Week 2:

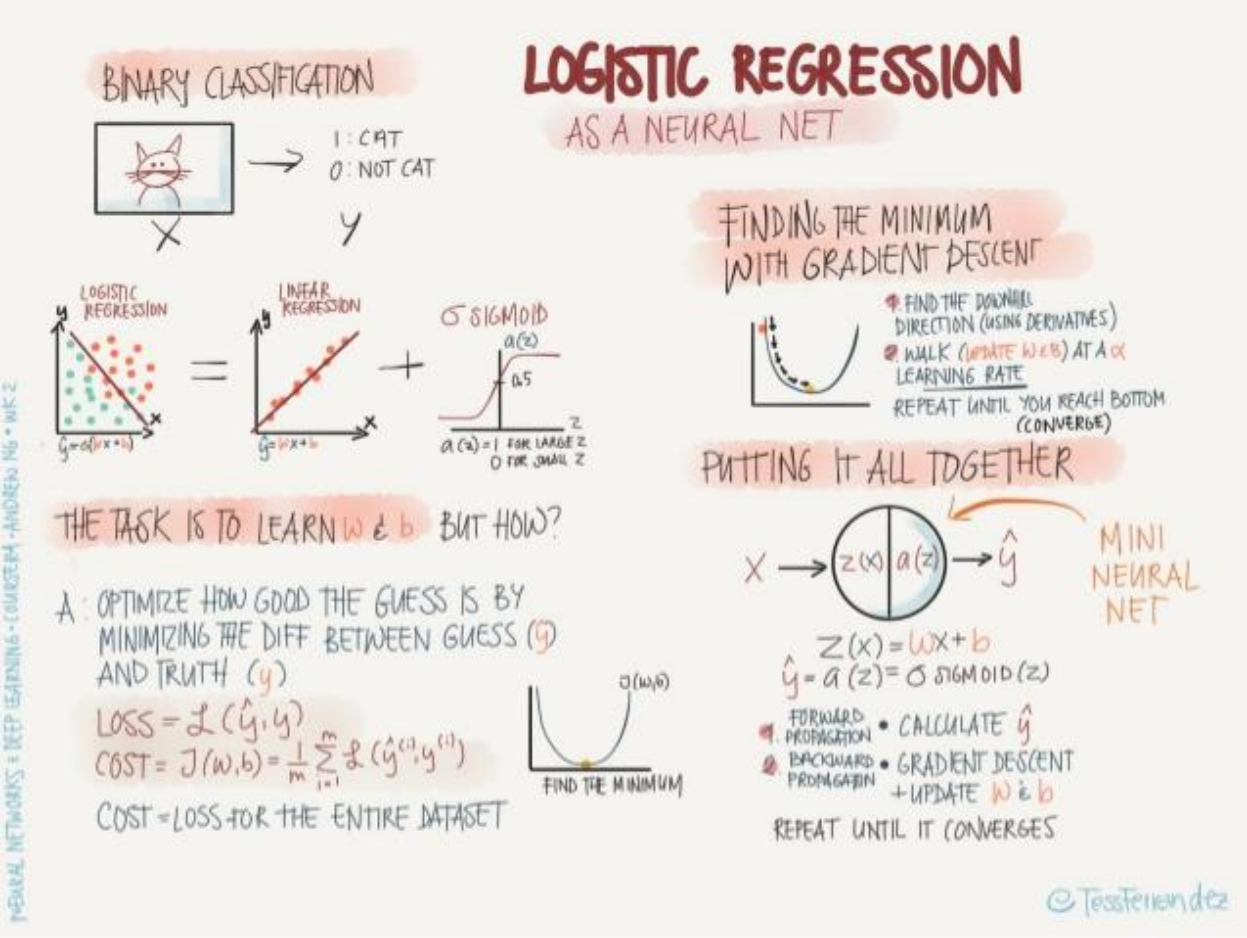


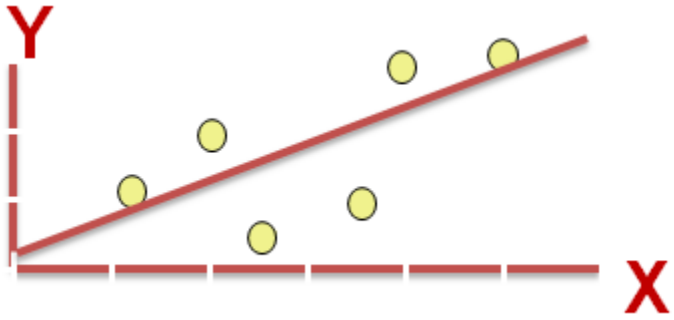
Table of contents

Neural Networks Basics

Learn to set up a machine learning problem with a neural network mindset. Learn to use vectorization to speed up your models.

