

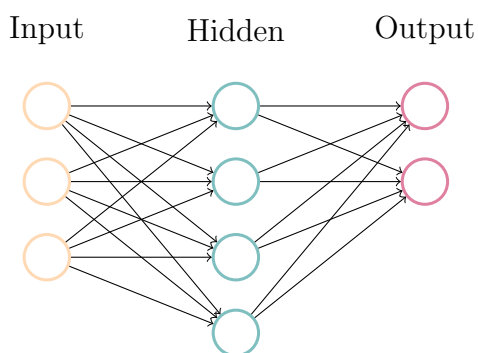
# 1 Week1

Introduction to Deep Learning.

## 1.1 What is a Neural Network?

In machine learning, a neural network (also artificial neural network or neural net, abbreviated ANN or NN) is a model inspired by the neuronal organization found in the biological neural networks in animal brains. (from wikipedia)

An artificial neural network is an interconnected group of nodes, inspired by a simplification of neurons in a brain. Here, each circular node represents an artificial neuron and an arrow represents a connection from the output of one artificial neuron to the input of another.



## 1.2 Supervised Learning with Neural Networks

Standard NN, Convolutional NN, Recurrent NN.

Structured Data vs. Unstructured Data.

## 1.3 Why is Deep Learning taking off?

Scale drives deep learning progress.

## 1.4 About this Course

1. Neural Networks and Deep Learning
2. Improving Deep Neural Networks: Hyperparameter tuning, Regularization and Optimization
3. Structuring your Machine Learning project
4. Convolutional Neural Networks
5. Natural Language Processing: Building sequence models

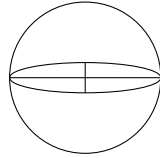
## 1.5 Outline of this Course

- Week1. Introduction
- Week2. Basics of Neural Network programming
- Week3. One hidden layer Neural Networks
- Week4. Deep Neural Networks

## 2 Week2

Basics of Neural Network Programming

How do I write an equation in L<sup>A</sup>T<sub>E</sub>X?



In 1902, Einstein created this equation:  $E = mc^2$

And Newton came up with this one:  $\sum F = ma$

$$5 + 5 = 10 \tag{1}$$

$$\begin{aligned} A &= \frac{5\pi r^2}{2} \\ A &= \frac{1}{2}\pi r^2 \end{aligned} \tag{2}$$

## 2.1 Neural Network Notations

### General comments:

superscript  $(i)$  will denote the  $i^{th}$  training example.

superscript  $[l]$  will denote the  $l^{th}$  layer.

### Sizes:

- $m$ : number of examples in the dataset
- $n_x$ : input size
- $n_y$ : output size
- $n_h^{[l]}$ : number of hidden units of the  $l^{th}$  layer.  
In a for loop, it is possible to denote  $n_x = n_h^{[0]}$  and  $n_y = n_h^{[numberoflayer+1]}$
- $L$ : number of layers in the network

### Objects:

- $X \in \mathbb{R}^{n_x \times m}$  is the input matrix
- $x^{(i)} \in \mathbb{R}^{n_x}$  is the  $i^{th}$  example represented as a column vector
- $Y \in \mathbb{R}^{n_y \times m}$  is the label matrix
- $y^{(i)} \in \mathbb{R}^{n_y}$  is the output label for the  $i^{th}$  example
- $W^{[l]} \in \mathbb{R}^{\# \text{ of units in next layer} \times \# \text{ of units in the previous layer}}$  is the weight matrix, superscript  $[l]$  indicates the layer
- 

### Common forward propagation equation examples:

- 
- 

### Examples of cost functions:

- 
-

## 2.2 Binary Classification

Use matrix without using for loops.

Computation using Forward propagation and Backward propagation.

