Machine Learning in Healthcare

HW 4

Asaf Licht, 311129522

Sapir Gershov, 204471577

Part 1: Fully connected layers

Task 2: Activation functions

Change the activation functions to LeakyRelu or tanh or sigmoid. Name the new model new_a_model. Explain how it can affect the model.

Rectified Linear Unit (ReLU) convert linear outputs of a neuron into nonlinear outputs by outputting x for all x >= 0 and 0 for all x < 0. Basically, ReLU function is max(x, 0). This simplicity makes it more attractive than the Sigmoid activation function and the Tangens hyperbolicus (Tanh) activation function, which use more difficult formulas and are computationally more expensive. In addition, ReLU is not sensitive to vanishing gradients, whereas the other two are, and also known to generalize well. Yet, ReLU have certain challenges:

- It tends to produce very large values given its non-boundedness on the upside of the domain. Theoretically, infinite inputs produce infinite outputs.
- If a neuron's weights are moved towards the zero output, it may be the case that they eventually will no longer be capable of recovering from this. They will then continually output zeros. This is especially the case when our network is poorly initialized, or when your data is poorly normalized.
- Small values, even the non-positive ones, may be of value; they can help capture patterns underlying the dataset. With ReLU, this cannot be done, since all outputs smaller than zero are zero.

We might prefer the Sigmoid function, especially when we estimate probabilities. This is because the Sigmoid outputs are between (0,1) and the probabilities have a very similar range of [0,1]. In binary classification problems, when we estimate the probability that the output is of some class, Sigmoid functions allow us to give a very weighted estimate.

Task 4: Mini-batches

Build the model_relu again and run it with a batch size of 32 instead of 64. What are the advantages of the mini-batch vs. SGD?

(mini-) Batch Gradient Descent involves calculations over the full training set at each step as a result of which it is very slow on very large training data. Thus, it becomes very computationally expensive to do Batch GD. However, this is great for convex or relatively smooth error manifolds. Also, Batch GD scales well with the number of features.

SGD tries to solve the main problem in Batch Gradient descent which is the usage of whole training data to calculate gradients as each step. SGD picks up a "random" instance of training data at each step and then computes the gradient, making it much faster as there is much fewer data to manipulate at a single time, unlike Batch GD. There is a downside of the Stochastic nature of SGD - once it reaches close to the minimum value then it doesn't settle down. Instead, it bounces around which gives us a good value for model parameters but not optimal.

Part 2: Convolutional Neural Network (CNN)

Task 1: 2D CNN

Have a look at the model below and answer the following:

- *How many layers does it have?*
- *How many filter in each layer?*
- *Would the number of parmaters be similar to a fully connected NN?*
- *Is this specific NN performing regularization?*

1. How many layers does it have?

   The following network has 8 layers

2. How many filters in each layer?

   Conv2D_1 has 64 filters, Conv2D_2 and Conv2D_3 have 128 filters, Conv2D_4 and Conv2D_5 have 256 filters

3. Would the number of parmaters be similar to a fully connected NN?

   Num of parmaters in a Conv layer = ((shape of width of the filter * shape of height of the filter * number of filters in the previous layer+1)*number of filters)

   Num of parmaters in a Fully connected = ((current layer neurons * previous layer neurons)+1*c)

   As we can cealry see, the number of parmaters of a fully connected NN would increas dramatically.

4. Is this specific NN performing regularization?

   This specific NN performs L2 regularization with value of 1e-2