

# Machine Learning Review

Francesco Saverio Zuppichini

November 3, 2017

The aim of this report is to summarise the topics threaded in my Machine Learning class at USI.

## 1 History of Machine Learning

Write something about the history of ML

## 2 Gradient Descend

The gradient descend is an iterative optimisation algorithm that follows the direction of the negative gradient in order to minimise an objective function. It can be effectively used as Learning Algorithm because it reduces the error function, Equation 3, and adjusts the weights properly. Equation ?? shows the generic update rule.

$$w_{k+1} = w_k - \eta \nabla E(w_k) \quad (1)$$

Where  $\eta$  is the step size, also called **learning rate** in Machine Learning. This parameter influences the behaviour of gradient descent, a small number can lead to local minimum, while a bigger learning rate could "over-shoot" and decreasing the convergence rate. Later in this project you will see how a wrong  $\eta$  can strongly change the output of a Neural Network.

For this reasons, numerous improvements have been proposed to avoid local minima and increase its convergence rate, some of them are: Conjugate Gradient and Momentum.

## 3 Perceptron

### 4 Definition

The **Perceptron** is binary **linear classifier** algorithm used in **supervised learning**. It can be seen as the most basic form of Neural Network. Equation 2 defines its output.

$$f(x) \begin{cases} 1 & \text{if } w \cdot x + b \geq 0 \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

Given a training set  $D = \{(x_1, t_1), \dots, (x_n, t_n)\}$ ,  $x_i \in X$  and  $t_i \in Y$  denotes the input vector and the target vector respectively. We express  $y = f(x)$  as the output of the algorithm,  $w$  the weight and  $b$

the bias. At each iteration the error is calculated using the Mean Square Error, defined in equation 3

$$E(w) = \frac{1}{N} \sum_{i=1}^N (\underbrace{y(x_i, w_i)}_{\text{predicted}} - \underbrace{t_i}_{\text{actual}})^2 \quad (3)$$

The algorithm uses stochastic **Gradient Descent** in order to update the weight at each iteration using the formula defined in Equation 1.

$$\frac{\partial E}{\partial w_k} = y - t \quad (4)$$

## 5 Neural Network

A **Neural Network** is a universal function approximation. It is a nested composite functions like  $f(g((h...)))$ . In its simplest representation, an FeedForward Neural Network, it is composed by a **input layer**, an **hidden layer** and an **output layer**. The size of the hidden layer is usually refers as the **depth** of the network.

### 5.0.1 Forward pass

In order to get the prediction out of our network we need to calculate the compute the activation at each layer  $l$ . Equation 5 shows the activation  $a$  of layer  $l$  for the  $j$ -th neuron on that layer.

$$a_j^l = \sigma(\sum_k w_{jk}^l a_k^{l-1} + b_j^l) \quad (5)$$

Where  $w_{jk}^l$  is the connection from neuron  $k$  in the  $l - 1$  layer to  $j$ ,  $a^{l-1}$  is the activation of the previous layer and  $b_j^l$  is the bias of  $j$ -th neuron in the  $l$  layer. With this in mind, we can rewrite 6 in a efficient vectorised form

$$a^l = \sigma(W^l a^{l-1} + b^l) \quad (6)$$

### 5.0.2 Delta rules

In a Neural Network the weights are iteratively changed in order to decrease the cost function, called  $E$ . We want to find out how much they should be updated, in order to do so we need the output error at each layer. Equation ?? defines  $\delta_j^l$  as the output error of neuron  $j$  in layer  $l$

$$\delta_j^l = \frac{\partial E}{\partial z_j^l} \quad (7)$$

Strictly speaking,  $\delta_j^l$ , is how much the error function changes by changing the weighted input on that layer. Applying the chain rule, Equation 7 becomes:

$$\delta_j^l = \frac{\partial E}{\partial a_j^l} \frac{\partial a_j^l}{\partial z_j^l} \quad (8)$$

By knowing that  $a_j^l = \sigma(z_j^l)$ , Equation 8 can be expressed as

$$\delta_j^l = \frac{\partial E}{\partial a_j^l} \sigma'(z_j^l) \quad (9)$$

Also, delta at layer  $l$  can be expressed by using the next  $l + 1$ -th delta. Equation 10 shows the new rule.

$$\delta^l = (W^{l+1}\delta^{l+1}) * a^l \quad (10)$$

### 5.0.3 Back Propagation

The **Back Propagation** algorithm defines an efficient and interactive method to calculate the gradient at each layer. We want to compute  $\frac{\partial E}{\partial w_{jk}^l}$ . We can applying the delta rule:

$$\frac{\partial E}{\partial w_{jk}^l} = \frac{\partial E}{\partial z_j^l} \frac{\partial z_j^l}{\partial w_{jk}^l} = \frac{\partial E}{\partial a_j^l} \frac{\partial a_j^l}{\partial z_j^l} \quad (11)$$

After some calculation, Equation 12 shows how to calculate the gradient for the weight  $w$  of the  $l$ -th layer for the  $j$ -th neuron.

$$\frac{\partial E}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l \quad (12)$$

While  $\delta^l$  can be calculated used the  $l+1$ -th layer's information, this is why is called *back* propagation since we used the last layer information in order to, each time, find out the next layer's delta until we reach the input layer.