Logistic regression is an algorithm for binary classification.

m training examples  $\{(X^{(1)}, Y^{(1)}), (X^{(2)}, Y^{(2)}), \dots, (X^{(m)}, Y^{(m)})\}$   
where  $X^{(i)} \in \mathbb{R}^{n_x}$  and  $Y^{(i)} \in \{0, 1\}$  for  $i \in [1, m]$

$$X = \begin{bmatrix} \vdots & \vdots & \vdots & \vdots \\ X^{(1)} & X^{(2)} & \dots & X^{(m)} \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix} \in \mathbb{R}^{n_x \times m}$$

$$X.shape = (n_x, m)$$

$$Y = [Y^{(1)}, Y^{(2)}, \dots, Y^{(m)}] \in \mathbb{R}^{1 \times m}$$

$$Y.shape = (1, m)$$

## 2.3 Logistic Regression

Given  $x$ , want  $\hat{y} = P(y = 1|x)$  where  $x \in \mathbb{R}^{n_x}$

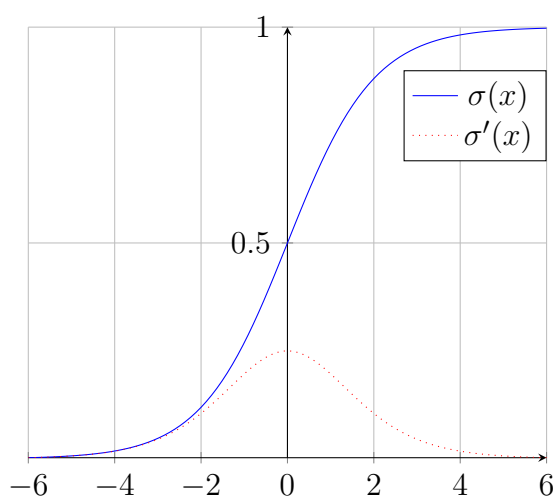
$$(\hat{y}^{(i)} = \sigma(w^T x^{(i)} + b) = \sigma(z^{(i)}) \text{ for } i = 1, 2, \dots, m_{train})$$

$$P(y|x) = P(y = 1|x) + P(y = 0|x) = \hat{y} + (1 - \hat{y}) = \hat{y}^y (1 - \hat{y})^{(1-y)}$$

Parameters:  $w \in \mathbb{R}^{n_x}$  a  $n_x$  dimensional vector,  $b \in \mathbb{R}$  a real number.

$$\text{Output } \hat{y} = \sigma(w^T x + b) = \sigma(z) = \sigma(z) = \frac{1}{1+e^{-z}}$$

Drawing a sigmoid function and its derivative in tikz



## 2.4 Logistic Regression Cost Function

To train the parameter  $w$  and  $b$  of a Logistic Regression Model, we need a cost function.

$$\hat{y} = \sigma(w^T X + b) \text{ where } \sigma(z) = \frac{1}{1+e^{-z}}$$

$$\hat{y}^{(i)} = \sigma(w^T X^{(i)} + b) \text{ where } \sigma(z^{(i)}) = \frac{1}{1+e^{-z^{(i)}}}$$

Given  $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$ , want  $\hat{y}^{(i)} \approx y^{(i)}$ .

Loss(error) function (for a **single** training Example):

$$\boxed{\mathcal{L}(\hat{y}, y)} = -(y \log \hat{y} + (1 - y) \log (1 - \hat{y}))$$

Cost function (for the **entire** training Examples;  $i = 1, 2, \dots, m$ ):

$$\boxed{\mathcal{J}(w, b)} = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m} \sum_{i=1}^m [y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log (1 - \hat{y}^{(i)})]$$

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.

In training logistic regression model, we will try to find  $w$  and  $b$  such that they minimize the Cost function  $\mathcal{J}(w, b)$ .

Logistic Regression can be seen as a very small Neural Network.

### Explanation of Logistic Regression's Cost Function:

$\log P(y|x) = y \log \hat{y} + (1 - y) \log(1 - \hat{y}) = -\mathcal{L}(\hat{y}, y)$  tells that in order to make probability large, we have to minimize the Loss function.

$$\begin{aligned}\log P(\text{labels in training set}) &= \log \prod_{i=1}^m P(y^{(i)}|x^{(i)}) \\ &= \sum_{i=1}^m \log P(y^{(i)}|x^{(i)}) = - \sum_{i=1}^m \mathcal{L}(\hat{y}^{(i)}, y^{(i)})\end{aligned}$$

(Maximum Likelihood Estimation: choose parameters to maximize it.)

