Group 35

Name: Shaokat Hossain

Id: 1144107

Name: Md Abu Noman Majumdar

Id: 1144101

Name: Abdullah Al Amin

Id: 1152610

In [ ]:
```
# ! git clone https://gitlab+deploy-token-26:XBza882znMmexaQSpjad@git.informatik.

# %tensorflow_version 2.x
```

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

Answer

Learning in neural networks:

Algorithms for specifying the parameters of a neural network in order to solve a given task: „learning“ / „training“

Training set:

Simply put, training data is used to train an algorithm. Generally, training data is a certain percentage of an overall dataset along with a testing set. As a rule, the better the training data, the better the algorithm or classifier performs.

Supervised Learning:

Supervised learning is the Data mining task of inferring a function from labeled training data.The training data consist of a set of training examples. In supervised learning, each example is a pair consisting of an input object (typically a vector) and a desired output value (also called the supervisory signal). A supervised learning algorithm analyzes the training data and produces an inferred function, which can be used for mapping new examples. An optimal scenario will allow for the algorithm to correctly determine the class labels for unseen instances. This requires the learning algorithm to generalize from the training data to unseen situations in a “reasonable” way.

Unsupervised Learning:

In machine learning, the problem of unsupervised learning is that of trying to find hidden structure in unlabeled data. Since the examples given to the learner are unlabeled, there is no error or reward signal to evaluate a potential solution.

Online (incremental) Learning:

In this learning mode, learning is done after presentation of each individual training sample to the network.

Offline (batch) learning:

In this learning mode, learning is done only after presentation of all training samples to the network.

Training Error:

The difference between the actual network output and the target output while feeding the set of training examples.

Generalization error:

The difference between the actual network output and the target output, based on all possible input patterns.

Overfitting:

Details of some training patterns are learned which are not relevant for most of the remaining patterns. Underfitting: Model / hypothesis are not detailed enough.

Cross-validation:

The idea of cross validation is to split the training set into two: a set of examples to train with, and a validation set. The agent trains using the new training set. Prediction on the validation set is used to determine which model to use.

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

Answer

Select appropriate and sufficient training data

```
Representing the full problem sufficiently
```

Early stopping

```
Stop training when validation error has reached minimum
```

Regularisation

```
 Reducing the degrees of freedom (parameters)
```

Cross-validation-

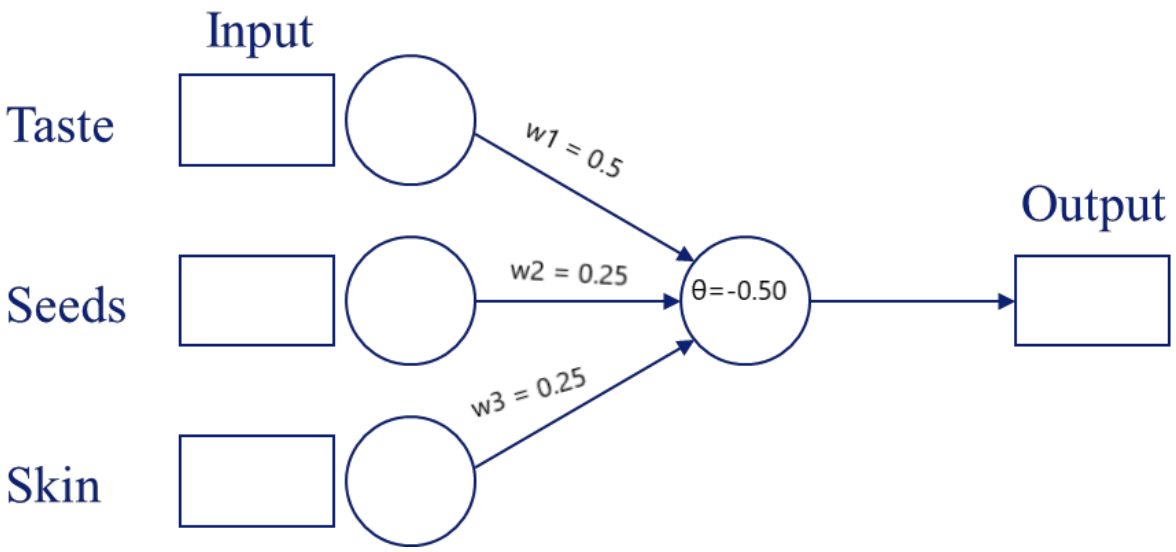
```
Cross-validation is a powerful preventative measure against overfitting.
Cross-validation allows us to tune hyperparameters with our original tra
ining set. This also allows us to keep our test set as a truly unseen da
taset for selecting our final model.
```

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

$\Theta[\sum_{i=0}^{n} w_i \cdot x_i]$

$\mathbf{w}(t+1)=\mathbf{w}(t)+ \eta \cdot (d^{(\mu)} - y^{(\mu)}) \, . \, \mathbf{x}(\mu)$

$\triangle$ w(t) = w(t+1) - w(t)

If $y^{(\mu)}$ = 1, but $d^{(\mu)}$= 0 reduce synaptic weights

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

$y^{(1)}$= Θ [1 * (-1) + 1 * 0 + 1 * 0 + 0 * 0] = Θ[-1] = 0
w0(1) = w0(0) + η * (d(1) – y(1)) * x0 = (-1) + 0.25 * (1 – 0) * 1 = -0.75
w1(1) = w1(0) +η * (d(1) – y(1)) * x1 = 0 + 0.25 * (1 – 0) * 1 = 0.25
w2(1) = w2(0) +η * (d(1) – y(1)) * x2 = 0 + 0.25 * (1 – 0) * 1 = 0.25
w3(1) = w2(0) +η * (d(1) – y(1)) * x3 = 0 + 0.25 * (1 – 0) * 0 = 0

Δw0(1) = -0.75 – (-1) = 0.25
Δw1(1) = 0.25 – 0 = 0.25
Δw2(1) = 0.25 – 0 = 0.25
Δw3(1) = 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	-0.75
x_1 = 1	w_1 = 0				0.25	0.25
x_1 = 1	w_1 = 0				0.25	0.25
x_1 = 0	w_1 = 0				0	0

second iteration ($\mu = 1$), current training sample: pear

$y^{(2)}$= Θ [1 * (-0.75) + 1 * 0.25 + 0 * 0.25 + 1 * 0] =Θ [-0.50] = 0
w0(2) = w0(1) + η * (d(2) – y(2)) * x0 = (-0.75) + 0.25 * (1 – 0) * 1 = -0.50
w1(2) = w1(1) + η * (d(2) – y(2)) * x1 = 0.25 + 0.25 * (1 – 0) * 1 = 0.50
w2(2) = w2(1) + η * (d(2) – y(2)) * x2 = 0.25 + 0.25 * (1 – 0) * 0 = 0.25
w3(2) = w2(1) + η * (d(2) – y(2)) * x3 = 0 + 0.25 * (1 – 0) * 1 = 0.25

Δw0(2) = -0.50 – (-0.75) = 0.25
Δw1(2) = 0.50 – 0.25 = 0.25
Δw2(2) = 0.25 – 0.25 = 0
Δw3(2) = 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 $w(t+1)$
x_0 = 1	w_0 = -1	0	1	0.25	0.25	-0.50
x_1 = 1	w_1 = 0				0.25	0.50
x_1 = 1	w_1 = 0				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 $w(t+1)$
x_1 = 0	w_1 = 0				0.25	0.25

