Machine Learning

Homework 2

Alexandros Kalimeris ds1200003
Antonios Papadakis ds1200009
Stylianos Kotzastratis ds1200006

DSIT 2020

Exercise 1

Question A

Having N number of data of K dimensions, and M neurons per layer r:

$$E = \frac{1}{2}\sum_{i}^{N}\sum_{j}^{K}(t_{ij} - y_{ij}^{(r)})^2 \text{ with } y_{ij}^{(r)} = f(\nu_{i\mu}^{(r)}) \text{ and } \nu_{i\mu}^{(r)} = \sum_{\mu}^{M} w_{j\mu}^{(r)} y_{ij}^{(r-1)}$$

Omitting the (r) notation for simplicity we use the chain rule:

$$\frac{\partial E}{\partial w_{j\mu}} = \frac{\partial E}{\partial y_{ij}} \cdot \frac{\partial y_{ij}}{\partial \nu_{i\mu}} \cdot \frac{\partial \nu_{i\mu}}{\partial w_{j\mu}} \text{ where } \frac{\partial E}{\partial y_{ij}} = y_{ij}^{(r)} - t_{ij} \text{ and } \frac{\partial \nu_{i\mu}}{\partial w_{j\mu}} = y_{ij}^{(r-1)}.$$

The range of $\frac{\partial E}{\partial y_{ij}}$ is $[0, 1]$ since we have a softmax final layer, and the range of $\frac{\partial \nu_{i\mu}}{\partial w_{j\mu}}$ is the range of the respective activation function that is used in layer $(r - i)$ where, $i = 1, 2, ...r - 1$.

For **Relu** $f(x) = max(0, x)$, so

$$\frac{\partial y_{ij}}{\partial \nu_{i\mu}} = \begin{cases} 1 & \text{if } \nu_{i\mu} \geq 0 \\ 0 & \text{if } \nu_{i\mu} < 0 \end{cases} \text{ so: } \frac{\partial E}{\partial w_{j\mu}^{(r)}} = \begin{cases} (y_{ij}^{(r)} - t_{ij}) y_{ij}^{(r-1)} & \text{if } \nu_{i\mu}^{(r)} \geq 0 \\ 0 & \text{if } \nu_{i\mu}^{(r)} < 0 \end{cases}$$

The range of Relu is $[0, +\infty]$, which makes the range of the gradient $\frac{\partial E}{\partial w_{j\mu}^{(r)}}$ to be also $[0, +\infty]$.

For **Hyperbolic Tangent** (tanh) we have $f(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})} = \frac{sinh(x)}{cosh(x)}$. The derivative of the tanh function is $\frac{\partial f(x)}{\partial x} = 1 - f^2(x)$, so:

$$\frac{\partial y_{ij}}{\partial \nu_{i\mu}} = 1 - \frac{(e^{\nu_{i\mu}} - e^{-\nu_{i\mu}})^2}{(e^{\nu_{i\mu}} + e^{-\nu_{i\mu}})^2} \text{ and}$$

$$\frac{\partial E}{\partial w_{j\mu}^{(r)}} = (y_{ij}^{(r)} - t_{ij}) y_{ij}^{(r-1)} \left(1 - \frac{(e^{\nu_{i\mu}^{(r)}} - e^{-\nu_{i\mu}^{(r)}})^2}{(e^{\nu_{i\mu}^{(r)}} + e^{-\nu_{i\mu}^{(r)}})^2}\right)$$

The range of $\tanh(x)$ is $[-1, 1]$, while the range of $\frac{\partial y_{ij}}{\partial \nu_{i\mu}}$ is $[0, 1]$. This makes the range of the gradient $\frac{\partial E}{\partial w_{j\mu}^{(r)}}$ to be $[-1, 1]$.

For **Sigmoid,** $f(x) = \frac{1}{1 + e^{-x}}$, and the derivative of the sigmoid function is $\frac{\partial f(x)}{\partial x} = f(x)(1 - f(x))$, so:

$$\frac{\partial y_{ij}}{\partial \nu_{i\mu}} = \frac{1}{1 + e^{-\nu_{i\mu}}} \left(1 - \frac{1}{1 + e^{-\nu_{i\mu}}}\right) \text{and}$$

$$\frac{\partial E}{\partial w_{j\mu}^{(r)}} = (y_{ij}^{(r)} - t_{ij}) y_{ij}^{(r-1)} \frac{1}{1 + e^{-\nu_{i\mu}^{(r)}}} \left(1 - \frac{1}{1 + e^{-\nu_{i\mu}^{(r)}}}\right)$$

The range of sigmoid is $[0, 1]$, while the range of $\frac{\partial y_{ij}}{\partial \nu_{i\mu}}$ is $[0, 0.25]$. Hence the range of $\frac{\partial E}{\partial w_{j\mu}^{(r)}}$ is also $[0, 0.25]$.

