon \iff c_2\frac{1-\alpha}{\alpha}\frac{\sigma\sqrt{k-1}}{\epsilon^{\frac{3}{2}}} \leq K,$$

for $c_2 = 2^{\frac{3}{2}}L\delta_0$. Lastly, for the third term, we bound d by $2(\alpha L + (1-\alpha)Lk)$, and find that:

$$\frac{d\delta_0}{\alpha K} \leq \epsilon \iff \frac{2\delta_0(\alpha L + (1-\alpha)Lk)}{\alpha\epsilon} \leq K \iff c_3\left(\frac{1}{\epsilon} + \frac{1-\alpha}{\alpha}\frac{k}{\epsilon}\right) \leq K,$$

for $c_3 = 2\delta_0 L$. Plugging our estimates back into (5.2.19) concludes the proof. □

5.2.3 Lookahead Convergence for Minimax Games

This section is based on [7].

In Section 4.3.1 we discussed the gradient descent algorithm in its most common form but we can also define GDA using an operator $G : \mathbb{R}^{(p+n)} \to \mathbb{R}^{(p+n)}$ which performs the updates of the iterates $\omega = (\theta, \varphi)$, i.e., $\omega^{(t)} = G \circ ... \circ G(w^{(0)})$. For GDA, the operator is given by $G(\omega) = \omega - \eta v(\omega)$, for $v(\omega) = (\nabla_\theta f(\theta,\varphi), -\nabla_\varphi f(\theta,\varphi))$. The Jacobian of G is given by $J_G(\omega) = I - \eta J_v(\omega)$.

Definition 19. (Fixed point). Let $G : \mathbb{R}^m \to \mathbb{R}^m$, then a point ω^* is called a fixed point if $G(\omega^*) = \omega^*$. Moreover, a fixed point ω^* is said to be stable if the spectral radius of the Jacobian of G at $\omega = \omega^*$ satisfies $\rho(J_G(\omega^*)) \leq 1$.

Using Theorem 7.1.1 (see Appendix), we can now prove a convergence criterion for LA-Minmax.

Theorem 5.2.7. *(Convergence of LA-Minmax, given converging inner optimizer [7]). Suppose the spectral radius of the Jacobian of the operator G^{base} of the inner optimizer satisfies $\rho(J_{G^{base}}(\omega^*)) < 1$, for a fixed point ω^*, then for $\omega^{(0)}$ in a neighborhood of ω^*, the iterates $\omega^{(t)}$ of LA-Minmax converge to ω^* as $t \to \infty$*

Proof. Let ω^* be a stable fixed point of G^{base}, i.e., $G^{base}(\omega^*) = \omega^*$ and $\rho(J_{G^{base}}(\omega^*)) < 1$.

Then, the operator of LA can be written as

$$\begin{aligned} G^{LA}(\omega) &= \omega + \alpha((G^{base})^k(\omega) - \omega) \\ &= (1-\alpha)\omega + \alpha(G^{base})^k(\omega), \end{aligned}$$

with $\alpha \in (0,1]$, and $k \in \mathbb{N}$, $k \geq 2$. Moreover, the Jacobian $J_{G^{LA}}$ of G^{LA} is consequently given by

$$\begin{aligned} J_G^{LA}(\omega^*) &= (1-\alpha)\omega + \alpha J_{(G^{base})^k}(\omega^*) \\ &= (1-\alpha)I + \alpha J_{(G^{base})^{k-1}}(G^{base}(\omega^*)) J_{G^{base}}(\omega^*) \\ &\vdots \\ &= (1-\alpha)I + \alpha(J_{G^{base}})^k(\omega^*), \end{aligned}$$

where $G^{base}(\omega^*) = \omega^*$ has been used repeatedly.

Now, for $\lambda^{base} \in \sigma(J_{G^{base}}(\omega^*))$ arbitrary but fixed and corresponding eigenvector u, observe that

$$\begin{aligned} J^{LA}u &= ((1-\alpha)I + \alpha(J^{base})^k)(\omega^*)u \\ &= ((1-\alpha) + \alpha(\lambda^{base})^k)u. \end{aligned}$$

It follows that u is also an eigenvector of $J_{G^{LA}}$ with eigenvalue $1 - \alpha + \alpha(\lambda^{base})^k$, and consequently $\sigma(J_{G^{LA}}) = \{1 - \alpha + \alpha\lambda^k : \lambda \in \sigma(J_{G^{base}})\}$.

By assumption, we have $\rho(J_{G^{base}}(\omega^*)) < 1$. Choose $\bar{\lambda}^{LA} \in \sigma(J_{G^{LA}})$ such that $|\bar{\lambda}^{LA}| = \rho(J_{G^{LA}}(\omega^*))$, then

$$\exists \lambda^{base} \in \sigma(J_{G^{base}}) : |\bar{\lambda}^{LA}| = |1 - \alpha + (\lambda^{base})^k|.$$

Consider the set of points $\{z \in \mathbb{C} : \alpha \in (0,1], z = 1 - \alpha + \alpha(\lambda^{base})^k\}$, describing a segment in the complex plane between $1 + 0i$ and $(\lambda^{base})^k$ (excluding $1 + 0i$). Since $|(\lambda^{base})^k| = |\lambda^{base}|^k \leq (\sigma(J_{G^{base}}))^k < 1$, it follows that both ends of the segment are in the unit circle, and therefore $|\bar{\lambda}^{LA}| = \rho(J_{G^{base}}(\omega^*)) < 1$. Applying Theorem 7.1.1 concludes the proof. □

$$\frac{dF_0}{\alpha K} \leq \epsilon \iff \frac{(1-\alpha)k + \alpha}{\alpha(1-\alpha)Lk} \frac{F_0}{\alpha\epsilon} \leq K \iff C\left(\frac{1}{\epsilon} + \frac{(1-\alpha)}{\alpha}\frac{k}{\epsilon}\right) \leq K,$$

for $C = \frac{F_0}{\alpha(1-\alpha)Lk}$.

6 Experiments

This chapter is dedicated to the analysis and evaluation of the experiments that were conducted as part of this thesis[1]. The main question is, whether LA increases the effectiveness of adversarial training. This question can be interpreted and tackled in multiple ways. We will divide the analysis into several parts. First, we are going to analyze LA's robustness to changes in its hyperparameters (γ, k, α). Secondly, we are going to take a closer look at the performance of the slow and fast weights of LA. Lastly, compare the effectiveness of LA compared to other optimizers.

6.1 Framework

The experiments were performed on three different data sets, MNIST, FashionMNIST, and CIFAR-10, and were implemented using the Pytorch deep learning library. In each run of every experiment, batches of 100 inputs have been used to train the models.

6.1.1 Data sets

MNIST

The MNIST data set [27] consists of a training set of 60,000 images of labeled handwritten digits, and a test set of 10,000 images. The scans of the digits are size-normalized and centered in a fixed-size image. For details and an illustration see the Appendix.

FashionMNIST

The FashionMNIST data set [47] consists of a training set of 60,000 images of garments associated with one of 10 classes, and a test set of 10,000 images. Similarly to the MNIST

[1] The code for the experiments is available at `https://github.com/neuhart/Adversarial_Learning_LA_Alg`

data set, the scans of the garments are size-normalized and centered in a fixed-size image. For details and an illustration see the Appendix.

CIFAR-10

The CIFAR-10 data set [25] consists of a training set of 50,000 examples of labeled colour images of objects and animals, and a test set of 10,000 images. For details and an illustration see the Appendix.