## 6 Convolutional Neural Network

## 7 Recurrent Neural Network

### 7.1 Definition

A Recurrent Neural Networks can remember past decision by taking as input not only the current input but also the last time state. For this reason it is said that a RNN has **memory**. Figure 1 shows a classic representation. Usually, a RNN is represented unfolded to highlight the time dependencies. Due to its ability to remember it mostly used in text and speech recognition.

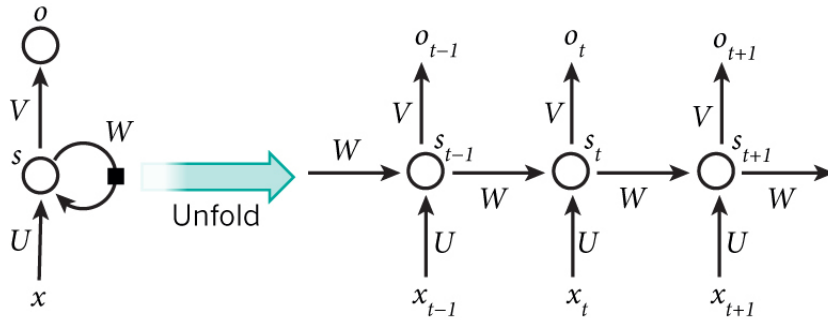


Figure 1: Fold and Unfold representation of a RNN

Very similar to a feedforward network, Equation 13 shows the weighted output at layer  $j$ . The first term on the right-hand is just the feedforward's weighted output, while the second term is the time-dependent term. Matrix  $\omega$  is a hidden-state-to-hidden-state matrix. Basically, we are adding previous informations to our new state at time  $t$ .

$$z[t]^l = (W^l a[t]^{l-1} + b_j^l) + (\omega^l a[t-1]^{l-1}) \quad (13)$$

Equation 14 shows the activation of layer  $j$  at time  $t$ .

$$a[t]^l = \sigma(z[t]^l) \quad (14)$$

While Equation 15 shows the updating rule for the weight  $w^l$

$$\begin{aligned} \frac{\partial E}{\partial w^l[T]l_{jk}} &= \sum_{t=0}^T \frac{\partial E}{\partial z[t]^l_j} \\ \frac{\partial E}{\partial w^l[T]l_{jk}} &= \sum_{t=0}^T a[t]_k^{l-1} \delta[t]_j^l \end{aligned} \quad (15)$$

## 7.2 Vanishing Gradient Problem

Since the network layers and time steps are related to each other through multiplication, the gradient becomes smaller and smaller at each  $t$ . Figure 2 shows the effect of applying *sigmoid* function over time. The data is flatted more and more at each step and therefore the slope will  $\rightarrow 0$ .

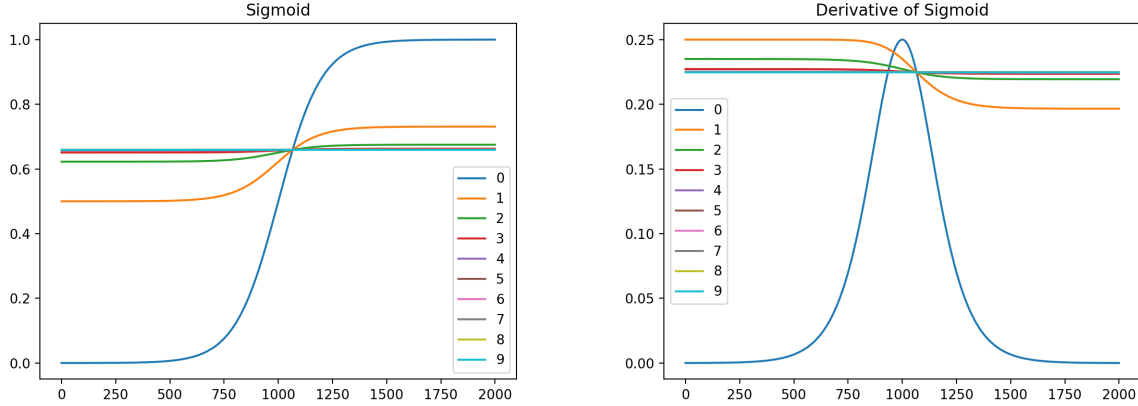


Figure 2: Sigmoid and its derivative over time

Since gradient expresses the change in all weights with respected to the error, without nothing it we cannot adjust properly the network.

## 8 Long Short Term Memory

### 8.1 Definition

The Long Short Term Memory networks, or just **LSTM**, are a special type of RNN capable of learning long-term dependencies. They were introduced by Juergen Schmidhuber in order to solve the vanish gradient problem. They are composed by LSTM cell, Figure 3 shows a unrolled representation.