Question B

Below we report the test scores requested, including the loss and accuracy, for 50 epochs. The code can be seen in file $ex1.py$

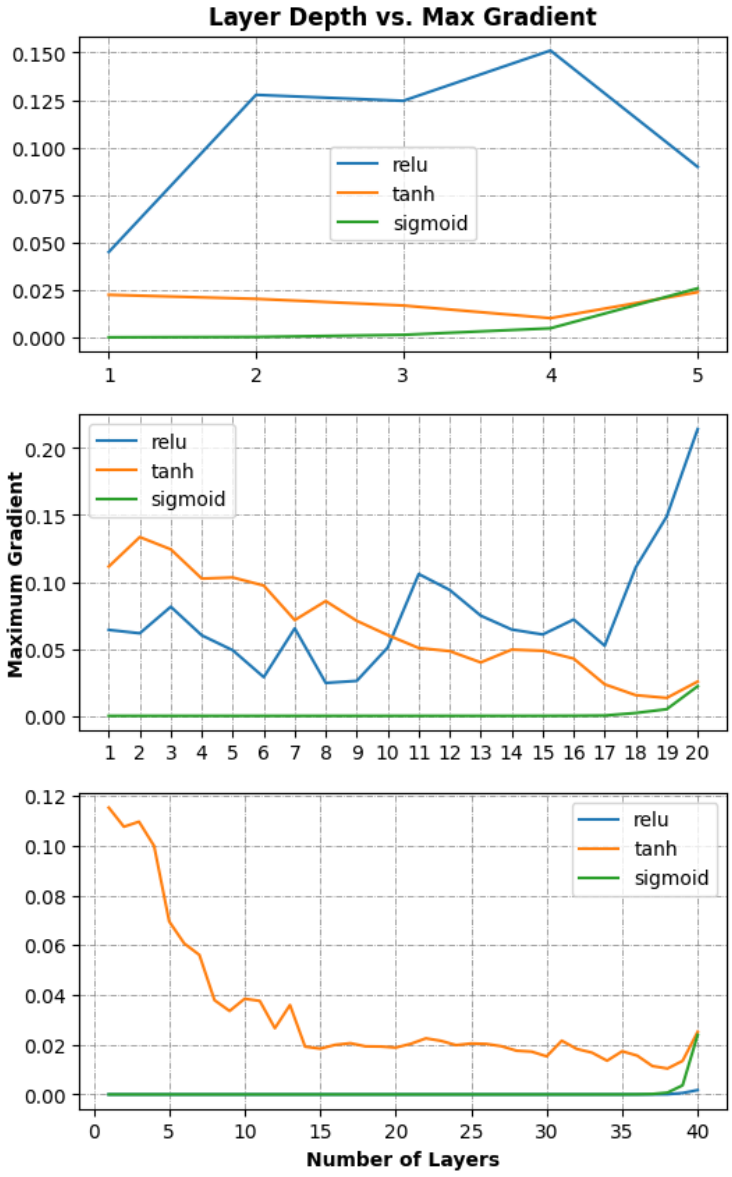
#Layers	Loss			Accuracy		
	relu	tanh	sigmoid	relu	tanh	sigmoid
5	**0.119**	**0.11**	0.956	**0.964**	**0.966**	0.697
20	0.2	0.2	2.301	0.96	0.954	0.113
40	1.06	0.25	2.302	0.629	0.947	0.113

We can observe that the best result for each model is reported for the 5 utilized layers, while the Sigmoid function shows the lowest performance. Beyond

that layer count, the model must be too deep and therefore not able to generalize well. Lastly, we can see a very steep worsening for the deeper networks that use the Sigmoid.

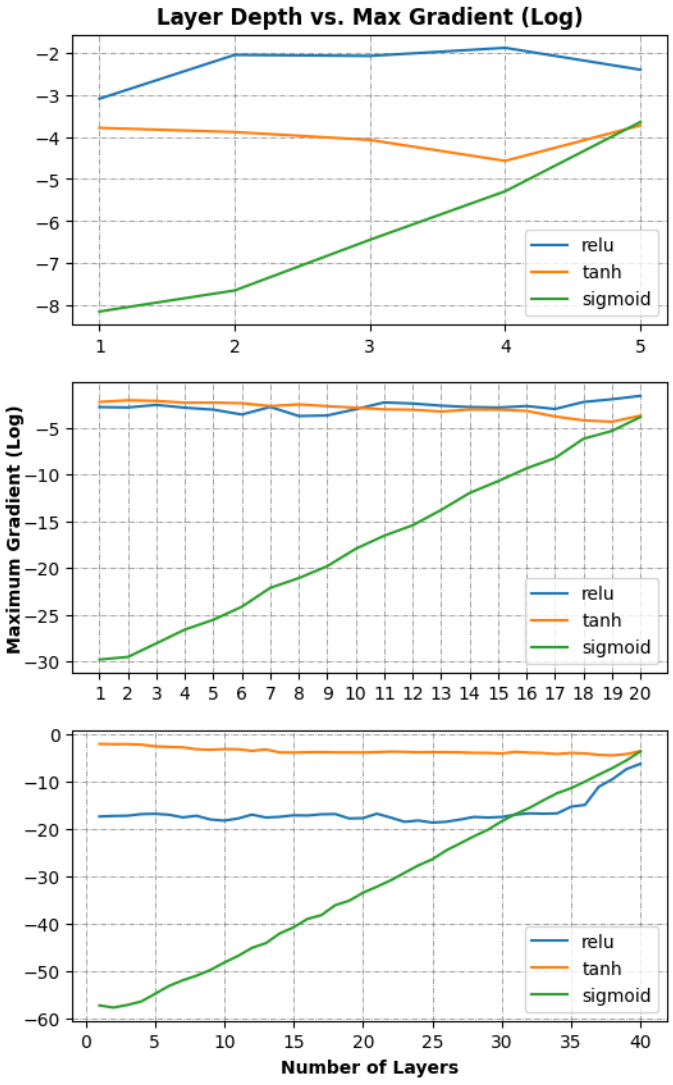
Question C

Below we present what is the maximum value of the gradient per layer given a specific batch after 3 training epochs, and given that we use each of the three activation functions. For these plots we didn't include the gradients of the bias, but having tried the same plots for it in our runs, the behavior doesn't show any particular differences.



