Neural Networks Basics

Logistic Regression as a Neural Network

$$\hat{y} = \sigma\left(w^T x + b\right), \text{ where } \sigma(z) = \frac{1}{1+e^{-z}}$$
$$\text{Given } \left\{\left(x^{(1)}, y^{(1)}\right), \ldots, \left(x^{(m)}, y^{(m)}\right)\right\}, \text{ want } \hat{y}^{(i)} \approx y^{(i)}$$

Loss(error) function:

Don't use this, non-convex

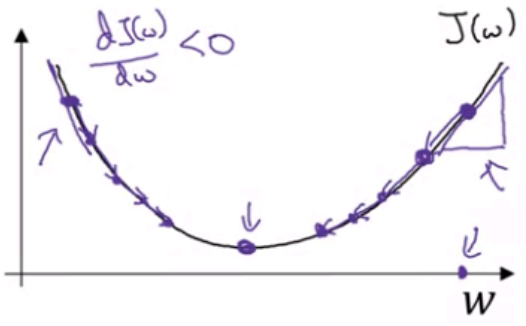
$$\ell(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$$

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.

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

$$\text{Want to find } w, b \text{ that minimize } J(w, b)$$

Gradient Descent



Repeat {

$w := w - \alpha \frac{dJ(w)}{dw}$

learning r

↑ ↑ "dw"

$w := w - \alpha dw$

}

$\frac{dJ(w)}{dw} = ?$

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

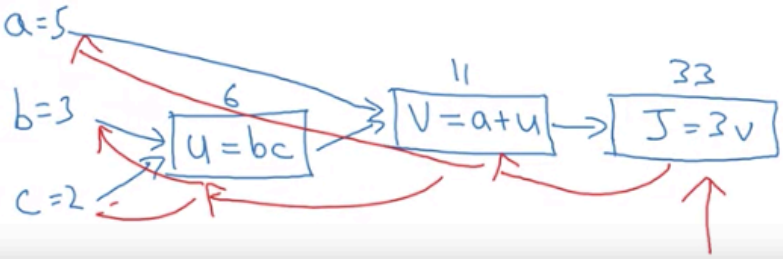
$b := b - \alpha \frac{\partial J(w,b)}{\partial b}$

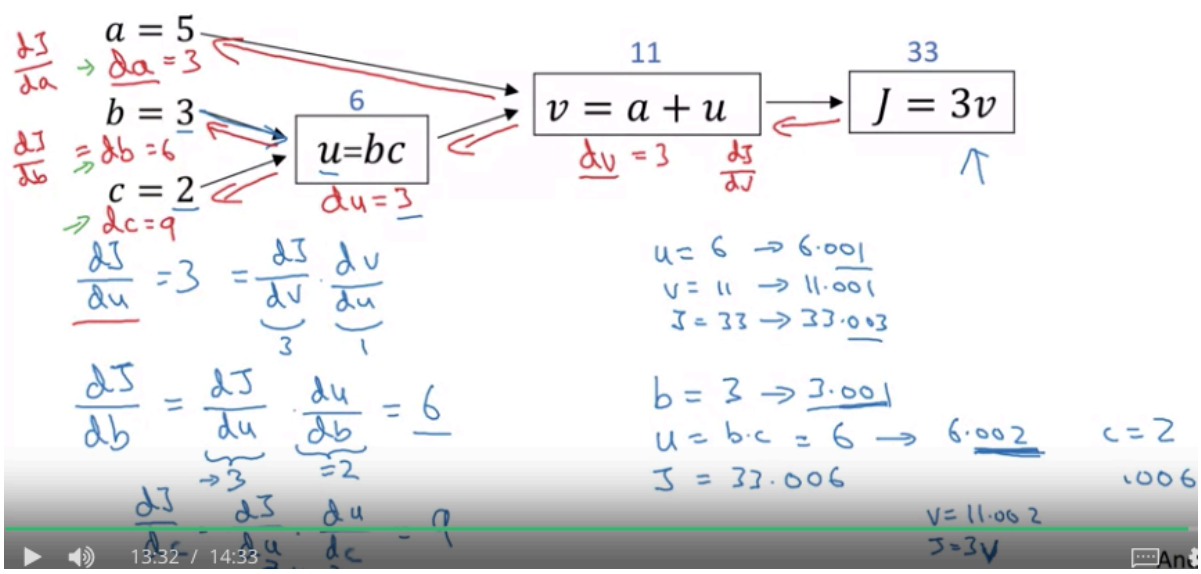
Computation Graph

$J(a,b,c) = 3(a + \underbrace{bc}_{u}) = 3(5 + 3\times 2) = 33$

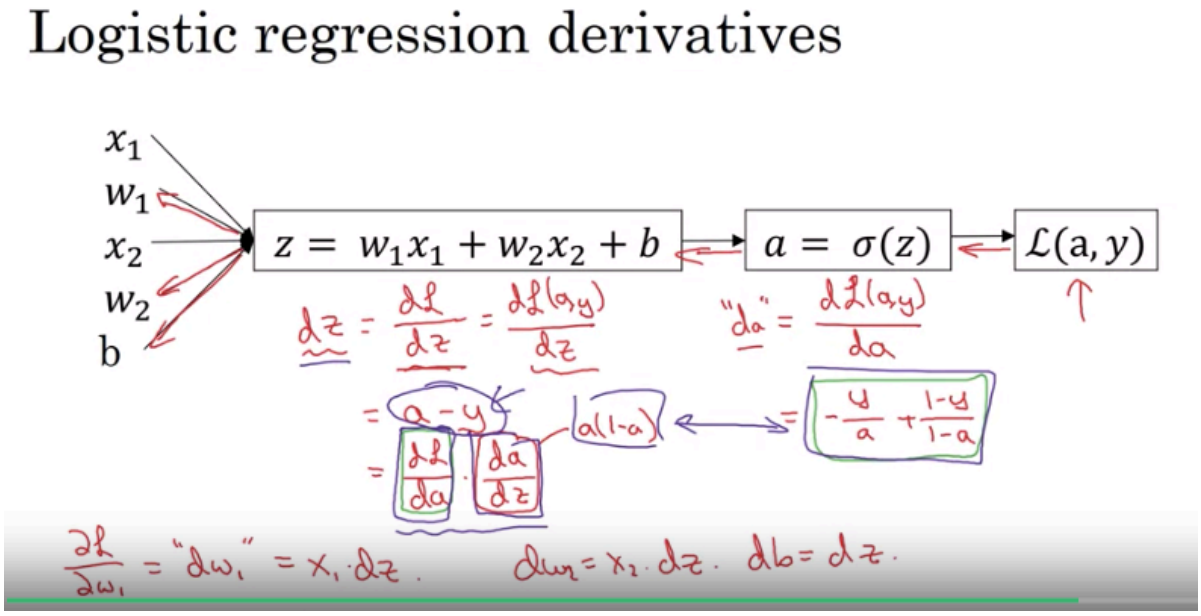
$v = a + u$, $J = 3v$

$u = bc$





Gradient Desent



Logistic regression on m examples

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

$\rightarrow a^{(i)} = \hat{y}^{(i)} = \sigma(z^{(i)}) = \sigma(w^T x^{(i)} + b)$ $\quad dw_1^{(i)}, dw_2^{(i)}, db^{(i)}$

$\frac{\partial}{\partial w_1} J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \underbrace{\frac{\partial}{\partial w_1} \mathcal{L}(a^{(i)}, y^{(i)})}_{dw_1^{(i)} \text{ for } (x^{(i)}, y^{(i)})}$

Step 1 : $\frac{dL}{da}$

$L = -(y \times \log(a) + (1-y) \times \log(1-a))$

$\frac{dL}{da} = -y \times \frac{1}{a} - (1-y) \times \frac{1}{1-a} \times -1$

$$
\begin{aligned}
\frac{d}{dx}\sigma(x) &= \frac{d}{dx}\left[\frac{1}{1+e^{-x}}\right] \\
&= \frac{d}{dx}\left(1+\mathrm{e}^{-x}\right)^{-1} \\
&= -(1+e^{-x})^{-2}(-e^{-x}) \\
&= \frac{e^{-x}}{(1+e^{-x})^2} \\
&= \frac{1}{1+e^{-x}} \cdot \frac{e^{-x}}{1+e^{-x}} \\
&= \frac{1}{1+e^{-x}} \cdot \frac{(1+e^{-x})-1}{1+e^{-x}} \\
&= \frac{1}{1+e^{-x}} \cdot \left(\frac{1+e^{-x}}{1+e^{-x}} - \frac{1}{1+e^{-x}}\right) \\
&= \frac{1}{1+e^{-x}} \cdot \left(1 - \frac{1}{1+e^{-x}}\right) \\
&= \sigma(x) \cdot (1-\sigma(x))
\end{aligned}
$$

In the previous video, Andrew refers to $dz = a(1-a)$

Note that Andrew is using "dz" as a shorthand to refer to $\frac{da}{dz} = a(1-a)$.

To clarify, earlier in this week's videos, Andrew used the name "dz" to refer to a different derivative: $\frac{dL}{dz} = a - y$.