$$\text{Cost: } \mathcal{J}(w, b) = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

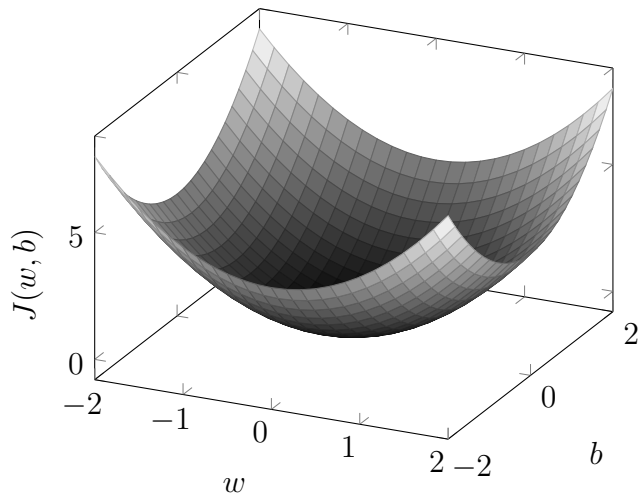
Minimize Cost in order to carry out the Maximum Likelihood Estimation with the logistic regression model under the assumption that our training examples were iid (independently distributed).



## 2.5 Gradient Descent

$$\mathcal{J}(w, b) = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m} \sum_{i=1}^m [y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log (1 - \hat{y}^{(i)})]$$

Want to find  $w$  and  $b$  that minimize the Cost function  $\mathcal{J}(w, b)$ .



$\mathcal{J}(w, b)$  is a convex function with a single local optimum.

No matter where you initialize the point, you should get to the same point (Global optimum).

Repeat:

$$w := w - \alpha \frac{\partial \mathcal{J}(w, b)}{\partial w} = w - \alpha dw$$

$$b := b - \alpha \frac{\partial \mathcal{J}(w, b)}{\partial b} = b - \alpha db$$

where  $\alpha$  is the learning rate.

## 2.6 Derivatives

derivatives; slope

Given  $f(a) = 3a$

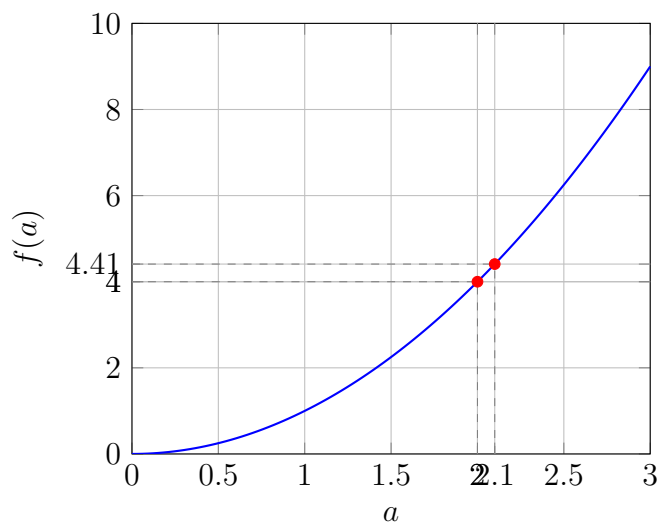
$\epsilon = .001, a = 2 + \epsilon$

$$\frac{f(a) - f(a + \epsilon)}{\epsilon}$$

make  $\epsilon$  close to zero  $\rightarrow$  derivatives.

$$\frac{df(a)}{da} = \frac{d}{da} f(a)$$

## 2.7 More Derivative Examples



$$a = 2, f(a) = 4$$

$$a = 2.001, f(a) = 4.004001$$

$$\frac{d}{da} f(a) = 4, \text{ when } a = 2$$

$$\frac{d}{da} f(a) = 10, \text{ when } a = 5$$

$$\frac{d}{da} f(a) = \frac{d}{da} a^2 = 2a$$

Given a nudge  $\epsilon = 0.001$  to  $a$ , the  $f(a)$  goes up  $2 * a$ .

## 2.8 Computation Graph

- Forward propagation step(forward pulse) : compute output of the network.
- Backward pulse: compute the gradients or derivatives.

$$J(a, b, c) = 3(a + bc)$$

$$u = bc$$

$$v = a + u$$

$$J = 3v$$

In order to compute derivatives, you go backward propagation.

