Chapter 1

Theoretical Background

Deep learning, *a class of Machine Learning (ML) algorithms based on neural networks (NNs)*, has revolutionized the way we tackle a problem from a ML perspective and is one if not the most important factor for recent ML achievements. Solving complex tasks such image classification or language translation, that for years have bedevilled traditional ML algorithms, constitutes the signature of Deep Learning (DL). Admittedly, *the advent of a deep convolutional neural network (CNN)*, the `AlexNet` (Krizhevsky et al. 2012) on September 30 of 2012, signified the "modern birthday" of this field. On this day, `AlexNet` not only won the `ImageNet` (Deng et al. 2009) Large Scale Visual Recognition Challenge (ILSVRC), but dominated it, achieving a top-5 accuracy of 85%, surpassing the runner-up which achieved a top-5 accuracy of 75%. `AlexNet` showed that NNs are not merely a pipe-dream, but they can be applied in real-world problems. It is worth to notice that ideas of NNs trace back to 1943, but it was until recently that these ideas got materialized. The reason for this recent breakthrough of DL (and ML) is twofold. First, the availability of large datasets—the era of big data—such as `ImagNet`. Second, the increase in computational power, mainly of GPUs for DL, accelerating the training of deep NNs and traditional ML algorithms.

1.1 Machine Learning Preliminaries

Since DL is a subfield of ML, it is necessary to familiarize with the later before diving into the former. In this section, the minimum theoretical background and jargon of ML is presented. Machine Learning can be defined as *"the science and (art) of programming computers so they can learn from the data"* (Géron 2017). A more technical definition is the following:

Definition 1.1 (Machine learning, Mitchell 1997)**.** *A computer program is said to learn from experience **E** with respect to some class of tasks **T** and some performance measure **P**, if its performance on **T**, as measured by **P**, improves with experience **E**.*

For instance, a computer program that classifies emails into spam and non-spam (the task **T**), can improve its accuracy, i.e. the percentage of correctly classified emails (the performance **P**), through examples of spam and non-spam emails (the experience **E**). But in order to take advantage of the experience aka *data*, it must be written in such a way that *adapts to the patterns in the data*. Certainly, a *traditional spam filter can not learn from experience*, since the latter does not affect the classification rules of the former and as such, its performance. For a traditional spam filter to adapt to new patterns and perform better, it must change its hard-wired rules, but by then it will be a different program. In contrast, a *ML-based filter can adapt to new patterns, simply because it has been programmed to do so*. In

other words, *in traditional programming we write rules for solving* $\boldsymbol{T}$ *whereas* ***in ML we write rules to learn the rules*** *for solving* $\boldsymbol{T}$. This subtle but essential difference is what gives ML algorithms the ability to take advantage of the data.

1.1.1 Learning paradigms

Depending on the type of experience they are allowed to have during their *training phase* (Goodfellow et al. 2016), ML approaches are divided into three main *learning paradigms*: ***unsupervised learning***, ***supervised learning*** *and* ***reinforcement learning***. The following definitions are not by any means formal, but merely serve as an intuitive description of the different paradigms.

Definition 1.2 (Unsupervised learning)**.** *The experience comes in the form* $\mathcal{D}_{train} = \{\boldsymbol{x}_i\}$*, where* $\boldsymbol{x}_i \sim p(\boldsymbol{x})$ *is the input of the* i*-th training instance. In this paradigm we are interested in learning useful properties of the underlying structure captured by* $p(\boldsymbol{x})$.

For example, suppose we are interested in generating images that look like Picasso paintings. In this case, the input is just the pixel values, i.e. $\boldsymbol{x} \in \mathbb{R}^{W \times H \times 3}$. The latter follow a distribution $p(\boldsymbol{x})$, so all we have to do is to train an unsupervised learning algorithm with many Picasso paintings to get a *model*, that is $\hat{p}(\boldsymbol{x})$. Assuming the estimation of the original distribution is good enough, new realistically looking paintings (with respect to original Picasso paintings) can be "drawn" by just sampling from $\hat{p}(\boldsymbol{x})$. In the ML parlance, inputs are also called *features, predictors or descriptors*.

Definition 1.3 (Supervised learning)**.** *The experience comes in the form* $\mathcal{D}_{train} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}$*, where* $(\boldsymbol{x}_i, \boldsymbol{y}_i) \sim p(\boldsymbol{x}, \boldsymbol{y})$ *and* $\boldsymbol{y}_i$ *is the output aka label of the* i*-th training instance. In this paradigm we are usually interested in learning a function* $f : X \to Y$.

