CENG 499

Special Topics: Introduction to Machine Learning

HOMEWORK 1 REPORT

ARDAN YILMAZ

2172195

Table of Contents

1. Sanity check

Sanity check on accuracy:

A random classifier would label an instance with the correct class with a probability of 1/10, as there are 10 classes (with an almost balanced number of instances). That's why I would expect that a trained model outperforms its random classifier.

The number of TP, FP, FN, and TN values can be calculated for a random classifier more precisely, and hence also its accuracy. And, one can apply Fisher's exact test to compare the predictive performance results of the random classifier with the trained model. However, it is not necessary to do it here, as one can safely say that the random classifier labels an instance with the true class with a probability of 1/10. And, this is basically the threshold a trained model's predictive performance needs to pass.

It is visible that all models whose hyperparameter configuration and predictive performance results are given below outperform their random classifier.

Sanity check on loss:

Plugging the probability value discussed above into the cross-entropy loss formula, it can be yielded that the value is log(10).

That is, the expected value of the cross-entropy loss for a random classifier is around log(10) = 1.

2. Grid Search

a. Grid Search on 1-Layered Network

Batch Size	Learning Rate	Optimizer	Epochs	Validation Loss	Validation Accuracy
64	0.01	SGD	25	2.1177	25.200%
64	0.01	SGD	50	2.1358	27.230%
64	0.01	SGD	75	2.1515	25.350%
64	0.01	SGD	100	2.1496	25.840%
64	0.01	Adam	25	2.5992	22.040%
64	0.01	Adam	50	2.4966	22.000%
64	0.01	Adam	75	2.4996	22.910%
64	0.01	Adam	100	2.5647	21.390%
64	0.001	SGD	25	2.0152	29.840%
64	0.001	SGD	50	2.0241	29.420%
64	0.001	SGD	75	2.0283	28.940%
64	0.001	SGD	100	2.0379	28.380%
64	0.001	Adam	25	2.0802	26.850%
64	0.001	Adam	50	2.0810	27.030%
64	0.001	Adam	75	2.1046	26.400%
64	0.001	Adam	100	2.1018	26.820%
64	0.0005	Adam	25	2.0115	29.940%
64	0.0005	Adam	50	2.0141	29.760%
64	0.0005	Adam	75	2.0190	29.390%
64	0.0005	Adam	100	2.0253	29.530%
64	0.0005	SGD	25	2.0388	28.430%

64	0.0005	SGD	50	2.0510	27.610%
64	0.0005	SGD	75	2.0595	27.480%
64	0.0005	SGD	100	2.0672	27.440%
32	0.01	SGD	25	2.3026	21.440%
32	0.01	SGD	50	2.2353	23.050%
32	0.01	SGD	75	2.2087	25.280%
32	0.01	SGD	100	2.2128	24.760%
32	0.01	Adam	25	2.7087	21.920%
32	0.01	Adam	50	2.8154	19.610%
32	0.01	Adam	75	2.7853	21.390%
32	0.01	Adam	100	3.0692	20.070%
32	0.001	SGD	25	2.0330	28.460%
32	0.001	SGD	50	2.0410	28.150%
32	0.001	SGD	75	2.0455	28.570%
32	0.001	SGD	100	2.0535	28.180%
32	0.001	Adam	25	2.0997	27.010%
32	0.001	Adam	50	2.1136	25.800%
32	0.001	Adam	75	2.1101	26.280%
32	0.001	Adam	100	2.1238	26.410%
32	0.0005	Adam	25	2.0155	30.000%
32	0.0005	Adam	50	2.0231	29.330%
32	0.0005	Adam	75	2.0320	28.630%
32	0.0005	Adam	100	2.0354	28.600%
32	0.0005	SGD	25	2.0538	28.000%
32	0.0005	SGD	50	2.0691	27.730%
32	0.0005	SGD	75	2.0752	27.090%

32	0.0005	SGD	100	2.0777	27.120%

b. Grid Search on 2-Layered Network

On the first row on the following table, their abbreviations of the hyperparameters tested, which stand for:
lr: learning rate, Num neurons: number of neurons, Act Funct: activation function, dropout: drop out rate, and epoch.
Adam is used as the optimizer, and batch size = 64,is used for all models whose metrics are shown below.

