Homework 1 - Neural Network

Davide Gallon

1 Regression task

1.1 Introduction

The first problem is a regression task. The aim is to train a neural network to approximate an unknown function: $f : \mathbb{R} \to \mathbb{R}$ in which training points are afflicted by some noise

$$\bar{y} = f(x) + \text{noise}$$

I decided to study different type of optimizers, loss functions and activation functions in order to be able to understand what are the ones that fits better in this type of problem. Due to the small dimension of the training dataset, hyperparameters are tuned using a grid search, I also use a k-fold cross-validation strategy.

1.2 Methods

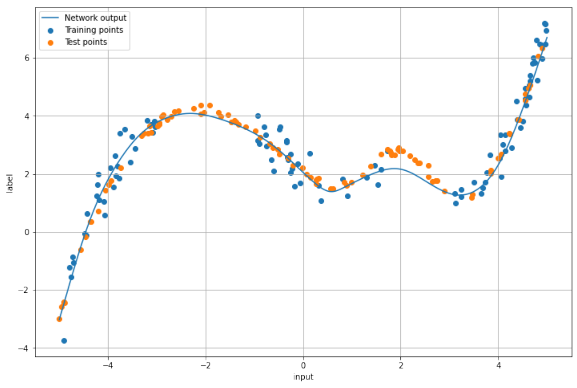
The first part of the code is standard. I define classes Classifier Dataset and Net. I decide to use a Neural Network with two hidden layers since the problem is not so complex.
I study different activation function: Tanh, ReLU, GELU and SiLU; considering the model that i have to fit, i decide to use GELU, a smooth function. In GELU the variable x is multiplied by $\phi(x)$ the Cumulative Distribution Function for Gaussian Distribution.
I investigate also the optimizer, i try to use SGD but also improving learning rate and momentum it doesn't give me results as good as in Adam.
Regarding the loss function, for a regression problem can work well MSE or L1Loss, looking at the documentation i find also the SmoothL1Loss that uses a squared term if the absolute element-wise error falls below a threshold and an L1 term otherwise.
Using the KFold function i am able to split my training dataset in 5 different combination of training* and validation dataset, each of dimension respectively 80 and 20. Thanks to the small dimension of data i implement grid search exploring:

- batch size: 20, 40
- learning rate: 1e-2, 1e-3, 1e-4
- number of neurons in the hidden layers: [64,128], [128,256]
- L2 regularization weight: 0, 5e-05

I keep fix the number of epochs but i use a early stopping strategy, that help me to avoid overfitting. I write the class EarlyStopping that give me the possibility to stop the training if my validation loss is not improving after n epochs; it is very helpful and decrease also computational time.

1.3 Results

In 'main code' for each combination of training* and validation dataset i use cycles 'for' to compare the validation loss changing hyperparameters, saving at the end the ones that obtain the best results in that fold. I get in that way five combination of parameters. See Figure 1.
To be sure that hyperparameters that i select perform well in the entire training dataset and not only for particular data, i take the average between all validation losses computed varying training* and validation dataset defined before. At that point i select parameters which give the lowest of the five means: batch size=40, number of hidden neurons=(128, 256), learning rate=0.01, L^2 regularization weight=0
Remain only to do the final training on entire training dataset, and evaluate the test loss. I have the following results: average training loss=0.1422, test loss=0.03227. The test loss is even lower than train loss, this is strange but could be explained looking at the position of the data, it is clear that test points are regular while training points are affected by some noise.



2 Classification task

2.1 Introduction

Now we have a simple image recognition problem, where the goal is to correctly classify images of handwritten digits (MNIST).
Since this time there are 60000 initial data with only 10 possible feature, i decide to split the dataset in a training dataset (80% on initial dataset) and a validation dataset (20%) without the KFold cross validation strategy.
I divide my notebook in two section, one is for the fully connected neural network, one for

the convolutional neural network. In the first section i investigate using a grid search the best activation function and the number of hidden unit. For both I insert a subsection in which I analyze results with plots and graph.

