Tutorial Problem Set #9

Due: Wednesday, November 20, 2024, 11:59 PM

Policy

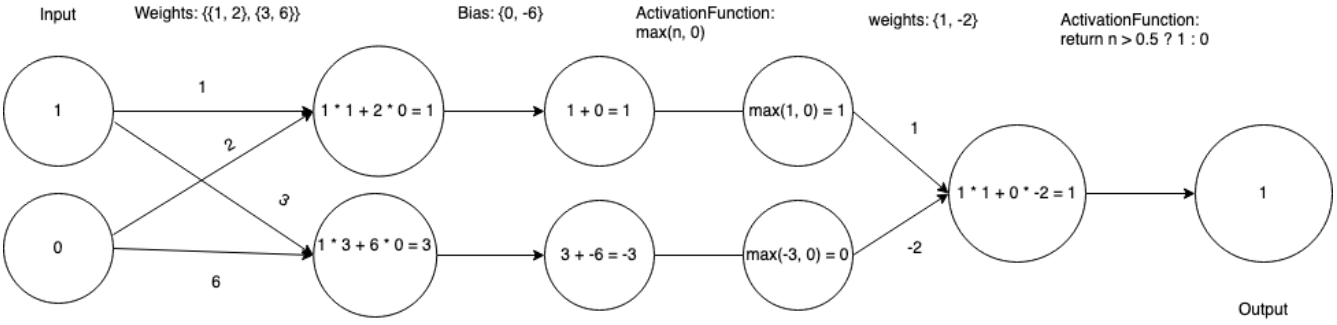
- Piazza questions on tutorial problems will be ignored or deleted. Questions will only be answered in your assigned tutorial section.
- Sample executables can be found in your 1249 git repository directory (run `git pull`).
- Completing the problem set will reduce the weight of the final exam by 0.5%. To complete a problem set, you must pass at least 50% of the public and secret tests.
- You may assume all input is valid. Tutorial problems **NEVER** require checking for invalid inputs.

Question 1

A *neural network* is a data structure used for solving problems involving artificial intelligence. In this question, you will use the Decorator design pattern and `std::vector` to perform the computations for a basic neural network. When completed, you will be able to use simple neural networks to compute boolean functions AND, OR, NOT, and XOR. To compute the output of a neural network, values flow through layers of nodes. We start with input nodes that are fed some initial `float` values. From there, we can perform transformations that change the values of nodes as they flow from layer to layer. Some examples are:

1. *Bias* - each node in the next layer gets the value of the node from the previous layer, plus some constant value.
2. *Activation Function* - each node in the next layer gets the value of the node from the previous layer, after being run through some function. Activation Functions are usually used to "smooth" outputs to some value between 0 and 1. Example activation functions include `tanh`, and `max(0, n)`.
3. *Weighting* - each node in the next layer gets a value that is some linear combination of the values of the previous layer of nodes. Unlike in the Bias and Activation Function transformations, the number of nodes may change from the previous layer.

For a worked example, see the following diagram. This neural network computes the XOR operation for input `(1, 0)`. Tracing the output for `(0, 0)`, `(0, 1)`, and `(1, 1)` will show that this network does indeed produce the correct output for all input values.



A neural network with weights, biases, and activation functions has the potential to approximate any function to any level of precision, which is what makes them useful. The hard part is determining the best number of layers, the best number of nodes in each layer, and the best biases, weights, and activation functions to use in the network. For this question we will

simply provide the correct parameters to compute the boolean functions. If you are interested in solving more challenging problems with neural networks (like handwriting recognition, for example), then you may take CS 479 which discusses algorithms to efficiently determine effective weights and biases.

You are given starter code in `main.cc`. The classes that make up the pattern are:

- `Component` - an abstract class that defines a pure virtual method `calculate`, which takes in a `std::vector<float>` as input to the neural network and returns a `std::vector<float>` as output of the neural network.
- `Input` - The concrete component that models the input nodes to a neural network. It just returns the argument given to `calculate`.
- `Decorator` - The abstract decorator class.
- `Bias` - A concrete decorator that is constructed with the bias vector, a `std::vector<float>`. The return value of `calculate` should perform vector addition: the bias vector plus the values obtained from the previous layer of the network.
- `ActivationFunction` - A concrete decorator that is constructed with a function pointer. The return value of `calculate` should be the input from the previous layer, where the supplied function is applied to each node value (i.e, each element in the vector).
- `Weight` - A concrete decorator that is constructed with a weights matrix `std::vector<std::vector<float>>`. Each vector in the matrix describes the weights used to apply the linear combination from the input. For example, if the previous layer provided {`0.5, 1.0`}, and our weight matrix is {{`2.0, -3.0`}, {`-4.0, 5.0`}, {`0.1, 6.0`}}, then the return value from `calculate` would be {`2.0 * 0.5 + -3.0 * 1.0, -4.0 * 0.5 + 5.0 * 1.0, 0.1 * 0.5 + 6.0 * 1.0`}. Essentially, we are perfoming matrix-vector multiplication:

$$\begin{bmatrix} 2 & -3 \\ -4 & 5 \\ 0.1 & 6 \end{bmatrix} \begin{bmatrix} 0.5 \\ 1.0 \end{bmatrix} = \begin{bmatrix} 2*0.5 + -3*1 \\ -4*0.5 + 5*1 \\ 0.1*0.5 + 6*1 \end{bmatrix}$$

Although this question may look challenging, each decorator's `calculate` method can be written in 10 lines of code or less. And, the classes you write are powerful enough to solve many interesting AI problems given the correct network parameters. This is a testament to neural networks as problem-solving tool, and to the Decorator as a useful design pattern.

Submission

Fill out the classes in `main.cc` and submit the file to Marmoset.