Answer Sheet: Lab3 , Team : 15

Faiz Ahmed	Mithun Das	Mutasim Fuad Ansari	Mohammad Abir Reza
1152231	1151651	1152109	1151705
stu225473@mail.uni-kiel.de	stu225039@mail.uni-kiel.de	stu225365@mail.uni-kiel.de	stu225093@mail.uni-kiel.de

Exercise 1 (Learning in neural networks)

a) Explain the following terms related to neural networks as short and precise as possible.

- Learning in neural networks
- Training set
- Supervised Learning
- Unsupervised Learning
- Online (incremental) learning
- Offline (batch) learning
- Training error
- Generalisation error
- Overfitting
- Cross-validation

Answer

- Learning in neural networks : "learning" refers to specifying the organization of the network(connectivity, neuronal elementsetc.) in such a way that a desired network response is achieved for a given set of input patterns(the "trainingset")
- Training set : Training set is Certain number of measured values, which is used as inputs and assesment of network output to fit the model. This is the actual dataset that we use to train the model .
- Supervised Learning : Supervised learning is a machine learning technique using data corresponding to target output.That means data is already tagged with the correct answer.It can be compared to learning which takes place in the presence of a supervisor or a teacher.
- Unsupervised Learning : Unsupervised learning data without targeted output. The network detects similarities and generates groups with similar characteristics. For example - Clustering problems.

- Online (incremental) learning : In online learning, machine learns after every training sample.
- Offline (batch) learning : After inputing all training samples machine learns in offline learning.
- Training error : Training error is the difference between network output and target output of training data set.

- Generalisation error : Generalization error (also known as the out-of-sample error) is a measure of how accurately an algorithm is able to predict outcome values for previously unseen data.
- Overfitting :Details of some training patterns are learned which are not relevant for most of the remaining patterns.It is caused for too much detail.
- Cross-validation : The concept of Cross-validation is to split the dataset D(with P patterns) into several part. One part is taken as test set rest of the parts are used as trainig set. In the next step another part is taken as test case and other parts are used ase training set. If the number of partitions is k with 2 <= k <= p, the

iterations goes for k times.

b) Name and briefly describe at least two methods to indicate or avoid overfitting when training neural networks.

Answer

Two methods to avoid overfitting are described below :

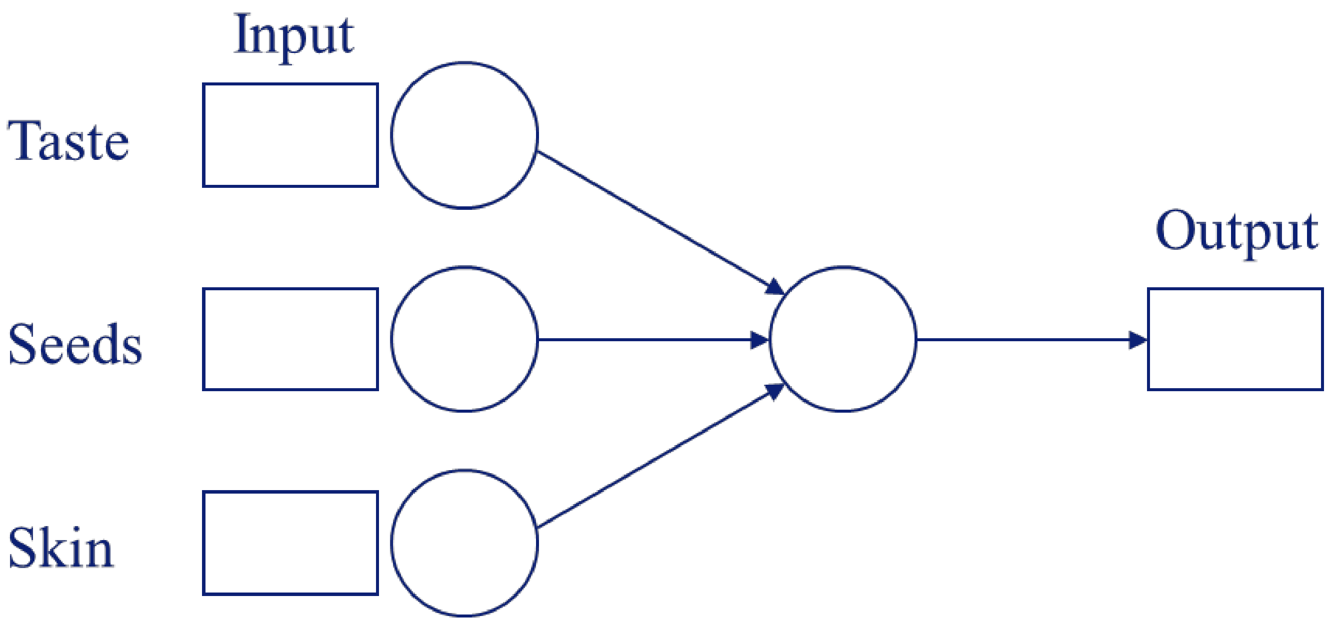
1. Early Stopping : When validation error reach the minimum value, training need to be stopped.
2. Regulerization : Too many network parameters may lead to complexity of the network and also cause overfitting. There are different techniques for regularization. Dropout (reducing node in hidden layer), weight regularization by adding weight penalty(L1,l2 regularizaiton), Data Augmentation (a strategy that enables practitioners to significantly increase the diversity of data available for training models, without actually collecting new data) are useful strategies to implement regularization.

Exercise 2 (Perceptron learning – analytical calculation)

The goal of this exercise is to train a single-layer perceptron (threshold element) to classify whether a fruit presented to the perceptron is going to be liked by a certain person or not, based on three features attributed to the presented fruit: its taste (whether it is sweet or not), its seeds (whether they are edible or not) and its skin (whether it is edible or not). This generates the following table for the inputs and the target output of the perceptron:

Fruit	Input Taste sweet = 1 not sweet = 0	Input Seeds edible = 1 not edible = 0	Input Skin edible = 1 not edible = 0	Target output person likes = 1 doesn’t like = 0
Banana	1	1	0	1
Pear	1	0	1	1
Lemon	0	0	0	0
Strawberry	1	1	1	1
Green Apple	0	0	1	0

Since there are three (binary) input values (taste, seeds and skin) and one (binary) target output, we will construct a single-layer perceptron with three inputs and one output.



Since the target output is binary, we will use the perceptron learning algorithm to construct the weights.

To start the perceptron learning algorithm, we have to initialize the weights and the threshold. Since we have no prior knowledge on the solution, we will assume that all weights are 0 ($w_1 = w_2 = w_3 = 0$) and that the threshold is $\theta = 1$ (i.e. $w_0 = -\theta = -1$). Furthermore, we have to specify the learning rate η. Since we want it to be large enough that learning happens in a reasonable amount of time, but small enough so that it doesn't go too fast, we set $\eta = 0.25$.

Apply the perceptron learning algorithm – in the incremental mode – analytically to this problem, i.e. calculate the new weights and threshold after successively presenting a banana, pear, lemon, strawberry and a green apple to the network (in this order).

Draw a diagram of the final perceptron indicating the weight and threshold parameters and verify that the final perceptron classifies all training examples correctly.

Note: The iteration of the perceptron learning algorithm is easily accomplished by filling in the following table for each iteration of the learning algorithm:

First iteration ($\mu = 1$), current training sample: banana

Input $x^{(\mu)}$	**Current Weights** $w(t)$	**Network Output** $y^{(\mu)}$	**Target Ouput** $d^{(\mu)}$	**Learning rate** η	**Weight Update** $\Delta w(t)$	**New weights** $w(t+1)$
x_0 = 1	w_0 =			0.25		
x_1 =	w_1 =			0.25		
x_2 =	w_2 =			0.25		
x_3 =	w_3 =			0.25		

Second iteration ($\mu = 2$), current training sample: pear ...

