Machine Learning	**Homework 3**
CSCI 5622 Fall 2017	**Due Time Oct 13, 2017**
Name: Payoj Jain	**CU identitykey: paja3104**

1 Back Propagation (35pts)

1.1 Programming questions

1.2 Analysis

1.2.1 What is the structure of your neural network (for both tinyTOY and tinyMNIST dataset)? Show the dimensions of the input layer, hidden layer and output layer.

Solution.
Structure for tinyTOY is given as below:

- Number of input features (input dimensions): 2
- Number of Hidden Layers: 1
- Number of Hidded layer units (hidden layer dimension): 30
- Number of classes (output dimensions): 2

Structure for tinyMNIST is given as below:

- Number of input features (input dimensions): 196
- Number of Hidden Layers: 1
- Number of Hidded layer units (hidden layer dimension): 50
- Number of classes (output dimensions): 10

1.2.2 What the role of the size of the hidden layer on train and test accuracy (plot accuracy vs. size of hidden layer using tinyMNIST dataset)?

Solution. Both train and test Accuracies first increase with increase in hidden layer dimension and then decrease. **Best test accuracy (89.6%)** is achieved when size of the hidden layer was 50 while **Best train accuracy (98.4%)** is achieved at size = 100.

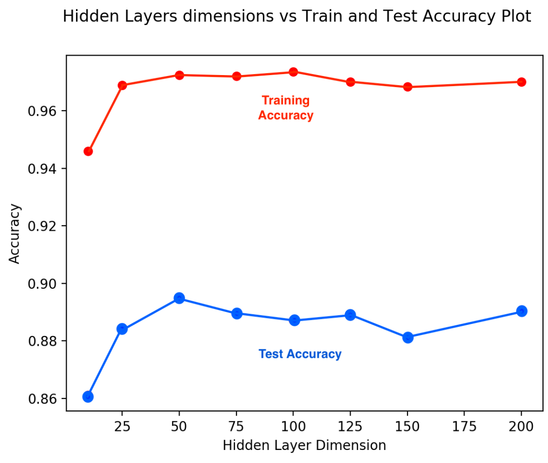


Figure 1: Graph shows that **accuracy first increases and then decreases** with increase in size of hidden layer.

1.2.3 How does the number of epochs affect train and test accuracy (plot accuracy vs. epochs using tinyMINST dataset)?

Solution. Both train and test accuracies increase significantly fast from epoch = 1 till 20. After 20 epochs both train and test accuracy increase slowly to achieve 98.4% and 89.6% respectively.
Note: Here hidden layer size is taken to be 50.

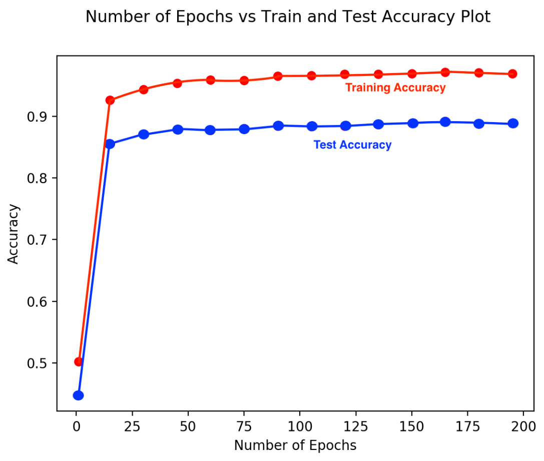


Figure 2: Graph clearly shows **how fast accuracies increase till epoch 20 and then become almost constant** with increase in number of epochs.

