Artificial Neural Networks
Lecture Notes

Prof. Dr. Sen Cheng

Winter Semester 2019/20

2 Optimization problems

Fitting a function is an optimization problem and has the following components: data, model class, loss function, and optimization algorithm.

2.1 Data

Since we focus on supervised learning in this class, the data consists of *independent variables* and *dependent variables*. Variables can be continuous, binary or categorical. The independent variables can also be thought of as the inputs (of the function):

$$X = (x_1, x_2, \ldots, x_N) \tag{1}$$

Each x_i can be a vector, i.e., a sequence of numbers, x_{ij}, where $j = 1, \ldots, m$. This formulation is quite general and many types of information can be represented this way. For instance, an image can be represented as vector of numbers by dividing the image into $a \times b$ pixels (Fig. 1), calculating the average luminance in each pixel on a scale from 0 to 1, and then concatenating the b columns into a single vector of length ab. For future use, we define the mean as

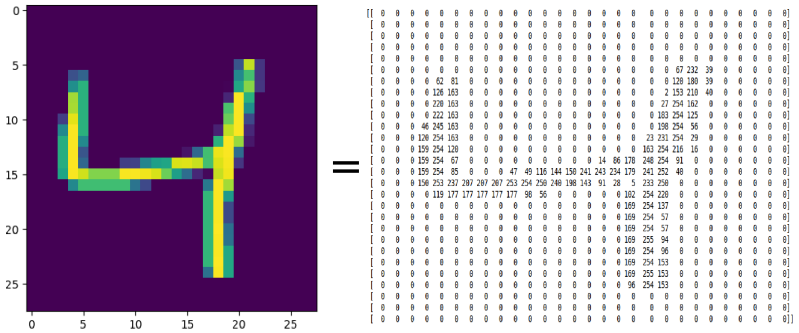

Figure 1: Representation of an image as vector. $18x18 = 324$ pixels or dimensions.

$$\bar{x} = \frac{1}{N} \sum_{i=1}^{N} x_i \tag{2}$$

and variance

$$\sigma^2(X) = \frac{1}{N} \sum_{i=1}^{N} (x_i - \bar{x})^2 \tag{3}$$

The dependent variables are often thought of as the outputs (of the function):

$$Y = (y_1, y_2, \ldots, y_N) \tag{4}$$

Each y_i can be a vector, i.e., a sequence of numbers, y_{ij}, $j = 1, \ldots, n$.

2.2 Model class

The choice of the model class is key to function fitting. The success of the fitting depends on whether the model class is a good description of the data. For convenience, all parameters of the model class are summarized in the vector θ. The function is said to be *parametrized* and often denoted by f_θ.

Examples for functions:

class	**linear**	**polynomial**	**exponential**	**ANN**
$f(x)$	$mx+b$	$\sum a_n x^n$	$\exp(ax+b)$	$\cdots$
θ	(m,b)	$(a_0,\ldots,a_k)$	(a,b)	$\cdots$

2.3 Loss function

The *loss function* measures how well a function is describing the data. The lower the loss, the better.

$$L(\theta,X,Y)=\sum_{i=1}^{N} l\left(y_i,\hat{y}_i\right) \tag{5}$$

where i is the sample index and $\hat{y}_i = f_\theta(x_i)$.

A popular choice for the loss function is a class of functions called *l-norm* (or *Minkowski distance*), because for $k \geq 1$ these function can be used to measure distances:

$$l_k\left(|y_i-\hat{y}_i|\right)=\left(\sum_{j=1}^{n}|y_{ij}-\hat{y}_{ij}|^k\right)^{1/k}, \tag{6}$$

where j is the dimension index. The l_2-norm $\left(l_2=\sqrt{\sum_j (y_{ij}-\hat{y}_{ij})^2}\right)$ is the most commonly used loss function. It is also known as the *Euclidian* norm or *Euclidian* distance. The l_1-norm $\left(l_1=\sum_j |y_{ij}-\hat{y}_{ij}|\right)$ is also known as the *Manhattan* distance. In the limiting case $n \to \infty$,

$$l_\infty=\lim_{k\to\infty}\left(\sum_{j=1}^{n}|y_{ij}-\hat{y}_{ij}|^k\right)^{1/k}=\max_{j=1}^{n}|y_{ij}-\hat{y}_{ij}| \tag{7}$$

The choice of the loss function is important since it determines what it means for a function to describe the data well. In other words, the result of fitting strongly depends on the loss function. The loss function has to be appropriate for the task. For instance, the above loss functions work well for regression[1] type problems. We will encounter different types of loss functions for different tasks later.