Recall that the relationship between $\frac{dL}{dz}$ and $\frac{da}{dz}$ is:

$\frac{dL}{dz} = \frac{dL}{da} \times \frac{da}{dz}$

$\frac{dL}{dz} = \frac{a-y}{a(1-a)} \times a(1-a) = a - y$

Vectorization

Logistic regression derivatives

$J = 0$, ~~$dw_1 = 0$, $dw_2 = 0$~~, $db = 0$ $\quad dw = np.zeros((n_x, 1))$

→ for $i = 1$ to m:

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

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

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

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

for $j = 1 \dots n_x$: $dw_j \mathrel{+}= \dots$

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

~~$dw_2 \mathrel{+}= x_2^{(i)} dz^{(i)}$~~ $\quad n_x = 2$ $\quad dw \mathrel{+}= x^{(i)} dz^{(i)}$

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

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

$dw /= m$.

Vectorizing Logistic Regression

$dz^{(1)} = a^{(1)} - y^{(1)}$ $\quad dz^{(2)} = a^{(2)} - y^{(2)}$ $\quad \dots$

$dZ = [dz^{(1)} \; dz^{(2)} \dots dz^{(m)}]$ $\quad (1 \times m)$ ←

$A = [a^{(1)} \dots a^{(m)}]$. $\quad Y = [y^{(1)} \dots y^{(m)}]$

→ $dZ = A - Y = [a^{(1)} - y^{(1)} \quad a^{(2)} - y^{(2)} \dots]$

→ $dw = 0$

$dw \mathrel{+}= X^{(1)} dz^{(1)}$

$dw \mathrel{+}= X^{(2)} dz^{(2)}$

$\vdots$

$dw /= m$

~~dw_1, dw_2, $\dots$~~

$db = 0$

$db \mathrel{+}= dz^{(1)}$

$db \mathrel{+}= dz^{(2)}$

$\vdots$

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

$db /= m$.

$db = \frac{1}{m} \sum_{i=1}^{m} dz^{(i)}$

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

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

$= \frac{1}{m} \begin{bmatrix} x^{(1)} & \dots & x^{(m)} \\ 1 & & 1 \end{bmatrix} \begin{bmatrix} dz^{(1)} \\ \vdots \\ dz^{(m)} \end{bmatrix}$

$= \frac{1}{m} [x^{(1)} dz^{(1)} + \dots + x^{(m)} dz^{(m)}]$

$n_x \times 1$

Implementing Logistic Regression

$J = 0,\ dw_1 = 0,\ dw_2 = 0,\ db = 0$

for i = 1 to m:

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

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

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

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

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

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

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

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

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

$db = db/m$

$Z = w^T X + b$

$= np.dot(w.T, X) + b$

$A = \sigma(Z)$

$dZ = A - Y$

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

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

$w := w - \alpha dw$

$b := b - \alpha db$

Andrew

Logistic regression cost function

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

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

$p(y|x)$

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

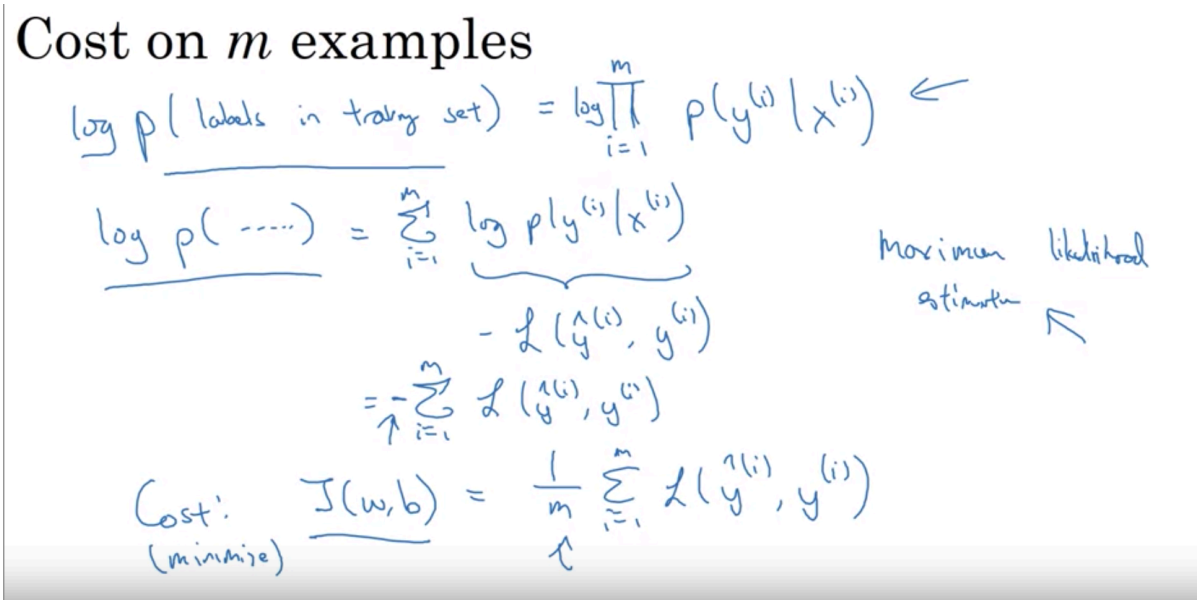
If $y=1$: $p(y|x) = \hat{y}^1 (1-\hat{y})^0 = \hat{y} \cdot 1 = \hat{y}$

If $y=0$: $p(y|x) = \hat{y}^0 (1-\hat{y})^{(1-0)} = 1 \times (1-\hat{y}) = 1-\hat{y}$

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

$= -\mathcal{L}(\hat{y}, y)$

Andrew Ng



Code

For convenience, you should now reshape images of shape (num_px, num_px, 3) in a numpy-array of shape (num_px $*$ num_px $*$ 3, 1). After this, our training (and test) dataset is a numpy-array where each column represents a flattened image. There should be m_train (respectively m_test) columns.

Exercise: Reshape the training and test data sets so that images of size (num_px, num_px, 3) are flattened into single vectors of shape (num_px $*$ num_px $*$ 3, 1).

A trick when you want to flatten a matrix X of shape (a,b,c,d) to a matrix X_flatten of shape (b$*$c$*$d, a) is to use:

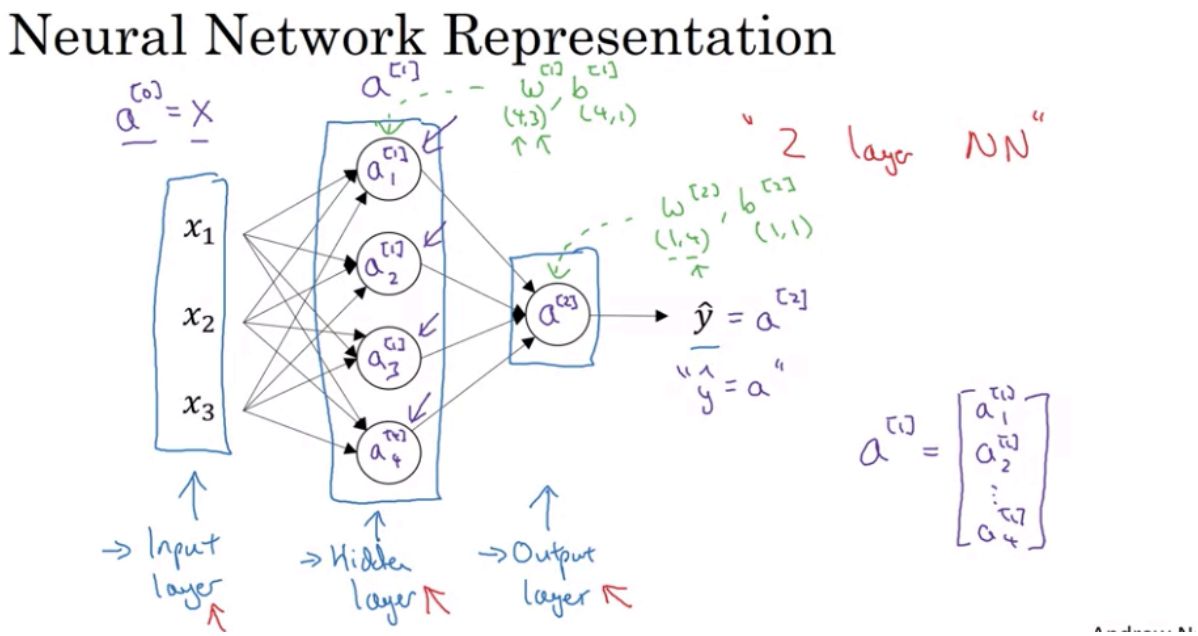
```
X_flatten = X.reshape(X.shape[0], -1).T      # X.T is the
transpose of X
```

To represent color images, the red, green and blue channels (RGB) must be specified for each pixel, and so the pixel value is actually a vector of three numbers ranging from 0 to 255.

One common preprocessing step in machine learning is to center and standardize your dataset, meaning that you substract the mean of the whole numpy array from each example, and then divide each example by the standard deviation of the whole numpy array. But for picture datasets, it is simpler and more convenient and works almost as well to just divide every row of the dataset by 255 (the maximum value of a pixel channel).

Let's standardize our dataset.