2.2 Methods

In the Fully connected neural network i use three linear layer of dimension $(28 * 28, N)$, (N, N), $(N, 10)$ (where N is a parameter of the grid search) and a 0.3 dropout probability, i opt for a not so complex network because the image recognition problem of the MNIST is quite simple and result are yet good.
I compare different activation function, SiLU, ReLU, ReLU6, GELU.
In the Convolutional neural network i choose something more interesting, i have two convolutional and max pooling layers and at the end a linear one that produce 10 dimensional output. As regularizator in CNN definition i insert two batch normalization and the dropout, i decide to use only this last one because it gives me a bit better results.

$$\begin{array}{cccccccc} \text{INPUT} & \rightarrow & \text{CONV} & \overset{ReLU}{\rightarrow} & \text{MAXPOOL} & \overset{dropout}{\rightarrow} & \text{CONV} & \overset{ReLU}{\rightarrow} \\ N \times 28 \times 28 \times 1 & & N \times 28 \times 28 \times 16 & & N \times 14 \times 14 \times 16 & & N \times 14 \times 14 \times 32 & \\ \hline \text{MAXPOOL} & \overset{dropout}{\rightarrow} & \text{FLATTEN} & \rightarrow & \text{LINEAR} & & & \\ N \times 7 \times 7 \times 32 & & N \times 1568 & & N \times 10 & & & \end{array}$$

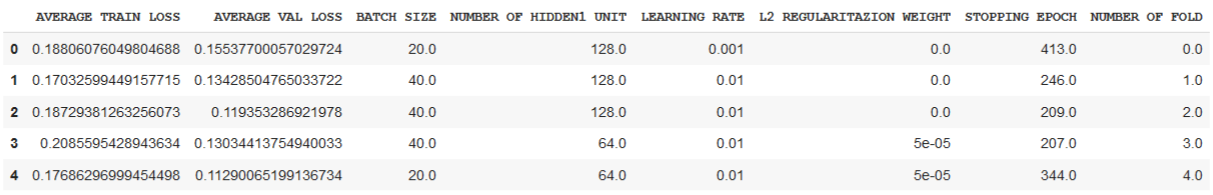
In Convolutional layers i use: kernel size=5, stride=1 and padding=2 while in Max pooling: kernel size=2, stride=2 and padding=0. Loss function remain the Cross entropy loss and optimizer is Adam.

2.3 Results

In the fully connected neural network my grid search obtain the best result with ReLU6 and 256 hidden units. See Figure 2.
Training the best model for 30 epochs i get a average training loss of 0.04221 and a average validation loss of 0.06341, see Figure 3 for the plot losses.
Also the test loss is quite small 0.05854, the accuracy is 98.11% calculated using the confusion matrix.
The optimizer used is Adamax, i do not include it in grid search since Adam and his variation overperform classical SGD, thesis confirmed in my study in which compare losses for different optimizer Figure 4.
I analyze the weights histogram of all layers, as we can see from Figure 5 is clear that i am not overfitting since the range of values is always between $(-0.4, 0, 4)$.
I study different behaviors of the network modifying initial weights, i select: all ones, taken from a uniform distribution between (-1,1), taken from a uniform distribution between $(-\sqrt{\frac{6}{m+n}}, \sqrt{\frac{6}{m+n}})$ (n is the input dimension and m the output), taken from a gaussian distribution with mean and variance (0, 1) and taken from a gaussian distribution with mean and variance $(0, \sqrt{\frac{1}{n}})$. Configurations that depend on layer dimension

