Neural Networks
Implementing a Neural Network with Keras

Anne-Claire Fouchier, Jingwen Yang

I. INTRODUCTION

The aim of this lab is to build neural networks for classification with the Keras framework in Python, and evaluate their performance with different parameter settings.

II. DATASET

The dataset comes from the MNIST database of handwritten digits. The MNIST dataset is one of the most used datasets in machine learning research. It consists of 70000 grayscale images of handwritten digits (of size 28x28), divided into a training set of 60,000 examples and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image[1].

Fig. 1. A few samples from the MNIST test dataset by Josef Steppan

III. IMPLEMENTATION

Keras is a deep learning API written in Python, running on top of TensorFlow, an open-source machine learning library. It was developed to enable fast experimentation, and with a focus on deep learning.

There are three python files for this assignment, $lab3\_1.py$, $lab3\_2.py$ and $lab3\_3.py$.

IV. METHOD

A. Basics

A neural network (NN) consists of a set of layers disposed linearly, and each layer is a set of (artificial) neurons. The model is simplified so that signals can only circulate from the bottom layer to the top layer. In this lab, the neural networks are fully-connected, meaning that each neuron in the k_th layer of the neural network is connected to all the neurons in the (k-1)_th layer, and the neurons in a given layer are all independent from each other. A neural network consists of the following components :

- An input layer, x
- An arbitrary amount of hidden layers
- An output layer, y
- A set of weights and biases between each layer, w and b
- A choice of activation function for each hidden layer, σ.

B. Architectures

1) Single neuron: The Single neuron network takes 28x28=784 inputs and has a single sigmoid unit generating the output. The neuron simply learns to distinguish between the digit 0 and the other digits in the MNIST dataset.

2) One hidden layer: A hidden layer with 64 units is introduced. It takes 784 inputs, apply weights and bias. The output of the hidden layer is the input of the output layer, same as single neuron.

3) Multiclass Neural Network: This network has a hidden layer and recognizes all the 10 classes by introducing 10 neurons in the output layer.

C. Hyper-parameters and room for modifications

Hyperparameters, contrary to parameters learned through training, are parameters whose values are used to control the learning process. They affect the speed and quality of the learning process.

1) Mini-batch: A mini-batch is a subset of the data. It is used for training purposes : the weights and biases are updated after processing each batch. Mini-batches are used to increase the training speed and optimize the computation, as not all the data is processed at once.

2) Learning rate: The learning rate is how much the back-propagation will be included in updated the weights and bias, i.e how fast the update is operated. The update should neither be too fast (high learning rate) or too slow (small learning rate).

3) activation function: The activation function determines the output of a neural network, as it determines whether a neuron should be activated or not, based on its input. Activation functions help normalize the output of each neuron to a range between 0 and 1 or -1 and 1. It is a ”gate” between

[1]http://yann.lecun.com/exdb/mnist/

the input in the current neuron and the output going to the next layer [2].

4) Optimizers: Optimizers update the model weights and biases in response to the output of the loss function, with loss function result telling the optimizer whether it is moving in the right direction[3]. The available optimizers in keras are : ADAM, RMSprop, Stochastic Gradient Descent, Adagrad, Adadelta, Nadam and ftrl.

size of layer The size of the layer is the number of neurons a layer has.

V. RESULTS AND ANALYSIS

In the evaluation of the three neural netowrks, we split the training dataset into 60% and 40%, of which 60% is used to train the network and the other 40% is used as validation set. When a parameter is not specified, then is it is the default one given by keras. Also, the graphs plot the validation loss and accuracy, not the test.

A. Single neuron network

In this task we are going to evaluate the influence of batch size on a single neuron network performance.

From the Figures 2(a), 2(c), 2(e), 2(g), and 2(i) we can see a tendency of divergence of the training accuracy and validation accuracy. Between them, the training accuracy is getting higher while the validation accuracy is getting slightly lower.