lr	Num neurons	Act Funct	dropout	Epoch	Validation Loss	Validation Accuracy
0.001	512	ReLU	0.2	25	2.0309	29.420%
0.001	512	ReLU	0.2	50	2.0389	29.050%
0.001	512	ReLU	0.2	75	2.0525	28.780%
0.001	512	ReLU	0.2	100	2.0600	28.060%
0.001	512	ReLU	0.5	25	2.0202	30.010%
0.001	512	ReLU	0.5	50	2.0268	29.610%
0.001	512	ReLU	0.5	75	2.0362	29.520%
0.001	512	ReLU	0.5	100	2.0456	28.670%
0.001	512	hardswish	0.2	25	2.0309	29.420%
0.001	512	hardswish	0.2	50	2.0389	29.050%
0.001	512	hardswish	0.2	75	2.0525	28.780%
0.001	512	hardswish	0.2	100	2.0600	28.060%
0.001	512	hardswish	0.5	25	2.0202	30.010%
0.001	512	hardswish	0.5	50	2.0268	29.610%
0.001	512	hardswish	0.5	75	2.0362	29.520%
0.001	512	hardswish	0.5	100	2.0456	28.670%

0.001	256	ReLU	0.2	25	2.0335	28.780%
0.001	256	ReLU	0.2	50	2.0391	28.600%
0.001	256	ReLU	0.2	75	2.0570	29.070%
0.001	256	ReLU	0.5	100	2.0547	28.130%
0.001	256	ReLU	0.5	25	2.0221	29.360%
0.001	256	ReLU	0.5	50	2.0256	29.370%
0.001	256	ReLU	0.5	75	2.0397	29.300%
0.001	256	ReLU	0.5	100	2.0396	28.590%
0.001	256	hardswish	0.2	25	2.0335	28.780%
0.001	256	hardswish	0.2	50	2.0391	28.600%
0.001	256	hardswish	0.2	75	2.0570	29.070%
0.001	256	hardswish	0.2	100	2.0547	28.130%
0.0005	512	ReLU	0.2	25	2.0404	29.290%
0.0005	512	ReLU	0.2	50	2.0428	28.910%
0.0005	512	ReLU	0.2	75	2.0525	28.630%
0.0005	512	ReLU	0.2	100	2.0603	28.090%
0.0005	512	hardswish	0.2	25	2.0404	29.290%
0.0005	512	hardswish	0.2	50	2.0428	28.910%
0.0005	512	hardswish	0.2	75	2.0525	28.630%
0.0005	512	hardswish	0.2	100	2.0603	28.090%
0.0005	256	ReLU	0.2	25	2.0353	28.630%
0.0005	256	ReLU	0.2	50	2.0420	28.400%
0.0005	256	ReLU	0.2	75	2.0582	28.760%
0.0005	256	ReLU	0.2	100	2.0554	28.070%

c. Grid Search on 3-Layered Network

On the first row on the following table, their abbreviations of the hyperparameters tested, which stand for:
lr: learning rate, #n1: number of neurons in the first layer, #n2: number of neurons in the second layer, Actv f1: activation function on the first layer, Actv f2: activation function on the second layer, dropout: drop-out rate, and epoch.
Adam is used as the optimizer for all models whose metrics are shown below.

lr	#n1	#n2	Actv f1	Actv f2	drop out	epoc h	Val loss	Val Acc
0.001	512	256	tanh	tanh	0	25	2.2263	43.540%
0.001	512	256	tanh	tanh	0	50	3.4150	41.100%
0.001	512	256	tanh	tanh	0	75	3.8591	42.420%
0.001	512	256	tanh	tanh	0	100	4.1510	42.210%
0.001	512	256	tanh	tanh	0.2	25	1.5756	45.180%
0.001	512	256	tanh	tanh	0.2	50	1.7321	45.600%
0.001	512	256	tanh	tanh	0.2	75	1.8976	45.200%
0.001	512	256	tanh	tanh	0.2	100	2.0424	43.950%
0.001	512	256	tanh	tanh	0.5	25	1.6139	42.550%
0.001	512	256	tanh	tanh	0.5	50	1.5682	43.820%
0.001	512	256	tanh	tanh	0.5	75	1.5725	44.560%
0.001	512	256	tanh	tanh	0.5	100	1.5725	44.560%
0.001	512	256	tanh	ReLU	0	25	2.0557	41.160%
0.001	512	256	tanh	ReLU	0	50	2.9660	39.060%
0.001	512	256	tanh	ReLU	0	75	3.5517	38.870%
0.001	512	256	tanh	ReLU	0	100	3.8430	38.340%
0.001	512	256	tanh	ReLU	0.2	25	1.5983	43.660%
0.001	512	256	tanh	ReLU	0.2	50	1.6882	43.500%