One step of backward propagation on a computation graph yields derivative of final output variable.

## 2.9 Computing derivatives.

$$u = bc$$

$$v = a + u$$

$$J = 3v$$

Given  $a = 5, b = 3, c = 2$ , then  $v = 11, J = 33$

We want to see how much  $J$  changes if we change the values of  $a, b, c, u, v$  for  $J = 3v$

If we increase  $v$  to 11.001, then  $J = 33.003$ .

$$\frac{dJ}{dv} = 3$$

$$a = 5 \rightarrow a = 5.001$$

$$v = 11 \rightarrow a = 11.001$$

$$J = 33 \rightarrow a = 33.003$$

By Chain Rule:

$$\frac{dJ}{da} = 3 = \frac{dJ}{dv} \frac{dv}{da} = 3 \times 1$$

$$\frac{dFinalOutputVar}{dvar} \text{ where } var \text{ can be } a, b, c, \dots$$

We can simply denote  $\frac{dJ}{dv} = dv$ ,  $\frac{dJ}{da} = da$ , etc with respect to  $J$ .

Similarly,  $\frac{dJ}{du} = \frac{dJ}{dv} \frac{dv}{du} = 3 \times 1$

$$\frac{dJ}{db} = \frac{dJ}{du} \frac{du}{db} = 3 \times 2 = 6, \text{ where } u = bc = 2b, \text{ and } \frac{du}{db} = 2, \text{ given } a = 5, b = 3, c = 2.$$

$$\frac{dJ}{dc} = \frac{dJ}{du} \frac{du}{dc} = 3 \times 3 = 9, \text{ where } u = bc = 3c, \text{ and } \frac{du}{dc} = 3, \text{ given } a = 5, b = 3, c = 2.$$

$$\frac{dJ}{da} = 3$$

$$\frac{dJ}{du} = 3$$

$$\frac{dJ}{db} = 6$$

$$\frac{dJ}{dc} = 9$$

The coding convention  $dvar$  represents: The derivative of a final output variable with respect to various intermediate quantities.

## 2.10 Logistic Regression Gradient Descent

Compute derivatives using Computation Graph(a bit overkill?) to implement/derive gradient descent for Logistic Regression.

Logistic regression recap (for a single  $i^{th}$  training example,  $x^{(i)}$ ):

$$z^{(i)} = w^T x^{(i)} + b$$

$$\hat{y}^{(i)} = a^{(i)} = \sigma(z^{(i)})$$

$$\mathcal{L}(a^{(i)}, y^{(i)}) = -[y^{(i)} \log(a^{(i)}) + (1 - y^{(i)}) \log(1 - a^{(i)})]$$

$$\mathcal{J}(w, b) = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(a^{(i)}, y^{(i)}) = -\frac{1}{m} \sum_{i=1}^m [y^{(i)} \log(a^{(i)}) + (1 - y^{(i)}) \log(1 - a^{(i)})]$$

Computation graph, given:  $x_1, w_1, x_2, w_2, b$

$$\boxed{z^{(i)} = w_1 x_1 + w_2 x_2 + b} \rightarrow \boxed{\hat{y}^{(i)} = a^{(i)} = \sigma(z^{(i)})} \rightarrow \boxed{\mathcal{L}(a^{(i)}, y^{(i)})}$$

$$, \text{ where } x^{(i)} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Modify  $w$  and  $b$  to reduce the loss  $\mathcal{L}(a^{(i)}, y^{(i)})$ . In order to find such  $w$  and  $b$ , we **compute the derivatives with respect to  $\mathcal{L}(a^{(i)}, y^{(i)})$** .

