Neural Networks

Homework 3

Juan Jose Mendez Torrero (s3542416)
Sharif Hamed (s2562677)

May 14, 2018

1 Theory questions

1. When we create a neural network, the challenge itself is to get the right values for the weights between each layer. Since we have to many hidden layer, it is difficult to know how much one node has contributed in the output error. This problem is known as *the credit assignment problem.*

2. After the forward pass has finished, the two most important factors, that determine the error of a neuron in a hidden layer, are the error of the hidden nodes, and the error in the output that these nodes have contributed in.

3. The sigmoid function is defined as:

$$\sigma(\alpha) = \frac{1}{1 + e^{-(\alpha-\theta)/\rho}}$$

   Where we can find:

   (a) α is the activation of the node.
   (b) $e = 2.7183$, mathematical constant.
   (c) ρ determines the shape of the function, is the value is large, then, the curve will be smoother. On the other hand, if it takes small values, the curve will rise more steeply.
   (d) θ is the threshold of the node.

   The sigmoid's derivative is defined as:

$$\sigma'(\alpha) = \frac{1}{\rho} \cdot \sigma(\alpha) \cdot [1 - \sigma(\alpha)]$$

4. Because the nodes reach outputs close to their extreme values. This is known as premature saturation. In this type of cases, the nodes have to unlearn the learning process.

5. In the case that we want to stop the learning process, we have to follow this three criteria:

   (a) Accept all the Boolean error that is next to the correct Boolean error. The problem with this, is that we just only can apply this criteria when we are working with Boolean networks.
   (b) We have to set a very low value for the error mean e_p.
   (c) Finally, if we get a really small value on the change error of the weight, we should stop the learning process. The problem with this criteria is that if the value reach the local minimum instead of the global minimum, we can lead the network to a partial solution.

6. The term, besides the learning rate, that could speed up our network is a momentum term. This term should be added to the learning rule: $\Delta w = \alpha\delta^j(n)x(n) + \lambda\Delta(n-1)$. λ can take values between zero and one, to ensure convergence.

7. We can verify that a network is generalized by inserting new patterns that it never has seen before. If it produces the correct output, hence, the network is generalized.
8. The *overfitting* problem occurs when the network has too many hidden layer and too much freedom to choose the its decision surface. Hence, the network will start to learn noise in the learning data.
9. The network pruning is the process that remove the non-important weights from the network, leaving the most relevant weights.
10. One of the ways to achieve pruning, is the weight decay. This process decrease the weight each training step by an specific uniform amount. In this case, both, large and small values, will reach the same rate, where the important weights will gain more value, meanwhile, the non-important weights will decrease until it gets the value 0 and therefore, removed from the network.
    Another way of pruning is the reduced error pruning, where each node is replaced with the most popular class, and, if the result is correct, the change will be applied, if not, the changes will be discarded.

2 An MLP on paper

1. We can see the MLP in Figure 1.

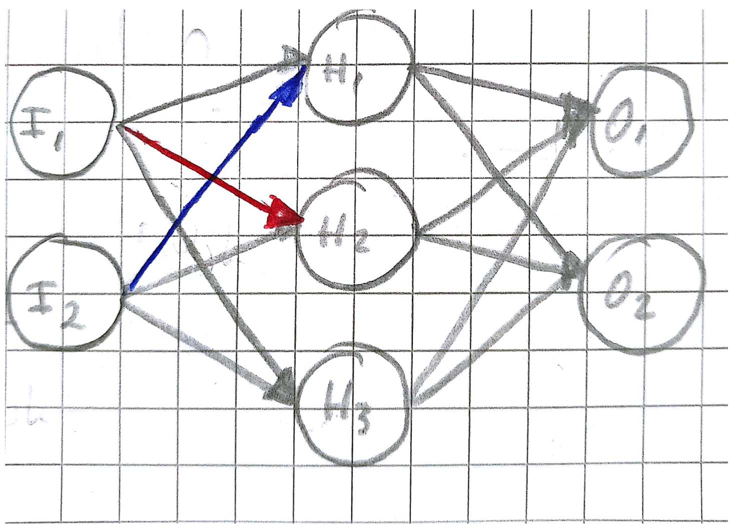

Figure 1: MLP on paper.

2. It is a two layer MLP because we just have to take into account the output and hidden layers.
3. To see which is the dimension of W^h and W^o are, we have to see the following matrices:

$$W^h = \begin{matrix} W_{i_1 j_1} & W_{i_1 j_2} & W_{i_1 j_3} \\ W_{i_2 j_1} & W_{i_2 j_2} & W_{i_2 j_3} \end{matrix}$$

$$W^o = \begin{matrix} W_{h_1 o_1} & W_{h_1 o_2} \\ W_{h_2 o_1} & W_{h_2 o_2} \\ W_{h_2 o_1} & W_{h_2 o_2} \end{matrix}$$

   As we can observe, W^h has a dimension of *2x3*, meanwhile W^o, has a dimension of $3x2$.
4. Each column correspond to the input of the next layer, in the case of, for example, the first column of W^h, the values correspond to the input's weight values that go to the first hidden nodes.