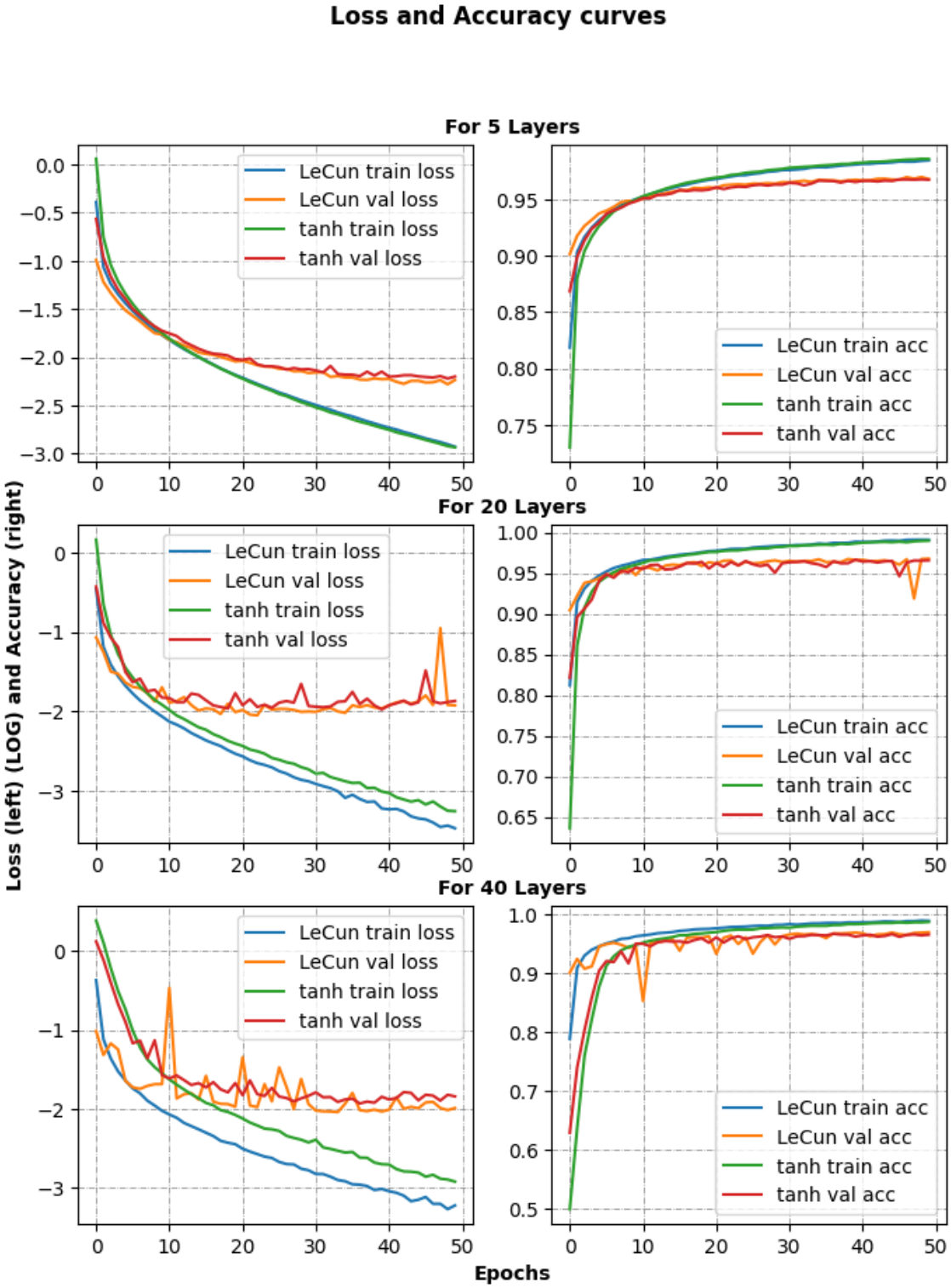
The graphs above are enlightening, as they explain the table of the previous question fully. As we see in the above plots, the Sigmoid function converges to zero in around 3 back-propagation steps, something that the other 2 activation functions avoid. This is the known problem of the Vanishing Gradient, and occurs when an activation function (like Sigmoid) squishes its large input space in a much smaller one, effectively nullifying the gradient that is sent backwards. This difference in this squishing was clearly visible already, if we compare the ranges of the gradients, as we derived them in the previous question.

Lastly, the small performance decrease of the Relu for 40 layers is also depicted in these plots, as it appears to also converge to zero. Yet, Relu doesn't suffer from the vanishing gradient problem, but from small gradient values which are by no means as small as the Sigmoid ones. This can be cleared up if we apply the logarithm function to the y axis of the previous plots. We can see that both the Relu and Tanh derived gradients behave rather steadily, while the Sigmoid ones are decreasing exponentially fast towards zero, which in turns effectively leads the training process to a halt.



Question D

Below we can see the learning curves for train and validation data, for the two functions. Namely the loss function (log) result and the accuracy, for 50 epochs. The curves appear to be rather similar between the two functions.



