Ahmet Akman 2442366
Middle East Technical University
Electrical and Electronics Engineering

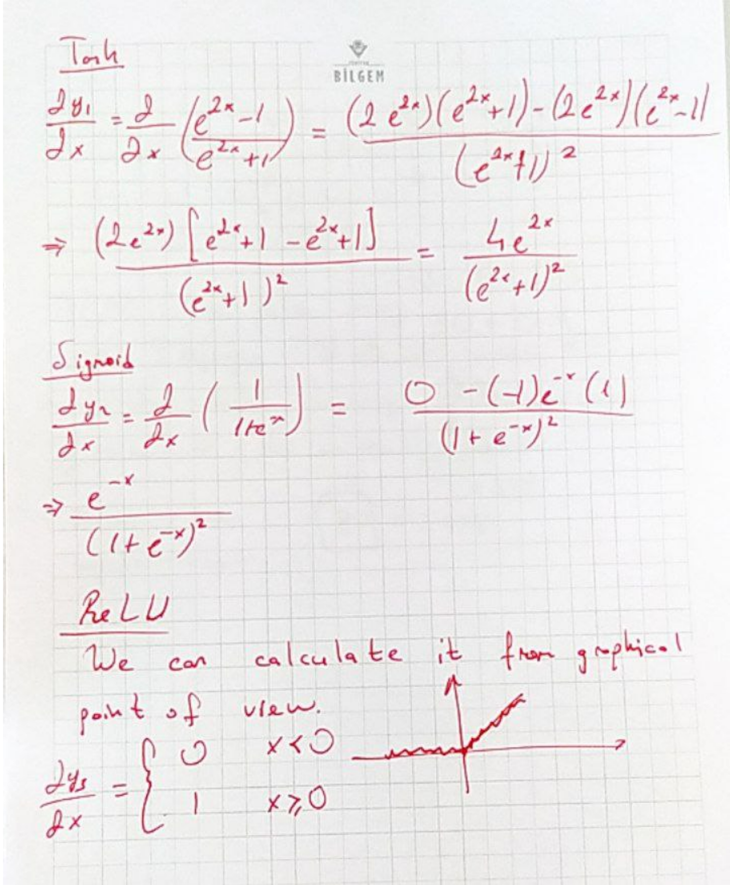
April 7, 2024

HOMEWORK 1 — Report

1 Question 1

1.1 Question 1.1 - Preliminaries

The partial derivaive calculation steps for Tanh, Sigmoid and ReLU activation functions are shown in Figure 1.



Tanh

$$\frac{\partial y_1}{\partial x} = \frac{\partial}{\partial x}\left(\frac{e^{2x}-1}{e^{2x}+1}\right) = \frac{(2e^{2x})(e^{2x}+1)-(2e^{2x})(e^{2x}-1)}{(e^{2x}+1)^2}$$

$$\Rightarrow \frac{(2e^{2x})\left[e^{2x}+1-e^{2x}+1\right]}{(e^{2x}+1)^2} = \frac{4e^{2x}}{(e^{2x}+1)^2}$$

Sigmoid

$$\frac{\partial y_2}{\partial x} = \frac{\partial}{\partial x}\left(\frac{1}{1+e^{-x}}\right) = \frac{0-(-1)e^{-x}(1)}{(1+e^{-x})^2}$$

$$\Rightarrow \frac{e^{-x}}{(1+e^{-x})^2}$$

ReLU

We can calculate it from graphical point of view.

$$\frac{\partial y_3}{\partial x} = \begin{cases} 0 & x<0 \\ 1 & x\geq 0 \end{cases}$$

Figure 1: Partial derivative calculation steps for Tanh, Sigmoid and ReLU activation functions.

Figure 2 illustrates the activation functions' response between -2 and 2. The plots are obtained using matplotlib library as instucted.

Figure 3 indicates the gradients of those functions in the same range. The gradients are calculated using the partial derivatives derived in Figure 1.

1.2 Question 1.2

In this part, utilizing the given template of code, MLP with one hidden layer is implemented. The input-output pairs are fetched from the XOR data provided in utils.py. Note that for all networks learning rate

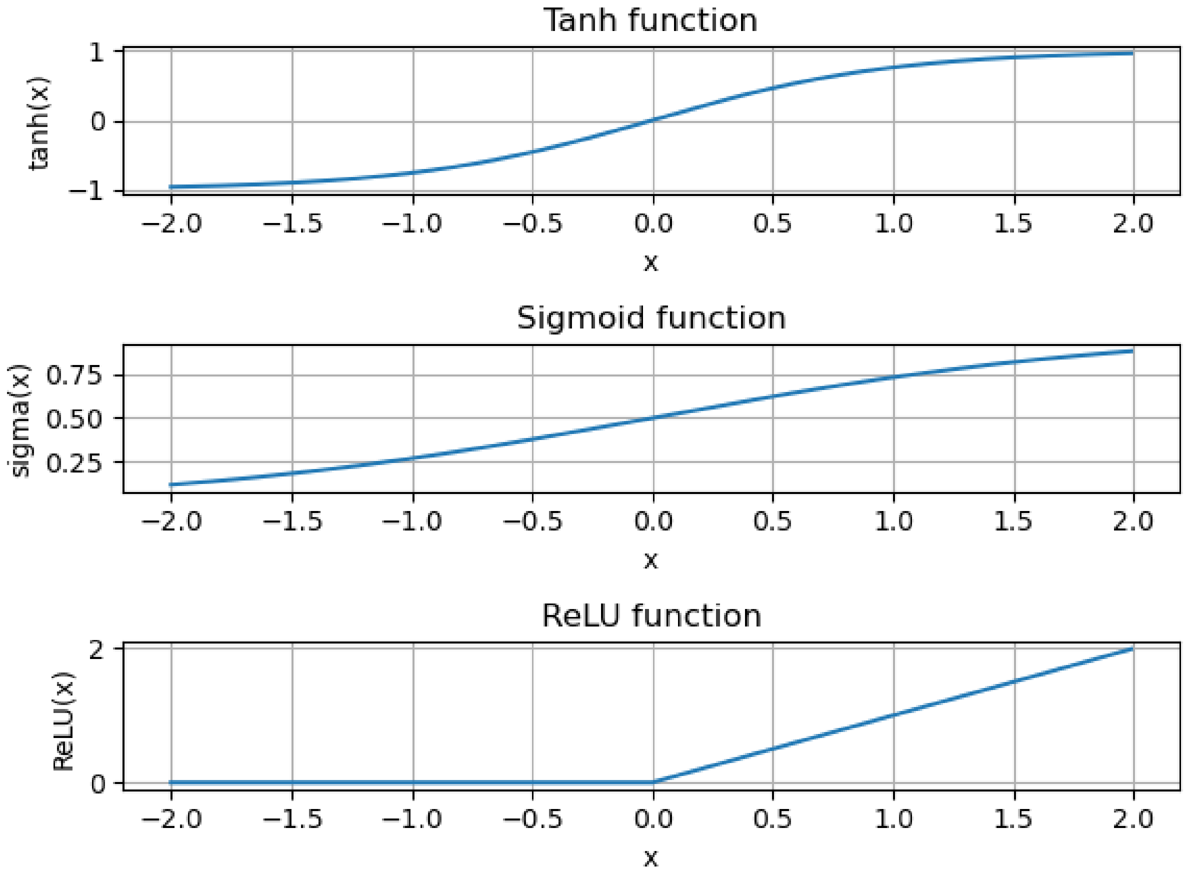


Figure 2: Activation functions plot.

