Individual Final Report

Jianing wang

1. Introduction

In today's world, businesses can no longer turn a blind eye to the opportunity for expansion, by providing goods and services online instead of just in stores. To target specific markets and demographics, implementing the use of A.I. and big data are becoming increasing mainstream. For example, if we can classify those pictures which have been uploaded by users to social media, then we could know the preference of users and classify the picture. On top of that, we can send different type of advertisements to different groups of users based on their preference, which would decrease the number of aimless advertisements.

This study aims to build a model to classify 100 different types of pictures. To solve this problem, we plan to use different platforms (Pytorch and Tensorflow) and network (MLP and CNN) to implement the network.

My part is to implement the data set in Pytorch by MLP.

2. Learning network and algorithm – MLP in Pytorch

PyTorch is one of the open-source machine learning libraries running on Python and built on Torch. PyTorch has more flexibility to offer than some other open-source platform because it supports dynamic computational graphing abilities.

In this project, the type of deep learning network we used is called a Multilayer Perceptron (MLP). The MLP is a deep, artificial neural network composed of more than 1 perceptron or layer. In Figure 1, the basic components of MLP model are comprised of: input, hidden layers and output layer. Data is brought in the neuron network from the Input component, and then the input data will proceed doc product with weight vector. After that, output of current layer will come out after finishing computation in transfer function. After all the computations which are in one or more than one hidden layer complete, the output layer will make the final prediction.

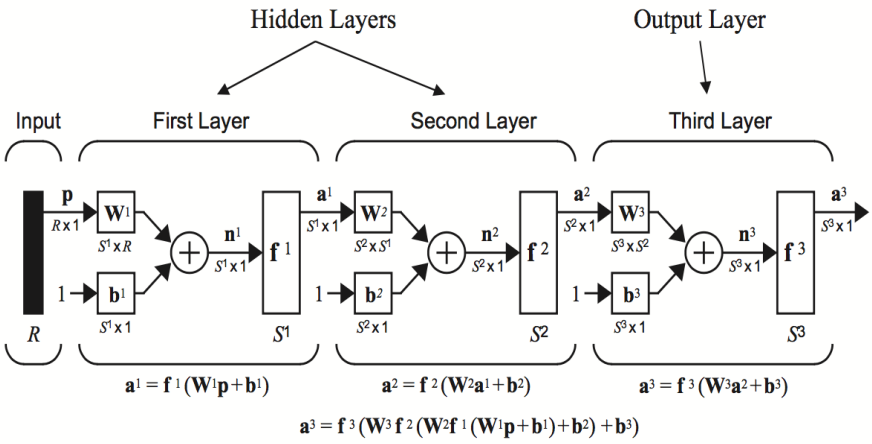


Figure 1

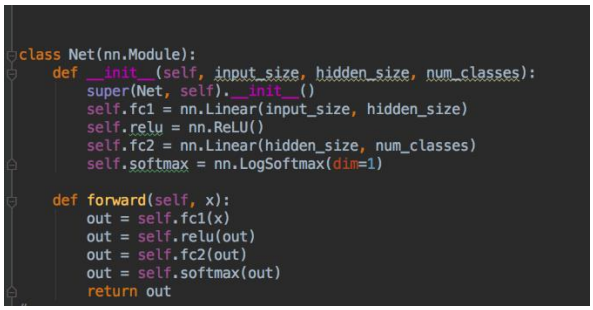
3. What I did in this project

To start building the MLP model in PyTorch, we need to decide the initial parameters of the model. These parameters are: input size, batch size, and neuron size (Figure 2). In CIFAR-100 dataset, all of the inputs are color images, meaning that each of them will have three color channels. The number of output layers should be equal to the number of classes. Hidden size presents the number of neurons. Also, Mini batch has been picked in this study instead of batch or stochastic gradient descent (SGD), because it is faster than the stochastic method and will achieve a higher accuracy than the batch method.

```
input_size = 3 * 32 * 32
hidden_size = 50
num_classes = 100
num_epochs = 20
batch_size = 100
learning_rate = 0.001
```

Figure 2

Second, I initiated the frame of the network (Figure 3). Relu and Softmax transfer functions have been chosen in the hidden layer and output layer respectively. On one hand, the Relu function is sparse and can reduce the likelihood of vanishing gradient. On the other hand, Softmax function can transfer the data to the range of 0-1, which is suitable for multi-classification.



```
class Net(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, num_classes)
        self.softmax = nn.LogSoftmax(dim=1)

    def forward(self, x):
        out = self.fc1(x)
        out = self.relu(out)
        out = self.fc2(out)
        out = self.softmax(out)
        return out
```

Figure 3

There are four parameters have been tested in this project: number of layers, number of neurons, types of optimizer and batch size.

