# Neural Networks and Deep Learning

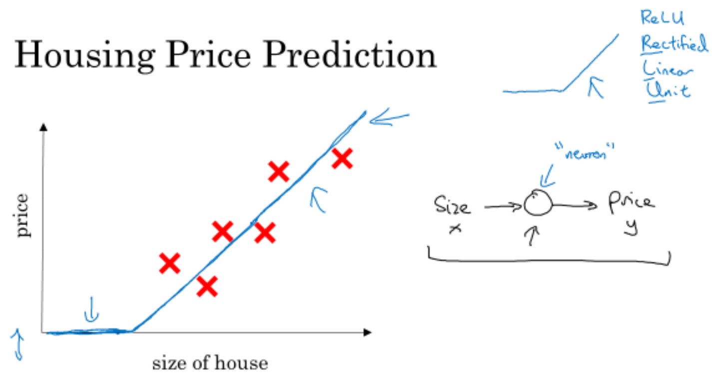
Study Notes by Shujuan Huang

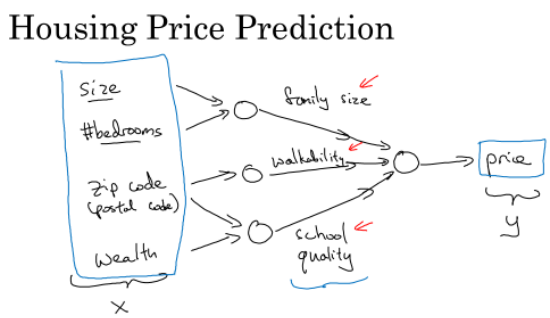
08172018

## Table of Contents

**Week 1 Introduction to deep learning**

**1.1 What is a Neural Network?**




**1.2 Supervised learning for Neural Network**

In supervised learning, we are given a data set and already know what our correct output should look like, having the idea that there is a relationship between the input and the output.

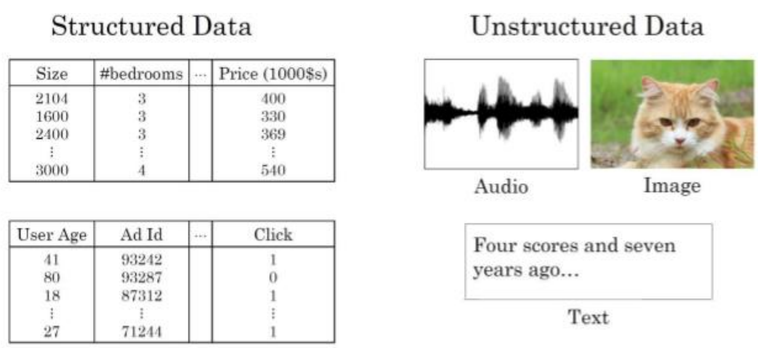
Supervised learning problems are categorized into "regression" and "classification" problems. In a regression problem, we are trying to predict results within a continuous output, meaning that we are trying to map input variables to some continuous function. In a classification problem, we are instead trying to predict results in a discrete output. In other words, we are trying to map input variables into discrete categories.

Here are some examples of supervised learning

| Input(x) | Output (y) | Application | |
|---|---|---|---|
| Home features | Price | Real Estate | Standard NN |
| Ad, user info | Click on ad? (0/1) | Online Advertising | |
| Image | Object (1,...,1000) | Photo tagging | CNN |
| Audio | Text transcript | Speech recognition | RNN |
| English | Chinese | Machine translation | |
| Image, Radar info | Position of other cars | Autonomous driving | Custom/Hybrid |

There are different types of neural network, for example Convolution Neural Network (CNN) used often for image application and Recurrent Neural Network (RNN) used for one-dimensional sequence data such as translating English to Chinses or a temporal component such as text transcript. As for the autonomous driving, it is a hybrid neural network architecture.

Structured vs unstructured data Structured data refers to things that has a defined meaning such as price, age whereas unstructured data refers to thing like pixel, raw audio, text.

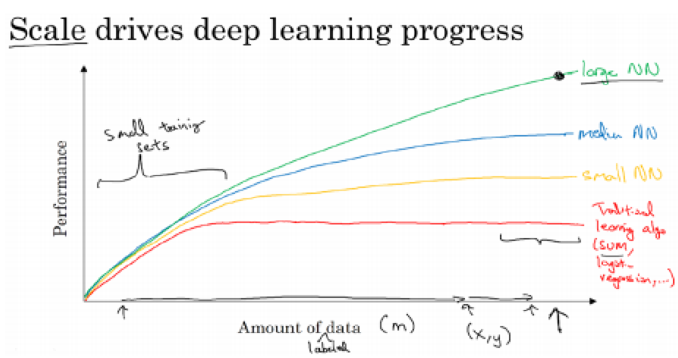



| Size | #bedrooms | ... | Price (1000$s) |
|---|---|---|---|
| 2104 | 3 | | 400 |
| 1600 | 3 | | 330 |
| 2400 | 3 | | 369 |
| ⋮ | ⋮ | | ⋮ |
| 3000 | 4 | | 540 |

| User Age | Ad Id | ... | Click |
|---|---|---|---|
| 41 | 93242 | | 1 |
| 80 | 93287 | | 0 |
| 18 | 87312 | | 1 |
| ⋮ | ⋮ | | ⋮ |
| 27 | 71244 | | 1 |

**1.3 Why is deep learning taking off**

Deep learning is taking off due to a large amount of data available through the digitization of the society, faster computation and innovation in the development of neural network algorithm.

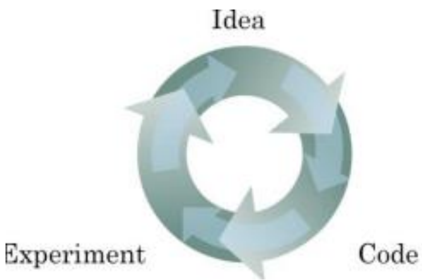



- Data
- Computation
- Algorithms (ex. From sigmoid to relu)

Two things have to be considered to get to the high level of performance:

- Being able to train a big enough neural network
- Huge amount of labeled data

The process of training a neural network is iterative




It could take a good amount of time to train a neural network, which affects your productivity. Faster computation helps to iterate and improve new algorithm.

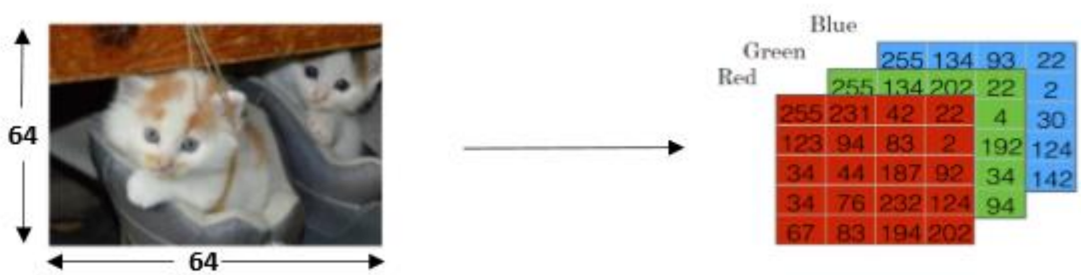
**Week 2 Neural Network Basics**

**2.1 Binary classification**

In a binary classification problem, the result is a discrete value output.

For example - account hacked (1) or compromised (0) - a tumor malign (1) or benign (0)

Example: Cat vs Non-Cat The goal is to train a classifier that the input is an image represented by a feature vector, $x$, and predicts whether the corresponding label $y$ is 1 or 0. In this case, whether this is a cat image (1) or a non-cat image (0).




An image is store in the computer in three separate matrices corresponding to the Red, Green, and Blue color channels of the image. The three matrices have the same size as the image, for example, the resolution of the cat image is 64 pixels X 64 pixels, the three matrices (RGB) are 64 X 64 each.

The value in a cell represents the pixel intensity which will be used to create a feature vector of n dimension. In pattern recognition and machine learning, a feature vector represents an object, in this case, a cat or no cat.