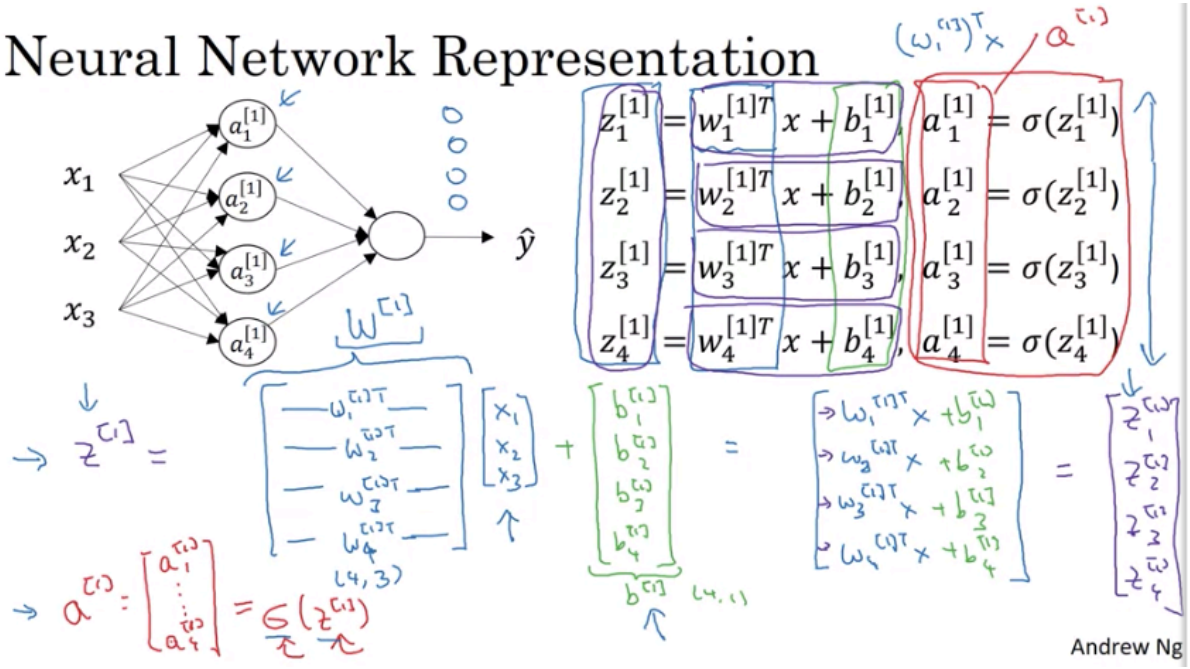
Neural Network



Shallow neural networks

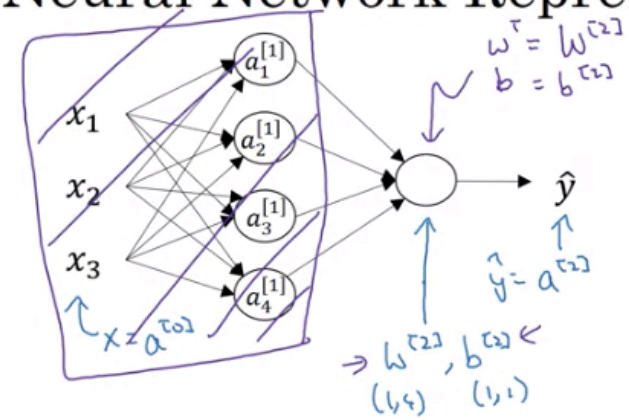
Representation

Neural Network Representation



Andrew Ng

Neural Network Representation learning



$z = w^T x + b$

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

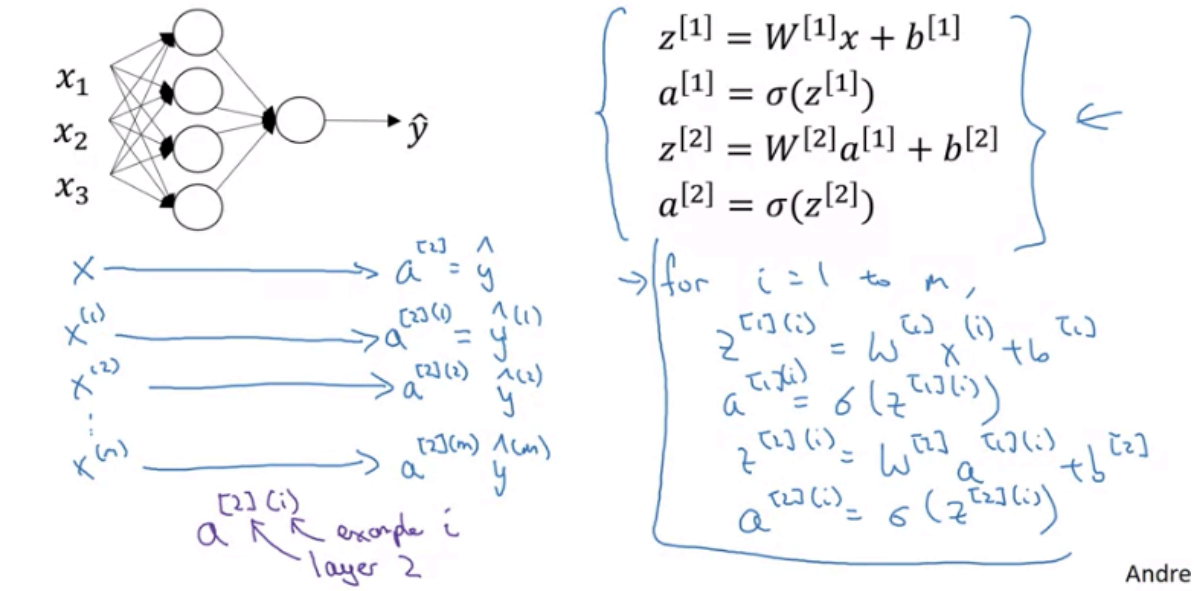
Given input x:

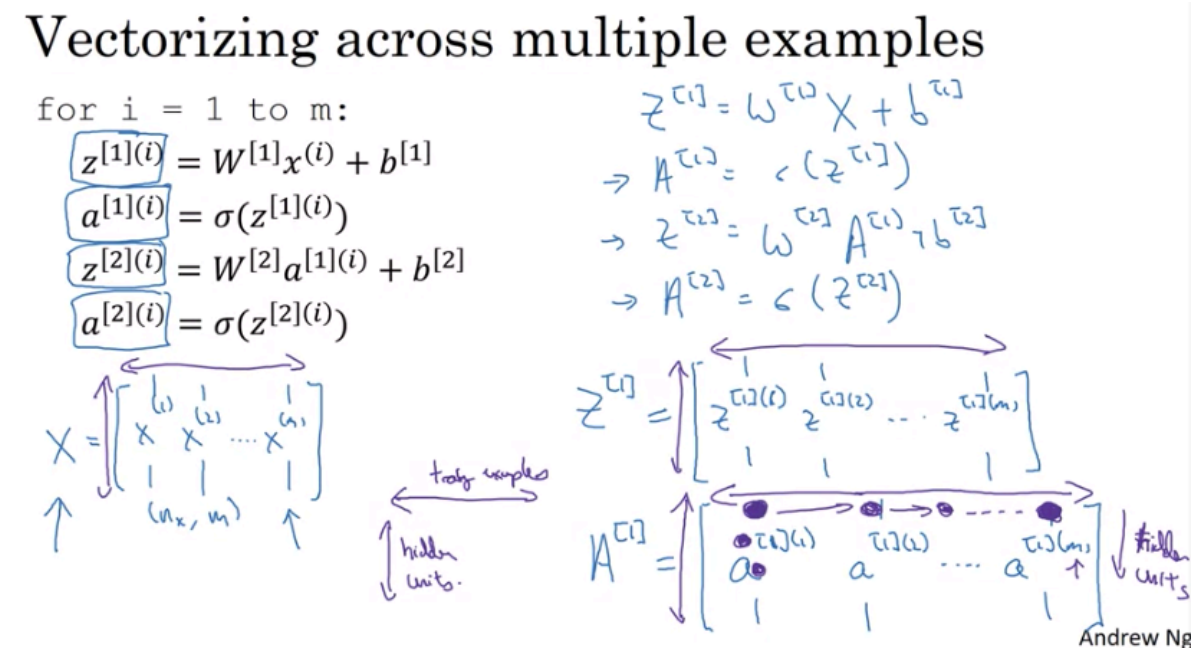
$$z^{[1]}_{(4,1)} = W^{[1]}_{(4,3)} a^{[0]}_{(3,1)} + b^{[1]}_{(4,1)}$$

