**Machine Learning in Healthcare**

**HW 4 – Neural Network Models**

**Ofek Aloush  
307841742**

**Asaf Tzur**

**312164932**

**Part I – fully connected layers**

**Task 2: Activation functions**

ReLU stands for rectified linear unit. It is an activation function and defined as the positive part of its argument:

However, LeakyRelu is defined as follows:

The main disadvantage of the ReLU activation function is of course the negative part. Negative input will result a gradient equal to zero and no learning will take place. This problem is referred to as the dead neurons problem – due to a random initialization, many neurons may “die”. Using LeakyRelu solves this problem since negative input will not get zero value and some learning will take place. Furthermore, those functions possess a tradeoff between computation time and overfitting. The model’s computation time may be shorter with ReLU, yet we risk overfitting (if too many neurons are dead because of lousy initialization). Another difference is that when LeakyRelu is applied, one does not have to worry about the random initialization, since the negative part is not zero.

**Task 3: Number of epochs**

We can see that there is no difference between 25 epochs and 40 epochs. It means that our model converged to a local minima fast enough (after 25 epochs at max) and therefore additional epochs will not improve the accuracy.

**Task 4: Mini-batches**

When we use gradient descent, we update our weights according to the gradient. The difference between stochastic gradient descent (SGD), mini-batches and batch gradient descent (BGD) is the update’s timing. When we use SGD, an update takes place after each example. This method reduces running time, yet it is less accurate and may converge to a local minima since we rely only on one sample each iteration. The most accurate way is BGD since the update takes into account all the examples at once. It usually takes too much computation time. The mini-batch method is somehow in the middle, meaning that the update takes place after a predetermined number of examples. In that way, we can obtain the perfect tradeoff between running time and reaching global minima. It is better than SGD in terms of accuracy and noise reduction (if there is one abnormal sample, its effect will abate when averaging over all the mini-batch).

**Part II – Convolutional Neural Network**

**Task 1 – 2D CNN**

Layers and filters:

|  |  |  |
| --- | --- | --- |
| Layer no. | Layer | Number of filters |
| 1 | Permute | 0 |
| 2 | Conv 2D | 64 |
| 3 | Dropout (optional) | 0 |
| 4 | Batch Normalization | 0 |
| 5 | Max Pooling 2D | 0 |
| 6 | Conv 2D | 128 |
| 7 | Dropout (optional) | 0 |
| 8 | Batch Normalization | 0 |
| 9 | Conv 2D | 128 |
| 10 | Dropout (optional) | 0 |
| 11 | Batch Normalization | 0 |
| 12 | Conv 2D | 256 |
| 13 | Dropout (optional) | 0 |
| 14 | Batch Normalization | 0 |
| 15 | Conv 2D | 256 |
| 16 | Dropout (optional) | 0 |
| 17 | Batch Normalization | 0 |
| 18 | Max Pooling 2D | 0 |
| 19 | Flatten | 0 |
| 20 | Dense | 0 |
| 21 | Dropout | 0 |
| 22 | Dense | 0 |
| 23 | Dense | 0 |

There is regularization in this network. Every Conv2D layer includes l2 regularization and in addition, if the dropout takes place, it is also a type of regularization.

The parameters’ number will be smaller than fully connected neural network because Conv2D CNN uses filters that reduce parameters’ number.

For example, consider the following:

An input of 1000\*1000 image with RGB channels to a NN with one hidden layer of 1000 neurons, will produce 3 billion parameters. On the contrary, in Conv2D CNN with 10 filters sized 3\*3, we shall have 3\*3\*3\*10 = 270 parameters. (example from the lecture)

**Task 2 – Number of filters**

Reducing the number of filters increased performance. Instead of 29.7% accuracy, we got 37.1%.