0.001	512	256	tanh	ReLU	0.2	75	1.8348	42.330%
0.001	512	256	tanh	ReLU	0.2	100	1.9257	42.160%
0.001	512	256	tanh	ReLU	0.5	25	1.6223	41.980%
0.001	512	256	tanh	ReLU	0.5	50	1.5682	43.970%
0.001	512	256	tanh	ReLU	0.5	75	1.5690	44.040%
0.001	512	256	tanh	ReLU	0.5	100	1.5825	43.990%
0.001	512	256	tanh	hardswish	0	25	2.0709	39.980%
0.001	512	256	tanh	hardswish	0	50	2.9173	39.850%
0.001	512	256	tanh	hardswish	0	75	3.4364	39.640%
0.001	512	256	tanh	hardswish	0	100	3.9128	39.250%
0.001	512	256	tanh	hardswish	0.2	25	1.5845	44.290%
0.001	512	256	tanh	hardswish	0.2	50	1.7263	43.300%
0.001	512	256	tanh	hardswish	0.2	75	1.8506	42.720%
0.001	512	256	tanh	hardswish	0.2	100	1.9438	41.500%
0.001	512	256	tanh	hardswish	0.5	25	1.6269	42.270%
0.001	512	256	tanh	hardswish	0.5	50	1.5757	43.510%
0.001	512	256	tanh	hardswish	0.5	75	1.5700	44.340%
0.001	512	256	tanh	hardswish	0.5	100	1.5673	44.110%
0.001	512	256	ReLU	tanh	0.2	25	1.6362	42.370%
0.001	512	256	ReLU	tanh	0.2	50	1.6058	42.890%
0.001	512	256	ReLU	tanh	0.2	75	1.6149	42.230%
0.001	512	256	tanh	tanh	0.2	100	1.6460	43.280%
0.001	512	128	tanh	tanh	0.2	25	1.5682	44.990%
0.001	512	128	tanh	tanh	0.2	50	1.6973	45.350%
0.001	512	128	tanh	tanh	0.2	75	1.8263	44.290%

0.001	**512**	**128**	**tanh**	**tanh**	**0.2**	**100**	**1.9600**	**43.750%**

3. Best Models' Evaluation and Discussion

The hyperparameter configuration with the best performing validation accuracy for each k-layered network is highlighted in the corresponding table.

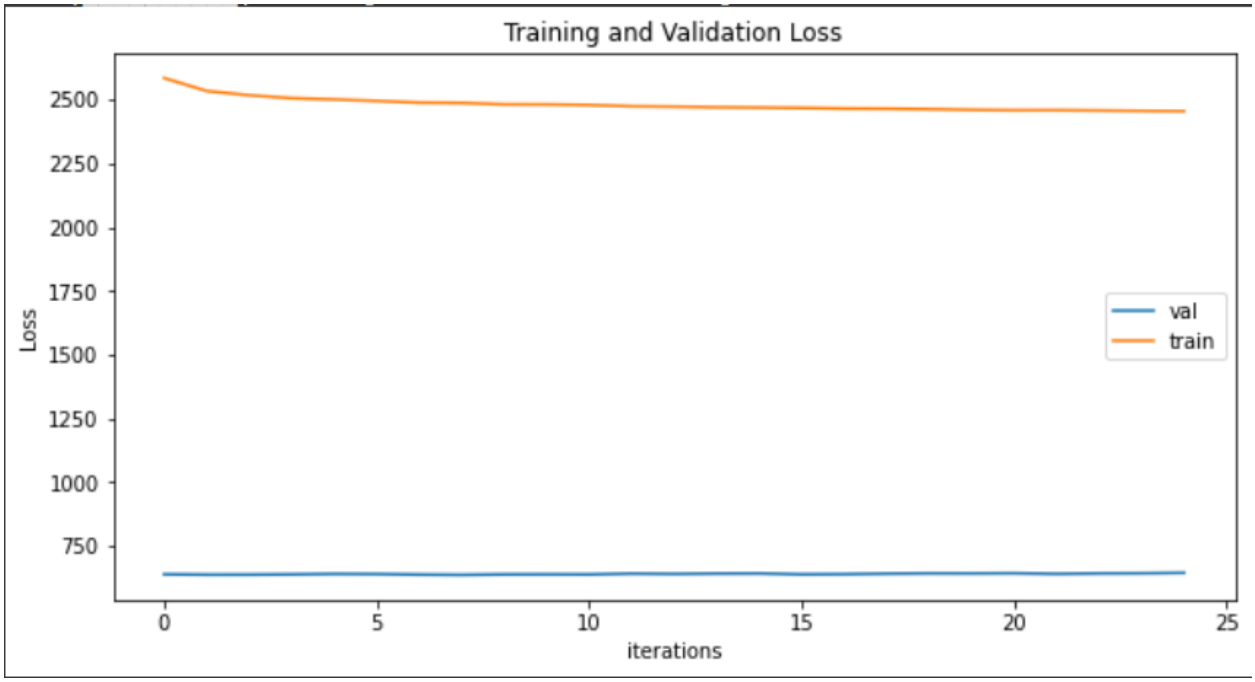
3.1. Best configuration for 1-Layered Network

