Week 2 - Neural Networks Basics

Binary Classification

Given a pair (x, y) where $x \in R^{n_x}, y \in \{0, 1\}$. With m training examples you would have $\{(x^1, y^1), (x^2, y^2), \ldots (x^m, y^m)\}$. Stacking them together you would come up with the matrices X and Y below which are much easier to train in batches since there are many vectorized optimizations readily available. $X \in R^{n_x \times m}, Y \in R^{1 \times m}$

Logistic Regression

- Given x we want the $\hat{y} = P(y = 1|x)$
- $x \in R^{n_x}, 0 \geq \hat{y} \geq 1$
- Parameters: $w \in R^{n_x}, b \in R$
- Output $\hat{y} = \sigma(w^T x + b)$
- $z = w^T x + b$
- $\sigma(z) = \frac{1}{1+e^{-z}}$
- If z is large ($\lim_{z \to \infty}$) then $\sigma(z) \approx \frac{1}{1+0} = 1$
- If z is a large negative number ($\lim_{z \to -\infty}$) then $\sigma(z) \approx \frac{1}{1+\infty} \approx 0$

Logistic Regression Cost Function

- Given $\{(x^1, y^1), (x^2, y^2), \ldots (x^m, y^m)\}$, we want $\hat{y}^{(i)} \approx y^{(i)}$
- Squared Error: $L(\hat{y}, y) = \frac{(\hat{y}-y)^2}{2}$ might not be a good option because it's not convex
- $L(\hat{y}, y) = (y \log \hat{y} + (1 - y) \log(1 - \hat{y}))$
- If $y = 1 : L(\hat{y}, y) = -\log \hat{y}$ <- want $\log \hat{y}$ to be large meaning we want $\hat{y}$ large
- If $y = 0 : L(\hat{y}, y) = -\log(1 - haty)$ <- want $\log(1 - \hat{y})$ to be large meaning we want $\hat{y}$ small
- **Cost function**: $J(w, b) = \frac{1}{m} \sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$, in other words, the cost function is the average of the loss on the given examples
- Loss function applies to a single training example
- Cost function applies to all the examples

Gradient Descent

- We want to find w, b that minimize $J(w, b)$
- This loss function is convex, therefore pointing to a single minima
- Gradient Descent: $w := w - \alpha \frac{\partial J(w,b)}{\partial w}$ and $b := b - \alpha \frac{\partial J(w,b)}{\partial b}$ where α is the learning rate

Derivatives

- Intuition about derivatives: $f(a) = 3a$ the slope of $f(a)$ at $a = 2$ is 3 and at $a = 5$ is 3 as well. Therefore $\frac{df(a)}{da} = 3 = \frac{d}{da} f(a)$

- In a line the slope doesn't change, therefore the derivative is constant

More Derivative Examples

- $f(a) = a^2$ the slope of $f(a)$ at $a = 2$ is 4 and at $a = 5$ is 10. Therefore $\frac{df(a)}{da} = 2a$
- The slope is different for different values as t doesn't increases linearly.
- $f(a) = a^3$ has derivative as $\frac{df(a)}{da} = 3a^2$
- $f(a) = \ln(a)$ has derivative as $\frac{df(a)}{da} = \frac{1}{a}$
- The derivative means the slope of the function
- The slope of the function can be different on different points of the function

Computation Graph

- $J = 3(a + bc)$
- $u = bc$
- $v = a + u$
- $J = 3v$
- One step of backward propagation on a computation graph yields derivative of final output variable.

Derivatives with a Computation Graph

- $\frac{dJ}{dv} = 3$
- $\frac{dJ}{da} = \frac{dJ}{dv}\frac{dv}{da} = 3 * 1 = 3$ (chain rule)
- $\frac{dJ}{du} = \frac{dJ}{dv}\frac{dv}{du} = 3 * 1 = 3$ (chain rule)
- $\frac{dJ}{db} = \frac{dJ}{du}\frac{du}{db} = 3 * 2 = 6$ (chain rule)
- $\frac{dJ}{db} = \frac{dJ}{du}\frac{du}{dc} = 3 * 3 = 9$ (chain rule)
- The best way is to flow from the right to left
- In this class, what does the coding convention dvar represent?
  - ☐ The derivative of any variable used in the code.
  - ☐ The derivative of input variables with respect to various intermediate quantities.
  - ☑ The derivative of a final output variable with respect to various intermediate quantities.