(Source of exercise: Langston, Cognitive Psychology)

Answer

Iteration for Banana

From Given information Output for banana, d1 = 1
Calculation :

Output y1 = Θ [1 * (-1) + 1 * 0 + 1 * 0 + 0 * 0] = Θ [-1] = 0

w0(1) = w0(0) + η * (d1 - y1) * x0 = (-1) + 0.25 * (1 – 0) * 1 = -0.75

w1 (1) = w1 (0) + η * (d1 - y1) * x1 = 0 + 0.25 * (1 – 0) * 1 = 0.25

w2 (1) = w2 (0) + η * (d1 - y1) * x2 = 0 + 0.25 * (1 – 0) * 1 = 0.25

w3 (1) = w2 (0) + η * (d1 - y1) * x3 = 0 + 0.25 * (1 – 0) * 0 = 0

$\Delta w0$(1) = w0(1) - w0 (0) = -0.75 – (-1) = 0.25

$\Delta w1$(1) = w1(1) - w1 (0) = 0.25 – 0 = 0.25

$\Delta w2$(1) = w2(1) - w2 (0) = 0.25 – 0 = 0.25

$\Delta w3$(1) = w3(1) - w3 (0) = 0 – 0 = 0

Input $x^{(\mu)}$	**Current Weights** $w(t)$	**Network Output** $y^{(\mu)}$	**Target Ouput** $d^{(\mu)}$	**Learning rate** η	**Weight Update** $\Delta w(t)$	**New weights** $w(t+1)$
x_0 = 1	w_0 = -1	0	1	0.25	0.25	-.75
x_1 = 1	w_1 = 0			0.25	0.25	0.25
x_2 = 1	w_2 = 0			0.25	0.25	0.25
x_3 = 0	w_3 = 0			0.25	0.00	0

Iteration for Pear

From Given information Output for Pear, d2 = 1
Calculation :

Output y2 = Θ [1 * (-0.75) + 1 * 0.25 + 0 * 0.25 + 1 * 0] = Θ [-0.50] = 0

w0 (2) = w0 (1) + η * (d2 – y2) * x0 = (-0.75) + 0.25 * (1 – 0) * 1 = -0.50

w1 (2) = w1 (1) + η * (d2 – y2) * x1 = 0.25 + 0.25 * (1 – 0) * 1 = 0.50

w2 (2) = w2 (1) + η * (d2 – y2) * x2 = 0.25 + 0.25 * (1 – 0) * 0 = 0.25

w3 (2) = w2 (1) + η * (d2 – y2) * x3 = 0 + 0.25 * (1 – 0) * 1 = 0.25

$\Delta w0$(2) = w0(2) - w0 (1) = -0.50 – (-0.75) = 0.25

$\Delta w1$(2)= w1(2) - w1 (1) = 0.50 – 0.25 = 0.25

$\Delta w2$(2) = w2(2) - w2 (1) = 0.25 – 0.25 = 0

$\Delta w3$(2) = w3(2) - w3 (1) = 0.25 – 0 = 0.25

Input $x^{(\mu)}$	**Current Weights** $w(t)$	**Network Output** $y^{(\mu)}$	**Target Ouput** $d^{(\mu)}$	**Learning rate** η	**Weight Update** $\Delta w(t)$	**New weights** $w(t+1)$

Input $x^{(\mu)}$	Current Weights $w(t)$	Network Output $y^{(\mu)}$	Target Ouput $d^{(\mu)}$	Learning rate η	Weight Update $\Delta w(t)$	New weights $w(t+1)$
x_0 = 1	w_0 = -1	0	1	0.25	0.25	-.50
x_1 = 1	w_1 = 0			0.25	0.25	0.50
x_2 = 1	w_2 = 0			0.25	0	0.25
x_3 = 0	w_3 = 0			0.25	0.00	0.25

Iteration for Lemon

From Given information Output for Lemon, d3 = 0
Calculation :

Output y3 = Θ [1 * (-0.50) + 0 * 0.50 + 0 * 0.25 + 0 * 0.25] = Θ [-0.50] = 0

w0 (3) = w0 (2) + η * (d3 – y3) * x0 = (-0.50) + 0.25 * (0 – 0) * 0 = -0.50

w1 (3) = w1 (2) + η * (d3 – y3) * x1 = 0.50 + 0.25 * (0 – 0) * 0 = 0.50

w2 (3) = w2 (2) + η * (d3 – y3) * x2 = 0.25 + 0.25 * (0 – 0) * 0 = 0.25

w3 (3) = w2 (2) + η * (d3 – y3) * x3 = 0.25 + 0.25 * (0 – 0) * 0 = 0.25

$\Delta w0$(3) = w0(3) - w0 (2) = -0.50 – (-0.50) = 0

$\Delta w1$(3)= w1(3) - w1 (2) = 0.50 – 0.50 = 0

$\Delta w2$(3) = w2(3) - w2 (2) = 0.25 – 0.25 = 0

$\Delta w3$(3) = w3(3) - w3 (2) = 0.25 – 0.25 = 0

Input $x^{(\mu)}$	Current Weights $w(t)$	Network Output $y^{(\mu)}$	Target Ouput $d^{(\mu)}$	Learning rate η	Weight Update $\Delta w(t)$	New weights $w(t+1)$
x_0 = 1	w_0 = -1	0	0	0.25	0	-.50
x_1 = 1	w_1 = 0			0.25	0	0.50
x_2 = 1	w_2 = 0			0.25	0	0.25
x_3 = 0	w_3 = 0			0.25	0.	0.25

Iteration for Strawberry

From Given information Output for Strawberry, d4 = 1
Calculation :

Output y4 = Θ [1 * (-0.50) + 1 * 0.50 + 1 * 0.25 + 1 * 0.25] = Θ [0.50] = 1

w0 (4) = w0 (3) + η * (d4 – y4) * x0 = (-0.50) + 0.25 * (1 – 1) * 1 = -0.50

w1 (4) = w1 (3) + η * (d4 – y4) * x1 = 0.50 + 0.25 * (1 – 1) * 1 = 0.50

w2 (4) = w2 (3) + η * (d4 – y4) * x2 = 0.25 + 0.25 * (1 – 1) * 1 = 0.25

w3 (4) = w2 (3) + η * (d4 – y4) * x3 = 0.25 + 0.25 * (1 – 1) * 1 = 0.25

$\Delta w0$(4) = w0(4) - w0 (3) = -0.50 – (-0.50) = 0

$\Delta w1$(4)= w1(4) - w1 (3) = 0.50 – 0.50 = 0

$\Delta w2$(4) = w2(4) - w2 (3) = 0.25 – 0.25 = 0

$\Delta w3$(4) = w3(4) - w3 (3) = 0.25 – 0.25 = 0

Input $x^{(\mu)}$	**Current Weights** $w(t)$	**Network Output** $y^{(\mu)}$	**Target Ouput** $d^{(\mu)}$	**Learning rate** η	**Weight Update** $\Delta w(t)$	**New weights** $w(t+1)$
x_0 = 1	w_0 = -1	1	1	0.25	0	-.50
x_1 = 1	w_1 = 0			0.25	0	0.50
x_2 = 1	w_2 = 0			0.25	0	0.25
x_3 = 0	w_3 = 0			0.25	0.	0.25

Iteration for Green Apple

From Given information Output for Green Apple, d5 = 0 **Calculation :**

Output y5 = Θ [1 * (-0.50) + 0 * 0.50 + 0 * 0.25 + 1 * 0.25] = Θ [-0.25] = 0

w0 (5) = w0 (4) + η * (d5 – y5) * x0 = (-0.50) + (-0.50) + 0.25 * (0 – 0) * 1 = -0.50

w1 (5) = w1 (4) + η * (d5 – y5) * x1 = 0.50 + 0.25 * (0 – 0) * 0 = 0.50

w2 (5) = w2 (4) + η * (d5 – y5) * x2 = 0.25 + 0.25 * (0 – 0) * 0 = 0.25

w3 (5) = w2 (4) + η * (d5 – y5) * x3 = 0.25 + 0.25 * (0 – 0) * 1 = 0.25

$\Delta w0$(5) = w0(5) - w0 (4) = -0.50 – (-0.50) = 0

$\Delta w1$(5)= w1(5) - w1 (4) = 0.50 – 0.50 = 0

$\Delta w2$(5) = w2(5) - w2 (4) = 0.25 – 0.25 = 0

$\Delta w3$(5) = w3(5) - w3 (4) = 0.25 – 0.25 = 0

Input $x^{(\mu)}$	**Current Weights** $w(t)$	**Network Output** $y^{(\mu)}$	**Target Ouput** $d^{(\mu)}$	**Learning rate** η	**Weight Update** $\Delta w(t)$	**New weights** $w(t+1)$
x_0 = 1	w_0 = -1	0	0	0.25	0	-.50
x_1 = 1	w_1 = 0			0.25	0	0.50
x_2 = 1	w_2 = 0			0.25	0	0.25
x_3 = 0	w_3 = 0			0.25	0.	0.25

Here after all 5 training set we get w1 = 0.50 , w2 = 0.25 , w3 = 0.25 , θ = 0.50.

