Problem 1 (statement)

Find an example of a (deep) neural network (either fully connected or convolutional) using MNIST in the deep learning framework of your choice. If you use Keras, you can find a number of examples at https://github.com/keras-team/keras/tree/master/examples including *mnist_cnn.py* (convolutional neural network for MNIST) and *mnist_mlp.py* (multi-layer perceptron, which is fully connected). After you train the network in an environment you set up, answer the following questions.

(a) Briefly describe the network architecture you have, including how many layers, what kinds of layers (including activation functions and the number of trainable parameters).

Solution(a):

Network Architecture

The problem states that I can choose an example of a deep neural network using MNIST in the deep learning framework. Hence, I have selected *mnist_mlp.py* as an example to work with which represents a fully connected multi-layer perceptron. Here is a brief description of the network-

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 512)               401920
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 512)               262656
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                5130
=================================================================
Total params: 669,706
Trainable params: 669,706
Non-trainable params: 0
```

Figure: Network Overview

mnist_mlp.py is a feedforward neural network which describes a Multi-layer Perceptron or MLP (in short) Neural Network model. MLP is the most common type of neural network consisting of input, hidden and output layers. This is a feedforward neural network which means the data computed from the input layer moves towards the output layer in forward direction. The interconnection between layers are assigned with weights and bias.

Network Layers

Input layer: MNIST dataset contains 60,000 training data and 10,000 test data which is used in this project to train and test the neural network. Each data in the dataset is basically a 28X28 size grayscale image of one of the 10 digits (0 to 9) which is flattened to form a 784 elements-long array to pass to the first dense layer.

First Dense Layer: Here, the batch size is 512 which is necessary for stateful recurrent network. The activation function used here is ReLU or Rectified Linear Unit.

First Dropout Layer: This is the first dropout layer. a number of 0.2 is passed which sets every hidden unit to 0 with a probability of 0.2. This means that there is a 20% chance that the output of a given neuron will be forced to 0. dropout is used here for regularization which in turn will mitigate overfitting.

Second Dense Layer: In this layer, the given batch size is 512 and the activation function is ReLU.

Second Dropout Layer: This layer has the same values and properties of first dropout layer. This is again used for regularization.

Output Layer: In this layer, there are 10 neurons that represent all the 10 digits from 0 to 9 and Softmax is used here as an activation function.

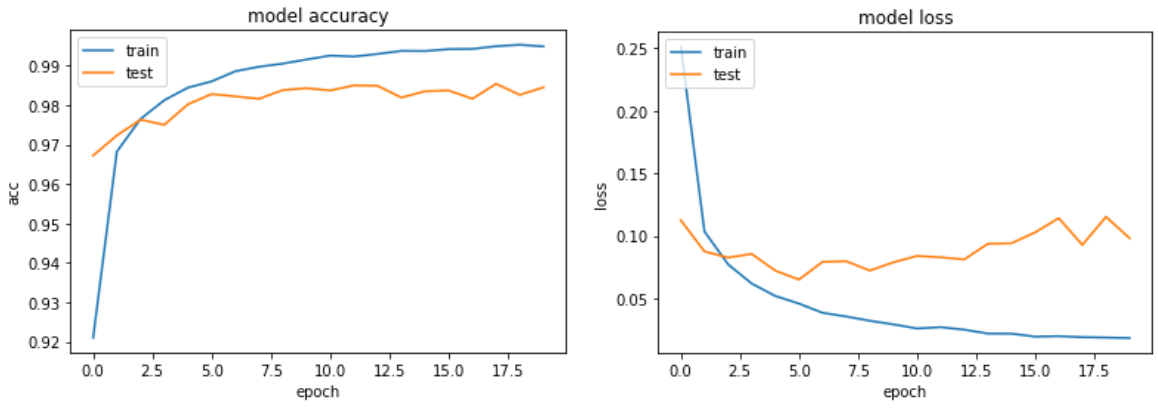
Total Number of Trainable Parameters

- Trainable parameters in the 1^{st} dense layer = 784 X 512 + 512 = 401,920
    - Here, 784 X 512 is the number of weight parameters and 512 is the number of bias parameters.
- Trainable parameters in the 2^{nd} dense layer = 512 X 512 + 512 = 262,656
    - Here, 512 X 512 is the number of weight parameters and 512 is the number of bias parameters.
- Trainable parameters in the 3^{rd} dense layer = 512 X 10 + 10 = 5,130
    - Here, 512 X 10 is the number of weight parameters and 10 is the number of bias parameters.
- Hence, The total number of trainable parameters = 669,706

(b) Report the performance on the training set and test set (which is actually a validation set in many implementations) by plotting the loss and recognition accuracy with respect to the number of epochs.

Solution(b):

The network is trained for 20 epochs with each epoch taking in the training set incrementally in batches of 128. The loss function is categorical cross-entropy, optimized by RMSprop optimizer. After every epoch, we check the loss value and accuracy of the model compared to the training, as well as the test set (passed in as validation set). The results are as follows:



(a) (b)

Figure: (a) accuracy vs epoch (b) loss vs epoch

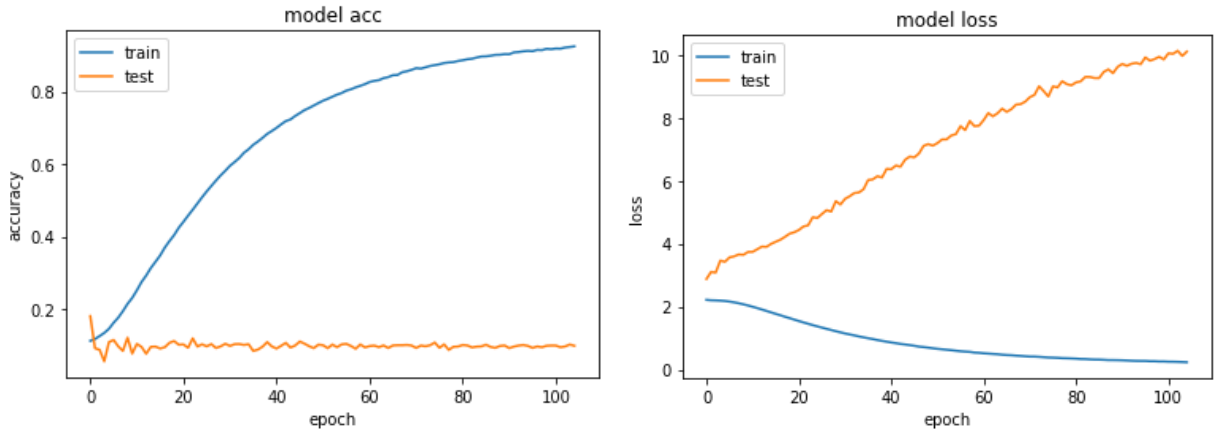
Problem 2 (statement)

Now modify the program you have for Problem 1 by assigning random labels (between 0 and 9) to the samples in the training set. After you train the network on the modified dataset, answer the following questions.

(a) Report the performance on both the training set and test set by plotting the loss and recognition accuracy with respect to the number of epochs.

Solution (a):

According to the 2nd problem statement we have to assign random labels. Now, for the second part of the problem, we assigned random labels (between 0 to 9) to all the samples in the training set. Then, we fed this randomized samples to the previous neural network only by changing the number of epoch, and removing dropout keeping all the other parameters fixed. We used a total of 105 epochs for this problem. The results for this randomization are given below:



(a) (b)

Figure: (a) accuracy vs epoch (b) loss vs epoch

b) Summarize the main performance difference between the two versions and try to explain why (hin: has the gap between the accuracy on the training set and the one on the test set increased significantly?)

Solution (b):

This problem demands a comparison between the two versions of the experiment. In the second case, where randomized samples is used, the accuracy of test dataset drops dramatically from almost 100% to almost 10 percent although the accuracy of the train samples is the same. This invokes a big generalization error. There might be many possible reasons.

In this experiment, it is clearly shown that there is no longer any relationship between the instances and the class labels. By randomizing samples we are actually transforming the training data. Since, there is no relation in between training data and testing data which certainly drops the accuracy metrics of test samples.

Regularizers are the standard tool in theory and practice to mitigate overfitting in the regime when there are more parameters than data points. In my case, a possible reason could be an overfitting of training data. Since we are not applying regularization technique like dropout there is nothing which can mitigate the overfitting

Problem 3 (statement)

Read the paper “understanding learning requires rethinking generalization”. Explain the challenges of achieving good generalization and some general strategies to improve generalization based on your experience with Problems 1 and 2 and the experiments in the paper.

Solution 3:

challenges:

In this paper, the author performs some experiments on several standard neural network architecture on data which are truly labeled and randomly labeled accordingly. Based on the findings the author questions about the existing technique used for generalizations.

Normally, generalization is measured by:
generalization error = |training error - test error|.

Usually, when overfitting occurs the training error becomes low but the test error becomes large which produces high generalization error. However, a complex neural network becomes breakable when overfitted. Statistical learning theory has proposed a number of complexity measures for controlling generalization error like VC dimension, Rademacher complexity and uniform stability.
In order to explore generalization, in a first set of experiments, the author trained a copy of data where randomized labels were used instead of true labels. And the finding of the experiment suggest that deep neural network easily fit random labels. Later, the author replaced true images by completely random pixels and used the same network again. again in this case, the neural network continue to fit the data as well with zero training error. This shows that despite their structure , cnn can fit random noise. However, in the case of randomized samples, the test error increases which in turn increases generalization error.
Based on the randomization experiments, the author describes the challenges for several traditional approaches that are supposed to control generalization. these are described below,

Radmecher complexity and VC-dimension: Since the author’s randomization tests suggests that many neural networks fit the training set with random labels perfectly, it is expected that Rademacher complexity of a hypothesis class on a dataset would be 1 for the corresponding model class. This is a trivial upper bound on the Rademacher complexity that does not lead to useful generalization bound in realistic settings. A similar reasoing can be applied to VC-dimension. These behaviours do not explain the generalization behaviour.

Uniform stability: Uniform stability of an algorithm A measures how sensitive the algorithm is to the replacement of a single example. However, it is a property of algorithm which does not take into account the distribution of the labels or the specifics of the data.

Now the solutions of these problems could be explained below,

Solutions:

In this paper the author shows that in deep learning, explicit generalization plays a role as a tuning parameter that often helps to improve the final test error of a model.

In the case of implicit regularization, early stopping could improve the generalization performance. Batch normalization (Ioffe & Szegedy, 2015) is an operator that normalizes the layer responses within each mini-batch. It has been widely adopted in many modern neural network architectures such as Inception. However, the absence of all regularization does not necessarily indicate poor generalization error as the networks continue to perform well after all the regularizers removed in their experiment. Hence, bigger gain can be conquered by modifying the model architecture, although regularization is important.