Machine Learning in Healthcare – HW4

Yonathan Belicha | I.D: 203538574 & Daniel Cherniavsky | I.D: 205847817

Part 1:

Task 2: Basic neural network is based on weighted inputs, bias, and the application of activation function on the sum of them. The system is consisting of several layers that are composed of those components, while the information moves from the input throughout the layers. The process is repeated until we reach the last layer.

There are 2 major types of activation functions: linear and non-linear activation functions. Non-linear activation function are mainly used and allow us to deal with more complex data, compute and learn almost any function representing a question, and provide us accurate predictions. In addition, the non-linear activation functions allow backpropagation, because it include derivative function which is related to the inputs.

Non-linear activation function:

- Sigmoid: this is a very simple activation function, which takes every value as input and gives probability of the outcome. It looks like 'S' shape. The function is:

$$f(x) = \frac{1}{1 - e^{-x}}$$

  This function is continuously differentiable, monotonic and has a fixed output range. The biggest advantage of this method is its simplicity. On the other hand, the main throwback of this function is the problem of 'Vanishing Gradients'. It means that it squishes large input information into a small input space between [0,1]. Therefore, quite large change in the input will cause only small changes in the output. In the sigmoid function, as the input become larger or smaller, the derivative values in this ranges are very small and converge to zero. It means that when dealing with quite many layers it can cause the gradient to be too small for training and work effectively. Because of using back propagation and chain-role, small gradients means that the weights and the biases of the initial layers will not be updated effectively. The initial layers are often crucial for the classifying assignment and thus, this problem may lead to inaccuracy of the whole network. Moreover, small gradients can lead to very small learning rate which can cost in high computational time.
- Tanh: this function looks like sigmoid function but better. The range of it is [-1,1] and also 'S' shaped. The function is:

$$f(x) = tanh\,(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

  The advantage of this function is that its derivative is more steep which means it can get more values, this means that it will be more efficient because it has a wider range for faster learning. But in this case, we also have the problem of gradients that are 'vanishing' at the edges of the function.
- Relu (Rectified Linear Unit): the function is:

$$f(x) = \begin{cases} x & x > 0 \\ 0 & x < 0 \end{cases}$$

  This function is non-linear. Although its positive part may seem linear at first glance, the negative part having a value of zero. That fact saves us computational time and thus creates much more intensive activation. Some of the neurons in this process can 'die' (getting a zero value), and thus the computational load is distributed more efficiently on the network. However, the advantage of this activation function is also the disadvantage of it, meaning that those dead neurons can be significant in some cases and thus the negative value may be treated inappropriately.
- The 'Leaky' Relu: the function is:

$$f(x) = \begin{cases} x & x > 0 \\ ax & x < 0 \end{cases}$$

this function is an improvement of the Relu activation function. The improvement here is that the negative values get non-zero values and the gradient has small positive value. This property allows backpropagation also for negative values.

In our case, we chose 'tanh' as the activation function. As we can see the results got a bit worst. Meaning that the loss got higher and the accuracy got lower.

Task 3: The number of epochs is a hyperparameter that defines the number times that the learning algorithm will work through the entire training dataset. As we can see the performance, in contrary to the expected, is quite equal and even a bit worse in comparison between 25 and 40 epochs. We can conclude that our data is not good enough or that our model is not suitable for this kind of data and that is why increasing the epochs does not improve the results. Another option is that our model has reached is best results and cannot be improved.

Task 4: In overall the difference between the techniques is based on the trade-off between computational time and accuracy of parameter update. The three main techniques are:

- **Batch-gradient descent:** in this method the gradient of the cost function is calculated for the entire training set. This method is quite slow for large dataset, but the objective function doesn't fluctuate heavily and therefore the accuracy is relatively high.
- **Stochastic gradient descent:** in this method the gradient of the cost function is calculated for each training example ("one by one"). This method is based on frequent updates with high variance and thus, the objective function fluctuate heavily, therefore less accurate. On the other hand, performing an update is relatively quick.
- **Mini-batch gradient descent:** this method is the compromise between the two previous methods. Therefore, we can observe here light fluctuations in the objective function. In this method the gradient of the cost function calculated for a fixed number of training examples. This method earns the benefits of both methods and therefore, presents relatively fast and accurate performance.

Task 5: Batch normalization is a technique that normalizing the mean and the variance of each layer during training. It based on changing the distribution of the input values to a given learning algorithm. This method causes a smoothness that induces more predictive and stable behavior of the gradients. Therefore, this normalization helps us to increase the learning rate and to improve the learning speed (computational time). In our model, the batch normalization has not caused better results (even a bit worse). It may happen when the data is not good enough.

Part 2:

Task 1:

- How many layers does it have? ## 8 layers
- How many filters in each layer? ## 64,128,128,256,256
- Would the number of parameters be similar to a fully connected NN? No, probably its smaller because the network is not fully connected.
- Is this specific NN performing regularization? Yes, it performs regularization by L2 norm.

Task 2:

Results before reduction:

test loss, test acc: [7.901212215423584, 0.36000001430511475]

Results after reduction:

test loss, test acc: [4.923205375671387, 0.3199999928474426]

As can be seen the accuracy on both models were poor, and the first model was a bit better. The major thing that can be seen is that the loss is much smaller after filters reduction. We

deduce that it′s because the first model is overfitted (″memorizing″ the data) and didn′t learn the data and therefore, the loss is higher. The simplest way to deal with overfitting is by decreasing the complexity of the model. For example, we can remove layers or reduce the number of filters to make the network smaller, as we did here.