To create a feature vector, $x$, the pixel intensity values will be “unroll” or “reshape” for each color. The dimension of the input feature vector $x$ is $nx$ = 64 $x$ 64 $x$ 3 = 12 288.

$$x = \begin{bmatrix} 255 \\ 231 \\ 42 \\ \vdots \\ 255 \\ 134 \\ 202 \\ \vdots \\ 255 \\ 134 \\ 93 \\ \vdots \end{bmatrix} \begin{array}{l} \left.\begin{matrix} \\ \\ \\ \\ \end{matrix}\right\} \text{red} \\ \left.\begin{matrix} \\ \\ \\ \\ \end{matrix}\right\} \text{green} \\ \left.\begin{matrix} \\ \\ \\ \\ \end{matrix}\right\} \text{blue} \end{array}$$

$n = n_x = 12288$

$x \longrightarrow y$

# Notation

$(x, y)$ $\quad x \in \mathbb{R}^{n_x}$, $y \in \{0, 1\}$

$m$ training examples: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$

$M = M_{train}$ $\qquad M_{test}$ = #test examples.

$$X = \underbrace{\begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}}_{m} \updownarrow n_x \qquad \cancel{\begin{bmatrix} - & x^{(1)T} & - \\ & \vdots & \\ - & x^{(m)T} & - \end{bmatrix}}$$

$Y = [y^{(1)}\ y^{(2)}\ \ldots,\ y^{(m)}]$

$Y \in \mathbb{R}^{1 \times m}$

$Y.\text{shape} = (1, m)$

$X \in \mathbb{R}^{n_x \times m}$ $\qquad X.\text{shape} = (n_x, m)$

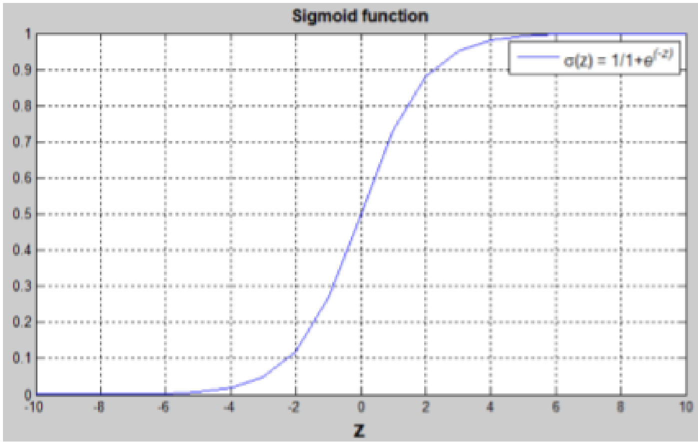
**2.2 Logistic Regression**

Logistic regression is a learning algorithm used in a supervised learning problem when the output $y$ are all either zero or one. The goal of logistic regression is to minimize the error between its predictions and training data. Example: Cat vs No - cat Given an image represented by a feature vector $x$, the algorithm will evaluate the probability of a cat being in that image.

$$Given\ x\ ,\ \hat{y} = P(y = 1|x), \text{where } 0 \ \leq \hat{y} \leq 1$$

The parameters used in Logistic regression are:

- The input features vector: $x \in \mathbb{R}^{n_x}$, where $n_x$ is the number of features
- The training label: $y \in 0,1$
- The weights: $w \in \mathbb{R}^{n_x}$, where $n_x$ is the number of features
- The threshold: $b \in \mathbb{R}$
- The output: $\hat{y} = \sigma(w^T x + b)$
- Sigmoid function: s = $\sigma(w^T x + b)$ = $\sigma(z)$= $\frac{1}{1+ e^{-z}}$




$(w^T x + b)$ is a linear function $(ax + b)$, but since we are looking for a probability constraint between [0,1], the sigmoid function is used. The function is bounded between [0,1] as shown in the graph above.

Some observations from the graph:

- If $z$ is a large positive number, then $\sigma(z)$ = 1
- If $z$ is small or large negative number, then $\sigma(z)$ = 0
- If $z = 0$, then $\sigma(z)$ = 0.5

## **Logistic Regression Cost Function (**Loss function: The smaller, the better)

To train the parameters $w$ and $b$, we need to define a cost function.

Recap:

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

$x^{(i)}$ the i-th training example

$Given \left\{\left(x^{(1)}, y^{(1)}\right), \cdots, \left(x^{(m)}, y^{(m)}\right)\right\},\ we\ want\ \hat{y}^{(i)} \approx y^{(i)}$

Loss (error) function:

The loss function measures the discrepancy between the prediction ($\hat{y}^{(i)}$) and the desired output ($y^{(i)}$). In other words, the loss function computes the error for a single training example.

$$L\left(\hat{y}^{(i)}, y^{(i)}\right) = \frac{1}{2}(\hat{y}^{(i)} - y^{(i)})^2$$

$$L\left(\hat{y}^{(i)}, y^{(i)}\right) = -( y^{(i)} \log\left(\hat{y}^{(i)}\right) + \left(1 - y^{(i)}\right)\log(1 - \hat{y}^{(i)})$$

- If $y^{(i)} = 1$: $L\left(\hat{y}^{(i)}, y^{(i)}\right) = -\log\left(\hat{y}^{(i)}\right)$ where $\log\left(\hat{y}^{(i)}\right)$ and $\hat{y}^{(i)}$ should be close to 1
- If $y^{(i)} = 0$: $L\left(\hat{y}^{(i)}, y^{(i)}\right) = -\log\left(1 - \hat{y}^{(i)}\right)$ where $\log\left(1 - \hat{y}^{(i)}\right)$ and $\hat{y}^{(i)}$ should be close to 0

Cost function

The cost function is the average of the loss function of the entire training set. We are going to find the parameters $w$ $and$ $b$ that minimize the overall cost function.

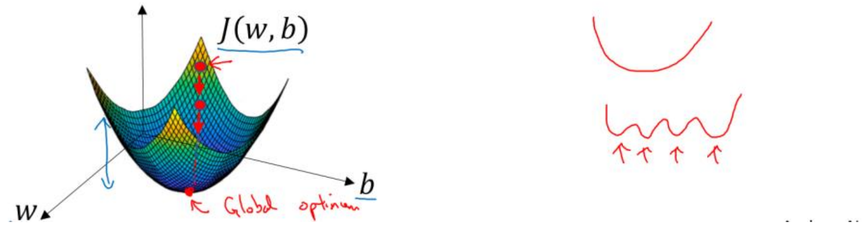
$$J(w, b) = \frac{1}{m}\sum_{i=1}^{m} L\left(\hat{y}^{(i)}, y^{(i)}\right) = -\frac{1}{m}\sum_{i=1}^{m}\left[\left( y^{(i)} \log\left(\hat{y}^{(i)}\right) + \left(1 - y^{(i)}\right)\log(1 - \hat{y}^{(i)}\right)\right]$$

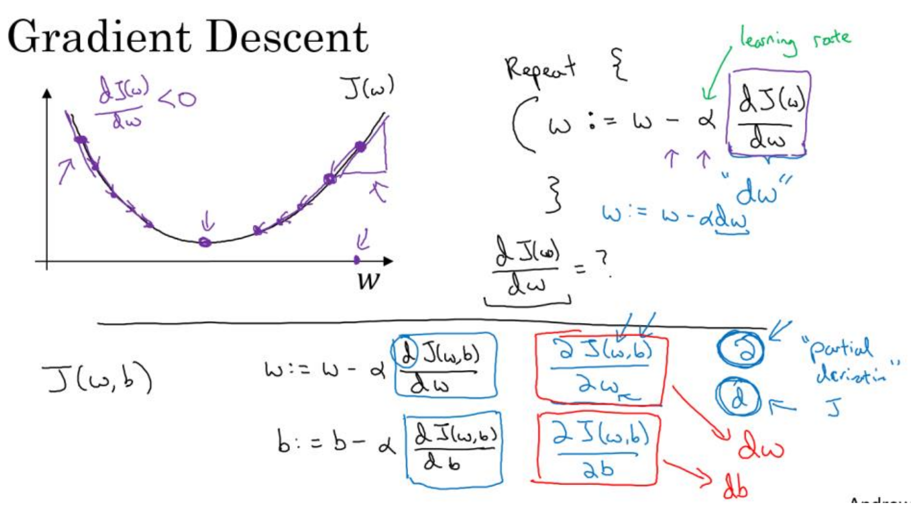
**2.3 Gradient Descent**

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

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

Want to find $w, b$ that minimize $J(w,b)$




Learning rate α controls how big a step we take on each iteration or gradient descent

Derivative: Slope of a function at a certain point

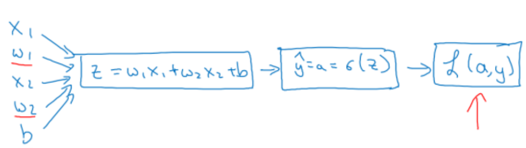
**2.4 Logistic Regression Gradient Descent**

