Project Imaging, Assignment 2 – Group 5

Group members:		*Project Supervisor:*	*Date of submission:*
Kilian Cozijnsen	1004704	Mitko Veta	00-02-2019
Meike van den Eijnden	1000381		
Camiel Kerckhaert	1018368		
Marieke Vermeulen	1004498		
Ryan De Vries	1004141		

Exercise 1

Perform a set of experiments with more complex models, e.g. with more layers (deeper models), more neurons per layer or a combination. Describe the set of experiments that you have performed. What is the accuracy of the best model? How did you determine which model is the best?

The script has been run with the following combinations of number of layers and number of neurons per layer, which resulted in different losses and accuracies. From these results, it seems as if more layers and more neurons per layers cause a higher accuracy, which can be seen in table 1.

Table 1 Different models with corresponding losses and accuracies

	Loss	**accuracy**
1 layer, 64 neurons	0.1885	0.945
1 layer, 128 neurons	0.1756	0.9484
1 layer, 256 neurons	0.1700	0.952
2 layers, 64 neurons per layer	0.1317	0.9601
3 layers, 64 neurons per layer	0.1258	0.9609
4 layers, 64 neurons per layer	0.1133	0.9655
5 layers, 64 neurons per layer	0.1306	0.9578
10 layers, 64 neurons per layer	0.1189	0.9651
20 layers, 64 neurons per layer	0.1775	0.9574
5 layers, 128 neurons per layer	0.0993	0.9693
5 layers, 256 neurons per layer	0.0935	0.9716
10 layers, 256 neurons per layer	0.0905	0.975

In order to find the optimal number of hidden neurons the following equation is used: [1]

$$N_h = (N_{in} + \sqrt{N_p}) / L$$

where L is the number of hidden layers, Nin is the number of input neuron and Np is the number of input sample. In case of the data that is used, Nin is 28x28 and Np is 5400. The formula has now changed to

$$N_h = 1016/L$$

To determine which combination of Nh and L results in the lowest loss and the highest accuracy, trial and error was used. And for multiple number of layers the optimal number of neurons is determined. The results of this experiment can be found in table 2.

Table 2 Optimal number of neurons per layer and corresponding losses and accuracies

Number of hidden layers	Numbers of hidden neurons per layer	Loss	Accuracy
1	1016	0.15356	0.9547
2	508	0.1143	0.9649
3	339	0.0905	0.9723
4	254	0.0916	0.9717
5	203	0.0953	0.969
6	169	0.0902	0.9723
7	145	0.0945	0.972
8	127	0.1052	0.9714
9	113	0.1065	0.9704
10	102	0.1272	0.9641
11	92	0.1441	0.96
12	85	0.1244	0.9657
13	78	0.1324	0.9632

From table 2 it can be concluded that the optimal number of hidden layers is 6 with 169 neurons per layer, which resulted in an accuracy of 0.9723. Three hidden layers with 339 neurons per layers also results in an accuracy of 0.9723, but it generates a higher loss. However when these results are compared to table 1 this is indeed the best model tested.

The code which is used can be found in Appendix A

Exercise 2

Compare the performance of the following three models:

1) *Neural network without any hidden layers (the input layer connects directly to the output layer).*
2) *Neural network with 3 hidden layers with ReLU activations.*
3) *Neural network with 3 hidden layers with linear activations (i.e. without nonlinearities between the layers).*

Analyze the performance of the three models.

The layers are composed of 64 neurons. There are 10 time steps
The three ReLU layers are added with:
model.add(Dense(64, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(64, activation='relu'))
And the three linear layers are added with
model.add(Dense(64, activation='linear'))
model.add(Dense(64, activation='linear'))
model.add(Dense(64, activation='linear'))

Table 3 Comparing three different models

	Loss	Accuracy
Without layers	0.3108	0.9142
3 hidden layers with linear activation	0.2813	0.9172
With 3 hidden layers with ReLU activation	0.1219	0.9635

3 layers with ReLU activation performs the best, because it has the highest accuracy and the lowest loss after 10 time steps. The one without hidden layers performs the worst, because it has the lowest accuracy and a much higher loss.