6.1.2 Models

The model used for the experiments on the MNIST and FashionMNIST data set, which we will refer to as the MNIST model in the following, is a 5-layer neural network with three convolutional layers with ReLU activation and two fully connected linear layers without activation function (see Appendix, Figure 7.3). For the experiments on CIFAR-10 the resnet-18 model, proposed by He et al. [17], was used.

6.1.3 Optimizers

In total, 10 different optimizer settings were used for the experiments: SGD, Adam, ExtraSGD, ExtraAdam, optimistic gradient descent (OGD), and five instances of LA each equipped with one of the five previous optimization algorithms as the inner optimizer (e.g.: LA with inner optimizer Adam). The group of the five optimizers SGD, Adam, ExtraSGD, ExtraAdam, and OGD will be referred to as the inner optimizers in the following.

6.1.4 Attacks

For the experiments, we used the PGD attack to adversarially train the models, and both the FGSM and PGD attack to evaluate the robustness of the trained models. For both attacks we used the ℓ_∞-norm and set $\epsilon = 0.3$, i.e., we set $S := \{r : \|r\|_\infty \leq 0.3\}$ in (SP) (for details see Section 3.3). For PGD, we allowed 40 iterations with step size $\alpha = 0.01$. Since we followed the argument of Madry et al. [28] that PGD serves as a universal attack, we will only analyze the results for PGD[2].

[2]Interested readers are referred to `https://github.com/neuhart/Adversarial_Learning_LA_Alg` for the corresponding FGSM results.

6.2 Robustness to changes in hyperparameters

Zhang et al. [53] claim that Lookahead improves the robustness of the inner optimizer in regards to changes in the hyperparameters (e.g.: learning rate). Additionally, [53] reports that LA is fairly robust to different choices of the LA-steps parameter k and the learning rate of the slow weights α. We are going to test these statements in an adversarial setting.

In order to analyze the robustness of LA to hyperparameter changes in comparison to other optimizers, we need to train the models for different choices of parameters. For CIFAR-10, we trained the resnet-18 model for 25 epochs on 36 different choices of the parameter triple (γ, k, α) for each of the five LA instances. To be precise, we used the following settings for the LA grid search,

$$\Omega := \{(\gamma, k, \alpha) : \gamma \in \{10^{-2}, 3 \cdot 10^{-3}, 10^{-3}, 3 \cdot 10^{-4}, 10^{-4}, 3 \cdot 10^{-5}\}, \\ k \in \{5, 10\}, \alpha \in \{0.5, 0.75, 0.9\}\}.$$

We also trained the model with each of the inner optimizers separately for 6 different choices of the learning rate $\gamma \in \tilde{\Omega} := \{10^{-2}, 3 \cdot 10^{-3}, 10^{-3}, 3 \cdot 10^{-4}, 10^{-4}, 3 \cdot 10^{-5}\}$. For convenience, in the following, we associate each model with the optimizer it was trained.

Analogously, we performed a grid search for the same hyperparameter values on the MNIST and FashionMNIST model for the same hyperparameter choices. Each model was trained for 12 epochs in both cases. For all three data sets the whole training set was used and the model performance was validated on the whole test set for each data set. The models were trained adversarially, i.e., each element of the training set was perturbed using the PGD attack to create an adversarial example. Moreover, the models were validated both on clean inputs and adversarially perturbed inputs using the FGSM and PGD attack. For convenience, we will refer to the validation accuracy on PGD (FGSM) perturbed inputs as PGD (FGSM) validation accuracy, and to the standard validation accuracy on unperturbed inputs as clean validation accuracy.

Does LA improve robustness to hyperparameter changes?

For CIFAR-10, let $\epsilon^i_{LA}(\gamma, k, \alpha)$ denote the PGD validation accuracy of LA for hyperparameters $(\gamma, k, \alpha) \in \Omega$ at epoch $i \in \{1, ..., 25\}$. Similarly, let $\epsilon^i(\gamma)$ denote the PGD validation accuracy of the corresponding inner optimizer with learning rate $\gamma \in \tilde{\Omega}$ at epoch i. Averaging the PGD validation results over all hyperparameter choices for each

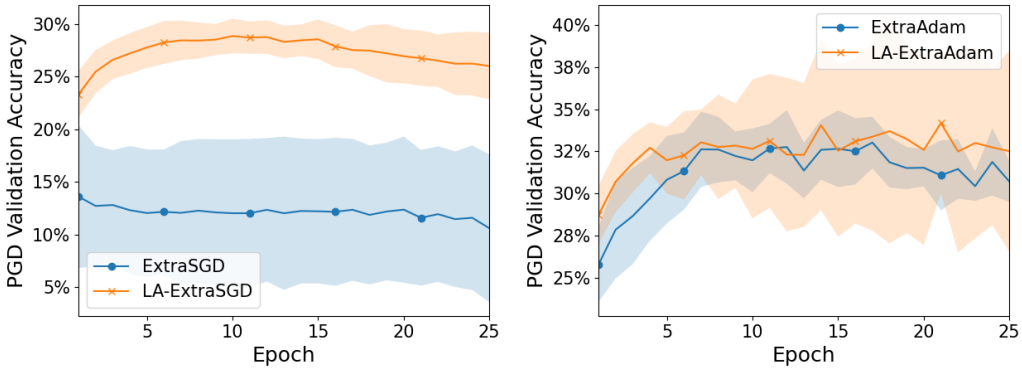
epoch, i.e., computing the mean, for $1 \leq i \leq 25$,

$$\bar{\epsilon}^i_{LA} = \frac{1}{36} \sum_{(\gamma,k,\alpha)\in\Omega} \epsilon^i_{LA}(\gamma, k, \alpha)$$
$$\bar{\epsilon}^i = \frac{1}{6} \sum_{\gamma\in\tilde{\Omega}} \epsilon^i(\gamma),$$

and computing the standard deviations, i.e., for $1 \leq i \leq 25$,

$$s^i_{LA} = \frac{1}{36} \sum_{(\gamma,k,\alpha)\in\Omega} (\epsilon^i_{LA}(\gamma, k, \alpha) - \bar{\epsilon}^i_{LA})^2$$
$$s^i = \frac{1}{6} \sum_{\gamma\in\tilde{\Omega}} (\epsilon^i(\gamma) - \bar{\epsilon}^i)^2,$$

enables us to compare the variability of the PGD validation accuracy in the training phase for each epoch separately (see Figure 6.1). Note that the figures for the inner optimizers



(a) Mean PGD validation accuracy of ExtraSGD (blue) and LA-ExtraSGD (orange), plotted over 25 epochs.

(b) Mean PGD validation accuracy of ExtraAdam (blue) and LA-ExtraAdam (orange), plotted over 25 epochs.

Figure 6.1: The plots show the PGD validation accuracies, i.e., the validation accuracy of the resnet-18 model on the *PGD-perturbed* CIFAR-10 test set, averaged over all hyperparameter settings used in the grid search. The shaded areas in light blue and light orange represent the accuracies with a maximum distance of one standard deviation from the mean values of the standalone inner optimizer and LA equipped with the inner optimizer, respectively.

are obtained by averaging over six different settings since only one hyperparameter, the learning rate γ, has been tuned, while for LA, we average over 36 different hyperparameter settings (γ, k, α). As shown in Figure 6.1, Lookahead did not decrease the PGD validation

accuracy variability for every inner optimizer. The standard deviation of the PGD validation accuracy of the model trained with ExtraAdam is considerably lower than for its LA counterpart for all epochs. However, in the case of ExtraSGD, the variance is decreased significantly and the mean PGD validation accuracy is a lot higher when LA is wrapped around it, i.e., ExtraSGD is used as the inner optimizer of LA. For the corresponding plots for Adam, SGD and OGD see the Appendix, Figure 7.4.

