Components of Machine Learning: Binding Bits and FLOPS

Alexander Jung

***Abstract*—Many machine learning problems and methods are combinations of three components: data, hypothesis space and loss function. Different machine learning methods are obtained as combinations of different choices for the representation of data, hypothesis space and loss function. After reviewing the mathematical structure of these three components, we discuss intrinsic trade-offs between statistical and computational properties of ML methods.**

I. Introduction

Machine learning (ML) methods implement the scientific principle of continuous verification and adaptation of a hypothesis about an observable phenomenon ("observable fact or event") [1]. Examples of a phenomena are:

- the visual scene recorded by the smartphone snapshot depicted in Figure 2.
- the hiking time required to reach the peak in Figure 2.
- the water temperature of the lake in Figure 2.

The verification and adaption of the hypothesis is based on the observation of data. ML theory and methods revolve around the implementation of the cycle underlying this principle using limited computational resources such as computation time and storage capacity.

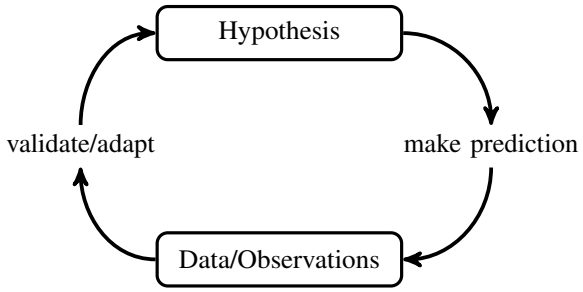

Fig. 1. The cycle of the scientific principle.

Modern ML methods execute the cycle in Figure 1 within a fraction of a second and using billions of data points [2]. A recently popularized class of ML method that implement the cycle of Figure 1 are deep learning methods, which represent hypothesis by artificial neural networks whose weights (parameters) are continuously adapted using variants of gradient descent [2].

A typical ML method consists of three components:

- data (mostly in the form of a huge number of bits)
- a hypothesis space (also referred to as a ML model) consisting of computationally feasible predictor functions.
- a loss function that is used to assess the quality of a particular predictor function.

To implement ML methods, given a limited amount of computational resources such as number of floating point operations per second (FLOPS), we need to be able to efficiently store and manipulate data and predictor functions. One extremely efficient approach to represent and manipulate data and predictor functions are matrices and vectors. The mathematical foundation of computing with matrices and vectors is linear algebra [3]. Therefore, a large part of ML theory and methodology is applied numerical linear algebra.

Indeed, data points can often characterized by a list of numeric attributes x_r which can be stacked into a vector $\mathbf{x} = \big(x_1, \ldots, x_n\big)^T$. Moreover, many ML methods (such as linear regression or logistic regression) use predictor functions of the form $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ with some weight vector $\mathbf{w}$. Note that once we restrict ourselves to linear functions of the form $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$, we can represent a predictor function by the weight vector $\mathbf{w}$. Indeed, given the weight vector $\mathbf{w}$, we can evaluate the predictor function for any feature vector $\mathbf{x}$ as $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$.

II. Data

The key component of any machine learning problem (and method) is data. There are many different sources of data such as text documents, sensor measurements, videos or image collections. Digital data is available in the form of a stream of bits which needs to be parsed into elementary units which represent individual data points. Data points might be represented by rows in a spreadsheet, the set of weather observations in Finland during a specific period of time, images, audio recordings or entire digital footprints of humans.

Typically, we have never full access to (every single detailed aspect of) data points. Some properties of a data point can be computed, measured or determined easily. These properties or characteristics are often referred to as *features*. Beside features, there is often also some higher-level information ("quantity of interest") associated with a data point. We will refer to this higher level information, or quantity of interest, as *labels*. Many ML methods revolve around finding efficient ways to determine the label of a data point given its features.

