Middle East Technical University

Electrical and Electronics Engineering Department

EE449

Computational Intelligence

Homework 1

Training Artificial Neural Network

Fazlı Oğulcan Baytimur

ID: 2231389

27/04/2022

Basic Concepts

1.1 Which Function

For the most of the nonlinear machine learning problems, Artificial Neural Networks have been used. They predict an output from a trained parameters. One of methods to adjust the parameters is the loss functions. There are several functions, but one of the most popular function is "Cross-Entropy" whose formula is shown in Figure 1.1. In the formula, output size is the batch size, y_i is the true labels and $\hat{y_i}$ is the predictions. When the model predicts correctly loss is zero. On the other hand, when the model could not guess correctly, it punishes itself with the loss function and adjusts weights accordingly.

$$\text{Loss} = -\frac{1}{\substack{\text{output}\\ \text{size}}} \sum_{i=1}^{\substack{\text{output}\\ \text{size}}} y_i \cdot \log \hat{y}_i + (1 - y_i) \cdot \log (1 - \hat{y}_i)$$

Figure 1.1: Cross-Entropy Loss Function Calculation Formula

1.2 Gradient Computation

As it was explained in the lecture notes, and can be seen ***here***, the formula is as it follows:

$$w_{k+1} = w_k - \gamma \nabla_w \mathcal{L}|_{w=w_k} \tag{1.1}$$

If this equation is rewritten:

$$\nabla_w \mathcal{L}|_{w=w_k} = \frac{w_k - w_{k+1}}{\gamma} \tag{1.2}$$

1.3 Some Training Parameters and Basic Parameter Calculations

1.3.1 What are batch and epoch in the context of MLP training?

Epoch is the number of times that the entire dataset is used to train the model. **Batch** is the number of samples the model sees without updating any weight.

1.3.2 Given that the dataset has N samples, what is the number of batches per epoch if the batch size is B?

As trivial as it is, the total number of batches in an epoch would be the dataset over batch size. I.E $\frac{N}{B}$

1.3.3 Given that the dataset has N samples, what is the number of SGD iterations if you want to train your ANN for E epochs with the batch size of B?

In each epoch number of iteration would be $\frac{N}{B}$. When the model is run trough E number of epochs, the iteration number would be $E * \frac{N}{B}$

1.4 Computing Number of Parameters of ANN Classifiers

1.4.1 Consider an MLP classifier of K hidden units where the size of each hidden unit is Hk for k=1, . . ., K. Derive a formula to compute the number of parameters that the MLP has if the input and output dimensions are $\mathbf{D_{in}}$ and $\mathbf{D_{out}}$, respectively.

$$D_{in} * H_1 + \sum_{n=2}^{K} H_{n-1} * H_n + D_{out} * H_K \tag{1.3}$$

1.4.2 Consider a CNN classifier of K convolutional layers where the spatial size of each layer is $\mathbf{H_k \times W_k}$ and the number of convolutional filters (kernels) of each layer is $\mathbf{C_k}$ for k=1, . . ., K. Derive a formula to compute the number of parameters that the CNN has if the input dimension is $\mathbf{H_{in} \times W_{in} \times C_{in}}$.