Robustness in the Training Phase

In order to get a measure of the overall sensitivity to changes in the hyperparameters of each optimizer during the training phase, i.e., sensitivity to changes in the hyperparameters of the PGD validation accuracy of the corresponding model, we average over the standard deviations of all 25 epochs, i.e.,

$$\bar{s}_{LA} = \frac{1}{25}\sum_{i=1}^{25} s^i_{LA}$$
$$\bar{s} = \frac{1}{25}\sum_{i=1}^{25} s^i.$$

We observe that LA is more sensitive to changes in hyperparameters than every inner optimizer except ExtraSGD (see Table 6.1). The results suggest that LA is decreasing

in %	SGD	Adam	OGD	ExtraSGD	ExtraAdam
$\bar{s}$	2.1	2.2	2.0	6.6	**1.9**
$\bar{s}_{LA}$	2.3	2.5	2.1	2.1	4.0

Table 6.1: Average standard deviation of the PGD validation accuracy for every optimizer setting (lower is better than higher). The entries of the first row correspond to the average standard deviation for models trained without LA, and the entries of the second row correspond to the average standard deviation of the corresponding LA version.

the variability in PGD validation accuracy during training *only for ExtraSGD* but not for the other inner optimizers. For MNIST and FashionMNIST, we obtain similar results. LA does not provide significant improvements in robustness during the training phase of the model.

Robustness at test time

To evaluate the robustness of each optimizer at test time, we consider only the mean and the standard deviation of the PGD validation accuracy after training completion (see Table 6.2). Again, LA increased the robustness to hyperparameter changes (at test time)

in %	SGD	Adam	OGD	ExtraSGD	ExtraAdam
$\bar{\epsilon}^{25}_{LA}$	30.4 ± 2.2	31.3 ± 2.0	30.0 ± 2.0	10.6 ± 7.0	30.7 ± 1.2
$\bar{\epsilon}^{25}$	30.6 ± 1.9	31.7 ± 1.8	30.7 ± 1.9	26.0 ± 3.2	**32.5** ± 6.0

Table 6.2: Mean PGD validation accuracy (± one standard deviation) averaged over all hyperparameter settings (36 for LA, 6 for inner optimizer) after the training phase.

only for ExtraSGD significantly. Based on these observations, we conclude that using LA does not necessarily improve the robustness of models to changes in hyperparameters.

Is LA more robust to changes in the fast learning rate?

Additionally to considering the robustness to changes in *any* hyperparameter of LA, we will explicitly look at LA's sensitivity to changes in the fast learning rate γ, i.e., the learning rate of the inner optimizer. Assuming that we have tuned the LA steps parameter k and the slow learning rate α, how robust is LA to changes in γ? For this reason, we will take a look at the grid search results for LA with k and α fixed. The values for k and α are taken from the best hyperparameter triple (γ, k, α) of the grid search. We can then plot the PGD validation accuracy for varying values of γ, and compare the results with those of the corresponding standalone inner optimizer (see Figure 6.2). In the case of Adam, LA did not have a stabilizing effect on the performance of the model in an adversarial setting. This also holds for SGD, OGD, and ExtraAdam in our experiments. For ExtraAdam, LA seems to induce even higher sensitivity in regards to hyperparameter changes and higher volatility of the PGD validation accuracy in the training phase (see the Appendix, Figure 7.7). Conversely, for ExtraSGD, the additional use of LA had a huge stabilizing effect as also described in the next section. However, in general, LA does not seem to be more robust to changes in the fast learning rate.

6.2.1 Model Collapse

Can LA prevent model collapse in adversarial training?

Now, given a model trained with LA, if we average only over the LA hyperparameters

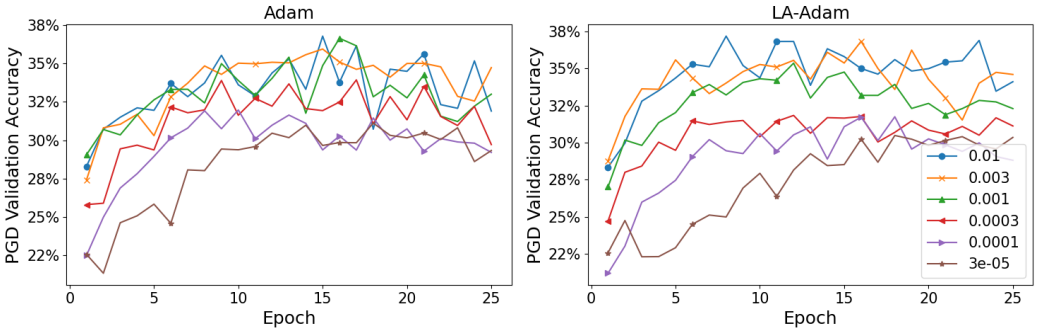


Figure 6.2: PGD Validation Accuracy for Adam (LHS) and LA-Adam (RHS) for six different values of the fast learning rate γ. For LA, we have $k = 10$ and $\alpha = 0.5$. LA-Adam shows no robustness improvements over Adam in regards to changes of the fast learning rate.

(k, α), we obtain a mean PGD validation accuracy for each learning rate, thus, enabling us to compare the robustness to changes in the fast learning rate γ of LA (without tuned k and α) with the robustness of its inner optimizer (see Figure 6.3). Interestingly, LA seems

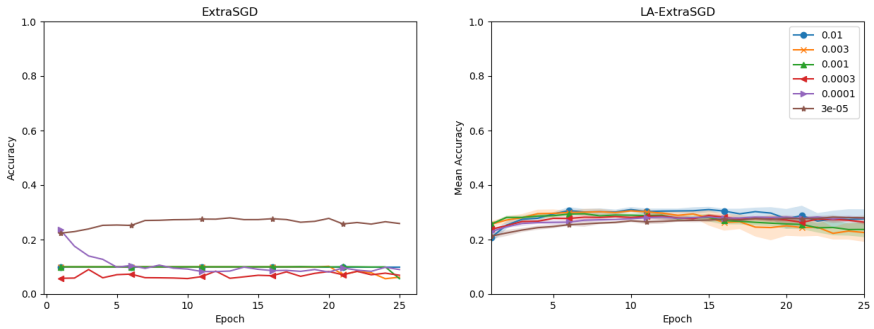


Figure 6.3: The left-hand side plot shows the PGD validation accuracies for ExtraSGD obtained in the grid search. The plot on the right depicts the mean (taken over (k, α)) PGD validation accuracy of LA-ExtraSGD for each fast learning rate γ separately. The shaded areas in the right-hand side plot represent the accuracies with a maximum distance of one standard deviation from the mean values of an inner optimizer and LA equipped with the inner optimizer, respectively.

to be significantly more robust to changes of the learning of the inner optimizer γ for ExtraSGD. On CIFAR-10, LA-ExtraSGD did not only prevent model collapse for tuned k and α but for all values of k and α. Although, it is worth noting that LA-ExtraSGD did not produce competitive results across all three data sets.