For the backprogation equations, we follow similar steps with Question A, with only differences in the loss and activation functions. We have:

$$E = -\sum_{i}^{N}\sum_{j}^{K} t_{ij}\log(y_{ij}^{(r)}) \text{ with } y_{ij}^{(r)} = f(\nu_{i\mu}^{(r)}) \text{ and } \nu_{i\mu}^{(r)} = \sum_{\mu}^{M} w_{j\mu}^{(r)} y_{ij}^{(r-1)}$$

Using the chain rule:

$$\frac{\partial E}{\partial w_{j\mu}} = \frac{\partial E}{\partial y_{ij}} \cdot \frac{\partial y_{ij}}{\partial \nu_{i\mu}} \cdot \frac{\partial \nu_{i\mu}}{\partial w_{j\mu}} \text{ where } \frac{\partial E}{\partial y_{ij}} = -\frac{t_{ij}}{y^{(r)}} \text{ and } \frac{\partial \nu_{i\mu}}{\partial w_{j\mu}} = y_{ij}^{(r-1)}.$$

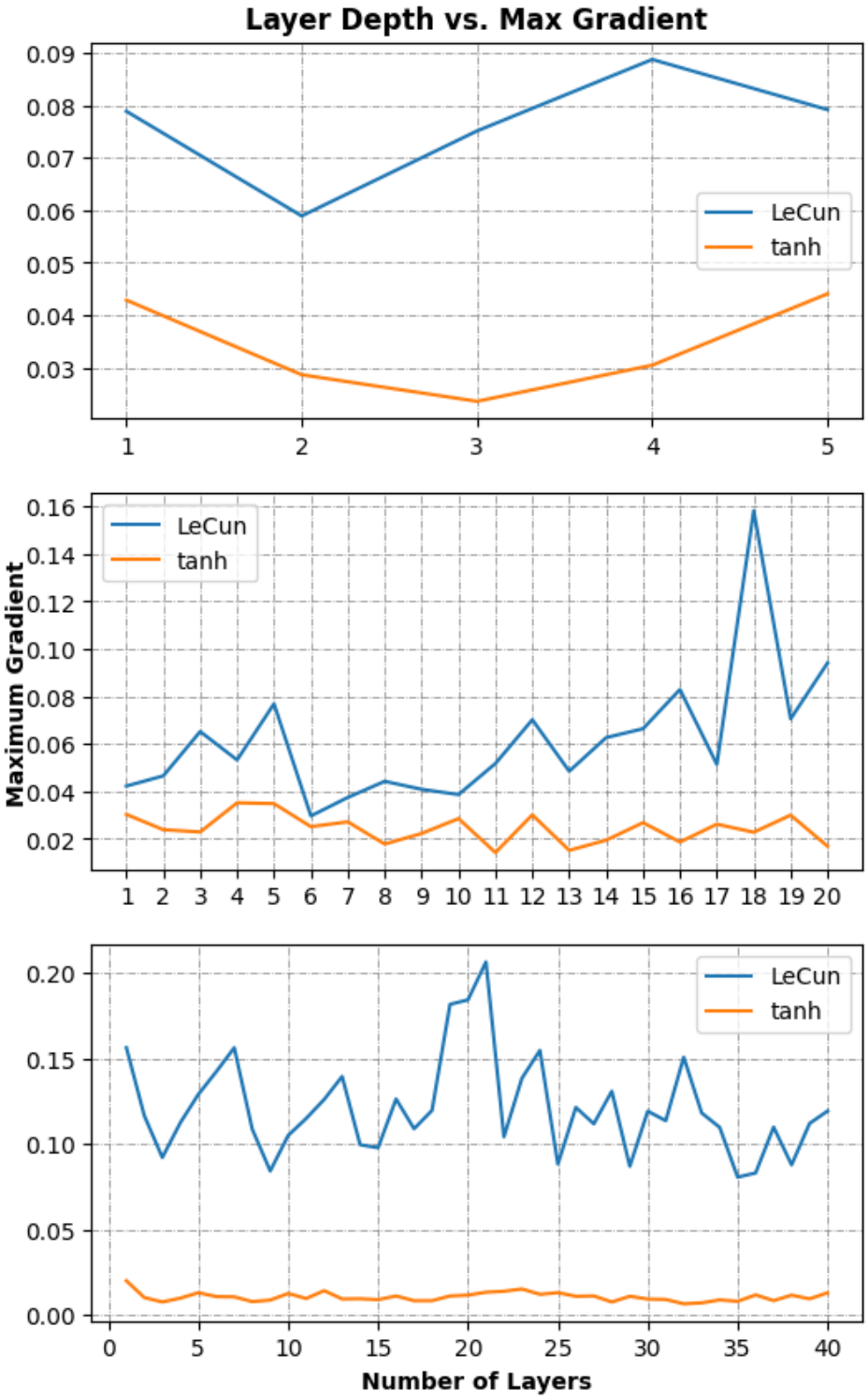
For the given **LeCun** activation function: $\frac{\partial f(x)}{\partial x} = 1.1439(1 - \tanh^2(\frac{2}{3}x)) + 0.01$ so,

$$\frac{\partial y_{ij}}{\partial \nu_{i\mu}} = 1.1439\left(1 - \frac{(e^{0.66\nu_{i\mu}} - e^{-0.66\nu_{i\mu}})^2}{(e^{0.66\nu_{i\mu}} + e^{-0.66\nu_{i\mu}})^2}\right) + 0.01 = 1.1539 - 1.1439\frac{(e^{0.66\nu_{i\mu}} - e^{-0.66\nu_{i\mu}})^2}{(e^{0.66\nu_{i\mu}} + e^{-0.66\nu_{i\mu}})^2}$$

$$\text{and } \frac{\partial E}{\partial w_{j\mu}^{(r)}} = -\frac{t_{ij}}{y^{(r)}} y_{ij}^{(r-1)}\left(1.1539 - 1.1439\frac{(e^{0.66\nu_{i\mu}^{(r)}} - e^{-0.66\nu_{i\mu}^{(r)}})^2}{(e^{0.66\nu_{i\mu}^{(r)}} + e^{-0.66\nu_{i\mu}^{(r)}})^2}\right)$$

