CNG 409

HW1 Part 3 Report

Talal Shafei, 2542371

Hyperparameter Configurations

Required:

Number of hidden layers	$\{3,\ 7\}$
Number of neurons	$\{128,\ 256\}$
Learning rates	$\{1 \times 10^{-3},\ 5 \times 10^{-3}\}$
Learning rates	{Relu, Leaky Relu}

Optional:

Mini batch size	{108, 463}

Note:

Epochs Eliminated because of the use or EarlyStopping callback which stops the training loop if the validation loss doesn't decrease or get worse for a particular number of epochs (assumed 5 in my code)

Result:

32 hyperparameter configurations will be tested in total

Table of the configurations

index	Hidden layers	Neurons	Learning rate	Activation Function	Mini batch size
1.	3	128	1×10^{-3}	Relu	108
2.	3	128	1×10^{-3}	Relu	463
3.	3	128	1×10^{-3}	Leaky Relu	108
4.	3	128	1×10^{-3}	Leaky Relu	463
5.	3	128	5×10^{-3}	Relu	108

6.	3	128	5×10^{-3}	Relu	463
7.	3	128	5×10^{-3}	Leaky Relu	108
8.	3	128	5×10^{-3}	Leaky Relu	463
9.	3	256	1×10^{-3}	Relu	108
10.	3	256	1×10^{-3}	Relu	463
11.	3	256	1×10^{-3}	Leaky Relu	108
12.	3	256	1×10^{-3}	Leaky Relu	463
13.	3	256	5×10^{-3}	Relu	108
14.	3	256	5×10^{-3}	Relu	463
15.	3	256	5×10^{-3}	Leaky Relu	108
16.	3	256	5×10^{-3}	Leaky Relu	463
17.	7	128	1×10^{-3}	Relu	108
18.	7	128	1×10^{-3}	Relu	463
19.	7	128	1×10^{-3}	Leaky Relu	108
20.	7	128	1×10^{-3}	Leaky Relu	463
21.	7	128	5×10^{-3}	Relu	108
22.	7	128	5×10^{-3}	Relu	463
23.	7	128	5×10^{-3}	Leaky Relu	108
24.	7	128	5×10^{-3}	Leaky Relu	463
25.	7	256	1×10^{-3}	Relu	108
26.	7	256	1×10^{-3}	Relu	463
27.	7	256	1×10^{-3}	Leaky Relu	108
28.	7	256	1×10^{-3}	Leaky Relu	463
29.	7	256	5×10^{-3}	Relu	108
30.	7	256	5×10^{-3}	Relu	463
31.	7	256	5×10^{-3}	Leaky Relu	108
32.	7	256	5×10^{-3}	Leaky Relu	463

Training setup

The classifier class is very flexible it can take number of hidden layers and neurons in those layers as a parameters in the constructor and then connects the layers in the forward pass using a for loop, also the fit depends on mini batch learning procedure

There will be an outer loop that will run for 32 times (total number of hyperparameter combinations)
And inner loop that will run for 10 times, number of training on each hyperparameter combination
accuracy list will collect the accuracy after each iteration in the inner loop so we can get it's mean and standard deviation then calculates the confidence interval

the code was executed on Google Colab and it took 5h 55m 35s on CPU runtime

Table of the Statistics and the 95 % Confidence Intervals

Each hyperparameter configuration will be executed 10 times and the mean of the accuracy scores and its standard deviation will be copied to the table with the confidence interval

index	Mean (%)	Standard Deviation	Confidence Intervals
1.	84.886	1.730	[84.287,85.485]
2.	85.906	0.686	[85.669,86.144]
3.	84.564	1.716	[83.969,85.158]
4.	86.157	1.168	[85.753,86.562]
5.	79.551	4.111	[78.126,80.975]
6.	82.665	2.402	[81.833,83.497]
7.	80.542	2.104	[79.813,81.271]
8.	82.674	1.051	[82.310,83.038]
9.	85.839	1.298	[85.390,86.289]