As shown in Table 1, with increasing the number of layers, accuracy declined Theoretically, the network with more layers is more complex network, so the network needs more training time to get a better result, However, even though we doubled the epoch number, the accuracy is still lower than two-layer network. It may get a higher accuracy if we keep adding epoch number, but considering cost from training time, we decided to use two-layers network. After trying several optimizer tests, we decided to use SGD which has highest accuracy under same iteration (Table 2).

	Layer	Total parameters	Accuracy
Test	2	317200	23 %
Test1	4a	316856	21%; 22% (double epochs)
Test2	7	317400	10%

Table 1 Accuracy with different layers under same total parameters

Optimizer	SGD	Adam	Adagrad	RMSprop	Adadelta
Accuracy	22%	21%	18%	9%	9%

Table 2 Accuracy with different optimizers

Small batch size needs more time to converge, and it takes less time to converge in bigger batch size.

If the memory capacity is bigger enough, we can choose large size of mini batch, but if the batch size is too big, it will increase the cost to get a higher accuracy. With same number of iterations, increasing batch size will decrease the training time, theoretically. Based on the result show in Table 3, when batch size is larger than 200, training time stops declining. What's more, larger batch size takes longer time to have a higher accuracy. As a result, 50 or 200 will be considered

as the final batch size in the network based on the consequence in Table 3, and 500 will be the hidden size (Table 4).

Batch size	50	100	200	500	1000	2000
Accuracy	23.84%	22.41	19%	15.73	12%	5%
Time	280	265	243	245	248	248

Table 3 Different batch size with accuracy and training time

Accuracy	19.45	20.2	21	21
Time	253	250	258	268

Table 4 Accuracy with different number of neurons

If learning rate is too small, it will be hard to converge in the network. Because the step size for updates of weight and bias are tiny. In another word, it will take longer time to get a better performance. However, it doesn't mean that the bigger the learning is, the quicker to get a high performance. The reason is that if learning rate is too big, it will lead to overfitting (light blue line in Figure 4).

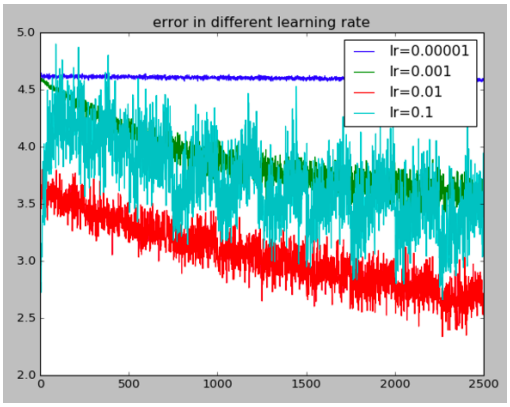


Figure 4

4. Result

According to the discussion above, we summarized several combinations which are shown in Table 5.

Accuracy	Learning rate	Batch size	epochs
20%	0.01	50	50
24%	0.01	200	50

25%	0.001	50	50
22%	0.001	200	50

Table 5

After adjusting several parameters compared above, the highest accuracy we got in MLP was 27%, and accuracy in subclasses was around 27% also (Figure 5).

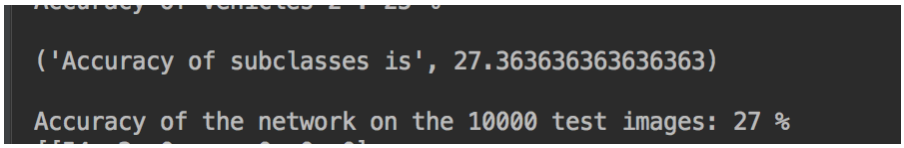


Figure 5

Since the accuracy of subclasses and total classes are same basically, so we plot the confusion matrix for subclasses instead of entire classes (Figure 6). It didn't show any spots with saturated color outside of diagonal, which means there is no a specific misclassification.

