Assignment 1 - Report

Question 4:

We have run our network with the following parameters:

Number Layers	4
Sizes of each layer	20,7,5,10 (left -> right)
Batch normalization	False
Dropout	False
Batch sizes	128
Learning rate	0.009
Epsilon for early stopping	0.0001
Number of training steps	100

The cost history from the L Layer model is depicted in the Figure in the conclusion (we didn't want to overload the document)

The results of the model from the current structure is depicted in the table in the next section with the batch normalization comparison.

Question 5:

We have run our network with the following parameters:

Number Layers	4
Sizes of each layer	20,7,5,10 (left -> right)
Batch normalization	True
Dropout	False
Batch sizes	128
Learning rate	0.009
Epsilon for early stopping	0.0001
Number of training steps	100

The cost history from the L Layer model is depicted in the Figure in the conclusion (we didn't want to overload the document)

The following table shows the accuracy over train, validation, and test with and without batch normalization. The Table also shows the running time and the number of iterations it took to converge.

Batch Size	Batchnorm	Train	Validation	Test	Running time	Iterations
32	FALSE	81.50%	85.06%	84.70%	30.46s	5700
	TRUE	86.02%	86.32%	86.05%	29.44s	5800
64	FALSE	79.90%	85.13%	85.33%	37.69s	5800
	TRUE	86.32%	86.32%	85.93%	39.86s	5200
128	FALSE	84.96%	88%	84.72%	64.17s	6000
	TRUE	91.46%	91.07%	90.08%	44.96s	3200
256	FALSE	78.64%	82.84%	82.45%	87.66s	4000
	TRUE	88.36%	88.63%	88.08%	115.92s	5700
512	FALSE	80%	83.55%	82.95%	169.2s	4400
	TRUE	82.36%	83.17%	82.42%	115.07s	3000

Question 6 (Bonus) :

In order to add the dropout functionality we needed to add the functionality that supported the dropout.
In the L_model_forward method, we have included a condition to whether to activate the dropout or not.
If yes, the following code has been activated to all layers but the last one.

```
if dropout is not None:
    d_i = np.random.rand(A_prev.shape[0], A_prev.shape[1]) < dropout
    A_prev = np.multiply(A_prev , d_i) / dropout
```

The first line creates a random array with the shape of the layers filled with numbers in the range [0,1]. In case the number is lower than the dropout probability declared in the main (0.8) The slot will be 0 else 1.
The second line takes the previously computed array and multiplies it with the old layer values such that all the slots of the array that are 0 will be 0 as well (meaning these neurons have been dropped). We also divide the neurons that have stayed after the dropout in order to increase them to avoid vanishing gradients (in case we have randomly chosen neurons with small values).

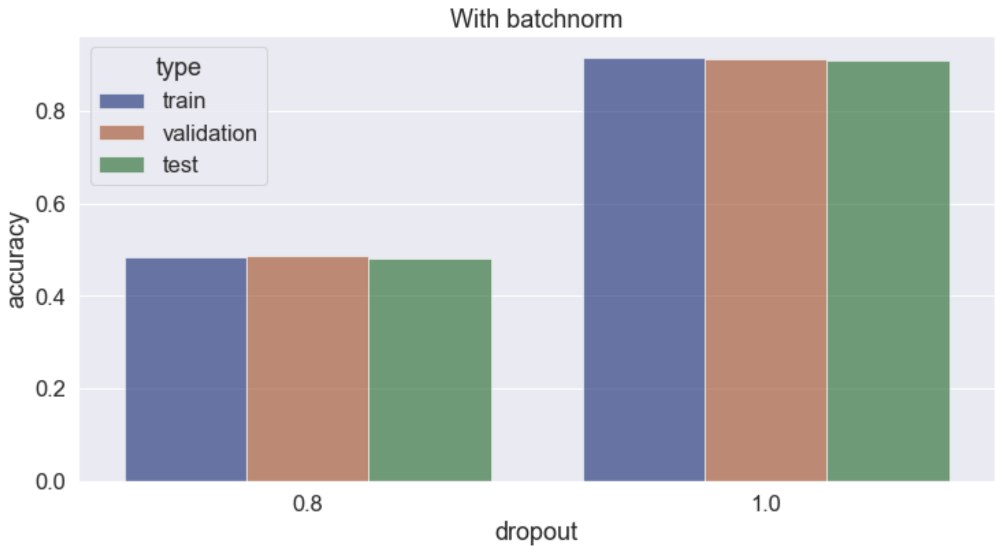
We have run our network with the following parameters:

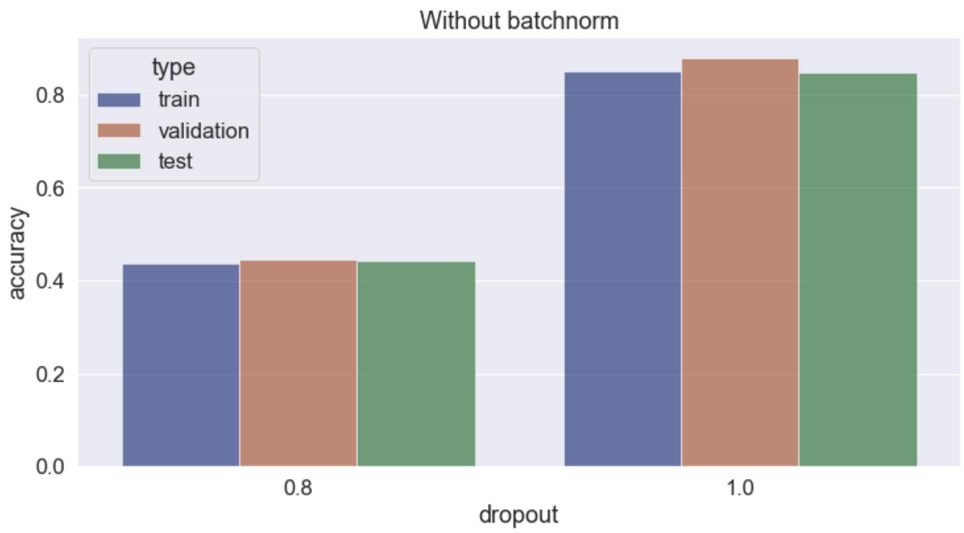
Number Layers	4
Sizes of each layer	20,7,5,10 (left -> right)
Batch normalization	True
Dropout	True
Batch sizes	128
Learning rate	0.009
Epsilon for early stopping	0.0001
Number of training steps	100

The cost history from the L Layer model is depicted in the Figure in the conclusion (we didn't want to overload the document)

Our best model achieved the following scores:

Parameters	With Dropout	Without Dropout
Number of iterations till converges	19400	6000
Running time	216.66s	64.17s
Train accuracy	43.77%	84.96%
Validation accuracy	44.59%	88%
Test accuracy	44.3%	84.72%





Auxilliary functions:

- linear_activation_backward:
  linear_activation_backward is the function that computes the activation backward propagation for a single layer (layer l). The function calls the correct derivate with respect to the layer's activation and then calls linear_backward to future backpropagation.
- print_overall_accuracy:
  The function receives the parameters of the trained neural network and data divided into X and Y (X is the raw data and Y are the labels). And compute Accuracy metrics with respect to the inputted data. The function prints different prints according to the input (Train, Validation, or Test).
- convert_to_onehot_vector:
  The function converts an input vector in the range [0, X] to a one-hot matrix of vectors where each vector is a binary vector with size X. For example, the input [1,10] will be converted to [[0,1,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,1]]
- load_data:
  The function Loads the MNIST Dataset from the TensorFlow framework. The function also reshapes the MNIST figures to a single vector sized 768 and normalizes the input to avoid exploding gradients.

Conclusions and Notes:

- We have included an early stopping criterion as follows: (previous_accuracy - current_accuracy) < 0.0001
- Regarding the accuracy, we can clearly see that the batch norm indeed increased the model performance without taking much time (the running time is nearly equal) therefore we will choose to include the batch norm.
- We have decided to add a parameter initialization strategy in order to improve the convergences time, we have decided to follow the lectures and add a ReLu parameter initialization which includes multiplying the initial random weights with sqrt(2/n), where n is the number of inputs to the layer.
- Loss comparison:

By comparing the loss over the training step we can notice that the dropout functionality causes the model to converge slower and worse in comparison to without it. And by comparing batch normalization and without, we can notice that with batch normalization the model converges faster but with a bigger loss, although we can conclude from the early table in this document, that batch normalization actually achieves better results. Therefore we can conclude that batch normalization takes a longer time to process but converges with a smaller number of iterations and achieves a higher score. We will then conclude that a network with a big amount of iterations will have to use batch normalization since it will result in faster processing.