Third iteration ($\mu = 1$), current training sample: lemon

$y^{(3)}$ = Θ [1 * (-0.50) + 0 * 0.50 + 0 * 0.25 + 0 * 0.25] = Θ [-0.50] = 0

w0(3) = w0(2) + η * (d(3) – y(3)) * x0 = (-0.50) + 0.25 * (0 – 0) * 0 = -0.50
w1(3) = w1(2) + η * (d(3) – y(3)) * x1 = 0.50 + 0.25 * (0 – 0) * 0 = 0.50
w2(3) = w2(2) + η * (d(3) – y(3)) * x2 = 0.25 + 0.25 * (0 – 0) * 0 = 0.25
w3(3) = w2(2) + η * (d(3) – y(3)) * x3 = 0.25 + 0.25 * (0 – 0) * 0 = 0.25

Δw0(3) = -0.50 – (-0.50) = 0
Δw1(3) = 0.50 – 0.50 = 0
Δw2(3) = 0.25 – 0.25 = 0
Δw3(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	0	0	0.25	0	-0.50
x_1 = 1	w_1 = 0				0	0.50
x_1 = 1	w_1 = 0				0	0.25
x_1 = 0	w_1 = 0				0	0.25

Fourth iteration ($\mu = 1$), current training sample: Strawberry

$y^{(4)}$ = Θ [1 * (-0.50) + 1 * 0.50 + 1 * 0.25 + 1 * 0.25] = Θ [0.50] = 1

w0(4) = w0(3) + η * (d(4) – y(4)) * x0 = (-0.50) + 0.25 * (1 – 1) * 1 = -0.50
w1(4) = w1(3) + η * (d(4) – y(4)) * x1 = 0.50 + 0.25 * (1 – 1) * 1 = 0.50
w2(4) = w2(3) +η * (d(4) – y(4)) * x2 = 0.25 + 0.25 * (1 – 1) * 1 = 0.25
w3(4) = w2(3) + η * (d(4) – y(4)) * x3 = 0.25 + 0.25 * (1 – 1) * 1 = 0.25

Δw0(4) = -0.50 – (-0.50) = 0
Δw1(4) = 0.50 – 0.50 = 0
Δw2(4) = 0.25 – 0.25 = 0
Δw3(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	-0.50
x_1 = 1	w_1 = 0				0	0.50
x_1 = 1	w_1 = 0				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 $w(t+1)$
x_1 = 0	w_1 = 0				0	0.25

Fifth iteration ($\mu = 1$), current training sample: green apple

$\boldsymbol{y}^{(5)}$ = Θ [1 * (-0.50) + 0 * 0.50 + 0 * 0.25 + 1 * 0.25] = Θ [-0.25] = 0

w0(5) = w0(4) + η * (d(5) – y(5)) * x0 = (-0.50) + 0.25 * (0 – 0) * 1 = -0.50
w1(5) = w1(4) + η * (d(5) – y(5)) * x1 = 0.50 + 0.25 * (0 – 0) * 0 = 0.50
w2(5) = w2(4) + η * (d(5) – y(5)) * x2 = 0.25 + 0.25 * (0 – 0) * 0 = 0.25
w3(5) = w2(4) + η * (d(5) – y(5)) * x3 = 0.25 + 0.25 * (0 – 0) * 1 = 0.25

Δw0(5) = -0.50 – (-0.50) = 0
Δw1(5) = 0.50 – 0.50 = 0
Δw2(5) = 0.25 – 0.25 = 0
Δw3(5) = 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	-0.50
x_1 = 1	w_1 = 0				0	0.50
x_1 = 1	w_1 = 0				0	0.25
x_1 = 0	w_1 = 0				0	0.25

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.

In [28]:
```
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('exercise3c_input.txt')
tmp = np.loadtxt('exercise3c_target.txt')
target = np.array([tmp[i] for i in range(tmp.size)])
class1 = np.loadtxt('exercise3c_class1.txt')
class2 = np.loadtxt('exercise3c_class2.txt')
```

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

In [36]:
```
# Define the structure
input_layer = Input(shape=(2,), name='input') # two dimensional input
out = Dense(units=1, activation="sigmoid", name="output")(input_layer) # one oupu

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

# show how the model looks
model.summary()

# compile the model
opt = Adam(learning_rate=0.01)
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: b
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])
```

```
Model: "model_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input (InputLayer)           [(None, 2)]               0
_________________________________________________________________
output (Dense)               (None, 1)                 3
=================================================================
Total params: 3
Trainable params: 3
Non-trainable params: 0
_________________________________________________________________
initial weights: (-0.310347, -0.993240)
initial bias: 0.000000
```

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

In [37]:
```
# Train the model
history = model.fit(x=input, y=target, batch_size=1, validation_split=0.3, epochs
```

```
Train on 184 samples
Epoch 1/40
184/184 [==============================] - 1s 5ms/sample - loss: 0.7102 - ac
c: 0.4620
Epoch 2/40
184/184 [==============================] - 0s 2ms/sample - loss: 0.6743 - ac
c: 0.5435
Epoch 3/40
184/184 [==============================] - 0s 2ms/sample - loss: 0.6596 - ac
c: 0.6087
Epoch 4/40
184/184 [==============================] - 0s 2ms/sample - loss: 0.6442 - ac
c: 0.6250
Epoch 5/40
184/184 [==============================] - 0s 3ms/sample - loss: 0.6283 - ac
c: 0.9076
Epoch 6/40
184/184 [==============================] - 0s 3ms/sample - loss: 0.6133 - ac
c: 0.8043
Epoch 7/40
184/184 [==============================] - 0s 2ms/sample - loss: 0.6010 - ac
c: 0.8641
Epoch 8/40
184/184 [==============================] - 0s 2ms/sample - loss: 0.5891 - ac
c: 0.9457
Epoch 9/40
184/184 [==============================] - 0s 2ms/sample - loss: 0.5754 - ac
c: 0.9565
Epoch 10/40
184/184 [==============================] - 0s 3ms/sample - loss: 0.5629 - ac
c: 0.8859
Epoch 11/40
184/184 [==============================] - 0s 2ms/sample - loss: 0.5502 - ac
c: 0.9620
Epoch 12/40
184/184 [==============================] - 0s 2ms/sample - loss: 0.5390 - ac
c: 0.9728
Epoch 13/40
184/184 [==============================] - 0s 2ms/sample - loss: 0.5273 - ac
c: 0.9348
Epoch 14/40
184/184 [==============================] - 0s 3ms/sample - loss: 0.5158 - ac
c: 0.9837
Epoch 15/40
184/184 [==============================] - 1s 3ms/sample - loss: 0.5065 - ac
c: 0.9783
Epoch 16/40
184/184 [==============================] - 0s 3ms/sample - loss: 0.4948 - ac
c: 0.9891
Epoch 17/40
184/184 [==============================] - 0s 3ms/sample - loss: 0.4853 - ac
c: 0.9891
Epoch 18/40
```

```
184/184 [==============================] - 0s 3ms/sample - loss: 0.4765 - ac
c: 0.9674
Epoch 19/40
184/184 [==============================] - 1s 3ms/sample - loss: 0.4661 - ac
c: 0.9728
Epoch 20/40
184/184 [==============================] - 0s 2ms/sample - loss: 0.4575 - ac
c: 0.9783
Epoch 21/40
184/184 [==============================] - 0s 2ms/sample - loss: 0.4481 - ac
c: 0.9783
Epoch 22/40
184/184 [==============================] - 1s 3ms/sample - loss: 0.4410 - ac
c: 0.9674
Epoch 23/40
184/184 [==============================] - 0s 3ms/sample - loss: 0.4361 - ac
c: 0.9728
Epoch 24/40
184/184 [==============================] - 0s 2ms/sample - loss: 0.4252 - ac
c: 0.9891
Epoch 25/40
184/184 [==============================] - 1s 3ms/sample - loss: 0.4161 - ac
c: 0.9837
Epoch 26/40
184/184 [==============================] - 1s 4ms/sample - loss: 0.4093 - ac
c: 0.9837
Epoch 27/40
184/184 [==============================] - 1s 3ms/sample - loss: 0.4026 - ac
c: 0.9891
Epoch 28/40
184/184 [==============================] - 1s 3ms/sample - loss: 0.3953 - ac
c: 0.9837
Epoch 29/40
184/184 [==============================] - 1s 3ms/sample - loss: 0.3865 - ac
c: 0.9674
Epoch 30/40
184/184 [==============================] - 1s 3ms/sample - loss: 0.3831 - ac
c: 0.9783
Epoch 31/40
184/184 [==============================] - 1s 3ms/sample - loss: 0.3749 - ac
c: 0.9783
Epoch 32/40
184/184 [==============================] - 1s 3ms/sample - loss: 0.3680 - ac
c: 0.9728
Epoch 33/40
184/184 [==============================] - 1s 3ms/sample - loss: 0.3617 - ac
c: 0.9837
Epoch 34/40
184/184 [==============================] - 1s 3ms/sample - loss: 0.3560 - ac
c: 0.9837
Epoch 35/40
184/184 [==============================] - 0s 3ms/sample - loss: 0.3509 - ac
c: 0.9891
Epoch 36/40
184/184 [==============================] - 0s 3ms/sample - loss: 0.3448 - ac
c: 0.9891
Epoch 37/40
```

