Tutorial - Exercise sheet 9

Pattern and Speech Recognition

Introduction

In this exercise you will implement a small feed-forward neural network for two-class classification. The NN will compute the function:

$$y(x, w) = \sigma(\sum_j^M w_j^{(2)} \sigma(\sum_{i=1}^D w_{j,i}^{(1)} x_i)) = \sigma(w^{(2)T} \sigma(w^{(1)T} x))$$

Where D is the number of dimensions of the training samples and M is the number of neurons in the hidden layer.

Notation:

- $a_j^{(1)} = \sum_{i=1}^D w_{j,i}^{(1)}$

  is the weighted sum of inputs in neuron j in layer 1, which is the hidden layer.

- $z_j^{(1)} = \sigma(a^{(1)}{}_j)$

- $a_k^{(2)} = \sum_{j=1}^M w_{k,j}^{(2)} z_j^{(1)}$

  is the weighted sum of inputs in layer two, which is the output layer. As it is a two class problem, we only need one output neuron so K=1, we can drop the lower script.

- $y(x, w) = z^{(2)} = \sigma(a^{(2)})$

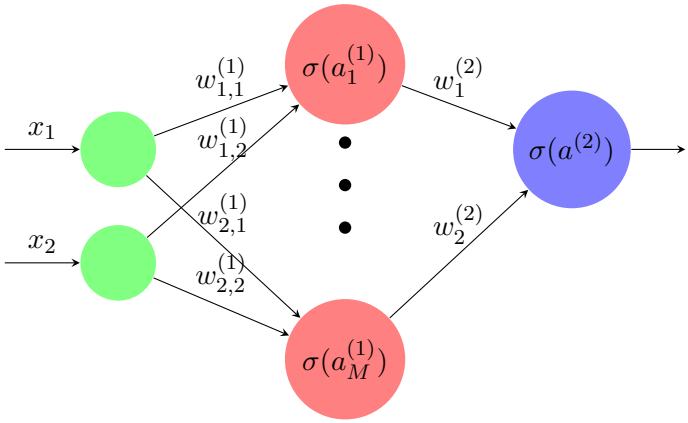
This shows that we are not using bias parameters.
The activation function is the logistic sigmoid: $\sigma(x) = \frac{1}{1+e^{-x}}$. The derivative of the sigmoid is: $\sigma(x) * (1 - \sigma(x))$
The parameters of your network are $w_{ML} = \{w_{(1)}, w_{(2)}\}$. For training the network you will use stochastic gradient descent. This method approximates the gradient of network by the gradient at a single example.
The error function for a single datapoint n is given by:

$$E_n(w) = \frac{1}{2}(y(x_n, w) - t_n)^2$$

$y(x, w)$ is the output of the sigmoid, so it is between 0 and 1. For each predicted value you can compute how far it is from its class label: t_n.
For back-propagation you will need the following derivatives:

$$\frac{\partial E_n}{\partial w_j^{(2)}} = \frac{\partial E_n}{\partial y_n}\frac{\partial y_n}{\partial a^{(2)}}\frac{\partial a^{(2)}}{\partial w_j^{(2)}}$$

$$\frac{\partial E_n}{\partial w_{j,i}^{(1)}} = \frac{\partial E_n}{\partial y_n}\frac{\partial y_n}{\partial a^{(2)}}\frac{\partial a^{(2)}}{\partial z_j^{(1)}}\frac{\partial z_j^{(1)}}{\partial a_j^{(1)}}\frac{\partial a_j^{(1)}}{\partial w_{j,i}^{(1)}}$$



Data

- Download the synthetic data set from https://www.dropbox.com/s/5cweo5ooc4tg4yt/dataset-train.csv?dl=0. The data is in csv format. The first two columns are the two features, the third column shows the class label. You will use your whole data for training. Do not forget to shuffle the input.

Implementation of the NN (6 points)

- Implement a function $predict(w_{ML}, data)$, which computes the prediction of the neural network for the data. It should return a vector of the length of the training samples.
- Write a function $trainNetwork(data)$, that returns your trained parameter set. First you will implement a network that has only one hidden layer with one hidden neuron. This means that your weight vector $w^{(1)}$ is 2 by 1 and $w^{(2)}$ is 1 by 1.
- Initialize your parameters sampling from a uniform random distribution between 0-1.
- You will traverse through your dataset sample by sample. For each point compute one forward pass of the NN then update the parameters using the backpropagation algorithm.
- One full pass through the training set is an epoch. After each epoch report the error of the whole traing set:
  $\frac{1}{2}\sum_{n=1}^{N}(y(x_n, w_{ML}) - t_n)^2$
  For this you can use the function $predict(w_{ML}, data)$. Plot the error after each epoch and

reshuffle your data .
Run 800 epochs.

- Use the plot of the error to tune your learning rate. Try differnet powers of 10 $\eta \in \{...10^{-1}, 10^{1}...\}$. What does the plot look like if the learning rate is too high or too low?

Plotting (2points)

- You can interpret the output of the network as the conditional probability: $p(class_1|x)$. This means you predict $class_1$ if $y(x_n, w) > 0.5$, otherwise the other class.
- Plot the data and the decision boundary. For this you can use the meshgrid function implemented both for python and matlab. Classify each point in your meshgrid using your NN, and plot the contour.

Adding hidden neurons (2 points)

- As you can probably see, you will need a more complex decision boudary than what one hidden neuron can achieve. One by one add hidden neurons to your hidden layer (up until 6). For this you only have to change the size of your weight vectors. Plot the decision boundary for each case.

Submission instructions

The following instructions are mandatory. If you are not following them, tutors can decide to not correct your exercise.

Submission architecture

You have to generate a **single ZIP file** respecting the following architecture:

```
tutorial1_<matriculation_nb1>_<matriculation_nb2>_<matriculation_nb3>
|
+--- source
|      |
|      +------- file 1
|      +------- file 2
|      +------- ...
+--- rapport.pdf
+--- README.txt
```

where

- **source** contains the source code of your project,
- **rapport.pdf** is the report where you present your solution with **the explanations (!)** and the plots,
- **README** which contains group member informations (name, matriculation numbers and emails) and a **clear** explanation about how to compile and run your source code

The ZIP filename has to be :

```
tutorial1_<matriculation_nb1>_<matriculation_nb2>_<matriculation_nb3>.zip
```

You have to choose between the following languages **python** or **matlab**. Other languages won't be accepted.

Some hints

We advice you to follow the following guidelines in order to avoid problems :

- Avoid building complex systems. The exercises are simple enough.
- Do not include any executables in your submission, as this will cause the e-mail server to reject it.

Grading

Send your assignment to the tutor who is responsible of your group:

- Gerrit Großmann gerritgr@gmail.com
- Sébastien Le Maguer slemaguer@coli.uni-saarland.de
- Kata Naszádi b.naszadi@gmail.com

The email subject should start with `[PSR TUTORIAL 8]`