As for the range, $\frac{\partial E}{\partial y_{ij}}$ has range $[-\infty, 0]$ as it is the negative of a fraction where both terms have a range of $[0, 1]$ (since we have a softmax activation function for the last layer). $\frac{\partial y_{ij}}{\partial \nu_{i\mu}}$ has range $[0.01, 1.1539]$, as $\lim_{x\to\pm\infty} tanh^2(x) = 1$ and thus the second term of $\frac{\partial y_{ij}}{\partial \nu_{i\mu}}$ is in the range of $[0, 1.1439]$. Finally, $\frac{\partial \nu_{i\mu}}{\partial w_{j\mu}}$ has range $[-\infty, +\infty]$ as this is the range of the LeCun activation function due to the $0.01x$ term. Therefore as $\frac{\partial \nu_{i\mu}}{\partial w_{j\mu}}$ dominates, the range of $\frac{\partial E}{\partial w_{j\mu}^{(r)}}$ is $[-\infty, +\infty)$.

Lastly, below we can see the max gradients per layer depth, for an untrained model paired with each of the two activation functions. The gradient of the tanh function is consistently smaller and seemingly more stable than the LeCun case.

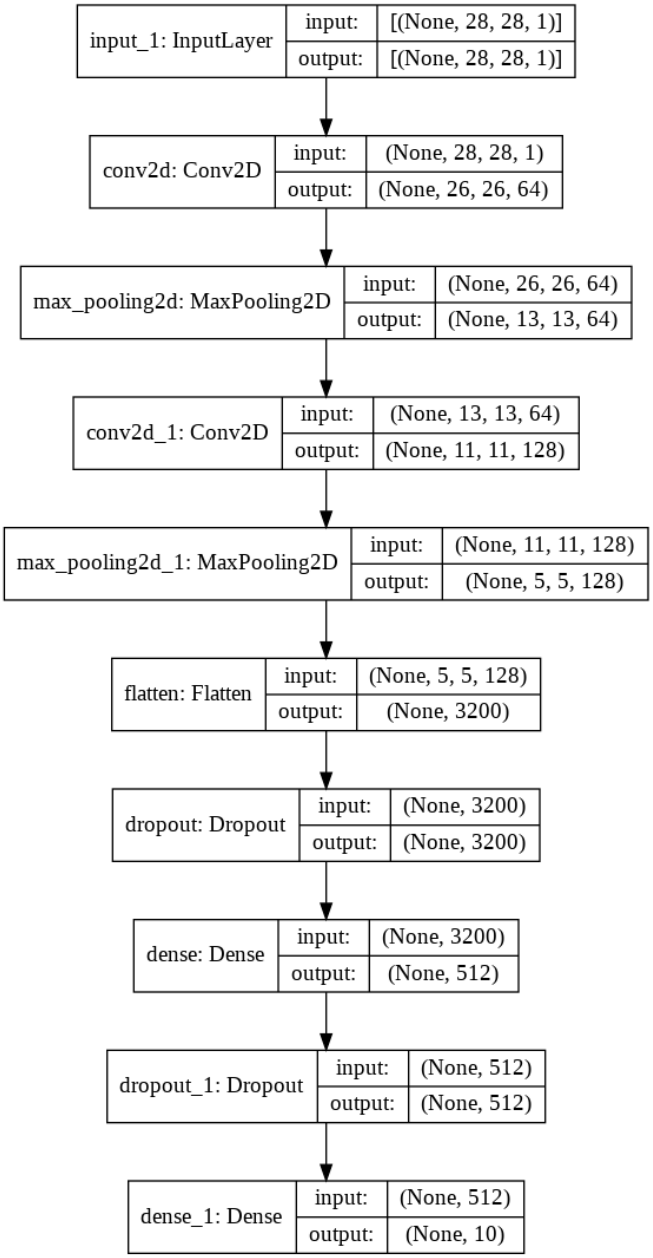

Layer Depth vs. Max Gradient
LeCun
tanh
Maximum Gradient
Number of Layers

Exercise 2

(implemented at : dsit_ml_ass2_2.ipynb)

Question A

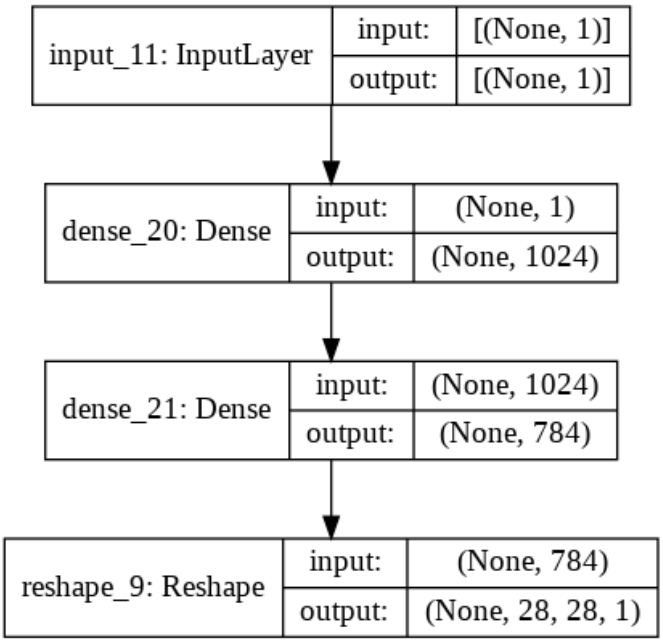
In this task, we were asked to implement a model that can perform classification on the MNIST dataset. For that purpose we used the following CNN architecture (we will refer to this model as C):



After a training of 20 epochs on the MNIST training set we got a quite satisfactory accuracy of 99.39% on the MNIST test set and then we proceeded to the next subtasks.

