Report

Ilter Taha Aktolga
2236891

24.11.2020

1 Sanity Checks

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

1.1 Loss

I have used $log\_softmax$ with nll_loss which makes the same effect as $softmax$ with $cross_entropy$. as stated in the ODTUCLASS. Mathematical formula for Cross Entropy loss is :

$$H(p,q) = -\sum_{\forall x} p(x)\log(q(x)) \tag{1}$$

where p(x) is the ground truth and the q(x) is the estimated values. In the beginning, we have 100 classes and for each class we have equal probability which is 0.01. When we make prediction without training, we will give 0.01 to the correct class label which is q(x). But the ground truth is 1 for correct class which is p(x). Thus, by putting the values into equation we get 4.605 as a result. This must be the initial result. When I check on the code I've got values around 4.61 which verifies our calculation.

1.2 Accuracy

After the random initialization, since we have balanced data, we have equal distribution for probabilities. Each class can be predicted with equal probability. We have 100 classes. Therefore for each class, we can predict it correctly with 0.01 probability. Since prediction of one image doesn't effect other's prediction, we can generalize it to the batch or dataset that we calculated the accuracy on. Therefore we must get values around 0.01. When I checked accuracy on the code, I got 0.012 in one layer network which verifies our calculation.

AF&HS	Learning Rate									
	0.05	0.01	0.005	0.003	0.001	0.0005	0.0003	0.0001	0.00003	0.00005
-,-	0.077	0.072	0.081	0.090	0.1	0.109	0.116	0.117	0.118	0.117

Table 1: 1-layer network

2 Hyperparameter optimization

2.1 1-layer (0-hidden-layer) network

One layer network contains only input layer which is defined as nn.linear with input dimension 1*32*64 and output dimension 100. And activation function at the output layer is $log\_softmax$ function. As mentioned in "Announcements", since I've used $log\_softmax$, I have used nll_loss() as a loss function. I used Adam optimizer. Since there is no hidden layer and therefore activations for this layer, I have run the model with only different learning rates which are 0.05,0.01,0.003,0.001,0.0005,0.0003,0.0001,
0.00003 and 0.00005. Accuracy scores on validation set can be seen from the table below.

2.2 2-layer (1-hidden-layer) network

Two layer network contains input layer with input dimension 1*32*64 and output dimension is hidden size that I choose. All the layers are defined as nn.linear. There is one hidden layer in this network. Output dimension of this hidden layer is 100. For input dimension of the hidden layer, I have selected alternative hidden sizes as 256,512,1024. Activation function at the output layer is $log\_softmax$ function. As mentioned in "Announcements", since I've used $log\_softmax$, I have used nll_loss() as a loss function. I used Adam optimizer. I have decided to try learning rates which are 0.01, 0.003, 0.001, 0.0003, 0.0001, 0.00003. I have run model with every combination of the learning rate, hidden size and different activation functions namely sigmoid, tanh and relu. Accuracy scores on validation set can be seen from the table below.

Layer	Learning Rate					
Activations	0.01	0.003	0.001	0.0003	0.0001	0.00003
S, 256	0.009	0.106	0.148	0.161	0.17	0.132
S, 512	0.01	0.095	0.159	0.167	0.163	0.142
S, 1024	0.01	0.089	0.170	0.159	0.162	0.152
T, 256	0.01	0.051	0.117	0.155	0.150	0.150
T, 512	0.009	0.047	0.130	0.140	0.147	0.156
T, 1024	0.008	0.059	0.121	0.130	0.143	0.149
R, 256	0.01	0.046	0.156	0.154	0.155	0.147
R, 512	0.01	0.094	0.175	0.164	0.159	0.153
R, 1024	0.01	0.047	0.172	0.175	0.152	0.149

Table 2: 2-layer network

2.3 3-layer (2-hidden-layer) network

Three layer network contains input layer with input dimension 1*32*64 and output dimension is hidden size that I choose. All the layers are defined as nn.linear There are two hidden layers in this network. Input and output dimension of the first hidden layer is hidden size that I selected. Also, second hidden layer's input dimension is the my selected hidden size. Output dimension of the second hidden layer is 100. For hidden layer size, I have selected alternative values as 256,512,1024. Activation function at the output layer is $log\_softmax$ function. As mentioned in "Announcements", since I've used $log\_softmax$, I have used nll_loss() as a loss function. I used Adam optimizer. I have decided to try learning rates which are 0.01, 0.003, 0.001, 0.0003, 0.0001, 0.00003. I have run the model with every combination of the learning rate, hidden size and different activation functions namely sigmoid, tanh and relu. Accuracy scores on the validation set can be seen from the table below.