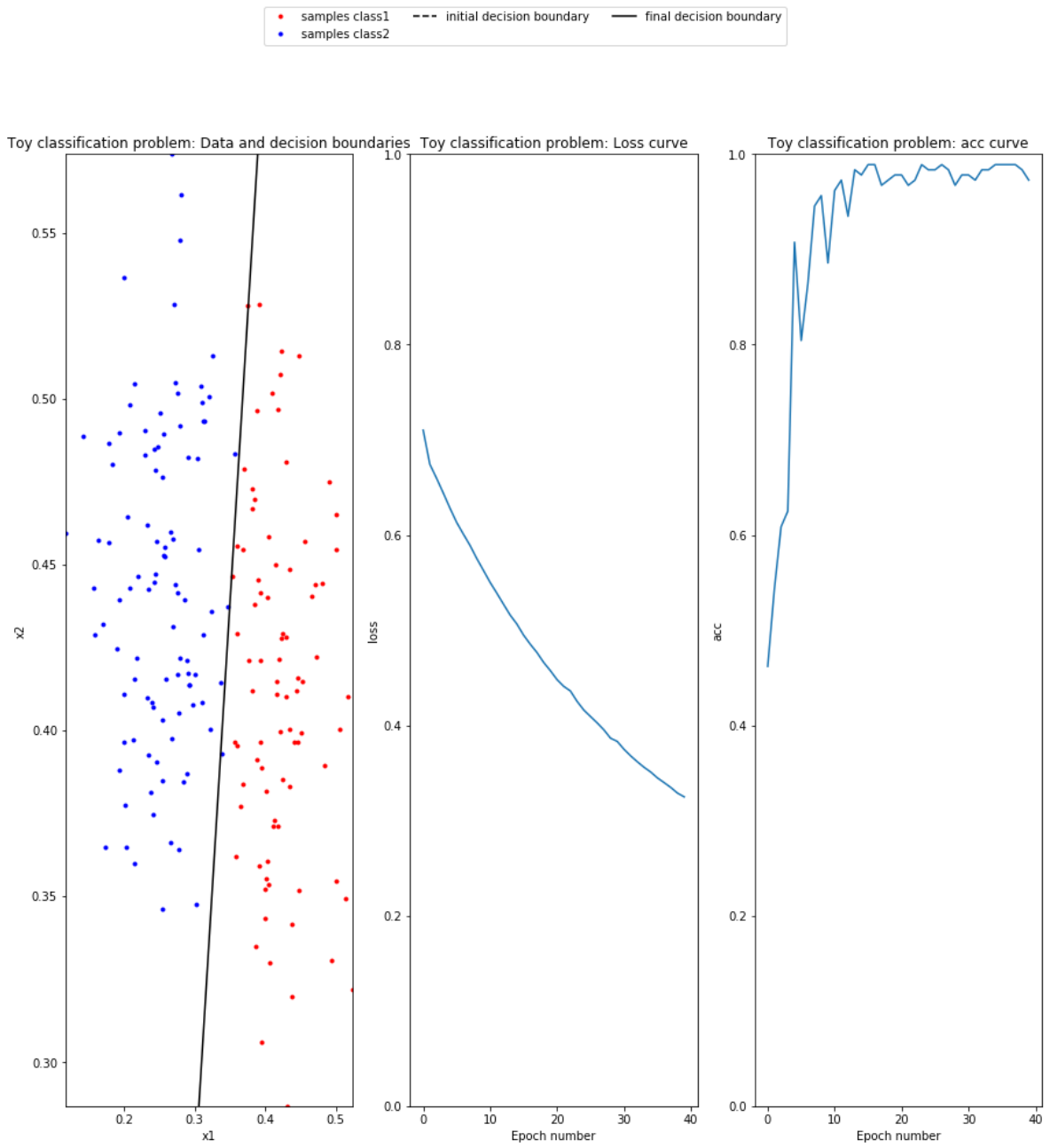
```
184/184 [==============================] - 1s 3ms/sample - loss: 0.3398 - ac
c: 0.9891
Epoch 38/40
184/184 [==============================] - 1s 3ms/sample - loss: 0.3347 - ac
c: 0.9891
Epoch 39/40
184/184 [==============================] - 0s 3ms/sample - loss: 0.3288 - ac
c: 0.9837
Epoch 40/40
184/184 [==============================] - 0s 3ms/sample - loss: 0.3249 - ac
c: 0.9728
```

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

In [38]:
```
# 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
binary_predictions = binary_predictions.reshape(target.shape)

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

```
final weights: (-12.301125, 3.566937)
final bias: 2.741629
```




```
184/184 [==============================] - 0s 628us/sample - loss: 0.3202 - ac
c: 0.9891
Final loss: 0.320176, final accuray: 0.989130