Exercise 3 (Single-layer perceptron, gradient learning, 2dim. classification)

The goal of this exercise is to solve a two-dimensional binary classification problem with gradient learning, using TensorFlow. Since the problem is two-dimensional, the perceptron has 2 inputs. Since the classification problem is binary, there is one output.

The (two-dimensional) inputs for training are provided in the file *exercise3b_input.txt*, the corresponding (1-dimensional) targets in the file *exercise3b_target.txt*. To visualize the results, the training samples corresponding to class 1 (output label “0”) have separately been saved in the file *exercise3b_class1.txt*, the training samples corresponding to class 2 (output label “1”) in the file *exercise3b_class2.txt*.

The gradient learning algorithm – using the sigmoid activation function – shall be used to provide a solution to this classification problem. Note that due to the sigmoid activation function, the output of the perceptron is a real value in [0,1]:

$$sigmoid(h) = \frac{1}{1 + e^{-h}}$$

To assign a binary class label (either 0 or 1) to an input example, the perceptron output y can be passed through the Heaviside function $\theta[y - 0.5]$ to yield a binary output y^{binary}. Then, any perceptron output between 0.5 and 1 is closer to 1 than to 0 and will be assigned the class label “1”. Conversely, any perceptron output between 0 and < 0.5 is closer to 0 than to 1 and will be assigned the class label “0”. As usual, denote the weights of the perceptron w_1 and w_2 and the bias $w_0 = -\theta$.

Task a)

Using the above-mentioned post-processing step $\theta[y - 0.5]$ applied to the perceptron output y, show that the decision boundary separating the inputs $x = (x_1, x_2)$ assigned to class label “1” from those inputs assigned to class label “0” is given by a straight line in two-dimensional space corresponding to the equation (see in python code at # *plot last decision boundary*):

$x_2 = -\frac{w_1}{w_2} x_1 - \frac{w_0}{w_2}$

Answer

Write your answer here.

Task b)

The classification problem (defined by the training data provided in *exercise3b_input.txt* and the targets provided in *exercise3b_target.txt*) shall now be solved using the TensorFlow and Keras libraries. The source code is given below and can be executed by clicking the play button (in colab or in a local installation with tensorflow and keras).

1. Train the model at least three times and report on your findings.
2. Change appropriate parameters (e.g. the learning rate, the batch size, the choice of the solver, potentially the number of epochs etc.) and again report on your findings.

Useful information about training and evaluation with Tensorflow and Keras can be found at [https://www.tensorflow.org/guide/keras/train_and_evaluate (https://www.tensorflow.org/guide/keras/train_and_evaluate)](https://www.tensorflow.org/guide/keras/train_and_evaluate)

In [0]:

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from os.path import join
from tensorflow.keras.layers import Dense
from tensorflow.keras import Model, Input
from tensorflow.keras.optimizers import SGD, Adam

###-----------------
# load training data
###-----------------
path_to_task = "nndl/Lab3"
input = np.loadtxt(join(path_to_task,'exercise3b_input.txt'))
tmp = np.loadtxt(join(path_to_task,'exercise3b_target.txt'))
target = np.array([tmp[i] for i in range(tmp.size)])
class1 = np.loadtxt(join(path_to_task,'exercise3b_class1.txt'))
class2 = np.loadtxt(join(path_to_task,'exercise3b_class2.txt'))
```

Define the neural network, here you can change the structure of network, the learning rate and the optimizer

In [0]:

```
# Define the structure
input_layer = Input(shape=(2,), name='input') # two dimensional input
out = Dense(units=1, activation="sigmoid", name="output")(input_layer) # one ouput

# create a model
model = Model(input_layer, out)

# show how the model looks
model.summary()

# compile the model
opt = SGD(learning_rate=0.1)
model.compile(optimizer=opt,loss="binary_crossentropy",metrics=["acc"])

# try to invoke one of the weight initializers
# initializer = tf.keras.initializers.GlorotUniform()
# shape = (2,1) # n_in, n_out
# random_weights = tf.Variable(initializer(shape=shape)) # returns tensor object; h
# random_weights = np.random.uniform(low = -1.0, high = 1.0, size=(2,1))

# weights: list; index 0: weights (numpy array of shape n_in x n_out), index 1: bia
# model.set_weights([ random_weights, np.array([0])])

# save initial weights
initial_weights = model.layers[-1].get_weights()

print("initial weights: (%f, %f)" % (initial_weights[0][0], initial_weights[0][1]))
print("initial bias: %f" % initial_weights[1][0])
```

This line actually trains the model. Changeable parameters batch_size and epochs.

In [0]:

```
# Train the model
history = model.fit(x=input, y=target, batch_size=1, epochs=100, verbose=True)
```

The following code snippet plots the results you create in the snippet before.

In [0]:

```
# plot setup
fig, axes = plt.subplots(1, 3, figsize=(15, 15))
legend = []

# plot the data
axes[0].set_title('Toy classification problem: Data and decision boundaries')
axes[0].set_xlabel('x1')
axes[0].set_ylabel('x2')

minx = min(input[:,0])
maxx = max(input[:,0])
miny = min(input[:,1])
maxy = max(input[:,1])
axes[0].set_xlim(minx, maxx)
axes[0].set_ylim(miny, maxy)
axes[0].plot(class1[:,0], class1[:,1], 'r.', \
    class2[:,0], class2[:,1], 'b.')
legend.append('samples class1')
legend.append('samples class2')

# initial weights
w0 = initial_weights[1][0] # bias
# weight components (list of of numpy arrays of shape n_in x n_out)
w1 = initial_weights[0][0][0]
w2 = initial_weights[0][1][0]
if ( w2 == 0 ):
    print("Error: second weight zero!")

# calculate initial decision boundary
interval = np.arange( np.floor(minx), np.ceil(maxx), 0.1 )
initial_decision_boundary = -w1*interval/w2 - w0/w2

# plot initial decision boundary
args = {'c': 'black', 'linestyle': 'dashed'}
axes[0].plot( interval, initial_decision_boundary, **args)
legend.append('initial decision boundary')

# get final weights
final_weights = model.layers[-1].get_weights()
w0 = final_weights[1][0] # bias
# weight components (list of of numpy arrays of shape n_in x n_out)
w1 = final_weights[0][0][0]
w2 = final_weights[0][1][0]
if ( w2 == 0 ):
    print("Error: second weight zero!")

print("final weights: (%f, %f)" % (final_weights[0][0], final_weights[0][1]))
print("final bias: %f" % final_weights[1][0])

# calculate final decision boundary
interval = np.arange( np.floor(minx), np.ceil(maxx), 0.1 )
final_decision_boundary = -w1*interval/w2 - w0/w2

# plot final decision boundary
args = {'c': 'black', 'linestyle': '-'}
axes[0].plot( interval, final_decision_boundary, **args)
legend.append('final decision boundary')

# plot training loss
```

```
axes[1].plot(history.history['loss'])
axes[1].set_title('Toy classification problem: Loss curve')
axes[1].set_xlabel('Epoch number')
axes[1].set_ylim(0, 1)
axes[1].set_ylabel('loss')