Logistic regression recap

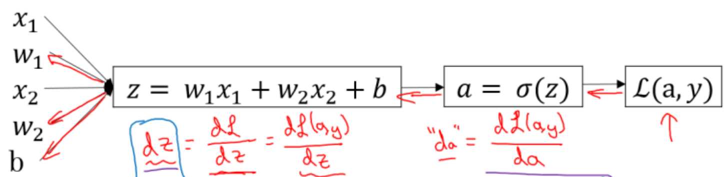
$z = w^T x + b$

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

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



# Logistic regression derivatives




$$\boxed{dz} = \frac{dL}{dz} = \frac{dL(a,y)}{dz} = a - y = \boxed{\frac{dL}{da}} \cdot \boxed{\frac{da}{dz}}$$

$$\text{"}da\text{"} = \frac{dL(a,y)}{da} = \boxed{-\frac{y}{a} + \frac{1-y}{1-a}}$$

$$\frac{da}{dz} = a(1-a)$$

$$\frac{\partial L}{\partial w_1} = \text{"}dw_1\text{"} = x_1 \cdot dz. \quad dw_2 = x_2 \cdot dz. \quad db = dz.$$

$$w_1 := w_1 - \alpha\, dw_1$$
$$w_2 := w_2 - \alpha\, dw_2$$
$$b := b - \alpha\, db.$$

**2.5 Vectorization**

# What is vectorization?

$z = w^T x + b$ $\quad w = \begin{bmatrix} \vdots \\ \vdots \end{bmatrix} \quad x = \begin{bmatrix} \vdots \\ \vdots \end{bmatrix} \quad w \in \mathbb{R}^{n_x} \quad x \in \mathbb{R}^{n_x}$

Non-vectorized:

```
z = 0
for i in range(n-x):
    z += w[i] * x[i]
z += b
```

Vectorized

$z = \underbrace{np.dot(w, x)}_{w^T x} + b$

→ GPU
→ CPU } SIMD – single instruction multiple data.

Z=np.dot(w,x)+b (in python)

# Whenever possible, avoid explicit for-loops.

$u = Av$

$u_i = \sum_i \sum_j A_{ij} v_j$

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

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.

$$v = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix} \rightarrow u = \begin{bmatrix} e^{v_1} \\ e^{v_2} \\ \vdots \\ e^{v_n} \end{bmatrix}$$

```
u = np.zeros((n,1))
for i in range(n):
    u[i]=math.exp(v[i])
```

```
import numpy as np
u = np.exp(v)
np.log(v)
np.abs(v)
np.maximum(v,0)
v**2        1/v
```

## Logistic regression derivatives

J = 0, ~~dw1 = 0, dw2 = 0~~, db = 0 $\quad$ dw = np.zeros((n_x, 1))

for i = 1 to n:

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

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

$\quad J \mathrel{+}= -[y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})]$

$\quad dz^{(i)} = a^{(i)}(1 - a^{(i)})$

for j = 1 … $n_x$: $\quad dw_j \mathrel{+}= \ldots$

$\quad \cancel{dw_1 \mathrel{+}= x_1^{(i)} dz^{(i)}}$ $\quad n_x = 2$ $\quad dw \mathrel{+}= x^{(i)} dz^{(i)}$

$\quad \cancel{dw_2 \mathrel{+}= x_2^{(i)} dz^{(i)}}$

$\quad db \mathrel{+}= dz^{(i)}$

J = J/m, $\cancel{dw_1 = dw_1/m, \; dw_2 = dw_2/m}$ db = db/m

dw /= m.

### 2.6 Broadcasting in Python

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

| | Apples | Beef | Eggs | Potatoes |
|---|---|---|---|---|
| Carb | 56.0 | 0.0 | 4.4 | 68.0 |
| Protein | 1.2 | 104.0 | 52.0 | 8.0 |
| Fat | 1.8 | 135.0 | 99.0 | 0.9 |

= A (3,4) $\quad$ 0 $\quad$ 1

59cal $\quad \frac{56}{59} \approx 94.9\%$

Calculate % of calories from Carb, Protein, Fat. Can you do this without explicit for-loop?

```
cal = A.sum(axis = 0)
percentage = 100*A/(cal.reshape(1,4))
```

(3,4) / (1,4)

**Broadcasting example**

$$\begin{bmatrix}1\\2\\3\\4\end{bmatrix} + 100 \rightarrow \begin{bmatrix}100\\100\\100\\100\end{bmatrix}$$

$$\underset{(m,n)\;(2,3)}{\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix}} + \underset{(1,n) \rightsquigarrow (m,n) \quad (2,3)}{\begin{bmatrix}100 & 200 & 300\end{bmatrix}} \rightarrow \begin{bmatrix}100 & 200 & 300\\100 & 200 & 300\end{bmatrix}$$

$$\underset{(m,n)}{\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix}} + \underset{(m,1) \rightarrow (m,n)}{\begin{bmatrix}100\\200\end{bmatrix}} \rightarrow \begin{bmatrix}100 & 100 & 100\\200 & 200 & 200\end{bmatrix}$$

**General Principle**

(m, n) matrix $\quad$ + − * / $\quad$ (1, n) $\rightsquigarrow$ (m, n)

$\quad$ (m, 1) $\rightsquigarrow$ (m, n)

(m, 1) + $\mathbb{R}$

$\begin{bmatrix}1\\2\\3\end{bmatrix} + 100 = \begin{bmatrix}101\\102\\103\end{bmatrix}$

$\begin{bmatrix}1 & 2 & 3\end{bmatrix} + 100 = \begin{bmatrix}101 & 102 & 103\end{bmatrix}$

Matlab/Octave: bsxfun

### A note on python/numpy vectors

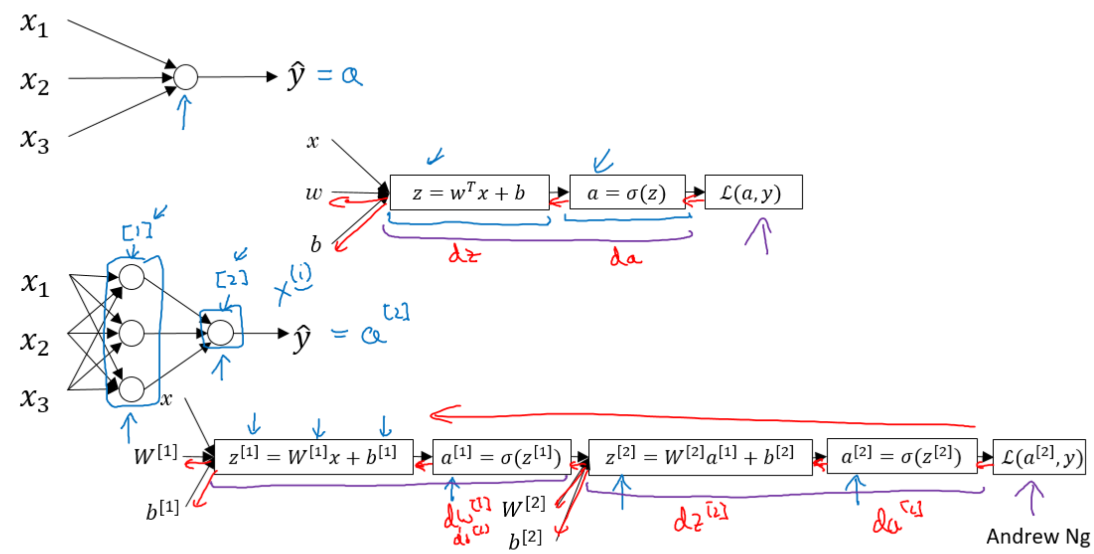
There are both strengths (simple and flexible) and weaknesses (no error message, hard to debug)

```
In [1]: import numpy as np

        a = np.random.randn(5)

In [2]: print(a)

        [ 0.50290632 -0.29691149  0.95429604 -0.82126861 -1.46269164]

In [3]: print(a.shape)

        (5,)

In [4]: print(a.T)

        [ 0.50290632 -0.29691149  0.95429604 -0.82126861 -1.46269164]

In [5]: print(np.dot(a,a.T))

        4.06570109321

In [6]: a = np.random.randn(5,1)
        print(a)

        [[-0.0967311 ]
         [-2.38617377]
         [-0.3243588 ]
         [-0.96216349]
         [ 0.54410384]]

In [7]: print(a.T)

        [[-0.0967311  -2.38617377 -0.3243588  -0.96216349  0.54410384]]

In [8]: print(np.dot(a,a.T))

        [[ 0.00935691  0.23081721  0.03137558  0.09307113 -0.05263176]
         [ 0.23081721  5.69382526  0.77397645  2.29588928 -1.2983263 ]
         [ 0.03137558  0.77397645  0.10520863  0.31208619 -0.17648487]
         [ 0.09307113  2.29588928  0.31208619  0.92575858 -0.52351684]
         [-0.05263176 -1.2983263  -0.17648487 -0.52351684  0.29604898]]
```

**Week 3 Shallow Neural Networks**

**3.1 Neural Network Overview**

# What is a Neural Network?




### 3.2 Neural Network Representation

**General comments:**

· superscript (i) will denote the $i^{th}$ training example while superscript [l] will denote the $l^{th}$ layer

**Sizes:**

·$m$ : number of examples in the dataset

·$n_x$ : input size

·$n_y$ : output size (or number of classes)

·$n_h^{[l]}$ : number of hidden units of the $l^{th}$ layer

In a for loop, it is possible to denote $n_x = n_h^{[0]}$ and $n_y = n_h{}^{[\text{number of layers } +1]}$.