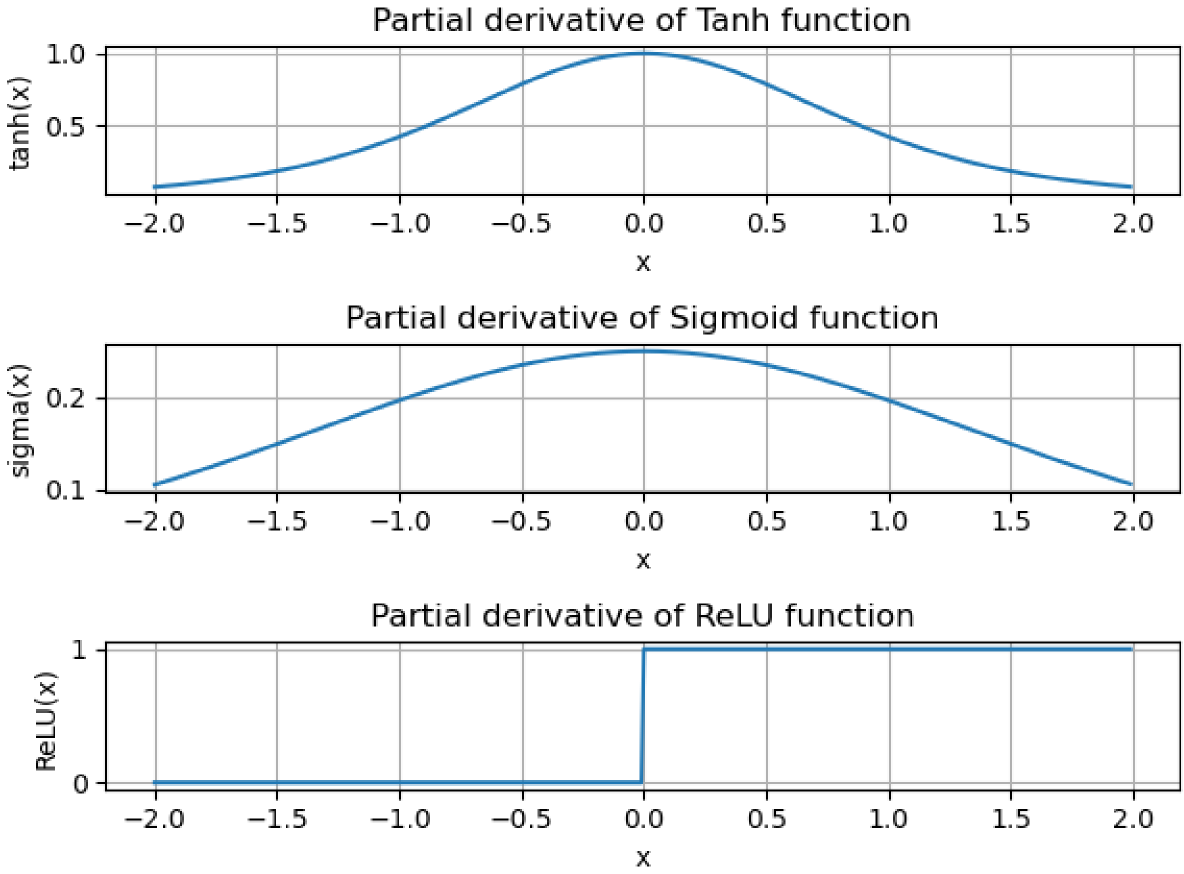


Figure 3: Gradients of the activation functions plot.

is fixed to 0.00001 and seed is utilized. Figure 4 shows the decision boundary for the sigmoid activated network. Similarly, Figure 5 is the decision boundary for the tanh activated network and Figure 6 is the decision boundary for the ReLU activated network.

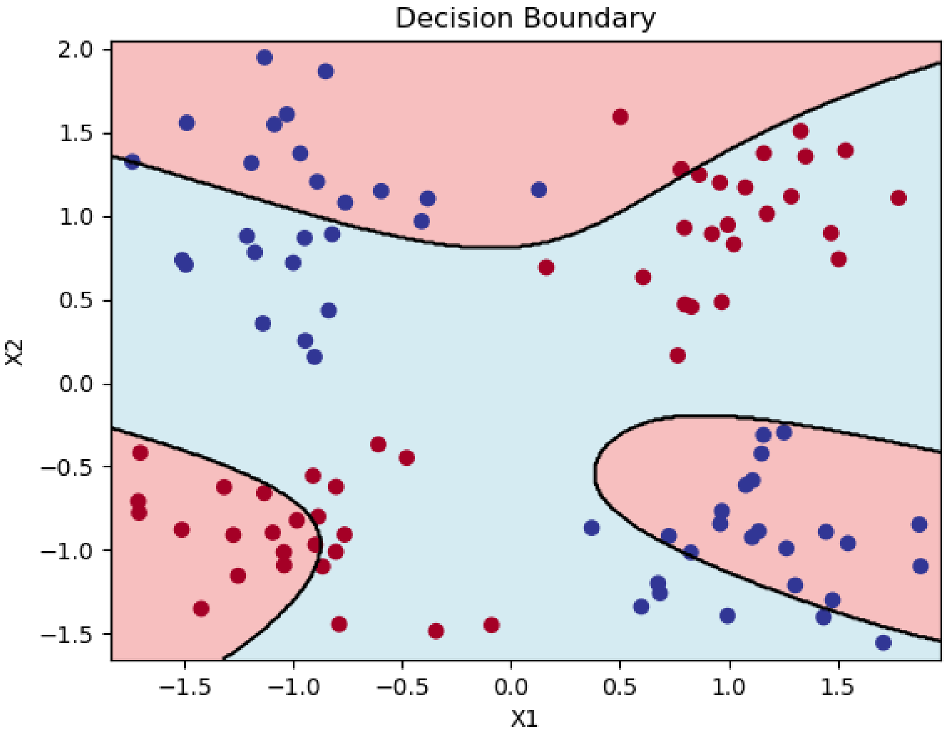


Figure 4: Sigmoid activated XOR problem output.