$$a^{[1]}_{(4,1)} = \sigma(z^{[1]}_{(4,1)})$$

$$z^{[2]}_{(1,1)} = W^{[2]}_{(1,4)} a^{[1]}_{(4,1)} + b^{[2]}_{(1,1)}$$

$$a^{[2]}_{(1,1)} = \sigma(z^{[2]}_{(1,1)})$$

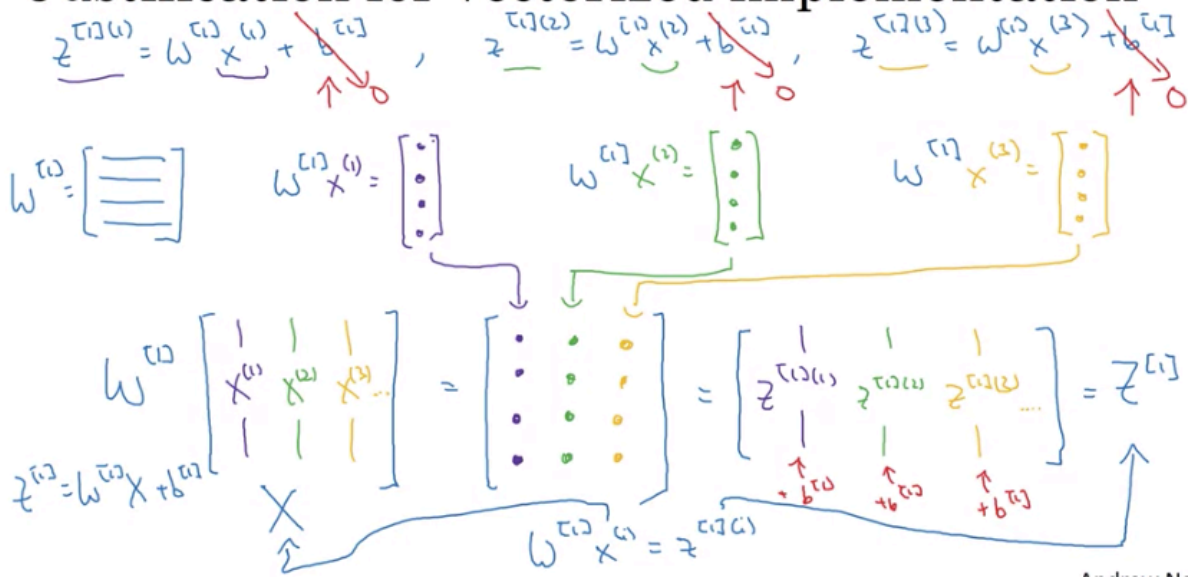




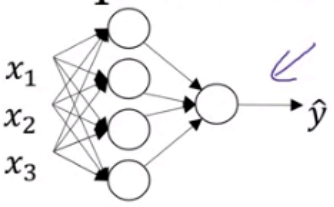
X：竖方向 features

横方向：training example

Justification for vectorized implementation



Recap of vectorizing across multiple examples



$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}$$

$$A^{[1]} = \begin{bmatrix} | & | & & | \\ a^{[1](1)} & a^{[1](2)} & \cdots & a^{[1](m)} \\ | & | & & | \end{bmatrix}$$

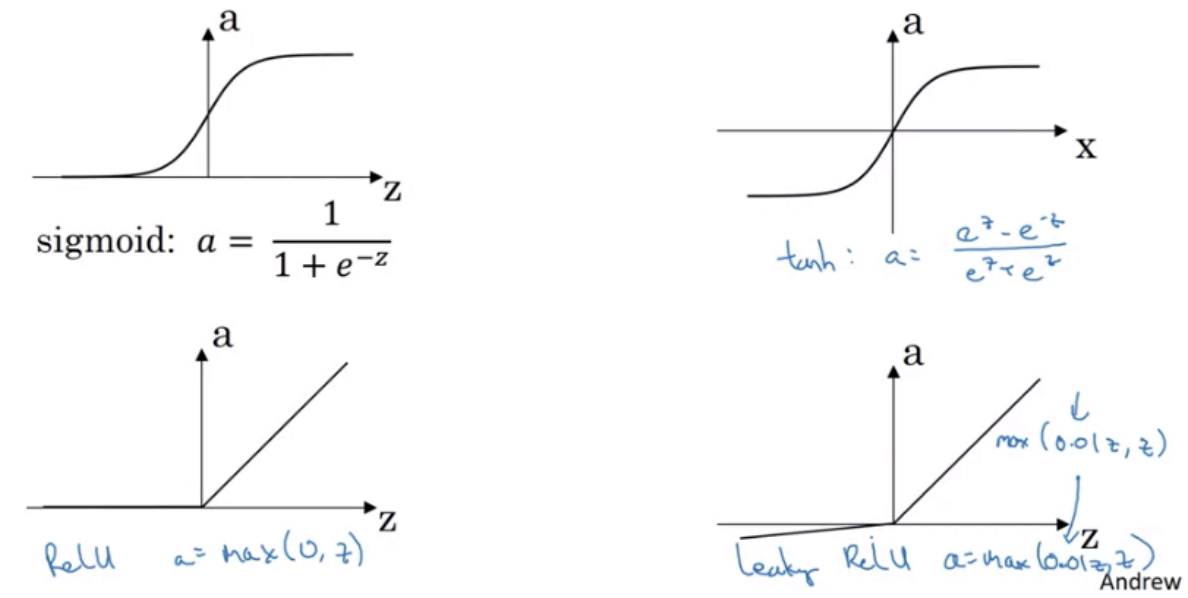
```
for i = 1 to m
```

$$z^{[1](i)} = W^{[1]}x^{(i)} + b^{[1]}$$
$$a^{[1](i)} = \sigma(z^{[1](i)})$$
$$z^{[2](i)} = W^{[2]}a^{[1](i)} + b^{[2]}$$
$$a^{[2](i)} = \sigma(z^{[2](i)})$$

$A^{[0]} = X = a^{[0]}$ $\quad x^{(i)} = a^{[0](i)}$

$$Z^{[1]} = W^{[1]}X + b^{[1]} \leftarrow w^{[1]}A^{[0]} + b^{[1]}$$
$$A^{[1]} = \sigma(Z^{[1]})$$
$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$
$$A^{[2]} = \sigma(Z^{[2]})$$

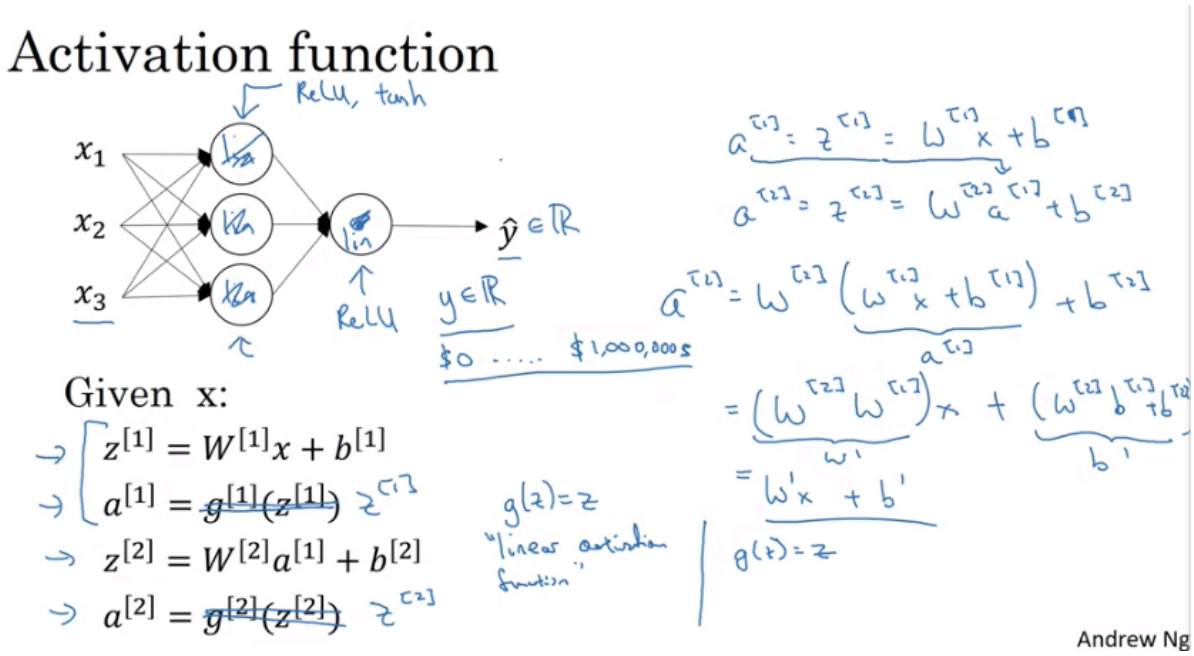
Andrew Ng



Activation function：

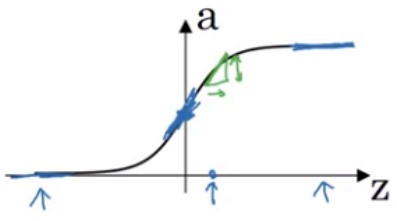
1. Sigmoid: 如果output是binary classification，可以考虑sigmoid function，一般nerver used
2. tanh function：
3. ReLU Rectified Linear Unit $a = \max(0, z)$: by default, 很多人用，faster，slope 1，or 0
4. Leaky ReLU

