MNIST Classification Using Multi-layered Perceptrons

akari@cinnamon.is

December 2019

1 Introduction

In the context of contemporary Artificial Intelligence, it is of absolute necessity that we mention deep learning. Indeed, as far as deep learning models continue to be developed, we believe that it will be beneficial for you to understand the architecture of a multi-layered perceptron[1]. This consists of:

- The linear algebra for a feed-forward network.
- The multivariate calculus for back-propagation.
- The implementation of both of those.

The next sections will be organized as follows: Section 2 performs the forward operation and 3 describe the the learning mechanism of the neural network.

2 Forward Propagation

The network has two phase: forward propagation and back-propagation, the former is used in both learning and inference while the latter is used only in learning stage. We follow a cleaned version of the network in [2], let $\mathbf{x} \in \Re^D$ be the input, $\mathbf{W} \in \Re^{F \times D+1}$ be the weight, $\phi : \Re^D \to \Re^{D+1}$ be the bias adding function (i.e $\phi = [1; x_1; x_2; \ldots; x_D]$) and $f : \Re^F \to \Re^F$ be the activation function and $\mathbf{y} \in \Re^F$. In short:

$$\mathbf{y} = f\left(\mathbf{w}^T \phi\left((x)\right)\right) \tag{1}$$

This combination makes up a layer in neural network, depicted in Figure 1. The stacking of cascaded layers leading to the architecture of a multi-layered perceptron in Figure 2. Your first task is implementing a neural network architecture satisfying the following criterias:

- Activation function for hidden and output layer is sigmoid function.
- The network should be a two-layer neuralnetwork (one hidden layer with nhidden units and 10 output units). The class predicted by the network is one corresponding to the most highly activated output unit.

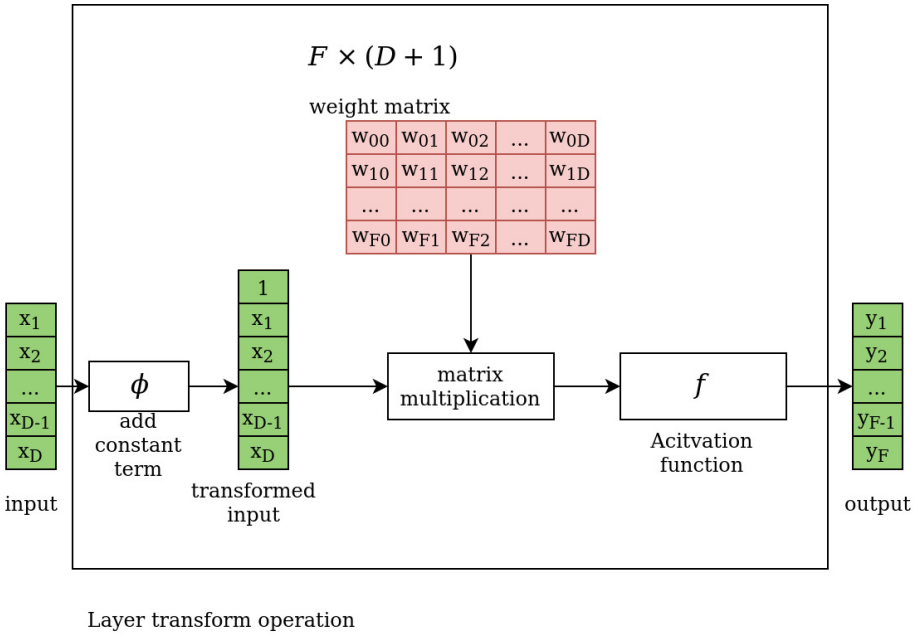

Figure 1: A layer in a deep neural network

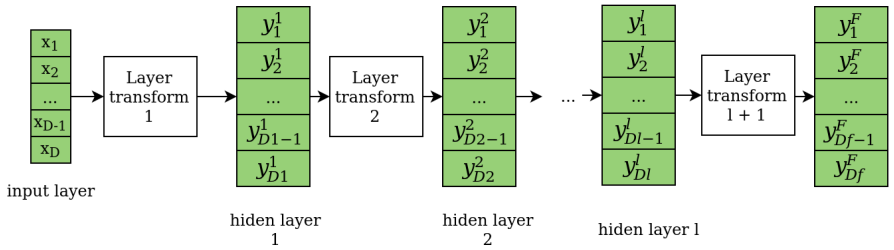

Figure 2: Deep neural network (Multi-layered Perceptron) is formed by stacking the layers