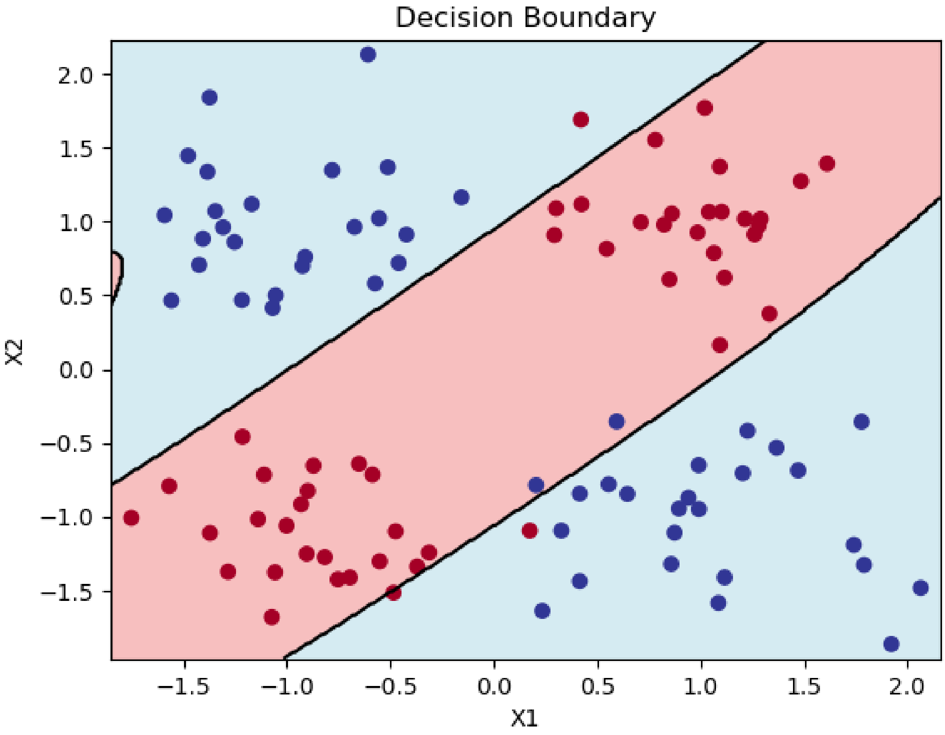


Figure 5: Tanh activated XOR problem output.

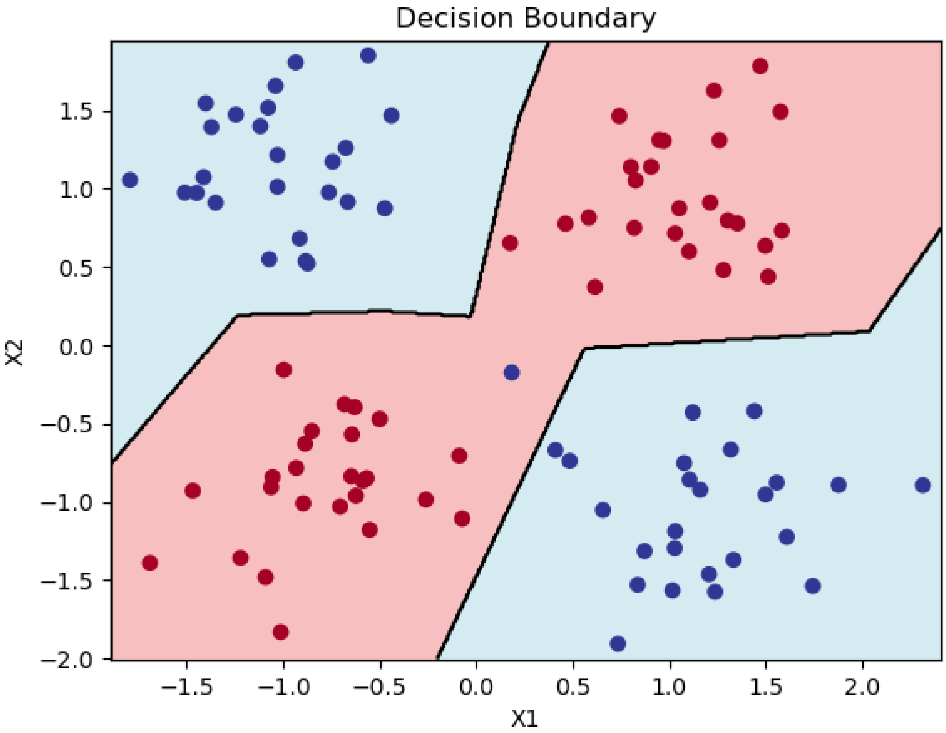


Figure 6: ReLU activated XOR problem output.

1.3 Question 1.3

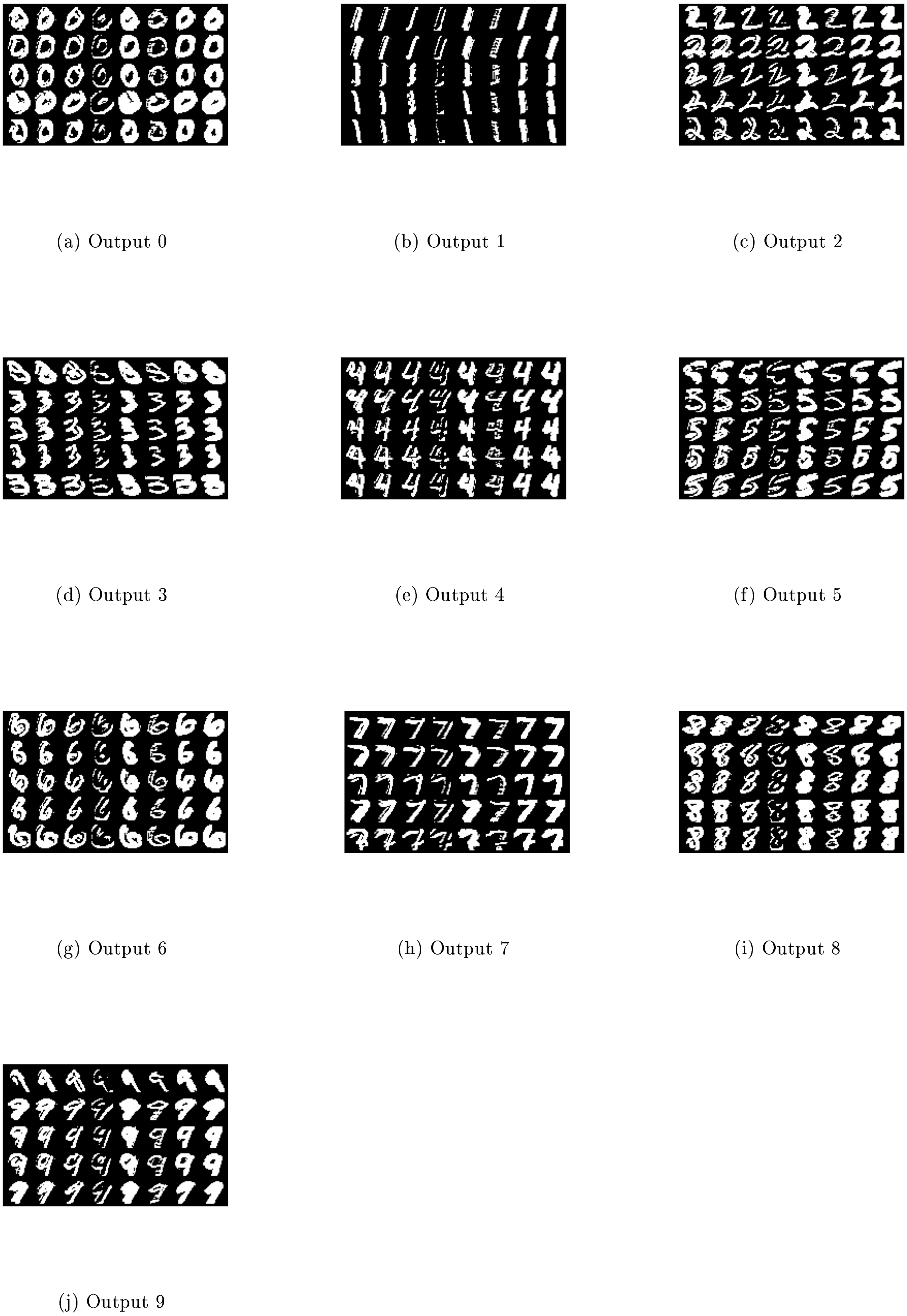
1. All of those activation functions provide a smooth transition from one state to another. The advantage of Tanh and Sigmoid is, they are limited in the range of -1 to 1 and 0 to 1, respectively. This property can be beneficial in some cases. However, the ReLU function is not limited in the range. It is also computationally cheaper than the other two. The disadvantage of ReLU is that it is not smooth at the origin. This can cause some problems in the optimization process. Another advantage of ReLU is negative gradients are zero. This may be helpful in some cases. **2.**

