Derya TINMAZ - 2380947

Part 3 - Report

In this part, we are expected to perform an hyper parameter search for the multi layer perceptron algorithm model. Our hyper parameters are number of hidden layers, number of neurons in these layers, learning rate, number of iterations (epochs), and activation functions. Among these five hyper parameters, I consider two different values for each, and end up with a 32-row grid search table.

Using Model	Activation Functions	Neurons	Hidden Layers	Epochs	Learning Rate	Accuracy Score	Confidence Interval
1	sigmoid	50	1	1500	0.1	47.98	(44.94, 51.02)
2	tanh	50	1	1500	0.1	53.20	(51.98, 54.43)
3	sigmoid	500	1	1500	0.1	44.48	(39.60, 49.36)
4	tanh	500	1	1500	0.1	72.10	(70.58, 73.62)
5	sigmoid	50	2	1500	0.1	45.62	(41.53, 49.71)
6	tanh	50	2	1500	0.1	60.52	(57.26, 63.88)
7	sigmoid	500	2	1500	0.1	10.68	(10.30, 11.06)
8	tanh	500	2	1500	0.1	71.83	(69.32, 74.34)
1	sigmoid	50	1	15000	0.1	42.49	(40.32, 44.63)
2	tanh	50	1	15000	0.1	49.38	(48.35, 50.41)
3	sigmoid	500	1	15000	0.1	41.86	(35.63, 48.07)
4	tanh	500	1	15000	0.1	70.13	(68.91, 71.38)
5	sigmoid	50	2	15000	0.1	50.22	(48.73, 51.72)
6	tanh	50	2	15000	0.1	61.77	(57.26, 66.28)
7	sigmoid	500	2	15000	0.1	10.63	(10.37, 10.90)
8	tanh	500	2	15000	0.1	56.38	(39.55, 73.21)
1	sigmoid	50	1	1500	0.01	56.37	(55.29, 57.46)
2	tanh	50	1	1500	0.01	53.97	(53.40, 54.65)
3	sigmoid	500	1	1500	0.01	66.12	(65.42, 66.81)
4	tanh	500	1	1500	0.01	65.60	(64.60, 66.60)
5	sigmoid	50	2	1500	0.01	56.85	(55.50, 58.22)
6	tanh	50	2	1500	0.01	53.24	(51.33, 55.14)
7	sigmoid	500	2	1500	0.01	67.53	(65.84, 69.22)
8	tanh	500	2	1500	0.01	53.94	(53.04, 54.84)
1	sigmoid	50	1	15000	0.01	57.09	(56.16, 58.02)

2	tanh	50	1	15000	0.01	54.38	(53.67, 55.09)
3	sigmoid	500	1	15000	0.01	71.66	(71.08, 72.25)
4	tanh	500	1	15000	0.01	70.88	(69.88, 71.87)
5	sigmoid	50	2	15000	0.01	54.91	(52.67, 55.15)
6	tanh	50	2	15000	0.01	51.29	(49.99, 52.60)
7	sigmoid	500	2	15000	0.01	70.70	(69.03, 72.38)
8	tanh	500	2	15000	0.01	53.35	(51.63, 55.07)

As you can see from the table, I tried two different activation functions which are sigmoid and tanh as you recommended us. For number of neurons inside the hidden layer/ layers, I wanted to try relatively a small(50) and a big number(500). For number of hidden layers, I did not want to make it too complicated, so I tuned among one and two hidden layers. For number of epochs, I also tried relatively small(1500) and big(15000) numbers. Learning rate generally takes small values, so I tried the values which are generally used in trainings, 0.1 and 0.01.

I have utilized the Adam optimizer, used batch gradient descent learning and used the cross entropy loss function during the training.

I gave number of epochs and learning rate as parameters to my function, I made 8 models for other 3 hyper parameters combinations. I trained my models with train dataset and tested them with validation dataset to tune the hyper parameters for 10 times for each hyper parameter combination. I take the accuracy results average and calculated the confidence interval for each. Even though I used cuda to speed up my code, this process took too many hours.