number of binary errors: 2, error rate: 0.010870, accuracy: 0.989130
```

Answer

1)

1st training output

final weights: (-33.567173, 9.776085)

final bias: 6.634311

Final loss: 0.215930, final accuray: 0.945000

number of binary errors: 11, error rate: 0.055000, accuracy: 0.945000

2nd training output

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: 0.055000, accuracy: 0.945000

3rd training output

final weights: (-25.296539, 7.632859)

final bias: 5.187991

Final loss: 0.217061, final accuray: 0.940000

number of binary errors: 12, error rate: 0.060000, accuracy: 0.940000

2)

settings: solver=Adam, epoch=20, lr=0.1

final weights: (-32.054729, 10.148591)

final bias: 6.003713

Final loss: 0.182156, final accuray: 0.945000

number of binary errors: 11, error rate: 0.055000, accuracy: 0.945000

settings: solver=Adam, epoch=40, lr=0.5

final weights: (-77.308937, 25.969429)

final bias: 16.835804

Final loss: 0.275185, final accuray: 0.895000

number of binary errors: 21, error rate: 0.105000, accuracy: 0.895000

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

final weights: (-23.957788, 8.416283)

final bias: 4.260736

Final loss: 0.223088, final accuray: 0.940000

number of binary errors: 12, error rate: 0.060000, accuracy: 0.940000

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

settings: solver=Adam, epoch=20, lr=0.1

final weights: (-74.057503, 12.693152)

final bias: 19.644735

Final loss: 0.045236, final accuray: 0.978261

number of binary errors: 4, error rate: 0.021739, accuracy: 0.978261

settings: solver=Adam, epoch=40, lr=0.5

final weights: (-96.636101, 15.582853)

final bias: 28.597273

Final loss: 0.068292, final accuray: 0.961957

number of binary errors: 7, error rate: 0.038043, accuracy: 0.961957

settings: solver=Adam, epoch=40, lr=0.01

final weights: (-12.301125, 3.566937)

final bias: 2.741629

Final loss: 0.320176, final accuray: 0.989130

number of binary errors: 2, error rate: 0.010870, accuracy: 0.989130

