# ML Section 1.3: Neural Networks & Backpropogation

Adam Hawley

February 27, 2019

## Contents



# 1 Intro to Neural Networks (NN)

The creation of NNs was inspired by neuroscience. There are multiple effective algorithms to train NNs from examples but on this course we will focus on one called **backpropogation**.

# 2 Preceptron

Perceptrons are the basic unit of NNs. They compute a linear function of $\mathrm{R}^{\mathrm{n}} \rightarrow \mathrm{R}$ using the sum of the weighted inputs.

## 2.1 Thresholding

Maybe we want a discrete output instead of a numerical output. To do this we use thresholding to instead compute a linear function:

$$R^n \implies \{-1, 1\} \tag{1}$$

## 2.2 Training Perceptrons

### 2.2.1 Smallest Neural Network

Changing the perceptron function means changing the weights (apart from $w_0$). Online learning means that we update the weights whenever a new training example (x,t) is presented.

$$w_i \leftarrow w_i + \Delta w_i \tag{2}$$

The basis of weight update is the **error** : Difference between perceptron output $o$ on $x$ and training example target value $t$.

### 2.2.2 A Simple Update Rule

Perceptron training rule:

$$\Delta w_i = \eta(t - o)x_i \tag{3}$$

Where:

- $\eta$ : Learning Rate (0-1)
- $t - o$ : Error
- $x_i$ : The *ith* input.

This training rule is guaranteed to converge within a finite number of steps to a correct weight vector if:

- Training examples are linearly separable.
- A sufficiently small $\eta$ is used.

If the examples are not linearly separable then see below.

### 2.2.3 Gradient Descent

Consider an unthresholded perceptron. The goal is to minimise the squared prediction error on training data (i.e. best-fit approximation).

$$E[\overrightarrow{w}] \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2 \tag{4}$$

Direction of error reduction: gradient of $E$:

$$\nabla E[\overrightarrow{w}] \equiv \left\{ \frac{\delta E}{\delta w_0}, \frac{\delta E}{\delta w_1}, \ldots, \frac{\delta E}{\delta w_n} \right\} \tag{5}$$

So the weight update follows:

$$\Delta \overrightarrow{w} = -\eta \nabla E[\overrightarrow{w}] \tag{6}$$

$$\Delta w_i = -\eta \frac{\delta E}{\delta w_i} \tag{7}$$

Where:

$$\frac{\delta E}{\delta w_i} = \sum_d (t_d - o_d)(-x_i, d) \tag{8}$$

1. Properties of Gradient Descent
   - Gradient Descent can be applied whenever:
     - The hypothesis space is defined by continous parameters (weights).
     - The error term can be differentiated with respect to these hypothesis parameters.
   - Problems:
     - Converging to a local minimum can be slow.
     - There is no guarantee that the procedure will find the global minimum if there are multiple local minima.
2. Online Versino of Gradient Descent Previously the error term was computed using the sum of all the training samples:

   $$\frac{\delta E}{\delta w_i} = \sum_d (t_d - o_d)(-x_i, d) \tag{9}$$

   The online version of gradient descent updates the weight based on a single training example d.

3. Online vs. Offline
    - Updating weights is quicker.
    - But *regular* gradient descent uses the true gradient, so a larger step size can be used.
    - Stochastic gradient descent can avoid falling into a local minimum.
    - This also works for thresholded perceptrons, but does not necesssarily minimise the number of misclassified training examples.

### 2.3 Perceptron Representational Power

A perceptron can only compute a linear function. That is, a perceptron can not compute the XOR function. A network of linear perceptrons also is restricted to linear functions. To overcome this we use a **Sigmoid Perceptron**.

## 3 Sigmoid Perceptrons

Sigmoid perceptrons enable NNs to compute non-linear functions.