Why need non-linear activation functions



Derivatives of activation functions

Sigmoid activation function



$$g(z) = \frac{1}{1+e^{-z}}$$

$a = g(z) = \frac{1}{1+e^{-z}}$

$g'(z) = \frac{d}{dz} g(z)$ = slope of $g(z)$ at z

$= \frac{1}{1+e^{-z}} \left(1 - \frac{1}{1+e^{-z}}\right)$

$= g(z)\,(1 - g(z))$ ←

$= a\,(1-a)$

$g'(z) = a(1-a)$

$z = 10$. $g(z) \approx 1$

$\frac{d}{dz} g(z) \approx 1\,(1-1) \approx 0$

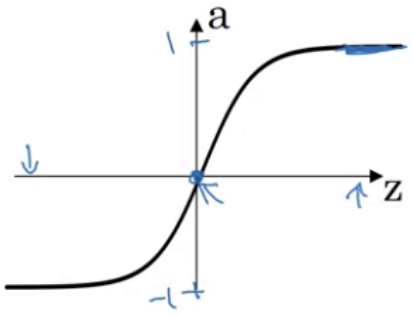
$z = -10$ $g(z) \approx 0$

$\frac{d}{dz} g(z) \approx 0 \cdot (1-0) \approx 0$

$z = 0$ $g(z) = \frac{1}{2}$

$\frac{d}{dz} g(z) = \frac{1}{2}\left(1 - \frac{1}{2}\right) = \frac{1}{4}$ Anc

Tanh activation function



$$g(z) = \tanh(z)$$

$= \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$

$g'(z) = \frac{d}{dz} g(z)$ = slope of $g(z)$ at z

$= 1 - (\tanh(z))^2$ ←

$a = g(z)$, $g'(z) = 1 - a^2$

$z = 10$ $\tanh(z) \approx 1$

$g'(z) \approx 0$

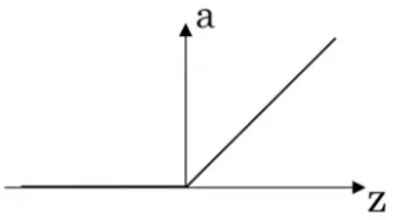
$z = -10$ $\tanh(z) \approx -1$

$g'(z) \approx 0$

$z = 0$ $\tanh(z) = 0$

$g'(z) = 1$

ReLU and Leaky ReLU

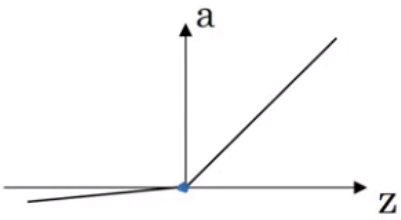


ReLU

$g(z) = \max(0, z)$

$g'(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases}$

$z = 0.0000\ldots0$



Leaky ReLU

$g(z) = \max(0.01z, z)$

$g'(z) = \begin{cases} 0.01 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases}$

Gradient descent for Neural Networks

np.sum(keepdims=True): prevent rank one arrays

Gradient descent for neural networks

Parameters: $w^{[1]}$ $(n^{[1]}, n^{[0]})$, $b^{[1]}$ $(n^{[1]}, 1)$, $w^{[2]}$ $(n^{[2]}, n^{[1]})$, $b^{[2]}$ $(n^{[2]}, 1)$

$n_x = n^{[0]}$, $n^{[1]}$, $n^{[2]} = 1$

Cost function: $J(w^{[1]}, b^{[1]}, w^{[2]}, b^{[2]}) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}, y)$

($w^{[2]}$, $b^{[2]}$; $\hat{y} = a^{[2]}$)

Gradient descent:

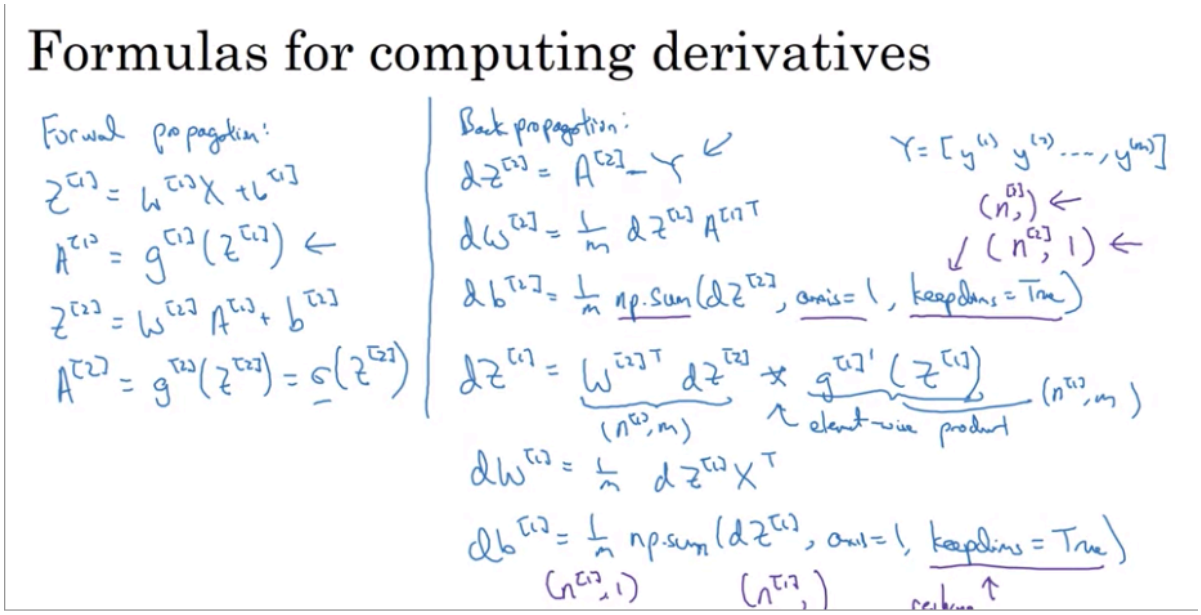
Repeat {

Compute predicts $(\hat{y}^{(i)}, i = 1 \ldots m)$

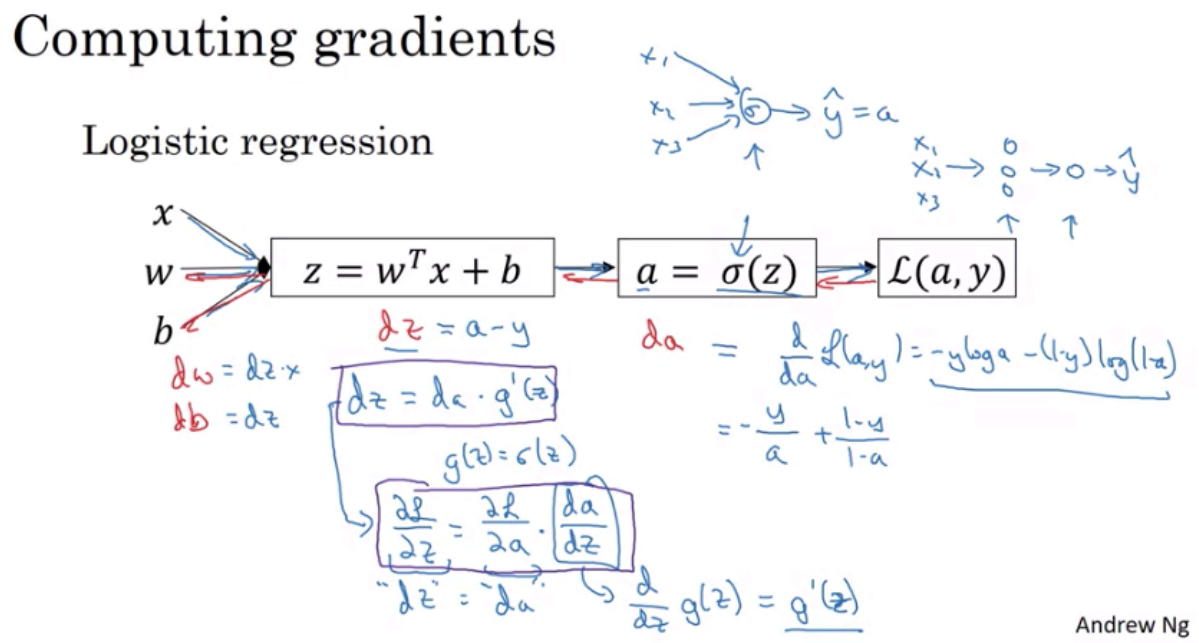
$dw^{[1]} = \frac{\partial J}{\partial w^{[1]}}$, $db^{[1]} = \frac{\partial J}{\partial b^{[1]}}$, ...