This paradigm comes mainly under two flavors: *regression and classification.* In regression the interest is in predicting a continuous value given an input, i.e. $y \in \mathbb{R}$, such as a molecular property given a mathematical representation of a molecule. In classification, the interest is to predict in which of k classes an input belongs to, i.e. $y \in \{1, \ldots, k\}$, such as predicting the breed of a dog image given the raw pixel values of the image. The term "supervised" is coined due to the "human supervision" the algorithm receives during its training phase, through the presence of the correct answer (the label) in the experience. In a sense, in this paradigm we "teach" the learning algorithm aka *learner*. It should be emphasized that the label is not constrained to be single-valued, but can also be multi-valued. In this case, one talks about *multi-label regression or classification* (Read et al. 2009).

A more exotic form of supervised learning is *conditional generative modelling*, where the interest is in estimating $p(\boldsymbol{x} \mid \boldsymbol{y})$. For example, one may want to build a model that generates images of a specific category or a *model that designs molecules/materials with tailored properties* (Kim et al. 2018; Yao et al. 2021; Gebauer et al. 2022). This is one approach of how ML can tackle the *inverse design problem* in chemistry.

Definition 1.4 (Reinforcement learning)**.** *The experience comes from the interaction of the learner, called agent in this context, with its environment. In other words, there is a feedback loop between the learner and its environment. In this paradigm we are interested in building an agent that can take suitable actions in a given situation.*

The agent observes its *environment*, selects and performs *actions* and gets *rewards or penalties* in return. The goal is to learn an optimal strategy, called a *policy*, that *maximizes the long-term reward* (Pedregosa et al. 2011). A policy simply defines the action that the agent should choose in a given sit-

uation. In contrast to supervised learning, where the correct answers are provided to the learner, *in reinforcement learning the learner must find the optimal answers by trial and error* (Bishop 2007). Reinforcement learning techniques find application in fields such as gaming (`AlphaGo` is a well known example), robotics, autonomous driving and recently chemistry (**li**; **Gow2022**). Since in the present thesis only supervised learning techniques were employed, the remaining of this chapter is devoted to this learning paradigm.

1.1.2 Formulating the problem of supervised learning

The general setting of supervised learning is as follows: we assume that there is some relationship between $\boldsymbol{x}$ and $\boldsymbol{y}$:

$$\boldsymbol{y} = f(\boldsymbol{x}) + \boldsymbol{\epsilon} \tag{1.1}$$

and we want to estimate f from the data. The function f represents the *systematic information* that $\boldsymbol{x}$ gives about $\boldsymbol{y}$ while $\boldsymbol{\epsilon}$ is a random *error term* independent of $\boldsymbol{x}$ and with zero mean. More formally, we have an input space X, an output space Y and we are interested in learning a function $\hat{h}\colon X \to Y$, called the *hypothesis*, which produces an output $\boldsymbol{y} \in Y$ given an input $\boldsymbol{x} \in X$. At our disposal we have a collection of input-output pairs $(\boldsymbol{x}_i, \boldsymbol{y}_i)$, forming the *training set* $\mathcal{D}_{\text{train}}$, with the pairs drawn i.i.d from $p(\boldsymbol{x}, \boldsymbol{y})$.

Ideally, we would like to learn a hypothesis that minimizes the *generalization error or loss*:

$$\mathcal{L} \coloneqq \int_{X \times Y} \ell(h(\boldsymbol{x}), \boldsymbol{y}) p(\boldsymbol{x}, \boldsymbol{y}) d\boldsymbol{x} d\boldsymbol{y} \tag{1.2}$$

that is, the expected value of some *loss function* ℓ over all possible input-output pairs. A loss function just measures the discrepancy of the prediction $h(\boldsymbol{x}) = \hat{\boldsymbol{y}}$ from the true value $\boldsymbol{y}$ and as such, the best hypothesis is the one that minimizes this integral. Obviously, it is impossible to evaluate the integral in Equation 1.2, since we don't have access to infinite data.

