1. Which of the following is stored in the 'cache' during forward propagation for latter use in backward propagation? 1 / 1 point

- [✓ $Z^{[l]}$]
- $W^{[l]}$
- $b^{[l]}$

Correct
Yes. This value is useful in the calculation of $dW^{[l]}$ in the backward propagation.

2. During the backpropagation process, we use gradient descent to change the hyperparameters. True/False? 1 / 1 point

- True
- [✓ False]

Correct
Correct. During backpropagation, we use gradient descent to compute new values of $W^{[l]}$ and $b^{[l]}$. These are the parameters of the network.

3. Which of the following statements is true? 1 / 1 point

- The earlier layers of a neural network are typically computing more complex features of the input than the deeper layers.
- [✓ The deeper layers of a neural network are typically computing more complex features of the input than the earlier layers.]

Correct

4. Vectorization allows us to compute $a^{[l]}$ for all the examples on a batch at the same time without using a for loop. True/False?

1 / 1 point

[✓ True]

False

Correct

Correct. Vectorization allows us to compute the activation for all the training examples at the same time, avoiding the use of a for loop.

5. Assume we store the values for $n^{[l]}$ in an array called layer_dims, as follows: layer_dims = [n_x, 4,3,2,1]. So layer 1 has four hidden units, layer 2 has 3 hidden units and so on. Which of the following for-loops will allow you to initialize the parameters for the model?

1 / 1 point

- ○

```
for i in range(1, len(layer_dims)/2):

parameter['W' + str(i)] = np.random.randn(layer_dims[i], layer_dims[i-1]) * 0.01

parameter['b' + str(i)] = np.random.randn(layer_dims[i], 1) * 0.01
```

- [✓]

```
for i in range(1, len(layer_dims)):

parameter['W' + str(i)] = np.random.randn(layer_dims[i], layer_dims[i-1]) * 0.01

parameter['b' + str(i)] = np.random.randn(layer_dims[i], 1) * 0.01
```

- ○

```
for i in range(1, len(layer_dims)/2):
```

parameter['W' + str(i)] = np.random.randn(layer_dims[i], layer_dims[i-1]) * 0.01

parameter['b' + str(i)] = np.random.randn(layer_dims[i-1], 1) * 0.01

○

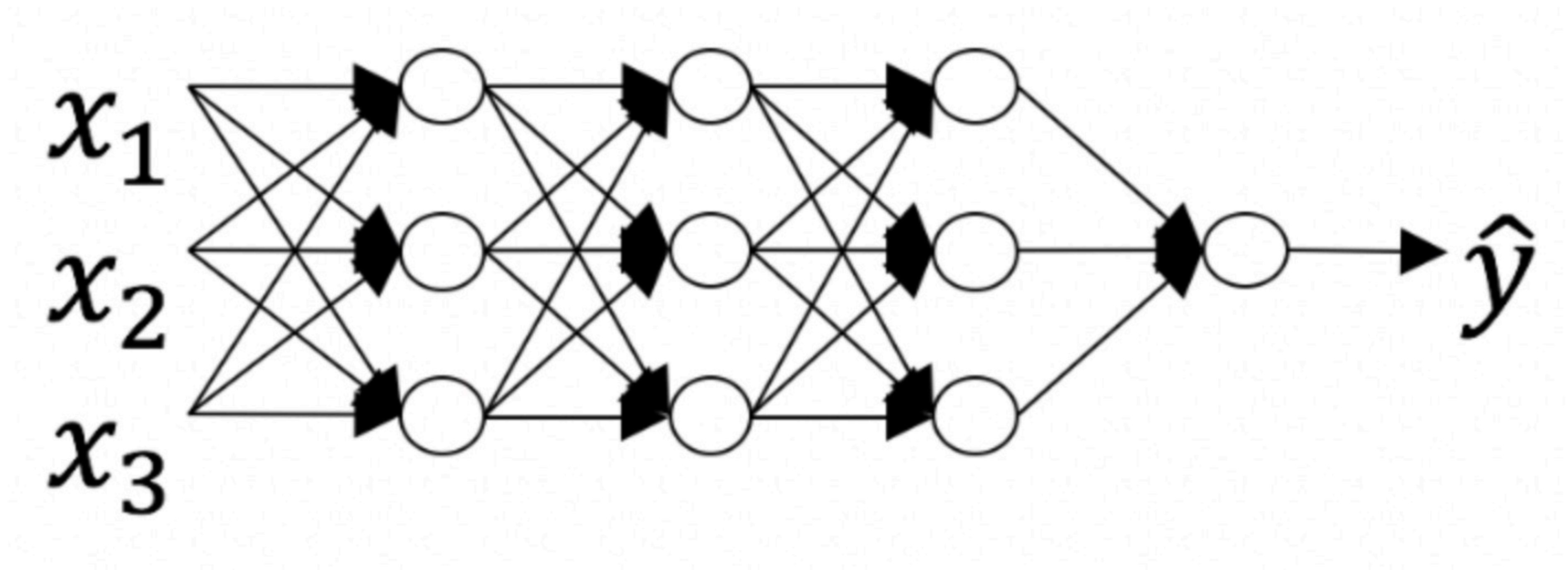
for i in range(1, len(layer_dims)):