In fact, as mentioned before by Madry et al. [28], in adversarial training, models tend to collapse onto one class, i.e., they always predict a certain class independent of the input, when the task of learning to recognize adversarial examples during the training process is too hard for the model. For example, given a data set with 10 equally distributed classes, if the model fails to recognize $\frac{1}{10}$ of adversarial examples during the training phase, it tends to collapse onto one class in order to maintain a minimum accuracy of 10%. This phenomenon can be observed for ExtraSGD, as illustrated in Figure 6.3. For the corresponding plots for Adam, SGD, OGD, and ExtraAdam see the Appendix, Figure 7.5 and Figure 7.6. While for standalone ExtraSGD, the resnet-18 model collapses for all but the smallest learning rate $\gamma = 3 \cdot 10^{-5}$, we find that training with LA-ExtraSGD instead, prevents this behaviour for all six fast learning rates γ, and we observe a significantly higher (average) PGD validation accuracy. We conclude that in the case of ExtraSGD, which is prone to model collapse in adversarial training, LA can prevent models from collapsing onto one class.

6.3 Fast weights vs. Slow weights

Are LA slow weights more stable?

To get a better understanding of the performances of the slow and fast weights within the training process, we plot the PGD validation accuracy of the resnet-18 model with tuned hyperparameters for the first 100 updates in epoch 15 and 25 (see Figure 6.4). The models were trained on adversarial examples generated by the PGD attack. For

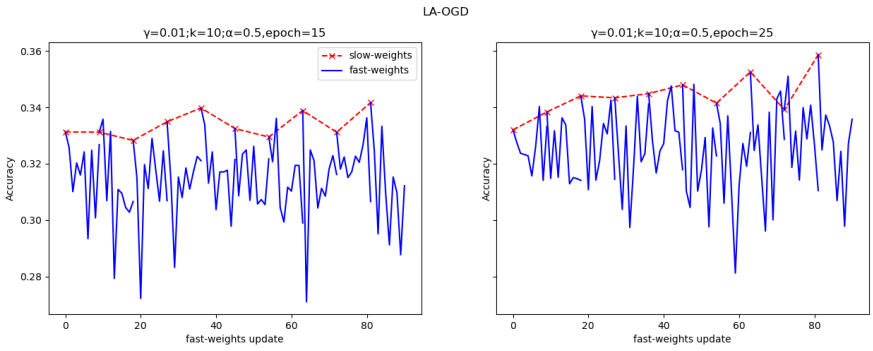


Figure 6.4: Illustration of inner loop versus outer loop PGD validation accuracy of LA-OGD for the first 100 iterates for epoch 15 and 25. In each inner loop (blue lines) the accuracy decreases overall. The accuracy of the iterates in the outer loop (red dashed line), i.e., the slow weights, is a lot more stable.

OGD, we observe that within each inner loop the accuracy drops but is recovered in the convex interpolation step of the outer loop. This pattern indicates that the inner loop exhibits a cycling behavior around a local minimum of the loss function. For the other inner optimizers, we did not observe such a clear pattern.

6.4 PGD Validation Accuracy

Does LA show improved generalization over inner optimizers?

6.4.1 CIFAR-10

In the next step, we used the tuned hyperparameters to adversarially train the resnet-18 model on the CIFAR-10 data set 10 times each. The results suggest that LA can show slight performance advantages over standard optimization algorithms like Adam or SGD in adversarial training but does not yield a significant increase in performance. On the other hand, in the case of ExtraAdam, on average the resnet-18 model achieves lower PGD validation accuracy with LA (see Figure 6.5). In other words, the model trained

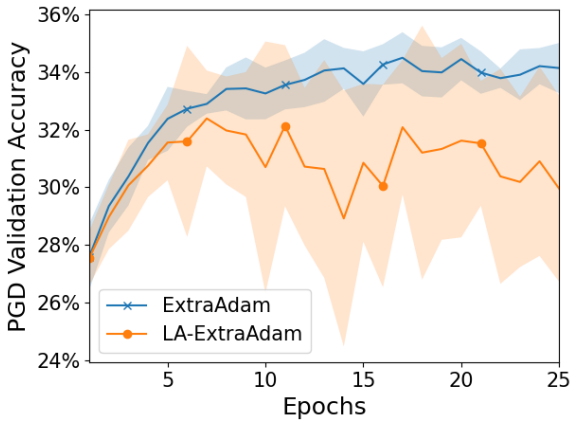


Figure 6.5: Comparison of the Mean PGD validation accuracy after 25 epochs for both standalone ExtraAdam and LA with ExtraAdam. For both optimizers tuned hyperparameters have been used, i.e., $\gamma = 10^{-3}$ and $(\gamma = 10^{-4}, k = 5, \alpha = 0.9)$ for ExtraAdam and LA-ExtraAdam, respectively. In comparison to ExtraAdam, the capability of the resnet-18 model trained with LA to detect adversarial examples at test time, was significantly.

with LA detects significantly less adversarial examples than the model trained with

ExtraAdam. Corresponding plots for the other optimizers can be found in the Appendix (see Figure 7.8).

Table 6.3 provides an overview of the mean PGD validation accuracy for each optimizer on CIFAR-10. Adam and LA-Adam performed the best with a mean PGD validation accuracy of 35.5% and 35.6%, respectively.

in %	SGD	Adam	OGD	ExtraSGD	ExtraAdam
PGD					
no LA	34.1 ± 1.0	35.5 ± 0.7	32.9 ± 0.9	28.0 ± 0.7	34.1 ± 0.9
LA	33.7 ± 0.8	**35.6** ± 0.7	33.9 ± 1.3	31.1 ± 0.8	30.0 ± 3.2
clean					
no LA	57.3 ± 1.0	61.3 ± 0.6	55.9 ± 1.0	42.5 ± 1.7	60.6 ± 0.8
LA	58.3 ± 0.5	**61.7** ± 0.6	56.1 ± 1.6	57.0 ± 1.0	36.1 ± 11.4

Table 6.3: Mean PGD validation accuracy and clean validation accuracy (± one standard deviation) after 25 epochs computed over 10 runs on CIFAR-10.

Note that for LA-ExtraAdam, the average PGD validation accuracy of 30% with tuned hyperparameters is considerably lower than the average PGD validation accuracy of 32.5% obtained by averaging over the results of the hyperparameter tuning (see Table 6.2). This is mainly driven by the instability of LA-ExtraAdam. More precisely, in the case of LA-ExtraAdam, we observed a large volatility of the PGD validation accuracy between the epochs during training. This makes it very likely that the best performing hyperparameter settings of one grid search run do not perform best across multiple runs. Ideally, the grid search should be executed multiple times, and the best performing hyperparameter setting across all runs should be considered. However, this was not possible for these experiments due to lack of computational power.

It is also worth mentioning that training a bigger model did not change the outcome of our experiments. The resnet-34 model shows similar generalization behavior on the perturbed CIFAR-10 test set compared to that of resnet-18. For a comparison of the PGD validation accuracy of ExtraSGD and LA-ExtraSGD see the Appendix, Figure 7.9.

6.4.2 MNIST and FashionMNIST

Again, for the experiments on MNIST and FashionMNIST, we obtain similar results (see Table 6.4 and Table 6.5. Although, on MNIST, LA-Adam performs significantly better,

scoring more than 1 percentage point above every other optimizer.

in %	SGD	Adam	OGD	ExtraSGD	ExtraAdam
PGD					
no LA	92.1 ± 0.4	94.2 ± 0.3	90.6 ± 3.1	10.2 ± 0.6	94.1 ± 0.4
LA	94.7 ± 0.4	**95.8** ± 0.2	92.1 ± 1.7	79.3 ± 1.3	93.8 ± 0.3
clean					
no LA	98.4 ± 0.1	98.7 ± 0.1	98.5 ± 0.4	10.2 ± 0.6	96.7 ± 0.1
LA	**99.0** ± 0.1	99.0 ± 0.1	98.6 ± 0.2	98.8 ± 0.2	98.7 ± 0.1

Table 6.4: Mean PGD validation accuracy and clean validation accuracy (± one standard deviation) after 25 epochs computed over 10 runs on MNIST.

in %	SGD	Adam	OGD	ExtraSGD	ExtraAdam
PGD					
no LA	58.5 ± 1.6	68.5 ± 1.1	57.1 ± 2.6	10.0 ± 0.0	**69.9** ± 1.6
LA	60.4 ± 7.0	68.2 ± 1.1	45.2 ± 23.1	46.2 ± 1.5	62.1 ± 1.6
clean					
no LA	70.8 ± 1.4	75.9 ± 0.6	70.3 ± 1.3	10.0 ± 0.0	**76.6** ± 0.6
LA	71.0 ± 1.4	76.3 ± 0.7	52.6 ± 27.9	69.0 ± 0.5	68.2 ± 2.7

Table 6.5: Mean PGD validation accuracy and clean validation accuracy (± one standard deviation) after 25 epochs computed over 10 runs on FashionMNIST.

