Machine Learning Assignment 1

2380343

Part 3

Experiment

I designed a neural network that has 2 hidden layers. I didn't wanted to change it with the grid search because I already had many configurations that I will mention below.

I've decided to optimize the hyperparameters Hidden Layer neurons, Learning Rate, Epoch count and Activation Functions. I have two different configurations for every hyperparameter except the learning rate.

While I was experimenting, I sensed that learning rate changes much among the configurations. So, I wanted to try three different learning rates instead of one. Hyperparameter configurations are:

Learning Rate = [0.5, 0.25]
Hidden Layer Count [1,2]
Hidden Layer Neurons = [100, 200] respectively for hidden layer 1 and 2
If I have two hidden layers, I am not changing the hidden layer neuron number, I am distributing it to both of the layers equally.
Activation Functions = [ReLU, Tanh]
I am applying these functions too all hidden layers.

I have trained my model using a cpu since I don't have a GPU.

I used SGD optimizer with momentum value of 0.85. Normally this is a hyper parameter too. However, since I have more than enought grid rows, and training them taking too long. I have decided to use a constant momentum value.

Furthermore, epoch count is another hyper parameter. While I was training the different configurations, I have observed the validation and training losses. After 200 epoch, almost no change occurs in the loss values, but when I decrease the epoch number, I ended up with higher loss values so it caused underfitting.

Hence, I decided to use constant epoch number, which is 200.

I am sharing a screenshot of my grid search script's output. It prints all the grid rows, losses, accuracies, confidence intervals. I believe it is pretty explanatory.

```
The Grid : Hidden Layer 1, Hidden Layer 2, Learning Rate, Activation Functions
1:              100,              None,          0.5,        ReLU,
2:              100,              None,          0.25,        ReLU,
3:              200,              None,          0.5,        ReLU,
4:              200,              None,          0.25,        ReLU,
5:              100,              None,          0.5,        Tanh,
6:              100,              None,          0.25,        Tanh,
7:              200,              None,          0.5,        Tanh,
8:              200,              None,          0.25,        Tanh,
9:              50,              50,          0.5,        ReLU,
10:              50,              50,          0.25,        ReLU,
11:              100,              100,          0.5,        ReLU,
12:              100,              100,          0.25,        ReLU,
13:              50,              50,          0.5,        Tanh,
14:              50,              50,          0.25,        Tanh,
15:              100,              100,          0.5,        Tanh,
16:              100,              100,          0.25,        Tanh,
Grid row number : 1, Confidance Interval : 94.31 - 95.16, Validation Loss : 0.3211, Training Loss : 0.2616
Grid row number : 2, Confidance Interval : 93.37 - 93.66, Validation Loss : 0.3681, Training Loss : 0.3213
Grid row number : 3, Confidance Interval : 94.54 - 95.38, Validation Loss : 0.3254, Training Loss : 0.2661
Grid row number : 4, Confidance Interval : 93.82 - 94.03, Validation Loss : 0.3575, Training Loss : 0.3108
Grid row number : 5, Confidance Interval : 94.42 - 94.7, Validation Loss : 0.2842, Training Loss : 0.2277
Grid row number : 6, Confidance Interval : 93.58 - 93.82, Validation Loss : 0.3467, Training Loss : 0.304
Grid row number : 7, Confidance Interval : 94.44 - 94.62, Validation Loss : 0.2853, Training Loss : 0.2245
Grid row number : 8, Confidance Interval : 93.29 - 93.5, Validation Loss : 0.3535, Training Loss : 0.3066
Grid row number : 9, Confidance Interval : 17.24 - 62.5, Validation Loss : 1.671, Training Loss : 1.599
Grid row number : 10, Confidance Interval : 92.96 - 93.63, Validation Loss : 0.4612, Training Loss : 0.4021
Grid row number : 11, Confidance Interval : 55.79 - 87.51, Validation Loss : 1.214, Training Loss : 1.124
Grid row number : 12, Confidance Interval : 93.63 - 94.59, Validation Loss : 0.4525, Training Loss : 0.3952
Grid row number : 13, Confidance Interval : 92.65 - 93.19, Validation Loss : 0.3088, Training Loss : 0.2233
Grid row number : 14, Confidance Interval : 93.22 - 93.51, Validation Loss : 0.3646, Training Loss : 0.3073
Grid row number : 15, Confidance Interval : 93.95 - 94.34, Validation Loss : 0.2868, Training Loss : 0.2115
Grid row number : 16, Confidance Interval : 93.48 - 93.73, Validation Loss : 0.353, Training Loss : 0.2952
Selected model configuration is number 3. Training the best model...
Confidance Interval : 96.01 - 96.32
```

So, my best model's confidence interval is [96.01 - 96.32].

Questions

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

Validation Error, Test Error, Training Error, and Epoch number.

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

If at some point, training error keeps decreasing and test error is increasing, this indicated that model is started to memorize the training set instead of learning it. This is called overfitting.

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?

No, because after some epochs, learning way might change and model might start to overfitting. Higher number of epochs doesn't mean it is better.

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.

There is no such a learning rate that outperforms all the other configurations in all cases. Different learning rates are winning in different configurations.

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.

No, it is the same case with learning rate. They beat each other with different configurations.

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

If your initialization is close to the global minima, small learning rate will take you that global minima steady and reliably. However, it will converge so slow, and also there is a big chance that you caught to a local minima which is not the goal.

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

It converges fast, but it might skip the you might at a loss values which is higher than the global minima.

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?

Cost would be huge. If we use batch or minibatch approach, the cost will decrease significantly due to the parallelism, but we have to wait for the previous data point so that the model can learn from it to pass to the next data point. Parallelism is gone with stochastic gradient.

I think it is not a good idea generally. Mini batch approach would be better to use.

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?)

This process is called normaliztion. Data normalization is important because of the activation functions. Some activation functions becomes constant after

some values such as sigmoid or ReLU. Relu is always zero before some point, if we don't normalize our data and if we have a value before the threshold value, which is the relu's threshold, our gradient will become zero. Model will not be able to learn with those gradients. Normalization is not a must, but highlt recommended.

You are dividing all the values to 255. So all the data points came to the interval 0 - 1, which is a nice value to have gradients for most of the activation functions, sigmoid, tanh, etc.