[✓] $W^{[1]}$ is a matrix with rows equal to the transpose of the parameter vectors of the first layer.

✓ **Correct**

Yes. We construct $W^{[1]}$ stacking the parameter vectors $w_j^{[1]}$ of all the neurons of the first layer.

[✓] $w_3^{[4]}$ is the column vector of parameters of the fourth layer and third neuron.

✓ **Correct**

Yes. The vector $w_j^{[i]}$ is the column vector of parameters of the i-th layer and j-th neuron of that layer.

[ ] $w_3^{[4]}$ is the column vector of parameters of the third layer and fourth neuron.

[ ] $w_3^{[4]}$ is the row vector of parameters of the fourth layer and third neuron.

[ ] W_1 is a matrix with rows equal to the parameter vectors of the first layer.

[ ] $W^{[1]}$ is a matrix with rows equal to the parameter vectors of the first layer.

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?

- tanh
- [✓ sigmoid]
- ReLU
- Leaky ReLU

Expand

Correct

Yes. 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.

Which of the following is a correct vectorized implementation of forward propagation for layer 2?

- ○ $Z^{[1]} = W^{[1]}X + b^{[1]}$
  $A^{[1]} = g^{[1]}(Z^{[1]})$
- ○ $Z^{[2]} = W^{[2]}X + b^{[2]}$
  $A^{[2]} = g^{[2]}(Z^{[2]})$
- [✓ $Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$
  $A^{[2]} = g^{[2]}(Z^{[2]})$]
- ○ $Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$
  $A^{[2]} = g(Z^{[2]})$

Expand

Correct

Yes. The elements of layer two are represented using a superscript in brackets.

The tanh activation is not always better than sigmoid activation function for hidden units because the mean of its output is closer to zero, and so it centers the data, making learning complex for the next layer. True/False?

- ○ False
- [✓ True]

Expand

Incorrect

No. 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.

Suppose you have built a neural network with one hidden layer and tanh as activation function for the hidden layer. You decide to initialize the weights to small random numbers and the biases to zero. The first hidden layer's neurons will perform different computations from each other even in the first iteration. True/False?

- [✓ True Yes. Since the weights are most likely different, each neuron will do a different computation.]
- False No. Since the weights are most likely different, each neuron will do a different computation.

Consider the following code:

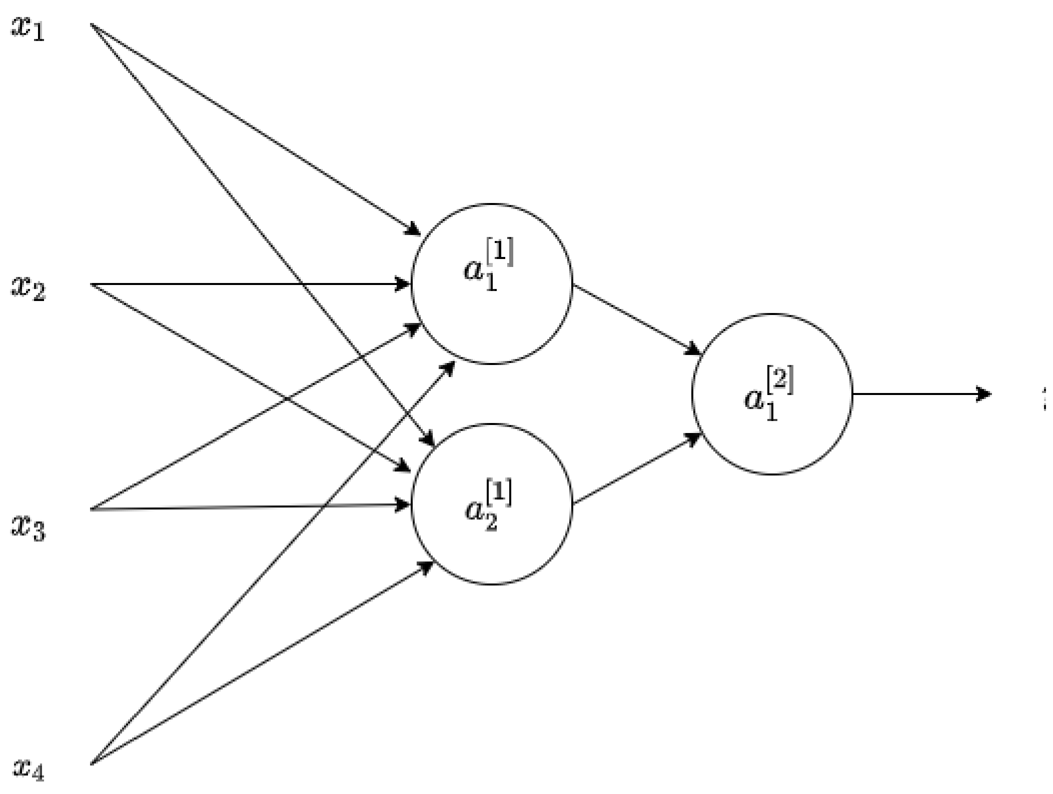
A = np.random.randn(4,3)

B = np.sum(A, axis = 1, keepdims = True)

What will be B.shape? (If you're not sure, feel free to run this in python to find out).

