Report

Ali Dogan

25/11/2020

1 Sanity Checks

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

1.1 Loss

The cross entropy loss function is as follows :

$$J = -\frac{1}{N}(\sum_{i=1}^{N} y_i log(\hat{y}_i))$$

The expectation of the resulting output label from a random initialized, untrained model is a vector containin 0.01 for each row. So, since we have a one-hot vector y_i, it means that the calculations will be as follows:

$$E(J) = -\frac{1}{N}(\sum_{i=1}^{N} E(y_i log(\hat{y}_i)))$$

$$E(J) = -\frac{1}{N}(\sum_{i=1}^{N} 1.0 * log(0.01))$$

$$E(J) = -\sum_{i=1}^{N} 1.0 * log(0.01)$$

log(0.01) = -4.60517018599, hence :

$$E(J) = 4.60$$

When we initiate the model without training, I get a very close loss. Here is the random losses I got from trials :

$$4.771172523498535, 4.770029067993164, 4.77113151550293$$

1.2 Accuracy

Before training, I ran couple of times and it gave 0.0112,0.0096, 0.0115 which are around 0.01. It is expected because there are 100 outputs and assuming the data we have is uniformly distributed, theoretical expectation is $\frac{1}{100}$

2 Hyperparameter optimization

2.1 The model architecture

Since I have used very similar networks for each n-layer implementation, I wanted to give a brief regarding my models basics and how I constructed it. All the fully connected networks and their reports I am mentioning below is the same module I have implemented, with different parameters.
NET(num-hidden-neurons-1 , num-hidden-neurons-2) module creates a model with given number of hidden neurons at first and second hidden layers. If both is given as 0 then it is 0-hidden layer and if layer 2 is given as 0 then it is 1-hidden layer network. The networks have the following design (FC: Fully connected layer, AF: activation function, BN: Batch Normalization):
1 - hidden layer :

$$FC- > BN- > AF- > Dropout(0.5)$$

2 - hidden layer :

$$FC- > BN- > AF- > Dropout(0.75)- > FC2- > BN- > AF- > Droput(0.5)$$

I tried different optimizers but I have concluded that Adam optimizer is much better. I did not spend time about the loss function and used CrossEntropy-Loss as stated in the homework document. I have splitted the given dataset to training and validation dataset (8000/2000 respectively). On the training phase, I used early stopping on validation. When the last three epoch was non-decreasing, it stopped the training. I have saved each hyperparameter optimization iterations over parameters along with their epoch number at which their training is done. I will attach the logs with the report. I have also plotted every trial but due to its size, I did not submit but here is the drive link to achieve them : **Plots**

2.2 1-layer (0-hidden-layer) network

The architecture is straightforward one layer network.I wanted to add training accuracies as well. The early stopping was causing the network learn nothing. So I did not use early stopping for 0-hidden layer.

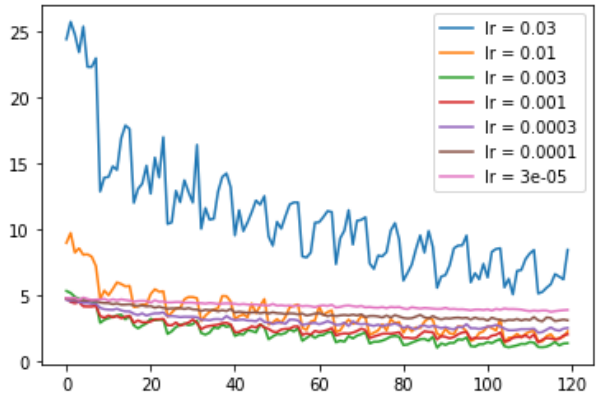

Figure 1: Training losses of 0-hidden network

	Learning Rate					
	0.03	0.01	0.003	0.001	0.0003	0.0001
train / validation	67/9.4	77/9.35	88/10.44	76/11.09	52/11.14	35.31/10.74