On both MNIST and FashionMNIST, LA provides significant improvements in PGD validation accuracy for models trained with ExtraSGD, mainly due to the fact that models tend to collapse with ExtraSGD (see Appendix, Figure 7.9). Similarly, LA raises the mean PGD validation accuracy for SGD but suffers from increased volatility. For Adam, OGD, and ExtraAdam, LA does not yield better generalizing models. Furthermore, LA-OGD induces high volatility in accuracy to the model on the FashionMNIST data set in comparison to OGD.

6.5 Conclusion

In conclusion, for adversarial training, LA only provides significant robustness improvements in regards to changes of the hyperparameters for ExtraSGD but does not do so for SGD, Adam, OGD or ExtraAdam, both at training time and test time. For ExtraAdam,

LA seems to even have an unfavorable effect in terms of robustness. While our models, trained on MNIST and FashionMNIST with ExtraSGD, collapsed for each learning rate used in the grid search, LA-ExtraSGD prevented this behavior for some hyperparameter settings. Similarly, on CIFAR-10, ExtraSGD caused the model to collapse for all but the smallest learning rate $\gamma = 3 \cdot 10^{-5}$, while LA-ExtraSGD prevented a model collapse for all hyperparameter settings used in the grid search. We therefore conclude that LA can successfully prevent model collapse.

Furthermore, our experiments with tuned hyperparameters suggest that LA *can* generate better generalizing models than its corresponding inner optimizer, and can even yield better results than Adam or SGD. However this was not the case for all experiments, e.g., on FashionMNIST, LA performed worse. Moreover, LA showed negative effects for ExtraAdam across most experiments. All in all, LA could not produce the expected performance advantages over other optimizers for adversarial training like for other optimization setting (e.g: training generative adversarial networks [7]).

Bibliography

[1] Naveed Akhtar and Ajmal Mian. "Threat of adversarial attacks on deep learning in computer vision: A survey". In: *Ieee Access* 6 (2018), pp. 14410–14430.

[2] Jimmy Ba and Rich Caruana. "Do deep nets really need to be deep?" In: *Advances in neural information processing systems* 27 (2014).

[3] Aharon Ben-Tal, Laurent El Ghaoui, and Arkadi Nemirovski. *Robust optimization*. Vol. 28. Princeton university press, 2009.

[4] Dimitri P Bertsekas. "Nonlinear programming". In: *Journal of the Operational Research Society* 48.3 (1997), pp. 334–334.

[5] Nicholas Carlini, Guy Katz, Clark Barrett, and David L. Dill. *Provably Minimally-Distorted Adversarial Examples*. 2017. URL: https://arxiv.org/abs/1709.10207.

[6] Nicholas Carlini and David Wagner. "Towards evaluating the robustness of neural networks". In: *2017 ieee symposium on security and privacy (sp)*. IEEE. 2017, pp. 39–57.

[7] Tatjana Chavdarova, Matteo Pagliardini, Martin Jaggi, Francois Fleuret, and Sebastian Stich. "Taming GANs with Lookahead-Minmax". In: (2021). URL: https://openreview.net/forum?id=ZW0yXJyNmoG.

[8] Dan Ciresan, Ueli Meier, Jonathan Masci, and Jurgen Schmidhuber. "A Committee of Neural Networks for Traffic Sign Classification". In: (2011). URL: https://people.idsia.ch/~ciresan/data/ijcnn2011.pdf.

[9] George Dantzig. "Linear programming and extensions". In: *Linear programming and extensions*. Princeton university press, 2016.

[10] Constantinos Daskalakis, Andrew Ilyas, Vasilis Syrgkanis, and Haoyang Zeng. "Training GANs with Optimism". In: *International Conference on Learning Representations*. 2018. URL: https://openreview.net/forum?id=SJJySbbAZ.

[11] Geir E Dullerud and Fernando Paganini. *A course in robust control theory: a convex approach*. Vol. 36. Springer Science & Business Media, 2013.

[12] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. "Robust physical-world

attacks on deep learning visual classification". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 1625–1634.

[13] Gauthier Gidel, Hugo Berard, Gaëtan Vignoud, Pascal Vincent, and Simon Lacoste-Julien. "A variational inequality perspective on generative adversarial networks". In: *arXiv preprint arXiv:1802.10551* (2018).

[14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. "Generative adversarial nets". In: *Advances in neural information processing systems* 27 (2014).

[15] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. *Explaining and Harnessing Adversarial Examples*. 2014. URL: https://arxiv.org/abs/1412.6572.

[16] Robert M Gower. "Convergence theorems for gradient descent". In: *Lecture notes for Statistical Optimization* (2018).

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[18] Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. "Distilling the knowledge in a neural network". In: *arXiv preprint arXiv:1503.02531* 2.7 (2015).

[19] Geoffrey Hinton et al. "Deep Neural Networks for Acoustic Modeling in Speech Recognition". In: *Signal Processing Magazine* (2012).

[20] Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Logan Engstrom, Brandon Tran, and Aleksander Madry. "Adversarial examples are not bugs, they are features". In: *Advances in neural information processing systems* 32 (2019).

[21] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. "Reluplex: An efficient SMT solver for verifying deep neural networks". In: *International conference on computer aided verification*. Springer. 2017, pp. 97–117.

[22] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014. URL: https://arxiv.org/abs/1412.6980.

[23] Anastasia Koloskova, Nicolas Loizou, Sadra Boreiri, Martin Jaggi, and Sebastian Stich. "A unified theory of decentralized sgd with changing topology and local updates". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 5381–5393.

[24] Galina M Korpelevich. "The extragradient method for finding saddle points and other problems". In: *Matecon* 12 (1976), pp. 747–756.

[25] Alex Krizhevsky, Geoffrey Hinton, et al. "Learning multiple layers of features from tiny images". In: (2009).

[26] Alexey Kurakin, Ian J Goodfellow, and Samy Bengio. "Adversarial examples in the physical world". In: *Artificial intelligence safety and security*. Chapman and Hall/CRC, 2018, pp. 99–112.

[27] Yann LeCun, Corinna Cortes, and CJ Burges. "MNIST handwritten digit database". In: *ATT Labs [Online]. Available: http://yann. lecun. com/exdb/mnist* 2 (2010).

[28] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. "Towards Deep Learning Models Resistant to Adversarial Attacks". In: *International Conference on Learning Representations*. 2018. URL: `https://openreview.net/forum?id=rJzIBfZAb`.

[29] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. "Deepfool: a simple and accurate method to fool deep neural networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2574–2582.

[30] Nicolas Papernot and Patrick McDaniel. "On the effectiveness of defensive distillation". In: *arXiv preprint arXiv:1607.05113* (2016).