parameter['W' + str(i)] = np.random.randn(layer_dims[i-1], layer_dims[i]) * 0.01

parameter['b' + str(i)] = np.random.randn(layer_dims[i], 1) * 0.01

✓ **Correct**

6. Consider the following neural network. 1 / 1 point



How many layers does this network have?

○ The number of layers L is 5. The number of hidden layers is 4.

○ The number of layers L is 4. The number of hidden layers is 4.

- ○ The number of layers L is 3. The number of hidden layers is 3.
- [✓ The number of layers L is 4. The number of hidden layers is 3.]

Correct
Yes. As seen in lecture, the number of layers is counted as the number of hidden layers + 1. The input and output layers are not counted as hidden layers.

7. During forward propagation, in the forward function for a layer l you need to know what is the activation function in a layer (sigmoid, tanh, ReLU, etc.). During backpropagation, the corresponding backward function also needs to know what is the activation function for layer l, since the gradient depends on it. True/False?

1 / 1 point

- [✓ True]
- ○ False

Correct
Yes, as you've seen in week 3 each activation has a different derivative. Thus, during backpropagation you need to know which activation was used in the forward propagation to be able to compute the correct derivative.

8. For any mathematical function you can compute with an L-layered deep neural network with N hidden units there is a shallow neural network that requires only $\log N$ units, but it is very difficult to train.