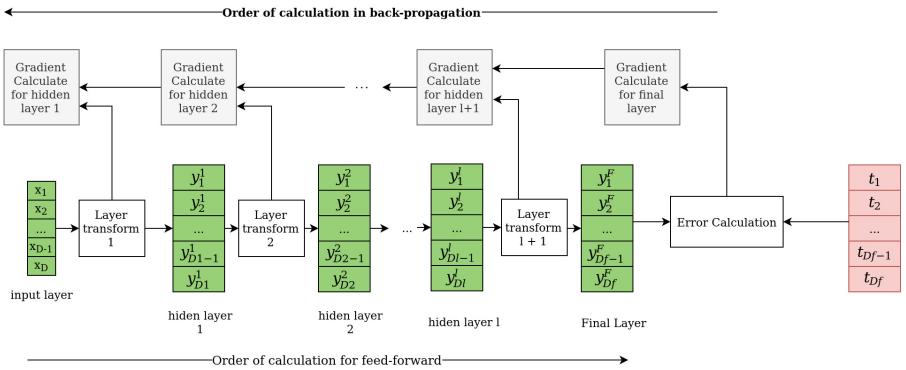

Figure 3: Back-propagation flow on the network

- Initial weights: Your network should start off with small random positive and negative weights (-0.5 $\leq w \leq$ 0.5).

3 Loss Function and Back-Propagation

The loss function is a distance measurement between the predicted (the output of the network) and the target. Let us use Kaggle preprocessed MNIST `https://www.kaggle.com/c/digit-recognizer/`, the common loss function $\mathbf{L}(\mathbf{y}, \mathbf{t})$ for the classification problem is cross entropy:

$$\mathbf{L}(\mathbf{y}, \mathbf{t}) = -\sum_{i=1}^{D_f} t_i \log(y_i) \tag{2}$$

Let us denote the set of all optimizable weight Θ, the goal of learning for neural network is find Θ such that:

$$\Theta = \arg\min_{\Theta} \mathbf{L}(\mathbf{y}, \mathbf{t}) \tag{3}$$

To do this, gradient descent is normally the choice, it is made by iterative through the two steps:

- $\Theta_{new} = \Theta - \eta \nabla_{\Theta} \mathbf{L} + \gamma \times (\Theta - \Theta_{old})$ where η is the so-called learning rate, γ is the momentum factor.
- Update Θ and get new $\mathbf{y}$

For each complete iteration through the entire dataset, we will call it an epoch. Your tasks:

- Derive the gradient formula for each weight set in each layer.
- Use it and implement the back propagation and use it to train the network with the experiments enumerated below.

- Split MNIST into training and testset by the ratio of $8:2$
- Experiment with different hidden dimmension (number of hidden units): 20, 50, 100.
- Experiment with different epochs: 20, 50, 100.
- Experiment with different input scaling: 0-to-1, 0-to-255, etc.
- Experiment with different learning rate: 0.1, 0.001, etc.
- Experiment with different momentumn factors: 0.9, 0.8, 0.1, 0.0.

4 Report

Report all of your experiments (with loss and accuracy curve) and answer the following questions:

- How does the number of hidden units affect the training?
- Is there any correlation between the number of hidden units and the number of epochs?
- Did your network overfit? Point out by your experiment results.
- Create a confusion matrix and summarize on which category your network did worst in each of the scenarios.
- Did your scaling affect your network's performance?
- What is the effect of momentum factor's value? Is there increase/decrease in term of accuracy? In number of epoch?

Please submit the report and proofs along with the well-committed code.

References

[1] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[2] Cristopher Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.