Fundamental Framework of Machine Learning

Sunmook Choi felixchoi@korea.ac.kr

August 14, 2023

1 Risk Minization

1.1 Empirical Risk Minimization

We describe a mathematical (statistical) formulation of supervised learning.

Definition 1 (Supervised Learning)**.** *Assume that there is some* ***fixed but unknown*** *joint distribution $P(X, Y)$ on $\mathcal{X} \times \mathcal{Y}$ where $X \in \mathcal{X}$ is a random (feature) vector and $Y \in \mathcal{Y}$ is a random target vector. Choose a loss function $\ell \colon \mathcal{X} \times \mathcal{Y} \times \mathcal{Y} \to [0, \infty)$ with the property $\ell(x, y, y) = 0$ for all $x \in \mathcal{X}$ and $y \in \mathcal{Y}$. We try to find a hypothesis (or a model) $h \in \mathcal{H}$ that describes the relationship between X and Y for some hypothesis class $\mathcal{H}$, i.e.,*

$$h^* = \arg\min_h R(h) = \int \ell(x, y, h(x)) P(x, y)\, dxdy. \tag{1.1}$$

Here, we call $R(h)$ *the expected risk*. Unfortunately, the distribution P is unknown in most practical cases, i.e., the integration is intractable. Instead of having access to P, we only have a set of observations, called training data, $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$, where $(x_i, y_i) \sim P$ for all $i = 1, \ldots, n$. Using this training data $\mathcal{D}$, we may approximate P by *the empirical density*

$$P_{emp}(x, y) = \frac{1}{n} \sum_{i=1}^{n} \delta_{(x_i, y_i)}(x, y), \tag{1.2}$$

where $\delta_{(x_i, y_i)}(x, y)$ denotes the δ-distribution, satisfying $\int \delta_{(x', y')}(x, y)\, f(x, y)\, dxdy = f(x', y')$. Using this empirical density, we now approximate expected risk by *empirical risk*:

$$R_\delta(h) = \int \ell(x, y, h(x))\, P_{emp}(x, y)\, dxdy = \frac{1}{n} \sum_{i=1}^{n} \ell(x_i, y_i, h(x_i). \tag{1.3}$$

In supervised learning, we find (or learn) $h \in \mathcal{H}$ by minimizing the empirical risk, and we call this process the *empirical risk minimization* (ERM).

Empirical risk minimization (ERM) is an efficient way to approximately minimize expected risk. In most of machine learning setups, after choosing a hypothesis class $\mathcal{H}$, an optimization algorithm is required to solve the problem:

$$h^* = \arg\min_{h \in \mathcal{H}} R_\delta(h) \tag{1.4}$$

1.1.1 Underfitting and Overfitting

Let's consider about the choice of a hypothesis class $\mathcal{H}$. If the class $\mathcal{H}$ has too small capacity, any model in this class cannot reduce the empirical risk sufficiently so that the model would perform terribly for unseen test samples. On the other hand, if the class $\mathcal{H}$ has too large capacity, then there should be a model $h \in \mathcal{H}$ such that it can 'memorize' all the data samples so that the empirical risk becomes zero. That is, ERM will find a model that memorizes the training samples rather than generalizes the training samples. Hence this may lead to the undesirable behavior of h outside the training data, which gives rise to great loss on test dataset. The former case is called *underfitting* while the latter is called *overfitting*.

When it comes to deep learning, the model class would be a subset of 'deep' neural networks that contains at least thousands of parameters. Therefore, overfitting problem is more common in deep learning practices rather than underfitting. The following section will describe how to handle the overfitting problems.

1.2 Structural Risk Minimization

1.3 Vicinal Risk Minimization

Due to the undesired behavior of ERM, especially overfitting, it is required to improve empirical density P_{emp}. One way is to replace the delta function $\delta_{(x_i,y_i)}(x,y)$ or $\delta(x = x_i, y = y_i)$ that makes ERM constrained only on the training samples (so the model fails to generalize the data).

Vicinal risk minimization (VRM) replaces the delta function into some probability distribution $\nu_{(x_i,y_i)}(\tilde{x},\tilde{y})$ or $\nu(\tilde{x},\tilde{y} \mid x_i, y_i)$ which describes the vicinity of the point (x_i, y_i). This

is a way to consider not only samples that are observed but also the virtual feature-target pairs $(\tilde{x}, \tilde{y})$ of given samples (x_i, y_i). The formalization of VRM is described below.