Consider a data point represented by the snapshot depicted in Figure 2. The features of this data point could be the red, green and blue intensities of each pixel in the image. We can stack these values into a vector $\mathbf{x} \in \mathbb{R}^n$ whose length n is given

Author is with the Department of Computer Science, Aalto University, Finland; firstname.lastname(at)aalto.fi

by three times the number of pixels in the image. The label y associated with this data point could be the expected hiking time to reach the mountain in the snapshot. Alternatively, we could define the label y as the water temperature of the lake visible in the snapshot.

Fig. 2. An image representing a data point.

The precise definition of what we use as features and labels of a data point is a design choice. The label is the quantity of interest for a particular application. If we are interested in developing a smartphone-app that predicts the hiking time given a snapshot of the mountain, we use this hiking time as label. However, if we are interested in developing a smartphone-app that predicts water temperature of a lake, we use this temperature as the label. For a given ML problem, we denote the set of all possible values that a label can take on by $\mathcal{Y}$. For a ML problem (method) using the choice $\mathcal{Y} = \mathbb{R}$, it is customary to refer to such a problem as a *regression* problem (method).

A data point is called *labeled* if, besides its features $\mathbf{x}$, the associated label y is known. While features are those properties or characteristics of data points that can be measured or computed easily, labels are difficult or costly to obtain. For the snapshot in Figure 2, we can easily determine the pixel intensities as features. However, if the label is the water temperature of the lake depicted on the snapshot, we need to actually measure this temperature. Acquiring labels typically involves human labor, such as handling a water thermometer at certain locations in a lake, and is costly. ML methods, which have to cope with limited resources available for acquiring labels, are geared to get along with as little labeled data points as possible.

Not only the label of a data point is a design choice but only what features are used to characterize a data point. In principle, we could use any quantity that can be easily computed or measured as a feature of a data point. Modern technology allows to compute a vast amount number of such quantities.

As a case in point, consider the data point "Alex Jung" obtained from a person which uses a smartphone to take snapshots. Let us assume that Alex takes five snapshots per day on average (sometimes more, e.g., during a mountain hike). This results in more than 1000 snapshots per year. Each snapshot contains around 10^6 pixels. If we only use the greyscale levels of the pixels in all those snapshots, we would obtain more 10^9 new features per year! In many modern ML applications we face extremely high-dimensional feature vectors which calls for methods from high-dimensional statistics [4], [5].

While it might seem that "the more features the better", it can actually be detrimental for the performance of ML methods to use an excessive amount of (irrelevant) features. It is non-trivial to decide which features are most relevant for a given task. However, there are ML methods that allow (to some extent) to automatically learn a small number of most relevant features from raw data.

III. Hypothesis Space

The scientific principle in Figure 1 involves a hypothesis for some phenomenon which generates observable data. We can think of a hypothesis as a simple explanation or conception of some complicated phenomenon. There is a great deal of different ways to express a hypothesis. One example is a probability distribution which characterizes the probability of observing a particular data point. Another example are simple rules such as, "if it rains in the morning, then the grass will be wet in the evening". Physical theories, such as the theory of relativity, are further examples of hypotheses.

In general, we do not consider one single hypothesis but a whole space of alternative hypotheses. The simplest non-trivial hypothesis space consists of two alternative hypotheses, such as "The earth is flat" versus "The earth is round". We denote a hypothesis space, which consists of a set of different hypotheses, by $\mathcal{H}$. The key idea behind many ML methods is to choose the best hypothesis out of a large hypothesis space $\mathcal{H}$ according to some performance measure (see Section IV).

