# Multilayer Perceptron and Backpropagation

alexandru.ionascu96@e-uvt.ro

June 2019

**Abstract**

# 1 Introduction

## 1.1 Resources

**Introduction to multi-layer feed-forward neural networks**, *Daniel Svozil, Vladimir Kvasnicka, Jiri Pospichal*

## 1.2 Motivation

I've prioritized this assessment since I spent a considerable amount of time trying to understand backpropagation. I believe that most of the tutorials available on the internet are not extremely helpful and I believe that most of them are overly-complex.

The situation changed with the help of Reddit's machine learning community to find, in my opinion, the best introduction in neural networks[1] that relies heavily on intuition and assumes no mathematical background.

# 2 Introduction

## 2.1 Supervised Learning

The objective of this report is to describe an algorithm for the following problem: given a collection of $(x, f(x))$ pairs, and an input vector $y$, then predict $f(y)$.

This reverse-engineering problem has two different approaches: continuous and discrete. In the continuous example we call it regression, and for the discrete case we call it classification.

Neural networks can be used for both classification and regression. Since the process is similar, we will focus on classification, the only difference being the output layer's representation.

### 2.2 Neural Network Classification

In this section we will describe shortly the classification process. We start with an n-dimensional input vector $x$.

A multilayered neural network may consist in multiple hidden layers and an output layer. Each neuron models intuitively a decision boundary with a hyper-plane and a certain threshold value, which usually we call it bias.

Each layer of neurons takes an input vector and applies a linear mapping, adding bias after that. This is called activation.

The output layer will be encoded as an n-dimensional vector, with the target values containing only one value as 1, the rest of them 0. If we have binary classification, we may skip this encoding and use a single neuron.

Before calculating the error, we may apply a softmax function in the output layer, thus having probabilities for each class.

## 3 Backpropagation Algorithm

We explained how feed-forwarding applies, but using it required proper weights for each neuron. Backpropagation is a training algorithm. We take each pair of input vector and label, we feed-forward and then calculate the error. With the obtained error, we try to update the weights in order to minimize the error. In order to speed-up the process, we will be using differentiable error function and activations, and apply gradient descent.

For the purpose of this assignment, we will stick to the following error function:

$$E = \frac{1}{2}||target - predicted||^2$$

We shall note hidden layers with $h_i$, where i is the i-th hidden layer, and *out* the output layer. A neural network is simply a function function composition of $h_1 \circ h_2 \circ ... \circ h_n \circ out$.

The convention is to note $w_{ij}$ the j-th weight in the i-th layer. The term backpropagation is self-explainatory. We start with the computed error and update from the last to the first layer's weights.

We will update the weights with the delta rule as follows:

$$\delta_{w_{ij}} = \frac{\partial E}{\partial w_{ij}}\alpha$$

where $\alpha$ is a small positive constant called learning rate. $\frac{\partial E}{\partial w_{ij}}$ can be calculated with the chain rule by applying it to each layer.

### 3.1 Limitations and Further Improvements

The error function is not always convex. Hence, by using gradient descent we might be stuck in a local minimum, not a global one.

The whole training process is done among multiple epochs. After each epoch we expect the error to decrease. We call it early stopping the moment to stop in the current epoch before proceeding to the next one, when the error starts to increase.

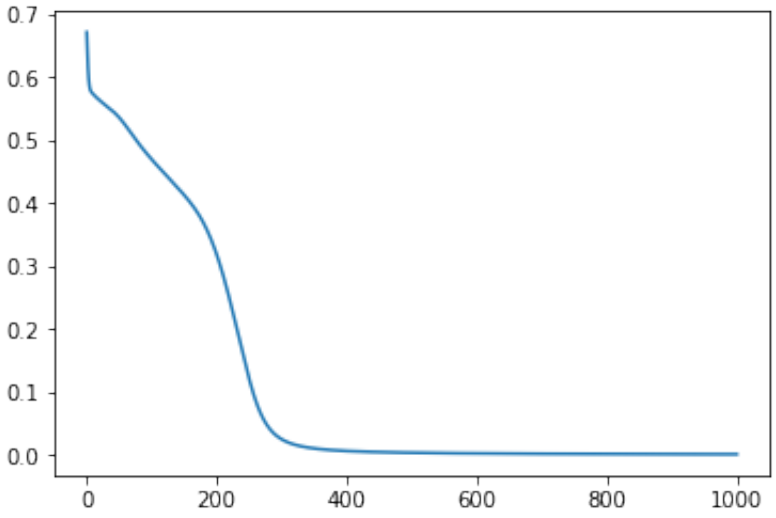
To prevent being stuck in a local minimum, we can reinitialize our exploration by randomly initializing our weights.

There are a couple of techniques which can dramatically improve the backpropagation algorithm:

- Dropout training: random pruning of weights to prevent over-fitting.
- Momentum: introducing a new term to speed up the descent.
- Batching: we may feed-forward multiple input vectors and only after that update the weights.

# 4 Results

The following algorithm has been implemented in Python 3. For this experiment, we approached XOR-gate problem. We used linear and later on tanh activation function, alpha 0.001, 2 neurons in the hidden layer and 2 neurons in the output layer. The training has been done in 1000 epochs, however we can see that early stopping can come after 300 epochs in these conditions.




# 5 Conclusions

In this report we gave a short description of backpropagation algorithm. We have seen it's application in classification, it's limitations, but also possible further improvements in robustness and speed.

# 6 References

1. **An introduction to neural networks**, *Kevin Gurney*, University of Sheffield