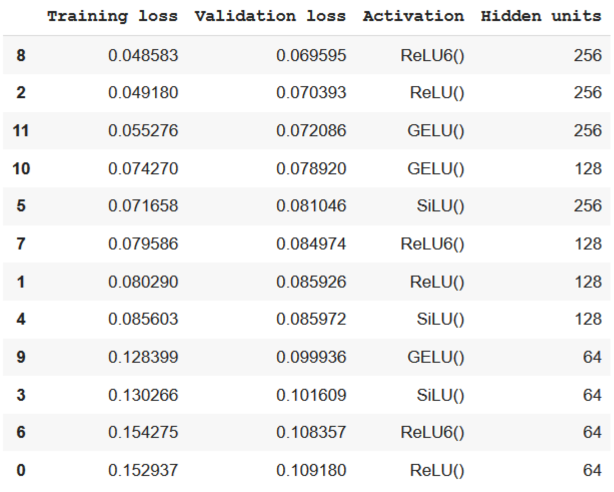
have better performance, similar with that with default initial weight, results are reported in Figure 6 ($x = \sqrt{\frac{1}{n}}$ $y = \sqrt{\frac{6}{m+n}}$).
Regarding activation fields, i plot layer activations of three numbers, two 4 and a 9. The first 4 seems a 9, in fact looking at Figure 7 appears the confusion of the network, we have a lot more neurons activated respect other case.
This section end with the bidimensional space visualization of second last linear layer. I use t-SNE a tool to visualize high-dimensional data. It converts similarities between data points to joint probabilities and tries to minimize the Kullback-Leibler divergence between the joint probabilities of the low-dimensional embedding and the high-dimensional data. See Figure 8.
Now we arrive to CNN, i already explain the architecture. After 30 epochs of training loop i obtain a training loss of 0.01734 and a validation loss of 0.03454. The test loss is 0.02400 and I achieve 99.30% accuracy.
I analyze also the softmax probability for the same 4 that seems a 9 studied before, now the network is more confident, see Figure 9.
In the last subsection i plot the network weights for the two convolutional layer and the activation for a input image, what i get are Figure 10, 11, 12, 13.
In trasformation section i comment the random rotation and the addition of gaussian noise in training data, enabling the rotation I notice a improvement in the accuracy for the fully connected network, while for the random noise there is a little worsening for both networks.

Figure 1: Grid search results



	AVERAGE TRAIN LOSS	AVERAGE VAL LOSS	BATCH SIZE	NUMBER OF HIDDEN1 UNIT	LEARNING RATE	L2 REGULARITAZION WEIGHT	STOPPING EPOCH	NUMBER OF FOLD
0	0.18806076049804688	0.15537700057029724	20.0	128.0	0.001	0.0	413.0	0.0
1	0.17032599449157715	0.13428504765033722	40.0	128.0	0.01	0.0	246.0	1.0
2	0.18729381263256073	0.119353286921978	40.0	128.0	0.01	0.0	209.0	2.0
3	0.2085595428943634	0.13034413754940033	40.0	64.0	0.01	5e-05	207.0	3.0
4	0.17686296999454498	0.11290065199136734	20.0	64.0	0.01	5e-05	344.0	4.0

Figure 2: Grid search results



	Training loss	Validation loss	Activation	Hidden units
8	0.048583	0.069595	ReLU6()	256
2	0.049180	0.070393	ReLU()	256
11	0.055276	0.072086	GELU()	256
10	0.074270	0.078920	GELU()	128
5	0.071658	0.081046	SiLU()	256
7	0.079586	0.084974	ReLU6()	128
1	0.080290	0.085926	ReLU()	128
4	0.085603	0.085972	SiLU()	128
9	0.128399	0.099936	GELU()	64
3	0.130266	0.101609	SiLU()	64
6	0.154275	0.108357	ReLU6()	64
0	0.152937	0.109180	ReLU()	64

