

1. What does a neuron compute?

- ☐ A neuron computes an activation function followed by a linear function ($z = Wx + b$)
- ☐ A neuron computes a function g that scales the input x linearly ($Wx + b$)
- ☐ A neuron computes the mean of all features before applying the output to an activation function
- ☒ A neuron computes a linear function ($z = Wx + b$) followed by an activation function

Correct

Correct, we generally say that the output of a neuron is $a = g(Wx + b)$ where g is the activation function (sigmoid, tanh, ReLU, ...).

2. Which of these is the "Logistic Loss"?

- ☐ $\mathcal{L}^{(i)}(\hat{y}^{(i)}, y^{(i)}) = |y^{(i)} - \hat{y}^{(i)}|$
- ☐ $\mathcal{L}^{(i)}(\hat{y}^{(i)}, y^{(i)}) = \max(0, y^{(i)} - \hat{y}^{(i)})$
- ☒ $\mathcal{L}^{(i)}(\hat{y}^{(i)}, y^{(i)}) = -(y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}))$

Correct

Correct, this is the logistic loss you've seen in lecture!

- ☐ $\mathcal{L}^{(i)}(\hat{y}^{(i)}, y^{(i)}) = |y^{(i)} - \hat{y}^{(i)}|^2$

3. Suppose `img` is a (32,32,3) array, representing a 32x32 image with 3 color channels red, green and blue. How do you reshape this into a column vector?

- ☐ `x = img.reshape((1,32*32,*3))`
- ☐ `x = img.reshape((3,32*32))`
- ☐ `x = img.reshape((32*32,3))`
- ☒ `x = img.reshape((32*32*3,1))`

Correct

4. Consider the two following random arrays "a" and "b":

```
1 a = np.random.randn(2, 3) # a.shape = (2, 3)
2 b = np.random.randn(2, 1) # b.shape = (2, 1)
3 c = a + b
```

What will be the shape of "c"?

- ☐ `c.shape = (2, 1)`
- ☒ `c.shape = (2, 3)`

Correct

Yes! This is broadcasting. `b` (column vector) is copied 3 times so that it can be summed to each column of `a`.

- ☐ `c.shape = (3, 2)`
- ☐ The computation cannot happen because the sizes don't match. It's going to be "Error"!

5. Consider the two following random arrays "a" and "b":

```
1 a = np.random.randn(4, 3) # a.shape = (4, 3)
2 b = np.random.randn(3, 2) # b.shape = (3, 2)
3 c = a*b
```

What will be the shape of "c"?

- ☐ `c.shape = (4,2)`
- ☒ The computation cannot happen because the sizes don't match. It's going to be "Error"!

Correct

Indeed! In numpy the "*" operator indicates element-wise multiplication. It is different from "np.dot()". If you would try "c = np.dot(a,b)" you would get `c.shape = (4, 2)`.

- ☐ `c.shape = (3, 3)`
- ☐ `c.shape = (4, 3)`

6. Suppose you have n_x input features per example. Recall that $X = [x^{(1)} x^{(2)} \dots x^{(m)}]$. What is the dimension of X ?

- ☐ $(1, m)$
- ☐ $(m, 1)$
- ☐ (m, n_x)
- ☒ (n_x, m)

Correct

7. Recall that "np.dot(a,b)" performs a matrix multiplication on a and b, whereas "a*b" performs an element-wise multiplication.

Consider the two following random arrays "a" and "b":

```
1 a = np.random.randn(12288, 150) # a.shape = (12288, 150)
2 b = np.random.randn(150, 45) # b.shape = (150, 45)
3 c = np.dot(a,b)
```

What is the shape of `c`?

- ☐ `c.shape = (12288, 150)`
- ☐ The computation cannot happen because the sizes don't match. It's going to be "Error"!
- ☒ `c.shape = (12288, 45)`

Correct

Correct, remember that a `np.dot(a, b)` has shape (number of rows of `a`, number of columns of `b`). The sizes match because :

"number of columns of `a` = 150 = number of rows of `b`"

- ☐ `c.shape = (150,150)`

8. Consider the following code snippet:

```
1 # a.shape = (3,4)
2 # b.shape = (4,1)
3
4 * for i in range(3):
5   *   for j in range(4):
6     *   c[i][j] = a[i][j] + b[j]
```

How do you vectorize this?

- ☒ `c = a + b.T`

Correct

- ☐ `c = a.T + b.T`
- ☐ `c = a.T + b`
- ☐ `c = a + b`

9. Consider the following code:

```
1 a = np.random.randn(3, 3)
2 b = np.random.randn(3, 1)
3 c = a*b
```

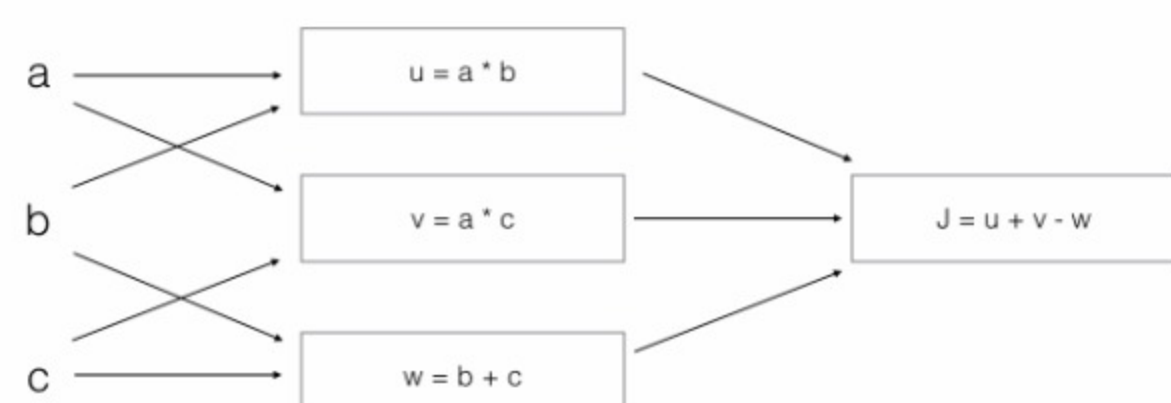
What will be `c`? (If you're not sure, feel free to run this in python to find out).

- ☒ This will invoke broadcasting, so `b` is copied three times to become (3,3), and `*` is an element-wise product so `c.shape` will be (3, 3)

Correct

- ☐ This will invoke broadcasting, so `b` is copied three times to become (3, 3), and `*` invokes a matrix multiplication operation of two 3x3 matrices so `c.shape` will be (3, 3)
- ☐ This will multiply a 3x3 matrix `a` with a 3x1 vector, thus resulting in a 3x1 vector. That is, `c.shape = (3,1)`.
- ☐ It will lead to an error since you cannot use "*" to operate on these two matrices. You need to instead use `np.dot(a,b)`

10. Consider the following computation graph.



What is the output `J`?

- ☐ `J = (c - 1)*(b + a)`
- ☒ `J = (a - 1) * (b + c)`

Correct

Yes. $J = u + v - w = a*b + a*c - (b + c) = a * (b + c) - (b + c) = (a - 1) * (b + c)$.

- ☐ `J = a*b + b*c + a*c`
- ☐ `J = (b - 1) * (c + a)`