CS479/679: Neural Networks
Assignment 1
Due: 11:59 AM (EST), Jan 30, 2023, submit on LEARN.
Include your name and student number!

Submit your writeup in pdf and all source code in a zip file (with proper documentation). Write a script for each programming exercise so that the TAs can easily run and verify your results. Make sure your code runs!

[Text in square brackets are hints that can be ignored.]

Question 1: LIF Firing Rate (25 pts) Recall that the sub-threshold membrane potential for a LIF neuron is governed by the DE,

$$\tau \frac{dv}{dt} = v_{\text{in}} - v \,. \tag{1}$$

Show that if the threshold v_{th} is 1, and if v_{in} is held constant, then the firing rate of a LIF neuron can be computed using

$$G(v_{\text{in}}) = \begin{cases} \frac{1}{\tau_{\text{ref}} - \tau \ln\left(1 - \frac{1}{v_{\text{in}}}\right)} & \text{for } v_{\text{in}} > 1 \\ 0 & \text{otherwise} \end{cases}$$

Hint: The time between spikes (t_{isi}, the "inter-spike interval") is the reciprocal of the firing rate, and is also the sum of the refractory time and the time it takes for v to climb from 0 to the threshold of 1.

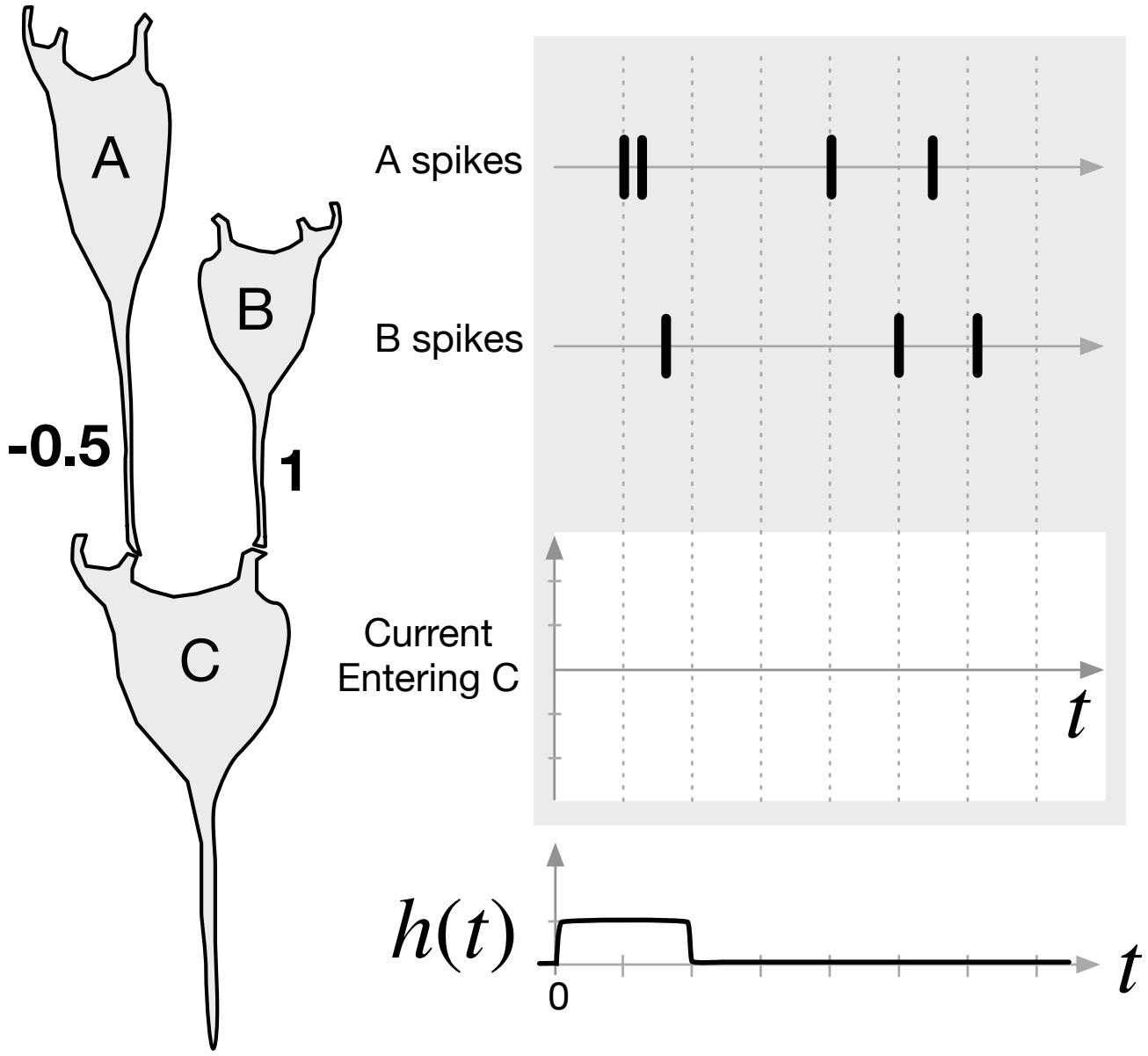
What to submit

For this question, you can:

- typeset your solutions using a word-processing application, such as Microsoft Word, LaTeX, Google docs, etc., or
- write your solutions using a tablet computer, or
- write your solutions on paper and take photographs.

In any case, it is **your responsibility** to make sure that your solutions are sufficiently legible.

Question 2: Input Current to the PS Neuron (25 pts) The diagram below shows three neurons on the left: two neurons, A and B, that synapse onto a neuron C with connection weights **-0.5** and **1**, respectively. The diagram on the right shows spike trains for A and B. Given the post-synaptic filter, $h(t)$, plotted below, **draw the net input current entering neuron C in the white box**.



What to submit

The drawing of your input current and how you got to that solution.

Question 3: Error Backpropagation and Loss Functions (50 pts)

Getting Started: Download the notebook `a02q23_YOU.ipynb` and the module `utils.py`. The notebook imports and uses the `utils` module, and also has a number of class definitions that you will use and modify.

Important: All `import` commands should appear in the same code cell (at the top of the notebook). Also, do not include any other lines of code in the code cells that contain function definitions or class definitions.

Part 1: Activation and Loss Functions