In the second and third model different activation functions are used. In order to understand the difference in performance the concept of linear and Rectified Linear Unit (ReLU) needs to be understood. Activation functions are essential in deep learning as it determines the output of the model, as it maps the output values into a certain range. So different activation functions lead to different outputs, which results in different performance of the classification problems

Linear activation is given by a linear equation and thus the function is not restricted to a certain range. The ReLU function is given by:

$$\sigma(x) = \max(x, 0)$$

This implies that all negative values are converted to the value zero and all positive values remain their value. This results in a range from zero to the maximum value. It is found that the ReLU activation function outperforms all other activation functions.[2] The great performance of this activation function is due to the sparsity of activations after ReLU.[3]

The code which is used can be found in Appendix B.

Exercise 3

Train a neural network model (the specific architecture is up to you) for a four class classification problem derived from MNIST in the following way:

1) *"vertical digits": 1, 7*
2) *"loopy digits": 0, 6, 8, 9*
3) *"curly digits": 2, 5*
4) *"other": 3, 4*

For the neural network to classify in these four classes, the known output data (y_train, y_test and y_val) should also be divided in these four classes. This is done using a big for loop, in which every y value is changes to the format is has to be with the four class classification problem. The format for the one to nine classification was an array containing zeros where the indexes do not correspond with the number is shown in the image, and a one where the index does correspond. For this problem, an array with four indexes (as defined in the exercise) has been constructed, with a one in the first position if the image shows a 1 or 7 and so forth. In the code this is done by reading the y_train, y_test and y_val in several if loops. After correcting the y values, the model is trained and now provides a four class classification.

The code which is used to train this model van be found in Appendix C.

References

[1] (K. Gnana Sheela and S. N. Deepa, “Review on Methods to Fix Number of Hidden Neurons in Neural Networks,” Mathematical Problems in Engineering, vol. 2013, Article ID 425740, 11 pages, 2013. https://doi.org/10.1155/2013/425740)

[2] Schmidt-Hieber, J. (2018). *Nonparametric regression using deep neural networks with ReLU activation function*. Retrieved from https://arxiv.org/pdf/1708.06633.pdf

[3] Xu, B., Wang, N., Kong, H., Chen, T., & Li, M. (n.d.). *Empirical Evaluation of Rectified Activations in Convolution Network*. Retrieved from https://github.com/

Appendix A

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from keras.datasets import mnist
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers import Flatten, Dense
from keras.callbacks import TensorBoard

NrNeurons = 64
NrLayers = 2

# load the dataset using the builtin Keras method
(X_train, y_train), (X_test, y_test) = mnist.load_data()


# derive a validation set from the training set
# the original training set is split into
# new training set (90%) and a validation set (10%)
X_train, X_val = train_test_split(X_train, test_size=0.10, random_state=101)
y_train, y_val = train_test_split(y_train, test_size=0.10, random_state=101)



# the shape of the data matrix is NxHxW, where
# N is the number of images,
# H and W are the height and width of the images
# keras expect the data to have shape NxHxWxH, where
# C is the channel dimension
X_train = np.reshape(X_train, (-1,28,28,1))
X_val = np.reshape(X_val, (-1,28,28,1))
X_test = np.reshape(X_test, (-1,28,28,1))


# convert the datatype to float32
X_train = X_train.astype('float32')
X_val = X_val.astype('float32')
X_test = X_test.astype('float32')


# normalize our data values to the range [0,1]
X_train /= 255
X_val /= 255
X_test /= 255


# convert 1D class arrays to 10D class matrices
y_train = np_utils.to_categorical(y_train, 10)
y_val = np_utils.to_categorical(y_val, 10)
y_test = np_utils.to_categorical(y_test, 10)
```

```
model = Sequential()
# flatten the 28x28x1 pixel input images to a row of pixels (a 1D-array)
model.add(Flatten(input_shape=(28,28,1)))
# fully connected layer with a number of neurons and ReLU nonlinearity
for i in range(NrLayers):
   model.add(Dense(NrNeurons, activation='relu'))


