INF283 - Exercise - 5

(**Exercise 3 deadline: 6th of October, 23.59**).
Submission details: Computer written or scanned pages.
Deliver here: https://mitt.uib.no/courses/12791/assignments

Weekly exercises are a compulsory part of the course. You will need to complete at least half of them. Weekly exercises give a total of 16 points to the final grade, of the total 8 exercises each then gives 2 points to you final grade, as long as you upload your answer to MittUIB.no/assignments before 23.59 on Fridays. They will then be reviewed, and points are added to your total grade score (If we see you have made an effort. So no score loss if your answer had some error in the calculation etc). If you have completed only a fraction of the tasks then you will get a fraction of 2 points.

If you follow these exercises and ask for help when you need it during the group sessions, it will help you a lot, especially through the more difficult parts of the course.

In this exercise we will learn about neural networks and how to create them.

Understanding neural networks

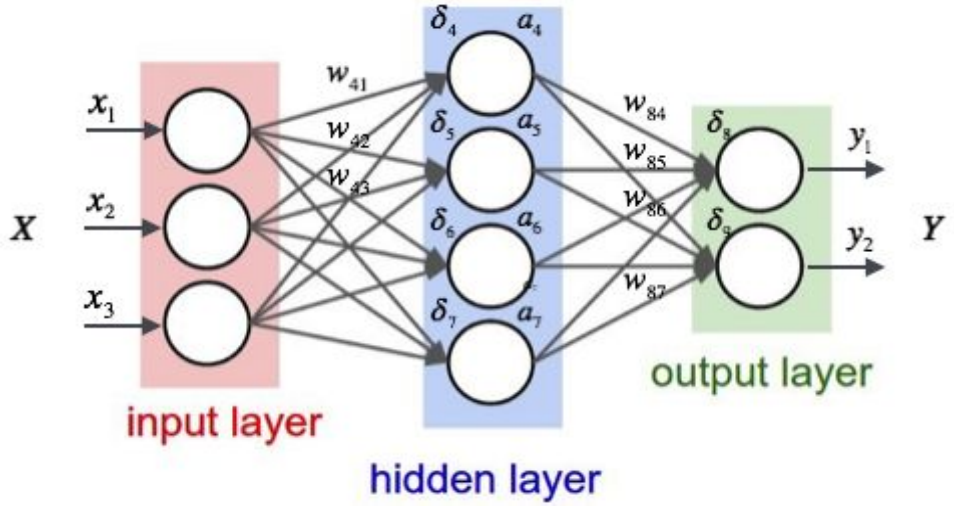

Figure 1. *Representation of a neural network with 1 hidden layer*

Artificial neural networks (**ANN**) or **connectionist systems** are computing systems vaguely inspired by the biological neural networks that constitute animal brains.[1] Such systems "learn" to perform tasks by considering examples, generally without being programmed with any task-specific rules. For example, in image recognition, they might learn to identify images that contain cats by analyzing example images that have been

manually labeled as "cat" or "no cat" and using the results to identify cats in other images. They do this without any prior knowledge about cats, e.g., that they have fur, tails, whiskers and cat-like faces.

1.1

From figure 1, shortly define these parts of a neural network:

a. X
b. Input layer
c. Weights w
d. Sum of weights function δ
e. Activation function α
f. Hidden layer
g. Output layer
h. Y

1.2

The weights are the part of the network that actually “learns”.
Explain why.

1.3

The weights are normally initialized to some random value (this makes neural networks “non-deterministic”). This is related to the fact that stochastic optimization algorithms such as stochastic gradient descent use randomness in selecting a starting point for the search and in the progression of the search (normally a value close to, but not 0).
Explain why randomization is used by giving an example where the network would not come to a good solution without it.

1.4

Given this simple neural network, lets say the weights are randomly initialized as the following, (In hidden layer, small number is input number, big number is sigmoid output), the output neuron has a sigmoid/logistic activation function. what will the output be before and after the activation function ?

