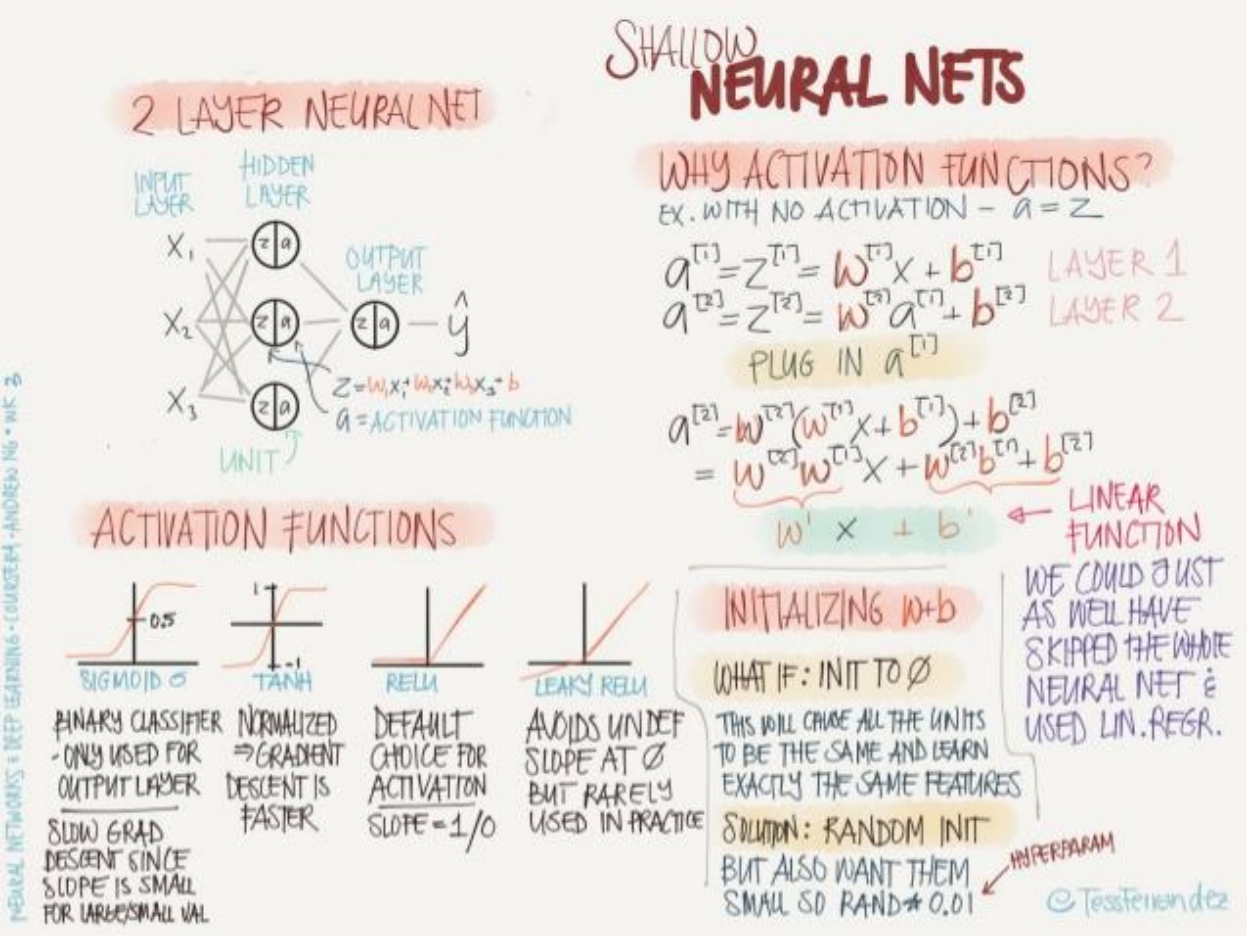
Shallow neural networks

- Neural Networks Overview
- Neural Network Representation
- Computing a Neural Network's Output
- Vectorizing across multiple examples
- Activation functions
- Why do you need non-linear activation functions?
- Derivatives of activation functions
- Gradient descent for Neural Networks
- Random Initialization



Shallow neural networks

Learn to build a neural network with one hidden layer, using forward propagation and backpropagation.

