Introduction of Machine Learning

No.
Date

幫機器找到好的 model 和 loss function

人工智慧 → 目標：讓機器同人一樣聰明

機器學習 → 手段：寫程式讓機器具學習能力

深度學習 → 機器學習的一種

如何賦予機器智能？

hand-crafted rule：用程式事先設定 ⇒ 無法超越人類

Machine Learning：Looking for a function that fits the input datas.

Steps:

1. prepare a "Model" (a set of functions)
2. Give "Training Data" ⇒ the AI judge the functions in the set according to the Training Data
3. test whether the AI can correctly work

⇒ Supervised learning

Learning Map

learning from teacher vs. learning from critics

Reinforcement learning：無資料 ⇒ 連普訊機器動作完成後結果好壞

Supervised Learning

Regression ⇒ 輸出是數值

Classification
- Binary：二元分類 ⇒ Yes or No
- Multi-class：將資料分為多類

⇒ models
- Linear Function：線性函數
- Non-linear Function ⇒ more complex

ex: Deep learning, SVM, decision tree, K-NN

↳ Structure learning ⇒ 輸出有結構性的資料

※ Training Data：Input/Output pairs ⇒ for target function

⇒ Output = label

Semi-supervised Learning：Data → part labeled / part unlabeled

Transfer Learning：use old models for similar problems ⇒ 遷移學習

Unsupervised Learning：machine self-learning only outputs, no inputs

scenario task method

No.
Date : :

Regression：輸出數值(scalar)

Ex：寶可夢進化cp值預測

⇒ input：寶可夢(x) output：進化後cp值(y)

Step 1：Model = a set of function

a) $y = b + w \cdot x_{cp}$ ⇒ linear model：$y = b + \sum w_i x_i$

x_i → feature：input x 的各種數值

w_i → weight：權重

b → bias：常數

Step 2：Goodness of function ⇒ Loss function (L)

→ input：a function ; output：how bad it is

→ $L(f) = L(w, b) = \sum_{n} (\hat{y}^n - (b + w \cdot x_{cp}^n))^2$ ⇒ 估測誤差

⇒ 這次 loss function 僅考慮一種 ⇒ 圖示以 w、b 為兩軸 L(w,b) 第三軸

Step 3：Best Function ⇒ Gradient Descent $\nabla L(\theta) = \begin{bmatrix} \partial L(w_1)/\partial w_1 \\ \partial L(w_2)/\partial w_2 \end{bmatrix}$

→ 由 w 判斷好壞：y軸 L(w)；x軸 w　正：w↓

→ random 選一 w_0 → compute $\frac{dL}{dw}|_{w=w_0}$ ⇒ 負：w↑

→ $w_{new} = w_n \pm \eta \frac{dL}{dw}|_{w=w_n}$ → η (eta)：learning rate = 加減幅度常數

→ 多次值行 ⇒ 到 local minimum ⇒ 在 regression 就是 global

⇒ 考慮 w 以為第二個因素，同理 ⇒ 偏微分 L 分別微分

$L(w, b) = \sum_{n=1}^{10} (\hat{y}^n - (b + w \cdot x_{cp}^n))^2$

$\frac{\partial L}{\partial w} = \sum_{n=1}^{10} 2(\hat{y}^n - (b + w \cdot x_{cp}^n))(-x_{cp}^n)$；$\frac{\partial L}{\partial b} = \sum_{n=1}^{10} 2(\hat{y}^n - (b + w \cdot x_{cp}^n))(-1)$

優化：

1. more complex Model：

error	training	testing
一次	31.9	35.0
二次	15.4	18.4
三次	15.3	18.1
四次	14.9	28.8
五次	12.8	232.5

⇒ 更複雜 ⇒ training error ↓

testing error 不一定

overfitting ⇒ model 大複雜 error 爆炸

underfitting ⇒ model 大簡單 error 高

如何在現實中 testing data Error 更小？

⇒ N-fold validation：將 training data 分為 N 份，每次選取一份作為 validation set（模擬 testing data），其他作為 training set

→ 將所有 model 跑過並輸出 Error 再取 average error

→ 取 av. error 最小的嘗試 public testing data

→ 這種結果較能應在其他 data 上的 error

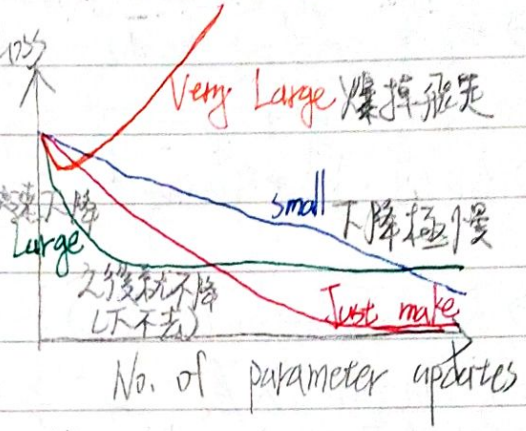
* Gradient Descent tips:

tip 1：tuning your learning rate

[太小：太慢

[太大：無法走到最低點

↑畫前幾次參數 update 下降趨勢圖：



tip 2：Adaptive Learning Rates → 一系列：Ada...