[31] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. *Transferability in Machine Learning: from Phenomena to Black-Box Attacks using Adversarial Samples*. 2016. URL: `https://arxiv.org/abs/1605.07277`.

[32] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. "Practical black-box attacks against machine learning". In: *Proceedings of the 2017 ACM on Asia conference on computer and communications security*. 2017, pp. 506–519.

[33] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. "The limitations of deep learning in adversarial settings". In: *2016 IEEE European symposium on security and privacy (EuroS&P)*. IEEE. 2016, pp. 372–387.

[34] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. "Distillation as a defense to adversarial perturbations against deep neural networks". In: *2016 IEEE symposium on security and privacy (SP)*. IEEE. 2016, pp. 582–597.

[35] Alexander Rakhlin and Karthik Sridharan. "Online learning with predictable sequences". In: *Conference on Learning Theory*. PMLR. 2013, pp. 993–1019.

[36] Tom Schaul, Sixin Zhang, and Yann LeCun. "No more pesky learning rates". In: *International conference on machine learning*. PMLR. 2013, pp. 343–351.

[37] Mahmood Sharif, Lujo Bauer, and Michael K Reiter. "On the suitability of lp-norms for creating and preventing adversarial examples". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2018, pp. 1605–1613.

[38] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. "Deep inside convolutional networks: Visualising image classification models and saliency maps". In: *arXiv preprint arXiv:1312.6034* (2013).

[39] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. "On the importance of initialization and momentum in deep learning". In: *Proceedings of the 30th International Conference on Machine Learning*. Ed. by Sanjoy Dasgupta and David McAllester. Vol. 28. Proceedings of Machine Learning Research 3. Atlanta, Georgia, USA: PMLR, 2013, pp. 1139–1147. URL: https://proceedings.mlr.press/v28/sutskever13.html.

[40] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. "Intriguing properties of neural networks". In: *International Conference on Learning Representations*. 2014. URL: http://arxiv.org/abs/1312.6199.

[41] Florian Tramèr, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. *The Space of Transferable Adversarial Examples*. 2017. URL: https://arxiv.org/abs/1704.03453.

[42] Jianyu Wang, Vinayak Tantia, Nicolas Ballas, and Michael Rabbat. "Lookahead Converges to Stationary Points of Smooth Non-convex Functions". In: *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2020, pp. 8604–8608.

[43] David Warde-Farley and Ian Goodfellow. "11 adversarial perturbations of deep neural networks". In: *Perturbations, Optimization, and Statistics* (2016), pp. 311–330.

[44] Martin Weiß. "Robust and optimal control : By Kemin Zhou, John C. Doyle and Keith Glover, Prentice Hall, New Jersey, 1996, ISBN 0-13-456567-3". In: *Autom.* 33 (1997), p. 2095.

[45] Tobias Weyand, Ilya Kostrikov, and James Philbin. "PlaNet - Photo Geolocation with Convolutional Neural Networks". In: *Computer Vision – ECCV 2016*. Springer International Publishing, 2016, pp. 37–55. URL: https://doi.org/10.1007%5C%2F978-3-319-46484-8_3.

[46] Yuhuai Wu, Mengye Ren, Renjie Liao, and Roger B. Grosse. "Understanding Short-Horizon Bias in Stochastic Meta-Optimization". In: *Proceedings of 6th International Conference on Learning Representations ICLR*. 2018.

[47] Han Xiao, Kashif Rasul, and Roland Vollgraf. *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms*. Aug. 28, 2017. arXiv: cs.LG/1708.07747 [cs.LG].

[48] Weilin Xu, David Evans, and Yanjun Qi. "Feature squeezing: Detecting adversarial examples in deep neural networks". In: *arXiv preprint arXiv:1704.01155* (2017).

[49] Xiaoyong Yuan, Pan He, Qile Zhu, and Xiaolin Li. "Adversarial examples: Attacks and defenses for deep learning". In: *IEEE transactions on neural networks and learning systems* 30.9 (2019), pp. 2805–2824.

[50] Xiaoyong Yuan, Pan He, Qile Zhu, and Xiaolin Li. "Adversarial examples: Attacks and defenses for deep learning". In: *IEEE transactions on neural networks and learning systems* 30.9 (2019), pp. 2805–2824.

[51] Guodong Zhang, Lala Li, Zachary Nado, James Martens, Sushant Sachdeva, George E. Dahl, Christopher J. Shallue, and Roger B. Grosse. "Which Algorithmic Choices Matter at Which Batch Sizes? Insights From a Noisy Quadratic Model". In: 2019, pp. 8194–8205.

[52] Guojun Zhang and Yaoliang Yu. "Convergence of Gradient Methods on Bilinear Zero-Sum Games". In: *International Conference on Learning Representations*. 2020. URL: `https://openreview.net/forum?id=SJlVY04FwH`.

[53] Michael Zhang, James Lucas, Jimmy Ba, and Geoffrey E Hinton. "Lookahead optimizer: k steps forward, 1 step back". In: *Advances in neural information processing systems* 32 (2019).

7 Appendix

7.1 Section A

The following result is particularly useful since it shows that an optimization method converges linearly to a stable fixed point of the corresponding operator G if the weights are initialized sufficiently close.

Theorem 7.1.1. *([4], Proposition 4.4.1). If the spectral radius $\rho(J_G(\omega^*)) < 1$, the fixed point ω^* is a point of attraction and for $\omega^{(0)}$ in a sufficiently small neighborhood of ω^*, the distance of $\omega^{(t)}$ to the stationary point ω^* converges at a linear rate.*

Proof. For a proof refer to [4]. □

Lemma 7.1.2. *([16, Lemma 1.3]). Let $f : \mathbb{R}^p \to \mathbb{R}$, be an L-smooth function. Then, the following inequality holds*

$$f(v) - f(x) \leq \langle \nabla f(x), v - x \rangle + \frac{L}{2}\|x - v\|^2, \quad \forall x, v \in \mathbb{R}^p.$$

Proof. Using the Taylor expansion of f at x, we obtain that,

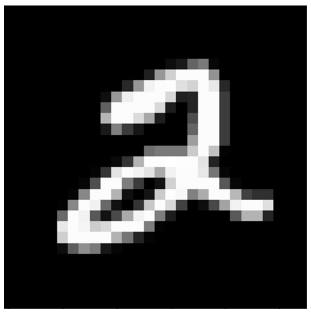
$$\begin{aligned} f(v) - f(x) &= \int_0^1 \langle \nabla f(x + t(v - x)), (v - x) \rangle dt \\ &= \langle \nabla f(x), v - x \rangle + \int_0^1 \langle \nabla f(x + t(v - x)) - \nabla f(x), (v - x) \rangle dt \\ &\leq \langle \nabla f(x), v - x \rangle + \int_0^1 \|\nabla f(x + t(v - x)) - \nabla f(x)\| \|v - x\| dt \\ &\leq \langle \nabla f(x), v - x \rangle + L \int_0^1 \|v - x\|^2 dt \\ &\leq \langle \nabla f(x), v - x \rangle + \frac{L}{2}\|v - x\|^2. \end{aligned}$$

□

7.2 Section B

7.2.1 Datasets

7.2.2 Additional Plots



(a) Example of the MNIST data set. Each example consists of 784 (28x28) pixels representing a black and white image of a handwritten digit, from zero through nine.

(b) Example (T-shirt) of the FashionMNIST data set. Each example consists of 784 (28x28) pixels representing a black and white image of ten different types of garments: T-shirt, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, and Ankle boot.