$$\boxed{da^{(i)}} = \frac{d\mathcal{L}(a^{(i)}, y^{(i)})}{da} = -\frac{d}{da}(y^{(i)} \log(a^{(i)}) + (1 - y^{(i)}) \log(1 - a^{(i)})) = -\frac{y^{(i)}}{a^{(i)}} + \frac{1 - y^{(i)}}{1 - a^{(i)}}$$

$$\boxed{dz^{(i)}} = \frac{d\mathcal{L}(a^{(i)}, y^{(i)})}{dz} = \frac{d\mathcal{L}(a^{(i)}, y^{(i)})}{da} \frac{da}{dz} = \left(-\frac{y^{(i)}}{a^{(i)}} + \frac{1 - y^{(i)}}{1 - a^{(i)}}\right) \frac{da}{dz} = a^{(i)} - y^{(i)}$$

$$, \text{ where } \frac{da}{dz} = \frac{d}{dz} \left( \frac{1}{1 + e^{-z^{(i)}}} \right) = \frac{e^{-z^{(i)}}}{(1 + e^{-z^{(i)}})^2} = \frac{1}{1 + e^{-z^{(i)}}} \left( 1 - \frac{1}{1 + e^{-z^{(i)}}} \right) = a^{(i)}(1 - a^{(i)})$$

**We first do Forward Propagation with initial (randomly initialized)  $w, b$ . Given incorrect prediction of  $y$ , do Backward Propagation using gradient descent and update the paramers  $w, b$ .**

Go backward to compute :

$$\boxed{dw_1} = \frac{\partial \mathcal{L}}{\partial w_1} = \frac{d\mathcal{L}(a,y)}{dw_1} = \frac{d\mathcal{L}(a,y)}{dz} \frac{dz}{dw_1} = x_1 dz$$

$$\boxed{dw_2} = \frac{\partial \mathcal{L}}{\partial w_2} = \frac{d\mathcal{L}(a,y)}{dw_2} = \frac{d\mathcal{L}(a,y)}{dz} \frac{dz}{dw_2} = x_2 dz$$

$$\boxed{db} = \frac{\partial \mathcal{L}}{\partial b} = \frac{d\mathcal{L}(a,y)}{db} = \frac{d\mathcal{L}(a,y)}{dz} \frac{dz}{db} = dz$$

Compute  $dz$  to compute  $dw_1$ ,  $dw_2$ ,  $db$  and update with gradient descent:

$$w_1 := w_1 - \alpha dw_1$$

$$w_2 := w_2 - \alpha dw_2$$

$$b := b - \alpha db$$

## 2.11 Gradient Descent on $m$ Examples

$$\mathcal{J}(w, b) = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(a^{(i)}, y^{(i)})$$

$$, \text{ where } a^{(i)} = \hat{y}^{(i)} = \sigma(z^{(i)}) = \sigma(w^T x^{(i)} + b)$$

$$dw_1 = \frac{\partial}{\partial w_1} \mathcal{J}(w, b) = \frac{1}{m} \sum_{i=1}^m \frac{\partial}{\partial w_1} \mathcal{L}(a^{(i)}, y^{(i)}) = \frac{1}{m} \sum_{i=1}^m dw_1^{(i)} = \frac{1}{m} \sum_{i=1}^m x_1^{(i)} dz^{(i)}$$

$$dw_2 = \frac{\partial}{\partial w_2} \mathcal{J}(w, b) = \frac{1}{m} \sum_{i=1}^m \frac{\partial}{\partial w_2} \mathcal{L}(a^{(i)}, y^{(i)}) = \frac{1}{m} \sum_{i=1}^m dw_2^{(i)} = \frac{1}{m} \sum_{i=1}^m x_2^{(i)} dz^{(i)}$$

$$db = \frac{\partial}{\partial b} \mathcal{J}(w, b) = \frac{1}{m} \sum_{i=1}^m \frac{\partial}{\partial b} \mathcal{L}(a^{(i)}, y^{(i)}) = \frac{1}{m} \sum_{i=1}^m db^{(i)} = \frac{1}{m} \sum_{i=1}^m dz^{(i)}$$

Logistic regression on  $m$  examples:

$$J = 0; dw_1 = 0; dw_2 = 0; db = 0$$

for  $i = 1$  to  $m$ :

$$z^{(i)} = w^T x^{(i)} + b$$

$$a^{(i)} = \sigma(z^{(i)})$$

$$\mathcal{J} += -[y^{(i)} \log(a^{(i)}) + (1 - y^{(i)}) \log(1 - a^{(i)})]$$

$$dz^{(i)} = a^{(i)} - y^{(i)}$$

\* The value of  $dw_1$ ,  $dw_2$ ,  $db$  in the code is cumulative:

$$dw_1 += x_1^{(i)} dz^{(i)}$$

$$dw_2 += x_2^{(i)} dz^{(i)}$$

...