·$L$ : 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

$\cdot Y \in \mathbb{R}^{n_y \times m}$ is the label matrix

$\cdot y^{(i)} \in \mathbb{R}^{n_y}$ is the output label for the $i^{th}$ example

$\cdot W^{[l]} \in \mathbb{R}^{\text{number of units in next layer} \times \text{number of units in the previous layer}}$ is the weight matrix,superscript [l] indicates the layer

$\cdot b^{[l]} \in \mathbb{R}^{\text{number of units in next layer}}$ is the bias vector in the $l^{th}$ layer

$\cdot \hat{y} \in \mathbb{R}^{n_y}$ is the predicted output vector. It can also be denoted $a^{[L]}$ where $L$ is the number of layers in the network.

**For representations:**

· nodes represent inputs, activations or outputs

· edges represent weights or biases

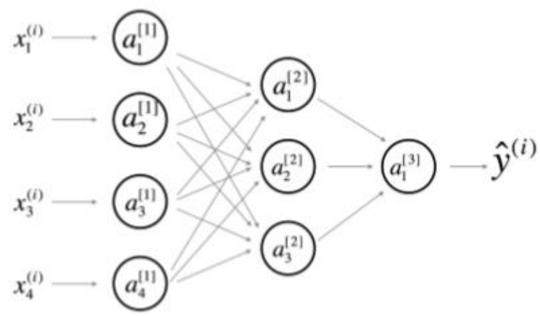



Figure 1: Comprehensive Network: representation commonly used for Neural Networks. For better aesthetic, we omitted the details on the parameters ($w_{ij}^{[l]}$ and $b_i^{[l]}$ etc...) that should appear on the edges

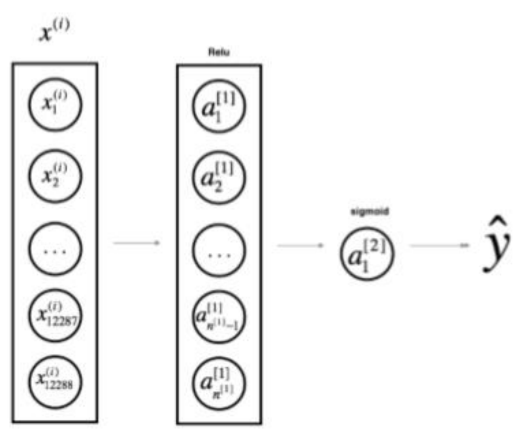



Figure 2: Simplified Network: a simpler representation of a two layer neural network, both are equivalent.

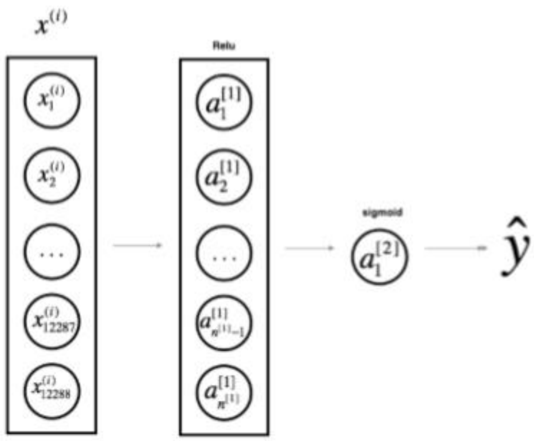
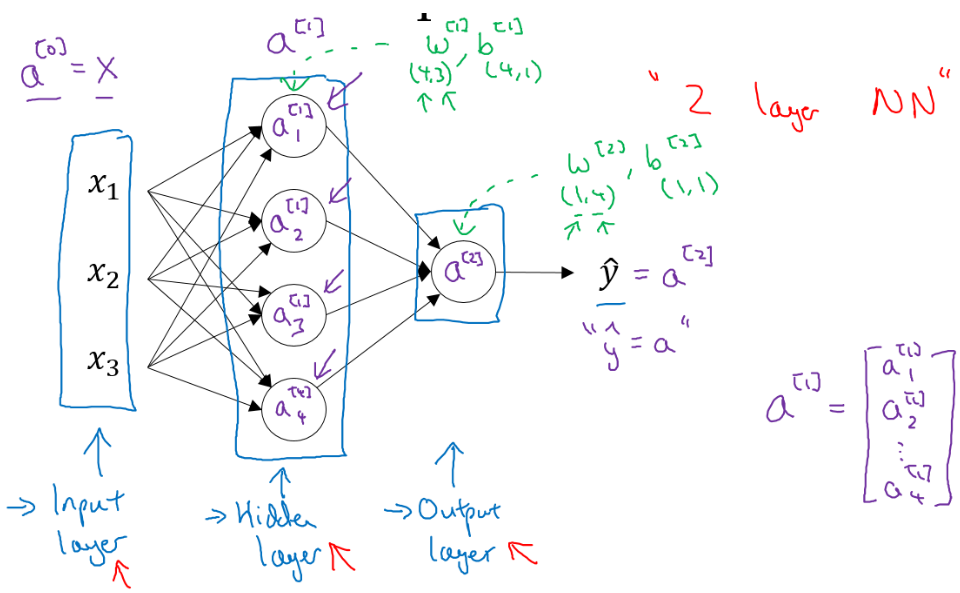
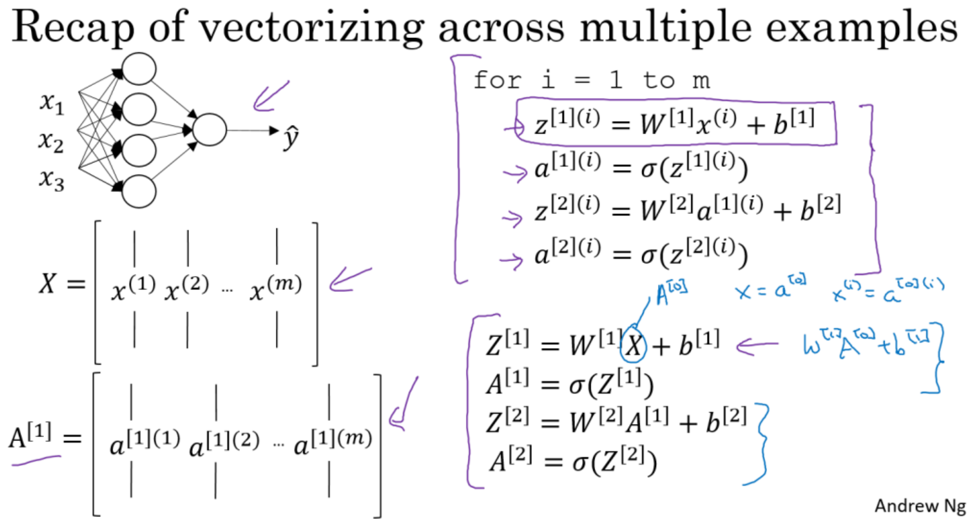
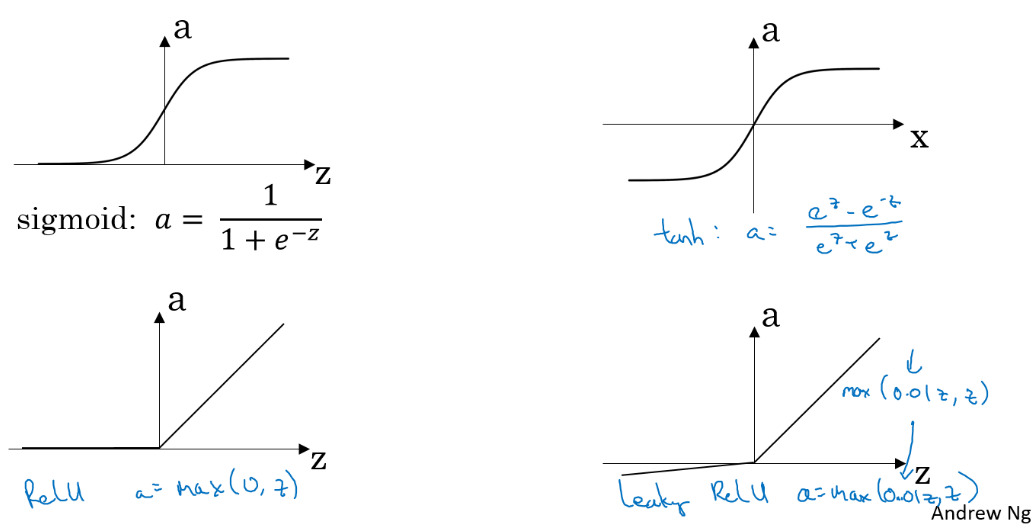



Figure 2: Simplified Network: a simpler representation of a two layer neural network, both are equivalent.






