### CNG 409 INTRODUCTION TO MACHINE LEARNING

### ASSIGNMENT 1: ANN GRID SEARCH OPTIMIZATION

Oğuz Kağan Altaş - 2385128

#### Abstract

We are assigned to train an Artificial Neural Network (ANN) on the CIFAR-10 dataset in this homework. The grand aim is to achieve some accuracy on the dataset only using ANN’s. I trained my network on a Linux environment on my local computer to be able to train faster than windows.

**Index Terms—**Artificial Neural Networks; Grid Search; Hyperparameter Configuration.

1. **ıntroductıon**

In this homework, we are assigned to train an Artificial Neural Network (ANN) on the CIFAR-10 dataset. The main aim of the assignment is to have a grid search algorithm to dynamically create ANN models and find the best hyperparameters among the choices. In this homework, I am working on multiclass classification. Therefore I used the softmax activation function in the output layer. I used it because we may use the softmax function when we want to represent a probability distribution over a discrete variable with n possible values. I used several hyperparameters to achieve the goal. The hyperparameters are the following:

1. Learning Rate
2. Activation Functions
3. metin içeren bir resim

   Açıklama otomatik olarak oluşturulduNumber of Hidden Layers
4. Number of Nodes in Hidden Layers

I used the Pytorch library, as mentioned in the assignment, to achieve the goal. I used four for loops to search model parameter configurations and saved the metrics to a text file. The parameters I saved are hyperparameters along with the train, validation, and test accuracies and losses.

1. **PREPROCESSING**

The dataset I am using is called the CIFAR-10 dataset. The CIFAR-10 dataset consists of 60000 32x32 color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. In this homework, we are required to use the dataset in grayscale. I have created three datasets as preprocessing or data preparation as training, validation, and test datasets. I created a validation set from the training dataset. I used torch.utils.data.random\_split() function to achieve my goal. I took %80 of the training set for training and %20 of the dataset for validation. I gave the dataset lengths and the random\_split() method returned me two datasets named validation and train. As a result, I have the following number of data points on each of my datasets:

1. Train: 40000
2. Test: 10000
3. Validation: 10000
4. **ANN - ArchItEcture**

I have created 54 different ANN configurations with my grid search algorithm. I used three different learning rates, three different numbers of neurons in a layer multiplier, three different hidden layer configurations, and two different activation functions. Therefore, 3\*3\*3\*2 = 54 different models. The hyperparameters are as follows:

Figure 1 Hyperparameters

Since the output shape of the network should match the number of classes, the number of units in the final layer is 10. According to the homework, we are required to have three different hidden layer configurations. I have three different parameters, as shown in Figure1, as hidden\_layers. The neurons parameter is a multiplier to the hidden\_layers parameter. With the help of that, I am scalar multiplying the current neurons parameter with the current hidden\_layers parameter in my grid search algorithm. For instance, if we are using neurons[1] and hidden\_layers[1], we will have the following hidden layer configuration: [400,200], which has two hidden layers. Layer 1 has 400 neurons, and layer 2 has 200 neurons.

1. **Traınıng results**

As I mentioned before, I am testing 54 different models. From the results file, I have sorted by validation accuracy. And the following figure2 includes the best models of each k-layered architecture.

One layered architecture hyperparameters are as follows:

Neurons: 2

Hidden\_layers: [200] 🡪[400]

Learning rate: 0.001

Activation Function: Relu

Two layered architecture hyperparameters are as follows:

Neurons: 2

Hidden\_layers: [200,100] 🡪 [400,200]

Learning rate: 0.001

Activation Function: Relu

Three-layered architecture hyperparameters are as follows:

Neurons: 2

Hidden\_layers: [200,100,50] 🡪 [400,200,100]

