STATS M231A Homework 1

Laxman Dahal

Abstract

This homework aims at giving theoretical and practical background on multiple machine learning algorithms. The written part emphasizes on theoretical aspects where we are required to compute the derivative of the loss. The derivatives are computed using chain rule. The coding part required understanding the tutorial on Pytorch and experiment with the baseline code with different modifications. The comparative study found that out of many parameters that can be manipulated, the most impactful one is to increase the number of convolution layers to create a denser network such that it will have a higher number of parameter to learn from.

1 Problem 1- Written part

Consider a multi-layer perceptron,

$$h_l = f_l(s_s)$$

$$s_l = w_l h_{l-1} + b_l$$

for l = 1,....,L, where $h_0 = x$ is one-hot vector for classification. h_L is the top layer score vector for computing soft-max probabilities. Derive the back-propagation derivatives.

The back-propagation derivatives can be computed using the chain rule as follows:

$$\begin{aligned}\frac{\partial L}{\partial h_{l-1}^\top} &= \sum_{k=1}^{d} \frac{\partial L}{\partial h_{l,k}} \frac{\partial h_{l,k}}{\partial s_{l,k}} \frac{\partial s_{l,k}}{\partial h_{l-1}^\top} \\ &= \sum_{k=1}^{d} \frac{\partial L}{\partial h_{l,k}} f_l'(s_{l,k}) W_{l,k} \\ &= \frac{\partial L}{\partial h_l^\top} f_l' W_l,\end{aligned}$$

where $W_{l,k}$ is the k-th row of W_l, and $f_l' = \text{diag}(f_l'(s_{l,k}), k = 1, ..., d)$.

$$\begin{aligned}\frac{\partial L}{\partial W_{l,k}} &= \frac{\partial L}{\partial h_{l,k}} \frac{\partial h_{l,k}}{\partial s_{l,k}} \frac{\partial s_{l,k}}{\partial W_{l,k}} \\ &= \frac{\partial L}{\partial h_{l,k}} f_l'(s_{l,k}) h_{l-1}^\top,\end{aligned}$$

thus,

$$\frac{\partial L}{\partial W_l} = f_l' \frac{\partial L}{\partial h_l} h_{l-1}^\top.$$