### 3.3 Activation functions



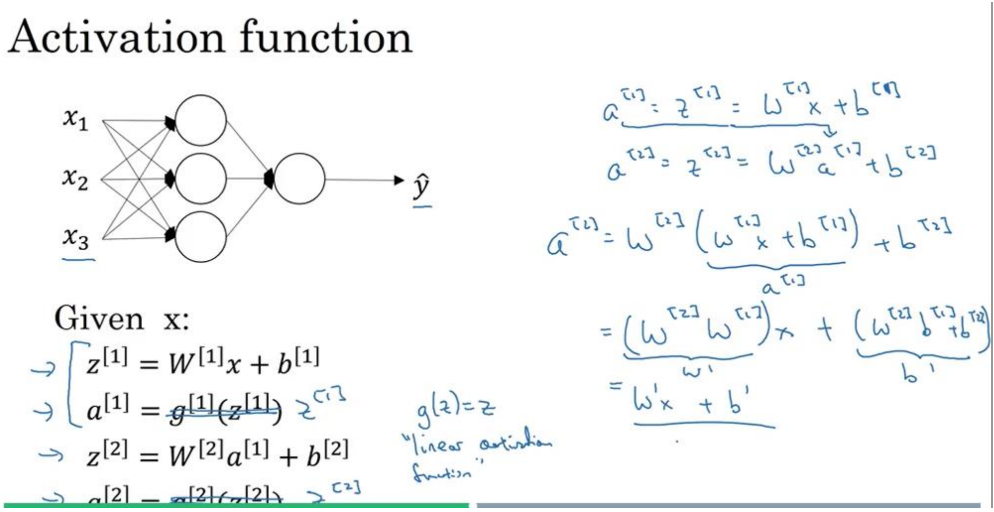

Tanh is pronounced as Tan H, which normally works better than sigmoid function.

You may have different activation functions for different layers, for example, tanh for hidden layer, and sigmoid for output layer.

### Leaky ReLU

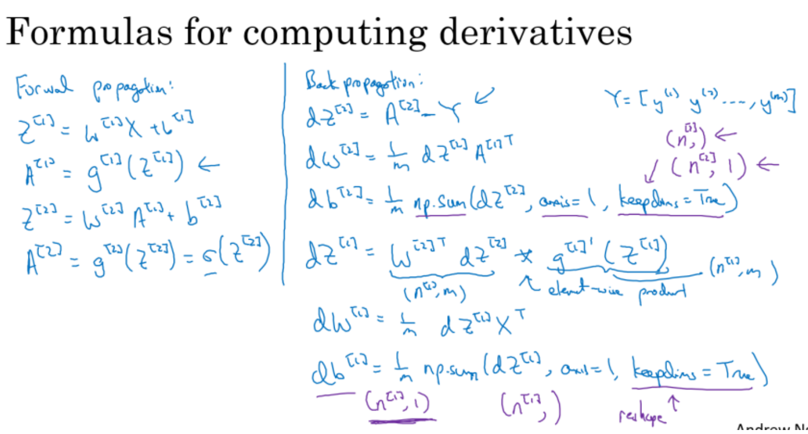
- Leaky version of a Rectified Linear Unit.
- It allows a small gradient when the unit is not active: `f(x) = alpha * x for x < 0`, `f(x) = x for x >= 0`
- $f(x)=\max(x,\alpha x)$ with $\alpha \in (0,1)$f(x)=max(x,αx) with α∈(0,1)
- Leaky ReLUs allow a small, non-zero gradient when the unit is not active.

### Why do you need non-linear activation functions?



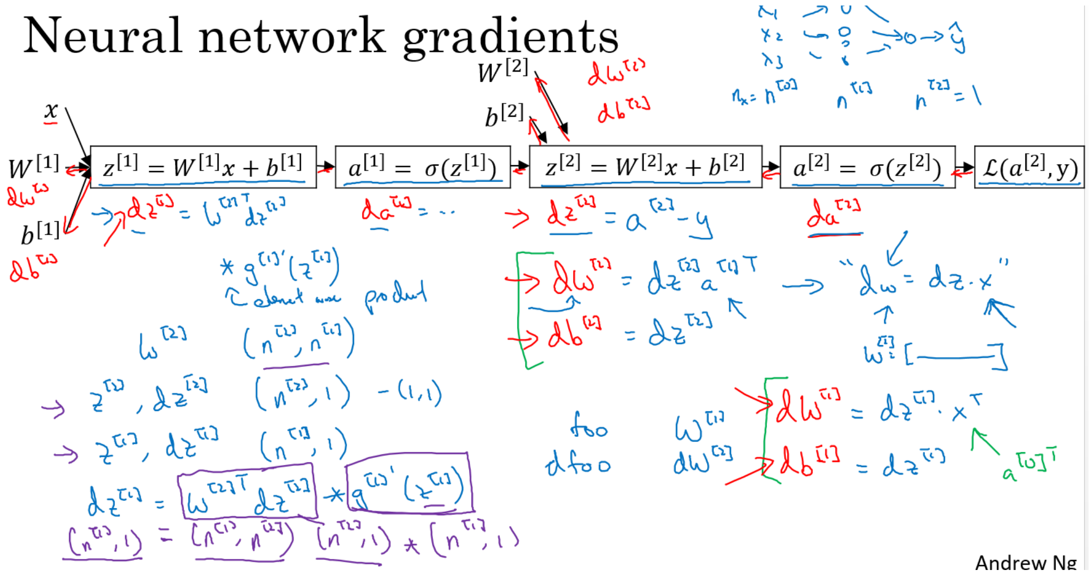

If you use a linear activation function or alternatively if you don't have an activation function, then no matter how many layers your neural network has always doing is just computing a linear activation function, so you might as well not have any hidden layers some of the cases that briefly mentioned it turns out that if you have a linear activation function here and a sigmoid function here, then this model is no more expressive than standard logistic regression without any hidden layer, so I won't bother to prove that but you could try to do so if you want but the take-home is that a linear hidden layer is more or less useless because on the composition of two linear functions is itself a linear function so unless you throw a non-linearity in there then you're not computing more interesting functions even as you go deeper in the network.

**3.4 Gradient Descent for Neural Networks**




Axis=1: summing horizontally

**3.5 Backpropagation intuition**




## Summary of gradient descent

$$dz^{[2]} = a^{[2]} - y$$

$$dW^{[2]} = dz^{[2]}a^{[1]^T}$$

$$db^{[2]} = dz^{[2]}$$

$$dz^{[1]} = W^{[2]T}dz^{[2]} * g^{[1]\prime}(z^{[1]})$$

$(n^{[1]}, 1)$

$$dW^{[1]} = dz^{[1]}x^T$$

$$db^{[1]} = dz^{[1]}$$

$$dZ^{[2]} = A^{[2]} - Y$$

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

$$dW^{[2]} = \frac{1}{m}dZ^{[2]}A^{[1]^T}$$

$$db^{[2]} = \frac{1}{m}np.sum(dZ^{[2]}, axis = 1, keepdims = True)$$

element-wise product

$$dZ^{[1]} = W^{[2]T}dZ^{[2]} * g^{[1]\prime}(Z^{[1]})$$

$(n^{[1]}, m)$ $\quad (n^{[1]}, m)$ $\quad (n^{[1]}, m)$

$$dW^{[1]} = \frac{1}{m}dZ^{[1]}X^T$$

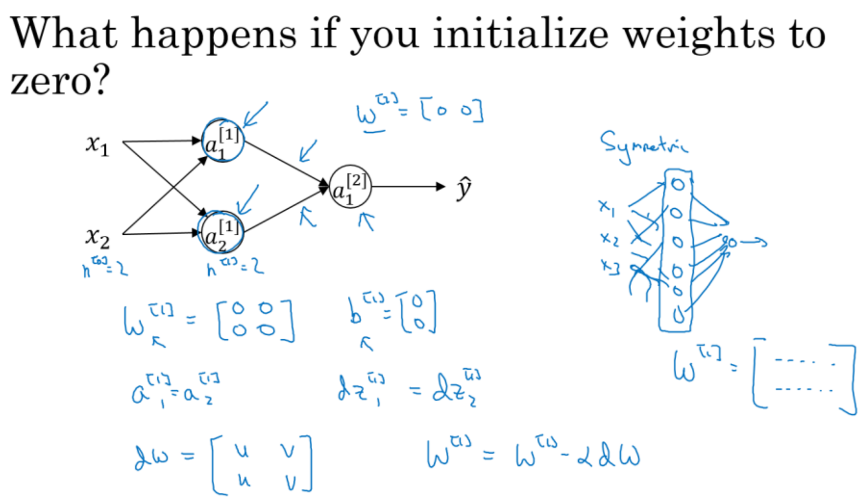
$$db^{[1]} = \frac{1}{m}np.sum(dZ^{[1]}, axis = 1, keepdims = True)$$