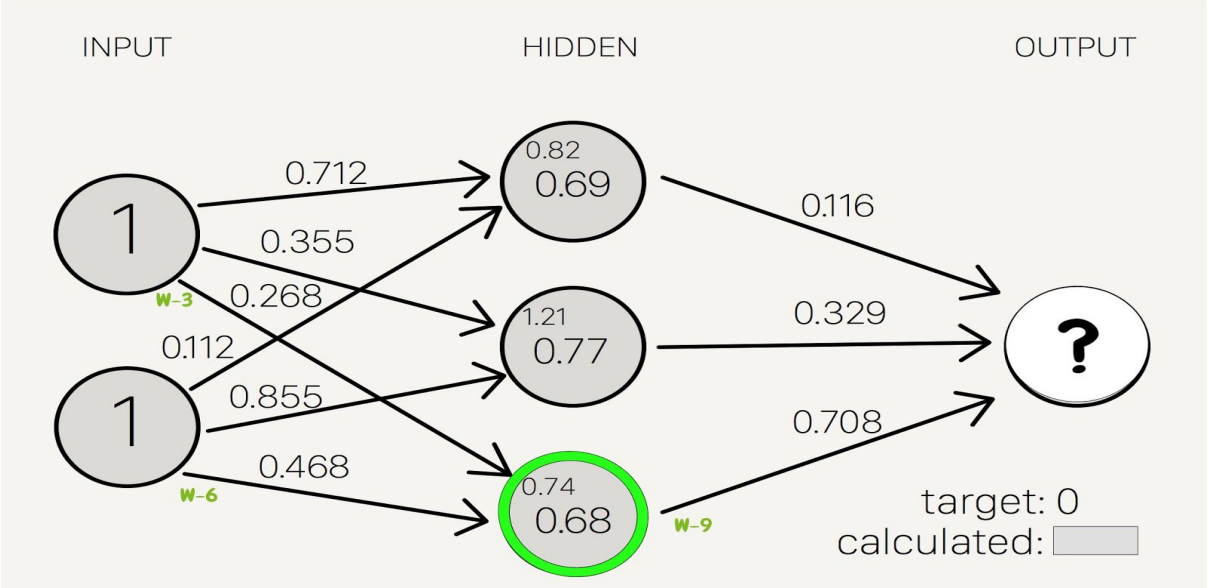

Figure 2. *A simple neural network*

1.5
In the neural network from task 1.4, the weights connected to the green neuron (i.e. w-3, w-6 and w-9,0) will be updated the most by gradient descent, because they affect the error the most. Given sigmoid/logistic as activation function and a learning rate (n) of 0.5 and a cost/error function of $E = ½(y - yi)\hat{}2$
What are the 3 partial derivatives from the backpropagation? Update the 3 weights (update 0.708, 0.468 and 0.268), not all!
Hints: Target is 0, see link for step by step procedure you should use:
https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/
Step1: Remember the partial derivatives are $\delta E / \delta Wi$, for weights 3, 6 and 9.
Step2: Update by: $Wi(next\ iteration) = Wi - n * (\delta E / \delta Wi)$
W9 should change more than W6, logically.

Simulation of neural networks

Now lets use Google's tensorflow playground to understand what each neuron represents.

2.1
Go to: https://playground.tensorflow.org
Press play and see how the network converges into a steady state quite quickly.
Look at the neurons in hidden layer 1, see how simple their shapes are compared to hidden layer 2.
Why ?
2.2
Change the learning rate to 3, what happens and why ?

2.3
With more complex shapes, like the spiral dataset (on the left), try running with learning rate 0.1. You will see that it will not converge to a good solution.

2.4 With the spiral dataset, Use 6 hidden layers, 8 neurons in each layer, activation function ReLu, and learning rate 0.03. Run until it converges (if not restart, and it should work). Explain why the additional neurons and hidden layers make it converge to a good solution, while fewer neurons makes it not converge to a good solution.

Using the keras package

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast

experimentation. They state on their site: *"Being able to go from idea to result with the least possible delay is key to doing good research".*

First install keras:
pip install Keras

Here is an example of a template, edit this code to make it work for your data:

Models in Keras are defined as a sequence of layers (the neural layers).

```
# Create Neural network classifier
from keras.models import Sequential
model = Sequential() # initialize
model.add(Dense(12, input_dim=8, activation='relu')) # 2 layers, input 8 and 12 hidden layer
model.add(Dense(8, activation='relu')) # input 8, activation relu
model.add(Dense(1, activation='sigmoid'))  # input 1, activation sigmoid
```

Then you need to compile model (for speed):

```
# Compile model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Fit the model (X is features, Y is output, this is for training data)
model.fit(X_train, Y_train, epochs=150, batch_size=10)
# epochs are the number of iterations wanted.
# batch size the number of iterations before update
# (150 epoches, 10 batch size = 15 #updates).
# score that should be at least 95%
scores = model.evaluate(X_test, Y_test)
print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

Now train a model using the iris dataset, creating a neural network that gives accuracy of at least 95 % on the testing data (30% of total). Justify the number of hidden layers and neurons in each layer.

Bonus task for curious people

This task in voluntary (no points for this task), and should be done if you really want to understand how neural networks work, and how you could program neural networks from scratch.

First task: View this tutorial (7 short episodes)
https://www.youtube.com/playlist?list=PLiaHhY2iBX9hdHaRr6b7XevZtgZRa1PoU

Second: Now choose what you want to investigate, either discuss some important mathematical aspect of neural networks based on the tutorial, or try to implement this neural network in python and choose your own data set to test it. A personal favorite is cats vs batman picture classification.