Similarly,

$$\frac{\partial L}{\partial b_l} = f_l' \frac{\partial L}{\partial h_l}.$$

2 Coding part: Neural Network (Fully connected Layer and CNN)

2.1 Neural Network

For the first part of the coding problem, convolution layers in the default setting were removed and four separate networks were created with 2, 3, 4, and 5 linear layers, respectively. This results in a fully connected neural networks because the out features of one linear layer equals to the input features of the subsequent linear layer. In other words, a fully connected neural network consists of series of fully connected layers that connect every neuron in one layer to every neuron in the other layer. The algorithm is used for image classification. Figure 1 shows an example network structure implemented. In this specific example, 5 fully connected linear layers are used. For other experimentation, this network is modified by reducing the linear layers. Please see attached code for more details. For forward pass, the underlying equations have been generalized as follows:

```
class Net_5linear(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(16 *2*32*3, 512)
        self.fc1a = nn.Linear(512, 256)
        self.fc1b = nn.Linear(256, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = torch.flatten(x, 1) # flatten all dimensions except batch
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc1a(x))
        x = F.relu(self.fc1b(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

Figure 1: Snippet of a network structure implemented for NN with 5 fully connected linear layers

$$h_0^{out} = X$$
$$h_i^{in} = h_{i-1}^{out} * W_i + B_i$$
$$h_i^{out} = F_i(H_i^{in})$$

where, i is the layer number, F is the activation function, W is neuron weights, and B is bias. The ReLU is used as the baseline activation function. It is formulated as follows:

$$y_i \sim \text{Bernoulli}(p_i)$$
$$p_i = sigmoid(H_i^T \beta)$$
$$h_{ik} = max(x_i^T \alpha_k, 0)$$

where the linear spline model is $f(x_i) = \beta_0 + \sum_{j=1}^{p} \beta_j \max(0, x_i - a_j)$.

The gradient is computed using Stochastic Gradient Descent (SGD). Let the training data be (x_i, y_i), $i = 1, ..., n$. Let $L_i(\theta) = L(y_i, x_i; \theta)$ be the loss due to (x_i, y_i). Now since the problem is a classification problem, it becomes

$$L(y_i, x_i; \theta) = -logp(y_i|x_i)$$

where $p(y_i|x_i)$ is modeled by a neural network with parameters θ. A softmax layer is used at the top. Let $Ł(\theta) = \frac{1}{n} \sum_{i=1}^{n} L_i(\theta)$ be the overall loss averaged over the whole training dataset. The gradient descent is one potential algorithm to compute the gradient but it is extremely low because it has

to sum over all the examples. Computationally more pleasing idea is to randomly select i from $\{1,2,...,n\}$ and update θ by doing the following operation

$$\theta_{t+1} = \theta_t - \eta_t L_i'(\theta_t),$$

where η_t is the step size or learning rate, and $L_i'(\theta_t)$ is the gradient only for the i-th example.