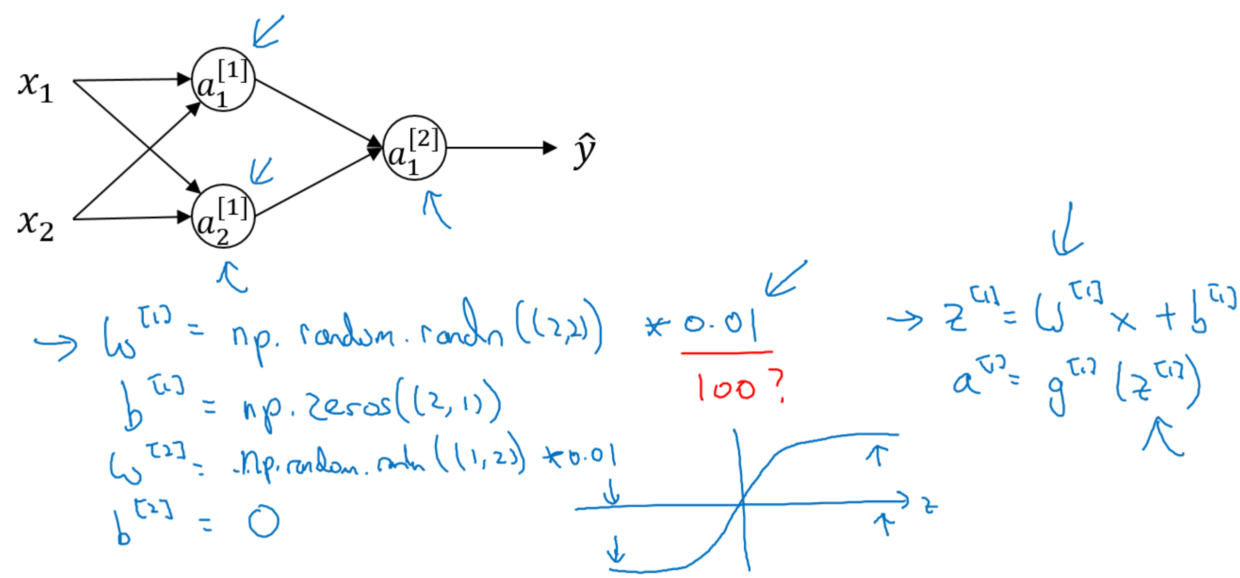
### 3.6 Random Initialization

If you initialize all the ways, all the values of w to 0, then because both hidden units start off computing the same function. And both hidden the units have the same influence on the output unit, then after one iteration, that same statement is still true, the two hidden units are still symmetric. And therefore, by induction, after two iterations, three iterations and so on, no matter how long you train your neural network, both hidden units are still computing exactly the same function. And so in this case, there's really no point to having more than one hidden unit. Because they are all computing the same thing. And of course, for larger neural networks, let's say of three features and maybe a very large number of hidden units, a similar argument works to show that with a neural network like this.

If you initialize the weights to zero, then all of your hidden units are symmetric. And no matter how long you're upgrading the center, all continue to compute exactly the same function. So that's not helpful, because you want the different hidden units to compute different functions. The solution to this is to initialize your parameters randomly.

So it's okay to initialize b to just zeros. Because so long as w is initialized randomly, you start off with the different hidden units computing different things. And so you no longer have this symmetry breaking problem.



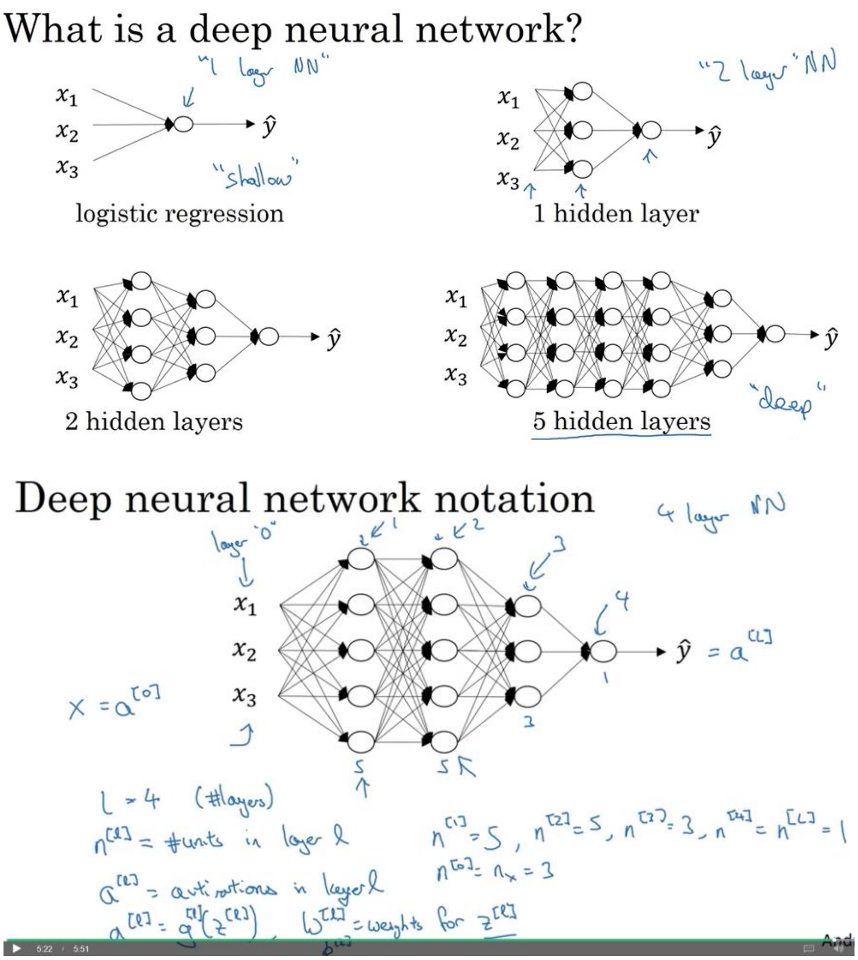




So that's why multiplying by 0.01?

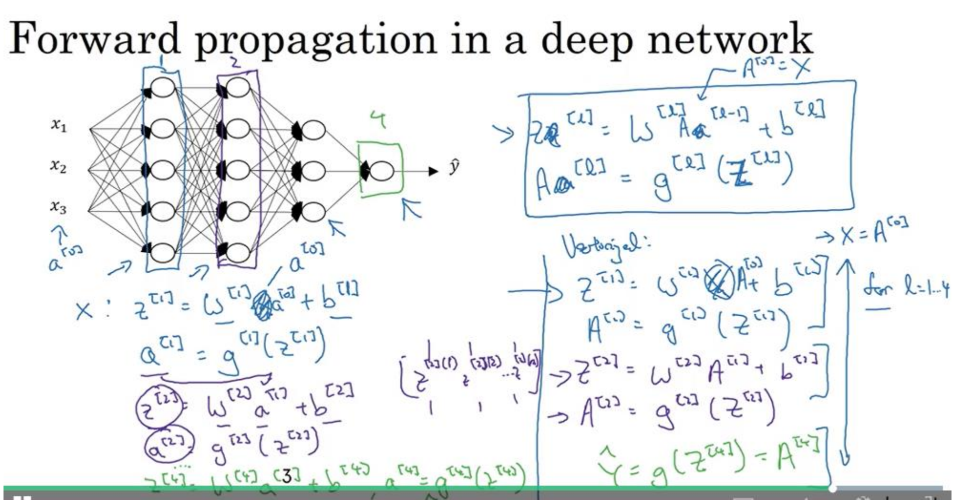
- When you're training a neural network with just one hidden layer, it is a relatively shallow neural network, without too many hidden layers. Set it to 0.01 will probably work okay.
- But when you're training a very very deep neural network, then you might want to pick a different constant than 0.01. But either way, it will usually end up being a relatively small number.