Figure 3: Plot losses

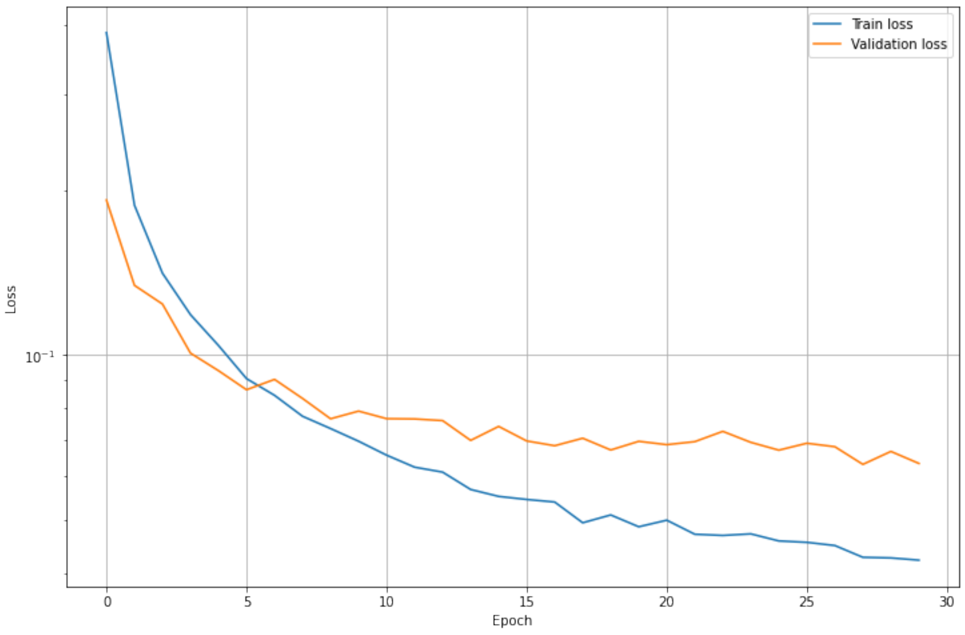


Figure 4: Training loss taken with different optimizer

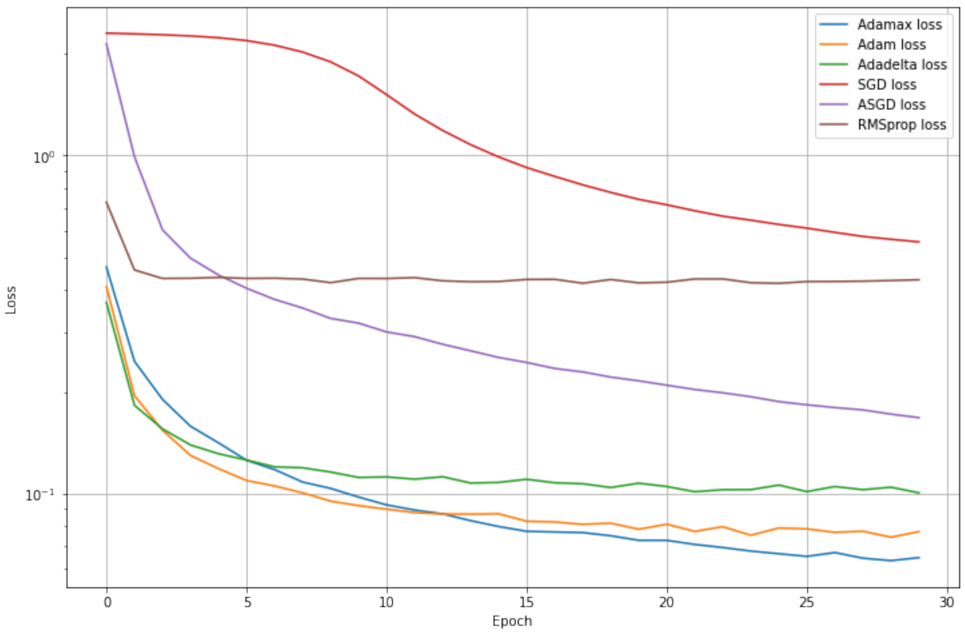