XOR problem is two input decision problem where inputs have to be different than each other to obtain 1. Since the output is always with respect to the state of two variables, decision boundary is not linear. Therefore, a single layer perceptron can not solve since in one step two case evaluations can not be done. Yet, by adding a hidden layer, the problem is solved. The activation functions are used to introduce non-linearity to the network. The decision boundary of the XOR problem is shown in Figures 4, 5 and 6. As can be seen from the figures, the MLP's with activation functions can solve the XOR problem at some extent.

3. The boundaries change in each run, since the initial weights and the data points are randomly generated. Therefore, the decision boundaries are dependent on the training process where initial randomness lead to different outputs.

2 Question 2

A convolution operator is implemented using a set of nested loop. The outputs are checked on a seperate code so that they are identical to the output of torch.conv2d. Using the provided inputs, outputs provided in Figure 7 are obtained.



(a) Output 0

(b) Output 1

(c) Output 2

(d) Output 3

(e) Output 4

(f) Output 5

(g) Output 6

(h) Output 7

(i) Output 8

(j) Output 9

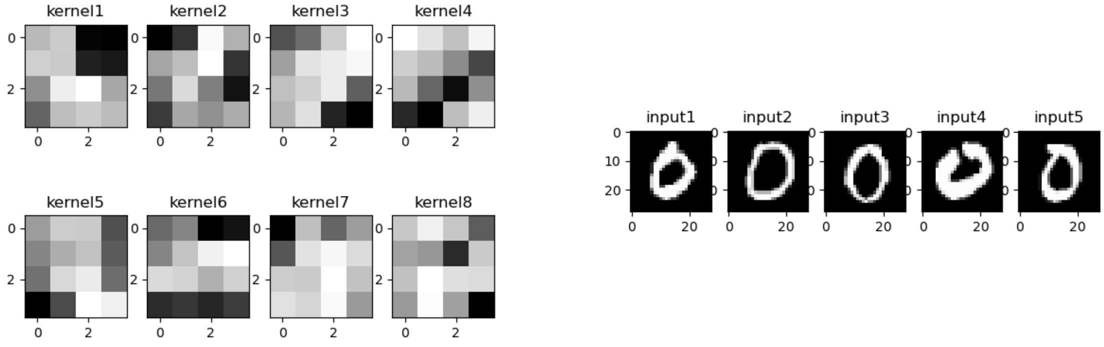
Figure 7: Convolution output

2.1 Question 2.2 - Discussions

1. Two dimensional convolution operation provides filtering in two dimension via a kernel. The kernel is applied to the input image in a sliding window fashion. The output is obtained by element-wise multiplication of the kernel and the input image. The kernel is then shifted by a stride and the process is repeated. That is, two dimensional convolution operation with learnable kernal entries are commonly used in image processing to extract features from the image. The kernel is learned during the training process. Those feature maps encode necessary information about distinct clues in the image. For example in the case of object recognition, a specific kernel can be trained to detect bicyle rims. Kernel of a convolution layer corresponds to the filter function in one dimension. It is used to suppress or enhance certain information in the input image.

2.The sizes of the kernel corresponds to the size of the filter. Therefore, the size list can be explained as follows (batch size, input channels, output channels, filter height, filter width) . Batch size is the number of input sets in a batch if batching is used, here we have not utilized it. Input channels are the number of depth dimension in the image, for example in RGB frames it is three. This might be different for different sensor inputs and representations. The output channel number determines the depth dimension of the output tensor. The filter height and width are the dimensions of the filter.

3. To understand what exactly happened here, let us plot the kernel and input for the number zero. Figure 8 illustrates the plot.



(a) Kernels

(b) Input set for number zero.

Figure 8: Kernel and input for number zero.

When we have a look at what output image represents, on each row an input image is convolved with a kernel. There are different kernels applied to the input image. The output image is the result of those convolutions. Figure 8 shows the input and kernel pairs in order.

4. Each convolution kernel embeds a certain feature of the image. That is, if we have a look at Figure 8.a we can see that kernel 4 enhances the countours on the image whereas kernel 6 enhances the filled part of those contours. This is the reason why we see similarly formatted outputs on the same column. **5.** Similarly, since we "highlight" different properties on the image with out filters, we see different patterns on the output side even if the input is the same.

6. So, we can interpret form 4, and 5 that the output of the convolution layer is the result of the feature extraction process. The output is the result of the convolution of the input image with the learned filters. The output is the feature map that encodes the information about the input image. By post-processing those feature maps, we can obtain the necessary interpreations about the input image.

3 Question 3

3.1 Question 3.1

The implementations related to each architectures are completed as instructed. The code is given in appendix. As the result the plot shown in Figure 9 is obtained. Also the first layer weights recorded are plotted and shown in Figure 10.