Binary classification

- Mainly he is talking about how to do a logistic regression to make a binary classifier.



  -
  - Image taken from 3.bp.blogspot.com
- He talked about an example of knowing if the current image contains a cat or not.
- Here are some notations:
  - `M is the number of training vectors`
  - `Nx is the size of the input vector`
  - `Ny is the size of the output vector`
  - `X(1) is the first input vector`
  - `Y(1) is the first output vector`
  - `X = [x(1) x(2).. x(M)]`
  - `Y = (y(1) y(2).. y(M))`
- We will use python in this course.
- In NumPy we can make matrices and make operations on them in a fast and reliable time.

Logistic regression

- Algorithm is used for classification algorithm of 2 classes.

- Equations:
  - Simple equation: `y = wx + b`
  - If x is a vector: `y = w(transpose)x + b`
  - If we need y to be in between 0 and 1 (probability): `y = sigmoid(w(transpose)x + b)`
  - In some notations this might be used: `y = sigmoid(w(transpose)x)`
    - While `b` is `w0` of `w` and we add `x0 = 1`. but we won't use this notation in the course (Andrew said that the first notation is better).
- In binary classification `Y` has to be between `0` and `1`.
- In the last equation `w` is a vector of `Nx` and `b` is a real number

Logistic regression cost function

- First loss function would be the square root error: `L(y',y) = 1/2 (y' - y)^2`
  - But we won't use this notation because it leads us to optimization problem which is non convex, means it contains local optimum points.
- This is the function that we will use: `L(y',y) = - (y*log(y') + (1-y)*log(1-y'))`
- To explain the last function lets see:
  - if `y = 1` ==> `L(y',1) = -log(y')` ==> we want `y'` to be the largest ==> `y'` biggest value is 1
  - if `y = 0` ==> `L(y',0) = -log(1-y')` ==> we want `1-y'` to be the largest ==> `y'` to be smaller as possible because it can only has 1 value.
- Then the Cost function will be: `J(w,b) = (1/m) * Sum(L(y'[i],y[i]))`
- The loss function computes the error for a single training example; the cost function is the average of the loss functions of the entire training set.

Gradient Descent

- We want to predict `w` and `b` that minimize the cost function.
- Our cost function is convex.
- First we initialize `w` and `b` to 0,0 or initialize them to a random value in the convex function and then try to improve the values the reach minimum value.
- In Logistic regression people always use 0,0 instead of random.
- The gradient decent algorithm repeats: `w = w - alpha * dw` where alpha is the learning rate and `dw` is the derivative of `w` (Change to `w`) The derivative is also the slope of `w`
- Looks like greedy algorithms. the derivative give us the direction to improve our parameters.
- The actual equations we will implement:
  - `w = w - alpha * d(J(w,b) / dw)` (how much the function slopes in the w direction)
  - `b = b - alpha * d(J(w,b) / db)` (how much the function slopes in the d direction)

Derivatives

- We will talk about some of required calculus.
- You don't need to be a calculus geek to master deep learning but you'll need some skills from it.