[20 pts] The notebook has a definition for a classed called `Operation`; it is an abstract base class from which activation functions and loss functions are derived. There are other classes derived from `Operation`, including the activation functions `Identity` and `Softmax`, and loss functions `MSE` and `CategoricalCE`.

To use these classes, you first create an instance of the class, and then use that instance as the function. For example, suppose you have input currents stored in `z`. Then, you can use the `Identity` class like this:

```
act = Identity()          # Creates function
h = act(z)                # Calls function
dhdz = act.derivative()   # Gets derivative
```

(a) Complete the `Logistic` class, which is an implementation of the logistic activation function. You should complete its `__call__` function, as well as its `derivative` function.

(b) Complete the `CrossEntropy` class, which is an implementation of the cross-entropy loss function. You should complete its `__call__` function, as well as its `derivative` function.

Part 2: Implementing Backpropagation

[30 pts] In this question, you will complete the implementation of learning by error backpropagation. In doing so, you will complete the functions `backprop` and `learn` in the `Network` class. If you've done it correctly, you can use your code to learn to classify the `UClasses` dataset in the notebook. And that will make you feel happy inside.

Here are the specific tasks:

(a) `backprop`: **Complete the** `Network.backprop` **function**, which performs an update to the network weights and biases using the error backpropagation algorithm. The method uses the **current (saved) network state**, including the activities of the output layer, and compares them to the targets.

You should assume that all layers after the input layer are of the `DenseLayer` class.

You should use the `derivative` methods of the loss function and activation functions.

(b) `learn`: **Complete the** `Network.learn` **function**, which tries to find the optimal network weights and biases. This function should call `backprop` to update the weights and biases.

There is some code at the end of the notebook that creates a network, adds some layers, and then calls the `learn` function to train it on the dataset. Your implementation should be able to achieve over 98% accuracy in about 5000 epochs.

What to submit

Your jupyter notebook, obviously.

Make sure you submit your solutions, and not just a copy of the supplied skeleton code. We suggest you rename the `ipynb` file by replacing "`YOU`" with your WatIAM ID (eg. `a02q23_jorchard.ipynb`). You should **not** submit `utils.py`.