In order to quickly search over a large hypothesis space $\mathcal{H}$, it is important to use a computer-friendly (representation of the) hypothesis space $\mathcal{H}$. One example of such a hypothesis space is given by linear predictors $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ with some weight vector $\mathbf{w} \in \mathbb{R}^n$. The resulting hypothesis space is

$$\mathcal{H} := \{h^{(\mathbf{w})}(\mathbf{x}) = \mathbf{w}^T\mathbf{x} : \mathbf{w} \in \mathbb{R}^n\}. \quad (1)$$

Each element $h^{(\mathbf{w})}$ of the hypothesis space $\mathcal{H}$ in (1) is a function from $\mathbb{R}^n$ to $\mathbb{R}$ which maps the feature vector $\mathbf{x}$ to the value $\mathbf{w}^T\mathbf{x}$. However, as indicated by the notation, each of the functions $h^{(\mathbf{w})}$ is fully characterized by the weight vector $\mathbf{w} \in \mathbb{R}^n$. Thus, we can parametrize the hypothesis space (1) using vectors $\mathbf{w}$ from the Euclidean space $\mathbb{R}^n$.

The linear space (1) is only one possible choice for the hypothesis space used in a ML method. We can also use another set of functions $h(\cdot) : \mathcal{X} \to \mathcal{Y}$ as hypothesis space. Decision trees define a hypothesis space using flow chart representations of the mapping $\mathbf{x} \mapsto h(\mathbf{x})$ (see Figure 3). An artificial neural network (ANN) defines a hypothesis space which consists of all functions that are obtained from compositions of matrix operations and simple non-linearities according to a network structure (see Figure 4) .

The choice for the hypothesis space $\mathcal{H}$ has to balance two conflicting requirements:

- It has to be sufficiently large (or rich) such that it contains a predictor map $\hat{h} \in \mathcal{H}$ that is able to represent

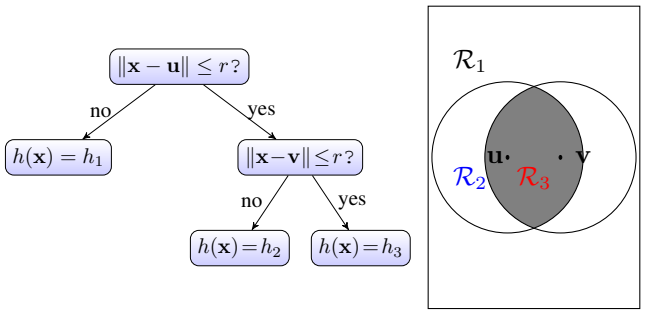

Fig. 3. A decision tree represents a hypothesis h which is constant on subsets $\mathcal{R}_m$, i.e., $h(\mathbf{x}) = h_m$ for all $\mathbf{x} \in \mathcal{R}_m$. Each subset $\mathcal{R}_m \subseteq \mathcal{X}$ corresponds to a leaf node in the decision tree.

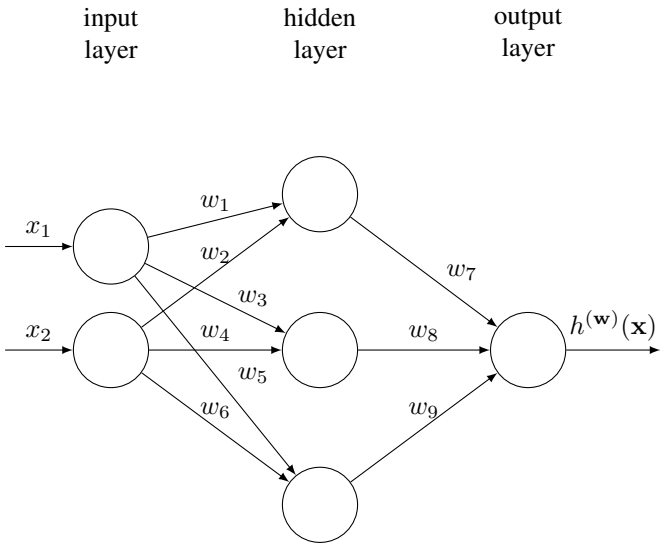

Fig. 4. ANN representation of a predictor $h^{(\mathbf{w})}(\mathbf{x})$ which maps the input (feature) vector $\mathbf{x} = (x_1, x_2)^T$ to a predicted label (output) $h^{(\mathbf{w})}(\mathbf{x})$.