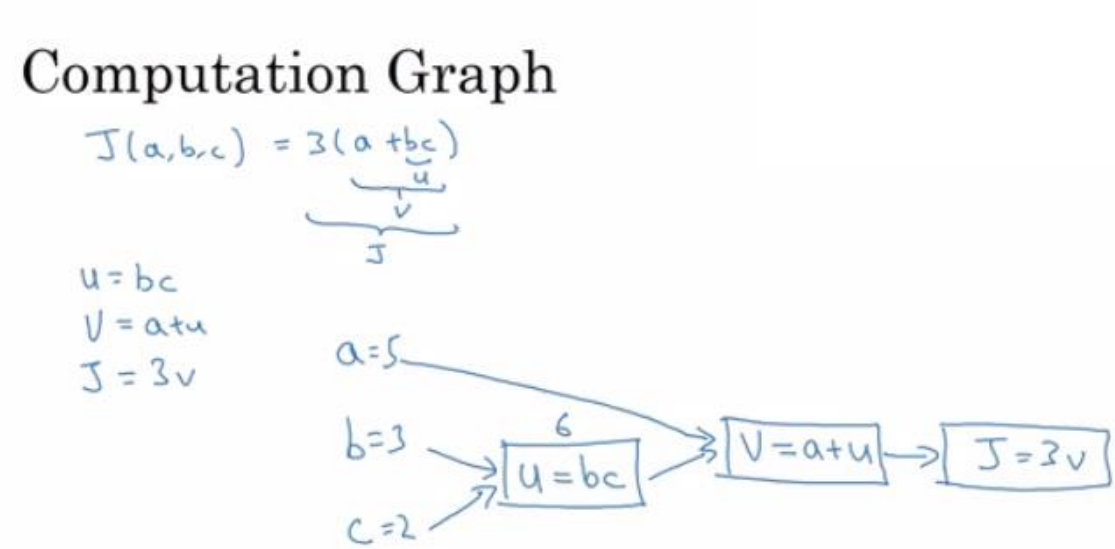
- Derivative of a linear line is its slope.
    - ex. `f(a) = 3a` `d(f(a))/d(a) = 3`
    - if `a = 2` then `f(a) = 6`
    - if we move a a little bit `a = 2.001` then `f(a) = 6.003` means that we multiplied the derivative (Slope) to the moved area and added it to the last result.

More Derivatives examples

- `f(a) = a^2` **==>** `d(f(a))/d(a) = 2a`
    - `a = 2` **==>** `f(a) = 4`
    - `a = 2.0001` **==>** `f(a) = 4.0004` approx.
- `f(a) = a^3` **==>** `d(f(a))/d(a) = 3a^2`
- `f(a) = log(a)` **==>** `d(f(a))/d(a) = 1/a`
- To conclude, Derivative is the slope and slope is different in different points in the function thats why the derivative is a function.

Computation graph

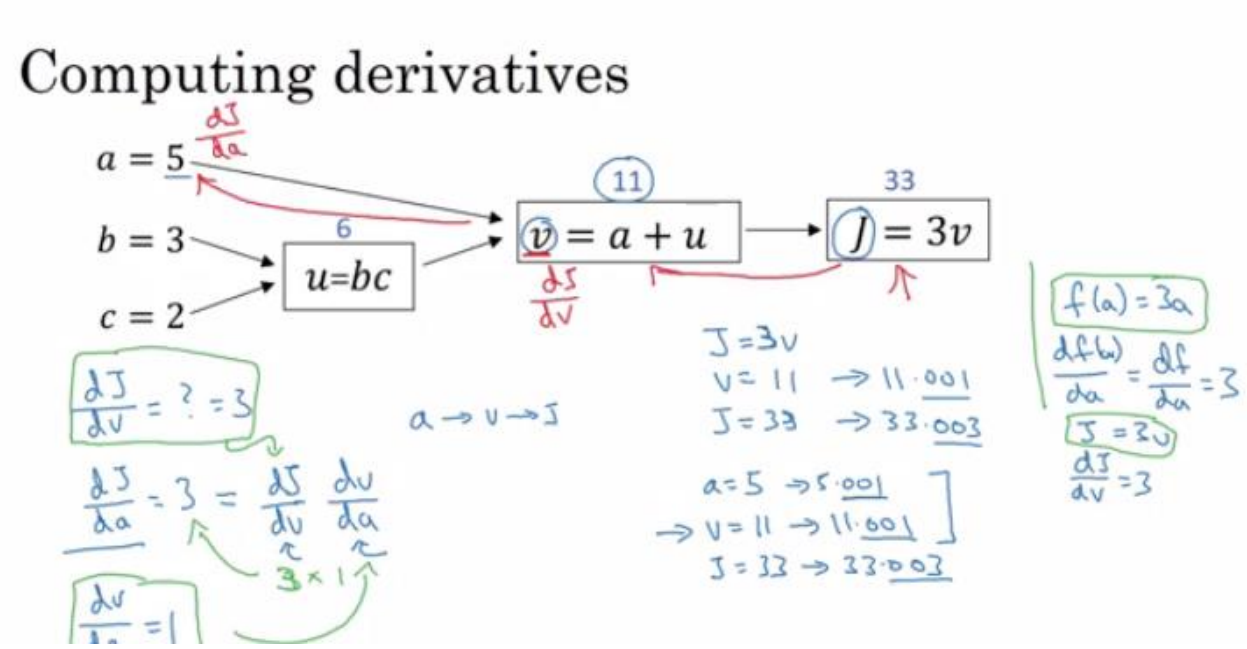
- Its a graph that organizes the computation from left to right.



