Backpropagation and Gradient Descent

Homework assignments are mandatory. In order to be granted with 8 ECTS and your grade, you must pass 9 out of 10 homeworks. Upload your solution as iPython notebook (*.ipynb*) and as HTML export until Saturday 4th November 23:59 into the public *Homework Submissions* folder on studip. Name your file as *<your_group_id >*_backpropagation-and-gradient-descent*<.ipynb/.html >*.

Further, you have to correct another group's homework until Monday 6th November 23:59. Please follow the instructions in the *rating guidelines* on how to do your rating.

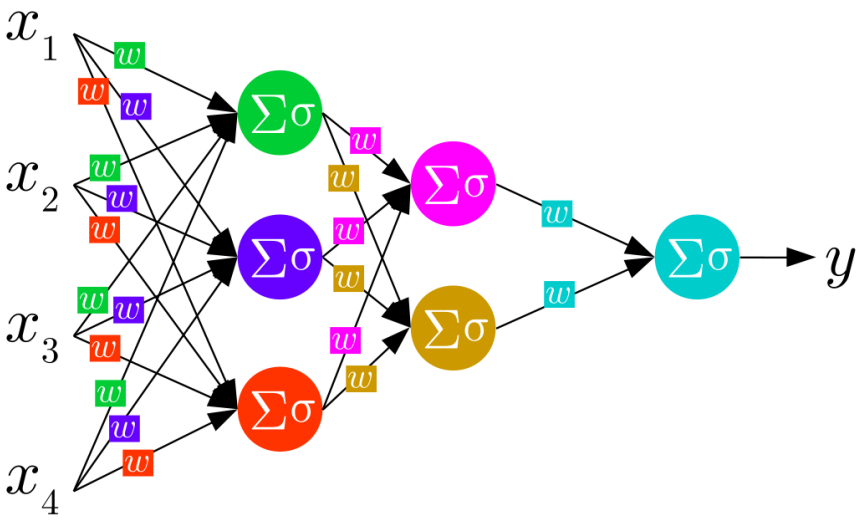
If you encounter problems, please do not hesitate to send your question or concern to lbraun@uos.de. Please do not forget to include [TF] in the subject line.

1 Videos

Watch the videos Artificial Neural Networks Demystified Part 3 and Artificial Neural Networks Demystified Part 4, which will repeat many of the aspects of backpropagation and gradient descent, that we have discussed in this week's lecture.

2 The backpropagation algorithm

Given the following four-layer feed-forward neural network



1. Write down the network-function (You can write mathematical formulas in ipython notebook with the help of latex)
2. Insert the network-function into the *Sum Squared Error Function* that we have introduced in the lecture
3. ***Optional*** Calculate the gradient of the error function with respect to the weight vector of the ouput layer ($\nabla_{w_{\text{out}}} loss$). It is sufficient to write σ' for the derivative of the activation function.
4. Calculate the derivative of the Logistic Function
   $\sigma(x) = \frac{1}{1+e^{-x}}$

3 Cats and dogs

In this assignment, we are going to implement a very simple network, with two input values, two weights and a single output neuron. Since the network has only two weights, we can actually plot the error surface and the training progress.

3.1 Setup

Please import numpy for numerical computations and matplotlib in order to be able to plott the error surface and the training progress of our netwok into your ipython notebook session.

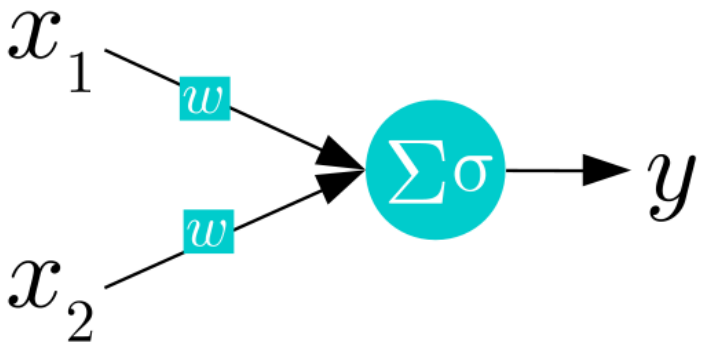
```
import numpy as np
import matplotlib.pyplot as plt
```

We can activate inline plotting with the following comand

```
%matplotlib notebook
```

3.2 The network

The structure of the network is really simple. We have tow input values and a single neuron, which projects the input to a single output value.



Hence, the network function is: $y = \sigma(\bar{x}^T \bar{w})$

3.3 Training data and task

We want to train the network, to distinguish between cats and dogs. The first input value of the network is the length and the second input value the height of the animal. Since I am afraid of dogs and cats are to fast for me, I was unable to gather real data. Hence, we are going to use artificial data to train our network.