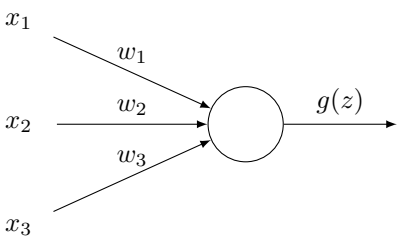

Fig. 5. Each single neuron of the ANN depicted in Figure 4 implements a weighted summation $z = \sum_i w_i x_i$ of its inputs x_i followed by applying a non-linear activation function $g(z)$.

(approximate) the underlying relation between the features and the label of a data point.

- It has to be sufficiently small (compact) such that it can be efficiently searched over to find good predictors during a training phase. This requirement typically necessitates that an arbitrary maps $h(\mathbf{x})$ contained in $\mathcal{H}$ can be evaluated (computed) efficiently [6].

IV. Loss Function

To find good predictor maps we need some quality measure that allows assess a given predictor function $h \in \mathcal{H}$. Many ML methods use the concept of a loss function $\mathcal{L}((\mathbf{x}, y), h)$ that represents the loss (error) incurred by using the predictor h to predict the label y of a data point with features $\mathbf{x}$.

Just like feature space, label space and hypothesis space, also the loss function is a design parameter. In principle, we can use any function $\mathcal{L} : \mathcal{X} \times \mathcal{Y} \times \mathcal{H} \to \mathbb{R}$ that maps a data point $(\mathbf{x}, y)$ and hypothesis $h \in \mathcal{H}$ a number $\mathcal{L}((\mathbf{x}, y), h)$ that represents the loss of using the predictor map h to predict the label $y \in \mathcal{Y}$ of a data point with features $\mathbf{x} \in \mathcal{X}$.

Popular choices are

- the squared error loss

$$\mathcal{L} = (y - \underbrace{h(\mathbf{x})}_{\hat{y}})^2, \tag{2}$$

  for regression problems with label space $\mathcal{Y} = \mathbb{R}$.
- the logistic loss

$$\mathcal{L} = -\log(1 + \exp(-yh(\mathbf{x}))), \tag{3}$$

  for binary classification problems with label space $\mathcal{Y} = \{-1, 1\}$.
- the Huber loss

$$\mathcal{L} = \begin{cases} (1/2)(y - h(\mathbf{x}))^2 & \text{for } |y - h(\mathbf{x})| \leq c \\ c(|y - h(\mathbf{x})| - c/2) & \text{else.} \end{cases} \tag{4}$$

  with some tuning parameter c. The Huber loss can be used for label space $\mathcal{Y} = \mathbb{R}$.

The choice of loss functions is guided by statistical and computational aspects. Learning a predictor by minimizing the squared error loss (2) amounts to *maximum likelihood estimation* if the labels are modeled as

$$y = \bar{h}(\mathbf{x}) + \varepsilon. \tag{5}$$

The model (5) involves some true predictor $\bar{h}$ (which is unknown) and a random variable $\varepsilon \sim \mathcal{N}(0, 1)$ which covers any modeling and measurement (labeling) errors. Thus, if the model (5) accurately describes the observed labels y of data points (which can be considered as statistically independent), the squared error loss (2) is a statistically optimal choice.

Using the logistic loss (3) amounts to maximum likelihood estimation when the labels $y \in \{-1, 1\}$ are modelled as random variables with probability

$$\text{Prob}\{y = 1\} = 1/(1 + \exp(-y\bar{h}(\mathbf{x}))) \tag{6}$$

with some true true predictor $\bar{h}$ (which is unknown).

Aside from their statistical properties, loss functions differ in their computational properties. The squared error loss (2) and

the logistic loss (3) are computationally attractive since they amount to minimizing a differentiable and convex function. Such smooth convex optimization problems can be solved efficiently via (stochastic) gradient descent methods [7], [8].