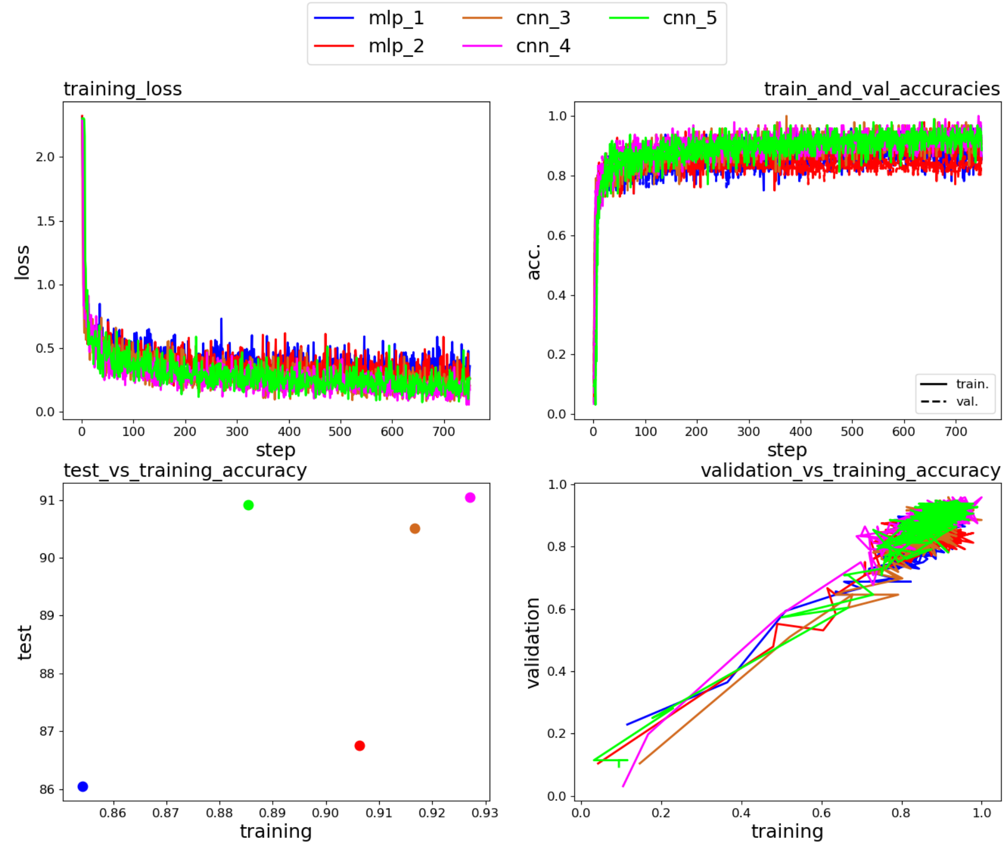


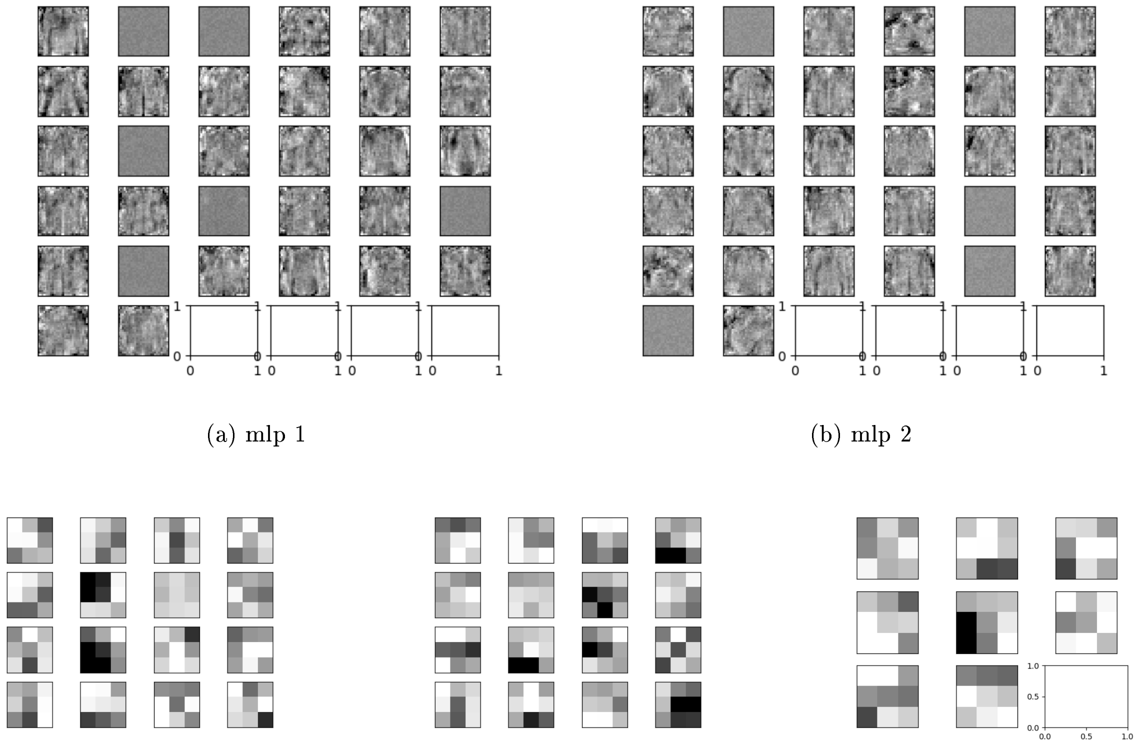
Figure 9: Benchmark of five different architectures.

3.2 Question 3.2

1. Generalization performance of a classifier is the ability of the classifier to perform well on unseen data. That is, the classifier should be able to generalize the patterns in the training data to the test data. Overfitting phenomena is the case where the model fits the training data too much and lose its generalization.

2. The plots, test vs training accuracy and validation vs training accuracy are the most informative in this sense.

3. Somehow, the validation vs training accuracy plot is hard to read out. Therefore deductions are made from test vs training accuracy plots. First of all, one can see right away from the plot is CNN based models generalized better where mlp based ones have testing accuracy is lower than that of training accuracy. On the other hand, amongst CNN based ones cnn4 has best accuracy in terms of both training and test accuracy. However, it can be deduced that cnn5 has the best generalization performance since it has quite high test accuracy compared to its training accuracy. The test scores are even better than



(a) mlp 1

(b) mlp 2

(c) cnn 3

(d) cnn 4

(e) cnn 5

Figure 10: Weights of the first layers.

the training scores.

4.