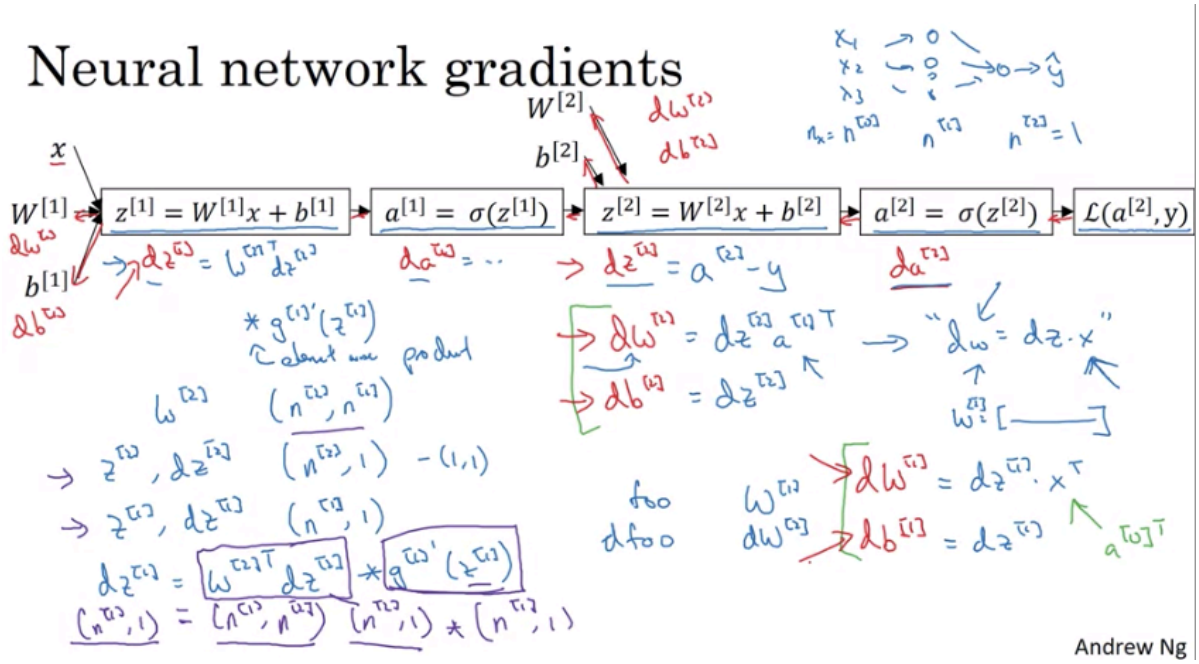
$w^{[1]} := w^{[1]} - \alpha\, dw^{[1]}$

$b^{[1]} := b^{[1]} - \alpha\, db^{[1]}$



Backpropagation intuition





https://medium.com/@pdquant/all-the-backpropagation-derivatives-d5275f727f60

Summary of gradient descent

$$dz^{[2]} = a^{[2]} - y$$
$$dW^{[2]} = dz^{[2]}a^{[1]T}$$
$$db^{[2]} = dz^{[2]}$$
$$dz^{[1]} = W^{[2]T}dz^{[2]} * g^{[1]\prime}(z^{[1]})$$
$$dW^{[1]} = dz^{[1]}x^{T}$$
$$db^{[1]} = dz^{[1]}$$

$$dZ^{[2]} = A^{[2]} - Y$$
$$dW^{[2]} = \frac{1}{m}dZ^{[2]}A^{[1]T}$$
$$db^{[2]} = \frac{1}{m}np.sum(dZ^{[2]}, axis = 1, keepdims = True)$$
$$dZ^{[1]} = W^{[2]T}dZ^{[2]} * g^{[1]\prime}(Z^{[1]})$$
$$dW^{[1]} = \frac{1}{m}dZ^{[1]}X^{T}$$
$$db^{[1]} = \frac{1}{m}np.sum(dZ^{[1]}, axis = 1, keepdims = True)$$

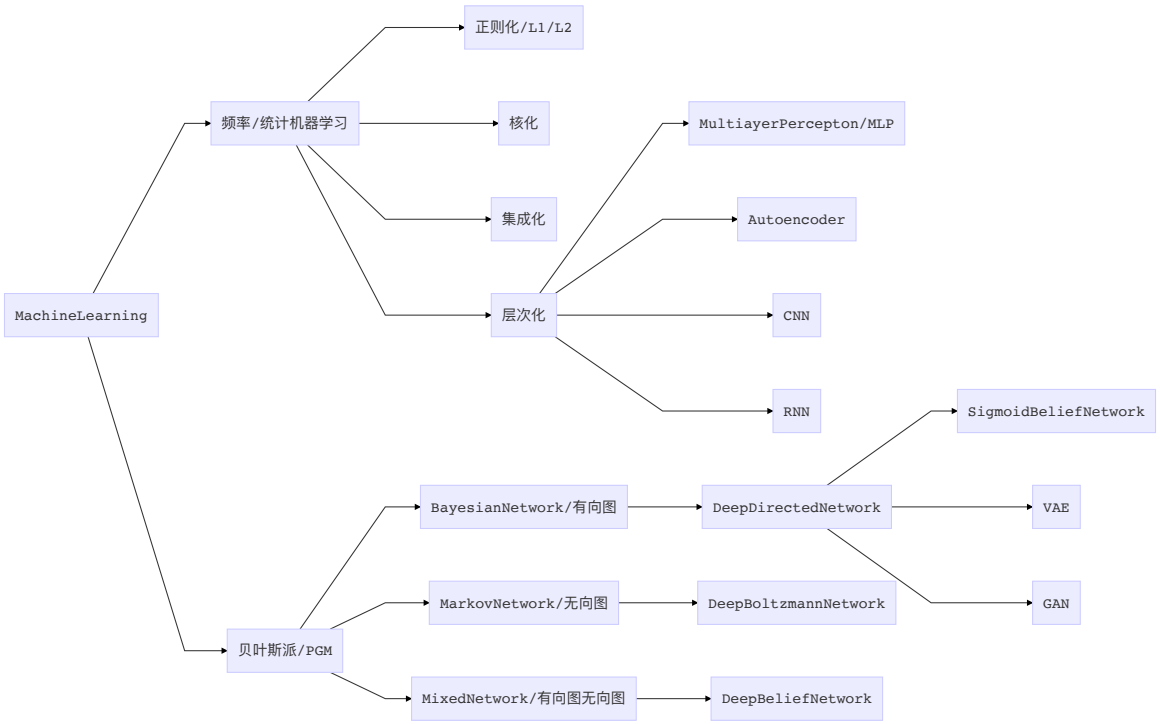
https://www.coursera.org/learn/neural-networks-deep-learning/discussions/weeks/3/threads/a38VuhyMEei5zw6yFhWyOg

Random Initialization

LR can be initialized as 0, but not neural network

Supplement

基础



统计机器学习

- 正则化：Loss Function + regularizer(L1/L2)
- 核化: Kernel SVM
- 集成化: AdaBoost, RandomForest
- 层次化: Neural Network/Deep Neural Network
  - MLP(Mutilayer Perceptron)
  - Autoencoder
  - CNN
  - RNN

贝叶斯派

- BayesianNetwork ⇒ Deep Directed Network
  - Sigmoid Belief Network
  - Variational Autoencoder(VAE)

    - GAN
- Markov Network ⇒ Deep Boltzmann Network
- Mixed Netowork ⇒ Deep Belief Network