Sometimes it is beneficial to use non-smooth (non-differentiable) loss functions. In applications where few data points are severely corrupted (e.g., by a broken device) it is beneficial to use the Huber loss (4) [9]. Optimizing non-smooth functions is typically more challenging, requiring more computational resources, compared to optimizing smooth functions.

V. Putting Together The Pieces

Many ML method are obtained by combining particular choices for feature space $\mathcal{X}$ and label space, hypothesis space $\mathcal{H}$ and loss function $\mathcal{L}$. One of the most basic and widely used ML methods is *linear regression*.

Linear regression chooses an optimal linear predictor out of the hypothesis space (1) by minimizing the average squared error loss, or mean squared error,

$$(1/m)\sum_{i=1}^{m}\big(y^{(i)}-h\big(\mathbf{x}^{(i)}\big)\big)^2 = (1/m)\sum_{i=1}^{m}\big(y^{(i)}-\mathbf{w}^T\mathbf{x}^{(i)}\big)^2. \quad (7)$$

The average squared error loss is obtained by comparing the prediction $h(\mathbf{x}^{(i)})$ of the linear predictor $h(\mathbf{x})=\mathbf{w}^T\mathbf{x}$ to the true label $y^{(i)}$ of a data point with features $\mathbf{x}^{(i)}$. Note that the criterion (7), requires m labeled data points with features $\mathbf{x}^{(i)}$ and known labels $y^{(i)}$.

In Figure 7, we depict a set of labeled data points which are used to learn a linear predictor by minimizing the average squared error (7). As hinted at in Section IV, learning a predictor by minimizing the average squared error (7) is statistically optimal if the labels and features are related by the additive Gaussian noise model (5).

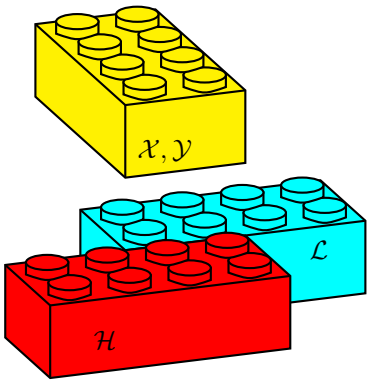

Fig. 6. Many ML method are obtained by combining particular choices for feature space $\mathcal{X}$ and label space, hypothesis space $\mathcal{H}$ and loss function $\mathcal{L}$.

For some datasets the model (7) does not accurately reflect the relation between features and labels. In particular, some data sets contain outliers which have fundamentally different properties compared to the bulk of (clean) data points. We can think of outliers as being the result of exceptional events such as failure of hardware (e.g., broken sensing device).

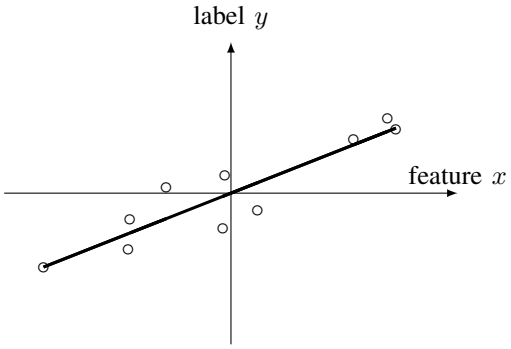

Fig. 7. A data set consisting of labeled data points $(x^{(i)},y^{(i)}$ (depicted as "o") and the linear predictor $h(x)=wx$ (solid line) obtained by minimizing the average squared error (7).

It turns out that learning a predictor by minimizing the squared error loss (7) is not robust against outliers. We illustrate this non-robustness in Figure 8 which depicts a data set that is obtained by corrupting one single data point form the data set shown in Figure 7. Minimizing the average squared error loss on the perturbed data set results in a different linear predictor (solid line in Figure 7) than for the clean data set (dotted line in Figure 7). Thus, if only one single data point is corrupted, minimizing the squared error loss results in significantly different predictors.