Figure 7.1: For both data sets, each pixel has a value associated with it, representing the gray scale of the pixel and ranging from 0 to 255.

Figure 7.2: Example (horse) of the CIFAR-10 data set. Each example consists of 1024 (32x32) pixels representing a 3-channel color image of the following ten classes of objects and animals: airplane, car, bird, cat, deer, dog, frog, horse, ship, truck.

7.2.3 Model

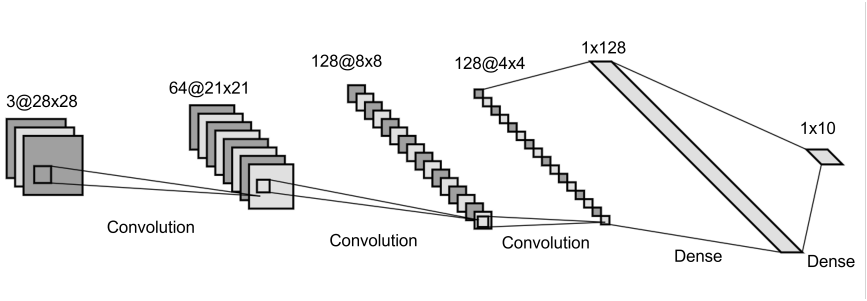
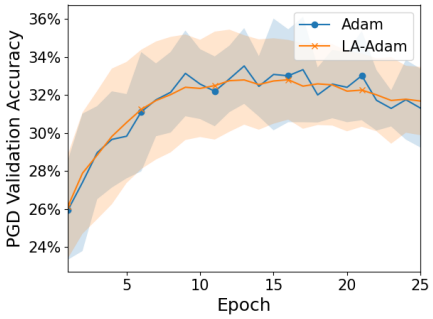
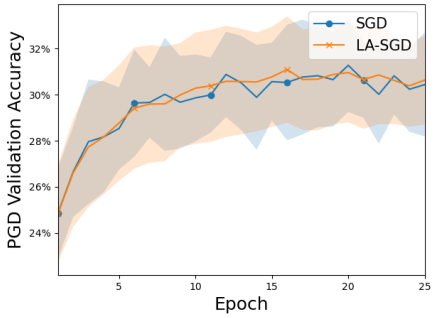


Figure 7.3: Architecture of 5-layer CNN used for adversarial training on the MNIST and FashionMNIST data set.

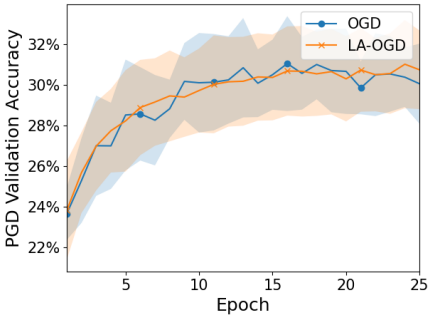
7.2.4 Additional Plots



(a) Mean PGD validation accuracy of Adam (blue) and LA-Adam (orange).

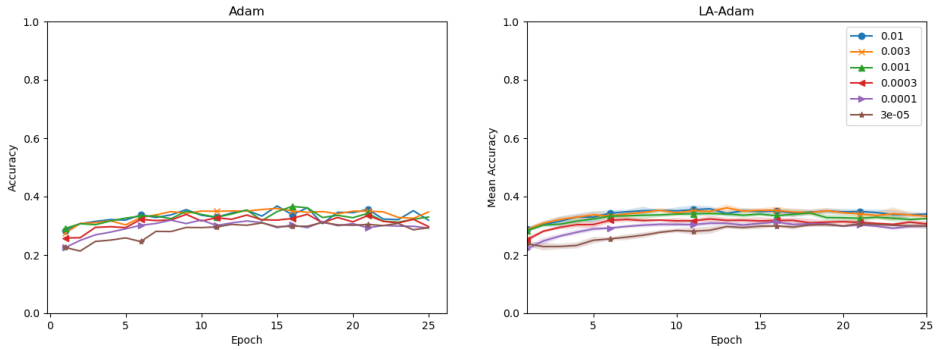


(b) Mean PGD validation accuracy of SGD (blue) and LA-SGD (orange).

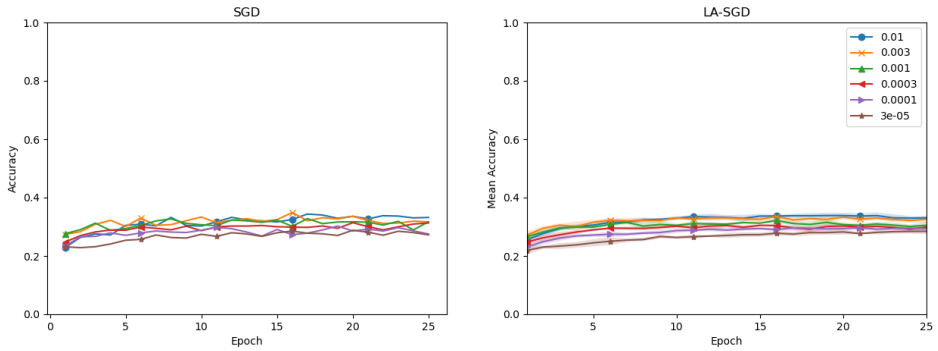


(c) Mean PGD validation accuracy of OGD (blue) and LA-OGD (orange).

Figure 7.4: The plots show the validation accuracies (± one standard deviation) of the resnet-18 model on the *PGD-perturbed* CIFAR-10 test set averaged over all hyperparameter settings used in the grid search.

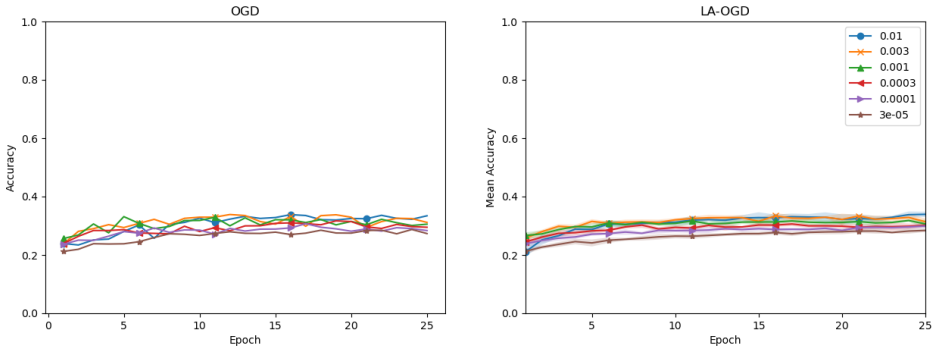


(a)

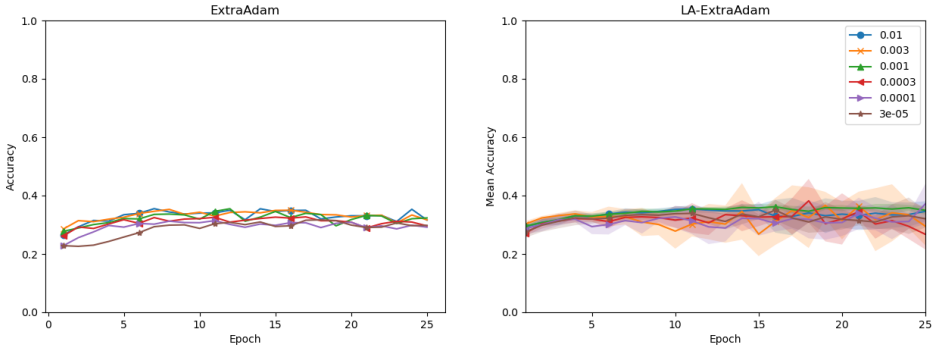


(b)

Figure 7.5: The plots on the left depict the PGD validation accuracies for each inner optimizer obtained in the grid search. The plots on the right show the PGD validation accuracies, i.e., the validation accuracy of the resnet-18 model on the *PGD-perturbed* CIFAR-10 test set, averaged over all values for k and α used in the grid search. The shaded areas in the right-hand side plots represent the accuracies with a maximum distance of one standard deviation from the mean values of an inner optimizer and LA equipped with the inner optimizer, respectively.