2 Keras CNN (35pts)

2.1 Programming questions

2.2 Analysis

2.2.1 Point out at least three layer types you used in your model. Explain what are they used for.

Solution. I have added following layers in building CNN model:

- **ZeroPadding2D**: This is used to avoid shrinking of layers which doing convolutions. This ensures that dimensions of a particular layer doesn't shrink.
- **Conv2D layers**: This layer is generally used for spatial convolution over images. It creates filter which does convolution with the input layer and produces tensor of outputs. It produces one activation maps corresponding to each kernel i.e. if there are 32 kernels in a Conv2D layer then the output depth will be 32. Lower level of Conv2D tend to learn edges and colors of image while high level learn complex object parts. **Most important part of convolution layer is it resolves the vanishing gradients problem in training multi-layer neural networks with many layers by using backpropagation.**
- **MaxPool2D**: It is used for down-sampling and is the most common used non-linear down-sampling function. It uses the maximum value from a sub-region of the output from previous layer. **It is basically used to reduce number of parameters and computations in the network.**
- **Dropout**: It is used to avoid overfitting of the model over training data. It is a type of regularisation where at each training stage, nodes are either dropped out (meaning not considered for training) with some probability or kept for training.**Network is trained without those nodes. After the network is trained, the dropped out nodes are again added with no discounting in weights.**
- **Flatten**: Used to reshape or convert n dimension tensors into a 2 dimension tensor. Flattens data samples of each batch.
- **Dense layers**: It is used for generating regular fully connected Neural Network.

2.2.2 How did you improve your model for higher accuracy?

Solution. As a baseline I just added one Conv2D layer in the network with 1 MaxPool2D and Dense. The model achieved an accuracy of 98.38%. Here hyperparameters like batch size, kernel size, number of epochs were taken as default. I then **added 1 more Conv2D** layer and to avoid overfitting I added **Dropout** with dropping probability of 0.25. Also, I tried the batch size of 64 and 256 but the accuracy went down. So, I kept **batch size = 128**. I tested by **changing the kernel size** in Conv2D layers to check if kernel size affects the accuracy. Also, trying **different activation functions** had significant effect on the test accuracy. **Best test accuracy was achieved for ReLU**. I added 2 more Dense layers. All the above mentioned changes increased the test accuracy to 98.87%. I then increased the Dropout argument to 0.5 and the test accuracy shot up to 99.1%.
To achieve optimal solution faster, I have used **Adam** as optimiser function. It is more efficient than SGD by achieving optimal solution faster.

2.2.3 Try different activation functions and batch sizes. Show the corresponding accuracy.

Solution. Besides ReLU, I tried CNN with ELU, tanh, softmax and sigmoid functions with 5 different batch sizes. The result showed that for a particular batch size ReLU achieved the highest accuracy as compared to other non-linear functions. The second best performing function was ELU while Sigmoid performed the worst.
NOTE: Graph below doesn't show accuracies for Sigmoid and Softmax functions since their accuracies were much lesser than accuracies using ReLU, ELU and tanh, which are not significant enough to be shown on the graph.

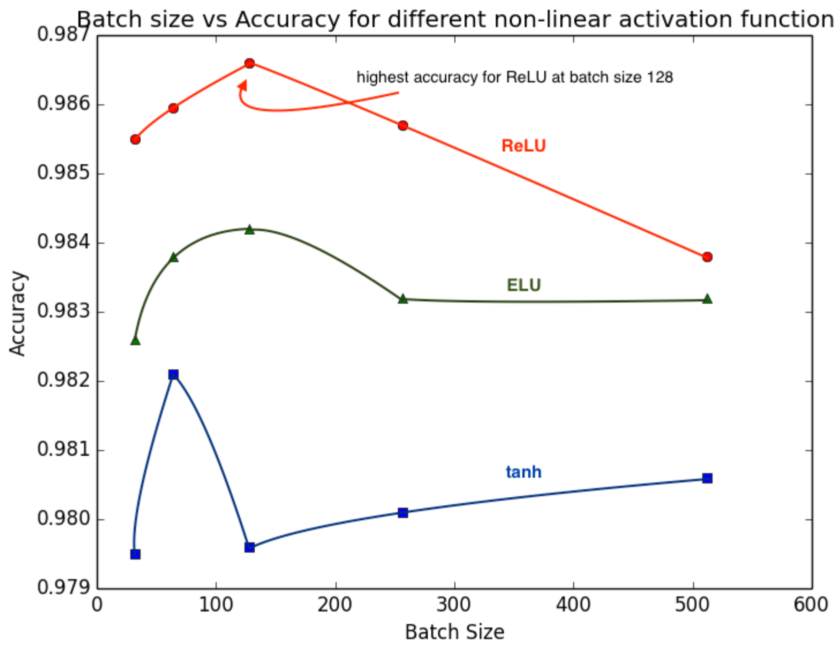


Figure 3: Graph shows that test accuracy for different non-linear activation functions at different batch sizes. **ReLU happened to perform the best in achieving the highest test accuracy.**