5. W^h_{12} is represented in Figure 1 with a red arrow. On the other hand, W^h_{21} is represented by a blue one.

6. In order to extend the MLP, we have to add a new node to the input layer. The result of this MLP, is represented in Figure 2.

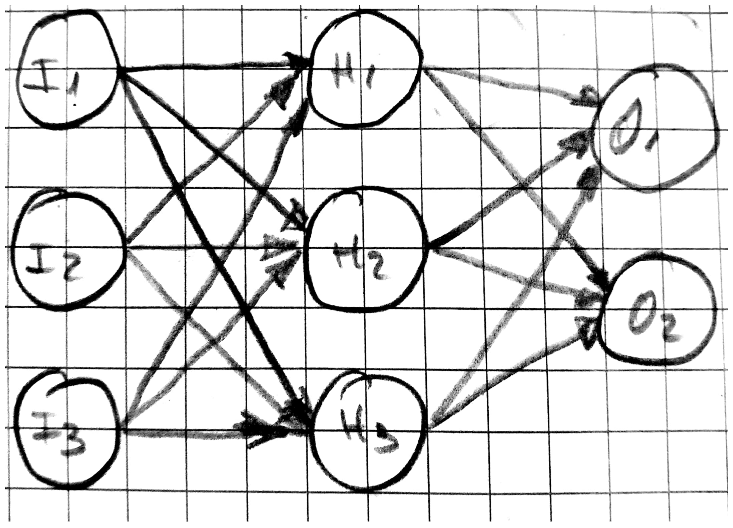

Figure 2: MLP extended.

7. When we create an augmented vector, we have to add the threshold to the original vector, but, this value, has to be negative. Hence, we can add the third row to the table:

Input Neuron	Weight to first hidden layer
1	0.3
2	0.6
3	-0.5

This values can be found on the first column of W^h.

8. If we have the input vector x=[0 1] and a weight vector w=[0.3 0.6], that keeps the value of each connection between the input layer with the next hidden layer. In order to calculate the activation for the fist hidden neuron, we need the formula $a = \Sigma(x \cdot w)$. Knowing this, we can obtain the activation as: $a = 1 \cdot 0.3 + 0 \cdot 0.6 = 0.3$. We can say that the activation for the first hidden layer, has the value of 0.3.

9. We can compute this by that way, because our input vector is a row vector, otherwise, if it would be a column vector, out activation's vector would become an activation's matrix, being that useless because we just only have three hidden nodes inside the hidden layer.

10. We can compute each activation for each node as: $a = (x \cdot w)$, being x a row vector. Hence, to calculate the activation for the entire layer, we have to sum all the activation, we can do that using the following formula, $a = \Sigma(x \cdot w)$. This means that the result will take a form like:

$$a = w_1 \cdot x_1 + ... + w_n \cdot x_n$$

Note: If the input is a column vector, we have to transpose it into a row one.

3 Backpropagation on Paper

1. **Linear activation function**: $f(a) = a$.
We can not apply backpropagation with this activation function. This is due to the fact that the derivative of $f'(a) = 1$, being c a constant, that means that the inputs are not related with the gradient descent of the training, so the changes that the weights have, in this case, is always the same.

2. **Sigmoid activation function**: $\sigma(a) = \frac{1}{1+e^{-(a-\theta)}}$.
   First of all, we are going to apply the forwardpropagation to this network. The two formulas that we are going to use in this step are: $a = \Sigma w \cdot x$ and $\theta(a) = \frac{1}{1+e^{-(a-\theta)}}$. Hence, the activation and output of the two first hidden nodes will be:
   $a_{i11} = -0.5 \cdot 1 + 0.5 \cdot 1 = 0$, $y_{i11} = \frac{1}{1+e^{-0}} = 0.5$
   $a_{i12} = 0 \cdot 1 + 0 \cdot 1 = 0$, $y_{i12} = \frac{1}{1+e^{-0.5}} = 0.731$

   For the next hidden layer, we are going to do the same, but this time, the inputs of this nodes are the outputs of the previous hidden nodes. Hence:
   $a_{j11} = -0.5 \cdot 0.5 + 0.5 \cdot 0.731 = 0.116$, $y_{j11} = \frac{1}{1+e^{-0.116}} = 0.529$
   $a_{j12} = 0 \cdot 0.5 + 0 \cdot 0.731 = 0.731$, $y_{j12} = \frac{1}{1+e^{-0.731}} = 0.675$