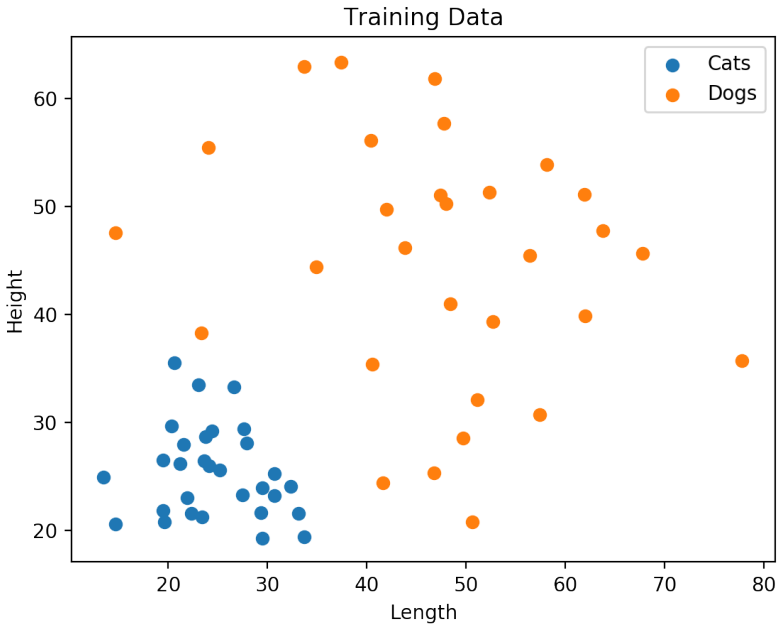
Generate a cat and a dog sample from a 2-D normal distribution. Cats are usually smaller then dogs, in addition, there is less variance in the size of cats than in the size of dogs. Thus, we sample the cat data from a normal distribution with a *mean* of 25 and a *variance* of 5 and the dog data from a normal distribution with *mean* 45 and a *variance* of 15.

By seeding the random number generator of numpy, we are guaranteed to always get the same data, even if we execute the generative parts twice or on two different days:

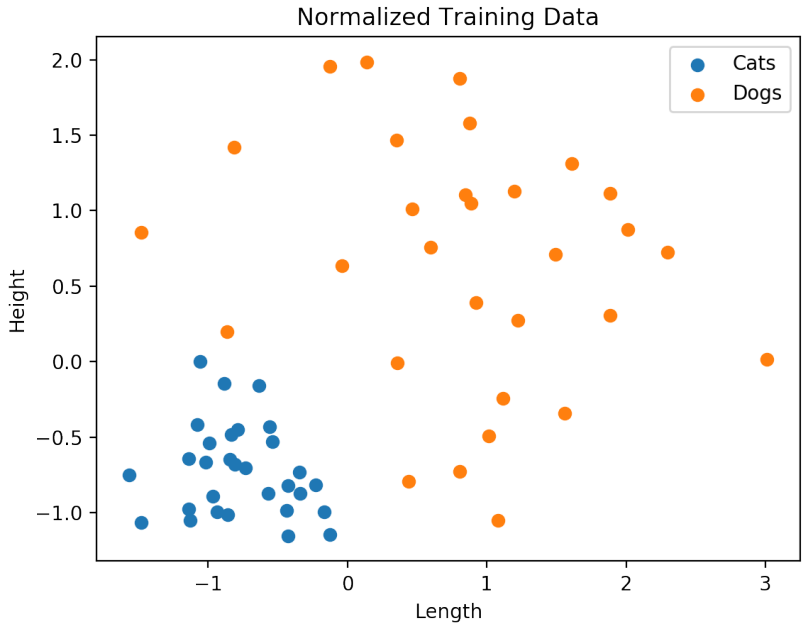
```
sample_size = 30

np.random.seed(1)
cats = np.random.normal(25, 5, (2, sample_size))
dogs = np.random.normal(45, 15, (2, sample_size))
```

3.4 Investigating the data

It is always a good idea to investigate the training data before you start working with it. Create a plot that shows all data points.



As you can see, the data values are rather large. Most of our activation functions are most sensitive for values around 0. Normalize your data by subtracting the mean and dividing by the standard-deviation in order to distribute the data around zero.



3.5 Activation function and target values

Select an appropriate activation function for the output neuron and generate appropriate target values $\hat{y}$ for the training data.

3.6 Forwardpropagation

Implement a function that is capable of forward propagating a (mini-)batch of inputs through the network and is returning the output of the network function y for each sample in the batch. This can be achieved by aranging several samples $\bar{x_i} = [legnth_i, height_i]^T$ into a matrix. Initialize your weights with $[-2.5, -2.5]$.

3.7 Loss

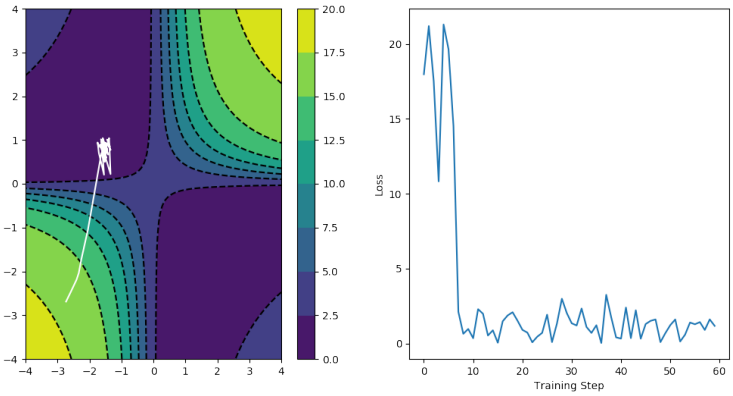
Implement a function, that calculates the *Sum Squared Error* between the network's output and the respective target values $\hat{y}$.

3.8 Backpropagation

Implement a function, that calculates the gradient of the loss with respect to the weight vector.

3.9 Gradient Descent

Implement a loop that feeds data through your network, calculates the loss and the gradient. Update the weight values with the gradient descent parameter update rule in each step. Create two plots, a surface plot, which contains the actual error surface (error for "all" possible combinations of weight values) and the past and updated weight values as a line and a second plot, which shows the loss over time.



Test your loop with batch, mini-batch and stochastic gradient descent and different learning rates (η).

4 Further resources

If you have some free time, read Yann LeCun's paper "Efficient BackProp" and have a look at Sebastian Ruder's online article about different optimizations of vanilla Gradient Descent.

5 Optional: Momentum

Try to accelerate the convergence by adding momentum to your gradient descent (explained twice, once in the paper by LeCun and also in the online article by Sebastian Ruder).