2.4 Optimization

The objective of function fitting is to find $\hat{\theta}$ such that the total loss function Eq. 5 is minimal, i.e.,

$$\hat{\theta}=\arg\min_{\theta} L(\theta,X,Y). \tag{8}$$

Since the derivative at minima (and at maxima) vanish, we can find $\hat{\theta}$ by solving the equation

$$\frac{\mathrm{d}}{\mathrm{d}\theta}L(\theta,X,Y)=0 \tag{9}$$

for θ. Note that Eq. 9 is a necessary condition, but not sufficient. The second derivative also has to be positive to ensure that we found a minimum. However, this step is often skipped since computing the second derivative is often very difficult. Fitting the parameters is often referred to as *training* the model, especially when the model is more complex. Equivalently, the model is sometimes said to *learn* from the data.

[1] We'll define this term later.

2.4.1 Analytical solution

In some simple cases Eq. 9 can be solved analytically. We will do this for linear regression in the next lecture.

2.4.2 Numerical solution: Newton's method

Most of the time, however, Eq. 9 cannot be solved analytically. We could use Newton's method to find the zeros of a function $g(x)$ numerically. Starting with a guess of the solution x_0, we approximate the function by a line going through the point $(x_0, g(x_0))$ (Fig. 2). We then find the root of that line

$$y = mx_1 + b \stackrel{!}{=} 0 \tag{10a}$$

$$x_1 = -\frac{b}{m} \tag{10b}$$

The y-offset b can be obtained by inserting the point into the equation for the line.

$$g(x_0) = g'(x_0)x_0 + b \tag{11a}$$

$$b = g(x_0) - g'(x_0)x_0 \tag{11b}$$

$$x_1 = -\frac{g(x_0) - g'(x_0)x_0}{g'(x_0)} \tag{12a}$$

$$= x_0 - \frac{g(x_0)}{g'(x_0)} \tag{12b}$$

This process is then iterated until it converges. For solving Eq. 9, we therefore use the iterative formula

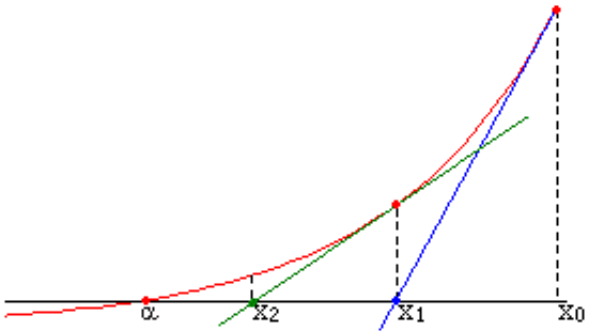

Figure 2: Newton's method for finding the root of a function. Source: `https://commons.wikimedia.org/wiki/File:Methode_newton.png`, User:Cham. Released into the public domain.

$$\theta_{n+1} = \theta_n - \frac{L'(\theta_n, X, Y)}{L''(\theta_n, X, Y)} \tag{13}$$

However, most of the time the second derivative is difficult to compute, and Newton's method only works well for convex problems.

2.4.3 Gradient descent

A versatile approach to find a minimum is *gradient descent*. It only requires that we are able to compute the *gradient* of the loss function $\frac{\mathrm{d}}{\mathrm{d}\theta} L(\theta, X, Y)$. To emphasize that the gradient is a vector the operator $\frac{\mathrm{d}}{\mathrm{d}\theta}$ is often written as ∇ ("nabla" operator) or as $\left(\frac{\partial}{\partial\theta_1}, \frac{\partial}{\partial\theta_2}, ..., \frac{\partial}{\partial\theta_k}\right)^T$, where k is the number of parameters of the model. The gradient