- (3,)
- (1, 3)
- (4,)
- [✓ (4, 1)]

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).

✓ **Correct**

Yes. The number of rows in $W^{[k]}$ is the number of neurons in the k-th layer and the number of columns is the number of inputs of the layer.

☑ $b^{[1]}$ will have shape (2, 1).

✓ **Correct**

Yes. $b^{[k]}$ is a column vector and has the same number of rows as neurons in the k-th layer.

☐ $W^{[1]}$ will have shape (4, 2).

☑ $W^{[2]}$ will have shape (1, 2)

✓ **Correct**

Yes. The number of rows in $W^{[k]}$ is the number of neurons in the k-th layer and the number of columns is the number of inputs of the layer.

☐ $b^{[1]}$ will have shape (4, 2)

Logistic regression's weights 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]

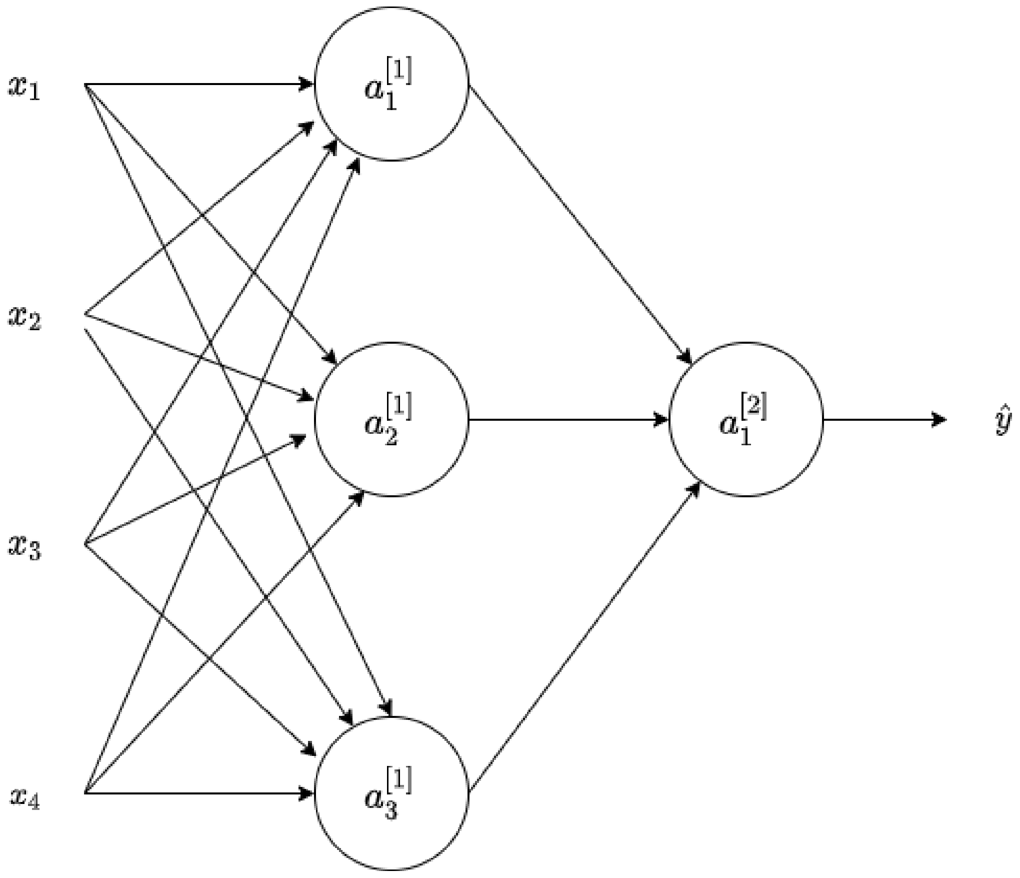
Expand

Correct
Yes, Logistic Regression doesn't have a hidden layer. If you initialize the weights to zeros, the first example x fed into 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.

Which of the following is true about the ReLU activation functions?

- [✓ They are the go to option when you don't know what activation function to choose for hidden layers.]
- They cause several problems in practice because they have no derivative at 0. That is why Leaky ReLU was invented.
- They are increasingly being replaced by the tanh in most cases.
- They are only used in the case of regression problems, such as predicting house prices.

Consider the following 1 hidden layer neural network:



What are the dimensions of $Z^{[1]}$ and $A^{[1]}$?

- ○ $Z^{[1]}$ and $A^{[1]}$ are (4, m)
- ○ $Z^{[1]}$ and $A^{[1]}$ are (3, 1)
- ○ $Z^{[1]}$ and $A^{[1]}$ are (4, 1)
- [✓ $Z^{[1]}$ and $A^{[1]}$ are (3, m)]

Expand

Correct

Yes. The $Z^{[1]}$ and $A^{[1]}$ are calculated over a batch of training examples. The number of columns in $Z^{[1]}$ and $A^{[1]}$ is equal to the number of examples in the batch, m. And the number of rows in $Z^{[1]}$ and $A^{[1]}$ is equal to the number of neurons in the first layer.