$$dw_{n_x} += x_{n_x}^{(i)} dz^{(i)}$$

$$db += dz^{(i)}$$

$$J /= m, dw_1 /= m, dw_2 /= m, db /= m$$

After the calculations for all  $m$  examples, we update(1 step of gradient descent):

$$w_1 := w_1 - \alpha dw_1$$

$$w_2 := w_2 - \alpha dw_2$$

...

$$w_{n_x} := w_{n_x} - \alpha dw_{n_x}$$

$$b := b - \alpha db$$

We have to do multiple steps of above gradient descent.

It has two weaknesses: two for-loops (one for  $m$  training examples and another for features:

$w_{(i)}$  where  $i$  can be big.) → Vectorization!

In the context of logistic regression, forward and backward propagation are used to learn the model parameters. Here's a brief overview:

**Forward Propagation:** In forward propagation, you pass the input data through the model to get the prediction. For a batch of  $m$  training examples, each of dimension  $n$ , the forward propagation is:

1. Compute the linear combination of the input features and weights:  $z = w^T X + b$ .
2. Apply the sigmoid function to  $z$  to get the predicted probabilities:  $a = \sigma(z)$ .
3. Compute the cost function  $\mathcal{J}$ , which is the average of the loss function  $\mathcal{L}$  over all training examples.

**Backward Propagation:** Backward propagation is used to update the model parameters (weights and bias) using the gradient of the cost function. It involves the following steps:

1. Compute the derivative of the cost function  $\mathcal{J}$  with respect to the predicted probabilities.
2. Compute the derivative of the predicted probabilities with respect to  $z$ .
3. Compute the derivative of  $z$  with respect to the weights and bias.
4. Update the weights and bias using the computed derivatives. The goal of these steps is to minimize the cost function  $\mathcal{J}$  by adjusting the model parameters. This process is **repeated for several iterations** or until the model's performance is satisfactory.



## 2.12 Vectorization

What is vectorization?

$$z = w^T + b, \text{ where } w \in \mathcal{R}^{n_x} \text{ and } x \in \mathcal{R}^{n_x}$$

---

```
import numpy as np
```

```
z = np.dot(w,x) + b
```

---

In Jupiter notebook:

---

```
import time
```

```
# 1. Vectorized version
```

```
a = np.random.rand(1000000)
```

```
b = np.random.rand(1000000)
```

```
tic = time.time()
```

```
c = np.dot(a,b)
```

```
toc = time.time()
```

```
print("1. Vectorized version:" + str(1000*(toc-tic))+ "ms")
```

```
# 2. For loop
```

```
c = 0
```

```
tic = time.time()
```

```
for i in range(1000000):
```

```
    c += a[i]*b[i]
```

```
toc = time.time()
```

```
print("2. For loop:" + str(1000*(toc-tic))+ "ms")
```

---

CPU and GPU has SIMD (single instruction multiple data). If you use built-in functions such as numpy's. It enables numpy to take better advantage of parallelization.

## 2.13 More Vectorization Examples

Whenever possible, avoid explicit for-loops

$$u = Av$$

$$u_i = \sum_j A_{ij}v_j \text{ for } i = 1, \dots, n$$

1. Non-vectorized:

---

```
import numpy as np

u = np.zeros((n,1))
for i in range(n):
    for j in range(m):
        u[i] += A[i][j]*v[j]
```

---

2. Vectorized:

---

```
import numpy as np

u = np.dot(A,v)
```

---

Vectors and matrix valued functions.

say you need to apply the exponential operation on every element of a matrix/vector.

$$v = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix}$$
$$u = \begin{bmatrix} e^{v_1} \\ \vdots \\ e^{v_n} \end{bmatrix}$$