Definition 2 (Vicinal Risk Minimization (VRM))**.** *Given a vicinity distribution ν, the true distribution P (in the ERM) is estimated by*

$$P_\nu(\tilde{x}, \tilde{y}) = \frac{1}{n} \sum_{i=1}^{n} \nu(\tilde{x}, \tilde{y} \,|\, x_i, y_i). \tag{1.5}$$

Then the vicinal risk is defined to be

$$R_\nu(h) = \frac{1}{m} \sum_{i=1}^{m} \ell(\tilde{x}_i, \tilde{y}_i, h(\tilde{x}_i)) \tag{1.6}$$

and the goal of VRM is to find $h \in \mathcal{H}$ that minimizes the vicinal risk, that is,

$$h^* = \arg\min_{h \in \mathcal{H}} R_\nu(h). \tag{1.7}$$

Vicinal risk minimization is also known as 'data augmentation'. Let's take a natural example of vicinity distribution, Gaussian vicinities:

$$\nu(\tilde{x}, \tilde{y} \,|\, x_i, y_i) = \mathcal{N}(\tilde{x} \,|\, x_i, \sigma^2) \cdot \delta(\tilde{y} = y_i). \tag{1.8}$$

This Gaussian vicinity distribution treats the data $\tilde{x}$ in a neighborhood of x_i to have the same target y_i as x_i. Here, the parameter σ controls the scale of density estimate. Note that the extreme case $\sigma = 0$ leads to ERM. More examples can be found in the paper [2].

2 Perceptron

The Perceptron is an artificial neural networks for binary classifiers. The algorithm was invented in 1958 by Frank Rosenblatt. Although the perceptron initially seemed promising, it was quickly proved that perceptrons could not be trained to recognize many kinds of patterns.

2.1 Assumptions of Perceptron

Definition 3 (Hyperplane)**.** *Let all data points lie on the d-dimensional space $\mathbb{R}^d$ with standard inner product. Here, we call $\mathbb{R}^d$ an ambient space. A hyperplane is a subspace of dimension $d-1$, or equivalently, of codimension 1, only possibly shifted from the origin by a vector. That is, for any $\boldsymbol{w} \in \mathbb{R}^d$ and $b \in \mathbb{R}$, the following set is a hyperplane in $\mathbb{R}^d$:*

$$\mathcal{H} = \left\{ \boldsymbol{x} \in \mathbb{R}^d \colon \boldsymbol{w}^\top \boldsymbol{x} + b = 0 \right\} \tag{2.1}$$

Definition 4 (Linearly Separable Dataset)**.** *Let $\mathcal{D} = (\boldsymbol{x}_i, y_i)$ be a dataset with $y_i \in \{+1, -1\}$. We say that $\mathcal{D}$ is linearly separable if there is a hyperplane $(\boldsymbol{w}, b)$ such that*

$$y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) > 0, \quad \forall (\boldsymbol{x}_i, y_i) \in \mathcal{D}. \tag{2.2}$$

Remark 5. *In low dimensional spaces, the linearly separable condition may not hold. In high dimensional spaces, on the other hand, data points tend to be far away from each other so that it almost always hold.*

Given a linearly separable dataset, the perceptron algorithm aims to find a hyperplane that separates one class from the other. For convenience, we usually use a trick to absorb the term b. Instead of using $\boldsymbol{x}_i$ and $\boldsymbol{w}$, we use

$$\boldsymbol{x}_i \leftarrow \begin{pmatrix} \boldsymbol{x}_i \\ 1 \end{pmatrix} \quad \text{and} \quad \boldsymbol{w} \leftarrow \begin{pmatrix} \boldsymbol{w} \\ 1 \end{pmatrix}. \tag{2.3}$$

Then, the hyperplane becomes $\mathcal{H} = \left\{ \boldsymbol{x} \in \mathbb{R}^d \colon \boldsymbol{w}^\top \boldsymbol{x} = 0 \right\}$ and the linearly separable condition Eq. (2.2) becomes $y_i \left(\boldsymbol{w}^\top \boldsymbol{x}_i \right) > 0$.

2.2 Algorithm

The perceptron algorithm is an iterative algorithm that finds a hyperplane that separates one class from the other.

Perceptron

Initialize $\boldsymbol{w} = \boldsymbol{0}$
while **True**:
    $m = 0$ # of misclassified points
    for $(\boldsymbol{x}, y) \in \mathcal{D}$
        if $y \left(\boldsymbol{w}^\top \boldsymbol{x} \right) \leq 0$ when $(\boldsymbol{x}, y)$ lies in the wrong side
            $\boldsymbol{w} \leftarrow \boldsymbol{w} + y\boldsymbol{x}$
            $m \leftarrow m + 1$
    if $m = 0$ then **break**

In the algorithm, if a data point $(\boldsymbol{x}, y)$ is not classified well, the weight vector $\boldsymbol{w}$ is updated as follows:

- if $y = +1$ and $\boldsymbol{w}^\top \boldsymbol{x} \leq 0$, then $\boldsymbol{w}^\top \boldsymbol{x}$ increases because $(\boldsymbol{w} + \boldsymbol{x})^\top \boldsymbol{x} = \boldsymbol{w}^\top \boldsymbol{x} + \boldsymbol{x}^\top \boldsymbol{x} > \boldsymbol{w}^\top \boldsymbol{x}$;

- if $y = -1$ and $\boldsymbol{w}^\top \boldsymbol{x} \geq 0$, then $\boldsymbol{w}^\top \boldsymbol{x}$ decreases because $(\boldsymbol{w} - \boldsymbol{x})^\top \boldsymbol{x} = \boldsymbol{w}^\top \boldsymbol{x} - \boldsymbol{x}^\top \boldsymbol{x} < \boldsymbol{w}^\top \boldsymbol{x}$.

