2 Backpropagation

As we saw the perceptron is quite limited. It is only good for binary classification problems. With a slight modification it can be use for linear regression. These limitations brought a big gap in the development of artificial neural networks. Before we can explore the next topic we need a short refresh on some calculus concepts.

2.1 Calculus 101

We will need some refreshing on some basic concepts. here are links that you might find useful. If you have alternative resources feel free to use them. However, without this the next steps will be rather difficult. simple derivatives
chain rule
Gradient Gradient2

2.2 Gradient Descent

How do we find good weight values given our dataset? This is an optimization problem that can be solved by gradient descent. Gradient descent is a first-order iterative optimization algorithm for finding a local minimum of a differentiable function. The idea is to take repeated steps in the opposite direction of the gradient of the function at the current point, because this is the direction of steepest descent.
This makes sense as calculating the function for all possible variable combinations is extremely costly when talking about multi-variable problems. Using this approach you start with a random set of weights and start walking down the slope (size of the steps down the slop is determined by the learning rate). Notice that such an algorithm does not promise convergence to the global minimum, it only promises that you will end up in a local minimum, which is often good enough.

2.3 Gradient Descent with the Perceptron

using Gradient descent to find weights of a perceptron we need a few extra steps:
1. we will notice that we can't use previous implementation as the step function is non-differentiable. let us first replace the step-function by a sigmoid function which is easily differentiable. A sigmoid function goes from 0 to 1. we can look at this as a probability (P) to belonging to class A (or 1-P for call B). 2. We need to think about which function do we want to minimize to find a good set of weights? we want a function that will capture the deviation between the the output and the true label. Such functions are called loss-functions. Let us use a very common loss function the mean square error.

$$Error = (out_i - l_i) \tag{4}$$

$$Loss = 1/N \sum_{n=1}^{N} (Error)^2 \tag{5}$$

Notice that this function will be differentialbe in respect to the weights IF the activation function is differentiable to the weights.

2.4 Backpropagation Algorithm

For number of epochs / until loss sufficiently low / loss no longer improves: update weights:

$$W_\nabla = EvaluateGradient(current_loss, X, W) \tag{6}$$

$$W = W - \alpha * Wgradient \tag{7}$$

2.5 EvaluateGradient

Evaluating the gradient is done in two steps:

1. Forward pass: Forward pass is the term used for passing input through the network till the loss function. first we calculate the net (dot product of the weights and input). We then pass the net through the sigmoid function (instead of the step function) (equation 8). Finally, we calculate the error (out-label) and then the loss.

** Calculate the loss function of input X=[2, 3] Y=[1] W=[0.1, 0.3]

2. backward pass: where we calculate the gradient of the loss function in respect to the each one of the weights. (This we do using the chain rule). Finally we update the weights. derivative of the sigmoid is given in equation 9

** Calculate the gradients and update the weights accordingly.

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{8}$$

$$\frac{\delta\sigma}{\delta x} = \sigma(x)(1-\sigma(x)) \tag{9}$$

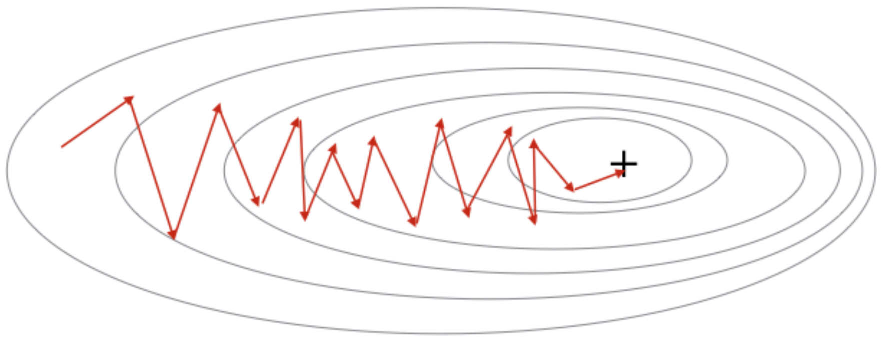
NEXT WE WILL CALCULATE THE ABOVE BUT FOR DATA CONSISTING OF MULTIPLE SAMPLES - PRACTICAL 2

2.6 Stochastic Gradient Descent -SGD

Gradient descent can be slow. In particular in cases where the data-set is big, the computation can become costly, yielding huge matrix calculations.

$$Net = W \cdot X \tag{10}$$

SGD is a way to accelerate the convergence to a good solution. This is achieved by making a few weight updates per epoch. The data-set is divided into smaller groups, called mini-batches. In every epoch a weights update is done after each mini-batch. This results in faster convergence, but with a somewhat less smooth trajectory. An important step is to shuffle the data between epochs, such that the min-batches are different for every epoch, making sure that the model sees different data in every epoch. An extreme case of SGD is an update after every sample (mini-batches of 1).



NEXT WE WILL DO PRACTICAL 3