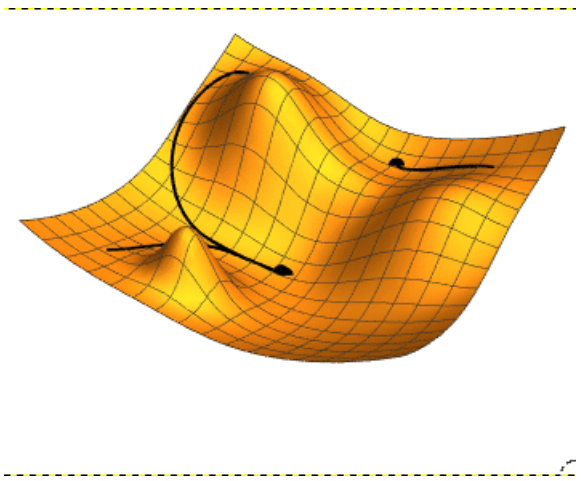

Figure 3: Gradient descent. Source: `https://commons.wikimedia.org/wiki/File:Gradient_descent.gif`. Author: Jacopo Bertolotti. Released into the public domain.

indicates the direction of steepest increase of the function. By moving into the opposite direction, i.e., $-\nabla L$, we can find a local minimum of the loss function (Fig. 3). Interations:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta L(\theta_t, X, Y). \tag{14}$$

η is the *learning rate*. Finding the appropriate learning rate for a given dataset is difficult (Fig. 4). Monitor the loss function in each iteration to spot potential issues.

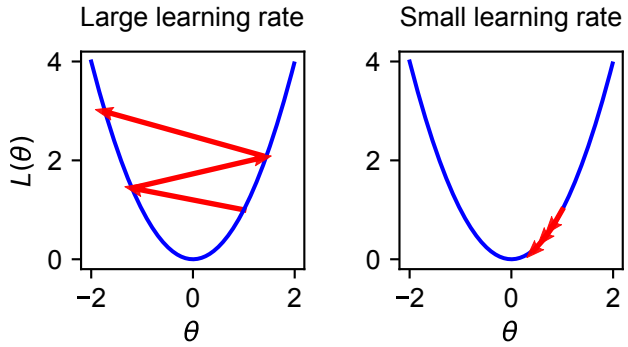

Figure 4: The effect of the learning rate. When it's too high, gradient descent may not converge (left) and when it is too small gradient descent requires much larger N (right).

A fundamental limitation of any optimization based on gradients is that the method can get stuck in *local minima*. While there are methods for avoiding this, there are never guarantees that they will find the global minimum.

2.4.4 Stochastic gradient descent, minibatch

The standard (or *batch*) gradient descent algorithm requires the evaluation of the gradients for all N datapoints before the parameter estimate can be updated

$$\theta_{t+1} = \theta_t - \eta \sum_{i=1}^{N} \nabla_\theta l\left(y_i, \hat{y}_i\right) \tag{15}$$

This is inefficient for very large datasets, because the parameters θ_t are kept fixed in the computation of all $\hat{y}_i$ and updated only after all N gradients are computed. This can take a long time, if the gradients are computationally costly. By contrast, if we updated the parameters after each gradient is computed, our parameter estimates would converge faster, but that would also make the fitting more sensitive to noise. That's why this method is called *stochastic gradient descent* (SGD).

$$\theta_i = \theta_{i-1} - \eta \nabla_\theta l\left(y_i, \hat{y}_i\right) \tag{16}$$

where i is iterated through the entire dataset. Typically multiple passes through the dataset are required. After each pass the data should be shuffled to prevent cycles.

To reduce the effect of the noise, but still speed convergence like in SGD, one can sum the gradients over mini-batches of size b

$$\theta_t = \theta_{t-1} - \eta \sum_{i=(t-1)b+1}^{tb} \nabla_\theta l\left(y_i, \hat{y}_i\right) \tag{17}$$

2.5 Convex optimization problems

Incremental optimization algorithm can always get stuck in a local minimum and therefore fail to find the globally optimal solution. However, for one class of functions convergence to the global optimal solution in guaranteed, if the learning rate is chosen appropriately.

Generally, an area is said to be *convex*, if a line connecting any two points within the area lies wholly within the area. A function is said to be convex, if the area above the graph of the function is convex. If a function $f(x)$ has a second derivative, $f(x)$ is convex, if $f''(x) \geq 0$ everywhere in its domain. If f is a function of multiple (m) variables and is twice differentiable, then the second derivative $\nabla^2 f(x)$ is an $m \times m$ matrix, called the *Hessian*. $f(x)$ is convex, if the Hessian is positive semi-definite everywhere in its domain. An $m \times m$ matrix M is semi-definite, if for any vector $z \in R^m$: $z^T M z \geq 0$. For convex functions, a local minimum is also a global minimum.

Therefore, in a *convex optimization problem*, i.e. one in which the loss function is convex, if gradient descent converges, the solution will be globally optimal.