$$H_1 * W_1 * C_1 * C_{in} + \sum_{n=2}^{K} H_K * W_K * C_K * C_{k-1} (1.4)$$

Experimenting ANN Architectures

2.1 Experimental Work

2.1.1 Training Plots

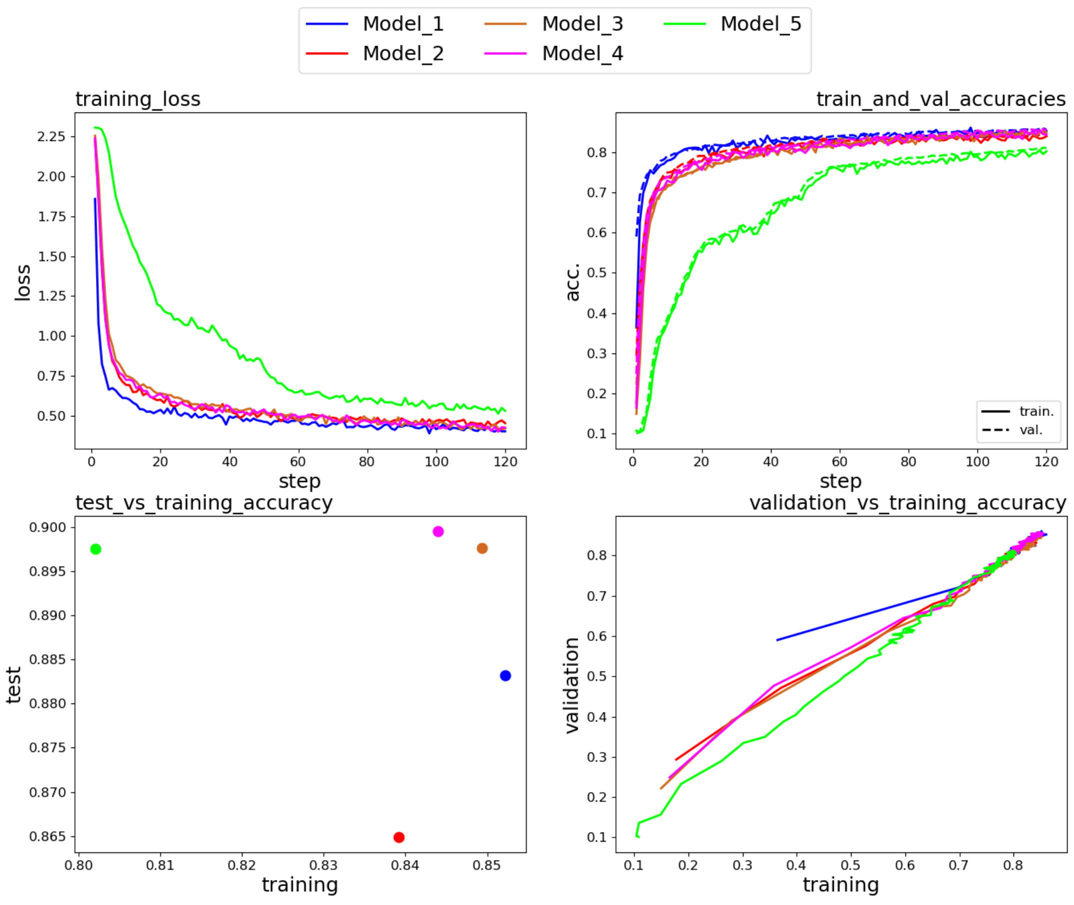


Figure 2.1: Performance Plots for the the models: Model_1=mlp1, Model_2=mlp2, Model_3=cnn3, Model_4=cnn4, Model_5=cnn5