    -

Derivatives with a Computation Graph

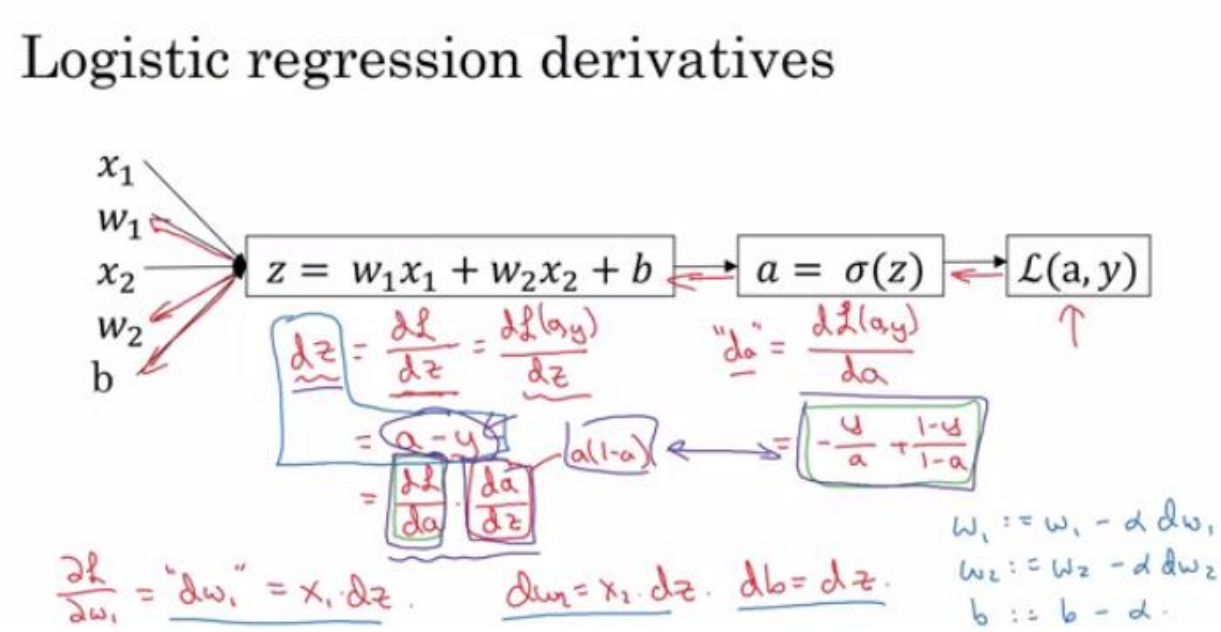
- Calculus chain rule says: If `x -> y -> z` (x effect y and y effects z) Then `d(z)/d(x) = d(z)/d(y) * d(y)/d(x)`
- The video illustrates a big example.



- We compute the derivatives on a graph from right to left and it will be a lot more easier.
- `dvar` means the derivatives of a final output variable with respect to various intermediate quantities.

Logistic Regression Gradient Descent

- In the video he discussed the derivatives of gradient decent example for one sample with two features `x1` and `x2`.