---

```
import numpy as np
u = np.zeros((n,1))

# 1. for-loop
for i in range(n):
    u[i] = math.exp(v[i])

# 2. Vectorized
u = np.exp(v)
u = np.log(v)
u = np.abs(v) # absolute value
u = np.maximum(v,0)
u = v**2
u = 1/v
```

---

Logistic regression derivatives

$J = 0; dw_1 = 0; dw_2 = 0; db = 0$

for  $i = 1$  to  $m$ :

$$z^{(i)} = w^T x^{(i)} + b$$

$$a^{(i)} = \sigma(z^{(i)})$$

$$\mathcal{J}+ = -[y^{(i)} \log(a^{(i)}) + (1 - y^{(i)}) \log(1 - a^{(i)})]$$

$$dz^{(i)} = a^{(i)} - y^{(i)}$$

\* The value of  $dw_1, dw_2, db$  in the code is cumulative:

$$dw_1+ = x_1^{(i)} dz^{(i)}$$

$$dw_2+ = x_2^{(i)} dz^{(i)}$$

...

$$dw_{n_x}+ = x_{n_x}^{(i)} dz^{(i)}$$

$$dw_2+ = x_2^{(i)} dz^{(i)}$$

$$db+ = dz^{(i)}$$

$J/ = m, dw_1/ = m, dw_2/ = m, \dots, dw_{n_x}/ = m, db/ = m$

## 2.14 Vectorizing Logistic Regression

$$z^{(i)} = w^T x^{(i)} + b$$

$$a^{(i)} = \sigma(z^{(i)}) \text{ for } i = 1, \dots, m$$

$$X = \begin{bmatrix} \vdots & \vdots & \dots & \vdots \\ x^{(1)} & x^{(2)} & \dots & x^{(m)} \\ \vdots & \vdots & \dots & \vdots \end{bmatrix} \in \mathbb{R}^{n_x \times m}$$

$$w \in \mathbb{R}^{n_x \times 1}$$

$$b = [b \quad b \quad \dots \quad b]$$

$$Z = w^T X + b = [z^{(1)} \quad z^{(2)} \quad \dots \quad z^{(m)}] = [w^T x^{(1)} + b \quad w^T x^{(2)} + b \quad \dots \quad w^T x^{(m)} + b]$$

, where  $Z \in \mathbb{R}^{1 \times m}$

$$A = [a^{(1)} \quad a^{(2)} \quad \dots \quad a^{(m)}] = [\sigma(z^{(1)}) \quad \sigma(z^{(2)}) \quad \dots \quad \sigma(z^{(m)})] = \sigma(Z)$$

---

```
import numpy as np
```

```
# Broadcasting: even though b is in 1xR, it spans as a vector
```

```
Z = np.dot(w.T, x) + b
```

---

## 2.15 Vectorizing Logistic Regression's Gradient Output

$$dz^{(i)} = a^{(i)} - y^{(i)}$$

$$dZ = [dz^{(1)}, \dots, dz^{(m)}]$$

$$A = [a^{(1)}, \dots, a^{(m)}]$$

$$Y = [y^{(1)}, \dots, y^{(m)}]$$

$$dZ = A - Y = [a^{(1)} - y^{(1)}, \dots, a^{(m)} - y^{(m)}]$$

$$J = 0; dw_1 = 0; dw_2 = 0; db = 0$$

for  $i = 1$  to  $m$ :

$$z^{(i)} = w^T x^{(i)} + b$$

$$a^{(i)} = \sigma(z^{(i)})$$

$$\mathcal{J} += -[y^{(i)} \log(a^{(i)}) + (1 - y^{(i)}) \log(1 - a^{(i)})]$$

$$dz^{(i)} = a^{(i)} - y^{(i)}$$

\* The value of  $dw_1, dw_2, \dots, dw_{n_x}, db$  in the code is cumulative:

$$dw_1 += x_1^{(i)} dz^{(i)}$$

$$dw_2 += x_2^{(i)} dz^{(i)}$$

...