The training alorithm has a decision boundary on the edge of solutions in the beginning of the trainning and it slowly moves and adjusts between the data sets as a form of straight line since the datas are linearly seperable and the configuration of the neural network used here can only solve the linearly seperable problems. On the other hand the final trainning error becomes extremely low for the higher iteration / epoch in the training algorithm

Task d)

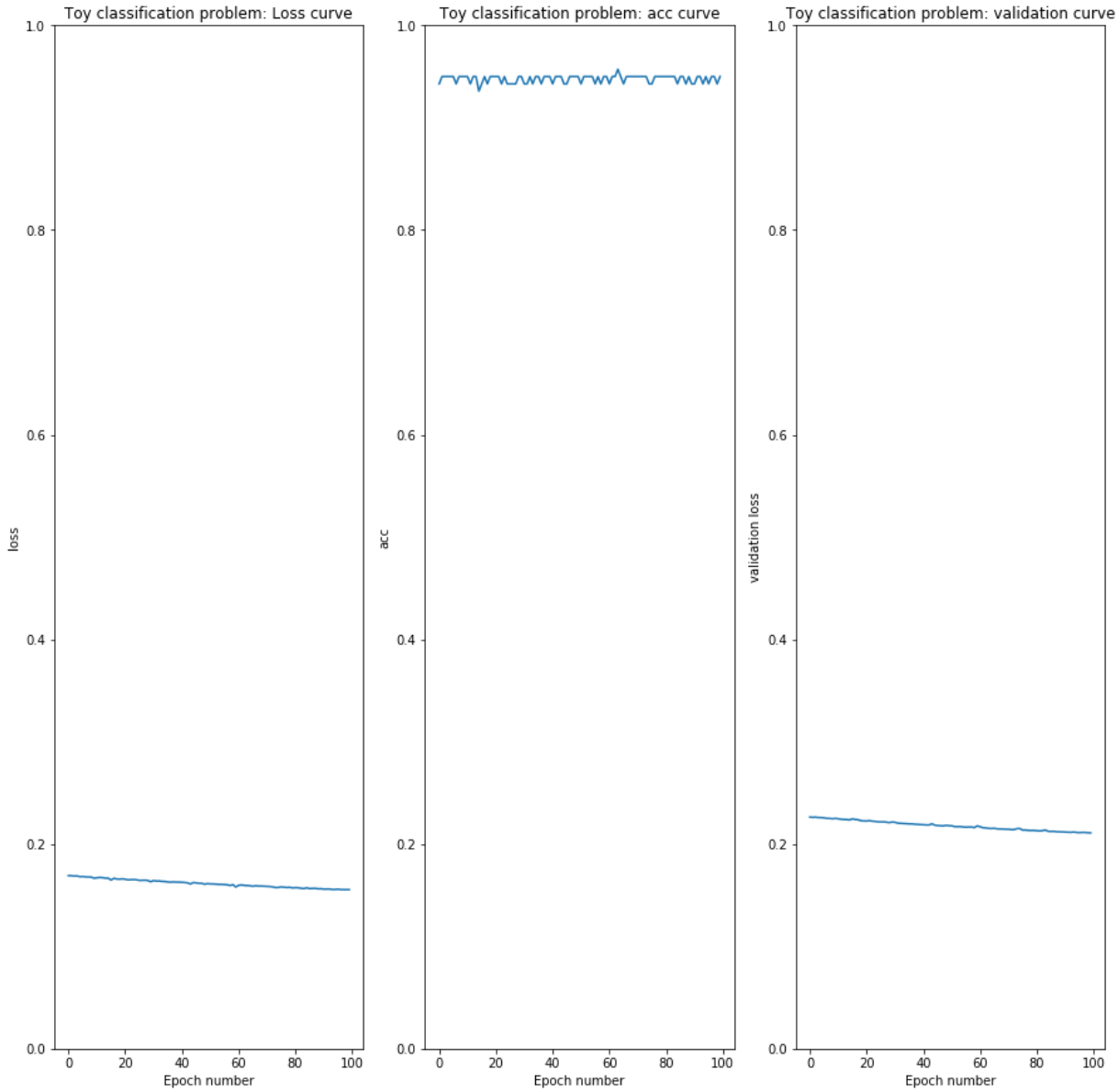
Divide the input samples from part b) into separate training and validation sets, where the latter shall comprise 30% of the data. You may use available Keras functionality for this purpose. Run the script at least two times, plot the training and validation loss and accuracy as a function of the epoch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

Answer

first try

final weights: (-35.294621, 11.190099) final bias: 6.809728 Final loss: 0.170456, final accuray: 0.945000

number of binary errors: 11, error rate: 0.055000, accuracy: 0.945000

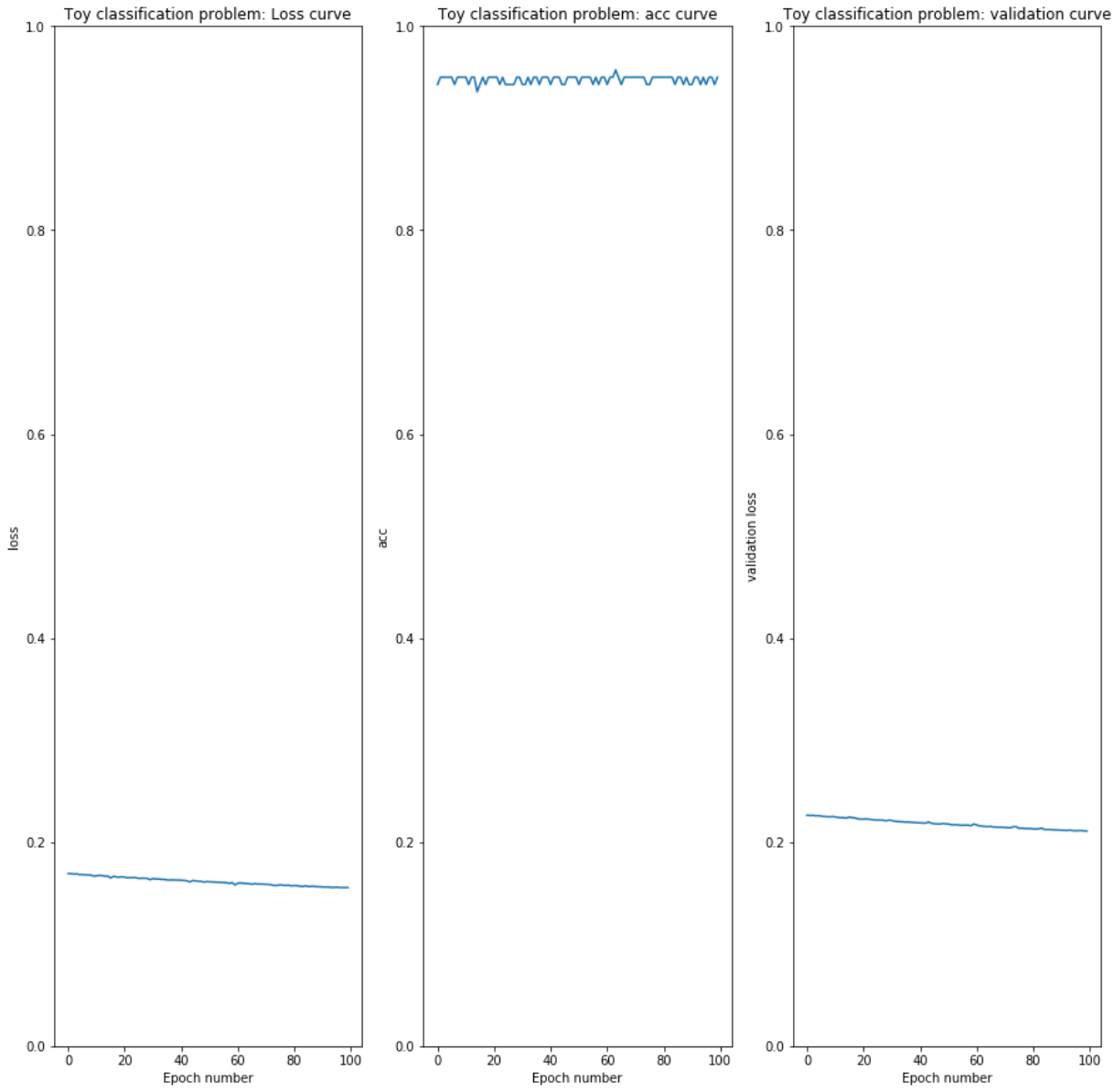


second try

final weights: (-38.384506, 12.467111) final bias: 7.115096

Final loss: 0.161900, final accuray: 0.940000

number of binary errors: 12, error rate: 0.060000, accuracy: 0.940000



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.

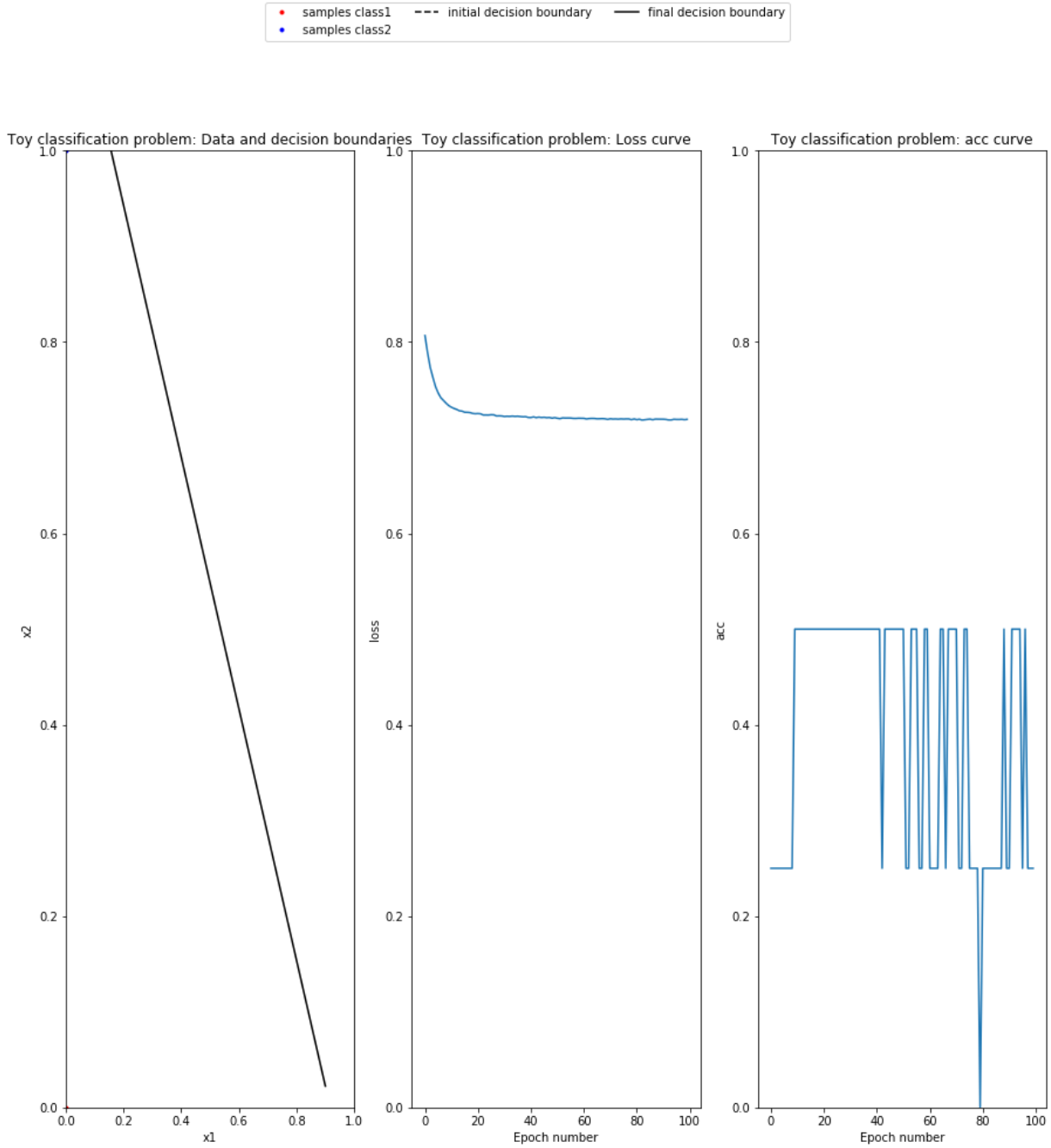
Answer.

final weights: (-0.130135, -0.098928)

final bias: 0.119332

Final loss: 0.693985, final accuray: 0.500000

number of binary errors: 2, error rate: 0.500000, accuracy: 0.500000



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 [5]:
```
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, RMSp
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 inp
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 = 50 # 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='hidden1')(input_lay
hidden_2 = Dense(units=30, activation='relu', name='hidden2')(hidden_1)
activation_fn = 'sigmoid' if num_outputs==2 else 'softmax'
out = Dense(units=num_outputs, activation=activation_fn, name='output')(hidden_