Architecture	Trainable Params
mlp 1	25,450
mlp 2	27,818
cnn 3	22,626
cnn 4	14,962
cnn 5	10,986

Table 1: Trainable Parameters of Different Architectures

As number of parameters increase number of points to fit to the data increases. Therefore, the model can fit the data better in most basic setting. The number of parameters for each architecture is given in Figure 1. So the classification performance increases as the number of parameters increase up to certain point as we can observe from multi layer perceptron trainings. From cnn3 to cnn4 both training and test accuracy increases but number of parameters are lower in cnn4. However, from cnn4 to cnn5, the test accuracy are almost same whereas the training accuracy decreases considerably. Therefore, it can be said that only using number of parameters to describe generalization and training capabilities is not the best idea.

5.

As the depth of the network increases, e.g. from mlp1 cnn5 network depth increases ,in general, both classification performance increases up to certain level then decreases. For generealization, we can say that it always increases according to the data we have right know in those examples.

6. The mlp weights are not interpretable. However, one may interpret what kind of filtering is done by the CNN weights by looking at the plots shown in Figure 10. First layers in general encodes the most basic features like cırbers, edges and so on.

7. The limited experience of the report writer does not allow to extract the information whether the filters are specific to the cases by looking at them. However, principally the certain CNN filters are trained to detect certain features in the input image. Again, first layers encodes the most simplistic features like cırbers and edges. Therefore, the filters are specific to the classes for better distingueshing them.

8. Similarly, it is hard to distinguish between the filters by looking at the weights. The most interpretable ones are from the cnn4 architecture. The filters are more clear in this case.

9. The mlp1 and mlp2 are the simplest architectures that are similar to each other. The mlp2 introduces one more layer. The cnn3, cnn4 and cnn5 are the convolutional neural network architectures. The cnn3 has three layers with three different spatial kernels. The cnn5 has six convolutional layers with fixed spatial dimension, but it is deepeest of all five of the architectures. As having done in the previous sections the mlp2 is performs better than mlp1 by increasing number of parameters. Cnn4 performs better than cnn3 while having similar number parameters and having higher depth. Cnn5 has significantly low number of parameters so it underfits to training set but generalizes quite nicely.

10. I would pick cnn4 since it has the best test accuracy and generalization performance. It seems it can fit the data well and do not overfit in this setting.

4 Question 4

4.1 Question 4.1

As instructed in the homework documentation, the implementation for both RuLU and sigmoid activate models are done. Necessary statistics are recorded. As a result the plot shown in Figure 11 is obtained. Note than the utils.py had to be modified (alpha value edited) to have a proper distinction. Also, Figure 12 illustrates the individual results for each architecture.

4.2 Question 4.2 - Discussions

1. – 2. Gradient behaviour is similar in each architecture if we focus on ReLU activated ones. The values fluctuates around 0 line as expected. However, the sigmoid activated ones have a different behaviour. The gradients are quite smaller. Also as the depth increases the gradients for sigmoid activated ones are gets smaller and quite close to 0. This is due to the vanishing gradient problem. The gradients are smaller and smaller as the depth increases in each leyer pass. Furthermore, for cnn5 case it is almost always 0. It can be said that using sigmoid activation function instead of ReLU is not the best idea for deep networks.

3.

In part 1.2. case the network was only one layer but still, the effect was similar where the gradient values were smaller in Sigmoid activated one. However it is important to note that in order to obtain a similar performance in both of the ReLU and Sigmoid one, learning rate needs to be adjusted where in overall the weight update step yields to similar scales. In this report to have a fair comparison the learning rate was fixed to a small value where ReLU did not blow-up the gradients.

4.

As pointed out in the previous paragraph, this time Sigmoid could have performed better and avoid vanishing the gradients. On the other hand depending on the rates, ReLU could have explode and improperly update the weights.

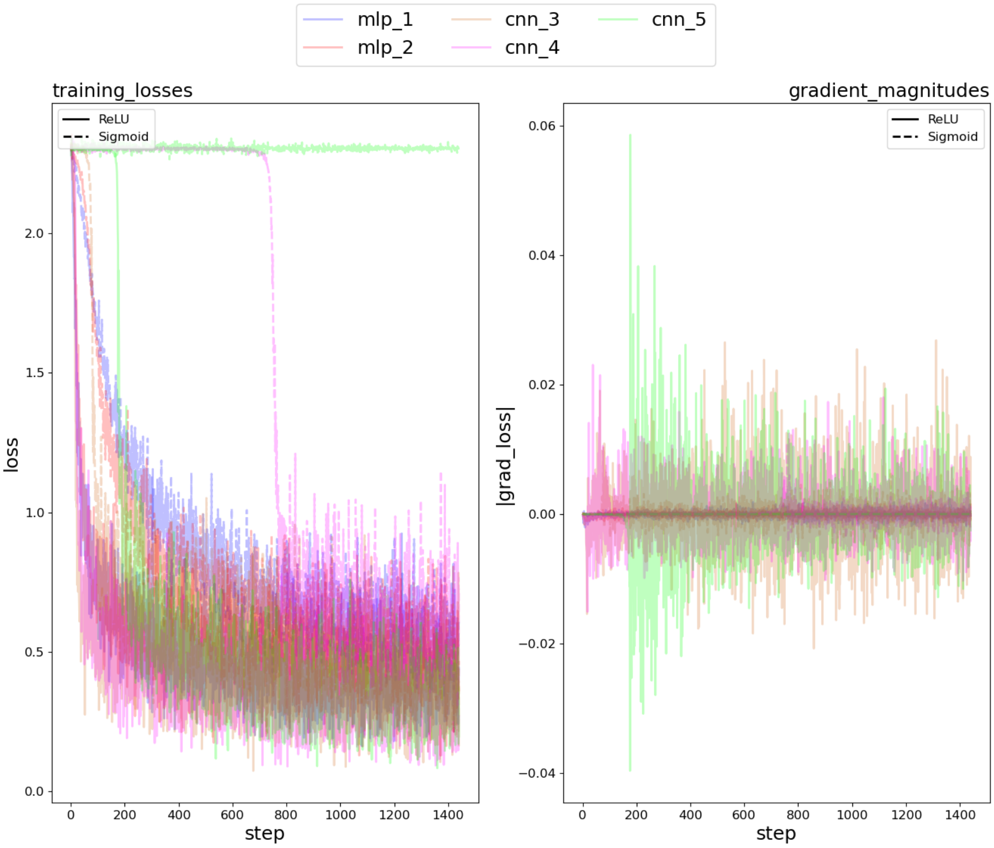
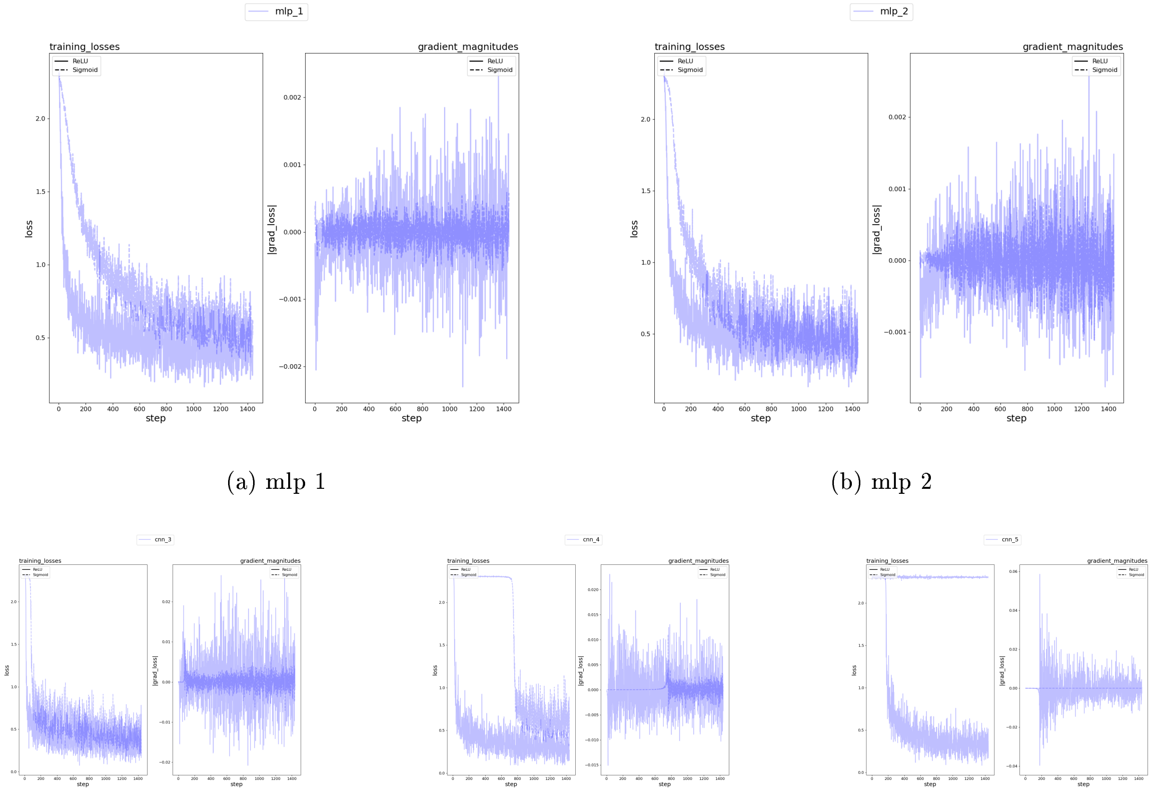