The following is the hyperparameter configuration to produce the highest accuracy value and the performance results for a 1-layered network.

batch size : 32,
Learning rate: 0.0005,
Optimizer: Adam,
Epoch: 25
Validation accuracy: 30.000%
Validation loss: 2.0155

Test accuracy: 28.770%

The following graph shows the validation and loss errors:



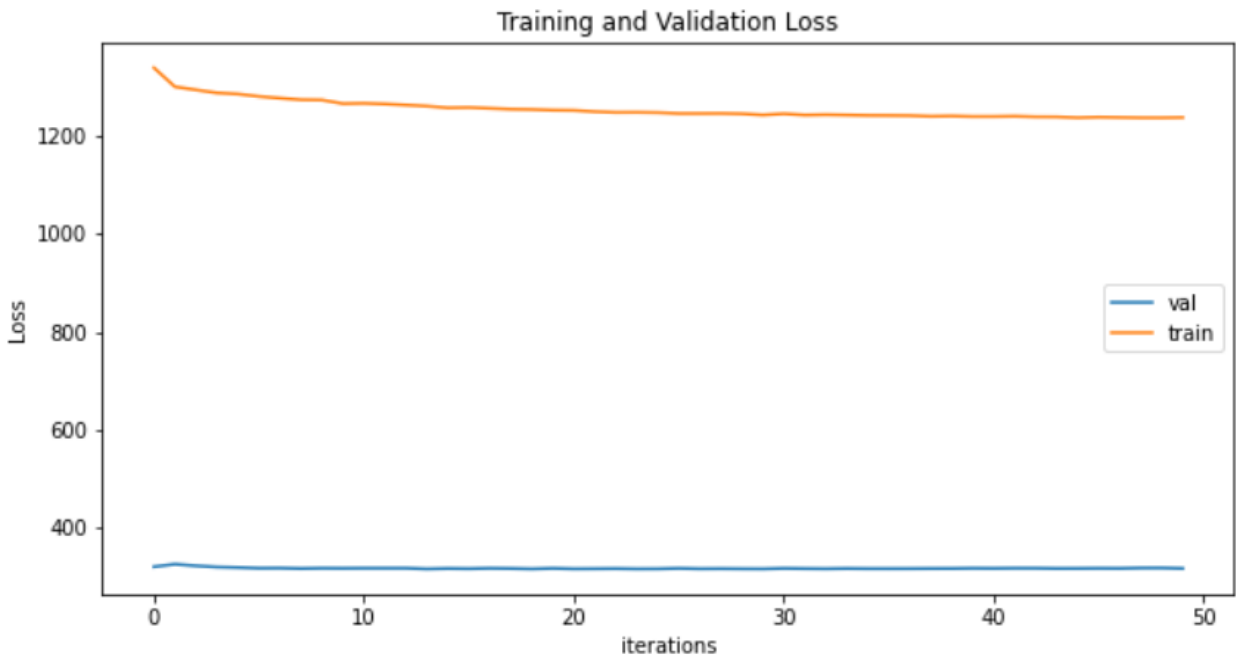
3.2. Best configuration for 2-Layered Network

The following is the hyperparameter configuration to produce the highest accuracy value and the performance results for a 3-layered network.