Figure 5: Histogram weight of layers

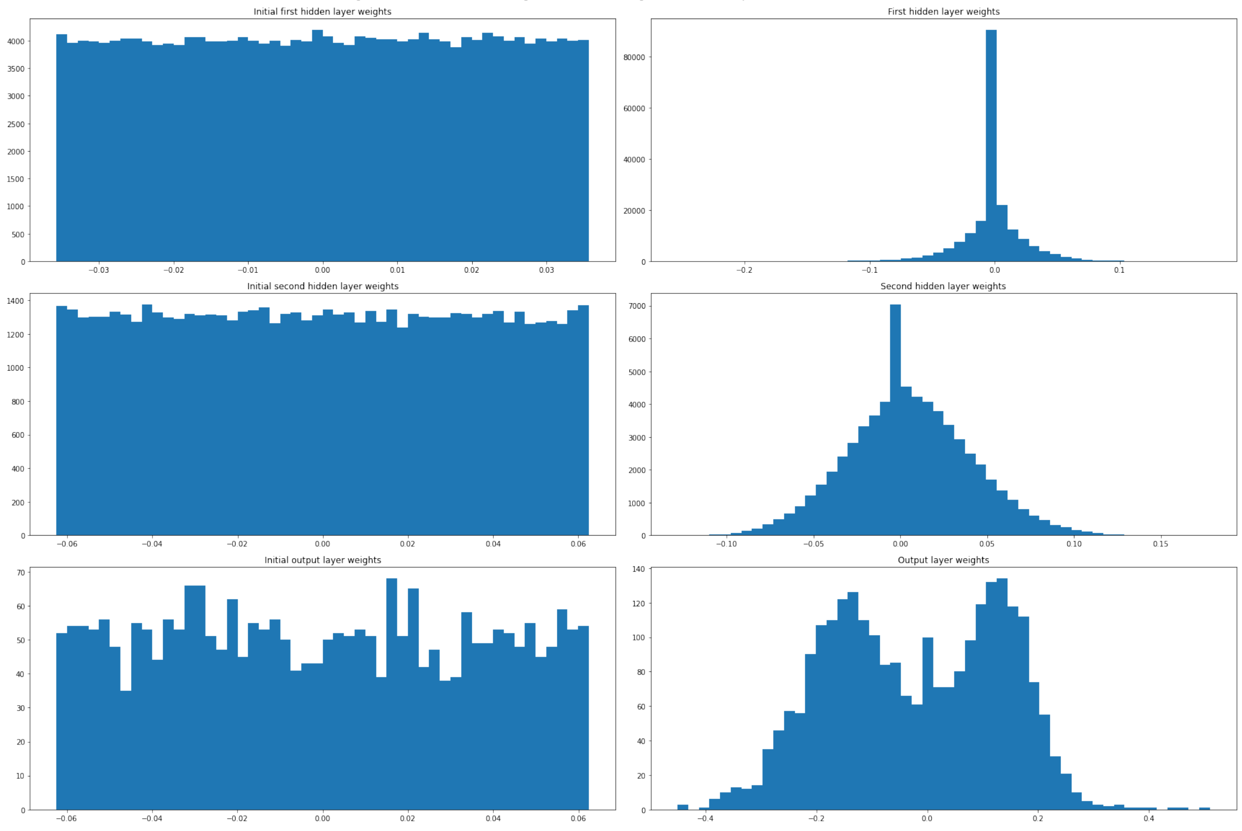


Figure 6: Network behavior with different initial weights

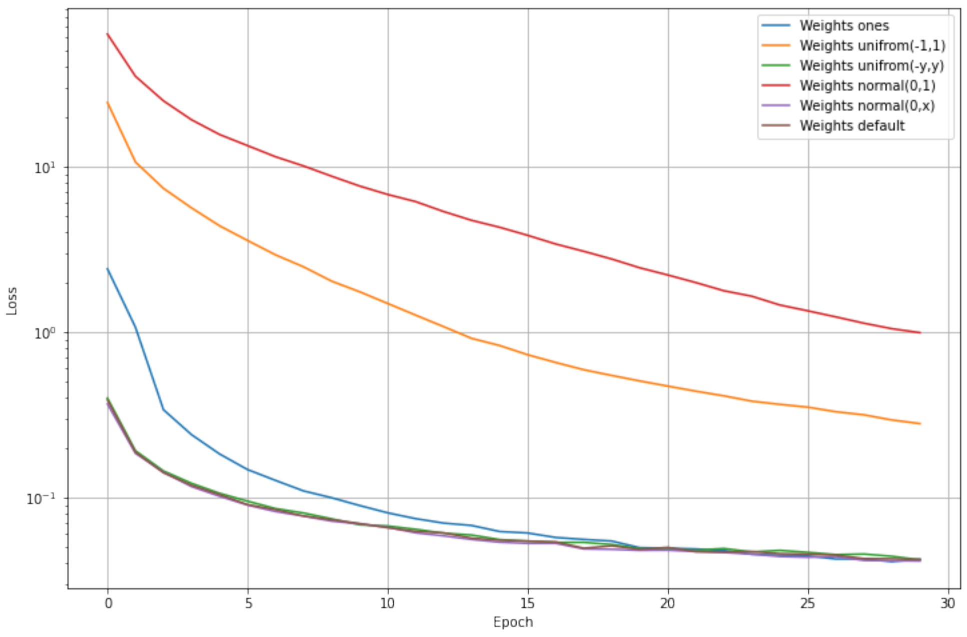