2.1.2 Weight Plots

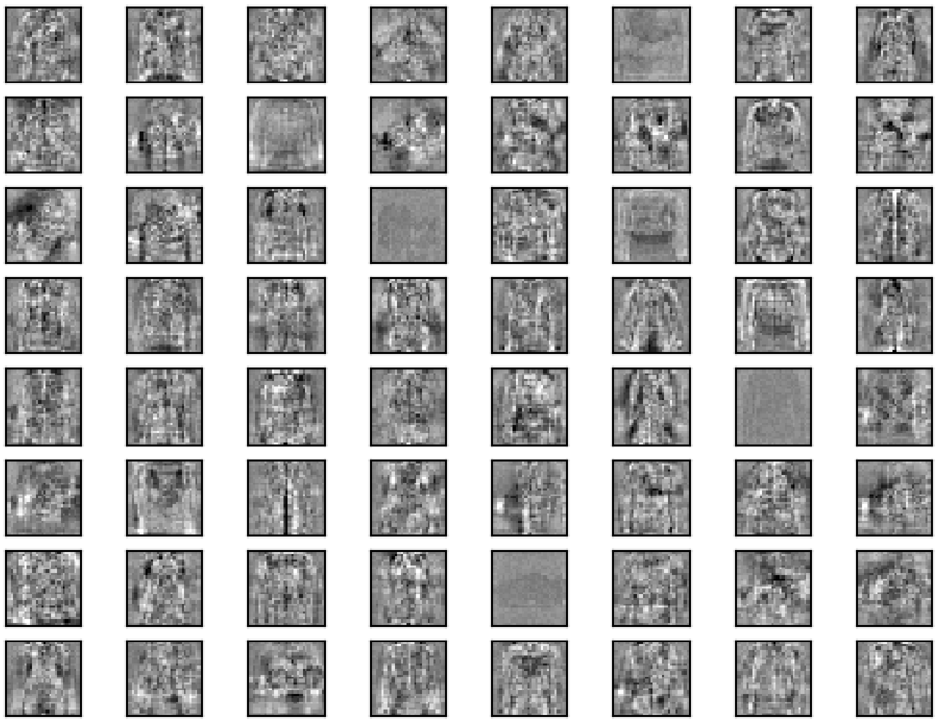


Figure 2.2: Weight Visualisation for mlp1

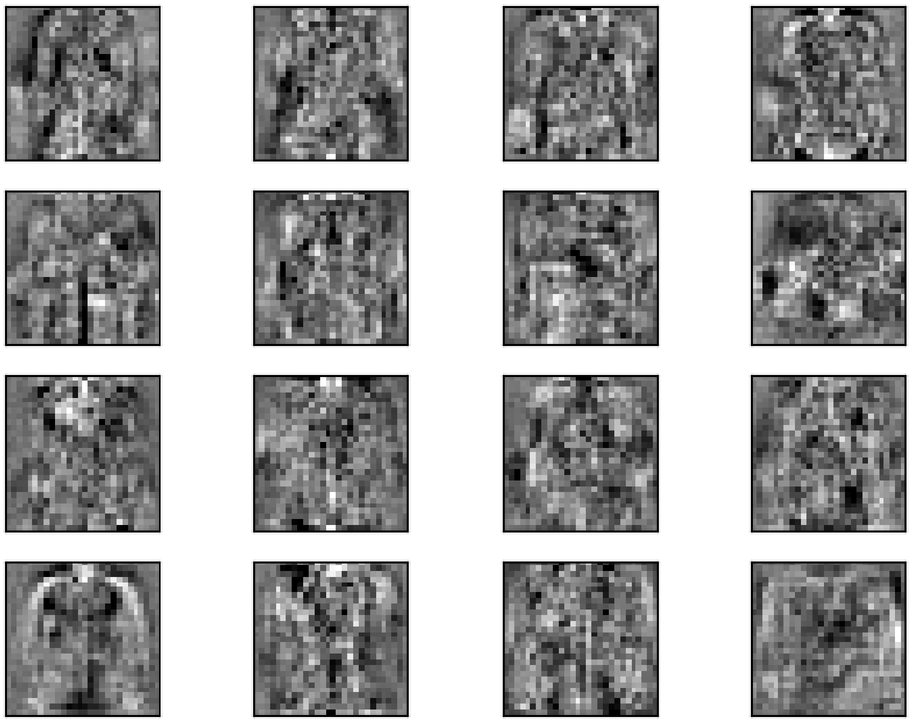


Figure 2.3: Weight Visualisation for mlp2

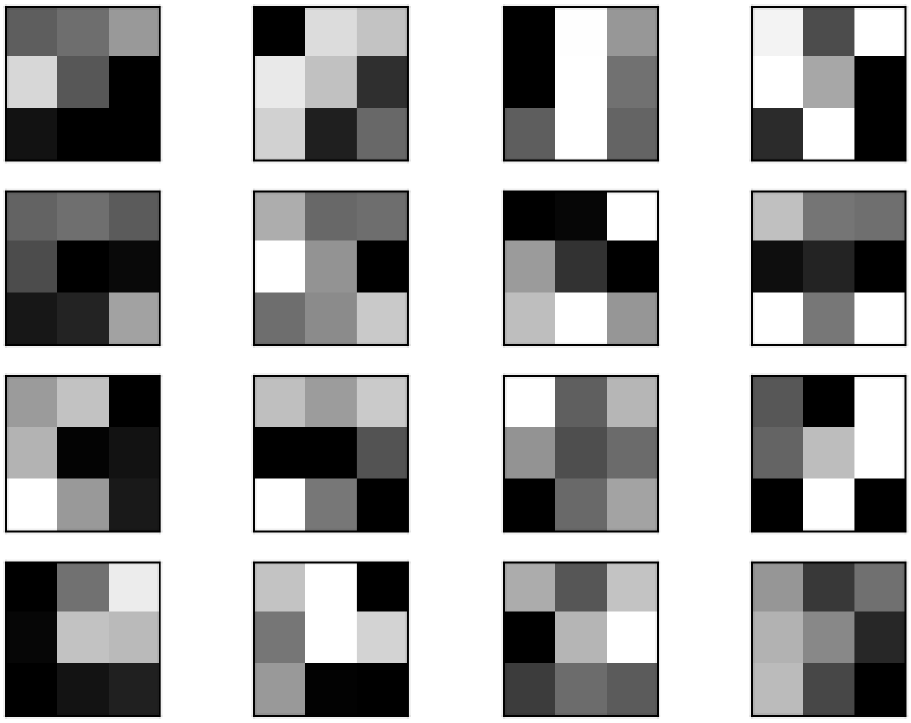


Figure 2.4: Weight Visualisation for cnn3

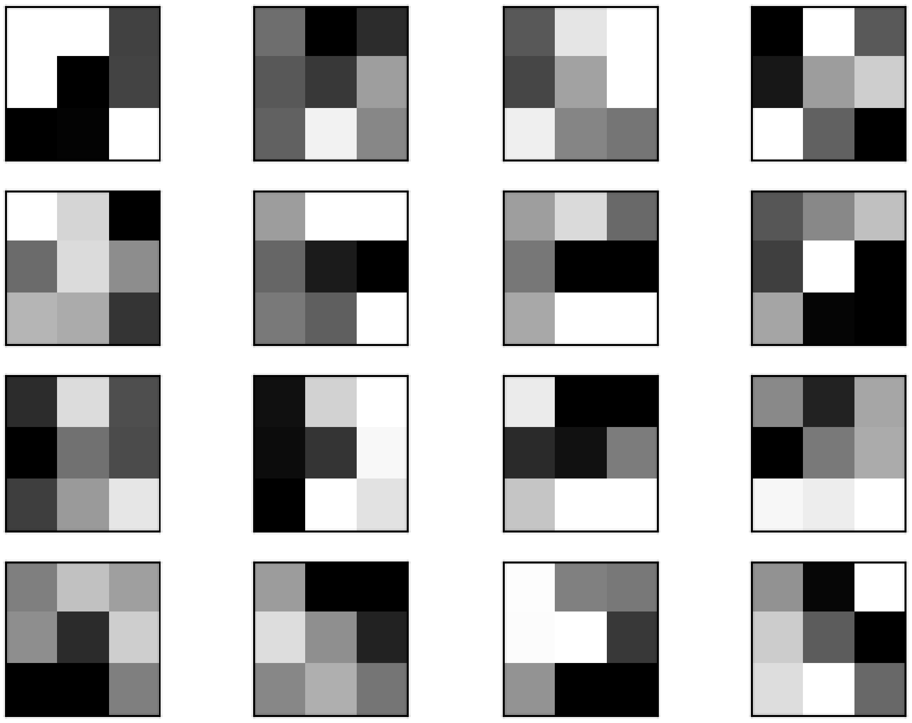


Figure 2.5: Weight Visualisation for cnn4

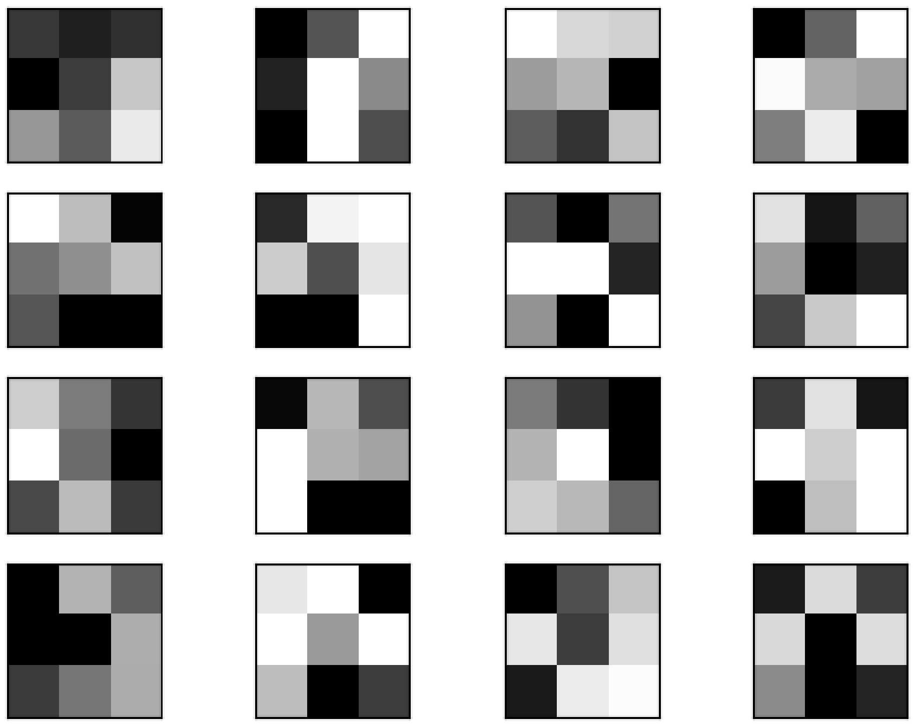


Figure 2.6: Weight Visualisation for cnn5

2.2 Discussions

Compare the architectures by considering the performances, the number of parameters, architecture structures and the weight visualizations.

2.2.1 What is the generalization performance of a classifier?

Generalization performance is the accuracy of the classifier on an unseen data. As it is trained, model can memorize the data and adjust the weights only for the seen data, which can result in a poor test accuracy. If a classifier has a good generalization performance it can predict any unseen data without a problem.

2.2.2 Which plots are informative to inspect generalization performance?

Generalization performance is the ability to classify the unseen data. Therefore, the test accuracy plot in Figure 2.1 should be inspected to decide on the generalization performance.

2.2.3 Compare the generalization performance of the architectures.

By just looking the test vs training plot, bottom left in Figure 2.1, one can assume CNN algorithms have better test accuracy, therefore, they have better generalization performances. However, that would be misleading because, the difference of these algorithms in test accuracy is significantly small, difference <%2. When Training Loss/Accuracy and Validation Accuracy graphs are also seen, MLP1 is the significant best architecture for this dataset.

2.2.4 How does the number of parameters affect the classification and generalization performance?

The architectures has different parameter numbers such that:
Mlp1 has higher number of parameters than mlp2 and they have more parameters than CNNs, cnn3 > cnn4 > cnn5. Parameter effects can be seen in Training Loss/Accuracy and Validation Accuracy graphs. The more parameters a model has, the faster it decrease its losses and increase its accuracy. However, one should always pay attention that high number of parameters can cause overfitting, i.e. model memorise the data with weights.

2.2.5 How does the depth of the architecture affect the classification and generalization performance?

Training Loss/Accuracy and Validation Accuracy graphs shows that, as the deepness increase, rate of change in these functions decrease. The deepest one, CNN5, is the slowest one to reach the final state since it is more complex. This complexity helps to understand complex & big data. However, in small datasets and basic data, it is not needed for a complex model, one can even avoid using deep models in basic data & datasets in order to prevent slowness. A good MLP model would be more suitable for such instances.

2.2.6 Considering the visualizations of the weights, are they interpretable?

For the MLP weights, yes they are. If looked carefully, clothe shapes can be distinguished. The contributes of the weights can be seen in the visualized plots Figure 2.2 2.3 2.4 2.5 2.6. The difference is CNN weights cannot implicate clothes shapes since they are filter weights.

2.2.7 Can you say whether the units are specialized to specific classes?

In Figure 2.2 and 2.3 one can see the silhouettes of the clothes. That means that some units are specialized for certain classes. However, they are not strictly distinguishable since they are not calculated for the human eyes to detect.

2.2.8 Weights of which architecture are more interpretable?

MLP architectures are more interpretable. It is because, the weights are not shared with each pixel, in other words each weight is responsible for a specific pixel. Therefore, the clothes shadows can be seen in the corresponding weight visiualizations Figure 2.2 & 2.3.

2.2.9 Considering the architectures, comment on the structures (how they are designed). Can you say that some architecture are akin to each other? Compare the performance of similarly structured architectures and architectures with different structure.