Table 1: 1-layer network

2.3 2-layer (1-hidden-layer) network

I have used one fully connected hidden layer with batch normalization and dropout regularizations as I mentioned above (FC: Fully connected layer, AF: activation function, BN: Batch Normalization):

$$FC- > BN- > AF- > Dropout(0.5)$$

I set the epoch number = 150 with an early stopping which is triggered when the last 3 epochs do not cause any decrease in the validation set. Most of the runs stopped with early stopping. I have tried the following parameters:

Learning rate : Learning rate : [0.03 , 0.01 , 0.003 , 0.001 , 0.0003 , 0.0001]
Hidden layer neuron : [128,256,512,1024]
Activation Functions in the hidden layer : [Relu,Sigmoid,Tanh]

Thus, I have tried 72 different configurations for 1-hidden layer network. I Saved all of the models training-validation loss plots and accuracies. I will add the plots and logs with the report as well. I will add the best resulting plot here.

Layer Activations	Learning Rate					
	0.03	0.01	0.003	0.001	0.0003	0.0001
S, 128	**0.2334**	0.2099	0.1944	0.1714	0.1584	0.1279
S, 256	0.2139	0.1679	0.1744	0.1789	0.1679	0.1554
S, 512	0.1894	0.1649	0.1664	0.1639	0.1519	0.1579
S, 1024	0.1879	0.1449	0.1539	0.1649	0.1564	0.1779
T, 128	0.1494	0.1634	0.1649	0.1454	**0.1679**	0.1544
T, 256	0.1389	0.1499	0.1419	0.1509	0.1544	0.1354
T, 512	0.1359	0.1264	0.1404	0.1614	0.1419	0.1449
T, 1024	0.1264	0.1329	0.1689	0.1389	0.1459	0.1349
R, 128	**0.2424**	0.1789	0.1799	0.1494	0.2374	0.2419
R, 256	0.2139	0.1824	0.2154	0.2334	0.2009	0.2029
R, 512	0.2029	0.1874	0.2049	0.2229	0.1634	0.1789
R, 1024	0.1839	0.1854	0.1609	0.1694	0.1599	0.1694

Table 2: 2-layer network

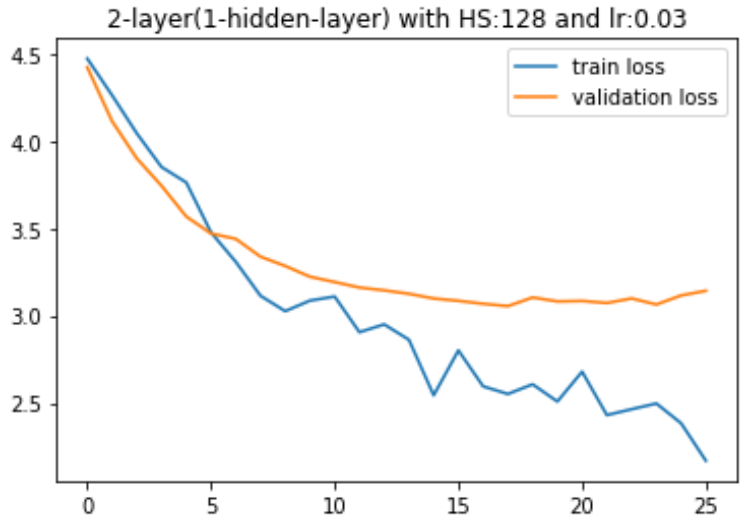

Figure 2: Best resulting model among two layers, with 0.2424 accuracy (ReLu)

2.4 3-layer (2-hidden-layer) network

for the 2-hidden layer network, I used the same epoch number and early stop policy. I have tried the following hyper parameters:

Learning rate : [0.03 , 0.01 , 0.003 , 0.001 , 0.0003 , 0.0001]
First Hidden layer neuron : [256,512,1024]
Second Hidden layer neuron : [128,256,512,1024]
Activation Functions in the hidden layer : [Relu,Sigmoid,Tanh]

So, I tried total of 216 different configurations of a 2-hidden layer architecture. I save the plots and logs as I did with 1-layer, but this time for the sake of simplicity, I will not fill the table below completely. I highlighted the best results among each activation function.

