Deep Learning Lab Course Assignment 1

Assignment 1: Neural Network

Hendrik Vloet

November 2, 2017

1 Brief Summary of MLPs in General

A Deep Feedforward Network or Multilayer Perceptron is a function approximator. This approximation can be used for various tasks within Machine Learning, e.g. classification. This means we have a Dataset $\mathcal{D}$, consisting of some kind of measurement data $\mathbf{X}$ which has the dimensions $N \times D$ and in case of supervised learning some label vector $\mathbf{Y}$ which is D-dimensional.
The classifier MLP now maps (with the approximation function f and some parameters $\mathbf{\Theta}$) one input data vector $\mathbf{x}$ (which is of $1 \times D$ dimensions large) to his corresponding label y. Formalized it is as in the following:

$$\mathbf{y} = f(\mathbf{x}; \mathbf{\Theta})$$

In case of a network, this corresponds to the nesting of several functions within each other that finally result in the best approximation of the classification. The nested functions are chained together and organized as so called layers. For example, we can use three layers, all using different functions which get chained up:

$$f(\mathbf{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\mathbf{x})))$$

1.1 Activation Functions and Derivatives

The above mentioned functions are also called activation functions and can be of various forms, depending on the task at hand. The are used to keep up certain properties of the data or even emphasize properties like non-linearities. The use of activation during the forward pass and during the backpropagation shall be illustrated with a short example of a 2-Layer MLP.

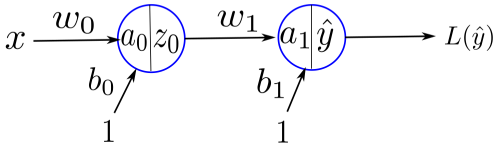


Figure 1.1: Example of a 2-layer MLP, taken for Machine Learning Lecture in Summer term 2017

1.2 Examples of Activation Functions and Derivatives

As mentioned above, activation functions can be of various forms. Below is a small list of commonly used functions and their derivatives:

$$
\begin{aligned}
\textbf{Linear:}\quad & f(\mathbf{x}) = \mathbf{x} \\
& \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} = \mathbf{1} \\
\textbf{ReLU:}\quad & f(\mathbf{x}) = \max(0, x) \\
& \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} = \begin{cases} \mathbf{0} & \text{for } x < 0 \\ \mathbf{1} & \text{for } x \geq 0 \end{cases} \\
\textbf{Sigmoid:}\quad & f(\mathbf{x}) = \text{sigmoid}(\mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{x})} \\
& \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} = \text{sigmoid}(\mathbf{x}) \cdot (1 - \text{sigmoid}(\mathbf{x})) \\
\textbf{Tanh:}\quad & f(\mathbf{x}) = \tanh(\mathbf{x}) = \frac{2}{1 + \exp(-2\mathbf{x})} - 1 \\
& \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} = 1 - \tanh(\mathbf{x})^2
\end{aligned}
$$

1.3 Forward Pass

The forward pass goes from input to the output of the network and therefore computes the prediction of some input point x, combines this data with some weights w and bias b to get a (hopefully) correct prediction $\hat{y}$. Here the output of a layer is called z and the activation function is $h(\cdot)$ Formalized:

$$\begin{aligned} \textbf{Layer 0:}\quad & a_0 = x \cdot w_0 + b_0 \\ & z_0 = h_0(a_0) \\ \textbf{Layer 1:}\quad & a_1 = z_0 \cdot w_1 + b_1 \\ & \hat{y} = h_1(a_1) \end{aligned}$$

1.4 Backward Pass

Similar to the forward pass, the backward pass uses the information from the output to propagate back information to the input. The goal of this is to compute the gradient of the network by applying the chain rule relentlessly.
First one calculates the gradient of the used loss function with respect to the prediction and then the gradient of the loss function with respect to the activation of the last layer and so on and so forth. According to the given example, this would look like this:

$$\begin{aligned} \textbf{Layer 1:}\quad & \frac{\partial L}{\partial \hat{y}} & & \frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_1} \frac{\partial a_1}{w_1} \\ & \frac{\partial L}{\partial a_1} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_1} & & \frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_1} \frac{\partial a_1}{b_1} \\ \textbf{Layer 0:}\quad & \frac{\partial L}{\partial z_0} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_1} \frac{\partial a_1}{\partial z_0} & & \frac{\partial L}{\partial w_0} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_1} \frac{\partial a_1}{\partial z_0} \frac{\partial z_0}{\partial a_0} \frac{\partial a_0}{\partial w_0} \\ & \frac{\partial L}{\partial a_0} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_1} \frac{\partial a_1}{\partial z_0} \frac{\partial z_0}{\partial a_0} & & \frac{\partial L}{\partial b_0} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_1} \frac{\partial a_1}{\partial z_0} \frac{\partial z_0}{\partial a_0} \frac{\partial a_0}{\partial b_0} \end{aligned}$$

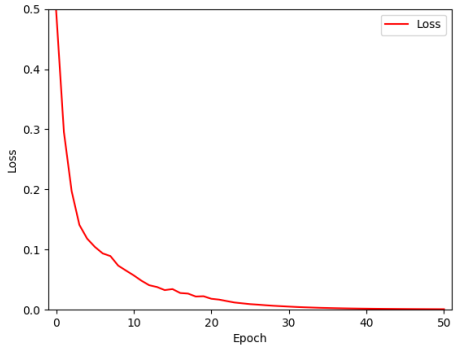
2 Setup

Training Setup:

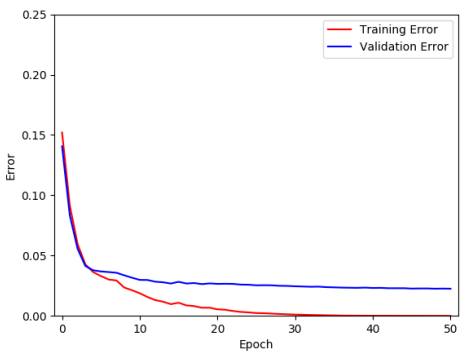
- In total 4 layers used:
  - Layer 1 (Input layer)
    - Units: 100
    - initial standard deviation: 0.01
    - Activation function: ReLU
  - Layer 2
    - Units: 100
    - initial standard deviation: 0.01
    - Activation function: ReLU
  - Layer 3
    - Units: 10
    - initial standard deviation: 0.01
    - Activation function: None
  - Layer 4 (Output Layer)
    - Units: 100
    - initial standard deviation: 0.01
    - Loss function: Softmax/linear loss
- Epochs: 50
- Batch size: 64
- learning rate: 0.01
- samples used: all training samples used

3 Results

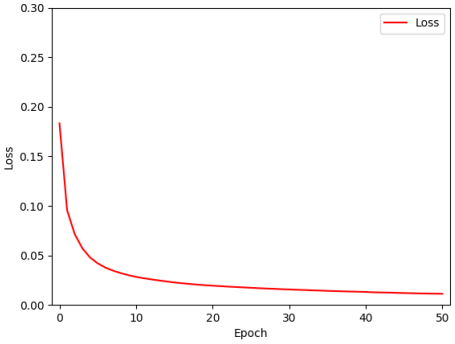
Loss curve with softmax output function



Learning curve with softmax output function. Overfitting starts around the 8th epoch



Loss curve with linear output function.



Learning curve with linear output function. Overfitting starts around the 8th epoch