INF3490/INF4490 Mandatory Assignment 2: Multilayer Perceptron

October 1, 2018

Rules

This is an individual assignment. You are not allowed to deliver together or copy/share source-code or answers with others. Before you begin the exercise, review the rules at this website:

`http://www.uio.no/english/studies/admin/compulsory-activities/mn-ifi-mandatory.html`

The assignment can be changed/updated after release (to clarify or correct mistakes). It is advised to always use the latest version, available **here**.

Delivering

Deadline: Tuesday, October 16th, 2018 23:59:00. All deliveries must be made through **Devilry**.

What to deliver?

Deliver one single zipped folder **(.zip, .tgz or .tar.gz)** which includes:

- PDF report containing:
  - Your name and username (!)
  - Instructions on how to run your program.
  - Answers to all questions from assignment.
  - *Brief* explanation of what you've done.
- Source code
- Data file (so the program can run *right away*)
- Any files needed for the group teacher to easily run your program on IFI linux machines.

Important: if you weren't able to finish the assignment, use the PDF report to elaborate on what you've tried and what problems you encountered. Students who have made an effort and attempted all parts of the assignment will get a second chance even if they fail initially. This exercise will be graded PASS/FAIL.

Preface

The purpose of this assignment is to give you some experience applying supervised learning with a multilayer perceptron (MLP). In order to control a robotic prosthetic hand, Prosthetic hand controllers (PHCs) reads the electromyographic signals generated by contracting muscles in the under arm. The idea is that we can, through supervised learning, get a PHC to learn which hand movement the user of a robotic prosthetic hand wants to perform.

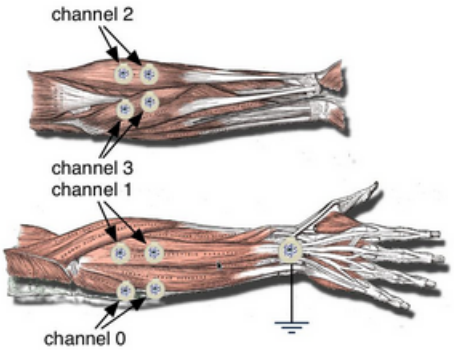

Fig. 1. Sensor placement (muscle anatomy taken from [21]).

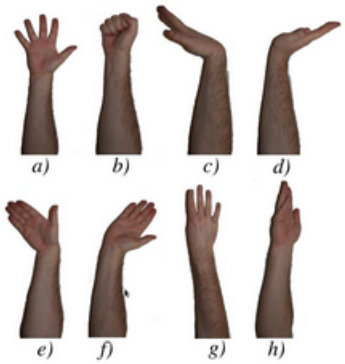

Fig. 2. Motion classes: *a)* open, *b)* close, *c)* flexion, *d)* extension, *e)* ulnar deviation, *f)* radial deviation, *g)* pronation and *h)* supination.

The data

The data you will be classifying are electromyographic (EMG) signals corresponding to various hand motions, collected over three days (one dataset per day).[1] Each row of the data has 40 features (4 sensors, 10 readings each) and one classification value (1-8, corresponding to the 8 hand motions.[2]). You can get the **data**(.zip or .tgz) from the **semester page**.

Getting started

You are free to use a programming language of your own choice, but we strongly recommend using Python. The precode is only available in Python (the files *movements.py* and *mlp.py* that are bundled with the data). If you choose another language, you will have to implement everything yourself. Along with the two precode files, you are also given four files with data:

- *data/movements_ day1.dat*
- *data/movements_ day2.dat*
- *data/movements_ day3.dat*
- *data/movements_ day1-3.dat*

As the last file contains data for all days, you don't *have to* use the other files. You can use the *movements_ day1-3.dat* file and split up the data into different sets.

The file *movements.py* will read one specified data file, and split it into training, validation and test sets. Then it will create an MLP, run the training on the MLP and call the confusion method of the MLP to print the confusion matrix.
A class mlp with four methods is given in the file *mlp.py*:

- *earlystopping*
- *train*
- *forward*
- *confusion*

[1] See section 2 of K. Glette, J. Torresen, Thiemo Gruber, Bernhard Sick, Paul Kaufmann, and Marco Platzner. *Comparing Evolvable Hardware to Conventional Classifiers for Electromyographic Prosthetic Hand Control.* (In Proceedings of the 2008 NASA /ESA Conference on Adaptive Hardware and Systems(AHS-2008), Noordwijk, The Netherlands, IEEE Computer Society, 2008, or `http://folk.uio.no/kyrrehg/pf/papers/glette-ahs08.pdf`) if you are curious about how the data was collected.