Question B

We created another model that represents the f generator described in the assignment. The architecture of that model can be seen below:



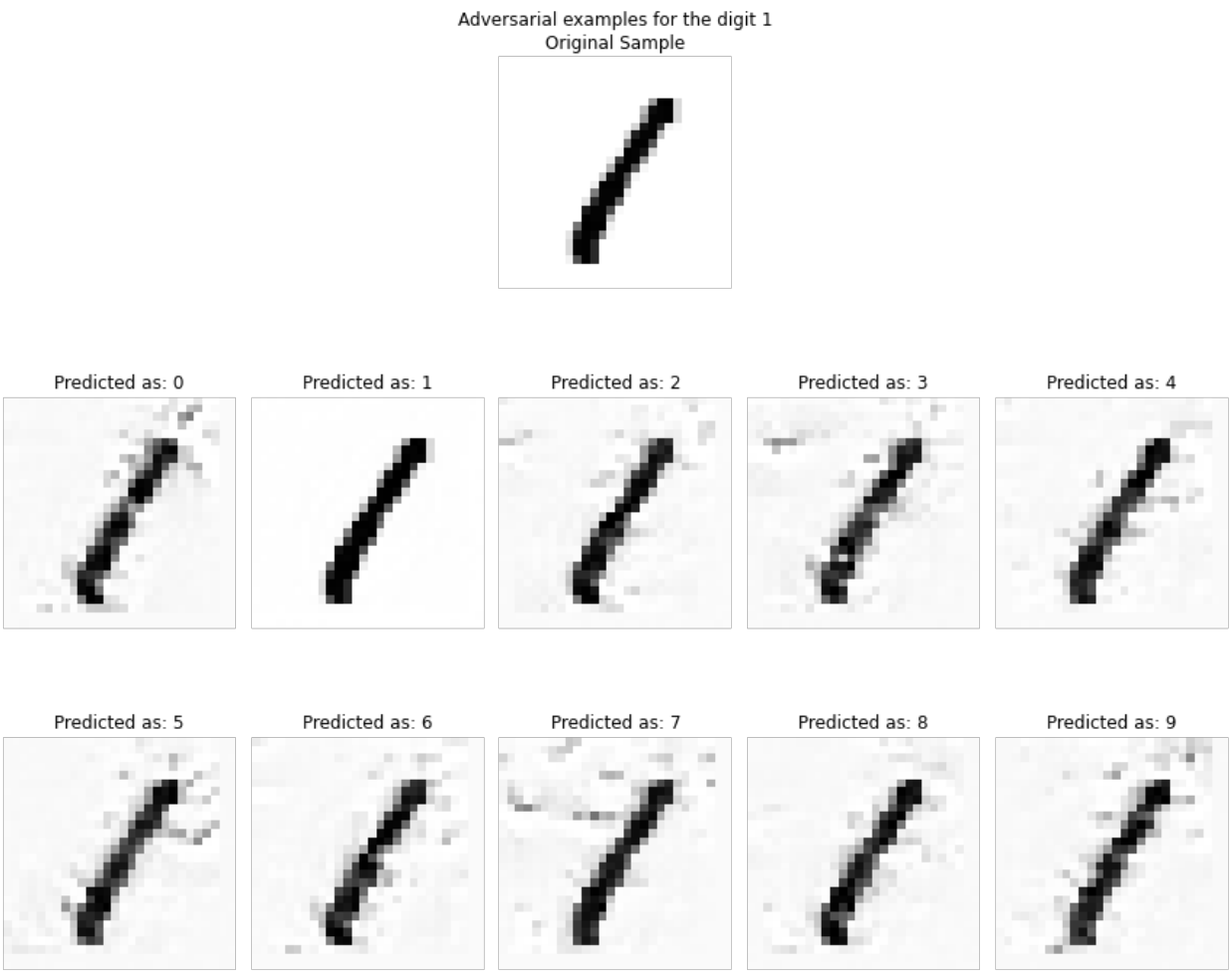
In order to create the adversarial examples as described in the assignment, we combined the two models as $C \circ f$ giving the output of f as the input of C and keeping the combined model with the weights of C frozen.

We were asked to create adversarial examples in the following way: pick our favourite digits and then create examples such that the original model C missclassifies them in every other class. For the sake of this exercise, we picked the digits 0, 1, 2, 3, 4 and performed the procedure described in the assignment.

By training the combined model with the 2 step process described in the exercise, for a sufficient amount of iterations we generated the adversarial

examples as asked. For the sake of simplicity, we will demonstrate the results for one digit below, the results for the other digits will be included in the assignment files.