The summary of comparison of different networks at 12,000 batch are shown in the Table 1. Please note that the losses reported in the Table 1 correspond to the 12,000 mini batch. To see the values for every 2000 mini-batches, please see the attached code.

Table 1: Comparison of training loss for 10 epochs with different linear layers for the 12,000th batch.

Epochs	**Loss 2 Layers**	**Loss 3 Layers**	**Loss 4 Layers**	**Loss 5 Layers**
1	1.565597	1.544693	1.554962	1.586585
2	1.491567	1.441870	1.430547	1.436592
3	1.430865	1.364034	1.350098	1.346465
4	1.382255	1.306483	1.266175	1.251513
5	1.366720	1.267281	1.225989	1.195680
6	1.315761	1.233986	1.173042	1.140344
7	1.295316	1.185828	1.128131	1.064082
8	1.283534	1.170780	1.093450	1.019045
9	1.207584	1.130068	1.020899	0.942915
10	1.215505	1.103396	0.998471	0.902194

The final training loss and accuracy and testing loss and accuracy are shown in the Table 2 below. Please note that the numbers represent values for the 10th epoch. For comparison between all 10 epochs, please see the attached jupyter notebook.

Table 2: Comparison of training and testing loss and training and testing accuracy for networks with 2, 3, 4, and 5 linear layers, respectively.

No. of Linear Layers	**Train Loss**	**Train Accuracy**	**Test Loss**	**Test Accuracy**	Total Trainable Params
2	0.24345	0.6636	0.42409	0.4994	369970
3	0.21209	0.6952	0.41165	0.5072	379774
4	0.16591	0.7602	0.43880	0.5212	828542
5	0.11859	0.8301	0.47987	0.5401	1746558

Figure 2 show comparison of the training and testing loss and training and testing accuracy for all 10 epochs. The graphs indicate that as the number of linear layer increases, the training and testing loss and accuracy increases. The result makes sense because ad highlighted in Table 2, the number of learnable parameter increases from 369970 for 2 linear layers to 1746558 parameters for 5 linear layers. As the network learns more parameters, it is expected to train and test with higher accuracy.

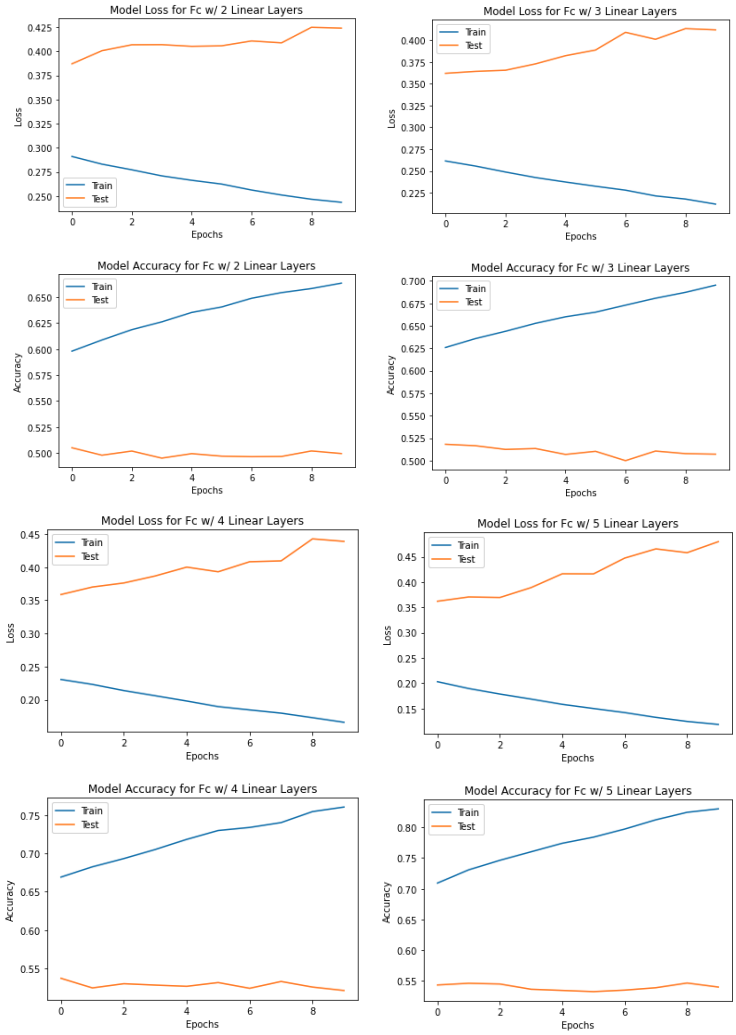

Figure 2: Training and testing performance comparison for networks with different number of linear layers

2.2 CNN

Different from neural network, convolution neural network has convolution layer (h_i) organized into a number of feature maps. Each map is often referred to as channel and is obtained by applying a filter or kernel operation on h_{i-1}. Each filter is then summed considering local weights, where a bias term is also included followed by a non-linear transformation such as ReLU. CNN is frequently used in computer vision problems such as object detection and object identification. It is a classification problem.

In the neural network, $h_l = f_l(W_l h_{l-1} + b_l)$, the linear transformation $s_l = W_l h_{l-1} + b_l$ that maps h_{l-1} to s_l can be viewed as the underlying equation in addition to the ones mentioned above. The equations mentioned thus far are also applicable.

2.2.1 Default Network Structure

Figure 3 shows the model loss and model accuracy for the training and testing dataset when default network structure is used. It can be seen that both the training and testing accuracy is at about 60 %. The

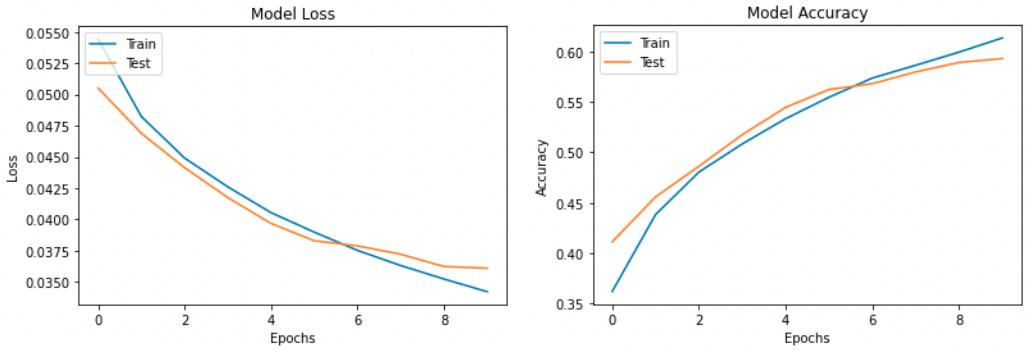