The idea is to use the *training error or loss*:

$$\mathcal{L}_{\text{train}} \coloneqq \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{i \in \mathcal{D}_{\text{train}}} \ell(h(\boldsymbol{x}_i), \boldsymbol{y}_i) \tag{1.3}$$

as an approximation for the generalization loss, and *choose the hypothesis that minimizes the training loss*, a principle known as *empirical risk minimization*. In other words, to get a hypothesis aka *model* $\mathcal{M}$ from the data, we need to solve the following optimization problem:

$$\hat{h} \leftarrow \arg\min_{h \in \mathcal{H}} \mathcal{L}_{\text{train}} \tag{1.4}$$

which is achieved, by just feeding the training data into the learning algorithm $\mathcal{A}$:

$$\mathcal{M} \leftarrow \mathcal{A}(\mathcal{D}_{\text{train}}) \tag{1.5}$$

1.1.3 Components of a learning algorithm

By breaking down Equation 1.4, i.e. the optimization problem the learner needs to solve, the components of a learner can be revealed. The latter is comprised of the following three "orthogonal" components: a ***hypothesis space***, a ***loss function*** and an ***optimizer***. We now look into each of them individually and describe the contribution of each one to the solution of the optimization problem. For the sake of clarity, in the remaining of this chapter we will stick to examples from simple (single-valued) regression and binary classification.

Definition 1.5 (Hypothesis space)**.** *The set of hypotheses (functions), denoted as $\mathcal{H}$, from which the learner is allowed to pick the solution of Equation 1.4.*

A simple example of a hypothesis space, is the one used in univariate *linear regression*:

$$\hat{y} = \beta_0 + \beta x \tag{1.6}$$

where $\mathcal{H}$ contains all lines (or hyperplanes in the multivariate case) defined by Equation 1.6. Of course, one can get *a more expressive* hypothesis space, by including polynomial terms, e.g.:

$$\hat{y} = \beta_0 + \beta_1 x + \beta_2 x^2 \tag{1.7}$$

The more expressive the hypothesis space, the larger the *complexity or representational capacity* of the learning algorithm. For a formal definition of representational capacity, the interested reader can look at *Vapnik-Chervonenkis Dimension* (**statlearn**).

Definition 1.6 (Loss function)**.** *A function that maps a set of predictions into a real number, intuitively representing the quality of a candidate hypothesis.*

For example, a typical loss function used in regression is the *squared loss*:

$$\ell(\hat{y}, y) \coloneqq (\hat{y} - y)^2 \tag{1.8}$$

where $y, \hat{y} \in \mathbb{R}$. A typical loss function for binary classification is the *binary cross entropy loss*:

$$\ell(\hat{y}, y) \coloneqq y \cdot \log(\hat{y}) + (1 - y) \cdot \log(1 - \hat{y}) \tag{1.9}$$

where $y \in \{0, 1\}$, indicating the correct class, and $\hat{y} \in [0, 1]$ which corresponds to the predicted probability for class-1. Notice that in both cases the loss is minimum when the prediction is equal to the ground truth. For the cross entropy loss, if $y = 1$ is the correct class, then the model must predict $\hat{y} = 1$ for the loss to be minimized.

Usually, we are not only penalizing a hypothesis for its mispredictions, but also for its complexity. This is done in purpose, since a learner with a very rich hypothesis space can easily memorize the training set but fail to generalize well to new unseen examples. *Every modification that is made to a learner in order to reduce its generalization loss but not its training loss, is called regularization* (Goodfellow et al. 2016).

A common—but not the only—way to achieve that, is by including another penalty term called *regularization term or regularizer*, denoted as $\mathcal{R}$, in Equation 1.3:

$$\mathcal{L}_{\text{train}} \coloneqq \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{i \in \mathcal{D}_{\text{train}}} \ell(h(\boldsymbol{x}_i), \boldsymbol{y}_i) + \lambda \mathcal{R} \tag{1.10}$$

The λ factor controls the strength of regularization and it is an *hyperparameter*, i.e. a parameter that is not learned during training but whose value is used to control the training phase. In order to see how λ penalizes model complexity, lets examine polynomial regression of degree k:

$$\hat{y} = \beta_0 + \sum_{i=1}^{k} \beta_i x^i \tag{1.11}$$

combining *mean squared loss* and *Lasso regularization* as training loss:

$$\mathcal{L}_{\text{train}} = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{i \in \mathcal{D}_{\text{train}}} \ell(\hat{y}_i - y_i)^2 + \lambda \sum_{i=1}^{k} |\beta_i| \tag{1.12}$$

Definition 1.7 (Optimizer)**.**

Mention that the loss affects represenational capacity. Mention why regularization is useful. Mention that the optimizer affects represenational capacity.

Index