The adversarial examples we got for the digit 1:



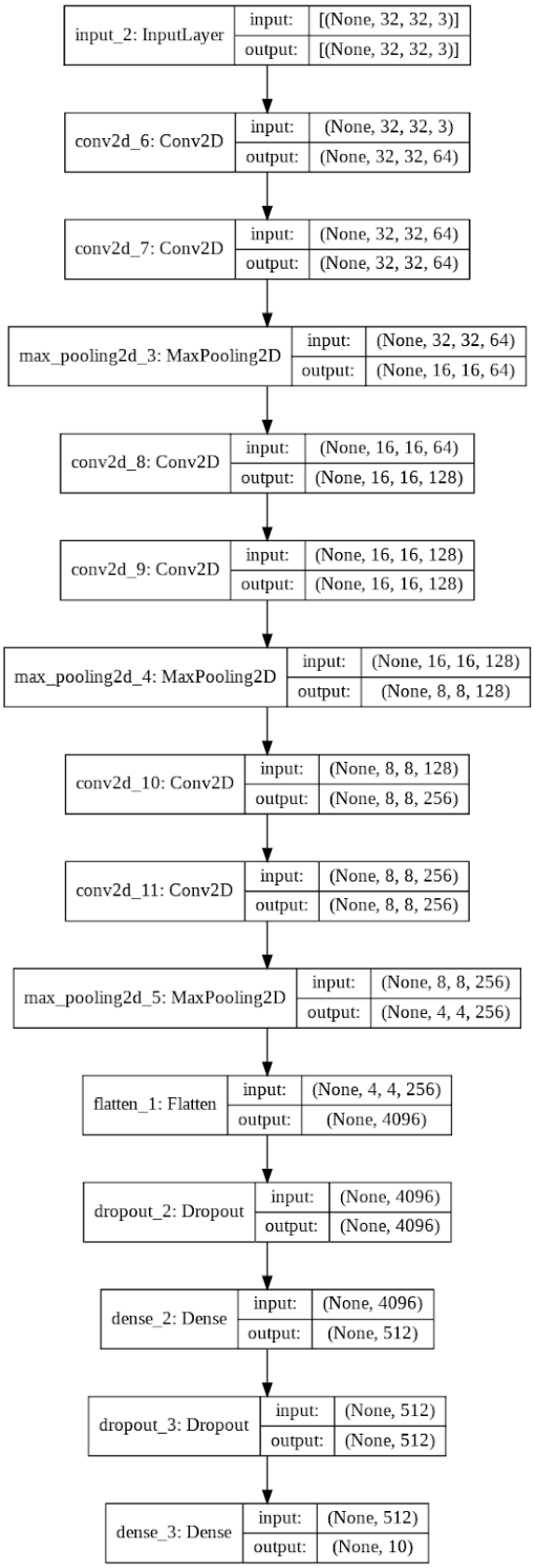
As we can see, the classifier model we created at **A** missclassifies the digit to the wanted class every time due to the added noise. However, for a human the correct choice would still be obvious.

Question C

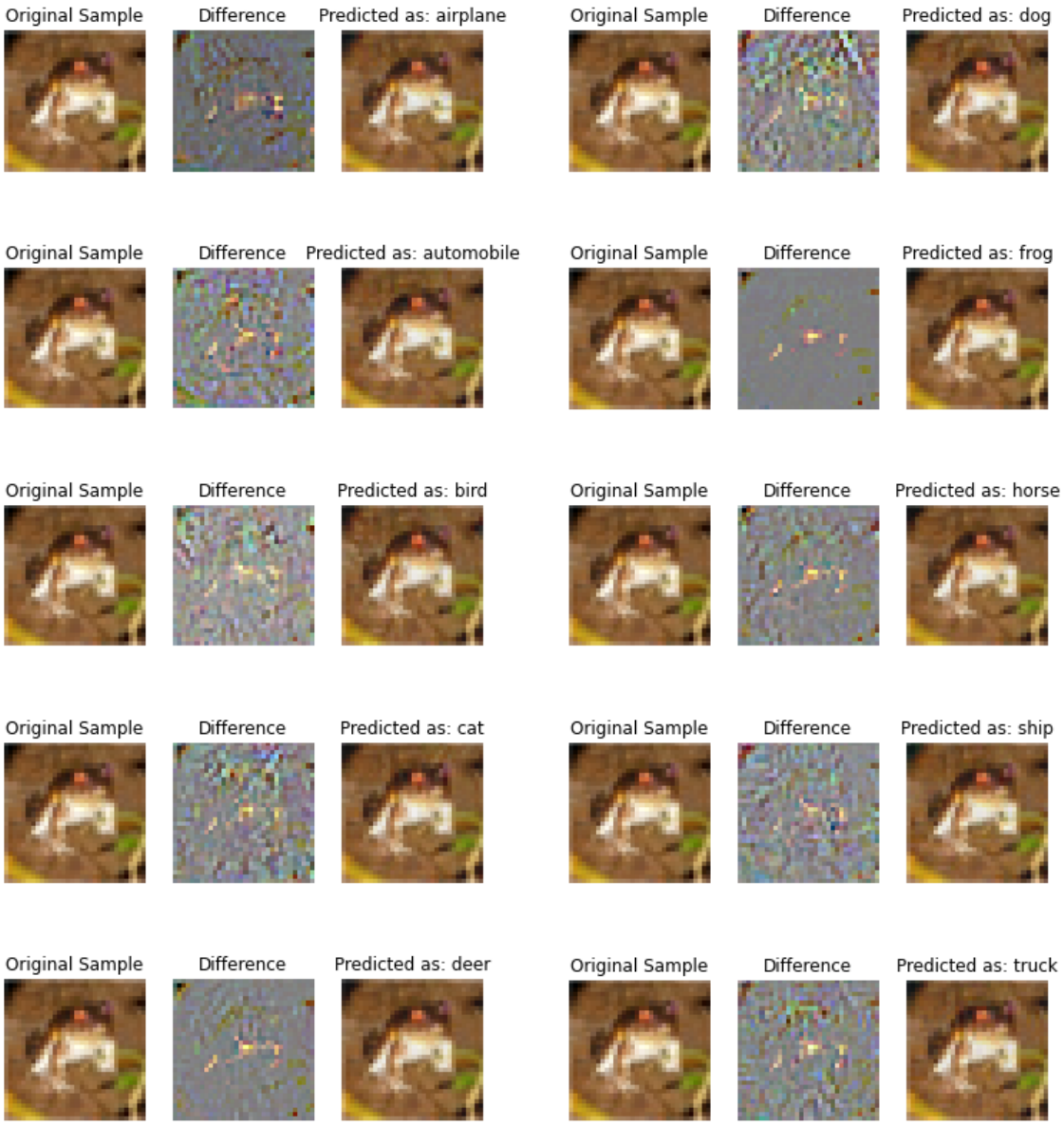
(implemented at : dsit_ml_ass2_2_cifar10.ipynb)