[2] Which, for the curious, are: open, close, flexion, extension, ulnar deviation, radial deviation, pronation, and supination

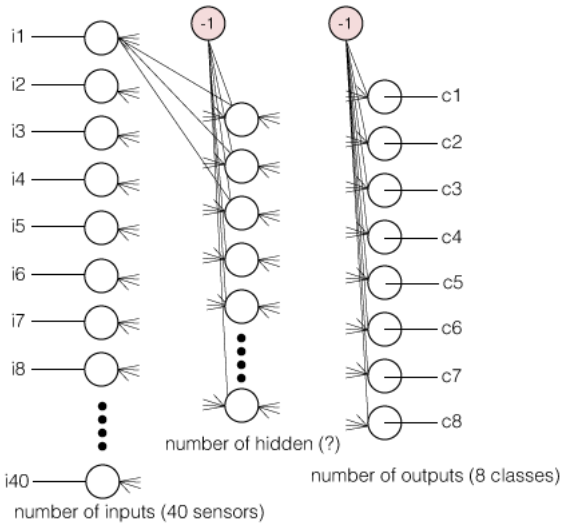

Figure 1: The neural net with one hidden layer

earlystopping is the method that starts the training of the network and keeps track of when to stop the training. It should, in each iteration, call MLPs train method and run the network forward. Before the network learns the trainingdata too well, we stop training to avoid overfitting. See Fig. 4.11 on page 88, S. Marsland.

train trains the network. This means running the backwards phase of the training algorithm to adjust the weights. See section 4.2.1 on page 77-79, S. Marsland.

forward run the network forward. When running the network forward, a perfect classifier should always have the highest value on the correct output. You will experience that this is not the case the first time(s) the network is run forward, which is why you are adjusting the weights in train.

confusion prints a confusion matrix. Run the network forward with the test set, and fill out the outputs in a matrix. The matrix should be a $c \times c$ matrix (where c is the number of classes), and have the correct classes on one axis and classified classes on the other axis. Print also the percentage of correct classes.

How the algorithm runs

After the input is given to the MLP, we start to train the network. Instead of running the training a fixed number of iterations, we split it up into parts, where we decide whether to stop in between the iterations. This means that we will check the error of the network on a validation set in *earlystopping*, and if our stopping criteria is met, we will stop training.

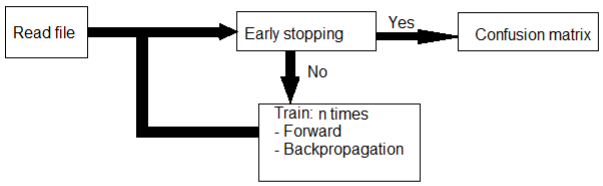

Figure 2: A diagram of how the algorithm runs

The training phase consists of the first running the network forwards, then running it backwards, adjust the

weights and repeat for as many iterations as we have chosen (10,100,1000). When training is done we return to *earlystopping* to check if we're done.

Once we're done we print out the confusion table based on how the network classified the data in the test set.

How we will implement the network

In this assignment we will be implementing the neural network in an iterative manner. Here we will use for-loops to compute the forward and backward passes in the network. This means that we loop iteratively over nodes and weights, to compute the weighted sum for a given node/neuron in the forward pass, and iterate again to compute deltas and update weights in the backward pass.

Recall the definition mentioned in the lectures, on how we compute the weighted sum h for a given node/neuron, that has its corresponding set with pairs of weights and input nodes (set of input nodes being the previous layer), w and $x \in M$:

$$h = \sum_{i=1}^{M} x_i w_i$$

.

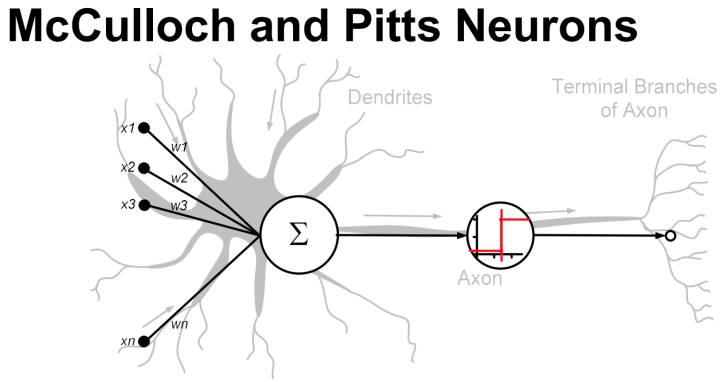

Figure 3: Weighted sum for a node

With the above definition for one neuron, we can extend this operation to N neurons in a given layer. Also see p. 78 in Marsland for the rest of the equations that are used in both forward and backward propagation in a network.