$$dw_{n_x} += x_{n_x}^{(i)} dz^{(i)}$$

$$db += dz^{(i)}$$

$$J/ = m, dw_1/ = m, dw_2/ = m, \dots, dw_{n_x}/ = m, db/ = m$$

$$db = \frac{1}{m} \sum_{i=1}^m dz^{(i)} = \frac{1}{m} \text{np.sum}(dz)$$

$$\begin{aligned} dw &= \frac{1}{m} X dz^T = \frac{1}{m} \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & \dots & x_1^{(m)} \\ x_2^{(1)} & x_2^{(2)} & \dots & x_2^{(m)} \\ \vdots & \vdots & \dots & \vdots \\ x_{n_x}^{(1)} & x_{n_x}^{(2)} & \dots & x_{n_x}^{(m)} \end{bmatrix} \begin{bmatrix} dz^{(1)} \\ \vdots \\ dz^{(m)} \end{bmatrix} \\ &= \frac{1}{m} \begin{bmatrix} \vdots & \vdots & \dots & \vdots \\ X^{(1)} & X^{(2)} & \dots & X^{(m)} \\ \vdots & \vdots & \dots & \vdots \end{bmatrix} \begin{bmatrix} dz^{(1)} \\ \vdots \\ dz^{(m)} \end{bmatrix} = \frac{1}{m} [X^{(1)} dz^{(1)} + \dots + X^{(m)} dz^{(m)}] \end{aligned}$$

Vectorize:

$$dw += x^{(i)} dz^{(i)}$$

$$Z = w^T + b = \text{np.dot}(w.T, X) + b$$

$$A = \sigma(Z)$$

$$dZ = A - Y$$

$$dw = \frac{1}{m} X dZ^T$$

$$db = \frac{1}{m} \text{np.sum}(dZ)$$

Single Iteration of Gradient Descent for Logistic Regression:

$$w := w - \alpha dw$$

$$b := b - \alpha db$$

## 2.16 Broadcasting in Python

Calories from Carbs, Proteins, Fats in 100g of different foods:

	<i>Apples</i>	<i>Beef</i>	<i>Eggs</i>	<i>Potatoes</i>
<i>Carb</i>	56.0	0.0	4.4	68.0
<i>Protein</i>	1.2	104.0	52.0	8.0
<i>Fat</i>	1.8	135.0	99.0	0.9

Calculate the percentage of calories for Carb, Potein, Fat:

---

```
import numpy as np

A = np.array([[56.0, 0.0, 4.4, 68.0],
              [1.2, 104.0, 52.0, 8.0],
              [1.8, 135.0, 99.0, 0.9]])
print(A)

# sum vertically(axis=0)
cal = A.sum(axis=0)
print(cal)

# compute percentage
percentage = 100* A/cal.reshape(1,4)
print(percentage)
```

---

General Principle:

$(m, n)$  matrix operation  $\rightarrow (1, n), (1, m) \rightarrow (m, n), (m, n)$

$(m, n) + \text{real number } \mathbb{R} \rightarrow (m+\mathbb{R}, n+\mathbb{R})$

## 2.17 A Note on Python/Numpy Vectors

---

```
import numpy as np

# Don't use this (rank 1 array):
# a = np.random.randn(5)

# Use this:
a = np.random.randn(5,1)
a = np.random.randn(1,5)
assert(a.shape == (5,1))
```

---



## 2.18 Explanation of Logistic Regression Cost Function(Optional)

Logistic regression cost function (1 example):

$$\text{If } y = 1 : P(y|x) = \hat{y}$$

$$\text{If } y = 0 : P(y|x) = 1 - \hat{y}$$

$$P(y|x) = \hat{y}^y (1 - \hat{y})^{(1-y)}$$

$$\log P(y|x) = y \log \hat{y} + (1 - y) \log(1 - \hat{y})$$

Logistic regression cost function (m examples):

$$\log P(\text{labels in training set}) = \log \prod_{i=1}^m P(y^{(i)}|x^{(i)})$$

$$= \sum_{i=1}^m \log P(y^{(i)}|x^{(i)}) = - \sum_{i=1}^m \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

Cost(minimize):

$$\mathcal{J}(w, b) = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

Minimizing the loss corresponds with maximizing  $\log P(y|x)$ .

## 2.19 Week2. Assignment

1. Python+Basics+With+Numpy+v3.ipynb
2. Logistic+Regression+with+a+Neural+Network+mindset+v5.ipynb