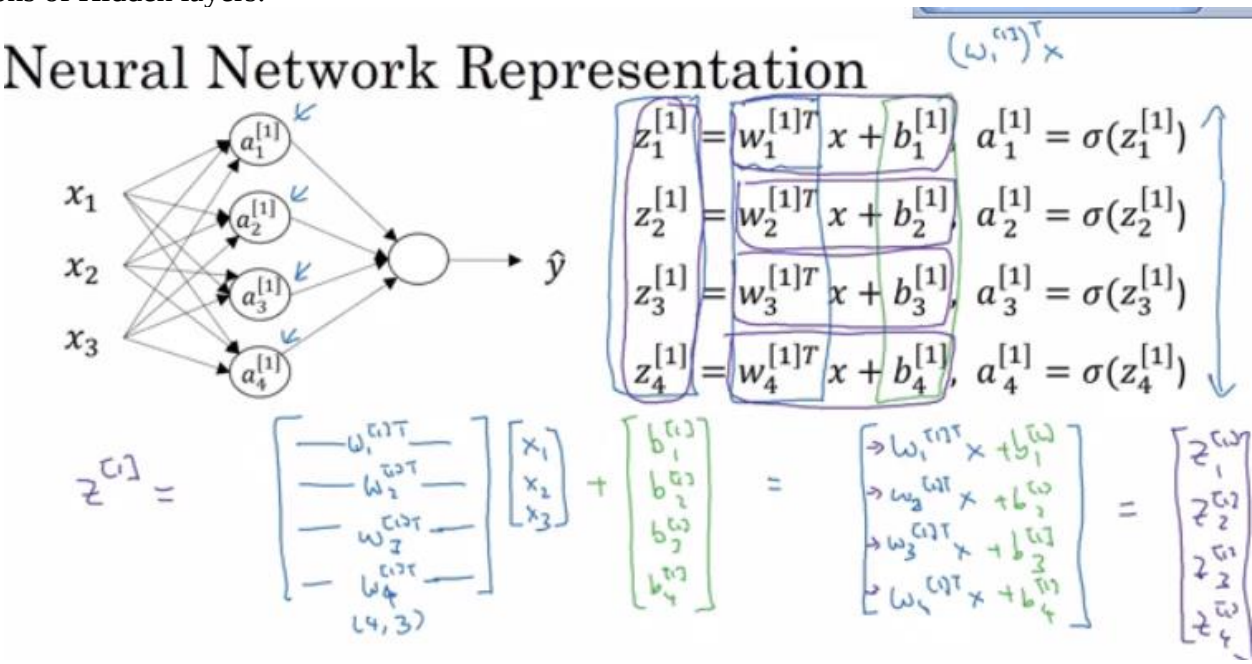
Neural Networks Overview

- In logistic regression we had:
- `X1  \`
- `X2   ==>  z = XW + B ==> a = Sigmoid(z) ==> l(a,Y)`
- `X3  /`
- In neural networks with one layer we will have:
- `X1  \`
- `X2   =>  z1 = XW1 + B1 => a1 = Sigmoid(z1) => z2 = a1W2 + B2 => a2 = Sigmoid(z2) => l(a2,Y)`
- `X3  /`
- `X` is the input vector `(X1, X2, X3)`, and `Y` is the output variable `(1x1)`
- NN is stack of logistic regression objects.

Neural Network Representation

- We will define the neural networks that has one hidden layer.
- NN contains of input layers, hidden layers, output layers.
- Hidden layer means we cant see that layers in the training set.
- `a0 = x` (the input layer)
- `a1` will represent the activation of the hidden neurons.
- `a2` will represent the output layer.
- We are talking about 2 layers NN. The input layer isn't counted.

Computing a Neural Network's Output

- Equations of Hidden layers:
  - 

- Here are some informations about the last image:
  - `noOfHiddenNeurons = 4`

- `Nx = 3`
- Shapes of the variables:
  - `W1` is the matrix of the first hidden layer, it has a shape of `(noOfHiddenNeurons,nx)`
  - `b1` is the matrix of the first hidden layer, it has a shape of `(noOfHiddenNeurons,1)`
  - `z1` is the result of the equation `z1 = W1*X + b`, it has a shape of `(noOfHiddenNeurons,1)`
  - `a1` is the result of the equation `a1 = sigmoid(z1)`, it has a shape of `(noOfHiddenNeurons,1)`
  - `W2` is the matrix of the second hidden layer, it has a shape of `(1,noOfHiddenNeurons)`
  - `b2` is the matrix of the second hidden layer, it has a shape of `(1,1)`
  - `z2` is the result of the equation `z2 = W2*a1 + b`, it has a shape of `(1,1)`
  - `a2` is the result of the equation `a2 = sigmoid(z2)`, it has a shape of `(1,1)`

