CENG499 HW1 Report

Onat Özdemir

1 Sanity check

According to `https://www.cs.toronto.edu/~kriz/cifar.html`, the number of samples for each label is equal. Hence our dataset is balanced. I did my calculations based on this information.

1.1 Theoretical values

1.1.1 Loss calculation

Cross-entropy loss function;

$$H(p,q) = -\sum_{m \in M} p(m) log q(m)$$

For an untrained model and balanced dataset with 10 classes, p(m) and expected q(m) = 0.1. Then;

$$H(p,q) = -0.1 \sum_{m \in M} log(0.1) = -0.1 * (10 * log(0.1))$$

$$H(p,q) = -log(0.1) = 2.30258509299$$

1.1.2 Test set accuracy calculation

The dataset contains 10 labels; therefore, for an untrained network and balanced dataset probability of choosing any of these labels should be equal to 0.1. Hence, we should expect the accuracy to be 10%.

1.2 Empirical values

1.2.1 Loss

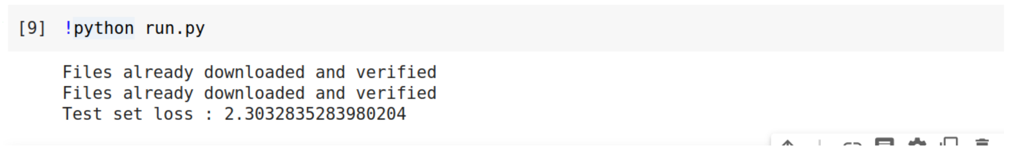
To calculate the test set loss of untrained model, I used the following code;

```python
model = ANN(2, [32 * 32, 512, 10], "relu").to(device)

loss_function = torch.nn.CrossEntropyLoss()
model.eval()
accum_test_loss = 0
with torch.no_grad():
    for j, (imgs, labels) in enumerate(test_loader, start=1):
        imgs, labels = imgs.to(device), labels.to(device)
        output = model(imgs)
        accum_test_loss += loss_function(output, labels).item()

    print(f'Test set loss : {accum_test_loss / j}')
```

which calculates the test set loss for an untrained model with 2 layers and 512 neurons for the hidden layer (I got similar results for the other configurations). As a result of this code, I got this result;



```
[9] !python run.py

    Files already downloaded and verified
    Files already downloaded and verified
    Test set loss : 2.3032835283980204
```

Figure 1: Sanity Loss

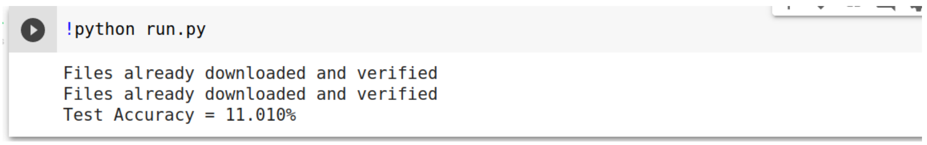
Since the loss is quite similar to the theoretical loss, we validate the sanity of our model for loss.

1.2.2 Test set accuracy

To test the accuracy of untrained model, I used the following code;

```python
model = ANN(2, [32 * 32, 512, 10], "relu").to(device)
test(model, device, test_loader)
```

which calculates the test set accuracy for an untrained model with 2 layers and 512 neurons for the hidden layer (I got similar results for the other configurations). As a result of this code, I got this result;



```
!python run.py

Files already downloaded and verified
Files already downloaded and verified
Test Accuracy = 11.010%
```

Figure 2: Sanity Accuracy

Since the accuracy is quite similar to the theoretical accuracy, we validate the sanity of our model for accuracy.

2 Creation of validation set

By using the torch.utils.data.random_split function, I divided the initial training set as final training and validation sets. The size of the final training set was selected as 0.8 * size of the initial training set.

3 Hyperparameter settings

3.1 1-layer (0-hidden layer) network

For each combination, I trained the 1-layer fully connected network for 30 epochs. The below results belong to the model states with the best validation losses and corresponding validation accuracy results.

	Learning Rate		
Activation Function	0.0001	0.001	0.01
Relu	(2.0139, 30.2300)	(2.0475, 28.5100)	(2.5784, 21.4100)
Sigmoid	(2.0134, 29.8300)	(2.0551, 28.6200)	(2.5647, 23.8500)
Tanh	(2.0141, 30.0100)	(2.0620, 27.1000)	(2.5585, 23.0300)

Table 1: 1-layer ANN validation loss and validation accuracy results. Left ones are the validation loss and the right ones are the validation accuracy.