As you can see from the table, 6 out of 32 hyper parameter settings are above 70, and they are so close to each other. Since our model is not deterministic, from time to time maybe these models work better. However, If we consider only these results, I got the best result (best accuracy) when:

Activation function:	**tanh**
Number of neurons:	**500**
Number of hidden layers:	**1**
Number of epochs:	**1500**
Learning rate:	**1**

After finding my best hyper parameter setting, I combined the given train and validation dataset, and their labels. Then, I trained my model with this datasets.
After testing my final model with test datasets I got:

Derya TINMAZ - 2380947

average accuracy:	**90.54**
confidence interval:	**(90.34, 90.75)**

This score actually surprised me, I have tried a few times too, but end up similar results. Therefore, I think my final model works well.

Questions

– What type of measure or measures have you considered to prevent overfitting?

I try not to iterate too much, the number of epochs that I used was not excessive, and we tuned our hyper-parameters with respect to the validation dataset, we tuned our weights with respect to the training set and tested it on test data set, to not to memorize the examples.

– How could one understand that a model being trained starts to overfit?

While its training accuracy is almost 100% and training loss is almost 0, but validation accuracy is starting to get lower and validation loss is starting to get higher after some point. (We can see it from numbers and also from graphs)

– Could we get rid of the search over the number of iterations (epochs) hyper parameter by setting it to a relatively high value and doing some additional work? What may this additional work be? (Hint: You can think of this question together with the first one.)

Yes, we could do it by observing the accuracy values of training dataset and validation dataset. When accuracy validation is almost %100, and validation accuracy is starting to get lower, we could stop the training data, and we would not be deciding the number of iterations. But this is not fully safe idea, because there can be more than one local minimum points in our loss function. This may lead us to a worse model.

– Is there a ”best” learning rate value that outperforms the other tested learning values in all hyper parameter configurations? (e.g it may always produce the smallest loss value and highest accuracy score among all of the tested hyper parameter configurations.). Please consider it separately for each task.

No, there is no such a best learning rate. Learning rate's performance depends on other hyper parameter values, such as epochs number. If our epoch number is small our learning rate would not be the best learning rate, it would be higher.

– Is there a ”best” activation function that outperforms the other tested activation functions in all hyper parameter configurations? (e.g it may always produce the smallest loss value and highest accuracy score among all of the tested hyper parameter configurations.). Please consider it separately for each task.

Derya TINMAZ - 2380947

Depending on the type of output that we need from our neural network, there might be a function that performs best.However, we cannot %100 say that this work best with all other hyper parameter configurations best. For extreme cases, it would not be the case.

– **What are the advantages and disadvantages of using a small learning rate?**

The change in our weights are small, so if we start with relatively high loss function value, it will take time and will require many updates before reaching the minimum point. However, since our changes are small, we will probably reach the minimum point eventually, without diverging.

– **What are the advantages and disadvantages of using a big learning rate?**

The change in our weights are big, so if we start with relatively high loss function value, it will not take too much time and will not require too many updates before reaching the minimum point. On the other hand, our big changes can lead to divergent behaviors, we may not reach to minimum point at the loss function at all.

– **Is it a good idea to use stochastic gradient descent learning with a very large dataset? What kind of problem or problems do you think could emerge?**

We consider only for instance to update weights, it is better for large dataset to use stochastic gradient descent learning rather than gradient descent learning. Since we consider all the points before updating the weights in gradient descent, it would take too much time. However, if the dataset is too large, to train the model better, we still probably have to do some number of epochs. Since we update the weights only with one example, it will make too much updates which takes time. Also, stochastic gradient descent learning's loss function is noisier, because it considers only one example while updating, so it can take more time to converge for a very large dataset.

– **In the given source code, the instance features are divided by 255 (Please recall that in a gray scale-image pixel values range between 0 and 255). Why may such an operation be necessary? What would happen if we did not perform this operation? (Hint: These values are indirectly fed into the activation functions (e.g sigmoid, tanh) of the neuron units. What happens to the gradient values when these functions are fed with large values?)**

We divide the instance features by 255 to normalize them, to get values between 0 and 1. This makes our computation more efficient. If we did not perform this action, its computation cost would be high and calculations would be too complex. Also, activation function values converges some number for bigger and smaller values (for example sigmoid function gives 0 for small values, 1 for high values). If we fed these function with high values we would get either 1 or 0. This would not gave us precise information about the data. The model would not work well. But if we fed them values close to 0, we achieve more precision in calculation.