Report

Mustafa Ozan Alpay

25/04/2021

1 Sanity Checks

Since we know that our dataset is balanced, we can do the following checks.

1.1 Loss

By using the Cross Entropy loss function formula:

$$
\begin{aligned}
H(p,q) &= -\sum_{x \in X} p(x) \log q(x) \\
\text{Since } p(x) = 1 \text{ and } q(x) = 0.1 \text{ due to 10 labels} & \\
&= -\sum_{x \in X} 1 \cdot \log(0.1) \\
L &= -\log(0.1) \\
&= 2.30259
\end{aligned}
$$

By adding the following lines right after the `log_softmax` call in my model, I was able to acquire loss values.

model.py

```
loss = F.nll_loss(x, labels)
print(loss.item())
```

Loss values from PyTorch: 2.3139240741729736, which fits well with my calculations.

1.2 Accuracy

Since there are 10 labels, initially, each label should have 0.1 as its probability. By printing the result of the softmax and exiting right after, I was able to see the initial values, which are as follows:

```
python
```

```
tensor([0.1129, 0.1025, 0.0823, 0.0817, 0.0940, 0.1081, 0.1071,
0.1032, 0.1019,0.1061], grad_fn=<SelectBackward>)
```

As we can see from the tensor above, each value is very close to 0.1, which meets out expectations.

2 Separating Validation Set

By using `torch.utils.data.random_split` , I managed to split the data randomly into two pieces, namely training and validation sets. I used 6-fold validation with training set consisting of 25.000 images and the validation set consisting of 5.000 images.

3 Hyperparameter optimization

3.1 1-layer (0-hidden-layer) network

While training the 1-layer network, I created a Fully-Connected layer, and I trained 6 different networks that are combinations of the following parameters:

- Learning Rate: 0.01, 0.003, 0.001, 0.0003, 0.0001, 0.00003

I used 100 epochs for the training.
The validation scores for the networks can be found at Table 1.

AF and HS	Learning Rate					
	0.01	0.003	0.001	0.0003	0.0001	0.00003
-, -	30.48	35.56	36.58	39.32	39.06	38.3

Table 1: 1-layer network

3.2 2-layer (1-hidden-layer) network

While training the 2-layer network, I created two Fully-Connected layers, and I trained 54 different networks that are combinations of the following parameters:

- Activation Functions: sigmoid, tanh, ReLU
- Hidden Layer Neuron Size: 256, 512, 1024
- Learning Rate: 0.01, 0.003, 0.001, 0.0003, 0.0001, 0.00003

I used 100 epochs for the training.
The validation scores for the networks can be found at Table 2.

Layer Activations	Learning Rate					
	0.01	0.003	0.001	0.0003	0.0001	0.00003
S, 256	10.12	9.72	53.2	55.66	54.82	43.08
S, 512	10.26	9.44	53.46	54.72	52.94	42.84
S, 1024	9.68	10.12	55.72	52.58	53.38	43.74
T, 256	9.72	10.12	28.92	56.32	55.74	51.02
T, 512	11.02	9.64	26.26	56.14	53.08	49.38
T, 1024	11.02	9.44	29.42	54.92	53.12	50.1
R, 256	9.44	35.98	53.86	57.36	55.54	50.86
R, 512	9.64	43.16	55.64	56.64	53.86	51.6
R, 1024	10.14	10.12	53.8	56.02	55.66	54.42

Table 2: 2-layer network

3.3 3-layer (2-hidden-layer) network

While training the 3-layer network, I created three Fully-Connected layers, and I trained 54 different networks that are combinations of the following parameters:

- Activation Functions: sigmoid, tanh, ReLU
- Hidden Layer Neuron Size: 256, 512, 1024
- Learning Rate: 0.01, 0.003, 0.001, 0.0003, 0.0001, 0.00003

I used 100 epochs for the training.
The validation scores for the networks can be found at Table 3.

Layer Activations	Learning Rate					
	0.01	0.003	0.001	0.0003	0.0001	0.00003
S, 256	10.24	10.12	51.28	52.7	55.84	45.42
S, 512	9.44	9.72	45.2	52.3	54.72	47.44
S, 1024	10.12	10.26	36.42	51.56	53.62	49.34
T, 256	9.64	10.24	10.26	56.72	53.86	54.04
T, 512	10.24	9.92	20.18	55.94	54.1	53.7
T, 1024	10.12	9.96	18.26	55.64	55.72	56.1
R, 256	10.24	33.16	57.1	58.58	58.44	57.5
R, 512	9.72	43.46	58.88	55.7	56.3	56.64
R, 1024	10.26	21.48	57.94	54.8	55.32	56.14

Table 3: 3-layer network

4 The best hyperparameter

4.1 Results

From the tables Table 1, Table 2 and Table 3, we can see that the best validation score is received with a 3-layer (2-hidden-layer) network when using the following hyperparameters:

Hidden Layer Size	512
Activation Function	ReLU
Learning Rate	0.001
Epochs	100
Test Accuracy	0.584267

The train loss - validation loss graphics can be seen at Figure 1.

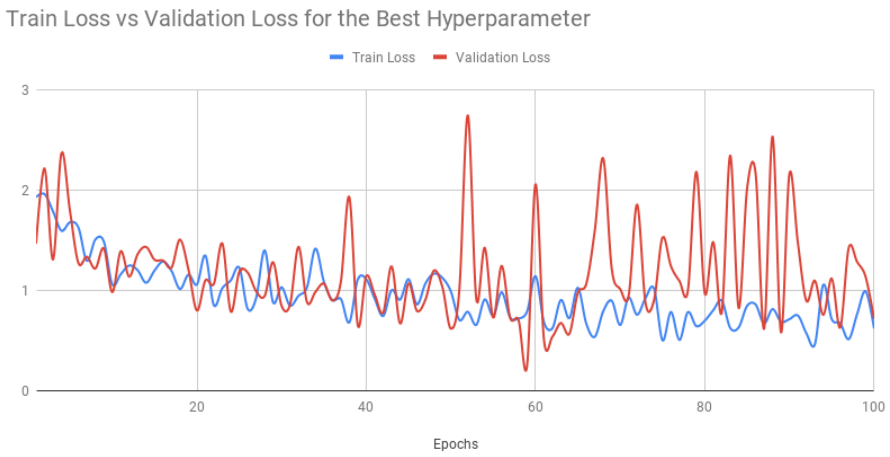

Figure 1: Train Loss vs Validation Loss for the Best Hyperparameter

4.2 Overfitting countermeasures

To overcome overfitting, I relied on K-Fold Cross-Validation; by splitting the data into training and validation, I was able to see if the trained model has overfitting. If the model had overfitting, I simply ignored that trained model to test my results.

I trained different models using different hyperparameters, and in the end I calculated training and validation accuracy values. When the training accuracy was more than 90% and the validation accuracy was significantly lower (such as 40%), it was clear that the model was overfitted. I've read online that there are some actions such as drops can be taken to prevent this, however due to lack of time I wasn't able to test it properly.

5 Comments

This assignment was quite interesting and fun, however my computer does not have a GPU, and I had some problems with utilizing inek's for this assignment as well. In the end, I decided to rent a GPU instance through Linode and managed to train the models and test them properly. This was a challange, however seeing the major difference in time between using a CPU and GPU to train a network was a huge eye opener.