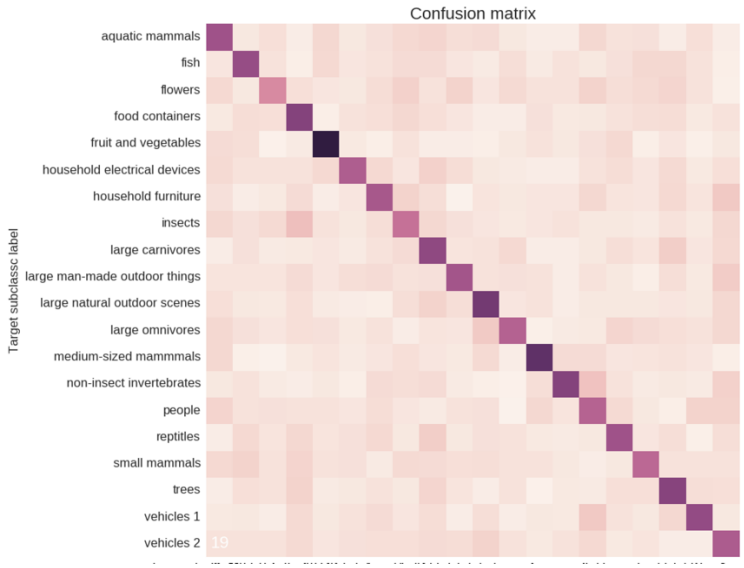


Figure 6

As shown in Figure 7, the best performance happened in class fox, mountain, and spider with accuracies around 73%, 70%, 65% respectively. Three of the top 10 classes belong to fruit and vegetables subclass, and half of them belong to different mammals' group. The finding in Figure 7 is consistent with the result in Figure 8 which the best performance is class fruit and vegetables.

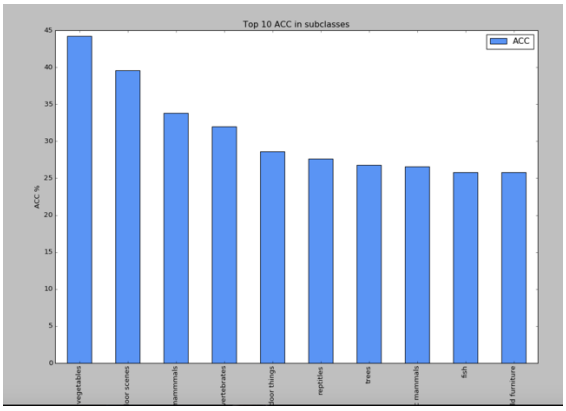


Figure 7

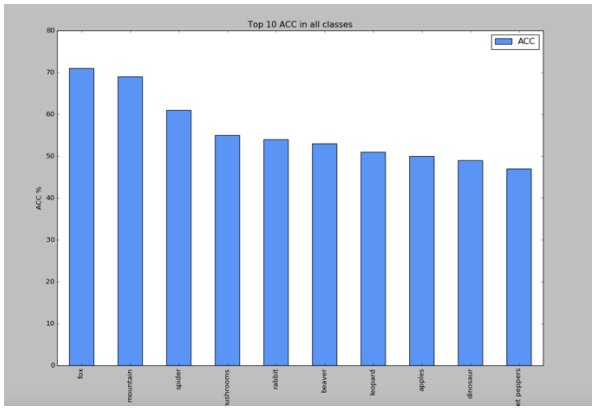


Figure 8

5. Conclusion

The result reveals that MLP is not a good model for CIFAR 100 as the accuracy is only 27%. However, it is suggested that this model is more suitable to classify fruit and vegetables and large natural outdoor senses whose performance are over 40%.

Comparing to CNN, the performance in MLP is lower. In CNN, the weight function is kernel which is image instead of vector. So, CNN simplifies the training processing, and using the image to train image will be more efficient and faster comparing with MLP.

6. Percentage of code

File name	Percentage
MLP in PyTorch.py	(130-20) / (258) * 100 = 44.5%
MLP-class-accuracy ploy.py	(130-25) / (411) * 100 = 26.5%
MLP-confusion matrix.py	(130-21) / (270) * 100 = 38%

7. Reference

Official hosting page for CIFAR-100 dataset can be found at:
https://www.cs.toronto.edu/~kriz/cifar.html.