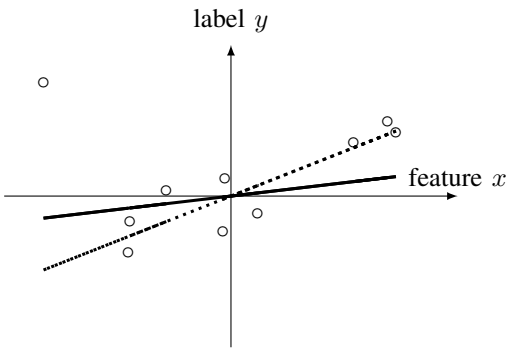

Fig. 8. Corrupted data set (depicted as "o") which is the same as in Figure 7 except for the left-most data point. The solid line represents the linear predictor $h(x)=wx$ (solid line) obtained by minimizing the average squared error (7) on the corrupted data set. The dotted line indicated the predictor obtained from the clean data set (solid line in (7)).

In order to obtain more robustness against few outliers in the data set we might use the Huber loss (4). Figure 9 depicts the same corrupted data set as used in Figure 8. The solid line depicts the linear predictor obtained by minimizing the average Huber loss incurred on the corrupted data set, while the dotted line indicated the linear predictor obtained by minimizing the average Huber loss on the clear data set (depicted as circles in Figure 7).

By comparing Figure 9 with Figure 8, we conclude that using the Huber loss (4) instead of the squared error loss (2) results in a more robust ML method. However, this comes at the price of a more challenging optimization problem since the Huber loss is non-differentiable.

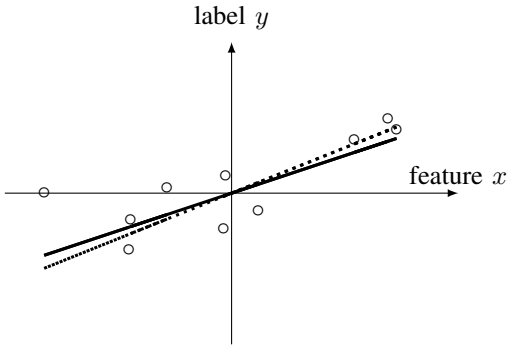

Fig. 9. Corrupted data set (depicted as "$\circ$") which is the same as in Figure 7 except for the left-most data point. The solid line represents the linear predictor $h(x) = wx$ (solid line) obtained by minimizing the average Huber loss (4) on the corrupted data set. The dotted line indicated the predictor obtained from minimizing the average Huber loss on the clean data set (depicted by the circles in Figure 7).

ACKNOWLEDGMENTS

We thank the students of the Aalto courses "Machine Learning: Basic Principles", "Artificial Intelligence" and "Machine Learning with Python" for their constructive and critical feedback. This feedback was instrumental for the author to learn how to teach ML.

REFERENCES

[1] K. Popper, *The Logic of Scientific Discovery*. London, UK: Routledge, 1992.
[2] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.
[3] G. Strang, *Introduction to Linear Algebra*, 5th ed. Wellesley-Cambridge Press, MA, 2016.
[4] P. Bühlmann and S. van de Geer, *Statistics for High-Dimensional Data*. New York: Springer, 2011.
[5] M. Wainwright, *High-Dimensional Statistics: A Non-Asymptotic Viewpoint*. Cambridge: Cambridge University Press, 2019.
[6] P. Austin, P. Kaski, and K. Kubjas, "Tensor network complexity of multilinear maps," *arXiv*, 2018.
[7] Y. Nesterov, *Introductory lectures on convex optimization*, ser. Applied Optimization. Kluwer Academic Publishers, Boston, MA, 2004, vol. 87, a basic course. [Online]. Available: http://dx.doi.org/10.1007/978-1-4419-8853-9
[8] E. Hazan, *Introduction to Online Convex Optimization*. Now Publishers Inc., 2016.
[9] P. J. Huber, *Robust Statistics*. New York: Wiley, 1981.