Vectorizing across multiple examples

- Pseudo code for forward propagation for the 2 layers NN:
- `for i = 1 to m`
- `  z[1, i] = W1*x[i] + b1  # shape of z[1, i] is (noOfHiddenNeurons,1)`
- `  a[1, i] = sigmoid(z[1, i])  # shape of a[1, i] is (noOfHiddenNeurons,1)`
- `  z[2, i] = W2*a[1, i] + b2  # shape of z[2, i] is (1,1)`
- `  a[2, i] = sigmoid(z[2, i])  # shape of a[2, i] is (1,1)`
- Lets say we have `X` on shape `(Nx,m)`. So the new pseudo code:
- `Z1 = W1X + b1  # shape of Z1 (noOfHiddenNeurons,m)`
- `A1 = sigmoid(Z1)  # shape of A1 (noOfHiddenNeurons,m)`
- `Z2 = W2A1 + b2  # shape of Z2 is (1,m)`
- `A2 = sigmoid(Z2)  # shape of A2 is (1,m)`
- If you notice always m is the number of columns.
- In the last example we can call `X` = `A0`. So the previous step can be rewritten as:
- `Z1 = W1A0 + b1  # shape of Z1 (noOfHiddenNeurons,m)`
- `A1 = sigmoid(Z1)  # shape of A1 (noOfHiddenNeurons,m)`
- `Z2 = W2A1 + b2  # shape of Z2 is (1,m)`
- `A2 = sigmoid(Z2)  # shape of A2 is (1,m)`

Activation functions

- So far we are using sigmoid, but in some cases other functions can be a lot better.
- Sigmoid can lead us to gradient decent problem where the updates are so low.
- Sigmoid activation function range is [0,1] `A = 1 / (1 + np.exp(-z)) # Where z is the input matrix`
- Tanh activation function range is [-1,1] (Shifted version of sigmoid function)
  - In NumPy we can implement Tanh using one of these methods: `A = (np.exp(z) - np.exp(-z)) / (np.exp(z) + np.exp(-z)) # Where z is the input matrix`

Or `A = np.tanh(z) # Where z is the input matrix`

- It turns out that the tanh activation usually works better than sigmoid activation function for hidden units because the mean of its output is closer to zero, and so it centers the data better for the next layer.
- Sigmoid or Tanh function disadvantage is that if the input is too small or too high, the slope will be near zero which will cause us the gradient decent problem.
- One of the popular activation functions that solved the slow gradient decent is the RELU function. `RELU = max(0,z) # so if z is negative the slope is 0 and if z is positive the slope remains linear.`
- So here is some basic rule for choosing activation functions, if your classification is between 0 and 1, use the output activation as sigmoid and the others as RELU.
- Leaky RELU activation function different of RELU is that if the input is negative the slope will be so small. It works as RELU but most people uses RELU. `Leaky_RELU = max(0.01z,z) #the 0.01 can be a parameter for your algorithm.`
- In NN you will decide a lot of choices like:
  - No of hidden layers.
  - No of neurons in each hidden layer.
  - Learning rate. (The most important parameter)
  - Activation functions.
  - And others..
- It turns out there are no guide lines for that. You should try all activation functions for example.

Why do you need non-linear activation functions?

- If we removed the activation function from our algorithm that can be called linear activation function.
- Linear activation function will output linear activations
  - Whatever hidden layers you add, the activation will be always linear like logistic regression (So its useless in a lot of complex problems)
- You might use linear activation function in one place - in the output layer if the output is real numbers (regression problem). But even in this case if the output value is non-negative you could use RELU instead.

