Forward Propagation in Neural Networks.

In this reading you will implement code to make predictions using a feed-forward neural network. This process is also called forward propagation, as we propagate information forward from the input layer, through the hidden layers to the output layer.

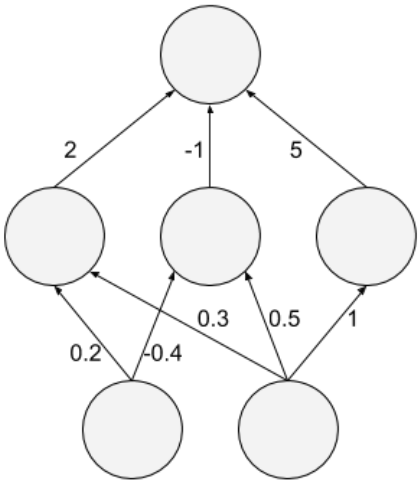
The first question we need to address is how do we represent the network's architecture and weights. Previously, we represented the weights for each neuron as a vector $w \in \mathbb{R}^n$ where n is the number of inputs to the neuron. This allows calculating the output of a neuron using vector notation, namely $z = b + w^t x$ for a linear neuron, or $\frac{1}{1+e^{-(z)}}$ for a logistic neuron.

Taking this a step forward we can represent the weights for all neurons in the same layer as a matrix W, where each column in the matrix is the weight vector of a specific neuron. Thus for a layer with m neurons, connected to n inputs, the dimension of W is $n \times m$. This representation is very useful, since it allows preforming calculations using matrix multiplication.

We can now represent the entire network as an array of matrices, where the number of layers in the network determines the number of matrices, and the number of neurons in each layer set the dimensions of each matrix. Note that this representation can accommodate networks in which not all nodes in one layer are connected to all nodes in the next layer, since a weight of 0 is equivalent to a missing edge.

Toy example

Consider the following neural network:



This network can be represented by two matrices

$$W_1 = \begin{bmatrix} 0.2 & -0.4 & 0 \\ 0.3 & 0.5 & 1 \end{bmatrix}$$

$$W_2 = \begin{bmatrix} 2 \\ -1 \\ 5 \end{bmatrix}$$

Implementing Forward Propagation

We will use Octave to implement Forward Propagation for a neural network with one hidden layer, where the hidden layer is composed of **logistic** neurons, and the output layer is a **linear** neuron. For simplicity we will assume there are no bias terms in our network.

Your function should take in:

$W_1 \in \mathbb{R}^{n \times m}$	A $n \times m$ matrix representing the weights from the input layer to the hidden layer.
$W_2 \in \mathbb{R}^{m}$	A m-dimensional vector representing weights between the hidden layer and the output layer.
$\mathbf{X} \in \mathbb{R}^{k \times n}$	A $k \times n$ matrix representing the data points

Where k is the number of data point, n is the number of inputs or features, and m the number of neurons in the hidden layer. Your implementation should work for a range of positive m and n.

Try to vectorize as much of your function as you can.

```
function predictions = predict(W1,W2, X)
  hidden_layer = X*W1;% Output is matrix of k x m
  hidden_layer = 1./(1+exp(-1*hidden_layer)); %Element-wise Sigmoid
  predictions = hidden_layer*W2; %Output of k x 1
endfunction
```

Run

Reset

✓ Complete