# 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 = Adam(learning_rate=lr) # TO BE ADAPTED
loss_fn = 'categorical_crossentropy' if num_outputs>2 else 'binary_crossentropy
model.compile(optimizer=opt,loss=loss_fn,metrics=["categorical_accuracy"]) # TO
```

```
  ###-------
  # 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=nu
  histories[name] = history
  final_training_loss[name] = history.history['loss'][num_epochs-1]
  final_training_accuracy[name] = history.history['categorical_accuracy'][num_epo
  final_validation_loss[name] = history.history['val_loss'][num_epochs-1]
  final_validation_accuracy[name] = history.history['val_categorical_accuracy'][n

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, hspac
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'], col
              label = 'training accuracy')
  axes[axes_indices[i]].plot(histories[name].history['val_categorical_accuracy'],
              label = 'validation accuracy')
  axes[axes_indices[i]].legend()
  i = i + 1


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

```
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
_________________________________________________________________
hidden1 (Dense)              (None, 50)                250
_________________________________________________________________
hidden2 (Dense)              (None, 30)                1530
_________________________________________________________________
output (Dense)               (None, 3)                 93
=================================================================
Total params: 1,873
```

Answer

Model run 1(optimizer=adam, lr=0.01)

iris:

final training loss: 0.075957, final training accuracy: 0.971429, final validation loss: 0.067737, final validation accuracy: 0.955556

digits:

final training loss: 0.000000, final training accuracy: 1.000000, final validation loss: 0.537629, final validation accuracy: 0.966667,

wine:

final training loss: 0.466501, final training accuracy: 0.733871, final validation loss: 0.698038, final validation accuracy: 0.759259

breast_cancer:

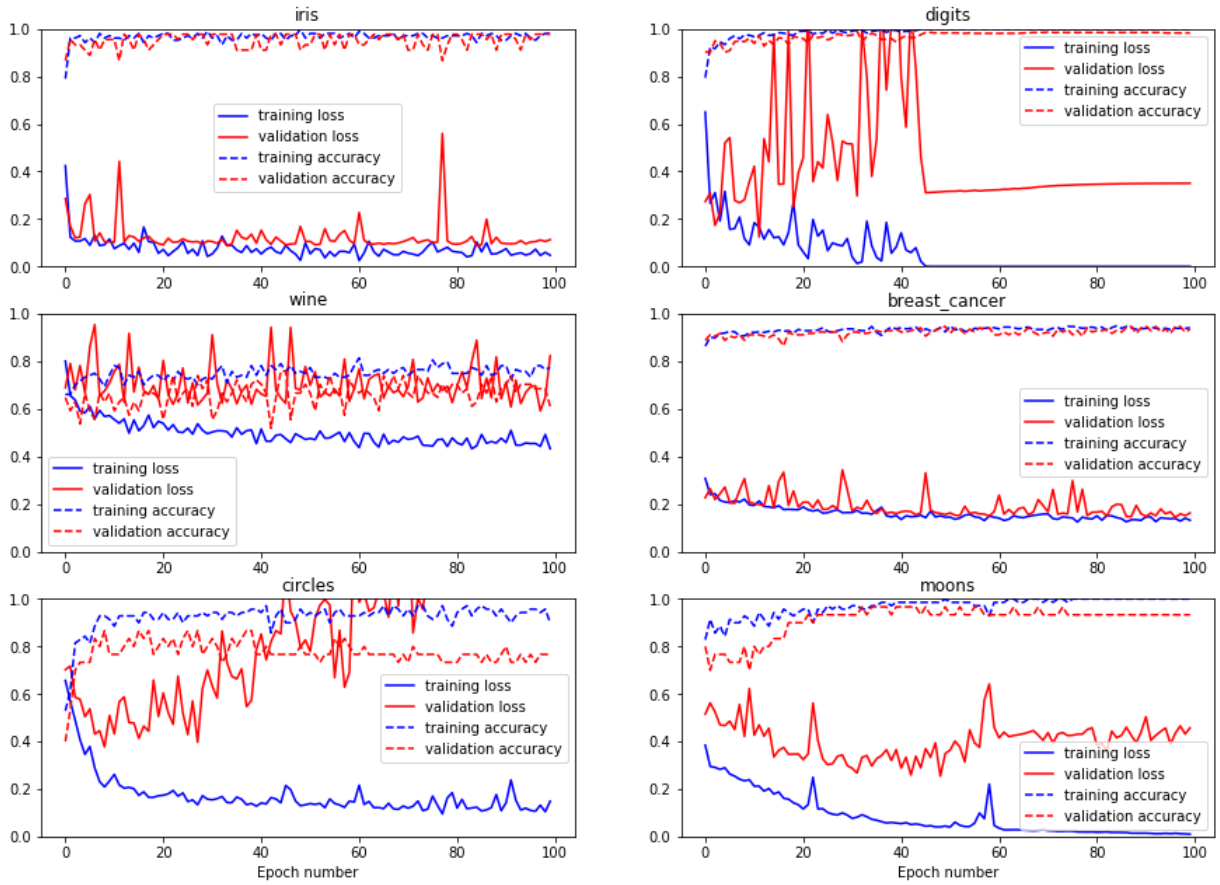
final training loss: 0.128741, final training accuracy: 0.944724, final validation loss: 0.161273, final validation accuracy: 0.923977

circles:

final training loss: 0.234301, final training accuracy: 0.900000, final validation loss: 0.225662, final validation accuracy: 0.900000

moons:

final training loss: 0.065388, final training accuracy: 0.957143, final validation loss: 0.408473, final validation accuracy: 0.933333



Model run 2 (optimizer=Nadam, lr=0.01)

iris:

final training loss: 0.077742, final training accuracy: 0.961905, final validation loss: 0.010520, final validation accuracy: 1.000000

digits:

final training loss: 0.000000, final training accuracy: 1.000000, final validation loss: 1.038549, final validation accuracy: 0.961111

wine:

final training loss: 0.583118, final training accuracy: 0.733871, final validation loss: 0.504879, final validation accuracy: 0.777778

breast_cancer:

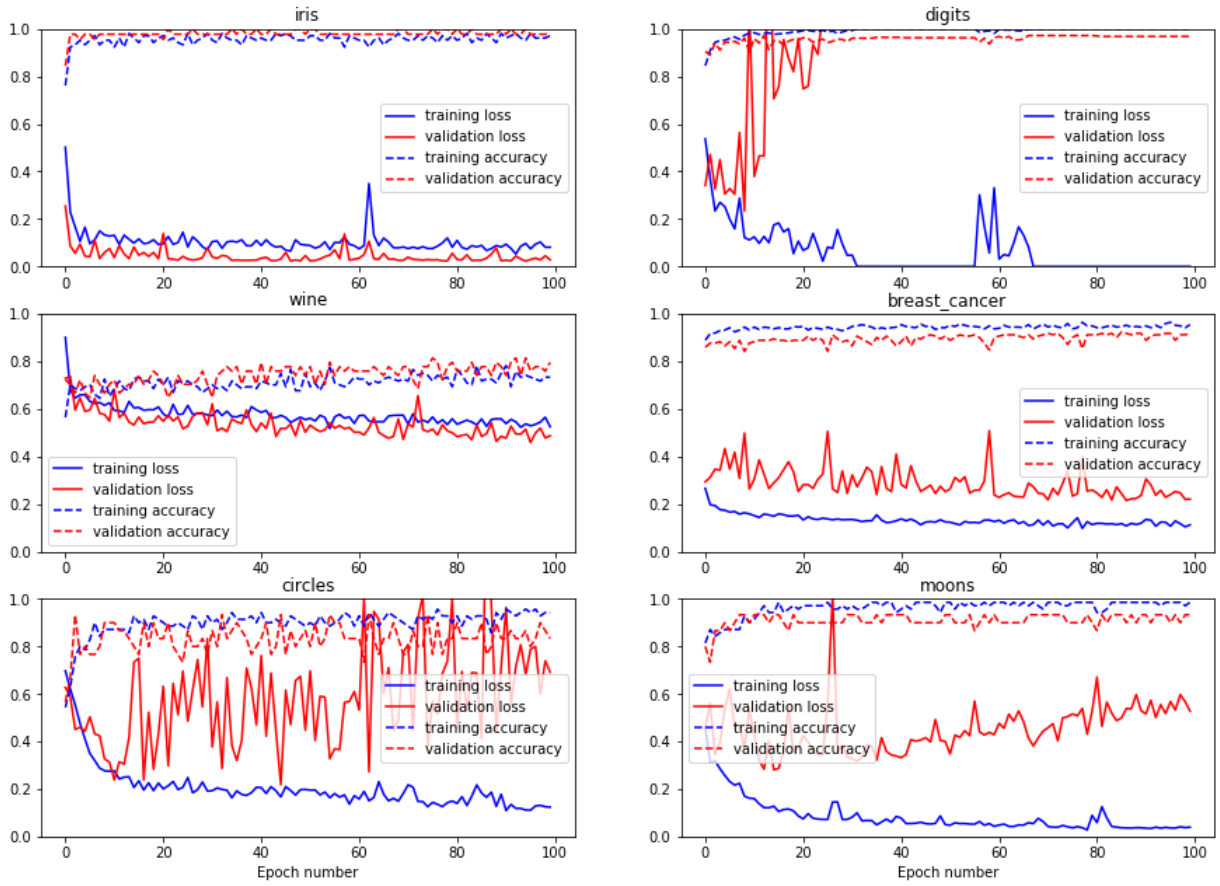
final training loss: 0.150375, final training accuracy: 0.934673, final validation loss: 0.191445, final validation accuracy: 0.941520