Logistic Regression Gradient Descent

- Suppose we only have two features x_1 and x_2
- Using the backward propagation on the formulas of the logistic regression in the computational graph we can update the correspondent changes in the weights and the bias
- In this video, what is the simplified formula for the derivative of the loss with respect to z? A: a - y

Gradient Descent on m examples

- Cost function is the mean of the loss function on all the m examples
- Let $J = 0, dw_1 = 0, dw_2 = 0, db = 0$
- For $i = 1$ to m:
  - $z^{(i)} = w^T x^{(i)} + b$

  - $a^{(i)} = \sigma(z^{(i)})$
  - $J += [y^{(i)} \log a^{(i)} + (1 - y^{(i)}) \log(1 - a^{(i)})]$
  - $dz^{(i)} += a^{(i)} - y^{(i)}$
  - $dw_1 += x_1^{(i)} d_z^{(i)}$
  - $dw_2 += x_2^{(i)} d_z^{(i)}$
  - $db += d_z^{(i)}$
- $J /= m$
- $dw_1 /= m$
- $dw_2 /= m$
- $db /= m$
- $w_1 := w_1 - \alpha dw_1$
- $w_2 := w_2 - \alpha dw_2$
- $b := b - \alpha db$
- The derivatives are being calculated cumulatively only to be divided by m in the end of the loop therefore calculating the mean
- Using explicit for loops are bad for parallelization. Since **vectorization** has appeared for loops are not a guideline anymore.

Vectorization

> *Vectorization is basically the art of getting rid of explicit folders in your code. – Andrew Ng*

- What is vectorization? Is the ability to remove the for loops from your code and implement it mostly with matrix (or tensor) operations

```
import numpy as np
a = np.array([1, 2, 3, 4])
print(a)
#Vectorized
import time
a = np.random.rand(1000000000)
b = np.random.rand(1000000000)
tic = time.time()
c = np.dot(a, b)
toc = time.time()
print(c)
print("Vectorized version:" + str(1000*(toc-tic)) + "ms") #about 1.5ms
# Non-vectorized
c = 0
tic = time.time()
for i in range(1000000000):
        c += a[i]*b[i]
toc = time.time()
print(c)
print("For loop:" + str(1000*(toc-tic)) + "ms") #about 400~500ms
```

- The rule of thumb to remember is whenever possible, avoid using explicit four loops
- Vectorization can be done without a GPU.

More vectorization examples

- Use built-in functions and avoid explicit for loops (This guideline also applies largely to numpy and pandas)
- Let $J = 0, db = 0$
- $dw = np.zeros((n_x, 1))$
- For $i = 1$ to m:
  - $z^{(i)} = w^T x^{(i)} + b$
  - $a^{(i)} = \sigma(z^{(i)})$
  - $J += [y^{(i)} \log a^{(i)} + (1 - y^{(i)}) \log(1 - a^{(i)})]$
  - $dz^{(i)} += a^{(i)} - y^{(i)}$
  - $dw += x^{(i)} d_z^{(i)}$
  - $db += d_z^{(i)}$
- $J /= m$
- $dw /= m$
- $db /= m$
- $w_1 := w_1 - \alpha dw_1$
- $w_2 := w_2 - \alpha dw_2$
- $b := b - \alpha db$

Vectorizing Logistic Regression

- Finally uses the X and Y matrices discussed in the beginning of the lesson
- Z = np.dot(w.T, X) + b
- A = σ(Z)