Maybe → Reduce learning rate by some factor every epochs.

∵ Learning rate cannot be one-size-fit-all

↑ Give different parameters different learning rates

Ex:

Adagrad：$w^{t+1} = w^t - \eta^t g^t \rightarrow w^{t+1} = w^t - \frac{\eta^t}{\sigma^t} g^t$

⇒ $\sigma^t = \sqrt{\frac{1}{t+1}\sum_{i=0}^{t}(g^i)^2} = \sqrt{[(\sigma^{t-1})^2(t)) + (g^t)^2]\frac{1}{t+1}}$ ∧ $\eta^t = \frac{\eta}{\sqrt{t+1}}$ ⇒ $\frac{\eta^t}{\sigma^t} = \frac{\eta}{\sqrt{\cdots}}$

but $w^{t+1} = w^t - \frac{\eta}{\sqrt{\cdots}} g^t$ gradient ↑ step ↑ → gradient ↑ step ↓ why?

Intuitive Reason：how suprise ⇒ 讓參數反差

∵ Gradient descent 認為微分↑與最低點距離↑ ⇒ 只在乎一次式微分成立

∴ best step = $\frac{|\text{一次微分}|}{\text{二次微分}}$ → Adagrad：$w^{t+1} \leftarrow w^t - \frac{\eta g^t}{\sqrt{\sum_{i=0}^{t}(g^i)^2}}$ 一次微分 反應 二次微分大小

→ 為何不直接計算二次微分？

數據量↑可能耗時太多

No.
Date : :

搜集更多資料

2. hidden factor
→ 使用 δ (delta)：species 的 true v false → 不同物種決定

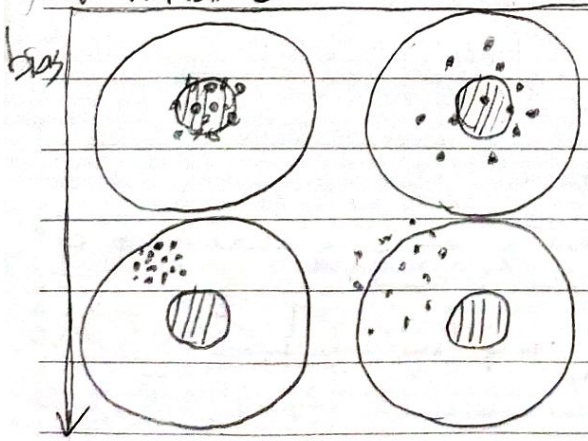
→ 考慮 weight、height、Hp ... → redesign Model
加入 $y = y' + Hp$ 次式 + weight 二次式 + height 二次式
→ 結果：overfitting！

→ 不考慮 bias？bias 只影響 function 的平移，不影響平滑程度

→ Regularization (Step 2)
Loss function 改為：$L = \sum_n (\hat{y}^n - (b + \sum w_i x_i))^2 + \lambda \sum (w_i)^2$ ← 常數
→ 期待更平滑 function → 結果波動更小，較不受 noise 影響
→ 入 α training error，更考慮 function 本值

Where does the error come from [due to bias：偏心準
                                [due to variance：精準度

$\hat{y} = f$ (實可夢) → only Niantic knows
but 可由 training data find f^* → 最佳解 (近似於)



→ Simpler model is less influenced by the sampled data → lower variance
→ 極端例子：$f = c$ 常數 → 完全不受 data 影響

training data loss function 大
→ large bias → 重新找 function
testing data loss function 大
→ large variance → More data

v → regularization → 強制曲線平滑 但 可能傷害 bias
∴ 利用 weigh 決定 regularization 程度

Stochastic Gradient Descent

⇒ Loss for only one example: $w^1 = w^0 - \eta \nabla L^n(w^0)$

⇒ 只隨機考慮一个取樣點

⇒ 在每个 example 都更新一次參數 ⇒ 20 times faster

(天下武功，唯快不破 XD)

(*Gradient Descent: $w^{new} = w^{old} - \eta \nabla L(w^{old})$, Loss = sum over all training example

$L^n = (\hat{y}^n - (b + \sum w x_i^n))^2$, $L = \sum_n L^n$)

tip3: Feature Scaling: 希望讓不同輸入元素 (feature) 分布差不多 (scale)

⇒ 讓各个 Feature 的 Loss surface 更接近正圓

≒ 數據標準化 ⇒ variation = 0

Gradient Descent Theory

*Warning of math:

If 取一隨機初始值 ⇒ can easily find the point with the smallest value (nearby.)

but how?