Gradient Descent on m Examples

- Lets say we have these variables:
- X1 Feature
- X2 Feature
- W1 Weight of the first feature.
- W2 Weight of the second feature.
- B Logistic Regression parameter.
- M Number of training examples
- Y(i) Expected output of i
- So we have:

```
X1  \
W1   \
X2   ===>  z(i) = X1W1 + X2W2 + B ===> a(i) = Sigmoid(z(i)) ===> l(a(i),Y(i)) = - (Y(i)*log(a(i)) + (1-Y(i))*log(1-a(i)))
Y2   /
B   /
```

- Then from right to left we will calculate derivations compared to the result:
- d(a) = d(l)/d(a) = -(y/a) + ((1-y)/(1-a))
- d(z) = d(l)/d(z) = a - y
- d(W1) = X1 * d(z)
- d(W2) = X2 * d(z)
- d(B) = d(z)
- From the above we can conclude the logistic regression pseudo code:

```
J = 0; dw1 = 0; dw2 =0; db = 0;                    # Devs.
w1 = 0; w2 = 0; b=0;                                                    # Weights
for i = 1 to m
        # Forward pass
        z(i) = W1*x1(i) + W2*x2(i) + b
        a(i) = Sigmoid(z(i))
        J += (Y(i)*log(a(i)) + (1-Y(i))*log(1-a(i)))

        # Backward pass
        dz(i) = a(i) - Y(i)
        dw1 += dz(i) * x1(i)
        dw2 += dz(i) * x2(i)
        db  += dz(i)
J /= m
dw1/= m
dw2/= m
db/= m

# Gradient descent
w1 = w1 - alpa * dw1
w2 = w2 - alpa * dw2
b = b - alpa * db
```

- The above code should run for some iterations to minimize error.
- So there will be two inner loops to implement the logistic regression.
- Vectorization is so important on deep learning to reduce loops. In the last code we can make the whole loop in one step using vectorization!

Vectorization

- Deep learning shines when the dataset are big. However for loops will make you wait a lot for a result. Thats why we need vectorization to get rid of some of our for loops.
- NumPy library (dot) function is using vectorization by default.
- The vectorization can be done on CPU or GPU thought the SIMD operation. But its faster on GPU.
- Whenever possible avoid for loops.
- Most of the NumPy library methods are vectorized version.

Vectorizing Logistic Regression

- We will implement Logistic Regression using one for loop then without any for loop.
- As an input we have a matrix `X` and its `[Nx, m]` and a matrix `Y` and its `[Ny, m]`.
- We will then compute at instance `[z1,z2...zm] = W' * X + [b,b,...b]`. This can be written in python as:
- ```
  Z = np.dot(W.T,X) + b    # Vectorization, then broadcasting, Z shape is (1, m)
  ```
- ```
  A = 1 / 1 + np.exp(-Z)   # Vectorization, A shape is (1, m)
  ```
- Vectorizing Logistic Regression's Gradient Output:
- ```
  dz = A - Y                  # Vectorization, dz shape is (1, m)
  ```
- ```
  dw = np.dot(X, dz.T) / m    # Vectorization, dw shape is (Nx, 1)
  ```
- ```
  db = dz.sum() / m           # Vectorization, dz shape is (1, 1)
  ```