Vectorizing Logistic Regression's Gradient Output

- Find a $dZ = [dz^{(1)}, dz^{(2)}, \dots, dz^{(m)}]$
- $A = [a^{(1)}, a^{(2)}, \dots, a^{(m)}]$
- $Y = [y^{(1)}, y^{(2)}, \dots, y^{(m)}]$
- $dZ = A - Y = [a^{(1)} - y^{(1)}, a^{(2)} - y^{(2)}, \dots, a^{(m)} - y^{(m)}]$
- $db = \frac{1}{m} \sum_{i=1}^{m} dz^{(i)} = \frac{1}{m} np.sum(dZ)$
- $dw = \frac{1}{m} X dZ^T$

- Now our algorithm looks like something:
- Forward pass
- Z = np.dot(w.T, X) + b
- A = σ(Z)
- Backpropagation pass
- $dZ = A - Y$
- $dw = \frac{1}{m} X dZ^T$
- $db = \frac{1}{m} np.sum(dZ)$
- Weights update pass
- $w := w - \alpha dw$

- $b := b - \alpha db$

Broadcasting in Python

```
import numpy as np
A = np.array([
        [56.0, 0.0, 4.4, 68.0],
        [1.2, 104.0, 52.0, 0.0],
        [1.8, 135.0, 99.0, 0.9]
        ])
print(A)
cal = A.sum(axis=0)
print(cal)
percentage = 100*A/cal.reshape(1, 4)
print(percentage)
```

- Given matrix (m,n) a given operation and a factor which is (1, n) or (m, 1) the factor is expanded to (m, n) by copy and the operation is executed element-wise
- Given a matrix (m, 1) (or a vector) and a numeric fator the factor is expanded to (m, 1) by copy and the operation is executed element-wise

A note on python/numpy vectors

```
        import numpy as np
        a = np.random.rand(5)
        a.shape # (5, ), rank 1 array
        a.T # == a which a.T.shape = (5,)
        np.dot(a, a.T)# returns a single number

        a = np.random.rand(5, 1)
        a.shape # (5, 1)
        a.T # != a which a.T.shape = (1,5)
        np.dot(a, a.T)# returns a matrix

        #(5, 1) = column vector
        #(1, 5) = row vector

        # To avoid you can use assert
        assert(a.shape == (5,1))
        #or reshape
        a.reshape(1, 5)
```

Quiz

1. What does a neuron compute?
   - ☐ A neuron computes an activation function followed by a linear function (z = 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
   - ☐ A neuron computes a function g that scales the input x linearly (Wx + b)

2. Which of these is the “Logistic Loss”?
   - (y^(i),y(i))=|y(i)−y^(i)|
   - (y^(i),y(i))=max(0,y(i)−y^(i))
   - (y^(i),y(i))=|y(i)−y^(i)|2
   - X(y^(i),y(i))=−(y(i)log(y^(i))+(1−y(i))log(1−y^(i)))
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))
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 = (3, 2)
   - ☑ c.shape = (2, 3)
   - ☐ c.shape = (2, 1)
   - ☐ 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”:

```
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)
   - ☐ c.shape = (4,2)
   - ☐ c.shape = (3, 3)
   - ☑ The computation cannot happen because the sizes don’t match. It’s going to be “Error”!
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?
   - ☑ (n_x, m)
   - ☐ (1,m)
   - ☐ (m,n_x)
   - ☐ (m,1)
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?

   - ☐ Te computation cannot happen because the sizes don’t match. It’s going to be “Error”!
   - ☐ c.shape = (12288, 45)
   - ☑ c.shape = (12288, 150)

- ☐ c.shape = (150,150)

8. Consider the following code snippet:

```
import numpy as np # a.shape = (3,4) and 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.T + b.T
- ☐ c = a + b
- ☑ c = a + b.T
- ☐ c = a.T + b

9. Consider the following code:

```
a = np.random.randn(3, 3)
b = np.random.randn(3, 1)
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)
- ☐ 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)
    - ☐ J = a*b + b*c + a*c
    - ☐ J = (b - 1) * (c + a)