Report

Abdulkadir Pamukçu

25.04.2021

1 Sanity Checks

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

torch.Size([32, 10]) tensor([0.0762, 0.1666, 0.0979, 0.1051, 0.1148, 0.0663, 0.0922, 0.0705, 0.1062, 0.1043], $grad_fn =< SelectBackward >$) tensor(1.0000, $grad_fn =< SumBackward0 >$)

1.1 Loss

What do you expect the loss to be initially? Calculate it and write your calculations and results here. After the random initialization of the weights do not train the model and calculate the loss. Is it similar to what you expected?

I expect loss as something high since model initially does not know about the data. It was close to my expectation. It was 5.53

1.2 Accuracy

What do you expect the accuracy to be initially? Calculate it and write your calculations and results here. After the random initialization of the weights do not train the model and calculate the accuracy on one of the splits and report it. Is it similar to what you expected?

I expected something about 0.1 since there were 10 outputs and a model does not know about data gives randomly. So 1/10 = 0.1

After initialization of the weights the accuracy were something about 0.09. It was similar.

2 Separating Validation Set

I splited the validation dataset to 20 percent of the dataset. According to my experiments it seemed like the best choice. Briefly mention how you separated train split into train and validation sets.

3 Hyperparameter optimization

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

Explain your network architecture and the hyperparameters you tested. Create a table for the scores for each hyperparameter setting you tried. **To calculate these scores you cannot use test set.** An example table is below AF: activation function, HS: hidden layer size (learning rates and hidden layer sizes are just placeholders, you can write the learning rates you tried):

In 1-layer artificial neural network, I used torch.nn to create the layer with the logsoftmax to do both softmax and cross entropy loss. Since it has 0 hidden layer there were no need to use a activation function. The dataset that we're given is already splited to data and test sets. I also split data set to validation and training sets with proportions of 0.8, 0.2. The batch size were 64, after some experiments i decided 64 is way to go. I chose number of epochs were 20 since time was limited and it was taking lots of time with the limited gpu that i have access to.

AF and HS	Learning Rate					
	0.01	0.003	0.001	0.0003	0.0001	0.00003
-, -	27.19	32.27	33.59	34.49	33.64	29.26

Table 1: 1-layer network

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

I build the model same as the 1 layer model as I explained earlier, but this had a one hidden layer, so I created another layer with sizes of [256,512,1024] and applied same logsoftmax too. Also applied activation function to the outputs of this layer. I used S:sigmoid, T:tanh, R:ReLU as activation functions. The hyper parameters of this similar to the before:
Epoch:20
Batch size: 64
Valid dataset: 0.2 * Data
Train dataset: 0.8 * Data

Layer	Learning Rate					
Activations	0.01	0.003	0.001	0.0003	0.0001	0.00003
S, 256	9.94	10.41	15.0	50.0	44.6	36.52
S, 512	10.13	9.86	22.0	29.0	43.49	val
S, 1024	9.79	10.53	9.0	52.0	45.95	val
T, 256	10.13	9.91	25.0	49.0	43.70	37.37
T, 512	10.11	10.53	15.0	49.0	44.05	val
T, 1024	9.86	10.41	26.0	49.0	47.30	val
R, 256	10.13	19.38	39.0	39.0	39.72	37.31
R, 512	10.11	10.13	47.0	48.0	41.14	val
R, 1024	10.13	20.01	51.0	48.0	45.10	val

Table 2: 2-layer network

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

Explain your network architecture and the hyperparameters you tested. Create a table for the scores for each hyperparameter setting you tried. **To calculate these scores you cannot use test set.** An example table is below, S:sigmoid, T:tanh, R:ReLU. Here, hidden sizes are specified to be the same. However, you can try different hidden sizes for different levels. (Learning rates are just placeholders, you can write the learning rates you tried)

This was also very similar to previous architectures, this one has two hidden layers both layer sizes differ with 256,512,1024. Both hidden layer's outputs given to an activation function before given to next layer as an input.

Epoch:20 (I also tried 40 for the best model).
Batch size: 64
Valid dataset: 0.2 * Data
Train dataset: 0.8 * Data

Layer	Learning Rate					
Activations	0.01	0.003	0.001	0.0003	0.0001	0.00003
S, 256	9.85	9.94	18.0	51.0	49.22	val
S, 512	9.14	10.13	10.0	55.0	51.09	val
S, 1024	9.79	10.41	16.0	53	50.15	val
T, 256	9.86	10.41	17.0	52.0	49.12	39.77
T, 512	10.11	9.14	18.0	50.0	51.39	val
T, 1024	10.13	9.94	16.0	52	49.12	val
R, 256	9.79	10.13	49.0	50.0	46.8	39.19
R, 512	9.86	9.14	49.0	54.0	52.54	val
R, 1024	10.13	9.86	53.0	52	55.67	val

Table 3: 3-layer network

4 The best hyperparameter

4.1 Results

Give the hyperparameter setting of the network that achieved the best validation score in hyperparameter optimization. Calculate its test accuracy and report it.

Epoch:40
Batch size: 64
2 hidden layer
Valid dataset: 0.2 * Data
Train dataset: 0.8 * Data
Hidden layer size: 1024
Activation function: RelU
Learning rate: 0.001

.

As can seen from the plot validation loss does not drop about 20-25 epochs, so training stops itself to prevent overfitting.

4.2 Overfitting countermeasures

In order to prevent overfitting I keep track of my validation loss through training, if validation loss does not decrease less than min_delta(with experiments i decided to be 0.01) in one epoch, i incremented a counter, if it starts to decrease again, i decremented the counter with keeping it above zero. If counter reaches 5, this is the signal of overfitting in my training since the validation loss is stagnating or maybe increasing so my model is not improving itself even deteriorating, so when this signal is received by training function it stops the training process and saves the model.

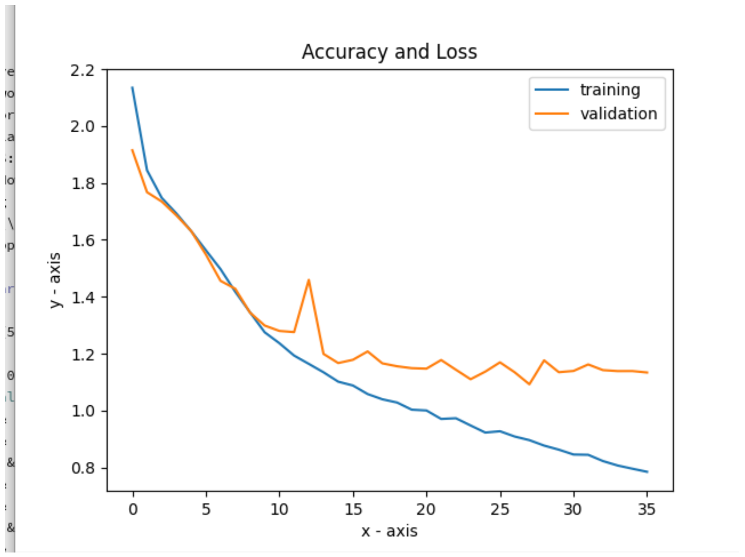

Figure 1: Training and validation losses

5 Comments

This section can include your additional comments. You can mention additional things you tried. It can be left empty.