**Week 4 Deep Neural Networks**

**4.1 Deep Layer Neural Network**

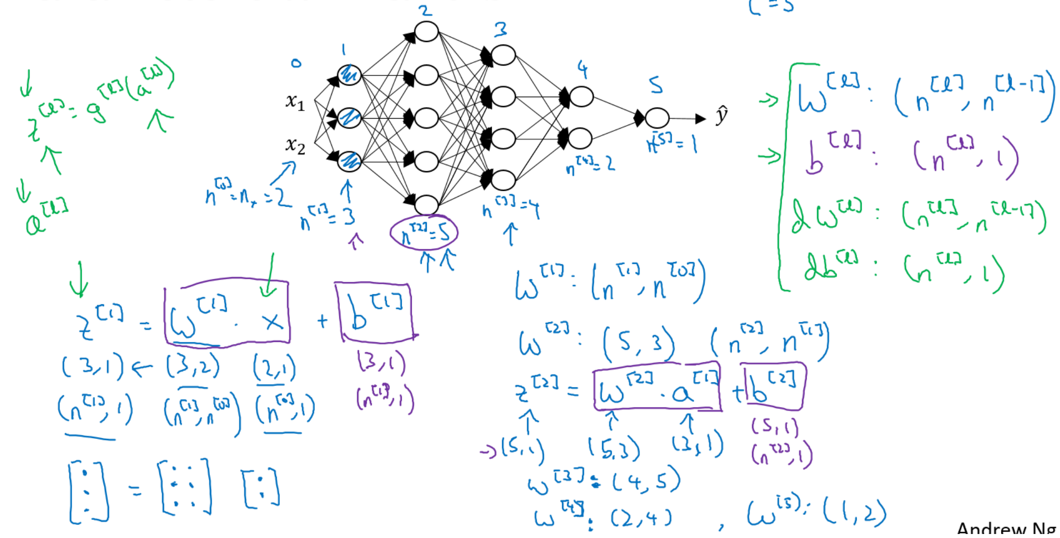



**4.2 Forward propagation in a deep network**

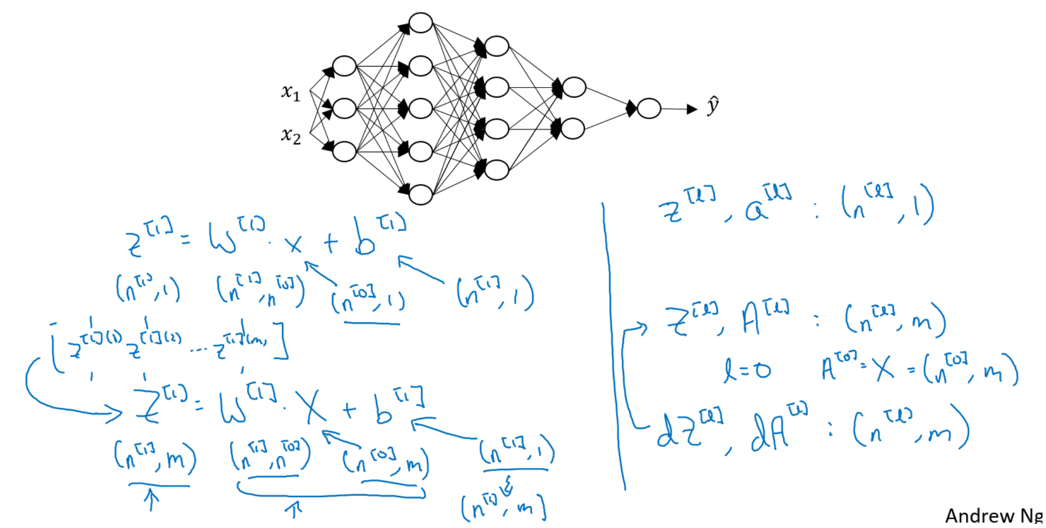


**4.3 Getting your matrix dimensions right**

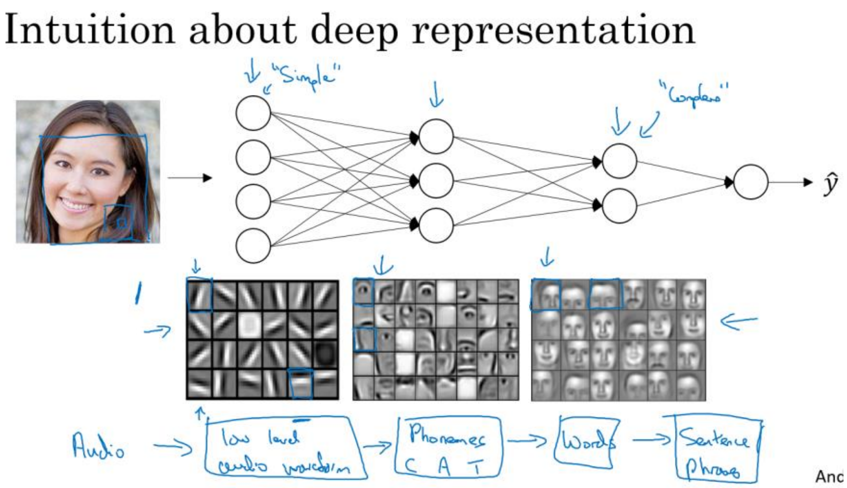
## Parameters $W^{[l]}$ and $b^{[l]}$



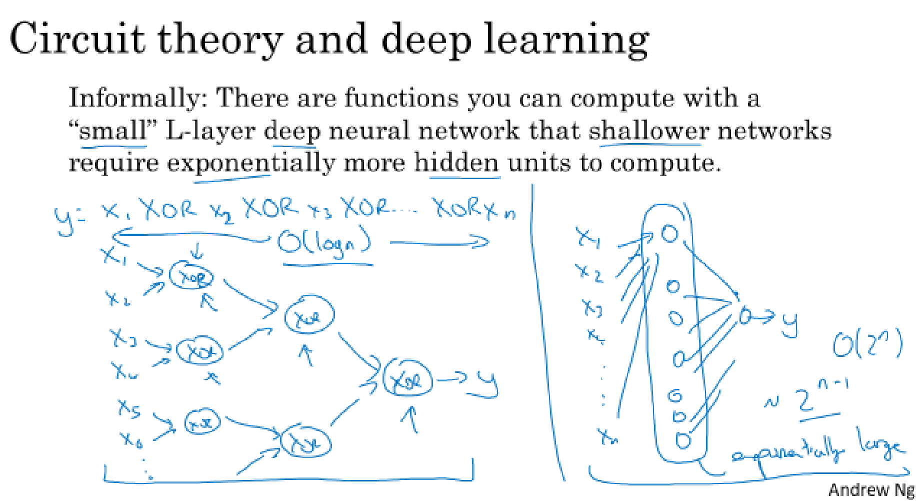

## Vectorized implementation



**4.4 Why deep representation**

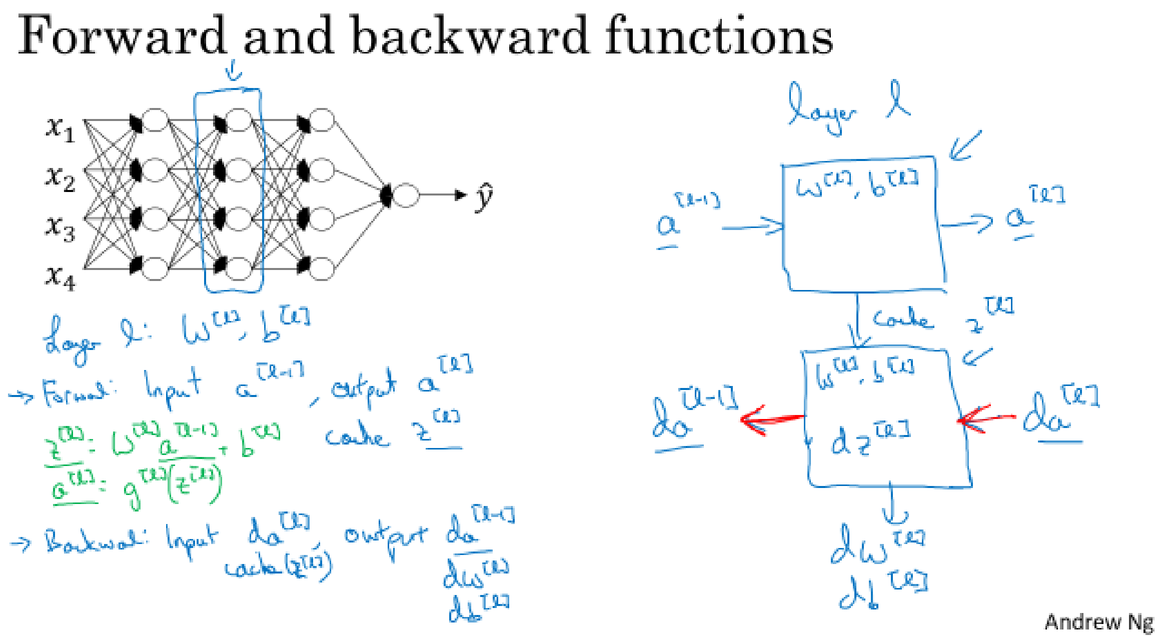


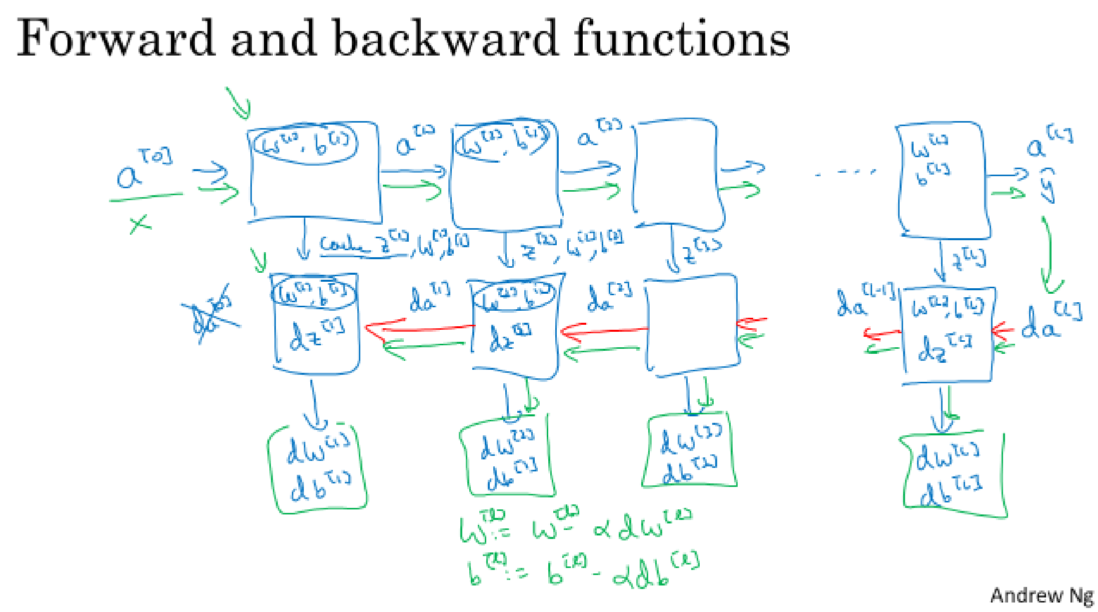

- Edges -> nose, eyes… -> face
- Audio -> low level audio wave form feature -> phonemes (basic unites for sound) -> words→ sentence / phase




If only one hidden layer is allowed, then it would be exponentially large.

**4.5 Building blocks of deep neural networks**



4.6 Forward and backward propagation

# Forward propagation for layer $l$

→ Input $a^{[l-1]}$

→ Output $a^{[l]}$, cache ($z^{[l]}$) $W^{[l]}, b^{[l]}$

$$z^{[l]} = W^{[l]} \cdot a^{[l-1]} + b^{[l]}$$
$$a^{[l]} = g^{[l]}(z^{[l]})$$

Vectorized:
$$Z^{[l]} = W^{[l]} \cdot A^{[l-1]} + b^{[l]}$$
$$A^{[l]} = g^{[l]}(Z^{[l]})$$

$a^{[0]}$

$A^{[0]}$

$X = A^{[0]} \rightarrow \square \rightarrow \square \rightarrow \square \rightarrow$

Andrew Ng

# Backward propagation for layer $l$

→ Input $da^{[l]}$

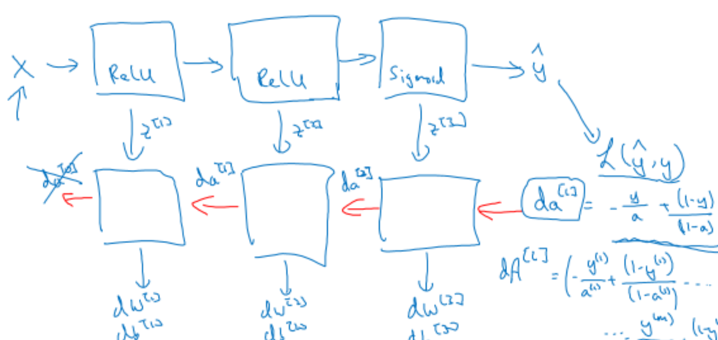
→ Output $\boxed{da^{[l-1]}}$, $dW^{[l]}$, $db^{[l]}$

$$dz^{[l]} = da^{[l]} * g^{[l]\prime}(z^{[l]})$$
$$dW^{[l]} = dz^{[l]} \cdot a^{[l-1]}$$
$$db^{[l]} = dz^{[l]}$$
$$\boxed{da^{[l-1]}} = W^{[l]T} \cdot dz^{[l]}$$
$$dz^{[l]} = W^{[l+1]T} dz^{[l+1]} * g^{[l]\prime}(z^{[l]})$$

$$dZ^{[l]} = dA^{[l]} * g^{[l]\prime}(Z^{[l]})$$
$$dW^{[l]} = \frac{1}{m} dZ^{[l]} \cdot A^{[l-1]T}$$