   Finally, for the output layer, we are going to do the same:
   $a_{k12} = -0.5 \cdot 0.529 + 0.5 \cdot 0.675 = 0.073$, $y_{k12} = \frac{1}{1+e^{-0.073}} = 0.518$
   $a_{i12} = 0 \cdot 0.529 + 0 \cdot 0.675 = 0.675$, $y_{i12} = \frac{1}{1+e^{-0.675}} = 0.663$

   Now we know the output of the output layer, we can obtain the error on the output layer with: $e = (t - y)$. Hence:
   $e_{k1} = (1 - 0.518) = 0.482$ $e_{k1} = (1 - 0.663) = 0.337$

   Once we have obtain the result of forwardpropagating the network, we can apply the backpropagation to it. Firstly, we are going to obtain the local gradient of the output layer. To do so, we are going to use the formula $\Delta w_{kj} = \alpha \delta_k x_j$, being δ_k the descent gradient of the output layer. Besides, we have to know that $\delta_k = \sigma'(a) \cdot (t - y_k)$ where $\sigma'(a) = \sigma(a) \cdot (1 - \sigma(a))$. Now we know this, we are going to obtain the local gradient for each weight in the output layer.
   $\Delta w_{k11} = 0.1 \cdot \frac{1}{1+e^{-0.073}} \cdot (1 - \frac{1}{1+e^{-0.073}}) \cdot (1 - 0.518) \cdot 0.529 = 0.0064$

   $\Delta w_{k12} = 0.1 \cdot \frac{1}{1+e^{-0.675}} \cdot (1 - \frac{1}{1+e^{-0.675}}) \cdot (1 - 0.663) \cdot 0.675 = 0.005$

   $\Delta w_{k21} = 0.1 \cdot \frac{1}{1+e^{-0.073}} \cdot (1 - \frac{1}{1+e^{-0.073}}) \cdot (1 - 0.518) \cdot 0.675 = 0.0081$

   $\Delta w_{k11} = 0.1 \cdot \frac{1}{1+e^{-0.675}} \cdot (1 - \frac{1}{1+e^{-0.675}}) \cdot (1 - 0.663) \cdot 0.529 = 0.0039$

   We have calculated how much the weights have to change to make it correctly. We can calculate the new value of the weight as $w_{new} = w_{old} - \alpha \cdot \Delta w_{old}$. Hence, we can say that:
   $w_{k11} = -0.5 - (0.1 * 0.0064) = -0.50064$
   $w_{k12} = 0.5 - (0.1 * 0.005) = 0.4995$
   $w_{k21} = 0 - (0.1 * 0.0081) = 0.00081$
   $w_{k22} = 1 - (0.1 * 0.0039) = 0.99961$

   Now, it is time to obtain the value of the local gradient of the hidden layer. This time, the formula is going to change into $\Delta w_{ji} = \alpha \sigma'(a_j) \Sigma \delta_k w_{kj} \cdot x_i$, where δ_k is the local gradient of the *L+1* layer.

3. **Threshold activation function**:$f(a) = (a > \theta) \cdot 1$.
   As in the first case, this time we can not apply backpropagation with this activation function. This is due to the fact that the derivative of the activation function is constant, 1 in this case. Hence, the changes that are made on the weights, are going to be always the same, leading this to a wrong conclusion.

4 Essay Question

1. The main difference between batch learning and online learning is, how they train their data each epoch.

On one hand, the batch learning data does not update the weight of each connection each iteration, but it does each epoch. Meanwhile, the error is computed each iteration.
On the other hand, online learning updates the weight each iteration. Likewise, the error is computed each iteration.
Furthermore, on-line training is very handy when the data set is very large and some data should be left out to make it computationally less expensive, this is possible because the on-line method does not need full data set and does not need to finish all of them. Furthermore, on-line learning is also used over batch learning when the data is generated over time and the network needs to keep adapting. A example is the stock market. Batch training uses a better approximation of the true gradient for its weight updates and so should get better results.

2. The vanishing gradient problem occurs when we are 'squashing' the inputs, that means, a large change in the inputs, does not mean a large change in the outputs, but make the change smaller. This can be solve if we use another activation function, in example Rectified Linear Unit instead of the Sigmoid function. The Rectified Linear Unit does not 'squash' their inputs, by setting the activation function as $max(0, x)$, being x the inputs.

3. The process of pruning consist on the reduction of the decision tree size. There are so many techniques to do the process of pruning, one of them is Reduced error pruning, that consist on changing each node into it's most popular class. If this change does not affect, then, we keep the change. This reduces overfitting because overfitting can occur better when there are many parameters(weights) because it can then adjust more precisely to the training set. Reducing the freedom of a network by pruning makes the network get more generalized results, which is what we want if a network is overfitting.

4. Since overfitting occurs more when there are many parameters and to much for the model that the network should train, a method of determining the right number of nodes(which is at the same time determining the right number of parameters) will reduce overfitting.