Note that $\boldsymbol{x} \neq 0$ because there is a nonzero entry in the last coordinate. Therefore, we can see that the update equation $\boldsymbol{w} \leftarrow \boldsymbol{w} + y\boldsymbol{x}$ is a way to make all data pairs $(\boldsymbol{x}, y)$ satisfy $y\,(\boldsymbol{w}^\top \boldsymbol{x}) > 0$.

2.3 Convergence of the algorithm

In this section, we prove the convergence of the perceptron algorithm. Without loss of generality, we assume that $\|\boldsymbol{w}^*\| = 1$. Since the size of the dataset is finite, we may also assume that $\|\boldsymbol{x}_i\| \leq 1$ for all $(\boldsymbol{x}_i, y_i) \in \mathcal{D}$ by rescaling. Let γ be the distance between the hyperplane and the closest point, that is,

$$\gamma = \min_{(\boldsymbol{x}_i, y_i) \in \mathcal{D}} |\, \boldsymbol{x}_i^\top \boldsymbol{w}^*| > 0. \tag{2.4}$$

This number γ is also known as the margin. This assumptions can be illustrated as in Fig. 2.3.

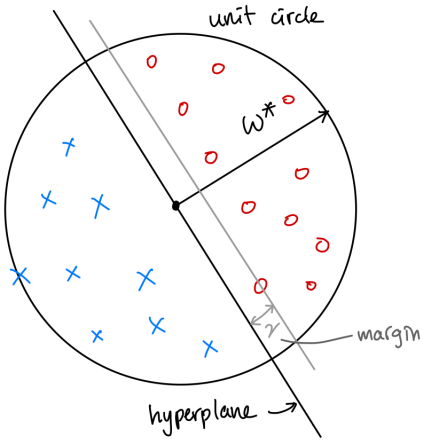

Figure 1: Assumptions of Perceptron

Theorem 6. *The perceptron algorithm makes at most* $1/\gamma^2$ *mistakes.*

Recall that we make an update $\boldsymbol{w} \leftarrow \boldsymbol{w} + y\boldsymbol{x}$ when we misclassified, i.e., $y\,(\boldsymbol{w}^\top \boldsymbol{x}) \leq 0$. Let $\boldsymbol{w}$ be a vector during optimization while $\boldsymbol{w}^*$ is the optimal hyperplane. Assume that a pair $(\boldsymbol{x}, y) \in \mathcal{D}$ is misclassified. We want to take a look at the change of two values: $\boldsymbol{w}^\top \boldsymbol{w}^*$ and $\boldsymbol{w}^\top \boldsymbol{w}$.

- The value $\boldsymbol{w}^\top \boldsymbol{w}^*$:

$$(\boldsymbol{w} + y\boldsymbol{x})^\top \boldsymbol{w}^* = \boldsymbol{w}^\top \boldsymbol{w}^* + y(\boldsymbol{w}^*)^\top \boldsymbol{x} \geq \boldsymbol{w}^\top \boldsymbol{w}^* + \gamma \tag{2.5}$$

  We can see the value $\boldsymbol{w}^\top \boldsymbol{w}^*$ grows by at least γ per each update.

- The value $\boldsymbol{w}^\top \boldsymbol{w}$:

$$(\boldsymbol{w} + y\boldsymbol{x})^\top (\boldsymbol{w} + y\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{w} + 2y\boldsymbol{w}^\top \boldsymbol{x} + y^2 \boldsymbol{x}^\top \boldsymbol{x} \leq \boldsymbol{w}^\top \boldsymbol{w} + 1 \tag{2.6}$$

  We can see the value $\boldsymbol{w}^\top \boldsymbol{w}$ grows by at most 1 per each update.

After M updates, since $\boldsymbol{w}$ was initialized to the zero, we obtain that

$$M\gamma \leq \boldsymbol{w}^\top \boldsymbol{w}^* = |\boldsymbol{w}^\top \boldsymbol{w}^*| \leq \|\boldsymbol{w}\| \cdot \|\boldsymbol{w}^*\| = \|\boldsymbol{w}\| = \sqrt{\boldsymbol{w}^\top \boldsymbol{w}} \leq \sqrt{M}. \tag{2.7}$$

Therefore, we have $M \leq 1/\gamma^2$, implying that the iteration terminates in finite number of steps. Notice that the number of steps only depends on the margin.

2.4 XOR problem

XOR (exclusive OR) problem is a classic problem in AI. In 1969 a famous book entitled *Perceptrons* by Minksy and Papert showed that it was impossible for perceptrons to learn an XOR function. Single layer perceptrons are capable of learning *linearly* separable patterns, that is, it is capable of separating data points with a single line. However, many data, including XOR inputs, are not linearly separable.

3 Multi-layer Perceptron

3.1 Gradient Descent Method

3.2 Algorithm

3.3 Vanishing Gradient Problem

4 More about Deep Learning

4.1 Activations

4.2 Batch Normalization

4.3 Dropout

4.4 Weight Initialization