MLPs and CNNs have similar concepts in their corresponding classes. Mlp1 and mlp2 are similarly constructed with a different deepness. Cnn3, cnn4 and cnn5 are also similar but have different deepness. This extra deepness does not change the working principle. In mlp architectures, models take flattened inputs, and cnn architectures take inputs as image. Since cnn architecture has use filters in a two dimensional convolution, they consider the surrounding pixels as well. Therefore, in spite of having less parameters than mlp architectures, their single parameters has more information than the respected one. That results that CNNs can show similar accuracy rates with less but more informed parameters on a simple dataset such as this one.

2.2.10 Which architecture would you pick for this classification task? Why?

Mlp1 would be the architecture to go. There are several reasons:

1. More parameter number than any other architecture
2. Has the one the best generalization performance
3. Loss & Accuracy function is the fastest
4. It needs less computational power compared to cnn

However, this answer based on this dataset. For a more complex and larger image size datasets cnn achitectures can be more suitable.

Experimenting Activation Functions

3.1 Experimental Work

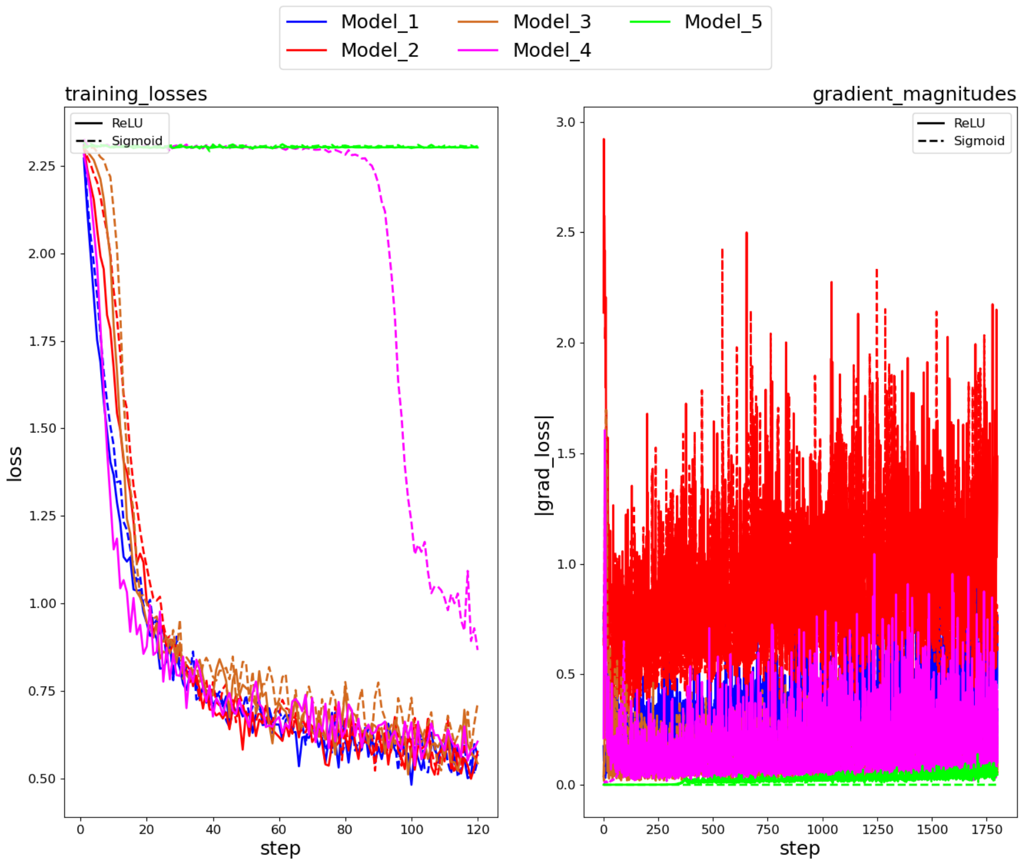
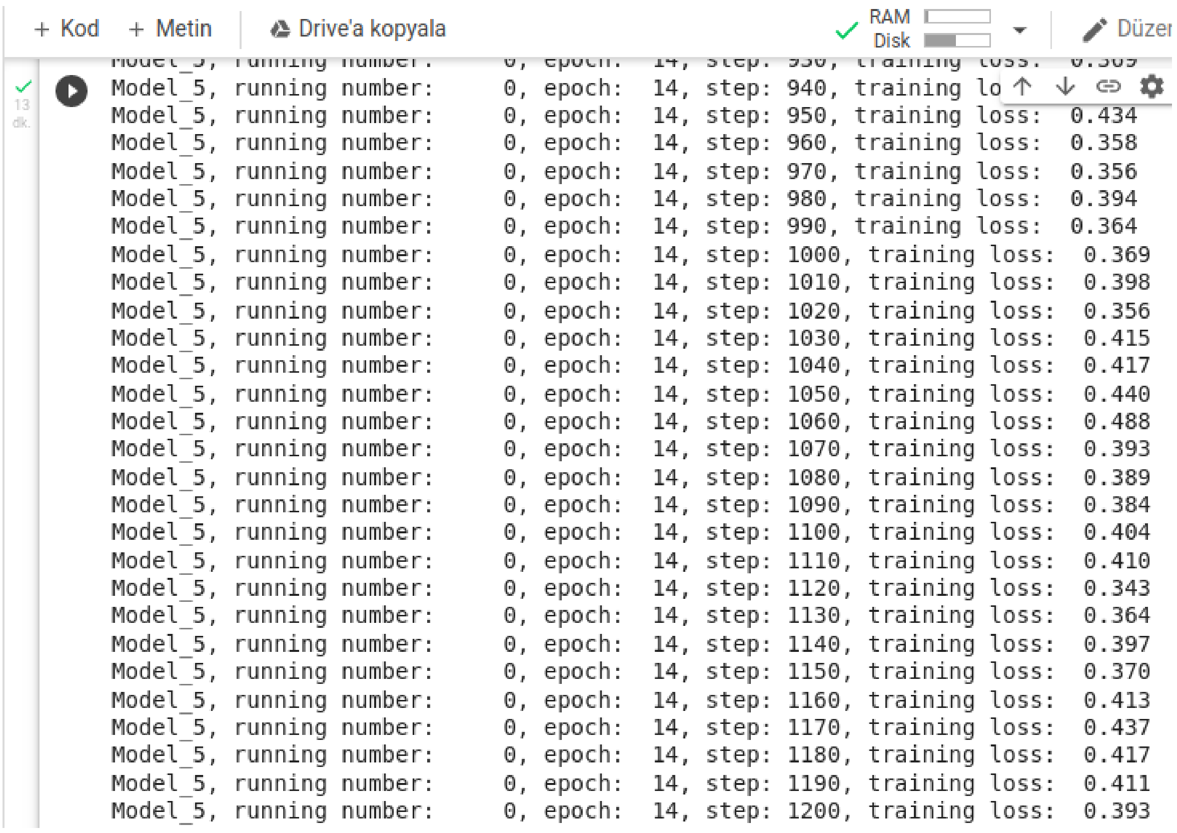