Since with smaller batch size there are more weights updates, overfitting can be observed faster than with the larger batch size. Nevertheless, with a small batch size we can easily and quickly get high accuracy comparing to a very big batch size. On Figure 2(m) we can see the final accuracy our network achieved with the batch size of 36000, which is the whole dataset, is only around 95%. It also did not converge before epoch 100. It is because the weights update is far from enough.

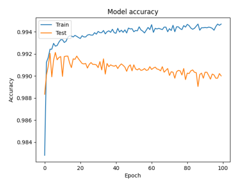
For the loss curve we can see similar patterns. Apart from that, it is very interesting to see that the bigger the batch size, the smoother the curves are.
As for the accuracy and loss on the test dataset, they all have very similar behaviors.
In conclusion, we should definitely choose a relatively small batch size but not too small to avoid over-fitting.

B. One hidden layer neuron network

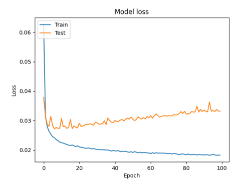
Here we are going to try out different activation functions on our hidden layer, and evaluate their performance. Besides that, we will discuss the number of neurons's influence on the model as well. Apart from those, the batch size remains 30 and the optimizer remains adam.

[2]https://missinglink.ai/guides/neural-network-concepts/7-types-neural-network-activation-functions-right/

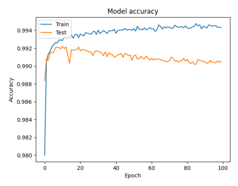
[3]https://algorithmia.com/blog/introduction-to-optimizers



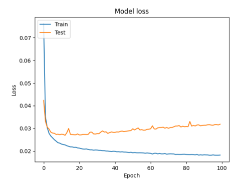
(a) accuracy, batch_size=10



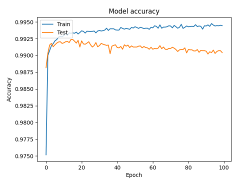
(b) loss, batch_size=10



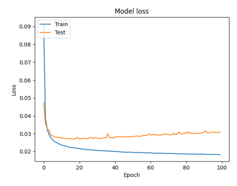
(c) accuracy, batch_size=20



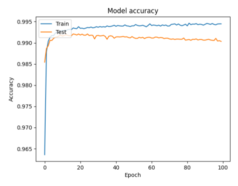
(d) loss, batch_size=20



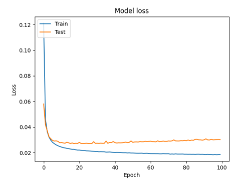
(e) accuracy, batch_size=30



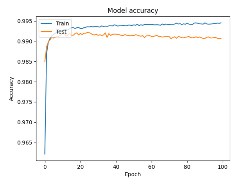
(f) loss, batch_size=30



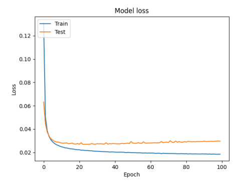
(g) accuracy, batch_size=40



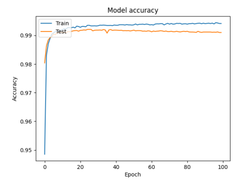
(h) loss, batch_size=40



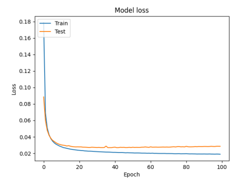
(i) accuracy, batch_size=50



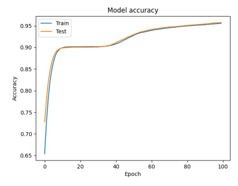
(j) loss, batch_size=50



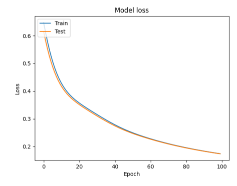
(k) accuracy, batch_size=100



(l) loss, batch_size=100

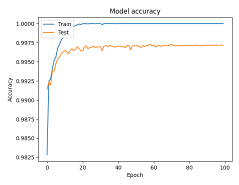


(m) acc, batch_size=36000

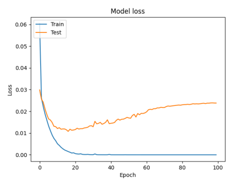


(n) loss, batch_size=36000

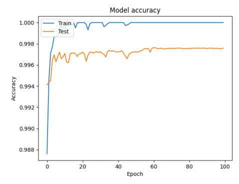
Fig. 2. Accuracy and loss graphs for various batch sizes,Single neuron network



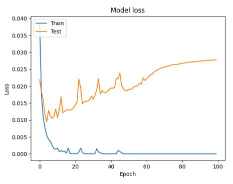
(a) accuracy, sigmoid



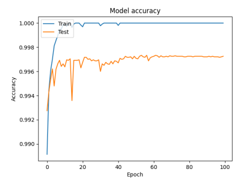
(b) loss, sigmoid



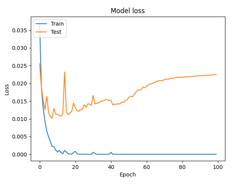
(c) accuracy, relu



(d) loss, relu



(e) accuracy, tanh



(f) loss, tanh

Fig. 3. Accuracy, loss graphs for various activation functions, one hidden layer network

From figure 3(a), 3(c), 3(e) we can see, all of them have almost the same accuracy on the validation set at around 99.75%. We can also see from the loss curves, sigmoid has the most smooth curve, tanh has the median, while relu has the least smooth loss curve. These behaviors are caused by the nature of these activation functions. The gradient is stronger for tanh than sigmoid(derivatives are steeper), while relu just drop anything smaller than 0. Weirdly, the loss all increased as the epoch number increased, which may be due to over-fitting, but the accuracies reamain stable.

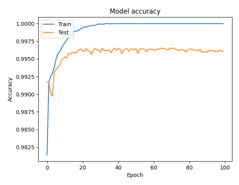
TABLE I

ACCURACY AND LOSS ON TEST SET WITH VARIOUS ACTIVATION FUNCTIONS

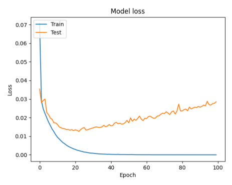
activation function	accuracy	loss
sigmoid	99.71	0.034
relu	99.80	0.032
tanh	99.78	0.028

From the table I, we can conclude that the network using relu activation function has the highest accuracy on the test set.

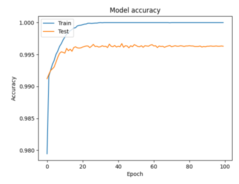
For a neuron number of 16, 32, 64, 128 they all have very similar performance on the validation dataset, among which the more the neurons, the slightly higher the accuracy value, at 0.1%. It can also be validated by the table II. With the highest neuron number of 128, we achieve the accuracy of 99.80% on the test dataset, whereas the model with 16



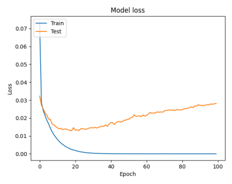
(a) accuracy, 16 neurons



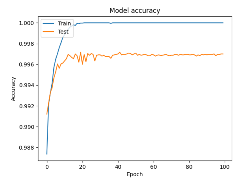
(b) loss, 16 neurons



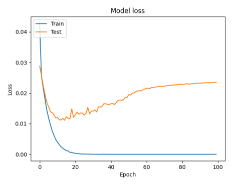
(c) accuracy, 32 neurons



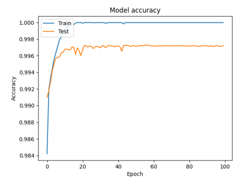
(d) loss, 32 neurons



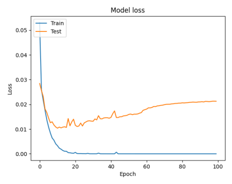
(e) accuracy, 64 neurons



(f) loss, 64 neurons



(g) accuracy, 128 neurons



(h) loss, 128 neurons

Fig. 4. Accuracy, loss graphs for various activation functions, one hidden layer network

neurons is the lowest, at 99.62%. The number of neurons should not be more than 2/3 of the input neurons, which is 784 in our case. The more the neurons, the less freedom and randomness the model has. Also, the more the neurons, the slower the training.

TABLE II

ACCURACY AND LOSS ON TEST SET WITH VARIOUS NEURON NUMBERS

neuron numbers	accuracy	loss
16	99.62	0.0324
32	99.74	0.0304
64	99.73	0.0298
128	99.80	0.0302

C. *Multi-class neuron network*

In this task we are going to evaluate the influence of the optimizer on the multi-class neural network with one hidden layer.

All the optimizers from keras were tested, with 64 neurons in the hidden layer, a batch size of 30 and the relu activation function. In Table III, we can see that the best results on the test set were yielded with Adam, followed closely by Adamax, and ftrl yields unsatisfying results.

TABLE III
ACCURACY AND LOSS ON TEST SET WITH VARIOUS OPTIMIZERS

optimizers	accuracy	loss
adam	96.91	0.20
rmsprop	96.72	0.18
SGD	95.28	0.16
adagrad	90.24	0.40
Adadelta	84.12	0.83
adamax	96.86	0.08
nadam	96.76	0.22
Ftrl	77.65	0.95

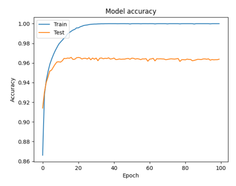
In Figure 5(a), we can see that the first to converge is the network with Adam. The networks with RMSprop yields similar results to the one with Adam; however, the validation accuracy slowly decreases (Figure 5(c)) which indicate that the network is overfitting. Both have their loss decrease then increase again. Figures 5(e) and 5(f) are shown here because the loss on the validation and training set are very similar, same with the accuracy. However, the results are not as good as the two previously mentioned. The loss function with Adamax in Figure 5(h) keeps decreasing, and plateaus, but does not increase again.

From these results, as well as the previous results, we tried to find the network that would yield the best accuracy on the test set. After several combinations of the parameters, we obtained our best test accuracy of 0.9823, with the Adam optimizer, the relu activation function, abatch size of 32, 300 neurons in the hidden layer and 80 epochs. We tried changing the learning rate but our choices only decreased the accuracy.

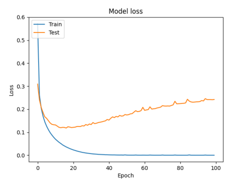
VI. CONCLUSION

Binary classification yields great results with simple networks. Multi-class classification is harder to tweak to try to yield as great results as binary classification.

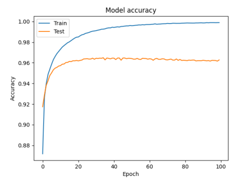
Keras is a great tool to implement neural networks. It was easy to try different implementations, with different parameter and to experiment with different batch sizes, activation functions, optimizers and layer sizes. It would now be interesting to also look into different loss functions and understanding chat range of learning rates we can safely use.



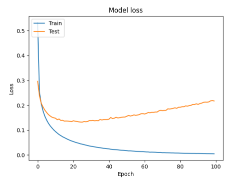
(a) accuracy, adam



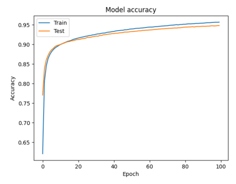
(b) loss, adam



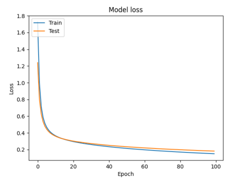
(c) accuracy, rmsprop



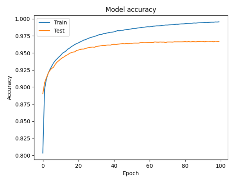
(d) loss, rmsprop



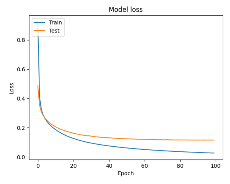
(e) accuracy,SGD



(f) loss, SGD



(g) accuracy,Adamax



(h) loss, Adamax

Fig. 5. Accuracy, loss graphs for various optimizers, multi-class network