Derivatives of activation functions

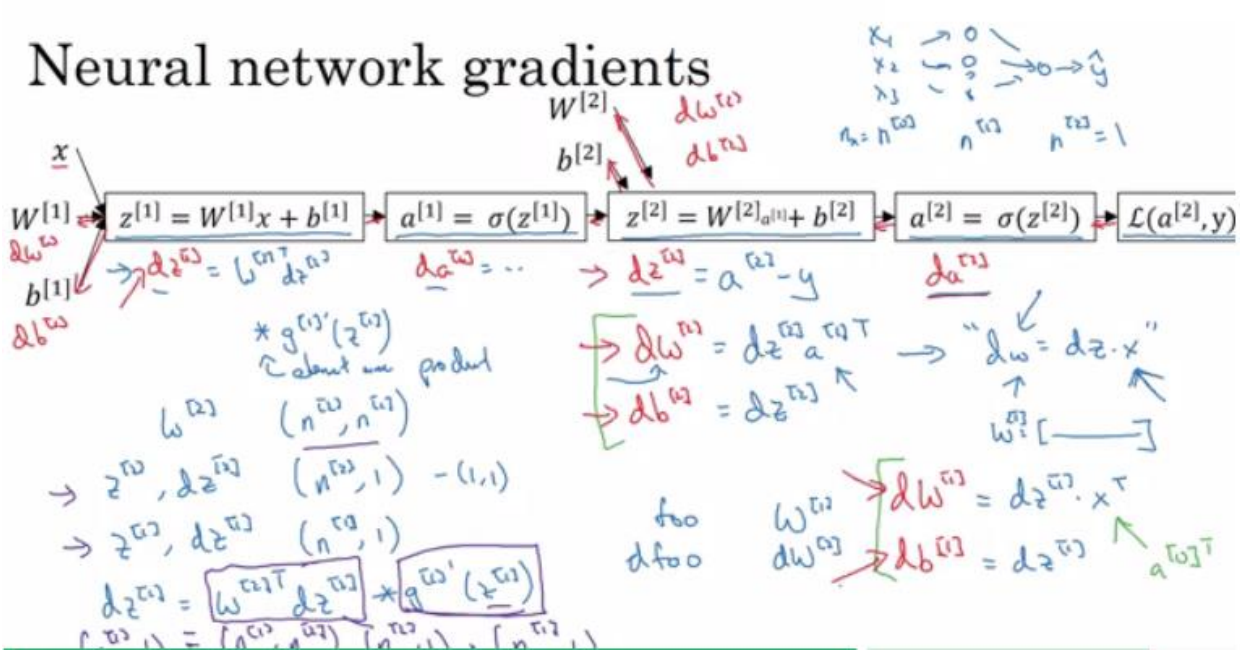
- Derivation of Sigmoid activation function:
- `g(z) = 1 / (1 + np.exp(-z))`
- `g'(z) = (1 / (1 + np.exp(-z))) * (1 - (1 / (1 + np.exp(-z))))`
- `g'(z) = g(z) * (1 - g(z))`
- Derivation of Tanh activation function:
- `g(z)  = (e^z - e^-z) / (e^z + e^-z)`
- `g'(z) = 1 - np.tanh(z)^2 = 1 - g(z)^2`
- Derivation of RELU activation function:
- `g(z)  = np.maximum(0,z)`
- `g'(z) = { 0  if z < 0`

- `1  if z >= 0  }`
- Derivation of leaky RELU activation function:
- `g(z)  = np.maximum(0.01 * z, z)`
- `g'(z) = { 0.01  if z < 0`
- `1     if z >= 0   }`

Gradient descent for Neural Networks

- In this section we will have the full back propagation of the neural network (Just the equations with no explanations).
- Gradient descent algorithm:
  - NN parameters:
    - `n[0] = Nx`
    - `n[1] = NoOfHiddenNeurons`
    - `n[2] = NoOfOutputNeurons = 1`
    - `W1` shape is `(n[1],n[0])`
    - `b1` shape is `(n[1],1)`
    - `W2` shape is `(n[2],n[1])`
    - `b2` shape is `(n[2],1)`
  - Cost function `I = I(W1, b1, W2, b2) = (1/m) * Sum(L(Y,A2))`
  - Then Gradient descent:
  - `Repeat:`
  - `Compute predictions (y'[i], i = 0,...m)`
  - `Get derivatives: dW1, db1, dW2, db2`
  - `Update: W1 = W1 - LearningRate * dW1`
  - `b1 = b1 - LearningRate * db1`
  - `W2 = W2 - LearningRate * dW2`
  - `b2 = b2 - LearningRate * db2`
- Forward propagation:
- `Z1 = W1A0 + b1    # A0 is X`
- `A1 = g1(Z1)`
- `Z2 = W2A1 + b2`
- `A2 = Sigmoid(Z2)      # Sigmoid because the output is between 0 and 1`
- Backpropagation (derivations):
- `dZ2 = A2 - Y      # derivative of cost function we used * derivative of the sigmoid function`
- `dW2 = (dZ2 * A1.T) / m`
- `db2 = Sum(dZ2) / m`
- `dZ1 = (W2.T * dZ2) * g'1(Z1)  # element wise product (*)`
- `dW1 = (dZ1 * A0.T) / m   # A0 = X`
- `db1 = Sum(dZ1) / m`
- `# Hint there are transposes with multiplication because to keep dimensions correct`