Figure 3.1: Plot for Part3 Experimental Work



```
Model_5, running number:    0, epoch:  14, step: 940, training lo
Model_5, running number:    0, epoch:  14, step: 950, training loss:  0.434
Model_5, running number:    0, epoch:  14, step: 960, training loss:  0.358
Model_5, running number:    0, epoch:  14, step: 970, training loss:  0.356
Model_5, running number:    0, epoch:  14, step: 980, training loss:  0.394
Model_5, running number:    0, epoch:  14, step: 990, training loss:  0.364
Model_5, running number:    0, epoch:  14, step: 1000, training loss:  0.369
Model_5, running number:    0, epoch:  14, step: 1010, training loss:  0.398
Model_5, running number:    0, epoch:  14, step: 1020, training loss:  0.356
Model_5, running number:    0, epoch:  14, step: 1030, training loss:  0.415
Model_5, running number:    0, epoch:  14, step: 1040, training loss:  0.417
Model_5, running number:    0, epoch:  14, step: 1050, training loss:  0.440
Model_5, running number:    0, epoch:  14, step: 1060, training loss:  0.488
Model_5, running number:    0, epoch:  14, step: 1070, training loss:  0.393
Model_5, running number:    0, epoch:  14, step: 1080, training loss:  0.389
Model_5, running number:    0, epoch:  14, step: 1090, training loss:  0.384
Model_5, running number:    0, epoch:  14, step: 1100, training loss:  0.404
Model_5, running number:    0, epoch:  14, step: 1110, training loss:  0.410
Model_5, running number:    0, epoch:  14, step: 1120, training loss:  0.343
Model_5, running number:    0, epoch:  14, step: 1130, training loss:  0.364
Model_5, running number:    0, epoch:  14, step: 1140, training loss:  0.397
Model_5, running number:    0, epoch:  14, step: 1150, training loss:  0.370
Model_5, running number:    0, epoch:  14, step: 1160, training loss:  0.413
Model_5, running number:    0, epoch:  14, step: 1170, training loss:  0.437
Model_5, running number:    0, epoch:  14, step: 1180, training loss:  0.417
Model_5, running number:    0, epoch:  14, step: 1190, training loss:  0.411
Model_5, running number:    0, epoch:  14, step: 1200, training loss:  0.393
```

Figure 3.2: Screen Shot of Model_5 Running Showing Training Loss Value

3.2 Discussions

Compare the architectures by considering the training performances:

3.2.1 How is the gradient behavior in different architectures? What happens when depth increases?

Just looking the plots, one can understand that CNN architecture's values are lower than MLP ones considering gradient behavior. Moreover, among the CNN structures, it is clear that as the deepness increase gradient magnitude tends to go lower. This results that gradient behavior tends to be more stable in deeper algorithms. Moreover, by looking the plot on the right in Figure3.1, CNN5 has a zero gradient loss for the sigmoid function. However, it is not zero for the RELU function.

3.2.2 Why do you think that happens?

The reason could be; when there are multiple layers, the architecture has more parameters defining the object. Since the gradient descent calculations has chain rule, more parameters mean that smaller changes required for the same learning. In my plot there is an error that for the MLP structures, MLP2 has greater gradient losses than MLP1.

However, this should not have occurred. MLP2 should have smaller, at least near equal, values. For this error the reason could be calculating only the first layer. MLP1's first layer has 784 inputs going 64 hidden nodes, whereas MLP2 has the same amount of inputs going to 16 hidden nodes. Since the weight number of MLP2 is smaller than MLP1's, gradient loss could be larger for the first layer.