# plot training accuracy
axes[2].plot(history.history['acc'])
axes[2].set_title('Toy classification problem: acc curve')
axes[2].set_ylim(0, 1)
axes[2].set_xlabel('Epoch number')
axes[2].set_ylabel('acc')

# show the plot
fig.legend(axes[0].get_lines(), legend, ncol=3, loc="upper center")
plt.show()

# final evaluation (here: on the training data)
eval = model.evaluate(x=input, y=target)
print("Final loss: %f, final accuray: %f" % (eval[0], eval[1]))

predictions = model.predict(x=input)
binary_predictions = np.heaviside(predictions - 0.5, 1) # second argument: output i
binary_predictions = binary_predictions.reshape(target.shape)

abs_binary_errors = np.where(binary_predictions != target)[0].size # np.where retur
rel_binary_errors = abs_binary_errors / len(target)
print("\nnumber of binary errors: %d, error rate: %f, accuracy: %f" % (abs_binary_e
```

Answer

1)

Output: 1st Tranining

final weights: (-32.864176, 9.796065)

final bias: 6.534341

Final loss: 0.355930, final accuray: 0.920000

number of binary errors: 12, error rate: 0.055600, accuracy: 0.920000

Output: 2nd Tranining

final weights: (-25.286375, 7.396742)

final bias: 5.090887

Final loss: 0.215550, final accuray: 0.945000

number of binary errors: 11, error rate: 0.058000, accuracy: 0.945000

Output: 3rd Tranining

final weights: (-25.686739, 8.692582)

final bias: 5.945707

Final loss: 0.269067, final accuray: 0.920000

number of binary errors: 10, error rate: 0.066000, accuracy: 0.

2)

configuration: solver=Adam, epoch=25, lr=0.1

final weights: (-31.854929, 11.147592)

final bias: 5.103813

Final loss: 0.282556, final accuray: 0.930000

number of binary errors: 11, error rate: 0.058000, accuracy: 0.930000

####** configuration: solver=Adam, epoch=45, lr=0.5**

final weights: (-78.309929, 24.959489)

final bias: 15.855894

Final loss: 0.245165, final accuray: 0.905000

number of binary errors: 21, error rate: 0.105000, accuracy: 0.905000

configuration: solver=Adam, epoch=100, lr=0.01

final weights: (-22.957848, 9.416083)

final bias: 3.290746

Final loss: 0.263088, final accuray: 0.935000

number of binary errors: 12, error rate: 0.070000, accuracy: 0.935000

Task c)

Repeat exercise b) with the training set *exercise3c_input.txt* and the targets *exercise3c_target.txt*. Those points have been generated from the input points of exercise b) by removing points from class 1 (i.e. those points the x-coordinate of which is below 0.35). Do not forget to modify the variables *class1* and *class2* to load the files *exercise3c_class1.txt* and *exercise3c_class1.txt*, respectively! Discuss the output of the training algorithm in terms of the resulting decision boundary and the final training error.

Answer

configuration: solver=Adam, epoch=30, lr=0.1

final weights: (-76.197903, 13.683178)

final bias: 19.684738

Final loss: 0.085936, final accuray: 0.958293 number of binary errors: 5, error rate: 0.021939, accuracy: 0.958293

configuration: solver=Adam, epoch=35, lr=0.5

final weights: (-95.036801, 16.9800853)

final bias: 28.787293

Final loss: 0.066991, final accuray: 0.959958

number of binary errors: 7, error rate: 0.0390048, accuracy: 0.959958

configuration: solver=Adam, epoch=35, lr=0.01

final weights: (-11.901829, 3.766939)

final bias: 3.761029

Final loss: 0.428174, final accuray: 0.971513

number of binary errors: 2, error rate: 0.020870, accuracy: 0.971513

Task d)

Divide the input samples from part b) into separate training and validation sets, where the latter shall comprise 30% of the data. You may use available Keras functionality for this purpose. Run the script at least two times, plot the training and validation loss and accuracy as a function of the epoch number and report on your findings.

Answer

Write your answer here.

Task e)

Modify the script to handle the XOR-problem, i.e. set

input = np.array([[0,0],[0,1],[1,0],[1,1]])
target = np.array([0, 1, 1, 0])

and plot the final decision boundary and the loss function. Report on your findings.

Answer

Write your answer here.

Exercise 4 (Multi-layer perceptron and backpropagation – small datasets)

The goal of this exercise is to apply a multi-layer perceptron (MLP), trained with the backpropagation algorithm as provided by Tensorflow Keras library, to four classification problems provided by the UCI repository (and contained in the scikit learn package; i.e. iris, digits, wine, breast_cancer) and two artificially generated classification problems (circles, moon). In particular, the influence of the backpropagation solver and of the network topology shall be investigated in parts a) and b) of the exercise, respectively.

Task a)

In this part of the exercise, a number of solvers (stochastic gradient descent, Adam, Adam with Nesterov momentum, AdaDelta, AdaGrad or RMSProp) shall be applied to the six datasets. An (incomplete) python script for this experiment is provided the Jupyter notebook. Complete the code (model definition, selection and configuration of an optimizer and model "compilation" including selection of an appropriate loss function; see *# TO BE ADAPTED* in the Jupyter notebook); consult the Tensorflow Keras documentation if needed. Furthermore, select suitable values of the most important parameters (e.g. learning rate, batch size...). Then, apply the script for at least three different optimizers, for a suitable baseline model configuration. Report the final training and validation loss and accuracy values and provide plots for the training and validation loss and accuracy curves as a function of the number of epochs (see script). What are your conclusions regarding the comparison of the optimization strategies? Also report on the database statistics.

The optimizer is selected e.g. with
opt = SGD(learning_rate=lr) # SGD or Adam, Nadam, Adadelta, Adagrad, RMSProp
Note that additional parameters of the optimizers can be set if desired (see the Tensorflow Keras documentation).

In [ ]:

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from os.path import join
from tensorflow.keras.layers import Dense
from tensorflow.keras import Model, Input
from tensorflow.keras.optimizers import SGD, Adam, Adadelta, Adagrad, Nadam, RMSpro
from tensorflow.keras.utils import normalize
from sklearn import datasets

###--------
# load data
###--------

data_sets = ['iris', 'digits', 'wine', 'breast_cancer', 'circles', 'moons']
histories = {}
final_training_loss = {}
final_training_accuracy = {}
final_validation_loss = {}
final_validation_accuracy = {}

for name in data_sets:
  print("\nProcessing data set %s" % name)
  if name == 'iris':
    iris = datasets.load_iris()
    input = iris.data
    target = iris.target
  elif name == 'digits':
    digits = datasets.load_digits()
    input = digits.data
    target = digits.target
  elif name == 'wine':
    wine = datasets.load_wine()
    input = wine.data
    target = wine.target
  elif name == 'breast_cancer':
    breast_cancer = datasets.load_breast_cancer()
    input = breast_cancer.data
    target = breast_cancer.target
  elif name == 'circles':
    circles = datasets.make_circles(noise=0.2, factor=0.5, random_state=1)
    input = circles[0]
    target = circles[1]
  elif name == 'moons':
    moons = datasets.make_moons(noise=0.3, random_state=0)
    input = moons[0]
    target = moons[1]
  else:
    print("name %s unknown" % name)
  input_dim = input.shape[1]
  print("input dimension: %d" % input_dim)
  print("input shape: " + str(input.shape))
  print("target shape: " + str(target.shape))
  num_classes = len(np.unique(target))
  print("number of classes: %d" % num_classes)
  print("class labels: " + str(np.unique(target)))

  ###-----------
  # process data
```

```
###-----------

# shuffle data
data = np.column_stack((input, target))
np.random.shuffle(data)
input = data[:,np.arange(input_dim)] # columns 0 ... input_dim - 1 (contain input
target = data[:,input_dim] # column input_dim (contains targets)

# normalize inputs
mean = np.mean(input)
std = np.std(input, ddof=1)
input = (input - mean) / std

# if necessary, transform labels to be in range 0 ... num_classes - 1
labels_for_one_hot = {}
for i in range(num_classes):
  labels_for_one_hot[np.unique(target)[i]] = i

# one-hot encoding
def one_hot(j):
  vec = np.zeros(num_classes)
  vec[j] = 1
  return vec

# transform targets to one-hot encoding
target_one_hot = np.zeros((len(target), num_classes))
for i in range(len(target)):
  target_one_hot[i] = one_hot( labels_for_one_hot[int(target[i])] )

###-----------
# define model
###-----------

# Define the structure of the neural network
num_inputs = input_dim
num_hidden = 100 # TO BE ADAPTED
num_outputs = num_classes
input_layer = Input(shape=(num_inputs,), name='input') # two dimensional input

hidden_1 = Dense(units=num_hidden, activation="relu", name="hidden_layer")(input_
hidden_2 = Dense(units=num_hidden, activation="relu", name="hidden_layer2")(hidde
out = Dense(units=num_classes, activation=("sigmoid" if num_classes==2 else "soft

# create a model
model = Model(input_layer, out)

# show how the model looks
model.summary()

# compile the model
lr = 0.01 # TO BE ADAPTED
opt = SGD(learning_rate=lr) # TO BE ADAPTED # SGD or Adam, Nadam, Adadelta, Adagr
loss_function = "binary_crossentropy" if num_classes==2 else "categorical_crossen
model.compile(optimizer=opt,loss=loss_function ,metrics=["categorical_accuracy"])

###-------
# training
###-------

# Train the model
num_epochs = 100 # TO BE ADAPTED
```

```
  batch_size = 1 # TO BE ADAPTED
  history = model.fit(x=input, y=target_one_hot, batch_size=batch_size, epochs=num_
  histories[name] = history
  final_training_loss[name] = history.history['loss'][num_epochs-1]
  final_training_accuracy[name] = history.history['categorical_accuracy'][num_epoch
  final_validation_loss[name] = history.history['val_loss'][num_epochs-1]
  final_validation_accuracy[name] = history.history['val_categorical_accuracy'][num

for name in data_sets:
  print("\n%s:\n" % name)
  print("final training loss: %f" % final_training_loss[name])
  print("final training accuracy: %f" % final_training_accuracy[name])
  print("final validation loss: %f" % final_validation_loss[name])
  print("final validation accuracy: %f" % final_validation_accuracy[name])

###-----------
# plot results
###-----------

# plot setup
fig, axes = plt.subplots(3, 2, figsize=(15, 10))
fig.tight_layout() # improve spacing between subplots, doesn't work
plt.subplots_adjust(left=0.125, right=0.9, bottom=0.1, top=0.9, wspace=0.2, hspace=
legend = []
i = 0
axes_indices = {0 : (0,0), 1 : (0,1), 2: (1,0), 3: (1,1), 4: (2,0), 5: (2,1)}

for name in data_sets:
  # plot loss
  axes[axes_indices[i]].set_title(name)
  if i == 4 or i == 5:
    axes[axes_indices[i]].set_xlabel('Epoch number')
  axes[axes_indices[i]].set_ylim(0, 1)
  axes[axes_indices[i]].plot(histories[name].history['loss'], color = 'blue',
            label = 'training loss')
  axes[axes_indices[i]].plot(histories[name].history['val_loss'], color = 'red',
            label = 'validation loss')
  axes[axes_indices[i]].legend()

  # plot accuracy
  axes[axes_indices[i]].plot(histories[name].history['categorical_accuracy'], color
            label = 'training accuracy')
  axes[axes_indices[i]].plot(histories[name].history['val_categorical_accuracy'], c
            label = 'validation accuracy')
  axes[axes_indices[i]].legend()
  i = i + 1


# show the plot
plt.show();
```