10.	86.676	0.984	[86.335,87.017]
11.	85.340	0.920	[85.022,85.659]
12.	86.819	0.462	[86.659,86.979]
13.	84.838	2.666	[83.914,85.762]
14.	84.047	1.711	[83.454,84.639]
15.	85.831	1.835	[85.196,86.467]
16.	84.220	1.600	[83.665,84.774]
17.	85.996	1.615	[85.437,86.556]
18.	85.652	0.875	[85.349,85.955]
19.	86.144	1.855	[85.502,86.787]
20.	85.648	1.809	[85.022,86.275]
21.	82.758	1.490	[82.242,83.274]
22.	84.737	1.742	[84.133,85.340]
23.	83.379	1.861	[82.735,84.024]
24.	83.993	2.098	[83.266,84.720]
25.	87.181	1.003	[86.833,87.528]
26.	86.660	1.010	[86.310,87.010]
27.	87.241	1.214	[86.820,87.662]
28.	87.134	1.647	[86.563,87.705]
29.	84.472	1.867	[83.825,85.119]
30.	84.611	2.731	[83.665,85.557]
31.	83.479	1.725	[82.882,84.077]
32.	85.402	2.760	[84.446,86.358]

Best Hyperparameter combination

Corresponds to Index 27 with 87.241% Mean, 1.214 Standard Deviation, and [86.820,87.662] Confidence Interval

- Number of hidden Layers: 7

- Number of Neurons: 256

- Learning Rate: 0.001

- Activation: Leaky Relu

- Batch Size: 108

Final test

In this part we combined the training set with the validation set and shuffled them then trained a new model on them with the best hyperparameters we got from above and we evaluated the model on the test set

After 10 execution we obtained these results:
Mean Accuracy: 94.290 %
Standard Deviation: 0.131
Confidence Interval: [94.244,94.336]

Very good accuracy for a sequential dense layers model on images classification problem

Additional Questions

Q1. What type of measure or measures have you considered to prevent overfitting?

A1. I used EarlyStopping Callback to make sure the training loop break when the Model starts to overfit, by comparing the current epoch validation loss with the previous validation loss, and if the validation loss increased or stayed the same for consecutive number of epochs (assumed in my code 5) the training loop will stop to prevent from the overfit

Q2. How could one understand that a model being trained starts to overfit?

A2. when the training loss is decreasing but the validation loss is increasing

Q3. Could we get rid of the search over the number of iterations (epochs) hyperparameter 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.)

A3. Yes, as I answered in the first question by using an Early Stopping call-back that keeps track of the validation loss

Q4. Is there a ”best” learning rate value that outperforms the other tested learning values in all hyperparameter configurations? (e.g it may always produce the smallest loss value and highest accuracy score among all of the tested hyperparameter configurations.). Please consider it separately for each task

A4. Learning rate 0.001 outperformed the 0.005 learning rate in most cases

Q5. Is there a ”best” activation function that outperforms the other tested activation functions in all hyperparameter configurations? (e.g it may always produce the smallest loss value and highest accuracy score among all of the tested hyperparameter configurations.). Please consider it separately for each task

A5. Leaky Relu function outperformed the Relu function in most cases, I guess the reason is that the derivative of the Relu function is 0 if the input is negative mean while its not 0 for the Leaky Relu which makes the gradient descent more informative and updates the parameters more efficiently

Q6. What are the advantages and disadvantages of using a small learning rate?

A6. Advantages never skip the Global minimum of the loss function, Disadvantages very slow to reach to the desired value (Global minimum or satisfactory point), also hard to escape local minimum (get stuck easily)

Q7. What are the advantages and disadvantages of using a big learning rate?

A7. Advantages very fast to reach to the desired value, Disadvantages are that the steps are big so it is very likely to skip the desired value (diverge)

Q8. 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?

A8. Because we update on each example the path towards the global minimum is very noisy and because of that it can mislead the gradient away from the global minimum, also we can’t vectorize because we are applying the update rule on each example sequentially so it might takes too long to converge

Q9. 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?)

A9. Dividing by 255 is to scale the value of the pixels between 0 and 1 since the maximum value can (a gray scale) pixel takes is 255, if we don't scale the data before feeding it to the neural network we will be feeding large values to the activation function and obtaining large values as predictions so our training loop will take much longer to converge, also while calculating the gradient the large values will have greater loss derivative corresponding to it so the direction of the optimization will be focused on the weights of the large pixels values so again our gradient path will be longer and converging will take much more time and it can be less accurate because the focus was on the dominating pixels

We can think of it for example as the loss function of two parameters with un-scaled inputs when plotted (parameters are the axis) will take the shape of the Oval with global minimum is at the centre
On the other hand the loss function of two parameters of scaled inputs when plotted will take the shape of a circle