Layer	Learning Rate					
Activations	0.03	0.01	0.003	0.001	0.0003	0.0001
S, 256-128	0.3023	...	...	...	...	0.1114
S, 256-256	0.3183	...	...	...	...	0.1104
S, ... - ...	...	...	...	...	...	...
S, 512-256	0.3978	0.3743	...	...	0.1674	0.1379
S, ...	...	...	...	...	...	...
S, 1024-128	**0.4003**	...	...	...	...	...
S, ...	...	...	...	...	...	...
S, 1024-1024	0.3638	...	...	...	...	0.1689
T, 256-128	0.1734	...	...	**0.2189**	...	0.1769
T, 256-256	0.1544	...	...	...	...	0.1784
T, ...	...	...	...	...	...	...
T, 1024-1024	0.1389	...	...	...	...	0.1779
R, 256-128	0.2194	...	...	...	...	0.2199
R, 256-512	0.1879	...	...	...	...	0.2489
R, ...	...	...	...	...	...	...
R, 512-256	0.2474	0.4378	0.3723	0.4598	0.4208	0.3098
R, ...	...	...	...	...	...	...
R, 1024,128	...	...	...	**0.4638**	...	...
R, ...	...	...	...	...	...	...
R, 1024-1024	0.4033	...	...	...	...	0.3168

Table 3: 3-layer network

The 2-hidden layer mdoel with relu was very promising and I got the best result with 1024-128 architecture with 0.46 accuracy on the validation test which takes 79 epochs.

3 The best hyperparameter

3.1 Results

The network that achieved the best validation accuracy was a 3-layer(2-hidden layer) network with the following hyper parameters:

learning rate : 0.001
first hidden layer size : 1024
second hidden layer size : 128
activation function : ReLu
test accuracy : 0.485100

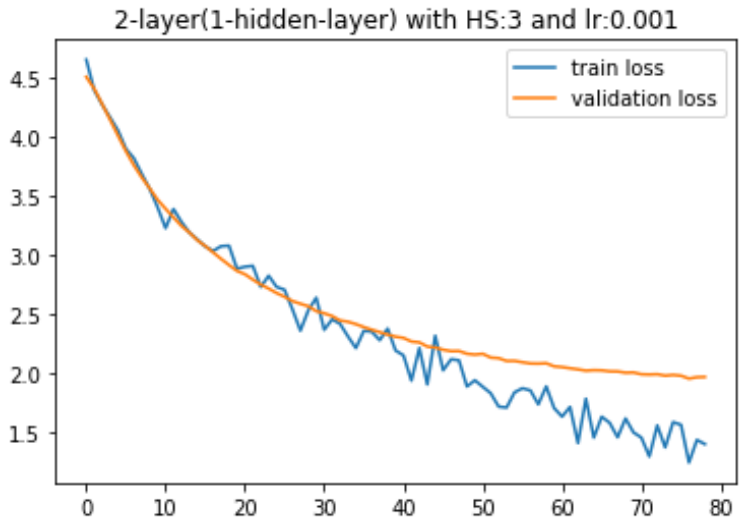

Figure 3: Best resulting model (HS : 1024-128) with 0.4638 accuracy on validation set (ReLu)

3.2 CNN Results

After a lot of trials on the fully connected network models and only getting around 0.45 accuracy, I decided to implement a Convolutional neural network. I have not spend so much time but it gave satisfying results. I used two convolutional layers and two fully connected layers. I used the same number of hidden neurons (1024-128) in the fully connected layers as in the fully connected model which gives the best result. I got the following accuracies from one of grid search.
training accuracy : 0.99
validation accuracy : 0.8916
test accuracy : 0.9059