What to do

All students

- Implement the iterative version of the neural net as explained above, with for-loops. Here, you are not allowed to use matrix operations, such as those found in Numpy. Don't use more than one hidden layer (see Fig. 1).
- **[Optional]** Implement a matrix-version of the neural net. Here, you are allowed to use matrix operations, such as those found in Numpy.
- Run the algorithm on dataset *movements_ day1-3.dat*. You should test with at least three different number of hidden nodes (e.g 6, 8, 12). Report your findings and provide the resulting confusion tables with percentages, one for each net with different number of hidden nodes. How many nodes do you find sufficient for the network to classify well?
- By only looking at your reported confusion tables, which classes where likely to be mistaken for each other?
- **[Optional]** For those that also implemented the matrix-version, report a benchmark/comparison of your iterative solution vs the matrix solution, where you compare the running times of the training of the

network. How is the training time affected, when we increase the number of hidden nodes? How is the difference in efficiency?

INF4490

- Implement k-fold cross-validation. Choose a suitable k. Report the correct percentages for each fold, along with the average and standard deviation. Please do not include all confusion tables in your report.

Frequently Asked Questions(FAQ)

What should my datastructure look like?

You are free to implement a datastructure of your own choice. In this mandatory assignment you are implementing a neural net with one hidden layer. That means you will have two weight layers with floats (one between input and hidden, and one between hidden and output). These two weight layers together constitute the neural network and it's basically the only datastructure you need.

However, at a certain point during backpropagation, you need a reference to the activation levels of each neuron too. You can either store these in their own datastructure, and make sure each neuron is correctly associated with its corresponding weight, or you can have a reference to the activations by returning them in your forward propagation function

For the iterative version, we recommend that you simply use 2D lists. For those who also implement the matrix version, we recommend using numpy 2D arrays/numpy matrices. Keep in mind that you need some additional structures to handle inputs, outputs, etc.

I think I am done, but my neural net performs poorly (less than 60% accuracy), why?

Make sure you're not using a sigmoid activation function on the output. If you used Eq. 4.7 from the book, replace it with a linear output (Eq. 4.13). When replacing this output function, you also need to change the error function accordingly; use eq. 4.14 instead of eq. 4.8.

What is one iteration in training? Is it one run with one input vector or is it running the whole training set through the net?

The training data is an array of n input vectors, and there are multiple ways to feed this through and update the network. You are free to define an iteration how you want, but make sure you train on all data. Here are three possible methods:

1. Feed the network with the whole training set. Then update the weights based on the n output errors. This is one iteration.

2. Feed the network with one input vector from the training set. Update the weights. Repeat until you've been through the training set. When done with the whole training set, you have one iteration. This method differs from 1. in the way that weights will be updated after every input from the training set. Now the order matters because training vector n will be run on a network trained on all the other vectors in the same iteration (and vector 1 will always be first in each iteration). In order to avoid this problem, the order of the training data should be randomized before each iteration.

3. Pick a random vector from the training set and feed it into the net, update the weights. Let's call this one iteration, but keep in mind that you need n times more iterations in order to get the same amount of training as in 1. and 2. This method is similar to 2., but instead of running one iteration over the whole set, we randomly pick one by one vector. This means that some vectors may never be chosen and some vectors might be used multiple times.

What libraries am I allowed to use?

You are not allowed to use machine learning libraries, as you have to *implement* the algorithm yourself as explained in this assignment. If you're implementing the matrix version (which is optional), you are allowed to use matrix operations, such as those found in Numpy. However, for the iterative version (mandatory for all) it is not allowed to use Numpy matrix operations, only for-loops. General purpose/scientific libraries for calculations, plotting etc. are allowed. Examples of this can be **pyplot** from the matplotlib library

Precode

The assignment folder includes some precode that you're encouraged to use. Modify `mlp.py` and run `movements.py`:

```
> python3 movements.py
```

Contact

The exercises are the same as those in the **GitHub repository**. If there are any suggestions to corrections of grammar, language or any additional suggestions, we appreciate all feedback! The TA's can be reached at **lonnekes@ifi.uio.no**, **hermankn@ifi.uio.no**, **a.s.skage@econ.uio.no** or **sharanak@ifi.uio.no**