3.2.3 What might happen if we do not scale the inputs to the range [1.0, 1.0]?

The sigmoid function has two asymptotes which has slope values of zero when the input is large enough. Thus, when the inputs are large the derivatives of these inputs on sigmoid function will be very small in the gradient descent calculations. Consequently, it will reduce the calculation speed since it is going to need more epochs. Therefore, best way increase efficiency of the calculations would be, having a small boundary around the symmetry point of the function to have a linear region to have great derivatives, which is [-1,1].

Experimenting Learning Rate

4.1 Experimental Work

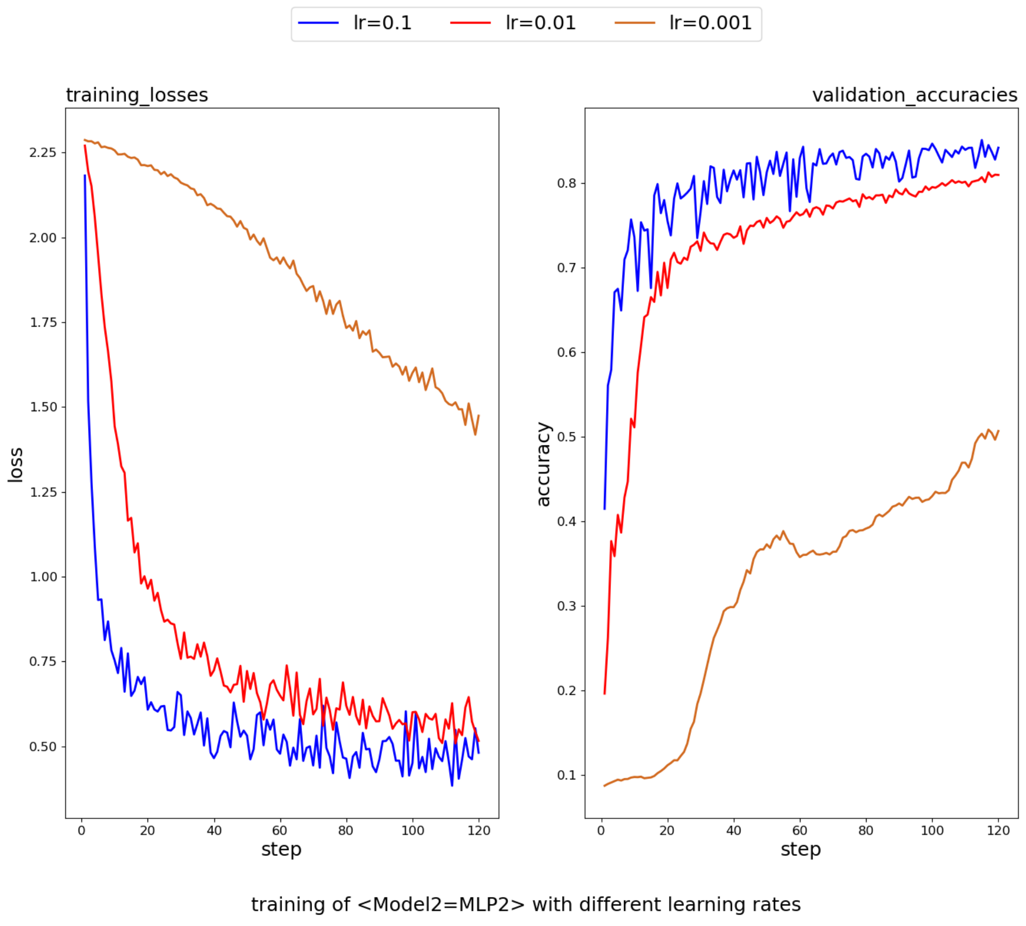


Figure 4.1: MLP2 Architecture With Different Learning Rates

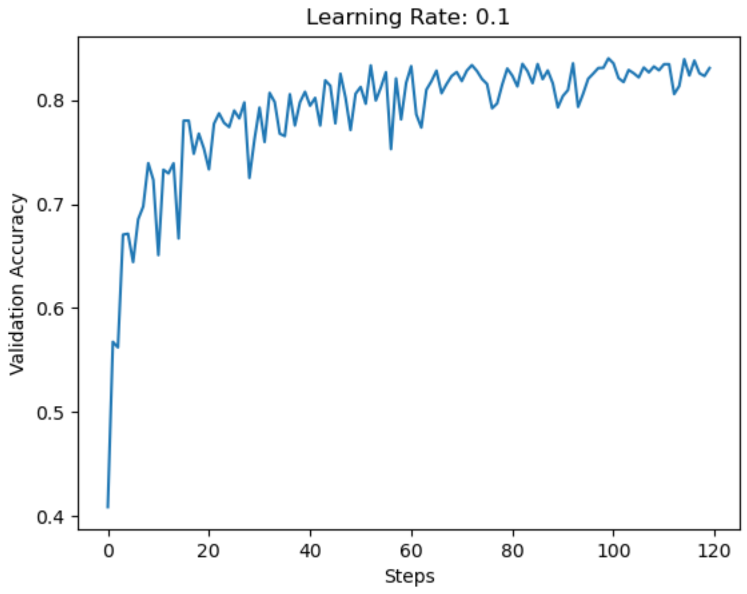


Figure 4.2: Validation accuracy for learning rate 0.1

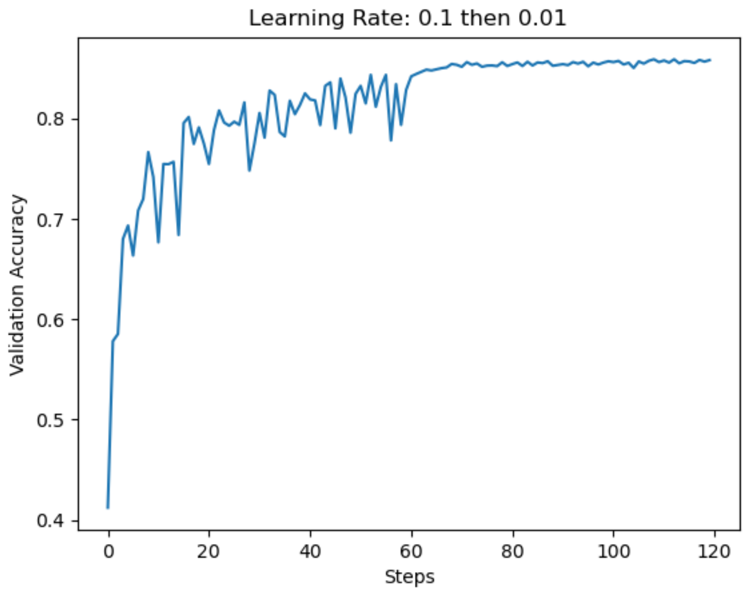


Figure 4.3: Validation accuracy for learning rate 0.1 then 0.01

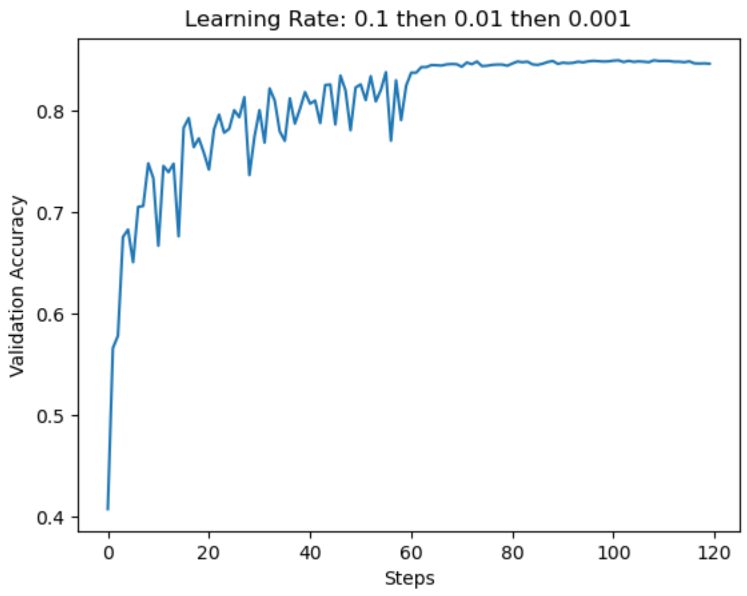


Figure 4.4: Validation accuracy for learning rate 0.1 then 0.01 and 0.001

4.2 Discussions

4.2.1 How does the learning rate affect the convergence speed?

As expected, the more learning rate, the less time it requires to converge. Therefore, a model can learn the fastest with the largest learning rate.

4.2.2 How does the learning rate affect the convergence to a better point?