Figure 7: Layer activation

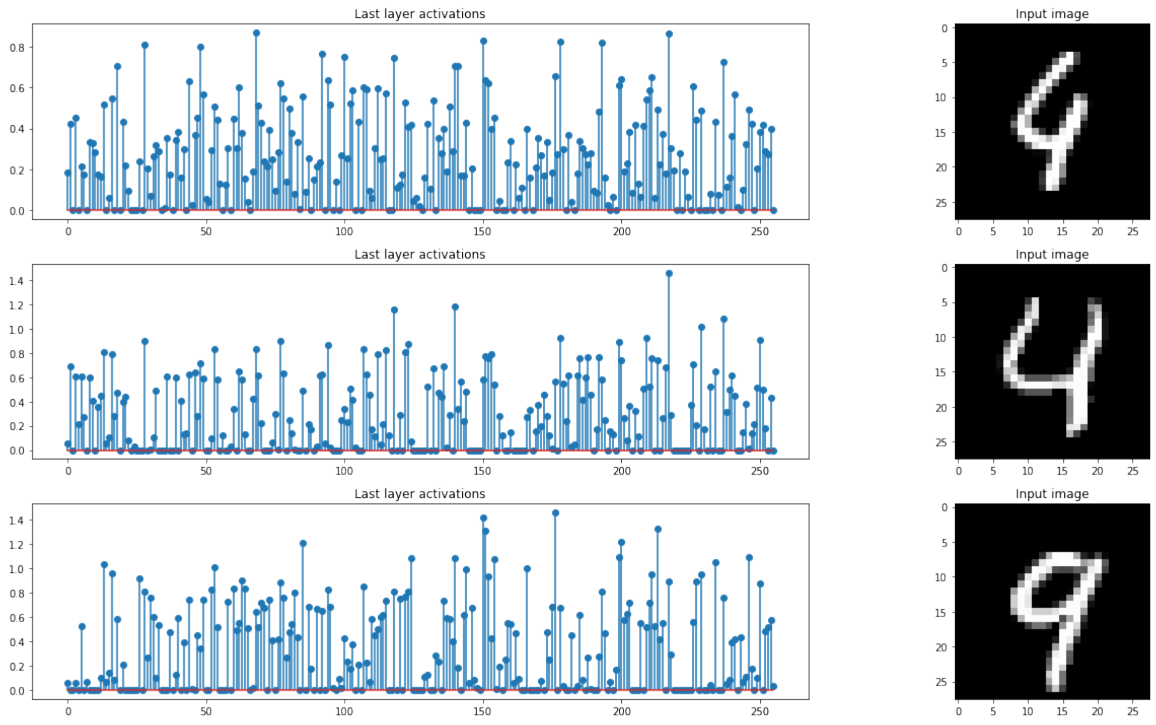


Figure 8: Bidimensional space visualization of feature space

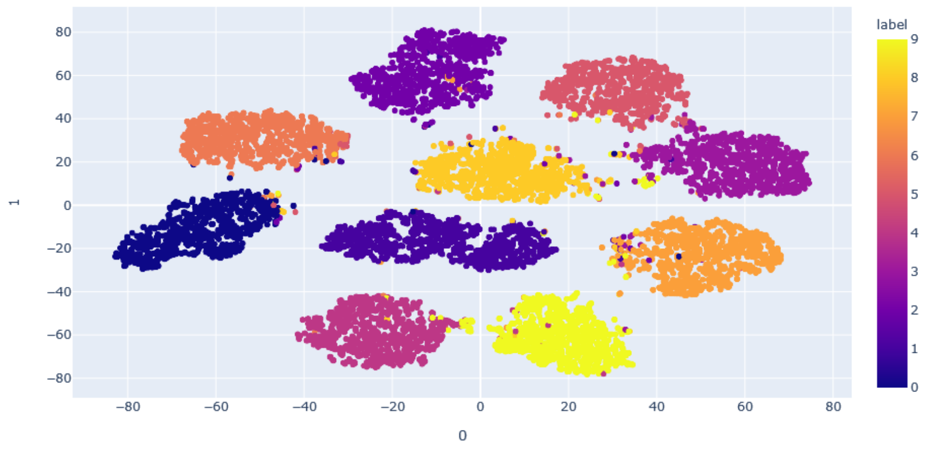