# output layer with 10 nodes (one for each class) and softmax nonlinearity
model.add(Dense(10, activation='softmax'))


# compile the model
model.compile(loss='categorical_crossentropy', optimizer='sgd', metrics=['accuracy'])

# use this variable to name your model
model_name="my_first_model"

# create a way to monitor our model in Tensorboard
tensorboard = TensorBoard("logs/{}".format(model_name))

# train the model
model.fit(X_train, y_train, batch_size=32, epochs=10, verbose=1, validation_data=(X_val, y_val),
callbacks=[tensorboard])


score = model.evaluate(X_test, y_test, verbose=0)


print("Loss: ",score[0])
print("Accuracy: ",score[1])
```

Appendix B

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from keras.datasets import mnist
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers import Flatten, Dense
from keras.callbacks import TensorBoard


# load the dataset using the builtin Keras method
(X_train, y_train), (X_test, y_test) = mnist.load_data()


# derive a validation set from the training set
# the original training set is split into
# new training set (90%) and a validation set (10%)
X_train, X_val = train_test_split(X_train, test_size=0.10, random_state=101)
y_train, y_val = train_test_split(y_train, test_size=0.10, random_state=101)



# the shape of the data matrix is NxHxW, where
# N is the number of images,
# H and W are the height and width of the images
# keras expect the data to have shape NxHxWxH, where
# C is the channel dimension
X_train = np.reshape(X_train, (-1,28,28,1))
X_val = np.reshape(X_val, (-1,28,28,1))
X_test = np.reshape(X_test, (-1,28,28,1))


# convert the datatype to float32
X_train = X_train.astype('float32')
X_val = X_val.astype('float32')
X_test = X_test.astype('float32')


# normalize our data values to the range [0,1]
X_train /= 255
X_val /= 255
X_test /= 255


# convert 1D class arrays to 10D class matrices
y_train = np_utils.to_categorical(y_train, 10)
y_val = np_utils.to_categorical(y_val, 10)
y_test = np_utils.to_categorical(y_test, 10)
```

```
model = Sequential()
# flatten the 28x28x1 pixel input images to a row of pixels (a 1D-array)
model.add(Flatten(input_shape=(28,28,1)))
# fully connected layer with 64 neurons
#choose parameter
parameter=3;#1=ReLU,2=linear,3=none
if parameter==1:
   model.add(Dense(64, activation='relu'))
   model.add(Dense(64, activation='relu'))
   model.add(Dense(64, activation='relu'))
elif parameter==2:
   model.add(Dense(64, activation='linear'))
   model.add(Dense(64, activation='linear'))
   model.add(Dense(64, activation='linear'))
elif parameter==3:
   pass
# output layer with 10 nodes (one for each class) and softmax nonlinearity
model.add(Dense(10, activation='softmax'))


# compile the model
model.compile(loss='categorical_crossentropy', optimizer='sgd', metrics=['accuracy'])

# use this variable to name your model
model_name="my_first_model"

# create a way to monitor our model in Tensorboard
tensorboard = TensorBoard("logs/{}".format(model_name))

# train the model
model.fit(X_train, y_train, batch_size=32, epochs=10, verbose=1, validation_data=(X_val, y_val),
callbacks=[tensorboard])


score = model.evaluate(X_test, y_test, verbose=0)


print("Loss: ",score[0])
print("Accuracy: ",score[1])
```

Appendix C

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from keras.datasets import mnist
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers import Flatten, Dense
from keras.callbacks import TensorBoard


# load the dataset using the builtin Keras method
(X_train, y_train), (X_test, y_test) = mnist.load_data()


# derive a validation set from the training set
# the original training set is split into
# new training set (90%) and a validation set (10%)
X_train, X_val = train_test_split(X_train, test_size=0.10, random_state=101)
y_train, y_val = train_test_split(y_train, test_size=0.10, random_state=101)



# the shape of the data matrix is NxHxW, where
# N is the number of images,
# H and W are the height and width of the images
# keras expect the data to have shape NxHxWxH, where
# C is the channel dimension
X_train = np.reshape(X_train, (-1,28,28,1))
X_val = np.reshape(X_val, (-1,28,28,1))
X_test = np.reshape(X_test, (-1,28,28,1))


# convert the datatype to float32
X_train = X_train.astype('float32')
X_val = X_val.astype('float32')
X_test = X_test.astype('float32')


# normalize our data values to the range [0,1]
X_train /= 255
X_val /= 255
X_test /= 255


# convert 1D class arrays to 4D class matrices, according to the groups from exercise 3
#First output node: "vertical digits": 1, 7
#Second output node: "loopy digits": 0, 6, 8, 9
#Third output node: "curly digits": 2, 5
```

```
#Fourth output node: "other": 3, 4
number_of_categories = 4
y_train_categories = np.zeros((len(y_train), number_of_categories), np.float32)
y_val_categories = np.zeros((len(y_val), number_of_categories), np.float32)
y_test_categories = np.zeros((len(y_test), number_of_categories), np.float32)

for i in range(len(y_train)):
   if (y_train[i] == 1) or (y_train[i] == 7):
      y_train_categories[i] = [1.,0.,0.,0.]
   elif (y_train[i] == 0) or (y_train[i] == 6) or (y_train[i] == 8) or (y_train[i] == 9):
      y_train_categories[i] = [0.,1.,0.,0.]
   elif (y_train[i] == 2) or (y_train[i] == 5):
      y_train_categories[i] = [0.,0.,1.,0.]
   elif (y_train[i] == 3) or (y_train[i] == 4):
      y_train_categories[i] = [0.,0.,0.,1.]
   else:
      print("invalid output in y_train at position" + str(y_train[i]))

for i in range(len(y_val)):
   if (y_val[i] == 1) or (y_val[i] == 7):
      y_val_categories[i] = [1.,0.,0.,0.]
   elif (y_val[i] == 0) or (y_val[i] == 6) or (y_val[i] == 8) or (y_val[i] == 9):
      y_val_categories[i] = [0.,1.,0.,0.]
   elif (y_val[i] == 2) or (y_val[i] == 5):
      y_val_categories[i] = [0.,0.,1.,0.]
   elif (y_val[i] == 3) or (y_val[i] == 4):
      y_val_categories[i] = [0.,0.,0.,1.]
   else:
      print("invalid output in y_val at position" + str(y_val[i]))

for i in range(len(y_test)):
   if (y_test[i] == 1) or (y_test[i] == 7):
      y_test_categories[i] = [1.,0.,0.,0.]
   elif (y_test[i] == 0) or (y_test[i] == 6) or (y_test[i] == 8) or (y_test[i] == 9):
      y_test_categories[i] = [0.,1.,0.,0.]
   elif (y_test[i] == 2) or (y_test[i] == 5):
      y_test_categories[i] = [0.,0.,1.,0.]
   elif (y_test[i] == 3) or (y_test[i] == 4):
      y_test_categories[i] = [0.,0.,0.,1.]
   else:
      print("invalid output in y_test at position" + str(y_test[i]))

model = Sequential()
# flatten the 28x28x1 pixel input images to a row of pixels (a 1D-array)
model.add(Flatten(input_shape=(28,28,1)))
# fully connected layer with 64 neurons and ReLU nonlinearity
model.add(Dense(64, activation='relu'))
# output layer with 10 nodes (one for each class) and softmax nonlinearity
model.add(Dense(4, activation='softmax'))

# compile the model
```

```
model.compile(loss='categorical_crossentropy', optimizer='sgd', metrics=['accuracy'])

# use this variable to name your model
model_name="my_first_model"

# create a way to monitor our model in Tensorboard
tensorboard = TensorBoard("logs/{}".format(model_name))

# train the model
model.fit(X_train, y_train_categories, batch_size=32, epochs=10, verbose=1, validation_data=(X_val,
y_val_categories), callbacks=[tensorboard])

print("checkpoint 3")

score = model.evaluate(X_test, y_test_categories, verbose=0)


print("Loss: ",score[0])
print("Accuracy: ",score[1])
```