Figure 11: Benchmark of five different architectures trained using ReLU and Sigmoid activation functions.



(a) mlp 1

(b) mlp 2

(c) cnn 3

(d) cnn 4

(e) cnn 5

Figure 12: ReLU vs Sigmoid for each architecture.

5 Question 5

This time validation set is selected as ten percent of the training set.

5.1 Question 5.1

I picked the cnn3 architecture for learning rate experimentation. The learning rates are selected as 0.1, 0.01, 0.001, 0.0001 and 0.00001. The results are shown in Figure 13.

Then scheduling is applied to the learning rate. Figure **??** illustrates the scheduling applied cases. The learning rate is decreased by a factor of 0.1 at each selected epochs.

It should be indicated that, an inconsistent behaviour for dropping learning rate only for once case is observed. For example, when exact same setup is run on different CUDA setups, the results were quite different yet this also can be observed from the initial part of the b). This is repeated multiple times in different computation units. This was assumed to be unsolved problem related random initializations and seed.

5.2 Question 5.2 - Discussions

1.

2.

3.

4.

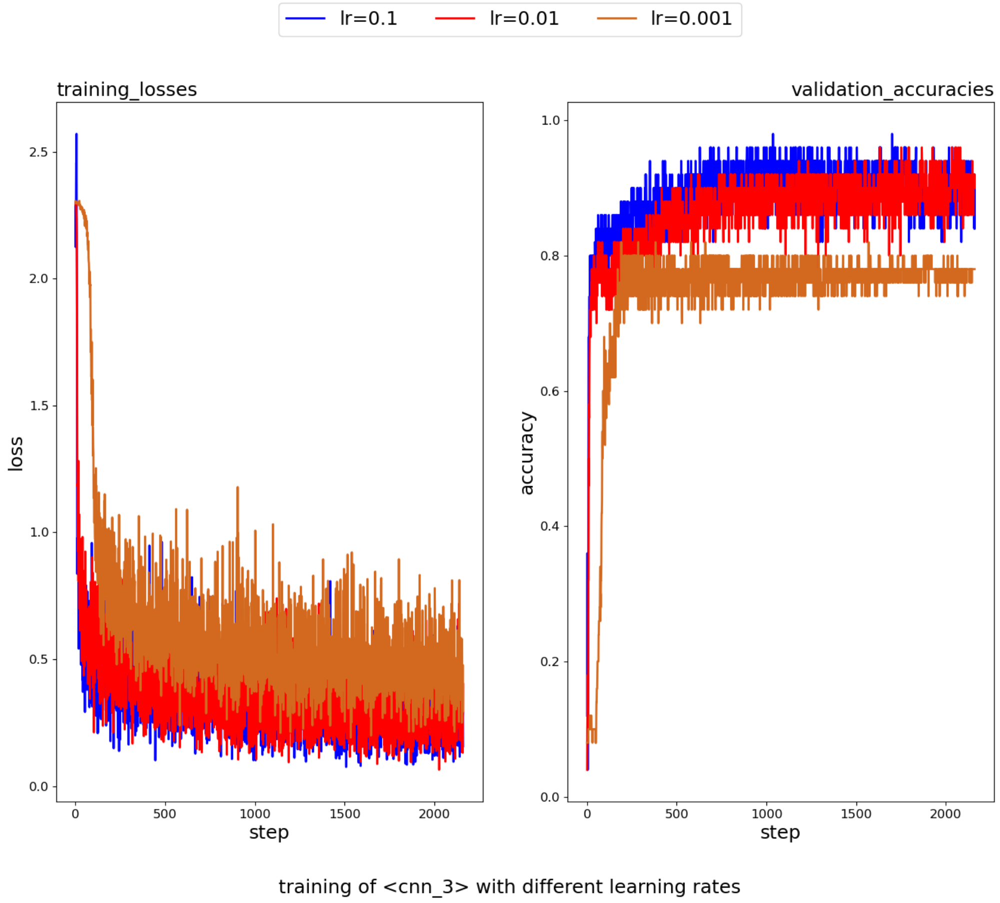


Figure 13: Different learning rate settings on cnn3.