batch size : 64,
Learning rate: 0.001,
Optimizer: Adam,
Number of neurons in the first layer: 512
Activation function after the first layer: hardswish
Epoch: 25
Drop out rate: 0.5
Validation accuracy: 30.010%
Validation loss: 2.0202

Test accuracy: 29.780%

The following graph shows the validation and loss errors:



3.3. Best configuration for 3-Layered Network

The following is the hyperparameter configuration to produce the highest accuracy value and the performance results for a 3-layered network.

batch size : 64,
Learning rate: 0.001,
Optimizer: Adam,
Number of neurons in the first layer: 512
Number of neurons in the second layer: 216
Activation function after the first layer: tanh
Activation function after the first layer: tanh
Epoch: 50
Validation accuracy: 45.600%
Validation loss: 1.7321

Test accuracy: 44.820%
The following graph shows the validation and loss errors:

Based on Vapnik's Structural Risk Minimization, the models fail to generalize, ie, fail to learn, or overfit when the training loss decreases too much and the validation loss starts to increase. That is, after that point of training the model, it memorizes rather than learning, so it fails on unseen data, that's why it fails to generalize.

The graphs drawn show both the validation and training losses obtained. Ideally, the point where the validation loss starts to increase and keeps increasing while training loss decreases is where we should stop training to avoid overfitting. However, this is not what we wee in the above graphs. That is, it seems that there is a need for more number of epochs of training, ie, models are underfit. However, especially 1 and 2-Layered networks seem that they are not complex enough to learn, or it would take too much time for them to learn. To illustrate, one can look at the highlighted rows in grid search tables, and deduce that with more epochs their loss does not change much. In fact, they start to perform worse with more number of epochs. In particular, the best models for 1 and 2 layered networks are trained on 25 epochs, and it can be seen in the same graphs, looking at rows that are 3 below, that with more epochs (keeping all the other hyperparameter configurations same), their predictive performance gets worse. That's why, the point where I stopped training seems not to be coherent with Vapnik's Structural Risk Minimization.

4. Performance Metric

Accuracy is merely a measure of how many instances were correctly classified. That's why it is a feasible metric to evaluate cases where the misclassified items are not that important. For example, if we are to trying to find the faulty systems based on some data, and finding the faulty ones are so important that even the "potentially-faulty ones" are of major importance, we might overlook some False Positives. However, the extreme case for what is explained might be just labeling every instance as positive, which is obviously not an acceptable performance. In fact, if there is a class imbalance, and the number of positive instances is truly higher than that of negative ones, it is possible for a classifier to just label all instances as positive and get extremely high accuracy results. Merely trying to maximize the accuracy value might lead to such a case. That is, hyperparameter optimization assessing other metric would be different than these.

The class imbalance is not the only reason which makes the accuracy value not a feasible metric. For example, if the interest in finding the negative ones is as high as or more than that in the positive ones, we better check other metrics, such as precision.

For this dataset, I first checked the number of instances for each class and did not observe a class imbalance problem. If it were the case, I would have checked their weighted accuracy values as well. However, since accuracy, in the end, is a metric that is only concerned about the correctly labeled instances, it still can be tricked easily. There are other metrics that are commonly used to assess the predictive performance such as F1-score, however, it might be biased towards the class with the higher number of annotations. One metric commonly used for such cases is matthews correlation coefficient (MCC).

5. Other Questions

5.1. What are the advantages/disadvantages of using small/big learning rates?

Learning rate is a hyperparameter to decide how much the weights are updated based on the calculated error. The neural networks' learning algorithm is based on gradient descent which is to find the minimum error. Although it is guaranteed that the algorithm will converge to a minimum, it might get stuck at some global minima too, which is undesirable. That is, choosing a large learning rate, the algorithm is susceptible to converge to a local minimum, and produce poor performance. Whereas, choosing a small learning rate makes the algorithm run slower, ie, it takes too many epochs for it to converge.

5.1. What are the advantages/disadvantages of using small/big batch sizes?

Batch size is a hyperparameter to determine the number of samples to be propagated through the network before its weights are updated. For example, for stochastic gradient descent algorithm has a batch size = 1. Choosing a large batch size makes the network train faster, as it can be optimized for parallel processing. However, although the reason for it is not well known, choosing a really large batch size causes overfitting, ie, prevents the model from generalizing well.