$$db^{[l]} = \frac{1}{m} \text{np.sum}(dZ^{[l]}, \text{axis}=1, \text{keepdims}=\text{True})$$
$$\boxed{dA^{[l-1]}} = W^{[l]T} \cdot dZ^{[l]}$$

Andrew Ng

# Summary




$$da^{[l]} = -\frac{y}{a} + \frac{(1-y)}{(1-a)}$$

$$dA^{[l]} = \left(-\frac{y^{(1)}}{a^{(1)}} + \frac{(1-y^{(1)})}{(1-a^{(1)})} \cdots \cdots -\frac{y^{(m)}}{a^{(m)}} + \frac{(1-y^{(m)})}{(1-a^{(m)})}\right)$$

Andrew Ng

$$
\begin{aligned}
Z^{[1]} &= W^{[1]}X + b^{[1]} \\
A^{[1]} &= g^{[1]}(Z^{[1]}) \\
Z^{[2]} &= W^{[2]}A^{[1]} + b^{[2]} \\
A^{[2]} &= g^{[2]}(Z^{[2]}) \\
&\vdots \\
A^{[L]} &= g^{[L]}(Z^{[L]}) = \hat{Y}
\end{aligned}
$$

"It's like the brain"

$$
\begin{aligned}
dZ^{[L]} &= A^{[L]} - Y \\
dW^{[L]} &= \frac{1}{m} dZ^{[L]} A^{[L]^T} \\
db^{[L]} &= \frac{1}{m} np.\mathrm{sum}(\mathrm{d}Z^{[L]}, axis = 1, keepdims = True) \\
dZ^{[L-1]} &= dW^{[L]^T} dZ^{[L]} g'^{[L]}(Z^{[L-1]}) \\
&\vdots \\
dZ^{[1]} &= dW^{[L]^T} dZ^{[2]} g'^{[1]}(Z^{[1]}) \\
dW^{[1]} &= \frac{1}{m} dZ^{[1]} A^{[1]^T} \\
db^{[1]} &= \frac{1}{m} np.\mathrm{sum}(\mathrm{d}Z^{[1]}, axis = 1, keepdims = True)
\end{aligned}
$$

**4.7 Parameters vs Hyperparameters**

Parameters: $W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}, W^{[3]}, b^{[3]}$ ...

Hyperparameters: learning rate $\alpha$

#iterations

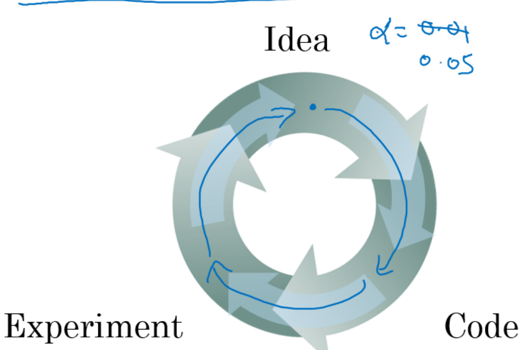
#hidden layers $L$

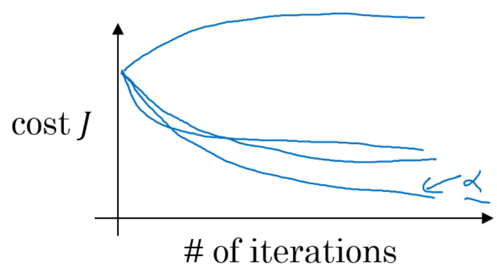
#hidden units $n^{[1]}, n^{[2]}, \ldots$

choice of activation function

Later: Momentum, mini-batch size, regularizations

# Applied deep learning is a very empirical process




Vision, Speech, NLP, Ad, Search, Recommendation.

Applications: Vision, speech, NLD, AD(online advertising), web search, product recommendation….