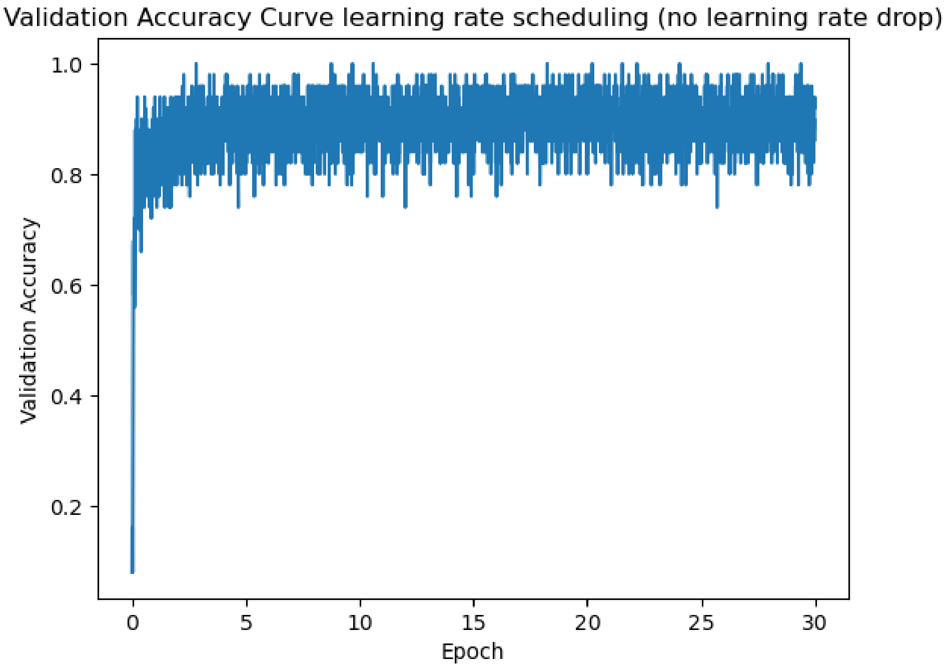


Figure 14: No scheduling

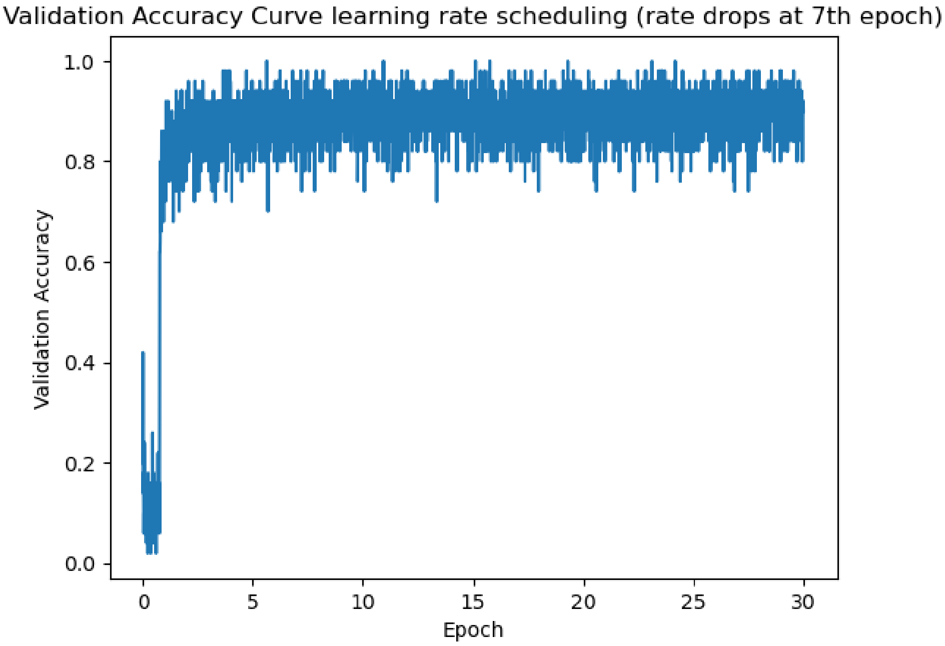


Figure 15: LR dropped at 7th epch

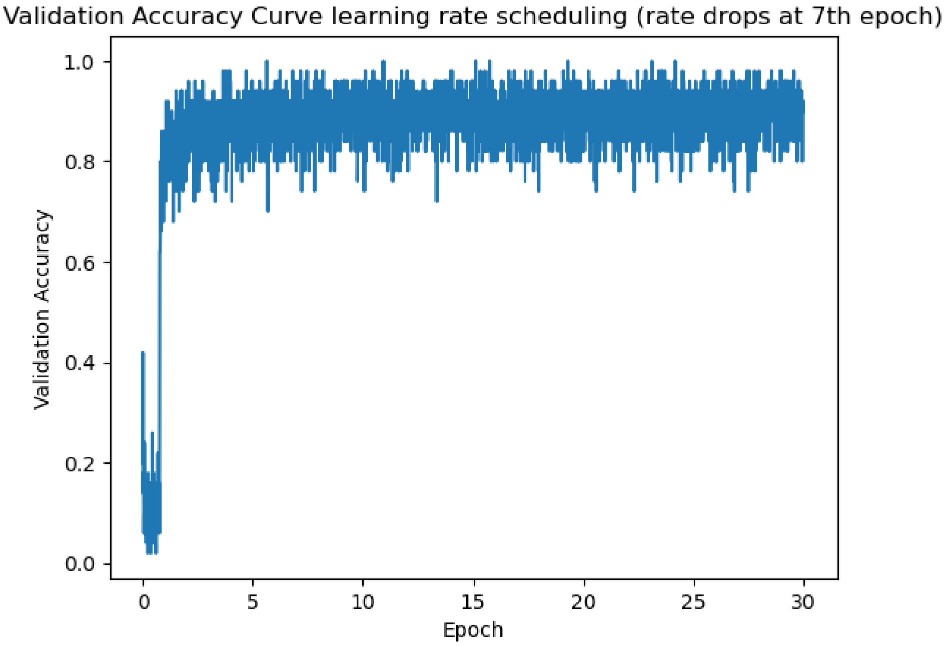


Figure 16: LR dropped at 7th epch

6 References

References

[1] M. Stimberg, R. Brette, and D. F. Goodman, “Brian 2, an intuitive and efficient neural simulator,” *eLife*, vol. 8, p. e47314, Aug. 2019.

[2] R. Brette, M. Rudolph-Lilith, N. T. Carnevale, M. L. Hines, D. Beeman, J. M. Bower, M. Diesmann, A. Morrison, P. H. Goodman, F. C. Harris, M. Zirpe, T. Natschläger, D. Pecevski, B. Ermentrout, M. Djurfeldt, A. Lansner, O. Rochel, T. Viéville, E. B. Müller, A. P. Davison, S. E. Boustani, and A. Destexhe, “Simulation of networks of spiking neurons: A review of tools and strategies,” *Journal of Computational Neuroscience*, vol. 23, pp. 349–398, 2006.

Submitted by Ahmet Akman 2442366 on April 7, 2024.