Answer
Iteration 1 , Optimizer : SGD, Learning Rate : 0.01####

```
**iris:**

final training loss: 0.072402

final training accuracy: 0.980952
```

final validation loss: 0.050727

final validation accuracy: 0.977778

****digits:****

final training loss: 0.000850

final training accuracy: 1.000000

final validation loss: 0.104347

final validation accuracy: 0.977778

****wine:****

final training loss: 0.566051

final training accuracy: 0.709677

final validation loss: 0.573185

final validation accuracy: 0.666667

****breast_cancer:****

final training loss: 0.152266

final training accuracy: 0.949749

final validation loss: 0.249697

final validation accuracy: 0.883041

****circles:****

final training loss: 0.332946

final training accuracy: 0.885714

final validation loss: 0.327478

final validation accuracy: 0.900000

****moons:****

final training loss: 0.263171

final training accuracy: 0.885714

final validation loss: 0.318069

final validation accuracy: 0.900000

`![IMAGE: SGD](images/4_SGD.png)`

Iteration 2 , Optimization Method: Adadelta, Training rate: 0.01

iris:

final training loss: 0.579217

final training accuracy: 0.790476

final validation loss: 0.605583

final validation accuracy: 0.800000

digits:

final training loss: 0.190644

final training accuracy: 0.951472

final validation loss: 0.211463

final validation accuracy: 0.940741

wine:

final training loss: 0.839041

final training accuracy: 0.604839

final validation loss: 0.894024

final validation accuracy: 0.666667

breast_cancer:

final training loss: 0.234960

final training accuracy: 0.919598

final validation loss: 0.290192

final validation accuracy: 0.894737

circles:

final training loss: 0.663483

final training accuracy: 0.542857

final validation loss: 0.681810

final validation accuracy: 0.466667

moons:

final training loss: 0.513655

final training accuracy: 0.800000

final validation loss: 0.580649

final validation accuracy: 0.766667

`![IMAGE: Adadelta](images/Adadelta.png)`

Iteration 3 , Opt : RMSProp, Iteration rate : 0.01

iris:

final training loss: 0.063413

final training accuracy: 0.980952

final validation loss: 0.590287

final validation accuracy: 0.911111

digits:

final training loss: 0.000000

final training accuracy: 1.000000

final validation loss: 0.522021

final validation accuracy: 0.981481

wine:

final training loss: 0.752013

final training accuracy: 0.733871

final validation loss: 0.749712

final validation accuracy: 0.777778

breast_cancer:

final training loss: 0.212206

final training accuracy: 0.934673

final validation loss: 0.149893

final validation accuracy: 0.959064

circles:

final training loss: 0.176986

final training accuracy: 0.942857

```
final validation loss: 0.723873

final validation accuracy: 0.900000


**moons:**

final training loss: 0.188970

final training accuracy: 0.985714

final validation loss: 0.178707

final validation accuracy: 0.966667

![IMAGE: RMSprop](images/RMSprop.png)



#### Conclusion ####
According to the graphs, we can say that Optimization strategy **SGD** is best
among three of them.
```

Task b)

Using the most successful optimizer from part a), in this part of the exercise different network topologies shall be investigated, i.e. the number of hidden layers and of hidden neurons shall be varied. To this end, modify the python script accordingly and systematically test the network performance. Provide the final training and validation loss and accuracy and provide the loss and accuracy curves as function of the number of epochs. You may also test further parameter settings. What are your conclusions regarding the network topology?