Notes on Python and NumPy

- In NumPy, `obj.sum(axis = 0)` sums the columns while `obj.sum(axis = 1)` sums the rows.
- In NumPy, `obj.reshape(1,4)` changes the shape of the matrix by broadcasting the values.
- Reshape is cheap in calculations so put it everywhere you're not sure about the calculations.
- Broadcasting works when you do a matrix operation with matrices that doesn't match for the operation, in this case NumPy automatically makes the shapes ready for the operation by broadcasting the values.
- In general principle of broadcasting. If you have an (m,n) matrix and you add(+) or subtract(-) or multiply(*) or divide(/) with a (1,n) matrix, then this will copy it m times into an (m,n) matrix. The same with if you use those operations with a (m , 1) matrix, then this will copy it n times into (m, n) matrix. And then apply the addition, subtraction, and multiplication of division element wise.
- Some tricks to eliminate all the strange bugs in the code:
  - If you didn't specify the shape of a vector, it will take a shape of `(m,)` and the transpose operation won't work. You have to reshape it to `(m, 1)`
  - Try to not use the rank one matrix in ANN
  ```

    - Don't hesitate to use `assert(a.shape == (5,1))` to check if your matrix shape is the required one.
    - If you've found a rank one matrix try to run reshape on it.
- Jupyter / IPython notebooks are so useful library in python that makes it easy to integrate code and document at the same time. It runs in the browser and doesn't need an IDE to run.
    - To open Jupyter Notebook, open the command line and call: `jupyter-notebook` It should be installed to work.
- To Compute the derivative of Sigmoid:
- ```
  s = sigmoid(x)
  ds = s * (1 - s)        # derivative  using calculus
  ```
- To make an image of `(width,height,depth)` be a vector, use this:
- ```
  v = image.reshape(image.shape[0]*image.shape[1]*image.shape[2],1)
  #reshapes the image.
  ```
- Gradient descent converges faster after normalization of the input matrices.

General Notes

- The main steps for building a Neural Network are:
    - Define the model structure (such as number of input features and outputs)
    - Initialize the model's parameters.
    - Loop.
        - Calculate current loss (forward propagation)
        - Calculate current gradient (backward propagation)
        - Update parameters (gradient descent)
- Preprocessing the dataset is important.
- Tuning the learning rate (which is an example of a "hyperparameter") can make a big difference to the algorithm.
- kaggle.com is a good place for datasets and competitions.
- Pieter Abbeel is one of the best in deep reinforcement learning.

Week 2 Quiz - Neural Network Basics

1. What does a neuron compute?
    - ☐ A neuron computes an activation function followed by a linear function (z = Wx + b)
    - ☑ A neuron computes a linear function (z = Wx + b) followed by an activation function
    - ☐ 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

Note: 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"?

- $\mathcal{L}^{(i)}(\hat{y}^{(i)}, y^{(i)}) = \mid y^{(i)} - \hat{y}^{(i)} \mid$
- **(selected)** $\mathcal{L}^{(i)}(\hat{y}^{(i)}, y^{(i)}) = -(y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}))$
- $\mathcal{L}^{(i)}(\hat{y}^{(i)}, y^{(i)}) = max(0, y^{(i)} - \hat{y}^{(i)})$
- $\mathcal{L}^{(i)}(\hat{y}^{(i)}, y^{(i)}) = \mid y^{(i)} - \hat{y}^{(i)} \mid^2$

---

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))
- **(selected)** x = img.reshape((32*32*3,1))
- x = img.reshape((32*32,3))

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

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

What will be the shape of "c"?

- (●) c.shape = (2, 3)
- ( ) c.shape = (3, 2)
- ( ) c.shape = (2, 1)
- ( ) The computation cannot happen because the sizes don't match. It's going to be "Error"!

b (column vector) is copied 3 times so that it can be summed to each column of a. Therefore, `c.shape = (2, 3)`.

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

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

What will be the shape of "c"?

- c.shape = (4, 3)
- The computation cannot happen because the sizes don't match. It's going to be "Error"!
- c.shape = (3, 3)
- c.shape = (4,2)

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

`(n_x, m)`

Note: A stupid way to validate this is use the formula Z^(l) = W^(l)A^(l) when l = 1, then we have

- A^(1) = X
- X.shape = (n_x, m)
- Z^(1).shape = (n^(1), m)
- W^(1).shape = (n^(1), n_x)

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":

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

What is the shape of c?

`c.shape = (12288, 45)`, this is a simple matrix multiplication example.

8. Consider the following code snippet:

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

How do you vectorize this?

`c = a + b.T`

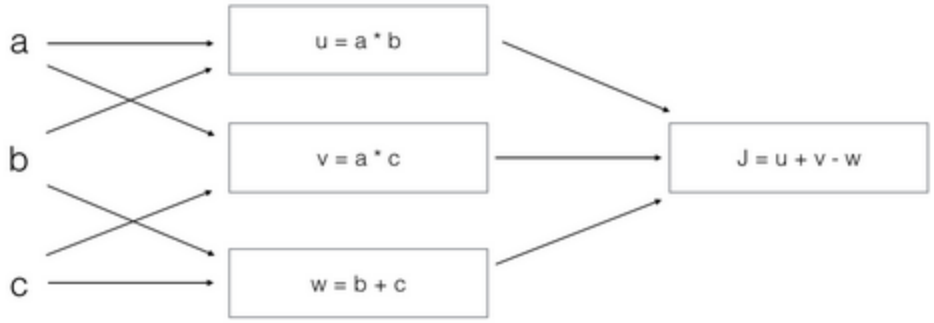
9. Consider the following code:

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

What will be c?

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

10. Consider the following computation graph.



What is the output J?

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

```
J = u + v - w
  = a * b + a * c - (b + c)
  = a * (b + c) - (b + c)
  = (a - 1) * (b + c)
```