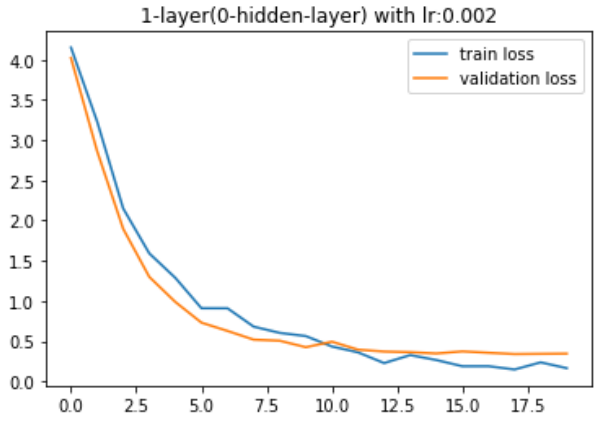

Figure 4: CNN model loss plot (ignore the title)

3.3 Overfitting countermeasures

I used dropout and batch normalization regularizations. I also used early stopping which was triggered for almost every hyperparameter optimizing iterations. In the trainings, my fully connected network models can be considered overfitted. Even the best model I had has a 0.46 accuracy on the validation set whereas the training accuracy is around 0.98. The numbers were way worse without dropout and batch normalizations.
At first, my networks were highly overfitted. In addition to the mistakes I was making (forgeting model.eval() during validation check, wrong trasformation values to name a few), I was not using any regularizations. After solving the problems, I was able to get solutions around 0.2-0.3 accuracies before regularizations. With ragularizations, my model improved significantly.
Understanding whether your model is overfitted or not can be understood by looking at the training-validation loss plots. If at some point, your training loss is decreasing whereas your validation loss is not, then it means any progress after that point will make the model overfitted to the seen data. For example, the plot below is from one of my first models. Unfortunately I do not remember its configurations such as dropout, batch norm regularizations, batch size etc.. but it can be seen that validation loss starts to increase at some point.

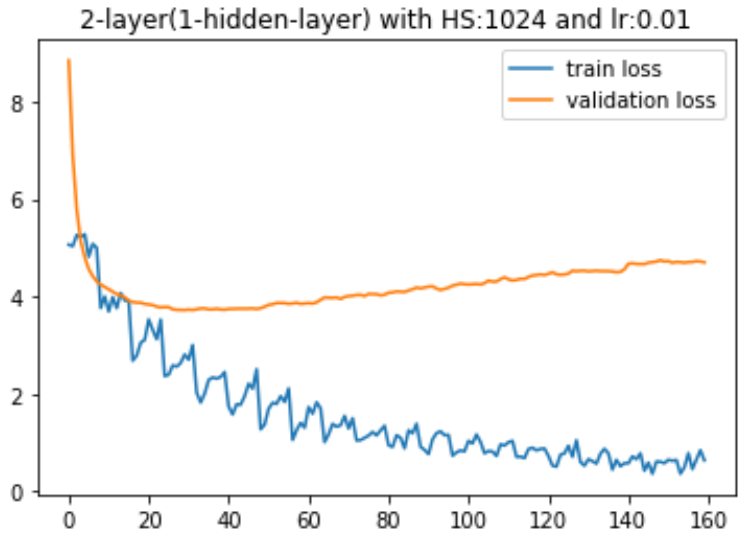

Figure 5: Overfitting example

4 Comments

It was a well-prepared end-to-end ML project to begin learning the fundamental concepts and practice the knowledge we had from classes. I Found the pytorch videos very helpfull. I thought getting 0.3 accuracy was very low and I would get much better. But with only fully connected layers, I learned that it was not so easy. Still it was nice to try and I got good results from CNN so I think it was the lack of power of fully connected layers but not me totally.