- How we derived the 6 equations of the backpropagation:



Random Initialization

- In logistic regression it wasn't important to initialize the weights randomly, while in NN we have to initialize them randomly.
- If we initialize all the weights with zeros in NN it won't work (initializing bias with zero is OK):
    - all hidden units will be completely identical (symmetric) - compute exactly the same function
    - on each gradient descent iteration all the hidden units will always update the same
- To solve this we initialize the W's with a small random numbers:
- `W1 = np.random.randn((2,2)) * 0.01  # 0.01 to make it small enough`
- `b1 = np.zeros((2,1))  # its ok to have b as zero, it won't get us to the symmetry breaking problem`
- We need small values because in sigmoid (or tanh), for example, if the weight is too large you are more likely to end up even at the very start of training with very large values of Z. Which causes your tanh or your sigmoid activation function to be saturated, thus slowing down learning. If you don't have any sigmoid or tanh activation functions throughout your neural network, this is less of an issue.
- Constant 0.01 is alright for 1 hidden layer networks, but if the NN is deep this number can be changed but it will always be a small number.

Week 3 Quiz - Shallow Neural Networks

1. Which of the following are true? (Check all that apply.)

- [ ] $a_4^{[2]}$ is the activation output by the 4^{th} neuron of the 2^{nd} layer
- [ ] $a^{[2](12)}$ denotes the activation vector of the 2^{nd} layer for the 12^{th} training example.
- [ ] X is a matrix in which each row is one training example.
- [ ] $a^{[2]}$ denotes the activation vector of the 2^{nd} layer.
- [ ] $a^{[2](12)}$ denotes activation vector of the 12^{th} layer on the 2^{nd} training example.
- [ ] $a_4^{[2]}$ is the activation output of the 2^{nd} layer for the 4^{th} training example
- [ ] X is a matrix in which each column is one training example.

Notice that I only list correct options.

- X is a matrix in which each column is one training example.
- a^[2]_4 is the activation output by the 4th neuron of the 2nd layer
- a^[2](12) denotes the activation vector of the 2nd layer for the 12th training example.
- a^[2] denotes the activation vector of the 2nd layer.

Note: If you are not familiar with the notation used in this course, check here.

2. The tanh activation usually works better than sigmoid activation function for hidden units because the mean of its output is closer to zero, and so it centers the data better for the next layer. True/False?
   - ☑ True
   - ☐ False

   Note: You can check this post and (this paper)[http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf].

   As seen in lecture the output of the tanh is between -1 and 1, it thus centers the data which makes the learning simpler for the next layer.

3. Which of these is a correct vectorized implementation of forward propagation for layer l, where $1 \leq l \leq L$?

- $Z^{[l]} = W^{[l]} A^{[l-1]} + b^{[l]}$
- $A^{[l]} = g^{[l]}(Z^{[l]})$

- $Z^{[l]} = W^{[l-1]} A^{[l]} + b^{[l-1]}$
- $A^{[l]} = g^{[l]}(Z^{[l]})$

- $Z^{[l]} = W^{[l]} A^{[l]} + b^{[l]}$
- $A^{[l+1]} = g^{[l]}(Z^{[l]})$

- $Z^{[l]} = W^{[l]} A^{[l]} + b^{[l]}$
- $A^{[l+1]} = g^{[l+1]}(Z^{[l]})$

  - Z^[l]=W^[l]A^[l−1]+b^[l]
  - A^[l]=g^[l](Z^[l])

4. You are building a binary classifier for recognizing cucumbers (y=1) vs. watermelons (y=0). Which one of these activation functions would you recommend using for the output layer?
   - ☐ ReLU
   - ☐ Leaky ReLU
   - ☑ sigmoid
   - ☐ tanh

   Note: The output value from a sigmoid function can be easily understood as a probability.