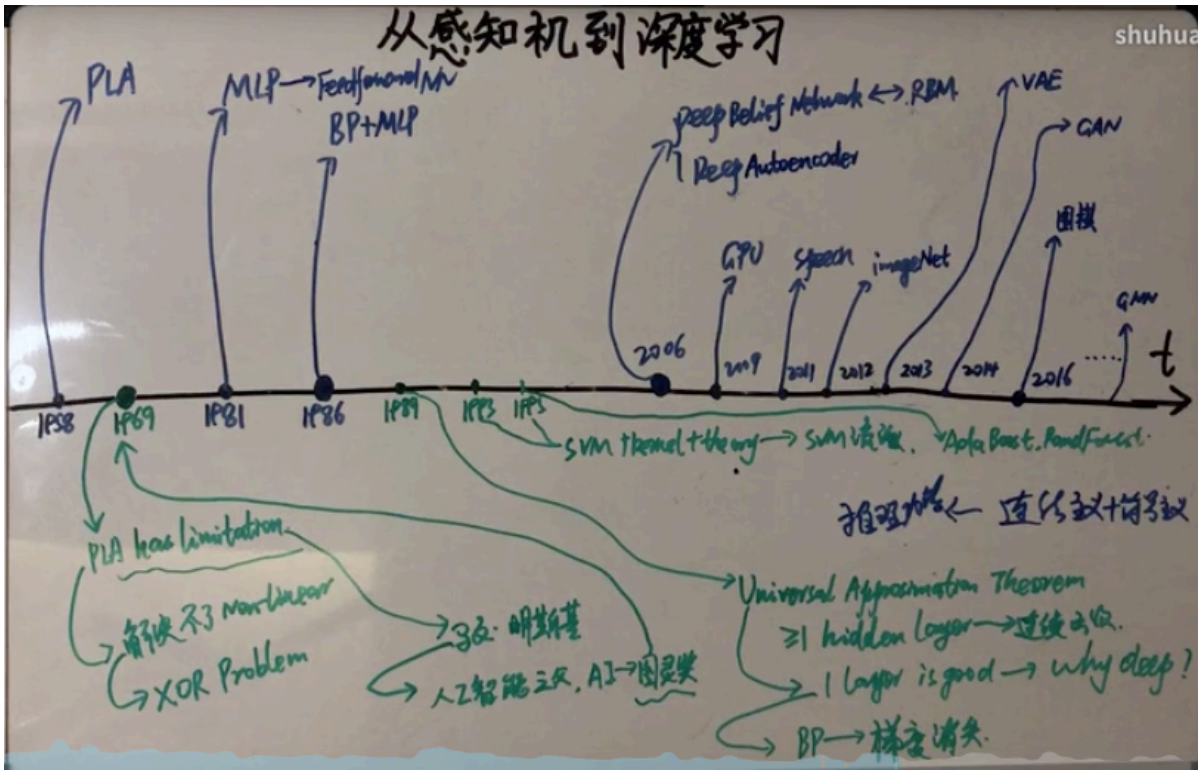
> 上述都是：Deep Generative Model

狭义的DeepLearning：Deep Neural Network

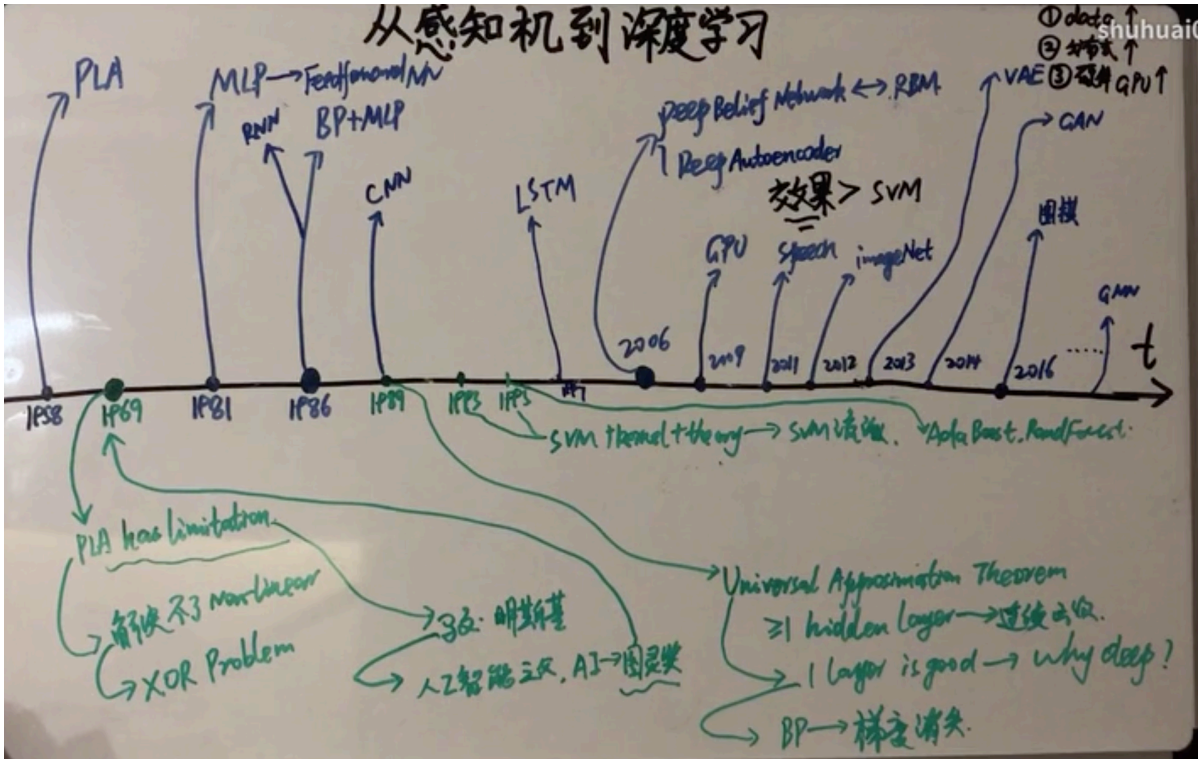
其实应该包括：

1. Deep Neural Network
2. Deep Generative Model（当层次非常多，推断非常困难）

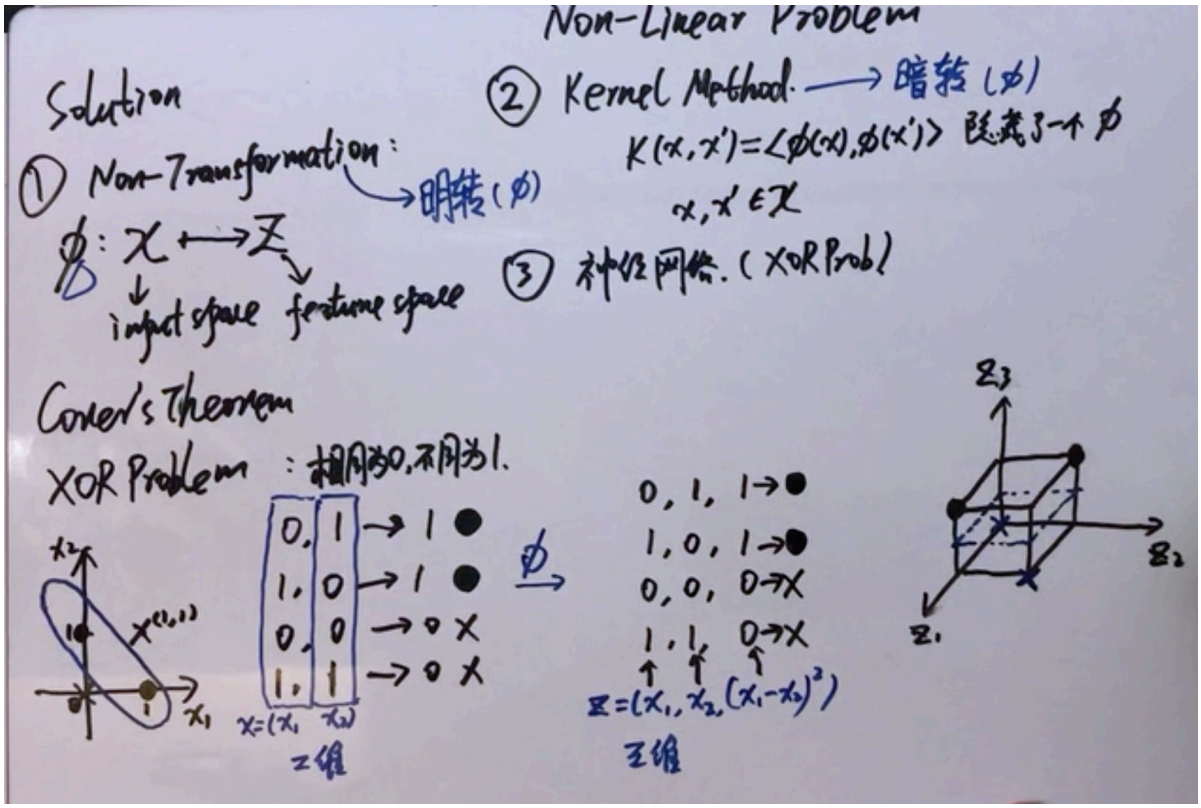
时间线



1. 深度学习的理论在2006年已经成型，直到现在，理论并没有根本性突破。
2. 为什么take off
    1. data
    2. 分布式
    3. 硬件 GPU
    4. 效果



Non-Linear Problem



Neural Network

$x, x' \in \mathcal{X}$

③ 神经网络. (XOR Prob)