Figure 3: Model loss and accuracy comparison for 32 batches and 10 epochs with the default network settings.

Figure 4 shows a random 20 images that have been either correctly or incorrectly labelled. As highlighted in red, a total of five images have been labelled incorrectly which means that for the 20 random sample, the testing accuracy is 75%. The overall accuracy of the default model is 59%. Since, the 20 images are selected at random, it's accuracy does not really mean anything. Sometimes it could be greater than 59% sometimes it could be lower.

Figure 4: Testing the labels of the figures

The following subsections look into the different experiments.

2.2.2 Different Activation Functions

Table 3 highlights the results obtained from training and testing the CNN with three different types of activation functions, namely ReLU, Leaky ReLU, and Tanh. The results indicate that Tanh has the highest training and testing accuracy.

Figure 5 shows how the training and testing loss and accuracy changes as a function of the epochs.

Table 3: Comparison of training and testing performances for different activation functions.

Activation	Train Loss	Train Accuracy	Test Loss	Test Accuracy
ReLU	0.03423	0.6134	0.03610	0.5930
Leaky ReLU	0.03507	0.6029	0.03588	0.5931
Tanh	0.02604	0.7072	0.03198	0.6502

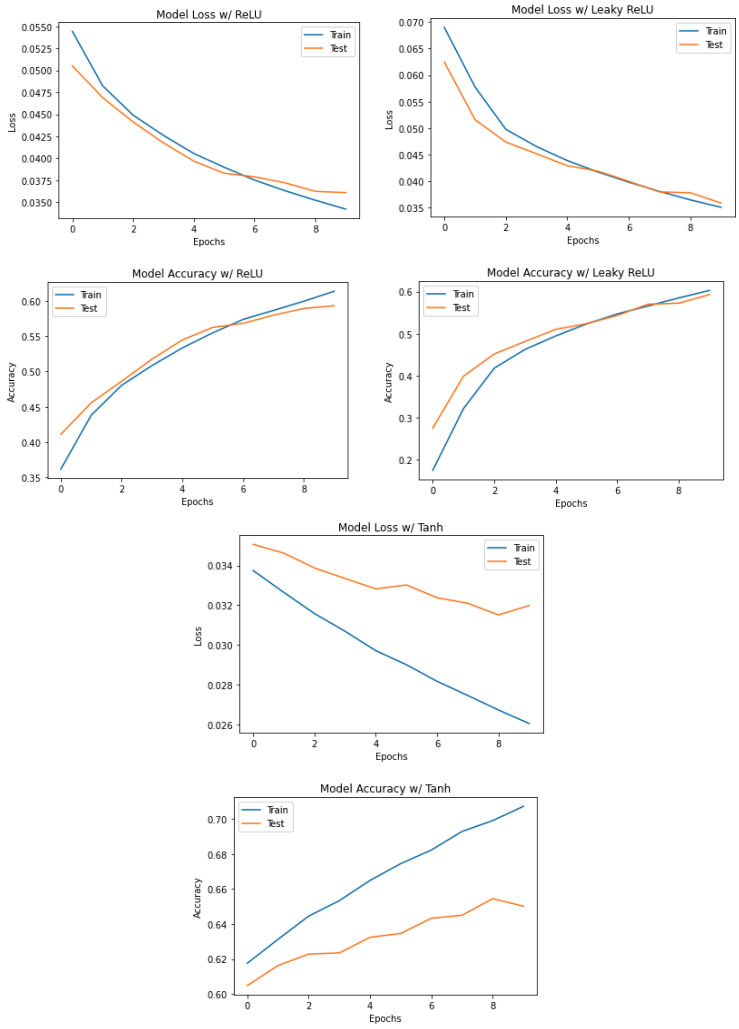

Figure 5: Training and testing loss comparison and training and testing accuracy comparison for networks with different activation functions

2.2.3 Different Learning Rate

Table 4 shows the results obtained from the training and testing the CNN with different learning rates. In addition to the default learning rate, two additional learning rates (0.0005 and 0.01) were used. The results indicate that the learning rate of 0.01 has better performance.

2.2.4 Different Conv Layers

Table 5 outlines the training and testing loss and accuracy obtained when different numbers of convolutional layers were used in the network. The default code had two conv layers. The results demonstrate that the network with three convolutional layers perform better as expected because

Table 4: Comparison of training and testing performance with different learning rates.

Learning Rate	Train Loss	Train Accuracy	Test Loss	Test Accuracy
0.0005	0.04399	0.4937	0.04407	0.4961
0.001 (default)	0.0323	0.6134	0.03610	0.5930
0.01	0.02776	0.6872	0.03616	0.6161

the network with three conv layers have much more parameters to learn from as compared with the networks with either one or two conv.

Table 5: Comparison of training and testing performance of the networks with different number of convolutional layers.

Conv Layers	Train Loss	Train Accuracy	Test Loss	Test Accuracy
1	0.04985	0.42351	0.051247	0.03921
2 (default)	0.0323	0.6134	0.03610	0.5930
3	0.01463	0.8347	0.02833	0.7195

2.2.5 Double Channels

In this experiment, the channels were doubled. The input channel of the first Conv layer cannot be double because the input only has three channels (R, G, B). However, the output channel and the input channels of the following Conv was doubled. For the double channel, the network structure used is shown in Figure 6.

