Project #1: Artificial Neural Networks

Amir Sadovnik
COSC 525: Deep Learning (Spring 2021)

---

1 Overview

In this project you will be implementing a simple artificial neural network (ANN) library. I will ask you to implement it in a specific object oriented manner. Although this will make the library less efficient it will allow you to gain a better understanding of how ANN's predict and learn.

Make sure to use your own solution and your own code.

2 Problem Description

Your are tasked with writing the following functions:

1. Write a `Neuron` class with the following properties:
   (a) Should be initialized with an activation function, the number of inputs, the learning rate, and possibly a vector of weights (if not set to random).
   (b) Needs to store the the input, output and partial derivative for each of its weights for back-propagation.
   (c) Should have an `activate` method which given a value returns its value after activation (depending on the activation function used).
   (d) Should have a `calculate` method which given an input calculates the output.
   (e) Should have a `activationderivaive` method which returns the derivative of the activation function with respect to the net.
   (f) Should have a `calcpartialderivative` method which calculates the partial derivative with respect to each weight (using the `derivative` method) and returns the vector $w \times \delta$.
   (g) Should have a `updateweights` which actually changes the weights using the partial derivative values (and the learning rate).

2. Write a `FullyConnectedLayer` class.

   (a) Should be initialized with the number of neurons in the layer, the activation function for all the neurons in the layer, the number of inputs, the learning rate, and possibly a vector of weights (if not set to random).

   (b) Should have a `calculate` method which given an input calculates the output of all the neurons in the layer.

   (c) Should have a `calculatewdeltas` which given the $\sum w \times \delta$ from the next layer, goes through all the neurons in the layer to calculate their partial derivative (using the `calcpartialderivative` method), updates all the weights (using the `updateweights` method) and returns its own $\sum w \times \delta$.

3. Write a `NeuralNetwork` class.

   (a) Should be initialized with the number of layers, number of neurons in each layer, a vector of activation functions for each layer, the number of inputs to the network, the loss function used, the learning rate, and possibly a vector of weights (if not set to random).

   (b) Should have a `calculate` method which given an input calculates the output of the network.

   (c) Should have a `calculateloss` method which given an input and desired output calculates the loss.

   (d) Should have a `lossderivative` method which returns the value of the loss derivative (depending on the loss).

   (e) Should have a `train` method, which given a single input and desired output takes one step of gradient descent. This method should first do a forward pass (give its own method

   (f) Any additional methods/instance variables needed for back-propagation `calculate`, then calculate the derivative of the loss (using `lossderivative`) and finally go through the layers backwards calling `calculatewdeltas` for each and passing $\sum w \times \delta$. as it goes. Note that the first time `calculatewdeltas` is called it should be passed the derivative loss.

4. Your `main` method should take the following command line variables to show the following:

   (a) If given `example`, simply run a single step of back-propagation using the example we did in class. Print out the weights after the 1-step update to show that you have gotten the same results as we did in class.

   (b) If given `and` should train a single perceptron to produce the output of the "and" logic gate. Train it for enough steps so that it converges. Show the results of the prediction for all 4 possible inputs.

(c) If given `xor` should train tow types of networks to produce the output of the "xor" logic gate. One is a single perceptron, while the other adds a hidden layer. Train it for enough steps so that it converges. Show the results of the prediction for all 4 possible inputs for each of them.

3 Additional Information

You must do so under the following constraints:

1. Your library should support at least two activation functions: logistic and linear
2. Your library should support at least two loss functions: square error and binary cross entropy loss
3. You must use Python3 with only the numpy and sys libraries allowed
4. Make sure to comment your code
5. **Work plan tip:** Start with implementing a feed-forward network. First implement the neuron methods for feed forward, then the fullyconnectedlayer for feed forward, and finally neuralnetwork for feed forward. For each of them you can do some manual calculations to ensure you are getting the correct results. Only then start working on back-propagation. Do so in the same order checking manually along the way.

4 Report

You should submit a short PDF report with the following (if you do not have anything to add for a section, simply put the section title and then state there is nothing to add):

1. A short introduction to the problem.
2. Assumptions/choices you have made.
3. Problems/Issues you were not able to solve.
4. How to run you code (if there is any difference from what is stated)
5. Show a graph of the loss dropping as a function of learning for different learning rates. Try to find learning rates that are too high/low. Describe what you see in the graph in words.

5 Submission

You are required to submit one zip file with the code and your report.