XOR ⊕ 复合运算

OR ∨ AND ∧ NOT ¬ 基本运算

XOR → 相同为0, 不同为1

$x_1 \oplus x_2 = (\neg x_1 \wedge x_2) \vee (x_1 \wedge \neg x_2)$

x_1 XOR x_2 = {1,0 ①; 0,1 ②}

= ① ∨ ②

= $(x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)$

OR: 包含

1, 0 → 1

0, 1 → 1

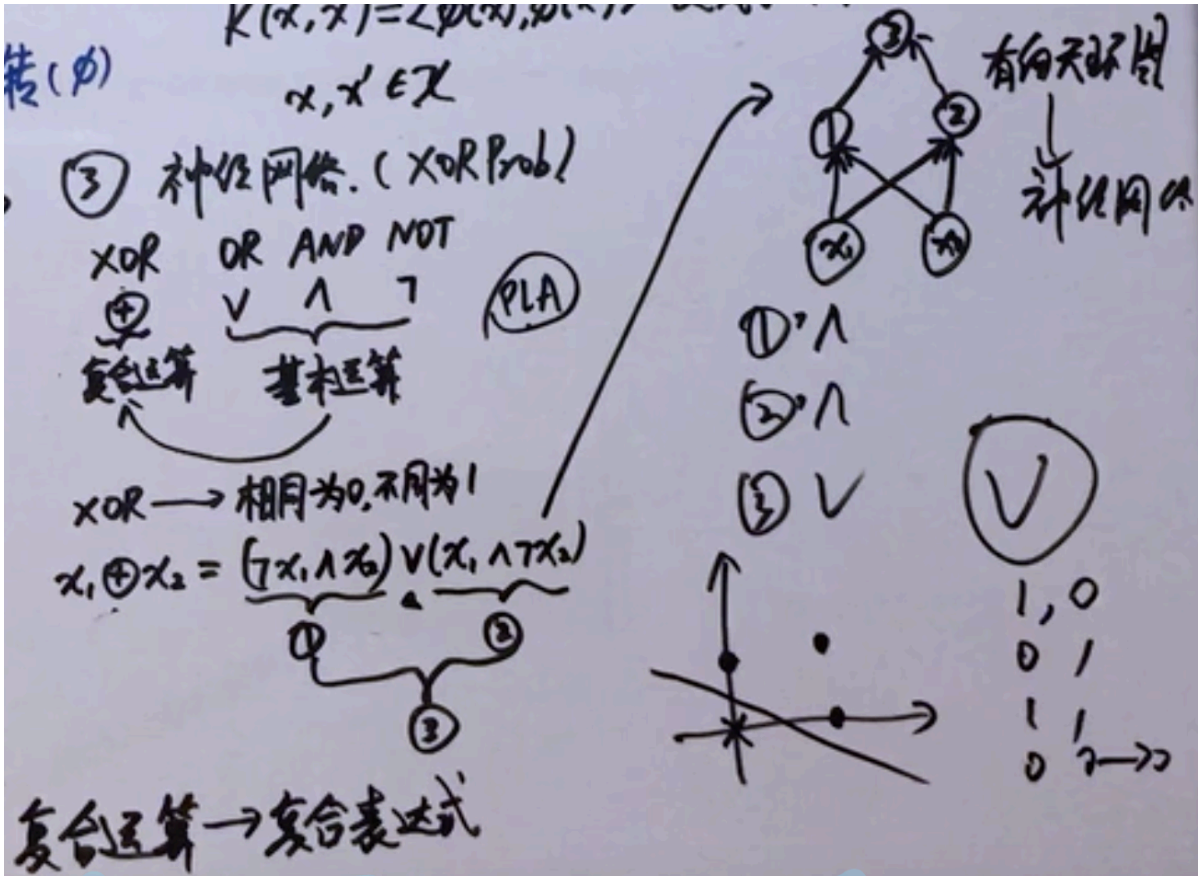
1, 1 → 1

AND 多者

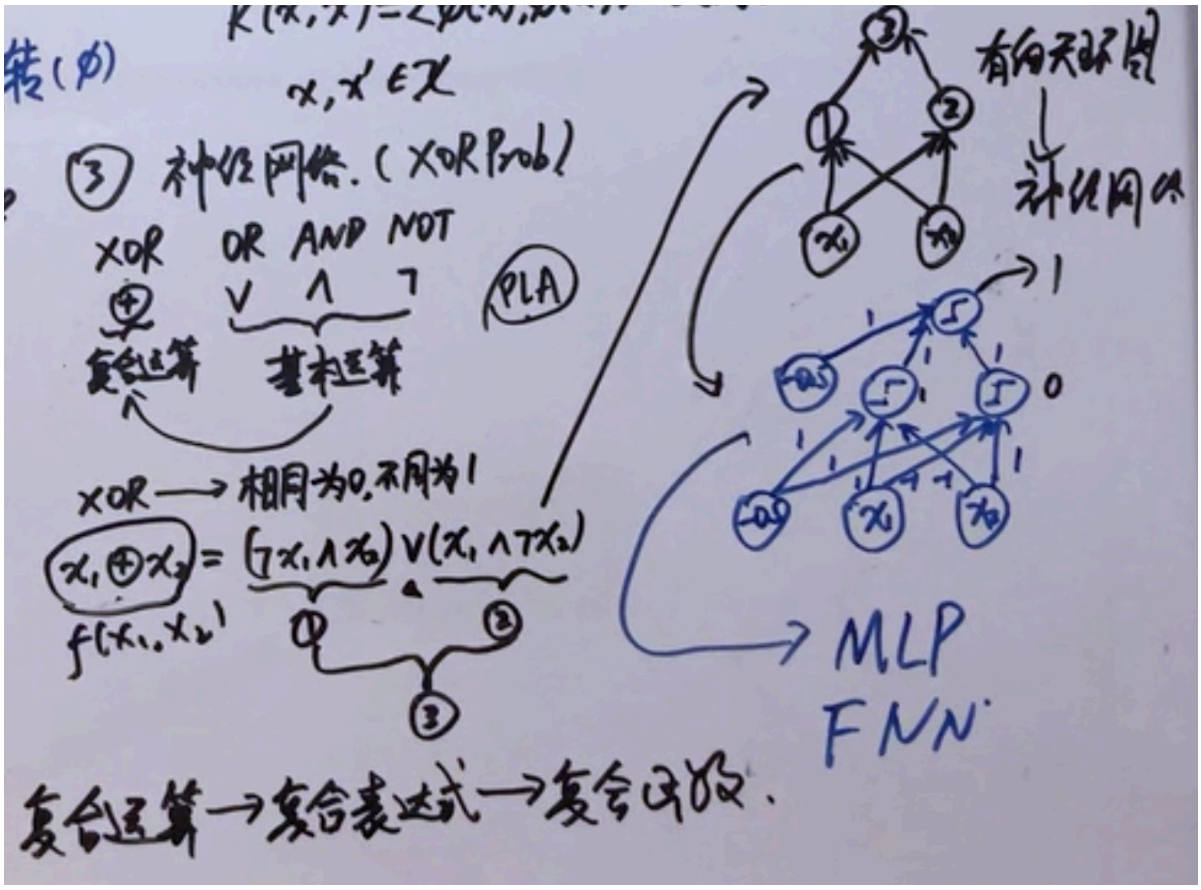
1, 1 → 1

NOT

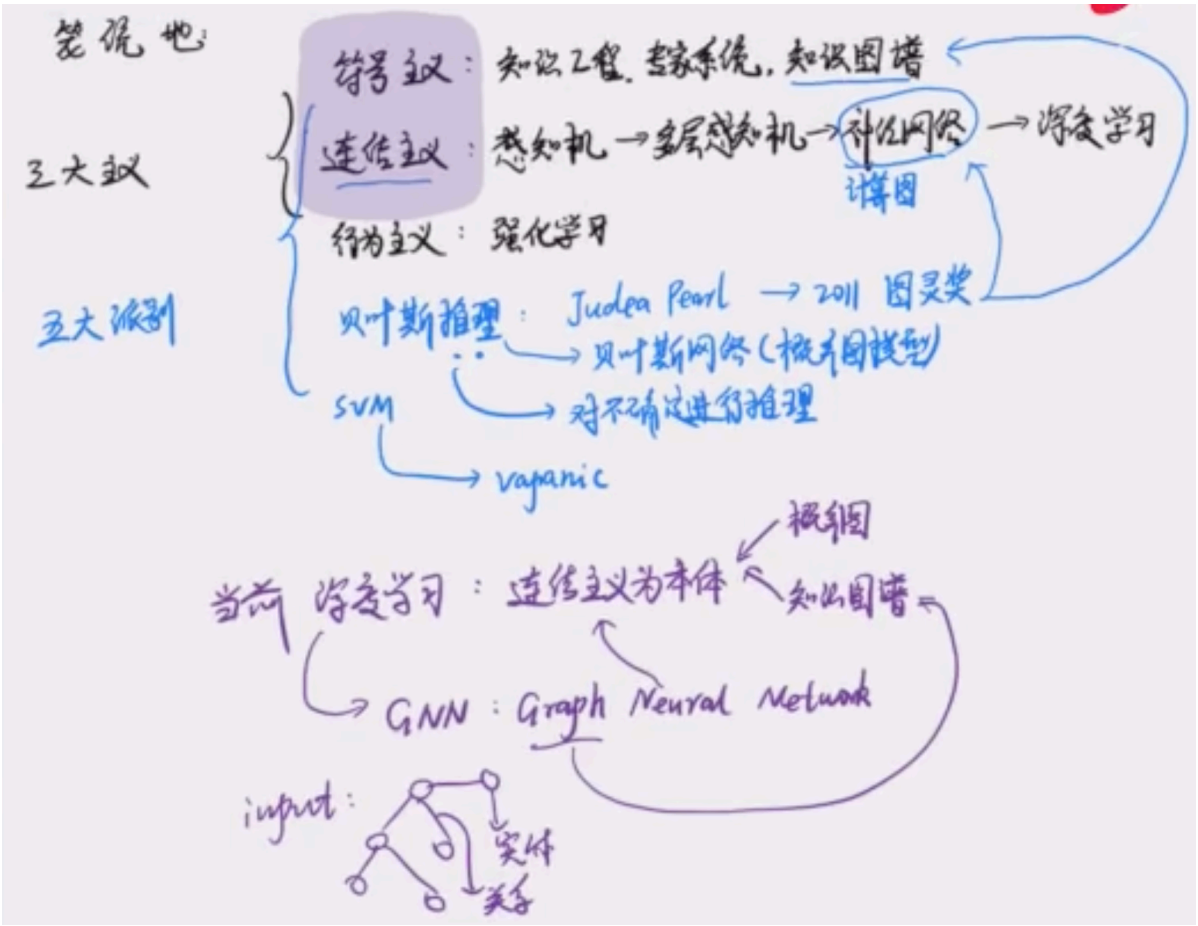
¬0 → 1



符合运算-》复合表达式-〉复合函数



人工智能 两大阵营 三大主义


简单地
三大主义
五大派别
符号主义：知识工程，专家系统，知识图谱
连结主义：感知机→多层感知机→神经网络→深度学习
计算图
行为主义：强化学习
贝叶斯推理：Judea Pearl → 2011 图灵奖
贝叶斯网络（概率图模型）
对不确定性进行推理
SVM
vapanic
概率图
知识图谱
当前 深度学习：连结主义为本体
GNN：Graph Neural Network
input：
实体
关系