3.2 2-layer (1-hidden layer) network

For each combination, I trained the 2-layer fully connected network for 30 epochs. The below results belong to the model states with the best validation losses and corresponding validation accuracy results.

I tried two values which are 512 and 1024, as the number of neurons in the hidden layer.

	Learning Rate		
Act Func - Layer size	0.0001	0.001	0.01
Relu / 512	(1.5663, 47.1100)	(1.6706, 43.3300)	(2.2392, 30.1600)
Relu / 1024	(1.5614, 46.8900)	(1.7061, 44.1500)	(2.9937, 28.4500)
Sigmoid / 512	(1.8106, 37.5800)	(1.7030, 40.6900)	(2.1958, 28.9600)
Sigmoid / 1024	(1.7853, 38.4300)	(1.6949, 42.3400)	(2.5124, 26.5800)
Tanh / 512	(1.7155, 41.2100)	(1.8000, 37.6900)	(2.4437, 22.0400)
Tanh / 1024	(1.7003, 41.7600)	(1.8381, 36.5000)	(3.0941, 19.2600)

Table 2: 2-layer ANN validation loss and validation accuracy results. Left ones are the validation loss and the right ones are the validation accuracy.

3.3 3-layer (2-hidden layer) network

For each combination, I trained the 3-layer fully connected network for 30 epochs. The below results belong to the model states with the best validation losses and corresponding validation accuracy results.

I tried two neuron configurations;

- 1st hidden layer with 1024 neurons and 2nd hidden layer with 512 neurons
- 1st hidden layer with 512 neurons and 2nd hidden layer with 256 neurons

	Learning Rate		
Act Func / Layer sizes	0.0001	0.001	0.01
Relu / 512 - 256	(1.5729, 45.7300)	(1.6323, 44.0200)	(2.1824, 15.1700)
Relu / 1024 - 512	(1.5600, 46.5900)	(1.6458, 43.4900)	(2.2042, 15.1400)
Sigmoid / 512 - 256	(1.6879, 41.2900)	(1.6708, 42.8400)	(1.9723, 28.9800)
Sigmoid / 1024 - 512	(1.6408, 42.8700)	(1.6611, 43.2100)	(2.0839, 25.5000)
Tanh / 512 - 256	(1.6581, 42.6300)	(1.6918, 41.1600)	(2.2299, 21.4500)
Tanh / 1024 - 512	(1.5991, 44.0900)	(1.7369, 39.4900)	(2.3509, 21.3500)

Table 3: 3-layer ANN validation loss and validation accuracy results. Left ones are the validation loss and the right ones are the validation accuracy.

4 The best hyperparameter configurations

4.1 Results

4.1.1 1-layer (0-hidden layer) network

- number of neurons in hidden layer: 0
- learning rate: 0.0001
- activation function: Relu

Test Set Accuracy : 30.290%

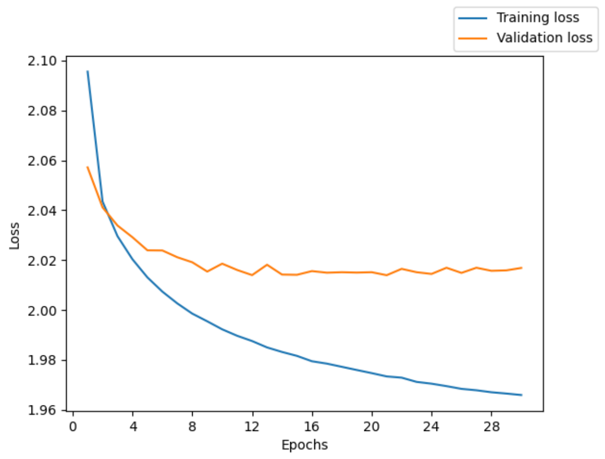


Figure 3: Epoch versus Training loss - validation loss for 1-layer ANN

4.1.2 2-layer (1-hidden layer) network

- number of neurons in hidden layer: 1024
- learning rate: 0.0001
- activation function: Relu