```
class Net2(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 12, 3, padding = 1)
        self.conv3 = nn.Conv2d(12, 32, 3, padding=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(32 * 4  * 4, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv3(x)))
        x = torch.flatten(x, 1) # flatten all dimensions except batch
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

Figure 6: Network structure used while doubling the default channels

Figure 7 shows the loss and accuracy comparison for network with two convolutional layers. The figure on the left represents network where default setting were used and the figure on the right represents where the channels were double. The result demonstrates that when the channels are doubled, both the training and testing accuracy increases.

How to run There are two jupyter notebook attached with this written report. The first notebook titled "cifar10_tutorial_part1.ipynb" is the notebook for the first part of the coding part with different number of linear layers. The second notebook titled "cifar10_tutorial_part2.ipynb" is used for the CNN. It is important to note that the default settings and all the experiments are conducted using this notebook. Both the notebooks can be executed in the local machine.

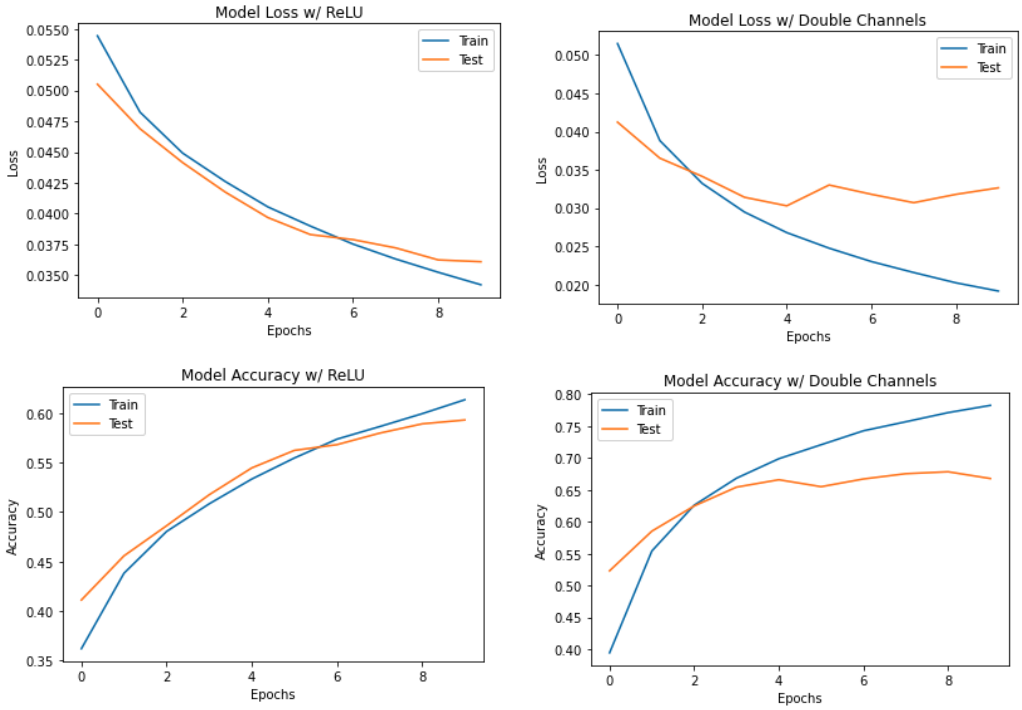

Figure 7: Model loss and accuracy comparison for default setting (left) and the network with double channels (right).

2.3 Conclusion

The experiments indicate that there are multiple ways to increase the training and testing accuracy of the network. Out of all the experiments that were performed, changing the number of convolution layers had the biggest impact in the performance. It makes sense that the deeper the network is, the better the performance is because the network would have a higher number of learnable parameter to make a decision. It indicates that even higher performance can be achieved by making the network even denser. The denser network would required higher computational capacity which is why it makes sense to use GPU for this purpose.