(a)



(b)

Figure 7.6: The plots on the left depict the PGD validation accuracies for each inner optimizer obtained in the grid search. The plots on the right show the PGD validation accuracies, i.e., the validation accuracy of the resnet-18 model on the *PGD-perturbed* CIFAR-10 test set, averaged over all values for k and α used in the grid search. The shaded areas in the right-hand side plots represent the accuracies with a maximum distance of one standard deviation from the mean values of an inner optimizer and LA equipped with the inner optimizer, respectively.

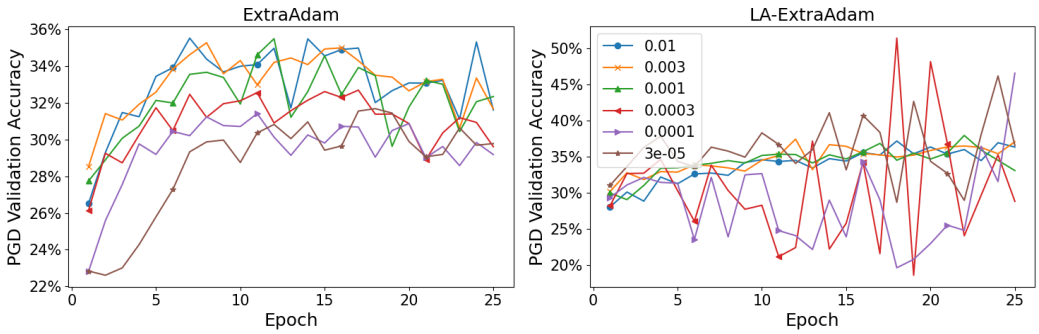
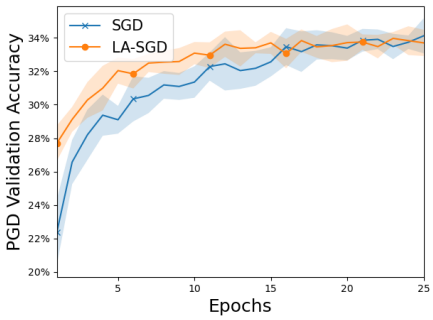
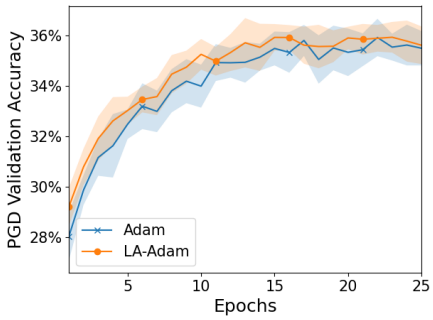


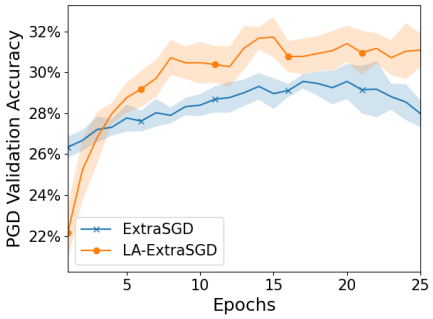
Figure 7.7: PGD Validation Accuracy for ExtraAdam (LHS) and LA-ExtraAdam (RHS) for six different values of the fast learning rate γ. For LA, we have $k = 5$ and $\alpha = 0.9$. LA seems to have a destabilizing effect on ExtraAdam.



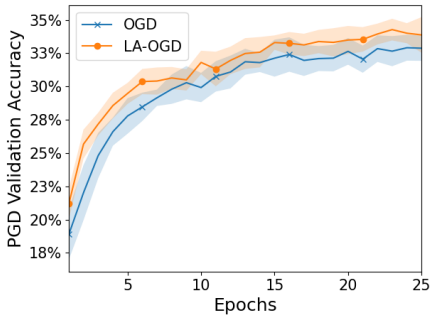
(a) Tuned hyperparameters: $\gamma = 10^{-2}$ for SGD and $(\gamma, k, \alpha) = (3 \cdot 10^{-3}, 10, 0.75)$ for LA-SGD.



(b) Tuned hyperparameters: $\gamma = 3 \cdot 10^{-3}$ for Adam and $(\gamma, k, \alpha) = (3 \cdot 10^{-3}, 10, 0.5)$ for LA-Adam.

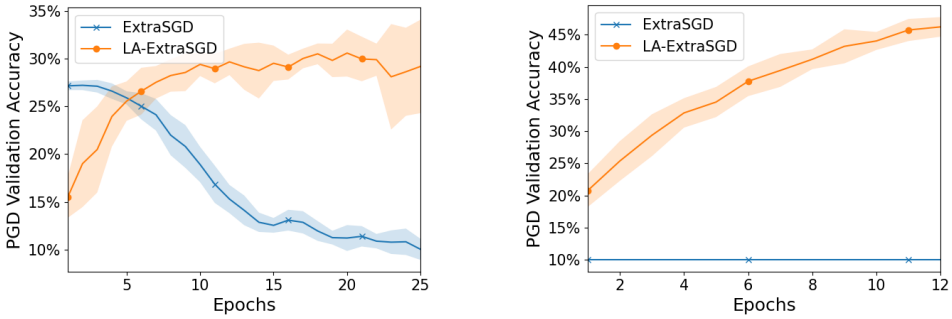


(c) Tuned hyperparameters: $\gamma = 3 \cdot 10^{-5}$ for ExtraSGD and $(\gamma, k, \alpha) = (10^{-2}, 5, 0.9)$ for LA-ExtraSGD.



(d) Tuned hyperparameters: $\gamma = 10^{-2}$ for OGD and $(\gamma, k, \alpha) = (10^{-2}, 5, 0.75)$ for LA-OGD.

Figure 7.8: Comparison of the Mean PGD validation accuracy (averaged over 10 runs) after 25 epochs for both standalone inner optimizer (SGD, Adam, OGD, ExtraSGD) and LA with inner optimizer. LA does not significantly increase the capability of the resnet-18 model to detect adversarial examples at test time.



(a) Comparison of the Mean PGD validation accuracy after 25 epochs for the resnet-34 model for both standalone ExtraSGD and LA with ExtraSGD. Note that the same hyperparameter settings as for resnet-18 ($\gamma = 3 \cdot 10^{-5}$ for ExtraSGD and $(\gamma, k, \alpha) = (10^{-2}, 5, 0.9)$ for LA-ExtraSGD) have been used for resnet-34.

(b) Comparison of the Mean PGD validation accuracy on FashionMNIST after 12 epochs for both standalone ExtraSGD ($\gamma = 10^{-3}$)and LA with ExtraSGD ($\gamma = 10^{-4}, k = 5, \alpha = 0.9$). LA increased the capability of the resnet-18 model to detect adversarial examples at test time significantly. This difference in PGD validation accuracy is mainly driven by the model collapses for ExtraSGD. In the grid search, the model collapsed for all six different learning rates.

Figure 7.9