Test Set Accuracy : 46.290%

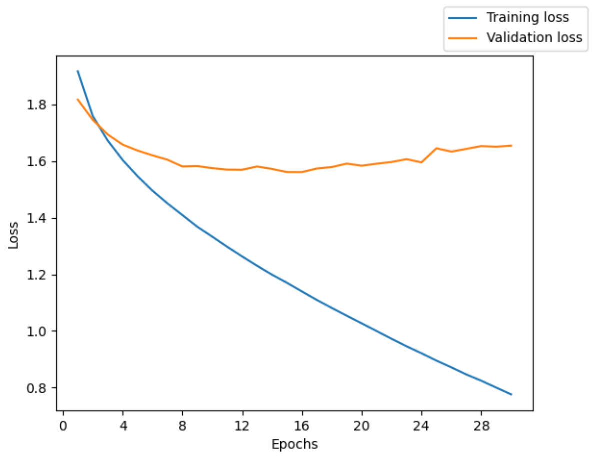


Figure 4: Epoch versus Training loss - validation loss for 2-layer ANN

4.1.3 3-layer (2-hidden layer) network

- number of neurons in hidden layer: 1024 in the first layer and 512 in the second hidden layer
- learning rate: 0.0001
- activation function: Relu

Test Set Accuracy : 45.050%

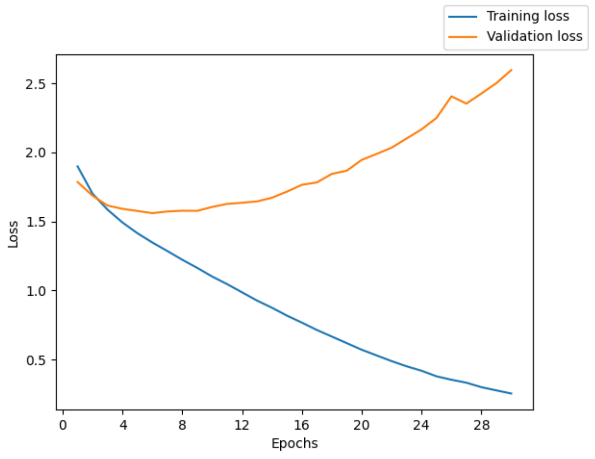


Figure 5: Epoch versus Training loss - validation loss for 3-layer ANN

4.2 Countermeasures against overfitting

First, I used the early stopping approach to detect the overfitting and stop the training earlier. Basically, if there were more than five consecutive increases in validation loss, I stopped the training. However, most of the time, it either stopped quite early (before reaching the best validation loss) or could not catch the overfitting.

My current approach is to train the network for a constant number of epochs (30 epochs) and save the state of the model with the best validation loss. That means for each epoch, I check whether the current validation loss is better than the previous best validation loss, and if this is the case, then I save the current state of the model.

Additionally, one can understand overfitting when the validation loss starts constantly increasing while the training loss continues to decrease. For instance, in Figure 5, approximately in Epoch 3, validation loss starts increasing while training loss continues to decrease. We can say that after Epoch 3, the model begins overfitting; hence, we should stop training at that point.

5 Discussions

5.1 Is the accuracy a suitable metric to measure the performance?

In my opinion, in a case like ours (with a balanced dataset), accuracy is a suitable metric to assess the performance of the machine learning model. However, for instance, if we had an unbalanced dataset with lots of negative samples and a few positive samples, then we would have to use different metrics.

5.2 What are the advantages/disadvantages of using a small learning rate?

The main disadvantage of using a small learning rate is to increase the time it takes to converge. However, since the step size is small, the possibility of jumping over the minimum point is quite low.

5.3 What are the advantages/disadvantages of using a big learning rate?

With a big learning rate, basically, we roughly search for the minima. Therefore, it is quite possible to jump around the minimum point. However, since the step size is large, it converges to the minima quicker. (it does not mean that training will be quicker, if it misses the local minima, it might have taken more time than the small learning rate)

5.4 What are the advantages/disadvantages of using a small batch size?

The small batch size provides a noisy estimate of the true gradient. As an extreme case, we can think of the stochastic gradient descent where each batch contains just one sample. In that case, because of this noisy nature, it is quite possible to escape from the local minima and converge to global minima. However, since it is noisy, it may take more iterations to obtain good results. It also uses less memory, which is good.

5.5 What are the advantages/disadvantages of using a big batch size?

One of the main disadvantages of using big batch sizes is the higher memory usage. Let's assume that we have one batch (standard gradient descent), then during training, all samples in the dataset must be placed into memory. However, it also provides a more precise estimation of true gradient and hence requires fewer iterations to converge.