Assignment 1 Part 3 Report

Yıldırım Özen 2521862

November 1, 2024

0.1 Hyperparameters

I have tested hyperparameter configurations:

Number of hidden layers = [1, 2]
Number of neurons = [32, 64]
Learning rate = [0.01, 0.001]
Activation functions = [tanh, sigmoid]

The number of neurons parameter determines the number of neurons in all hidden layers.
I eliminated the need for epochs as hyperparameters since I used early stopping with hybrid parameter check. The function takes loss and accuracy as variables and checks both loss improvement and accuracy stagnation. The function utilizes an internal counter that's set to zero on first initialization. If there is a small loss improvement and some accuracy stagnation, the function increases the counter and when there is a big loss improvement it resets the counter. If the counter reaches a set value, the function decides to stop the training. I've also set an upper limit to epochs as 15000.

0.2 Model

For the model, I used Adam optimizer and PyTorch cross entropy function as the error function. I've written a flexible weight initializer and forward pass functions(*init_weightsandforward_pass*) that work for any amount of hidden layers and hidden neurons. I did not implement softmax in $forward_pass$ since it is built into the PyTorch cross entropy function.
I employ the hyperparameter selection model through these steps:

1- I initialize weights and biases according to my hyperparameters. All weights and biases are initialized with 0 mean 1 std deviation normal distribution.
2- I initialize Adam optimizer using my parameters and learning rate.
3- I use the forward pass function to generate model outputs.
4- I calculate mean cross entropy loss using PyTorch *cross_entropy* function.
5- I zero the gradient of the optimizer.
6- I calculate the gradient of the error function using *backward*.
7- I update the weights and biases according to gradients using *step*
8- I then check the weights and biases on the validation set, using no_grad since we won't update our parameters based on the validation set.

I do these 10 times for every configuration, then take the average of and store every hyperparameter configuration's results in an array. When all configurations are finished, I select the configuration with the highest validation accuracy. Then I train that configuration with the train and validation combined dataset and use it on the test dataset to see test accuracy. I do this 10 times and obtain the mean test accuracy and the confidence interval.
During all this, I document the losses and accuracies in the terminal every 100 steps. Furthermore, I store every configuration's mean losses, mean accuracies and confidence intervals in a txt file named *part3_results.txt*. The results are appended to the file, so viewing earlier generated results are also possible. Every run is separated by a line of dashes.

0.3 Results

Please add explanatory comments on your implementation since all the work is going to be graded manually. Please, in your report, explain your training setup (hyperparameters, values, neural network architecture, etc.), procedure (how you have trained your model, the accuracy performance results of hyperparameter configurations on the validation dataset, etc.). and the final test set accuracy results (all result's confidence intervals should be provided explicitly). The results of one run are as follows:

Hidden Layer	Neurons	Learning rate	Activation function	Train loss	Validation loss	Train Accuracy	Validation Accuracy
1	32	0.01	tanh	0.3084	0.4963	0.8937	0.8417
1	32	0.01	sigmoid	0.2755	0.4223	0.9050	0.8570
1	32	0.001	tanh	0.3818	0.4997	0.8672	0.8324
1	32	0.001	sigmoid	0.3428	0.4228	0.8805	0.8515
1	64	0.01	tanh	0.2580	0.5585	0.9107	0.8450
1	64	0.01	sigmoid	0.2292	0.4342	0.9210	0.8637
1	64	0.001	tanh	0.3232	0.4987	0.8874	0.8403
1	64	0.001	sigmoid	0.3060	0.4107	0.8927	0.8571
2	32	0.01	tanh	0.3556	0.5339	0.8791	0.8298
2	32	0.01	sigmoid	0.2439	0.4485	0.9156	0.8570
2	32	0.001	tanh	0.4235	0.5714	0.8533	0.8116
2	32	0.001	sigmoid	0.3286	0.4189	0.8853	0.8541
2	64	0.01	tanh	0.2033	0.6958	0.9312	0.8247
2	64	0.01	sigmoid	0.2070	0.4396	0.9291	0.8616
2	64	0.001	tanh	0.3060	0.7540	0.8945	0.8049
2	64	0.001	sigmoid	0.2906	0.4056	0.8975	0.8589

Hidden Layer	Neurons	Learning rate	Activation function	Confidence interval+	Confidence interval-
1	32	0.01	tanh	0.8455	0.8378
1	32	0.01	sigmoid	0.8600	0.8539
1	32	0.001	tanh	0.8344	0.8304
1	32	0.001	sigmoid	0.8530	0.8499
1	64	0.01	tanh	0.8460	0.8440
1	64	0.01	sigmoid	0.8663	0.8611
1	64	0.001	tanh	0.8426	0.8379
1	64	0.001	sigmoid	0.8584	0.8556
2	32	0.01	tanh	0.8330	0.8265
2	32	0.01	sigmoid	0.8585	0.8555
2	32	0.001	tanh	0.8169	0.8062
2	32	0.001	sigmoid	0.8567	0.8514
2	64	0.01	tanh	0.8279	0.8213
2	64	0.01	sigmoid	0.8632	0.8598
2	64	0.001	tanh	0.8096	0.8002
2	64	0.001	sigmoid	0.8607	0.8570

The best hyperparameter config was the 1 hidden layer 64 neurons 0.01 learning rate sigmoid function.
Mean Test Loss: 0.6516 Mean Test Accuracy: 0.8555
Test accuracy confidence interval is between 0.8577 and 0.8532

0.4 Implementation Details

forward_pass function:
Matrix multiplies the first layer, adds first layer bias, then for every hidden layer except the last, multiplies the corresponding weights with their input in a while loop and adds corresponding bias. Then, the matrix multiplies the output with the last hidden layer.

init_weights function:
Creates two lists, one for weights and one for biases. First appends a (num_of_features, num_of_neurons) randomly initialized tensor to weights and (1, num_of_neurons) to biases. Then for every hidden layer except the last one, appends a (num_of_neurons, num_of_neurons) tensor to weights and (1, num_of_neurons) tensor to biases. For the last hidden layer appends (num_of_neurons, output_class_amount) tensor to weights and (1, output_class_amount) tensor to biases.

Cuda core initialization:
I check whether the running machine has any Cuda cores at the beginning of the file, then initialize my tensors according to this information. I use Cuda cores when available since they speed up the training process a lot.

0.5 Questions

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

I initially thought of trying a 3-layered and 128 neuron model, however after implementing I saw that it was overfitting, thus I went with at most 2 layers and 64 neurons. Other than that my early stop algorithm detects accuracy stagnation and stops the training even when it's combined with a slow loss improvement, thus preventing overfitting.

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

It can be understood by looking at test and validation accuracies, if test accuracy is increasing or staying the same while validation loss and validation accuracy is decreasing, the model is overfitting.

– 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.)

As I have also implemented, we can consider an early stop strategy that stops whenever there is no more meaningful increase in accuracy and decrease in validation loss.

– 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.).

In my testing, the learning rate of 0.01 was a bit ahead of 0.001 in every hyperparameter configuration.

– 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.).

On my testing, the sigmoid function was ahead of tanh in every hyperparameter configuration.

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

Some of the advantages are more stable convergence and better generalization performance. Some of the disadvantages are longer training time and the chance of getting stuck in local minima.

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

Some advantages are short training time and local minima avoidance. Some disadvantages are the chance of overshooting minima and poor generalization.

– 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?

Since stochastic gradient descent uses one learning example on each step, using it on a large dataset would introduce fluctuations on the loss function and a very slow convergence.

– 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?)

Since these activation functions take exponents of the values that are fed to them, this can cause gradients to disappear or be equal to the highest number possible.