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:

$$f(x) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases} = \max(0, x)$$

However, LeakyRelu is defined as follows:

$$f(x) = \begin{cases} x, & x \geq 0 \\ 0.01x, & x < 0 \end{cases}$$

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	64
	Conv 2D	
	Dropout (optional)	
	Batch Normalization	
2	Max Pooling 2D	128
	Conv 2D	
	Dropout (optional)	
	Batch Normalization	
3	Conv 2D	128
	Dropout (optional)	
	Batch Normalization	
4	Conv 2D	256
	Dropout (optional)	
	Batch Normalization	
5	Conv 2D	256
	Dropout (optional)	
	Batch Normalization	
6	Max Pooling 2D	0
	Flatten	
	Dense	
	Dropout	
7	Dense	0
8	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.

Task 2 – Number of filters

The model was run a few times and the results were inconsistent regarding the accuracy.

This can happen due to two main reasons: the data and the neural network. If the data is not good enough, the filters' number will have little influence over the accuracy. In addition, it is possible that this specific neural network is not suitable for our classification.

The results presented in the Jupyter notebook are the ones of the last run. There we can see that reducing the number of filters improved accuracy from 29.7% to 37.1%.