Learning rate: 0.001

Activation Function: Relu

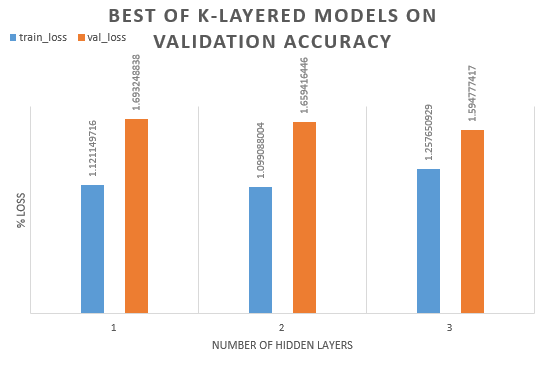


Figure 2 Best three models based on k-layered architecture

According to the results, these are the best performing three models on validation accuracy. We can conclude that using the Relu activation function on our model will perform best on our data. We can see that number of hidden layers does not affect the accuracy. Also, using 0.001 on Adam optimizer for the learning rate is performed best among all other configurations on each model above.

I also took some countermeasures against overfitting my models. I have implemented an early stopping mechanism that watches the validation accuracy. The mechanism stops the training if the patience parameter of the early stopping reaches zero. Furthermore, the mechanism also keeps track of the most accurate epoch on the validation set and saves that model locally. We can understand that our model is overfitting by checking if validation accuracy goes low. Another way to understand overfitting is if training loss goes low and validation loss increases after a point.

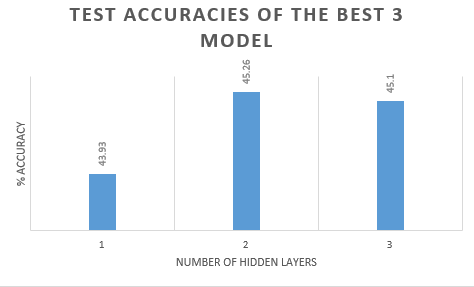


Figure 3 Test accuracies of best three models

As a sanity check, I have tested the following model, saved the test accuracy and loss. Then I compared the results after training.

|  |  |  |  |
| --- | --- | --- | --- |
| neurons | hidden\_layers | lr | activation\_funcs |
| 1 | [200; 100; 50] | 0.01 | ReLU |

|  |  |  |
| --- | --- | --- |
|  | test\_loss | test\_accuracy |
| Not Trained | 2.308942224 | 10.36 |
| Trained | 1.80238466 | 40.46 |

The accuracies are not similar. Because before the training, in our network, the hyperplane we are trying to configure is not separating the data correctly. But after we train the model, our hyperplane is more well defined than a randomly initialized hyperplane.

1. **Additional questions**
2. What are the advantages/disadvantages of using a small learning rate?

As a disadvantage, A small learning rate may cause the model to get stuck on local minima and prevent it from converging on an optimal solution. However, it may take very long to train. As an advantage, A lower learning rate may allow the model to learn a more globally optimal set of weights because it does not take big steps, which allows for higher accuracy.

1. What are the advantages/disadvantages of using a big learning rate?

A big learning rate may cause the model to converge too quickly to a solution, which means we might miss the local minima because of the big steps. As an advantage, training takes less more time than a lower learning rate.

1. What are the advantages/disadvantages of using a small batch size?

The advantage of having a small batch size is that we can converge faster into a good generalization. The disadvantage of having a smaller batch size is that it is not guaranteed to converge to global optima.

1. What are the advantages/disadvantages of using a big batch size?

The advantage of having a big batch size is about training time. The training period will take less time. However, a big batch size tends to lead to a poor generalization of the data.

1. **gRID sEARCH rESULT TABLES**

tablo içeren bir resim

Açıklama otomatik olarak oluşturuldu

Figure 4 Three-layered parameter searches

metin, dolap içeren bir resim

Açıklama otomatik olarak oluşturuldu

Figure 5 Two-layered parameter searches

metin, dolap içeren bir resim

Açıklama otomatik olarak oluşturuldu

Figure 6 One-layered parameter searches

tablo içeren bir resim

Açıklama otomatik olarak oluşturuldu

Figure 7 All parameter searches