   Sigmoid outputs a value between 0 and 1 which makes it a very good choice for binary classification. You can classify as 0 if the output is less than 0.5 and classify as 1 if the output is more than 0.5. It can be done with tanh as well but it is less convenient as the output is between -1 and 1.

5. Consider the following code:

```
A = np.random.randn(4,3)
B = np.sum(A, axis = 1, keepdims = True)
```

   What will be B.shape?

```
B.shape = (4, 1)
```

   we use (keepdims = True) to make sure that A.shape is (4,1) and not (4,). It makes our code more rigorous.

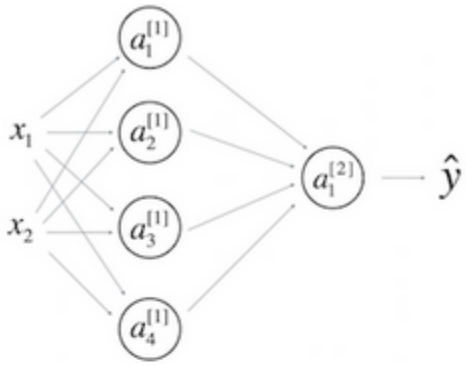
6. Suppose you have built a neural network. You decide to initialize the weights and biases to be zero. Which of the following statements are True? (Check all that apply)
    - ☑ Each neuron in the first hidden layer will perform the same computation. So even after multiple iterations of gradient descent each neuron in the layer will be computing the same thing as other neurons.
    - ☐ Each neuron in the first hidden layer will perform the same computation in the first iteration. But after one iteration of gradient descent they will learn to compute different things because we have “broken symmetry”.
    - ☐ Each neuron in the first hidden layer will compute the same thing, but neurons in different layers will compute different things, thus we have accomplished “symmetry breaking” as described in lecture.
    - ☐ The first hidden layer’s neurons will perform different computations from each other even in the first iteration; their parameters will thus keep evolving in their own way.
7. Logistic regression’s weights w should be initialized randomly rather than to all zeros, because if you initialize to all zeros, then logistic regression will fail to learn a useful decision boundary because it will fail to “break symmetry”, True/False?
    - ☐ True
    - ☑ False

      Logistic Regression doesn't have a hidden layer. If you initialize the weights to zeros, the first example x fed in the logistic regression will output zero but the derivatives of the Logistic Regression depend on the input x (because there's no hidden layer) which is not zero. So at the second iteration, the weights values follow x's distribution and are different from each other if x is not a constant vector.

8. You have built a network using the tanh activation for all the hidden units. You initialize the weights to relative large values, using np.random.randn(..,..)*1000. What will happen?
    - ☐ It doesn’t matter. So long as you initialize the weights randomly gradient descent is not affected by whether the weights are large or small.
    - ☐ This will cause the inputs of the tanh to also be very large, thus causing gradients to also become large. You therefore have to set α to be very small to prevent divergence; this will slow down learning.
    - ☐ This will cause the inputs of the tanh to also be very large, causing the units to be “highly activated” and thus speed up learning compared to if the weights had to start from small values.
    - ☑ This will cause the inputs of the tanh to also be very large, thus causing gradients to be close to zero. The optimization algorithm will thus become slow.

tanh becomes flat for large values, this leads its gradient to be close to zero. This slows down the optimization algorithm.

9. Consider the following 1 hidden layer neural network:



Which of the following statements are True? (Check all that apply).

- [ ] $W^{[1]}$ will have shape (2, 4)
- [ ] $b^{[1]}$ will have shape (4, 1)
- [ ] $W^{[1]}$ will have shape (4, 2)
- [ ] $b^{[1]}$ will have shape (2, 1)
- [ ] $W^{[2]}$ will have shape (1, 4)
- [ ] $b^{[2]}$ will have shape (4, 1)
- [ ] $W^{[2]}$ will have shape (4, 1)
- [ ] $b^{[2]}$ will have shape (1, 1)

- b[1] will have shape (4, 1)
- W[1] will have shape (4, 2)
- W[2] will have shape (1, 4)
- b[2] will have shape (1, 1)

Note: Check here for general formulas to do this.

10. In the same network as the previous question, what are the dimensions of Z^[1] and A^[1]?
    - Z[1] and A[1] are (4,m)

Note: Check here for general formulas to do this.