Figure 9: Softmax probability output

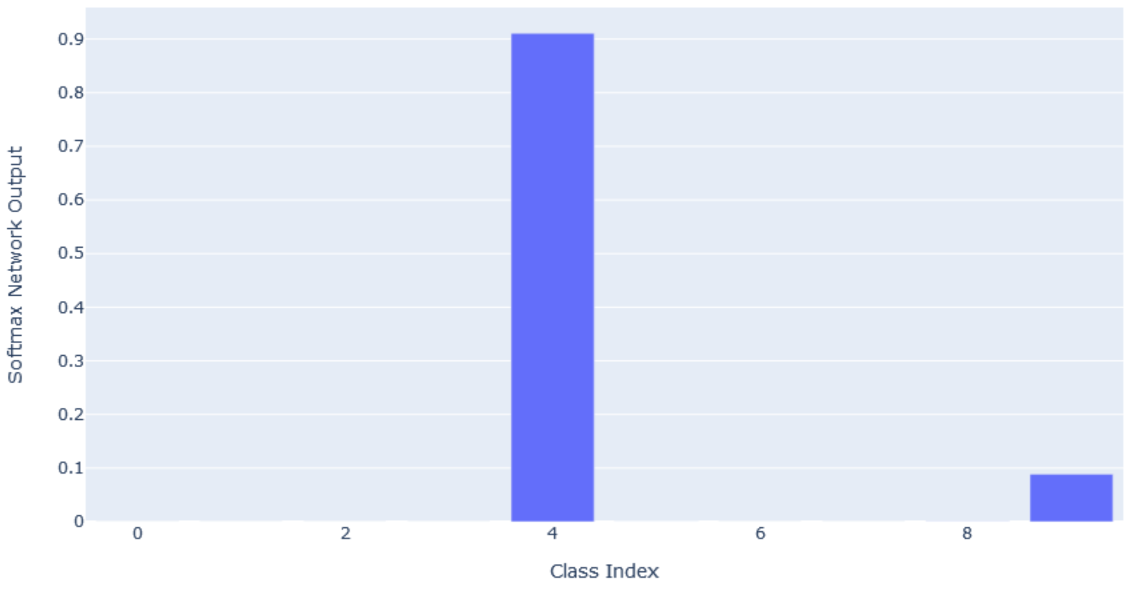


Figure 10:

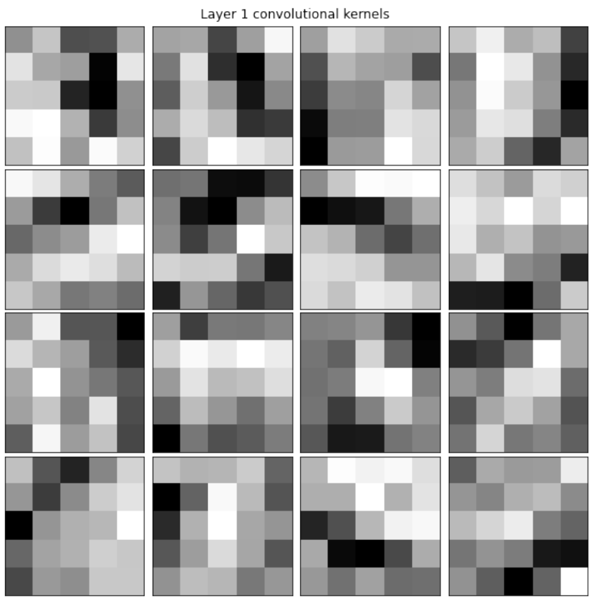


Figure 11:

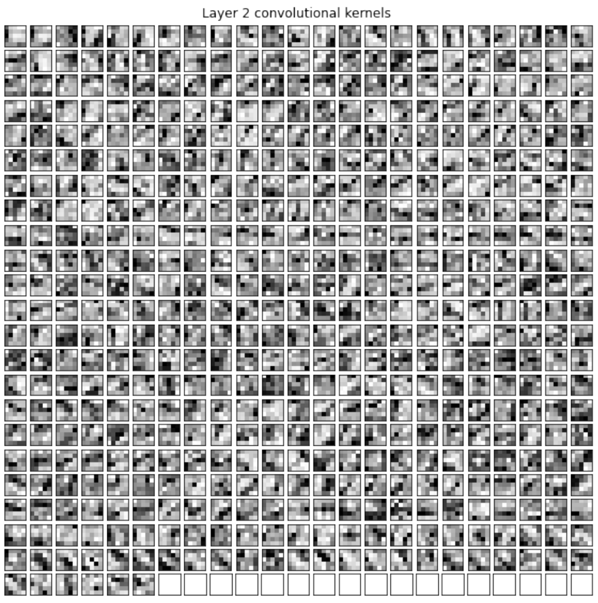


Figure 12:

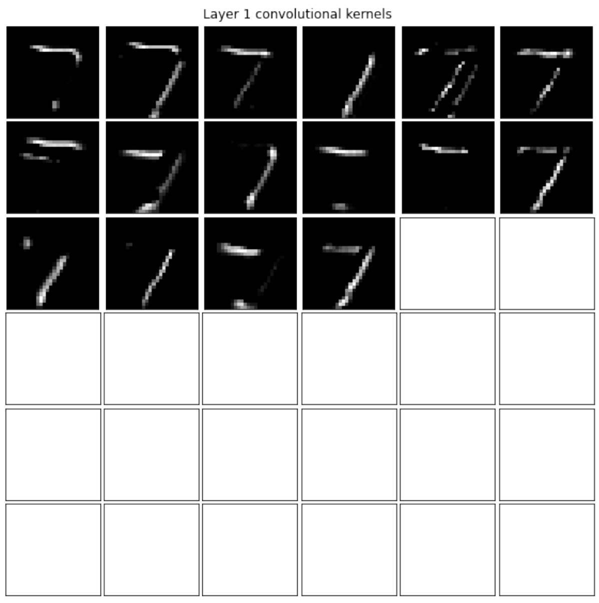
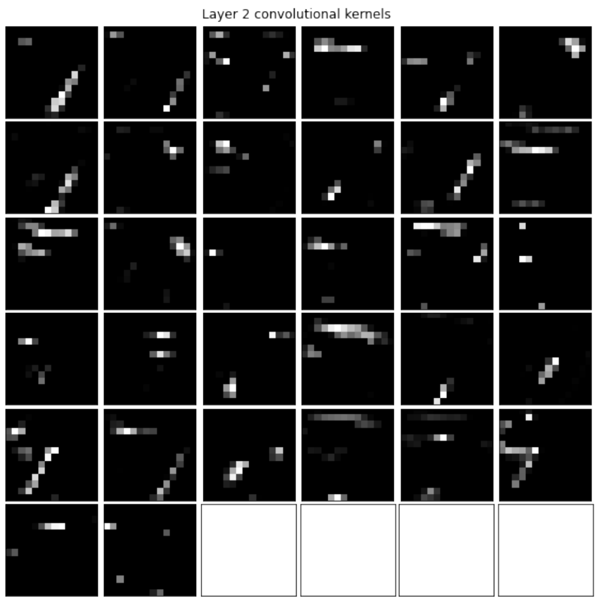


Figure 13:


Layer 2 convolutional kernels