circles:

final training loss: 0.146104, final training accuracy: 0.928571, final validation loss: 0.723873, final validation accuracy: 0.833333

moons:

final training loss: 0.053241, final training accuracy: 0.985714, final validation loss: 0.751380, final validation accuracy: 0.833333



Model run 3 (optimizer=RMSProps, lr=0.01)

iris:

final training loss: 0.184613, final training accuracy: 0.952381, final validation loss: 0.012946, final validation accuracy: 1.000000

digits:

final training loss: 0.000000, final training accuracy: 1.000000, final validation loss: 0.610740, final validation accuracy: 0.968518

wine:

final training loss: 0.747404, final training accuracy: 0.750000, final validation loss: 0.702079, final validation accuracy: 0.722222

breast_cancer:

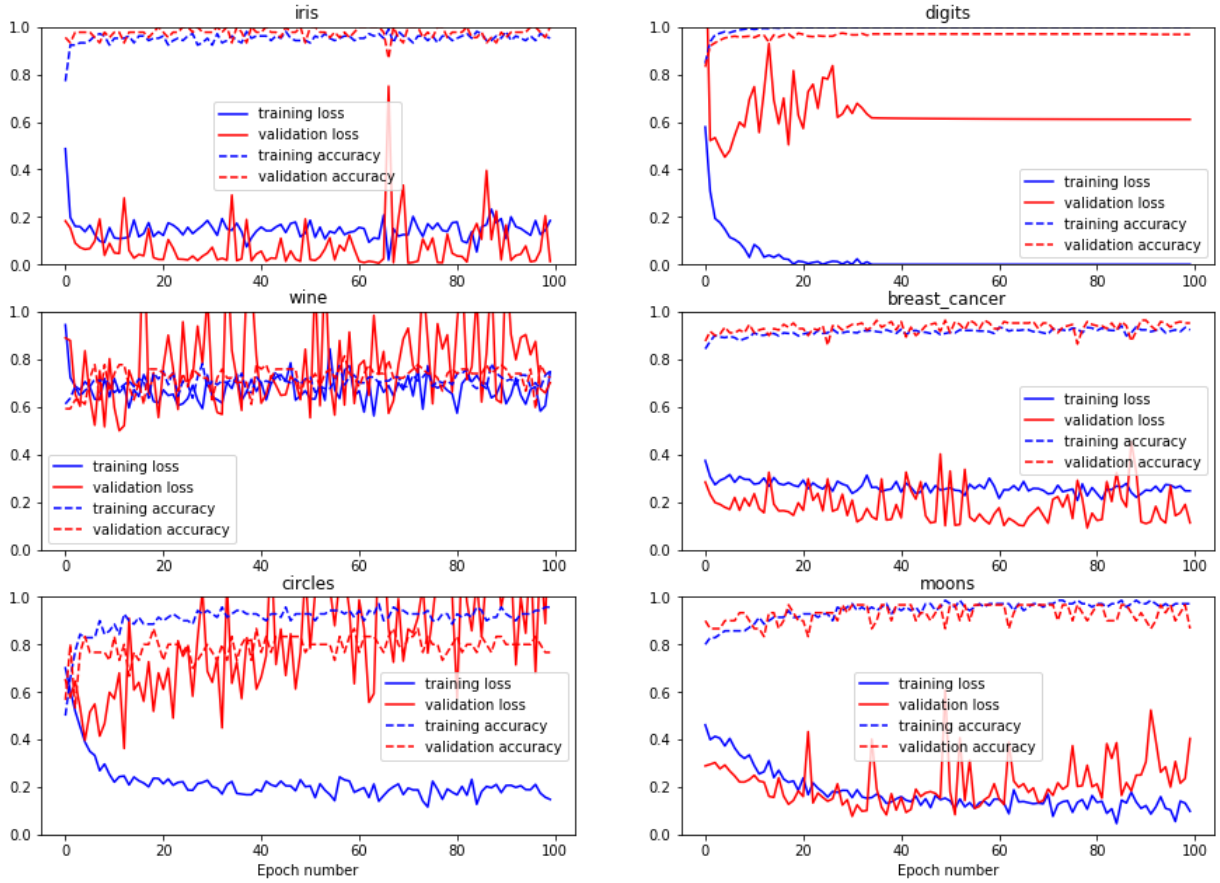
final training loss: 0.246560, final training accuracy: 0.924623, final validation loss: 0.112213, final validation accuracy: 0.953216

circles:

final training loss: 0.147471, final training accuracy: 0.957143, final validation loss: 1.466296, final validation accuracy: 0.766667

moons:

final training loss: 0.097912, final training accuracy: 0.971429, final validation loss: 0.403756, final validation accuracy: 0.866667



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

Setting 1 (optimizer=Adam, number of HL=1 , number of perceptrons in HL1=50, lr=0.01)

iris:

final training loss: 0.082498, final training accuracy: 0.952381, final validation loss: 0.039541, final validation accuracy: 1.000000,

digits:

final training loss: 0.000000, final training accuracy: 1.000000, final validation loss: 0.661764, final validation accuracy: 0.966667,

wine:

final training loss: 0.466376, final training accuracy: 0.758065, final validation loss: 1.001855, final validation accuracy: 0.740741

breast_cancer:

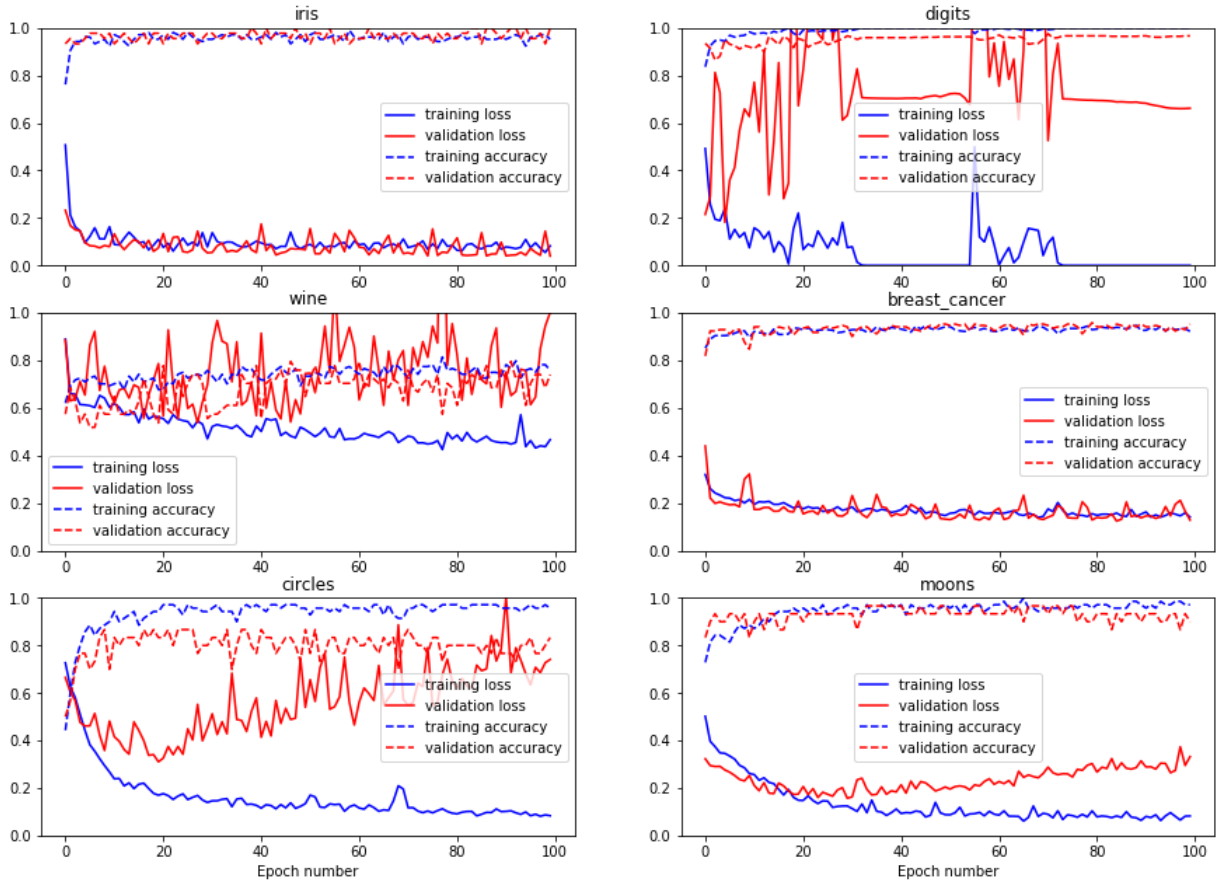
final training loss: 0.140357, final training accuracy: 0.922111, final validation loss: 0.127782, final validation accuracy: 0.953216

circles:

final training loss: 0.083070, final training accuracy: 0.957143, final validation loss: 0.741089, final validation accuracy: 0.833333

moons:

final training loss: 0.081898, final training accuracy: 0.971429, final validation loss: 0.331804, final validation accuracy: 0.900000



Setting 2 (optimizer=Adam, number of HL=2 , number of perceptrons in HL1=50, number of perceptrons in HL1=30, lr=0.01)

iris:

final training loss: 0.049387, final training accuracy: 0.971429, final validation loss: 0.215833, final validation accuracy: 0.911111

digits:

final training loss: 0.105684, final training accuracy: 0.990453, final validation loss: 1.798946, final validation accuracy: 0.948148

wine:

final training loss: 0.499332, final training accuracy: 0.741935, final validation loss: 0.696220, final validation accuracy: 0.611111

breast_cancer:

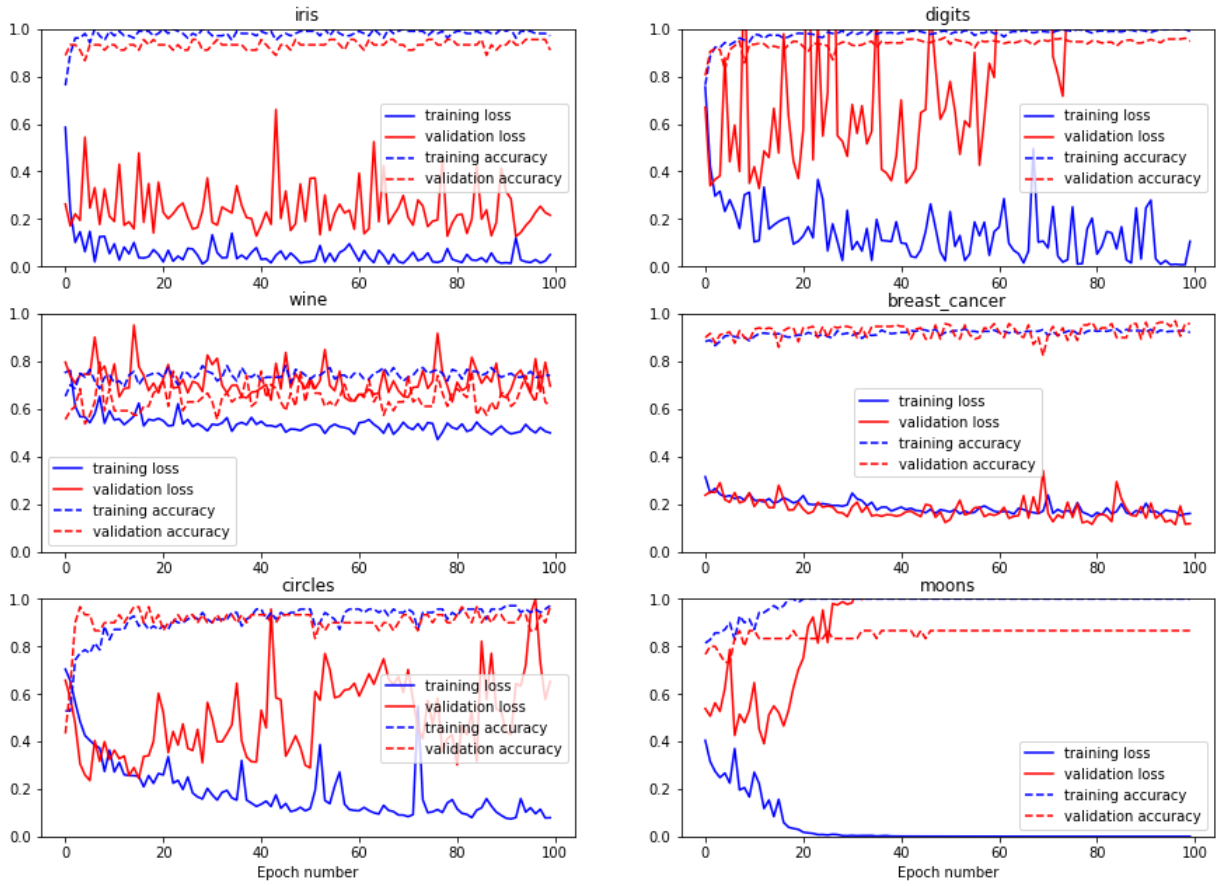
final training loss: 0.159909, final training accuracy: 0.922111, final validation loss: 0.117443, final validation accuracy: 0.959064

circles:

final training loss: 0.078894, final training accuracy: 0.971429, final validation loss: 0.652290, final validation accuracy: 0.966667

moons:

final training loss: 0.000039, final training accuracy: 1.000000, final validation loss: 1.559152, final validation accuracy: 0.866667



For a single hiden layer nn the accuracy on the datasets seems better than of the nn with two hidden layers. In case of the wine data set the accuracy drops from 0.740741 to 0.611111. This is due to the fact that these datasets contain small number of examples and are therefore not suited to train a deep neural network of two hidden layers since they require much more example and parameters to train them

In [ ]: