Ceng499 - THE1 Report

Seda Civelek - 2237147

April 2021

1 Sanity Checks

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

1.1 Loss

Since we used cross entropy loss, we can calculate loss using the probability of the true class. Since there are 10 classes, the probability of the true class is **0.1** . Now we can calculate loss in the following way:

$$-\log(0.1) = 2.30 \tag{1}$$

Note that we take minus natural logarithm of the probability of true class. However, in my model, after the random initialization of weights, the result was **2.14** . It is close to the value that we theoretically calculated. We can say that, initial weights are better initialized than expected.

1.2 Accuracy

Since there are 10 classes and the dataset is balanced, we can say that the accuracy can be calculated in the following way:

$$\frac{1}{10} = 0.1 \tag{2}$$

However, in my model, after the random initialization of weights, the result was **0.18** . It is close to the value that we theoretically calculated. We can say that, initial weights are better initialized than expected.

2 Separating Validation Set

I separated train split into train and validation sets using

```
torch.util.data.random_split(trainSet,[24000,6000])
```

Since there are 30000 images in the train set, I divided 80 percent of it for train and 20 percent of it for validation.

3 Hyperparameter optimization

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

I used validation set to calculate accuracy rate and loss. I used 0.01, 0.003, 0.001, 0.0003, 0.0001 and 0.00003 as learning rate values. I added two tables that one represents the accuracy rate and the other represents the loss for different hyperparameter values.

Accuracy rate for different hyperparameters:

Hidden	Learning Rate					
Layer Size	0.01	0.003	0.001	0.0003	0.0001	0.00003
none	0.3073	0.3200	0.3568	0.3660	0.3715	0.3561

Table 1: 1-layer network accuracy rate

Loss for different hyperparameters:

Hidden	Learning Rate					
Layer Size	0.01	0.003	0.001	0.0003	0.0001	0.00003
none	1.9339	1.5927	1.5461	1.3822	1.4597	1.6813

Table 2: 1-layer network losses

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

I used 256, 512 and 1024 as layer size values and 0.01, 0.003, 0.001, 0.0003, 0.0001 and 0.00003 as learning rate values. In hidden layer, I tried 3 different activation functions that are sigmoid, tanh and ReLU. I added two tables that one represents the accuracy rate and the other represents the loss for different hyperparameter values.

Accuracy rate for different hyperparameters:

Layer	Learning Rate					
Activations	0.01	0.003	0.001	0.0003	0.0001	0.00003
S, 256	0.1078	0.1878	0.4870	0.5513	0.4291	0.3850
S, 512	0.095	0.0945	0.4898	0.5483	0.4218	0.3798
S, 1024	0.0970	0.1005	0.5203	0.5390	0.4923	0.3938
T, 256	0.1078	0.0990	0.0945	0.5515	0.5166	0.3793
T, 512	0.1031	0.0945	0.1810	0.5176	0.5123	0.4100
T, 1024	0.0950	0.0965	0.2691	0.5448	0.5315	0.4320
R, 256	0.0965	0.3750	0.4986	0.5185	0.4871	0.4150
R, 512	0.1038	0.4298	0.5346	0.5351	0.5103	0.4588
R, 1024	0.0965	0.4050	0.5395	0.5266	0.5165	0.4846

Table 3: 2-layer network accuracy rate

Loss for different hyperparameters:

Layer	Learning Rate					
Activations	0.01	0.003	0.001	0.0003	0.0001	0.00003
S, 256	2.2730	1.7596	1.0411	0.9757	1.3578	1.5329
S, 512	2.2338	2.2791	1.0311	0.9420	1.2856	1.4969
S, 1024	2.3345	2.2548	1.0719	1.0002	1.2116	1.4004
T, 256	2.3097	2.2518	2.2689	0.9902	1.0710	1.4088
T, 512	2.2670	2.2636	1.8580	0.9539	1.1551	1.4173
T, 1024	2.3505	2.3142	1.6126	0.8443	1.0092	1.4184
R, 256	2.2926	1.2296	1.0802	1.1245	1.2259	1.3701
R, 512	2.2933	1.2395	0.9037	0.9549	1.0638	1.2485
R, 1024	2.2931	1.2595	0.8934	0.9582	0.9847	1.2205

Table 4: 2-layer network loss

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

I used 256, 512 and 1024 as layer size values and 0.01, 0.003, 0.001, 0.0003, 0.0001 and 0.00003 as learning rate values. In two hidden layers, I tried 3 different activation functions that are sigmoid, tanh and ReLU. I added two tables that one represents the accuracy rate and the other represents the loss for different hyperparameter values.

Accuracy rate for different hyperparameters:

Layer	Learning Rate					
Activations	0.01	0.003	0.001	0.0003	0.0001	0.00003
S, 256	0.0965	0.1026	0.4535	0.5385	0.4425	0.3551
S, 512	0.0945	0.0970	0.4738	0.5336	0.5058	0.3871
S, 1024	0.1038	0.1078	0.3036	0.5333	0.5076	0.3990
T, 256	0.0950	0.1026	0.2843	0.5406	0.5371	0.4776
T, 512	0.1031	0.0950	0.1038	0.5416	0.5375	0.4873
T, 1024	0.1078	0.1005	0.1978	0.5433	0.5451	0.4926
R, 256	0.1026	0.3895	0.5421	0.5385	0.5193	0.4725
R, 512	0.0945	0.1880	0.5440	0.5825	0.5596	0.5408
R, 1024	0.0950	0.2195	0.5331	0.5626	0.5513	0.5313

Table 5: 3-layer network accuracy rate

Loss for different hyperparameters:

Layer	Learning Rate					
Activations	0.01	0.003	0.001	0.0003	0.0001	0.00003
S, 256	2.2927	2.2968	1.1533	0.7961	1.1987	1.5122
S, 512	2.2790	2.2975	1.0602	0.8504	1.1020	1.4099
S, 1024	2.2534	2.2447	1.5626	0.9043	0.9557	1.3644
T, 256	2.2817	2.2823	1.5310	1.0079	0.9577	1.1762
T, 512	2.3569	2.2295	2.2495	0.9363	0.9662	1.1395
T, 1024	2.3092	2.2829	1.8102	0.9456	0.8259	0.9834
R, 256	2.2904	1.3089	0.8375	0.7961	0.9018	1.1295
R, 512	2.2824	1.8103	0.7324	0.7491	0.9213	1.0380
R, 1024	2.2930	1.7371	0.8802	0.8754	0.9155	0.9010

Table 6: 3-layer network loss

4 The best hyperparameter

4.1 Results

From the scores we get doing grid search, we can say that the best validation loss and best accuracy rate are obtained by the hyperparameters that are **2-hidden layers**, **512** layer size, **ReLU** function and **0.001** learning rate. Using this network, the test accuracy is calculated as **0.56**. You can see the training and validation loss graph below.

Since using larger epoch values requires too much time, I decided to use 50 epochs for training. So, in the best validation and training loss graph, training is not early stopped since the decrease in losses continues.

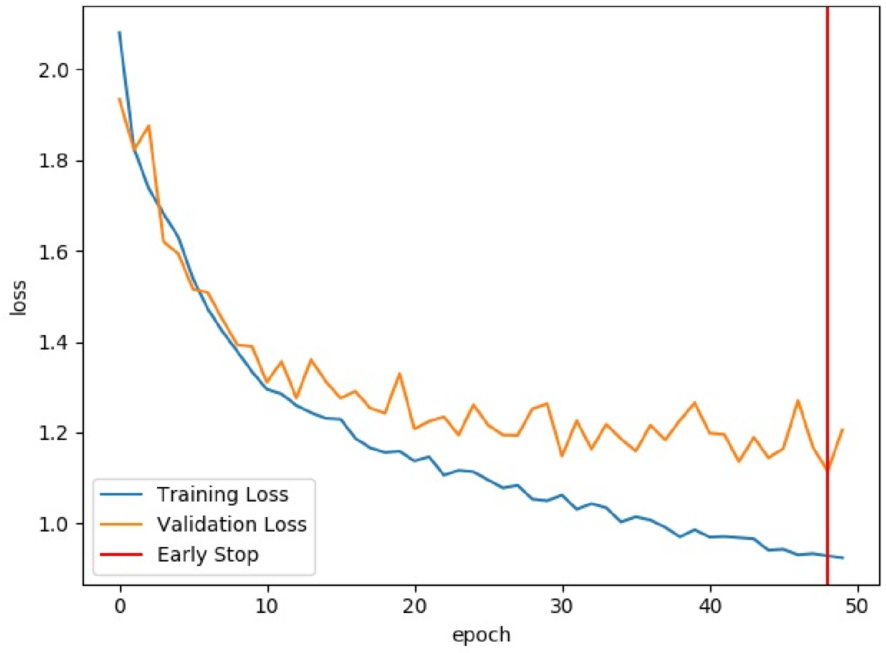

Figure 1: Training and Validation Loss

4.2 Overfitting countermeasures

To avoid overfitting, I implemented an early stopping mechanism that works like the following way:

```
best_loss = infinity
for each epoch:
```

```
loss <- calculate loss
if loss < best_loss:
    best_loss = loss
```

We can say that if loss is not decreasing anymore, network starts to overfit during training. Hence, we can find the minimum loss value and save the model in the minimum point and we can solve the overfitting problem.

5 Comments

At the beginning, I tried 100 epochs to train and its result gave me **0.586** accuracy in test. However, due to limited time, I decreased the epoch value to 50 to test all hyperparameters.