This time we were asked to repeat the same process again but for the cifar10 dataset. Due to cifar10 posing a harder problem since the objects are more complex and coloured in rgb we had to make some changes to our classifier architecture. For

the cifar10 dataset we used the architecture that is displayed below the classifier network:



By using the aforementioned architecture and training on the cifar10 training set for 50 epochs our model achieved an accuracy of 0.8159 that is satisfactory for the purposes of this task. By using the above architecture and performing the same steps as with the MNIST dataset we managed to create adversarial examples for a cifar10 pattern. The obtained results are shown below:



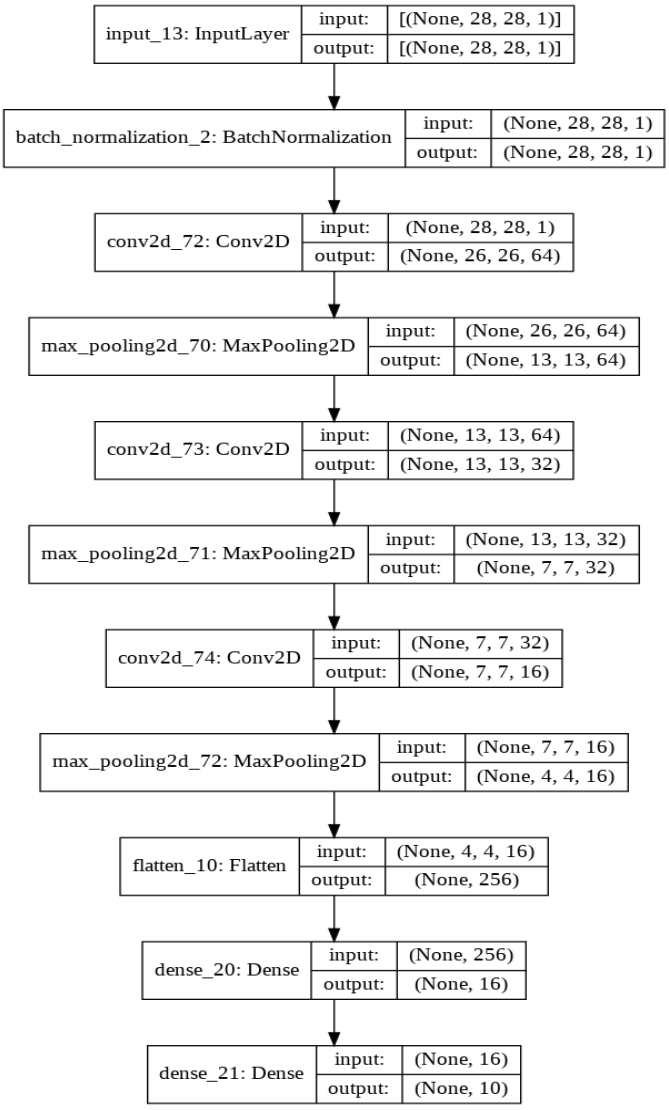
In this case we can see that the image of a frog is missclassified every time to another class of our liking by generating the respective adversarial example. It is

worth noting that in this case of a more complex RGB image the difference between the original and the adversarial example is indistinguishable by a human eye.

Exercise 3

(the model requested in exercise 3 is model_final.h5)

In this exercise we used a CNN along with the MNIST dataset in order to create a predictive model for the given handwritten digit dataset. Below you can see the model we used. **Important**: In order to use the h5 model we sent you, you have to turn the labels to categorical (we used *from keras.utils import to_categorical*) and also scale your images to the [0,1] space by dividing with 255. Our model takes as Input tensors of size (N,28,28,1).



We should also mention our experimentation with several handwritten digit datasets. We experimented also with the EMNIST (extended MNIST) handwritten digit part and with the kensanata handwritten digit dataset found on github. After all, we decided to utilize only the regular MNIST, since we didn't observe any improvement with the EMNIST or kensanata addition.

	MNIST	MNIST+kensanata	MNIST + EMNIST+kensanata
accuracy	**40%**	30%	28%

It is also worth mentioning that the performance of our model improved greatly due to data augmentation and by reducing the number of parameters trained in our deep network. More specifically we used zoom augmentations on the images, as well as width shift augmentations. In addition to those we also tried augmentations by height shift, rotations, flips and gaussian filters, brightness adjustment and gamma correction, which unfortunately did not improve our models performance on the given test set.

*****