APPM 4370/5370, Computational Neuroscience: Homework 6

Due Wed. Apr. 8, 2020
by 11am (by email)

PAGES 1&2: MATH EXERCISES; PAGE 2: NO COLLABORATION PROBLEM; PAGE 3: PYTHON EXERCISES.

Math exercises.

1. **Logical operations in multilayer perceptrons.** The perceptron is a feedforward (FF) neural network model popularized by Rosenblatt (1957) and inspired by work of McCulloch & Pitts (1946). Minsky & Papert (1969) proved several shortcomings of *single layer* perceptrons, including the fact that they cannot implement the XOR logical function ($x_1 \oplus x_2 = 1$ if $x_1 = 1$ and $x_2 = 0$ or if $x_1 = 0$ and $x_2 = 1$ but not if both $x_1 = x_2 = 1$). This led to an AI winter in which computer scientists and statisticians largely abandoned the study of neural networks (however, mathematical modeling in neuroscience using neural networks still flourished in the 1970s). This winter did not thaw until the early 1980s when the notion of using backpropagation to train *multilayer* perceptrons became widely know, and people realized mulitlayer perceptrons could be used to approximate any function (universal approximation theorem: Cybenko, 1989).

   Here you will explore a bit about this distinction between single and multilayer networks in a small networks and examine the mechanics of backpropagation by hand.

   (a) Show that the single neuron perceptron $y = H(w_1x_1 + w_2x_2 + \theta)$ cannot implement the XOR function for x_1 and x_2 no matter how the parameters w_1, w_2, and θ are tuned, where $H(x)$ is the Heaviside function. (This can be proved for any increasing transfer function $f(x)$, but you need not show this).

   (b) Design a two layer perceptron network that implements the XOR function. Restrict yourself to two neurons in the *hidden* layer y_1 and y_2 and one neuron in the *output* layer z:

$$\begin{aligned} y_1 &= H(w_{11}x_1 + w_{12}x_2 + \theta_1), \\ y_2 &= H(w_{21}x_1 + w_{22}x_2 + \theta_2), \\ z &= H(J_1y_1 + J_2y_2 + \eta). \end{aligned}$$

   There are multiple solutions to the problem. I suggest you sit down with pen and paper and start with a guess, and then refine your guess iteratively.

   (c) Lastly, how to train the two layer neural network? This does not work as well with Heaviside transfer functions since they are insensitive to change unless inputs are near threshold. We thus turn to the sigmoid $f(x) = 1/(1 + \mathrm{e}^{-x})$ and consider again the network:

$$y_1 = f(w_{11}x_1 + w_{12}x_2 + \theta_1), \tag{1a}$$
$$y_2 = f(w_{21}x_1 + w_{22}x_2 + \theta_2), \tag{1b}$$
$$z = f(J_1y_1 + J_2y_2 + \eta). \tag{1c}$$

   Work out a single step of backpropagation (of all the weights) by hand in the case initially taking the weights $w_{11} = w_{12} = w_{21} = w_{22} = J_1 = J_2 = 1$. Take the case in which $x_1 = x_2 = 1$ and the desired output is $z = 0.05$, so the error is $E = (z - 0.05)^2$. Take a learning rate of $r = 0.5$. Note, only update the weights w_{jk} and J_k not the biases θ_j and η. Remember weight updates take the form

$$w_{jk} \mapsto w_{jk} - r \cdot \frac{\partial E}{\partial w_{jk}}, \qquad J_k \mapsto J_k - r \cdot \frac{\partial E}{\partial J_k},$$

   where the partial derivatives with respect to the current weights are found using the chain rule. Take biases to be $\theta_1 = -1$, $\theta_2 = -1$, and $\eta = -1$. Compare the output z before and after the weight update. Is the error reduced after this step? Could you speed up the learning process by changing the learning rate r?

2. **Motivating the sigmoidal transfer function.** Here you will show how sigmoidal functions emerge naturally in statistical classification tasks.

(a) Consider the task of recognizing which of two multivariate normal distributions a length 2 random vector $\mathbf{z} = (z_1, z_2)$ comes from. Both have the same mean ((0,0)) but different variances, so the probability of $\mathbf{z}$ given $s \in \{0, 1\}$ is

$$P(\mathbf{z}|s) = \frac{\mathrm{e}^{-z_1^2/2\sigma_{s,1}^2}}{\sqrt{2\pi}\sigma_{s,1}} \cdot \frac{\mathrm{e}^{-z_2^2/2\sigma_{s,2}^2}}{\sqrt{2\pi}\sigma_{s,2}}$$

where $\sigma_{s,j}^2$ is the variance of z_j ($j = 1, 2$) when the source is s and $P(s = 0) = P(s = 1)$. Show that

$$P(s = 1|\mathbf{z}) = \frac{1}{1 + \exp\left[-\mathbf{w} \cdot \mathbf{x} + \theta\right]},$$

where $x_j = z_j^2$. Hint: Use Bayes' rule and the law of total probability.