1 / 1 point

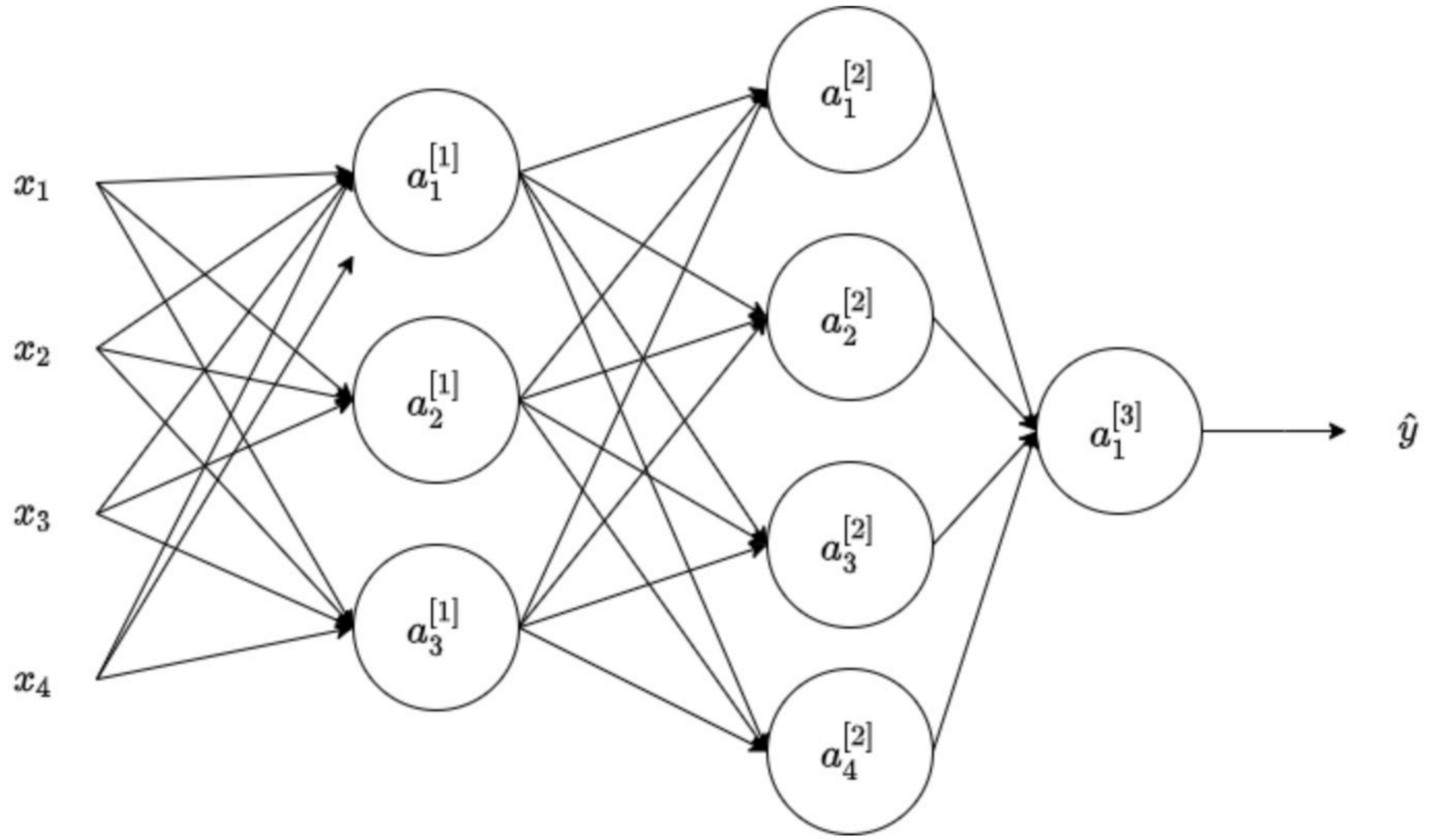
- ○ True
- [✓ False]

Correct

Correct. On the contrary, some mathematical functions can be computed using an L-layered neural network and a given number of hidden units; but using a shallow neural network the number of necessary hidden units grows exponentially.

9. Consider the following 2 hidden layers neural network:

1 / 1 point



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

- ☐ $W^{[2]}$ will have shape (3, 4)
- ☐ $W^{[2]}$ will have shape (1, 3)

- [x] $b^{[1]}$ will have shape (3, 1)

Correct
Yes. More generally, the shape of $b^{[l]}$ is ($n^{[l]}$, 1).

- [ ] $b^{[1]}$ will have shape (4, 1)
- [ ] $W^{[1]}$ will have shape (4, 3)
- [ ] $b^{[1]}$ will have shape (1, 3)
- [x] $W^{[1]}$ will have shape (3, 4)

Correct
Yes. More generally, the shape of $W^{[l]}$ is ($n^{[l]}$, $n^{[l-1]}$).

- [x] $W^{[2]}$ will have shape (4, 3)

Correct
Yes. More generally, the shape of $W^{[l]}$ is ($n^{[l]}$, $n^{[l-1]}$).

- [ ] $W^{[2]}$ will have shape (3, 1)

10. Whereas the previous question used a specific network, in the general case what is the dimension of W^{[l]}, the weight matrix associated with layer l?

1 / 1 point

● $W^{[l]}$ has shape $(n^{[l]}, n^{[l-1]})$

○ $W^{[l]}$ has shape $(n^{[l]}, n^{[l+1]})$

○ $W^{[l]}$ has shape $(n^{[l+1]}, n^{[l]})$

○ $W^{[l]}$ has shape $(n^{[l-1]}, n^{[l]})$

Correct

True