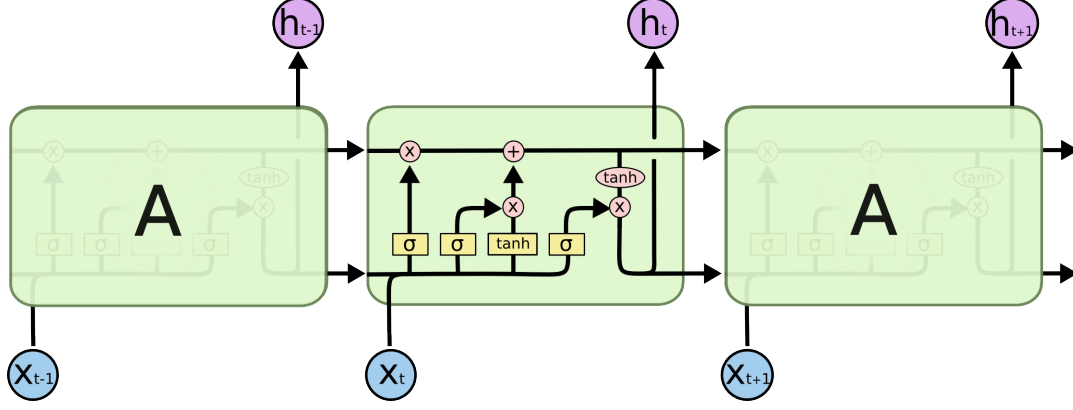
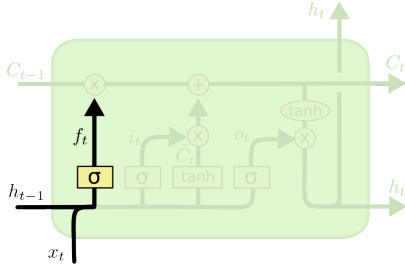


Figure 3: An unrolled LSTM

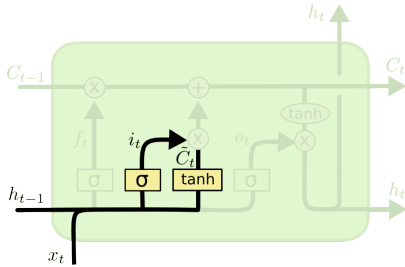
Each cell is composed by 3 gate: **forget** gate ( $f_t$ ), **input** gate  $i_t$  and **output** gate ( $o_t$ ). It takes as input the previous output  $h_{t-1}$  and the old cell state  $C_{t-1}$ , it outputs the next prediction and state,  $h_t$  and  $C_t$ . The cell computes four basic operations:

1. Forget Gate



$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

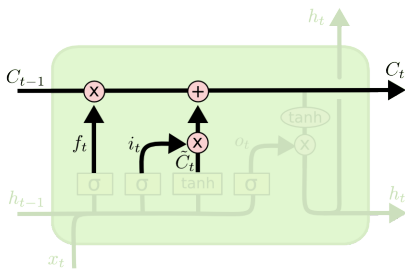
2. Input Gate



$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

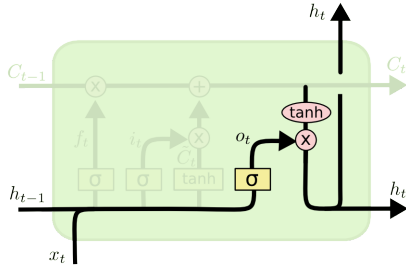
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

3. Update Cell State



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

4. Output Gate



$$o_t = \sigma(W_o [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh(C_t)$$

## 9 Support Vector Machine

## 10 Deep Learning

### 10.1 Supervised Learning

In **supervised** learning there is a fixed training set consists on a set of example patterns and their targets. The objective is to learn a map between the inputs and the targets. 'Supervised' comes from the likeness between a student that tries to answer a question, and a teacher that knows the solution.

Therefore, at each step the algorithm can know how much wrong is prediction was, using a cost function, and it can improve it by applying a learning algorithm such as gradient descent in order to reduce the error.

### 10.2 Reinforcement Learning

In **reinforcement** learning the *agent* takes *actions* in an *environment* in an to maximise a *reward*. Figure 4 shows the actions flow.

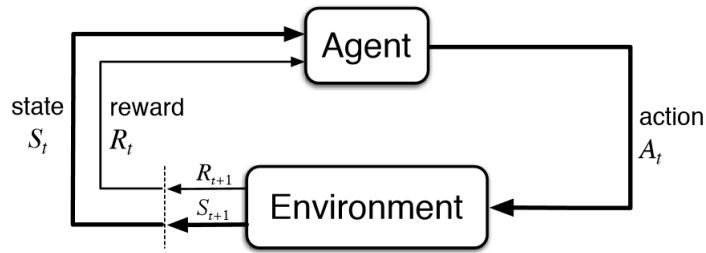


Figure 4: Reinforcement Learning Flow

### **10.3 Unsupervised Learning**

### **10.4 Training Techniques**

#### **10.4.1 Mini-batch**

### **10.5 Regularisation**

#### **10.5.1 L1 Regularisation**

#### **10.5.2 Dropout Regularisation**

### **10.6 Activations Functions**