(b) Consider an LED display with 7 elements (as on a digital clock), shown below. The state of the display is a vector $\mathbf{x}$. When the controller wants to display a number $s \in \{0, 1, 2, ..., 9\}$, each of the elements x_j ($j = 1, 2, ..., 7$) either adopts its intended state $c_j(s)$ with probability $1 - f$ or is flipped with probability f. Call the two states of x_j '+1' if the element is 'on' and '0' if the element is 'off.'

Assume one knows the intended character s is either a 2 or 3 and $P(s = 2) = P(s = 3)$, what is the probability of s given the state $\mathbf{x}$? Show $P(s = 2|\mathbf{x})$ can be written

$$P(s = 2|\mathbf{x}) = \frac{1}{1 + \exp\left[-\mathbf{w} \cdot \mathbf{x} + \theta\right]},$$

and compute the values of the weights $\mathbf{w}$ in the case $f = 0.1$. Hint: You will need to express a subset of the likelihood ratios $\log\left[P(x_j|s = 3)/P(x_j|s = 2)\right]$ as functions of x_j.

(c) Consider your findings in (b) and propose a way for the system which encodes the 10 different numbers $s \in \{0, 1, 2, ..., 9\}$ could be improved to reduce the probability of confusing two numbers, while still restricting your self to 7 LED elements. You do not have to work this out in detail, but just describe a problem with the system as is, and how you could try and fix this.

3. ***NO COLLABORATION PROBLEM!*** **anti-BCM rule in a recurrent network.**

(a) Consider the following model of a self-coupled excitatory network subject to the 'anti-BCM' or 'homeostatic' plasticity rule. The neural population v and its self coupling w are given:

$$\frac{dv}{dt} = -v + wv, \qquad \frac{dw}{dt} = \gamma(1 - v)v, \qquad \gamma > 0.$$

Describe what each of the terms in each differential equation means neurobiologically. In particular, what happens for large v versus small $v > 0$ in the w differential equation.

(b) Find the equilibria of the system of differential equations and classify their linear stability.

(c) Sketch the nullclines, fixed point(s), and two example trajectories in the phase plane consistent with your findings in (b). Explain what occurs in the model, and interpret your findings neurobiologically.

python exercises.

4. **Training the NOT AND function into a two layer network.** Write a python code for training the weights of the two layer perceptron given in Eq. (1) in Problem 1c to yield a "soft NOT AND" function, so $x_1 = x_2 = 0$ implies $z = 0.95$; $x_1 = x_2 = 1$ implies $z = 0.05$; $x_1 = 1$ and $x_2 = 0$ implies $z = 0.95$; and $x_1 = 0$ and $x_2 = 1$ implies $z = 0.95$. Pick random initial weights and biases (each drawn uniform randomly between -1 and 1). Then train the weights and biases by presenting $\mathbf{x} = (0, 0)$, (1,0), (1,0), (1,1) in succession and backpropagating once based on the error $E = (z - z_{\rm targ})^2$ from the required NOT AND outputs $z_{\rm targ}$, and then repeating this process over and over (in a loop) until a required error tolerance is reached. Note, you should compute the functional form of all the error function derivatives (e.g., $\frac{\partial E}{\partial J_1}$, $\frac{\partial E}{\partial w_{11}}$, $\frac{\partial E}{\partial \theta_1}$) ahead of time and use these in your code.

   (a) Run your code, taking the learning rate $r = 0.5$. Terminate when the maximum error $E = (z - z_{\rm targ})^2$ across all four conditions is less than 0.05 or when you reach 10000 iterations. Plot the maximum error at each iteration as a function of iteration number. Does it always decrease? Report on what you see.

   (b) Run your code 10 times each for the cases where the learning rate is $r = 20$, $r = 10$, $r = 5$, $r = 1$, $r = 0.5$: Make sure you randomize the initial weights and biasesagain each time. Which learning rate leads to the most rapid convergence of the neural network? Explain why you think this is based on what you have learned about feedforward neural networks. What are some ways you could modify this neural network to try and ensure it converges to the desired output function more rapidly?

5. **TensorFlow Playground Exploration.** The last exercise will make use of the online TensorFlow Playground GUI at `https://playground.tensorflow.org` to train binary classifiers in just a few clicks. You will tweak the model's architecture and its hyperparameters to get some intuition on how neural networks work.

   (a) Train the default neural network by clicking the Run button (top left). Notice how it quickly finds a good solution for the classification task. Then replace the tanh activation function with a ReLU activation function, and train the network again. Does the network find a solution quicker or slower than before? Speculate on why you think this is.

   (b) Modify the network to have just one hidden layer with three neurons. Train it multiple times (to reset the network weights, click the Reset button next to the Play button). What happens? Explain why.

   (c) Remove one neuron to keep just two in the hidden layer. Now what happens? Why?

   (d) Increase the number of neurons in the hidden layer to eight, and train the network several times. What happens? Why?

   (e) Select the spiral dataset, and change the network architecture to have four hidden layers with eight neurons each. Run training for a long time. You should observe the consequences of what is often called the 'vanishing gradients' problem. Explain why you think that phrase is relevant to what you are seeing.

   (f) Explore some other aspect of TensorFlow Playground and report what you see.