NEUR503 – Introduction to Computational Neuroscience

Assignment on Credit Assignment

Blake Richards

Winter 2024

In this assignment, you will train a multi-layer perceptron (MLP) with four different learning algorithms that solve the credit assignment problem in different ways: backpropagation-of-error (backprop), weight perturbation, feedback alignment, and the Kolen-Pollack algorithm. The last three avoid weight transport in credit assignment, unlike backprop.

The purpose of this assignment is three-fold:

1) To get you familiar with how neural networks really work, so that you understand the inner working of the sort of models we build with help from machine learning libraries nowadays.

2) To force you to think about the actual learning algorithm for a neural network and how to implement that in code in an elegant manner.

3) To have you engage and think more concretely about how biological constraints can impact learning performance and what this tells us about learning in the brain.

The assignment is out of 50 marks, and there are 5 bonus marks available (see the marking scheme below).

Part 0 – Getting started (no marks, though if you don't do this you won't get any marks!)

- To do this assignment you need to have Python installed, as well as numpy, and a plotting library (e.g. matplotlib), and Jupyter notebook. One way to kill multiple birds with one stone here is to install Anaconda or Miniconda: https://docs.conda.io/en/latest/
- Make sure you can download the template code provided and open the Jupyter notebook (`credit_assign_blank.ipynb`), which gives you the skeleton for your code. **The only bits of code you will need to change in this notebook are the weight update functions**, `perturb`, `backprop`, `feedback`, and `kolepoll` – look for the comment (`CHANGE THESE LINES`). To open the notebook you use Jupyter, e.g., on a Linux system if you have Jupyter installed you can type this into the command prompt:
  - `$ jupyter notebook credit_assign_blank.ipynb`
  - If everything is installed correctly, then you should see a series of loss and accuracy curves plotted that just stay flat (since the weights are not being updated).
- You must put your code into this notebook, and submit it with a new name, along with a Word doc with your answers to Part 5. The name of the notebook must adhere to:
  - `credit_assign_[firstname]_[lastname]_[mcgillid].ipynb`
  - Where the items in brackets must be replaced with your specific info.

Part 1 – Implementing weight perturbation (10 marks total)

First, you must write the code for training your MLP with weight perturbation (see the lecture notes). The MLP is a three-layer feed-forward MLP (one input layer, one hidden layer, and one output layer), and the number of units in the hidden layer can be modified. You will be graded on both whether your algorithm learns (i.e. loss goes down), and the elegance of the solution you code up. Note that this second question is semi-subjective, and will be decided on by the grader.

Part 2 – Implementing backprop (10 marks in total + 3 bonus marks)

Now, build the code to update the weights using backprop. Again, you will be graded on whether the loss goes down and the elegance of your solution. In line with this, you can get three bonus marks if your solution does not involve any for-loops.

Part 3 – Implementing feedback alignment (10 marks in total)

Now, build the code to update the weights using feedback alignment. Again, you will be graded on whether the loss goes down and the elegance of your solution.

Part 4 – Implementing Kolen-Pollack (10 marks in total + 2 bonus marks)

Now, build the code to update the weights using the Kolen-Pollack algorithm. Again, you will be graded on whether the loss goes down and the elegance of your solution. There are also two bonus marks if you can do the updates for the forward and backwards weights using the same error-propagation term (i.e. without calculating the first term of the Kolen-Pollack updates more than once).

Part 5 – Observations (10 marks in total)

Answer the five following questions, two marks for each:

- Which of these algorithms provides the best performance?
- Which provides the worst?
- How does their performance scale with the size of the network?
- How does their performance change with different batch sizes?
- Given your answers to the questions above, and biological constraints that you can think of, which of these algorithms are most likely to exist in the brain, in your opinion? Provide your reasoning. There is no "right" answer here. You will be marked on how cogent your reasoning is.