Answer

Opt: SGD, Hidden Layer : 1, Hidden Neurons : 50

iris:

final training loss: 0.047442

final training accuracy: 0.990476

final validation loss: 0.170617

final validation accuracy: 0.911111

digits:

final training loss: 0.001037

final training accuracy: 1.000000

final validation loss: 0.071326

final validation accuracy: 0.981481

wine:

final training loss: 0.546695

final training accuracy: 0.733871

final validation loss: 0.585359

final validation accuracy: 0.611111

breast_cancer:

final training loss: 0.154872

final training accuracy: 0.939699

final validation loss: 0.208050

final validation accuracy: 0.918129

circles:

final training loss: 0.281645

final training accuracy: 0.914286

final validation loss: 0.336740

final validation accuracy: 0.833333

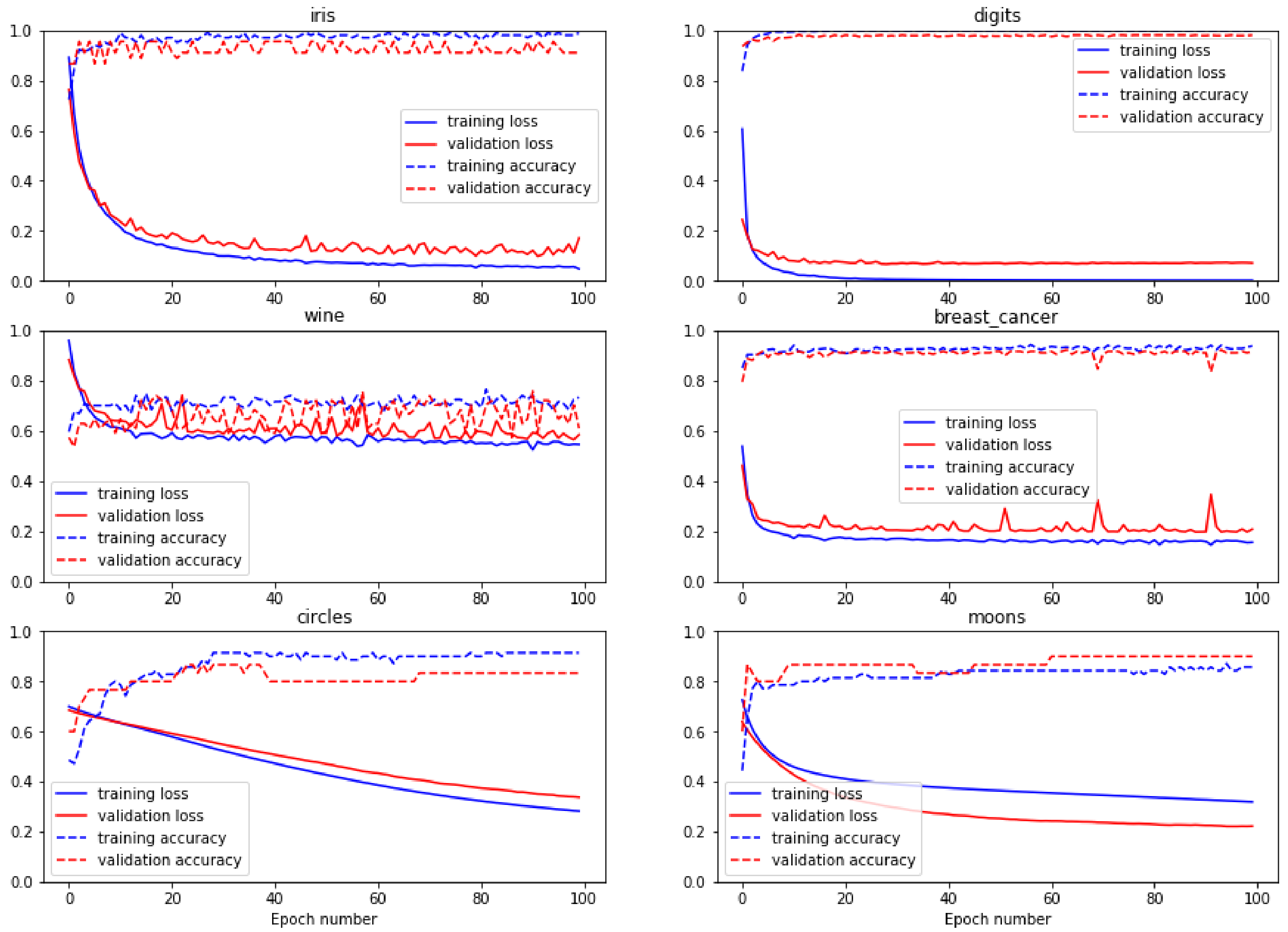
moons:

final training loss: 0.318536

final training accuracy: 0.857143

final validation loss: 0.221821

final validation accuracy: 0.900000



Opt: SGD, Hidden Layer : 1, Hidden Neurons : 100

iris:

final training loss: 0.072402

final training accuracy: 0.980952

final validation loss: 0.050727

final validation accuracy: 0.977778

digits:

final training loss: 0.000850

final training accuracy: 1.000000

final validation loss: 0.104347

final validation accuracy: 0.977778

wine:

final training loss: 0.566051

final training accuracy: 0.709677

final validation loss: 0.573185

final validation accuracy: 0.666667

breast_cancer:

final training loss: 0.152266

final training accuracy: 0.949749

final validation loss: 0.249697

final validation accuracy: 0.883041

circles:

final training loss: 0.332946

final training accuracy: 0.885714

final validation loss: 0.327478

final validation accuracy: 0.900000

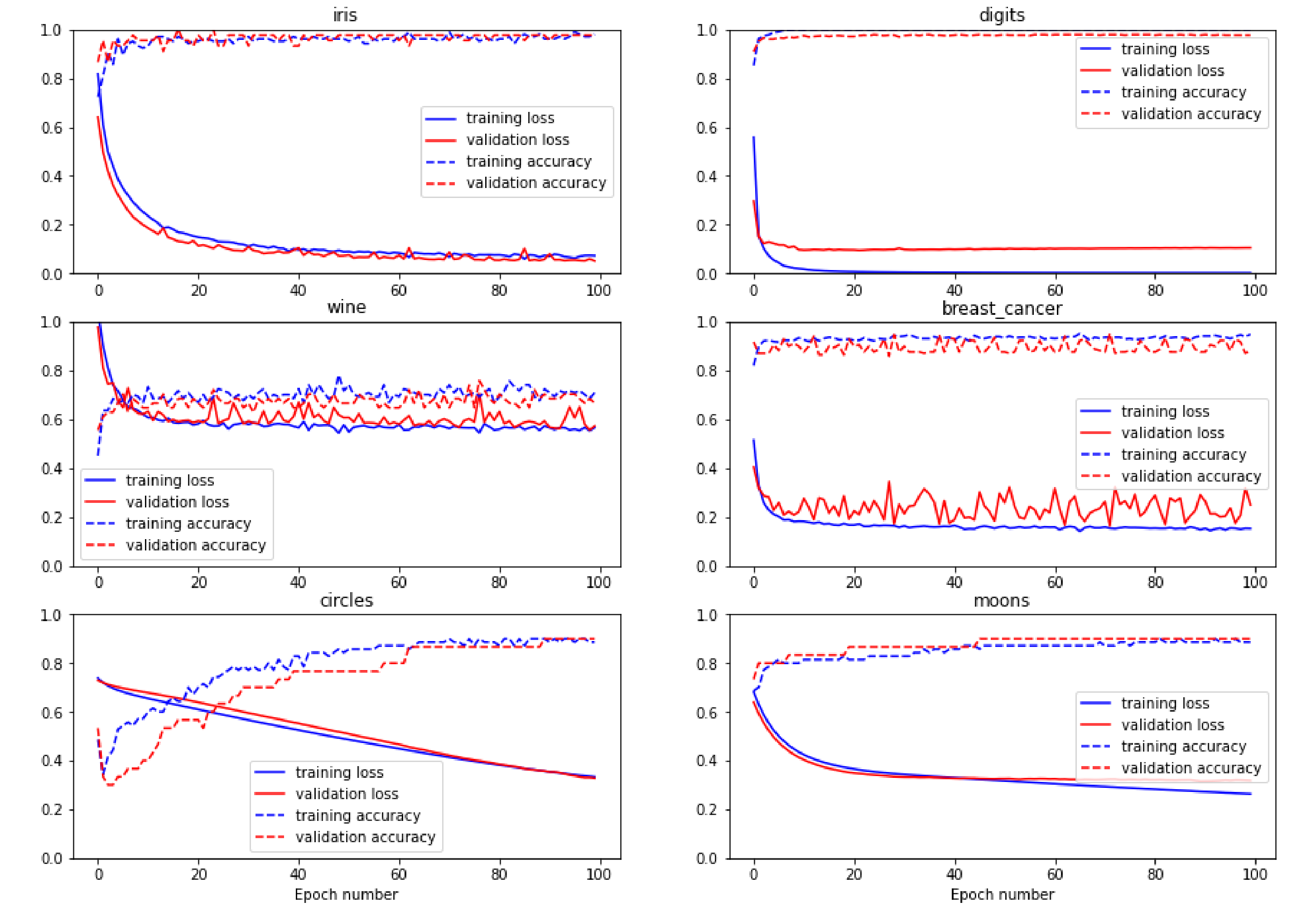
moons:

final training loss: 0.263171

final training accuracy: 0.885714

final validation loss: 0.318069

final validation accuracy: 0.900000



Opt: SGD, Hidden Layer : 1, Hidden Neurons : 150

iris:

final training loss: 0.069464

final training accuracy: 0.980952

final validation loss: 0.057866

final validation accuracy: 1.000000

**digits:

final training loss: 0.000863

final training accuracy: 1.000000

final validation loss: 0.041053

final validation accuracy: 0.981481

**wine:

final training loss: 0.528201

final training accuracy: 0.741935

final validation loss: 0.639217

final validation accuracy: 0.648148

**breast_cancer:

final training loss: 0.181956

final training accuracy: 0.917085

final validation loss: 0.154664

final validation accuracy: 0.929825

**circles:

final training loss: 0.283462

final training accuracy: 0.928571

final validation loss: 0.374384

final validation accuracy: 0.866667

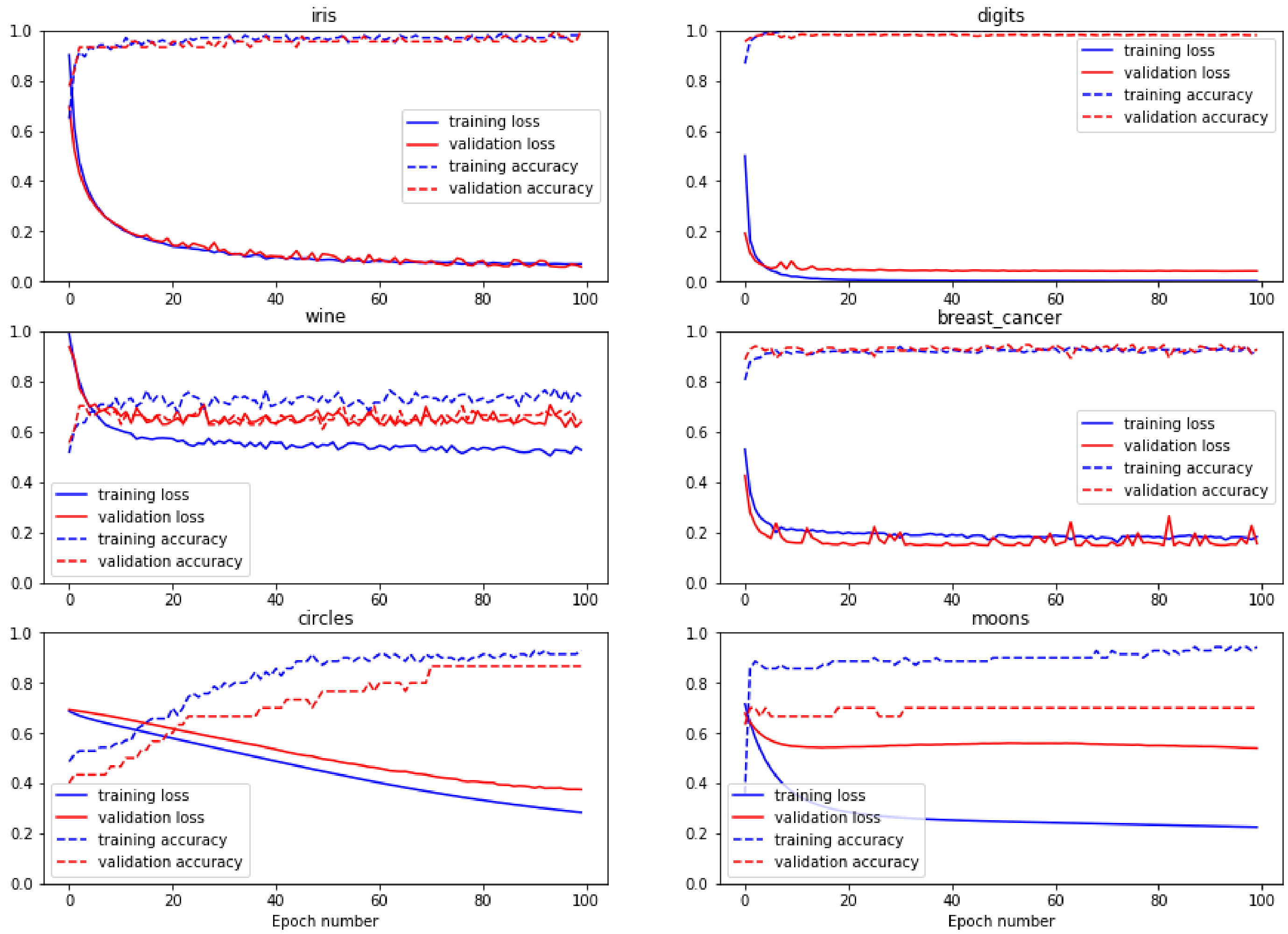
**moons:

final training loss: 0.224648

final training accuracy: 0.942857

final validation loss: 0.539297

final validation accuracy: 0.700000



Opt : SGD, Hidden Layer : 2 , Neurons: 150

Processing data set iris input dimension: 4 input shape: (150, 4) target shape: (150,) number of classes: 3 class labels: [0 1 2] Model: "model_12"

Layer (type) Output Shape Param

=== input (InputLayer) [(None, 4)] 0

---

hidden_layer (Dense) (None, 150) 750

---

hidden_layer2 (Dense) (None, 150) 22650

---

final_output (Dense) (None, 3) 453

=== Total params: 23,853

===

iris:

final training loss: 0.078948

final training accuracy: 0.961905

final validation loss: 0.044705

final validation accuracy: 1.000000

digits:

final training loss: 0.000216

final training accuracy: 1.000000

final validation loss: 0.109893

final validation accuracy: 0.977778

wine:

final training loss: 0.582043

final training accuracy: 0.669355

final validation loss: 0.527156

final validation accuracy: 0.814815

breast_cancer:

final training loss: 0.170408

final training accuracy: 0.924623

final validation loss: 0.184235

final validation accuracy: 0.929825

circles:

final training loss: 0.230768

final training accuracy: 0.885714

final validation loss: 0.155399

final validation accuracy: 0.933333

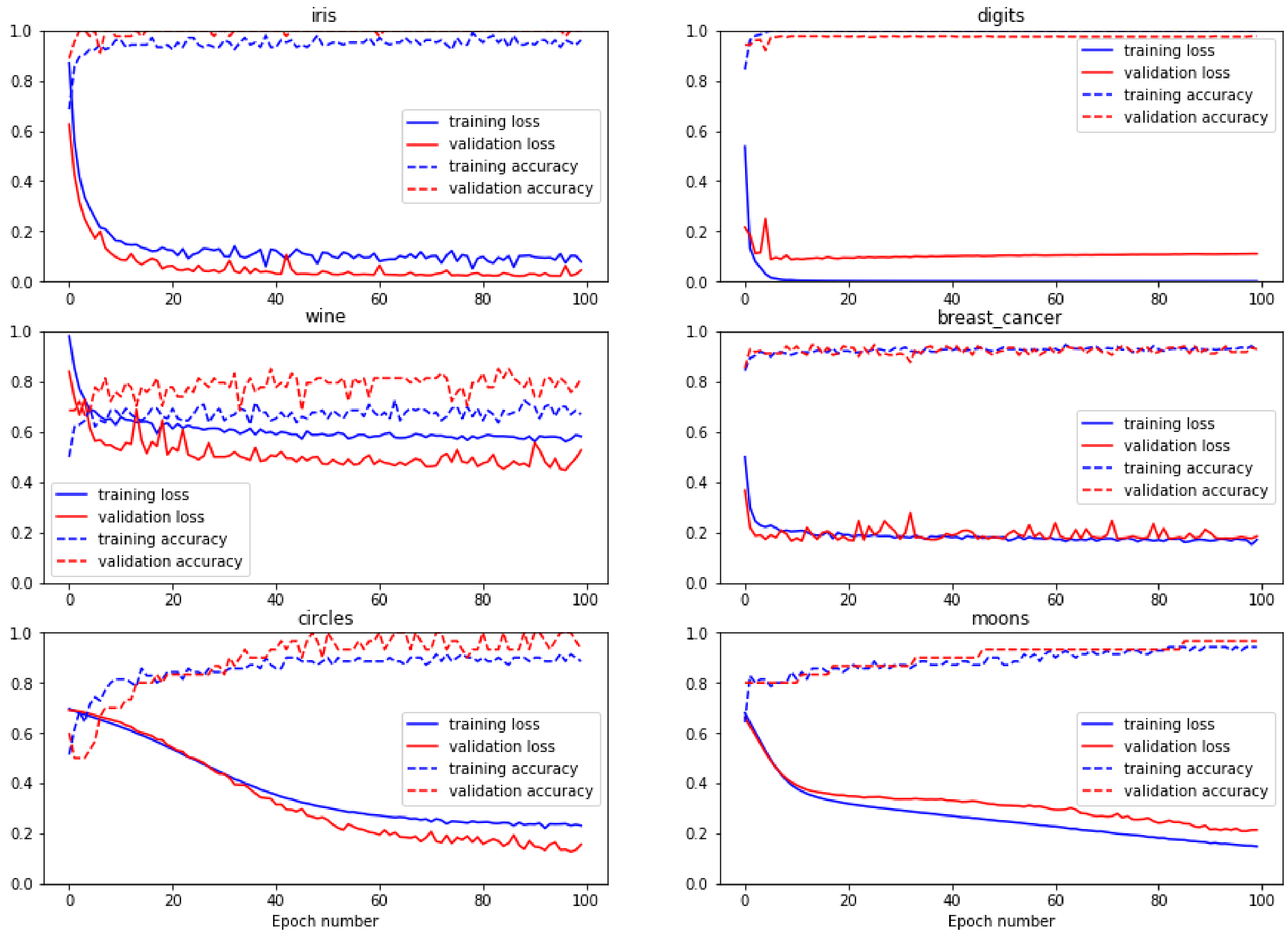
moons:

final training loss: 0.147394

final training accuracy: 0.942857

final validation loss: 0.213510

final validation accuracy: 0.966667



Opt: SGD , Hidden layer : 2, Neurons: 100

iris:

final training loss: 0.043750

final training accuracy: 0.971429

final validation loss: 0.019908

final validation accuracy: 1.000000

digits:

final training loss: 0.000223

final training accuracy: 1.000000

final validation loss: 0.108348

final validation accuracy: 0.979630

wine:

final training loss: 0.523984

final training accuracy: 0.709677

final validation loss: 0.622331

final validation accuracy: 0.740741

breast_cancer:

final training loss: 0.172765

final training accuracy: 0.932161

final validation loss: 0.172633

final validation accuracy: 0.929825

circles:

final training loss: 0.163506

final training accuracy: 0.914286

final validation loss: 0.316757

final validation accuracy: 0.933333

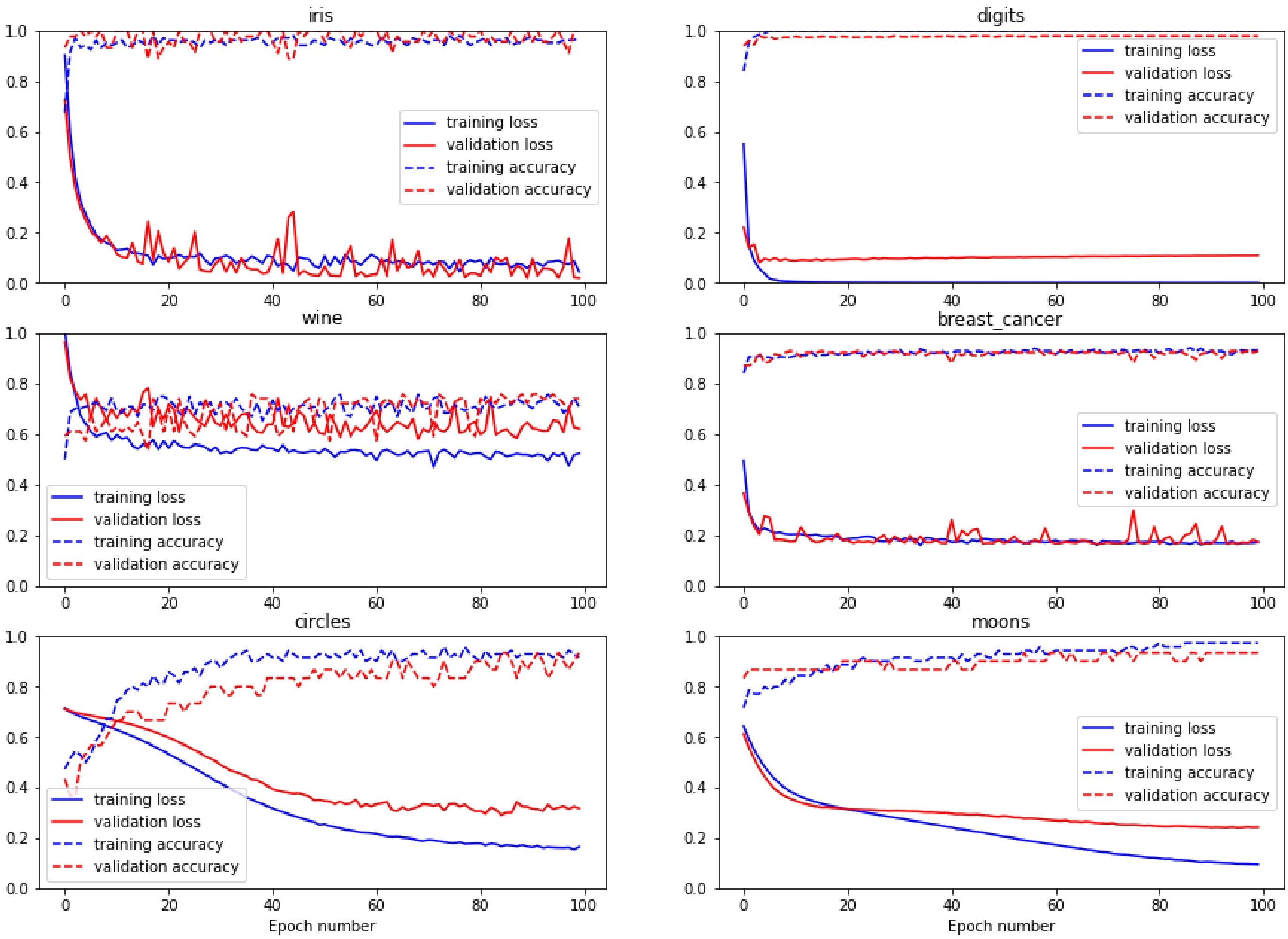
moons:

final training loss: 0.094640

final training accuracy: 0.971429

final validation loss: 0.241356

final validation accuracy: 0.933333



Accuracy Table (HL = # hidden layer, N = # of Neurons)

Sample	HL 1, N 50	HL 1, N 100	HL 1, N 150	HL 2, N 100	HL 1, N 150
iris	0.990476	0.980952	0.980952	.971429	0.961905
digits	1.000000	1.000000	1.000000	1.000000	1.000000
wine	0.733871	0.709677	0.741935	0.709677	0.669355
breast_cancer	0.939699	0.949749	0.917085	0.932161	0.924623
circles	0.914286	0.885714	0.928571	0.914286	0.885714
moons	0.857143	0.885714	0.942857	0.971429	0.942857

Conclusion

According to the previoius table, We can say that(w.r.t Accuracy) , Network with Hidden Layer 1 and Number of Neuron 50 is the best network.