When the learning rate is too large, it can omit a local or global minimum point as it is seen in Figure 4.5. The reason is that, gradient computation has the learning rate component in it. With very large learning rates, it can oscillate on the outside of the minimum point. In conclusion, learning rate should be large enough to increase the learning speed but small enough to not dodge necessary extremum points and find the perfect point.

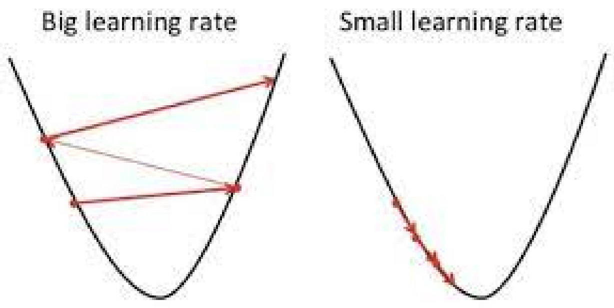


Figure 4.5: Learning rate's magnitude effects on model's learning

4.2.3 Does your scheduled learning rate method work? In what sense?

Yes, it does. The reason is that as it was explained in 4.2.2, with a larger starting learning rate, model quickly goes to its steady state. Afterwards, with a smaller learning rate it finds a better stable point on the loss function. The same idea, can be done continuously. However, the benefit starts to decrease after the first couple of learning rate changes and the change will be nothing but time waste.

4.2.4 Compare the accuracy and convergence performance of your scheduled learning rate method with Adam.

Convergence speed of the proposed is higher in terms of time since the starting learning rate is larger than ADAM's. However, the ADAM optimizer has a better accuracy curve since it adapts itself over the training. In this proposed method, learning rate, even it is changing in specified points, is constant. Therefore it cannot adapt itself over the training. Even though, converge speed of the scheduled method is better, ADAM's accuracy is better than the scheduled one.

Appendix

Since I have written this report in latex, it is both inefficient and eye tiring to add multiple pages of code here. You can reach the project codes, and their PDF versions by clicking ***here.***