Layer Activations	Learning Rate					
	0.01	0.003	0.001	0.0003	0.0001	0.00003
S, 256	0.01	0.024	0.196	0.261	0.185	0.01
S, 512	0.014	0.0245	0.234	0.302	0.245	0.135
S, 1024	0.01	0.011	0.233	0.302	0.296	0.165
T, 256	0.01	0.009	0.163	0.20	0.237	0.177
T, 512	0.008	0.01	0.177	0.190	0.233	0.187
T, 1024	0.008	0.01	0.159	0.167	0.153	0.167
R, 256	0.01	0.103	0.2155	0.225	0.206	0.159
R, 512	0.01	0.066	0.206	0.25	0.233	0.186
R, 1024	0.01	0.040	0.21	0.262	0.266	0.223

Table 3: 3-layer network

3 The best hyperparameter

3.1 Results

I got best results on test dataset with 3-layer network with 0.0001 as a learning rate, sigmoid activation functions and 1024 hidden size for each layer. Training lasted 100 epochs without early stopping. Validation loss of the model is 2.714. And the validation accuracy is 0.2965. Score on the test set is 0.309500.The train and validation loss changes can be seen from the figure below. At the end of the 100 epoch, train loss was dropped to 1.048.

Note that 100 epochs was the threshold that I decided at the beginning because of my system requirements and to reduce long training period. (For early start algorithm please refer to **Section 3.2**)

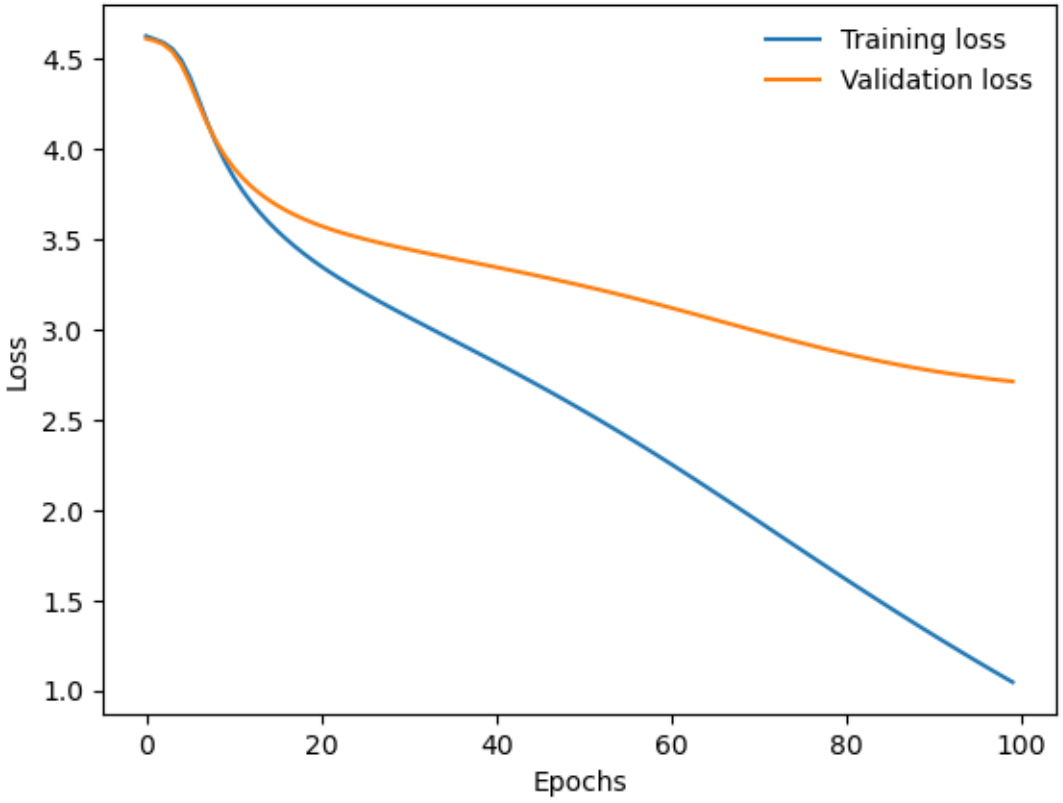

Figure 1: Best Validation Result (Loss vs Epochs)

3.2 Overfitting countermeasures

While the train loss is decreasing, if the validation loss is increasing, we can say that our model is overfitting to the training data. Also, I faced with scenarios that while the accuracy on validation set was not increasing but the train loss was dropping. As a countermeasure, I have implemented a "early-stopping" control to stop training with the best validation score with validation accuracy. In my algorithm, which can be seen in *train* function in **train.py** file, at each epoch, I was calculating the validation loss on the validation set. After at least 20 epochs, code starts to check last 20 validation loss with the current validation loss. If the current validation loss is at least lower than one of the past 20 validation losses, training process continues. Otherwise, training process ends with early stopping at that epoch. Due to system requirements of my computer, I have limited the maximum epochs with 100. So, by early stopping,

I have eliminated overfitting while satisfying the 0.3 accuracy on the test set.

4 Comments

This homework allowed me to learn Pytorch. I wrote and debugged neural networks which was a great hands-on experience. I have examined that finding good hyperparameters task is as challenging as the choosing a good neural network model.