3 Keras RNN (30pts)

3.1 Programming questions

3.2 Analysis

3.2.1 What is the purpose of the embedding layer? (Hint: think about the input and the output).

Solution. Word embedding means to basically map words or sequence of words of a given vocabulary to vectors of numbers in a low dimensional space (lower than the vocabulary size). **The embedding layer transforms words (here indices which represent words in IMDB review) to a vector of size 128 (here).** 128 here is the feature dimension which means the number of dimensions each embedding vector has.

3.2.2 What is the effect of the hidden dimension size in LSTM?

Solution. Accuracy first increases and then decreased with increase in number of hidden layer units. Also, With increase in the number of hidden dimension size the model takes more time to train.

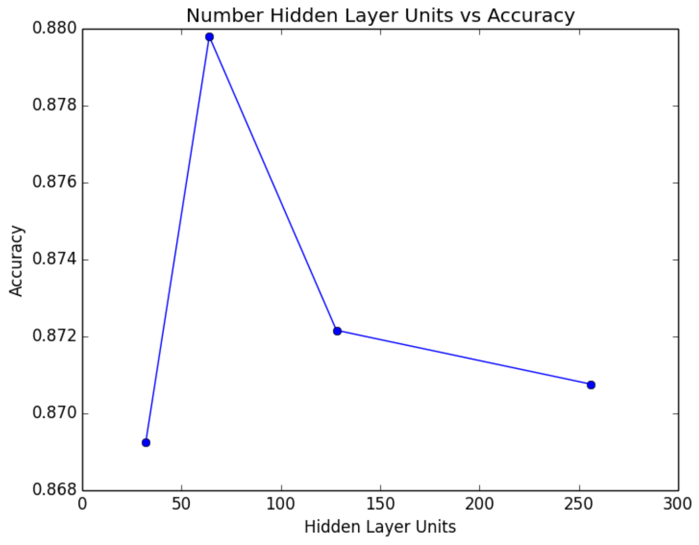
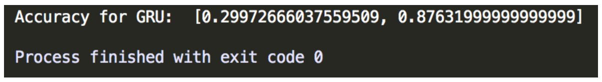


Figure 4: Graph shows that test accuracy for different Hidden layer units in LSTM.

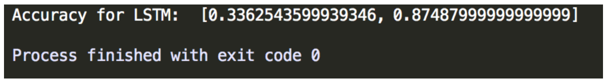
3.2.3 Replace LSTM with GRU and compare their performance.

Solution. I ran the code for 5 epochs and 128 hidden layer units. For this data GRU seemed to perform a little better (87.631%). The reason could be the size of data is small. LSTM needs good amount of data to train better.
Also, I observed that GRU trained faster than LSTM. Reason for this could be that GRU has only 2 gates and no memory unit. It has less complex structure therefore computationally more efficient than LSTM. Below are the accuracies I got from each.



```
Accuracy for GRU:  [0.29972666037559509, 0.87631999999999999]

Process finished with exit code 0
```



```
Accuracy for LSTM:  [0.3362543599939346, 0.87487999999999999]

Process finished with exit code 0
```

3.2.4 EXTRA CREDITS (5pts) Try to use pretrained word embeddings to initialize the embedding layer and see how that changes the performance.

Solution. For this exercise I have used Glove Embedding. After 5 epochs Accuracy for LSTM using pretrained word embeddings increased slightly to 87.07% as compared to 87.9% using without pretrained word embeddings. This is when number of hidden layer units were 64.