→ Taylor Series 泰勒級數 (泰勒展開式):

Let $h(x)$ be any function infinitely differentiable around $x = x_0$

$h(x) = \sum_{k=0}^{\infty} \frac{h^{(k)}(x_0)}{k!}(x-x_0)^k$ ⇒ when $x \to x_0$ ⇒ $h(x) \doteq h(x_0) + h'(x_0)(x-x_0)$

(高次項都 → 0)

→ Multivariable Taylor Series: if $x, y \to x_0, y_0$

$h(x,y) \doteq h(x_0,y_0) + \frac{\partial h(x_0,y_0)}{\partial x}(x-x_0) + \frac{\partial h(x_0,y_0)}{\partial y}(y-y_0)$

∴ 若範圍夠小，可用泰勒展開式簡化 loss function

$L(\theta) \doteq \underbrace{L(a,b)}_{令為 s} + \underbrace{\frac{\partial L(a,b)}{\partial \theta_1}}_{令為 u}(\theta_1 - a) + \underbrace{\frac{\partial L(a,b)}{\partial \theta_2}}_{令為 v}(\theta_2 - b) = s + u(\theta_1 - a) + v(\theta_2 - b)$

$\Delta\theta_1 \quad \Delta\theta_2$

⇒ 求 $u\Delta\theta_1 + v\Delta\theta_2$ 之 min, $\Delta\theta_1^2 + \Delta\theta_2^2 \le d^2$

$\Rightarrow \begin{bmatrix}\Delta\theta_1 \\ \Delta\theta_2\end{bmatrix} = -\eta \begin{bmatrix}a \\ b\end{bmatrix} \Rightarrow \begin{bmatrix}\theta_1 \\ \theta_2\end{bmatrix} = \begin{bmatrix}a \\ b\end{bmatrix} - \eta \begin{bmatrix}u \\ v\end{bmatrix}$

↳ 若沒有就不成立

No.
Date

More Limitation of Gradient Descent
1. Stuck at local minima
2. Stuck at saddle point ⇒ $\partial L/\partial w = 0$ 實作時可能造成之停止點
3. Stuck at a point that's very slow at the plateau ⇒ $\partial L/\partial w \approx 0$

Classification:
Ex: 寶可夢分類
input: 一隻寶可夢 → 轉為可輸入數值 vector
output: 寶可夢種類 (x 18)
方法1: 用 regression 硬爆? if binary 極端數值 (離二分界太遠者) 會造成分界並非最佳 ⇒ 會嘗試調整分界平衡了極端數值
if multipul ⇒ 極難做到 ∴ 並非理想方法
→ Ideal Alternatives
Function (Model):
Binary: a Model contains if, else

Loss function: $L(f) = \sum_n \delta(f(x^n) \neq \hat{y}^n)$ ⇒ 判斷是否正確
Find the best function: ex: perceptron, SVM
→ Two classes ⇒ Binary: Class 1、Class 2
Given a x: $P(C_1|x) = \frac{P(x|C_1)P(C_1)}{P(x|C_1)P(C_1) + P(x|C_2)P(C_2)}$ ⇒ 看 x 比較可能來自哪個 Class
= Generative Model: $P(x) = P(x|C_1)P(C_1) + P(x|C_2)P(C_2)$
→ Feature: 考慮 Df 跟 sp Df ⇒ 畫出分布圖
⇒ 假設點都是 Gaussian Distribution (常態分布) 取樣
⇒ 計算 μ (= mean 平均) 及 Σ (分布狀況 = 聚集 ↔ 分散)
⇒ 即可算新點出現機率 ↳ covariance 變異數

To find best function: Maximum Likelihood

$L(\mu, \Sigma) = f_{\mu,\Sigma}(x^1) f_{\mu,\Sigma}(x^2) \cdots f_{\mu,\Sigma}(x^{79})$ 次系

$f_{\mu,\Sigma}(x) = \left(\frac{1}{(2\pi)^{D/2}}\right)\left(\frac{1}{|\Sigma|^{1/2}}\right) \exp\left\{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right\}$

$\mu^*, \Sigma^* = \arg\max_{\mu,\Sigma} L(\mu, \Sigma)$

→ Class 1: water $\mu^1 = \begin{bmatrix} 75.0 \\ 71.3 \end{bmatrix}$ $\Sigma^1 = \begin{bmatrix} 874 & 329 \\ 329 & 929 \end{bmatrix}$

Class 2: normal $\mu^2 = \begin{bmatrix} 55.6 \\ 59.8 \end{bmatrix}$ $\Sigma^2 = \begin{bmatrix} 841 & 422 \\ 422 & 685 \end{bmatrix}$

→ Now we can do classification: if $P(C_1|x) > 0.5$ ⇒ x belongs to class 1

results ⇒ testing data: 47% accuracy

All? total, hp, atk, sp atk, de, sp de, speed (7 features)

μ^1, μ^2: 7-dim vector; Σ^1, Σ^2: 7×7 matrices ⇒ 54% accuracy

於是嘗試 ⇒ 讓不 class 有相同 Σ ∴ μ^1, μ^2 相同, $\Sigma = \frac{79}{140}\Sigma^1 + \frac{61}{140}\Sigma^2$

boundary 由曲線 → linear ⇒ accuracy: 54% → 73%

總結 Three steps:

1. Function Set (Model)
2. Goodness of a function: The mean μ and covariance Σ that maximize the likelihood (the probability of generating data)
3. Find the best function ↗ easy XD

→ Probability Distribution (你喜歡的就好 XD)

⇒ binary features: Bernoulli distribution

⇒ all the dimensions are independent: Naive Bayes Classifier

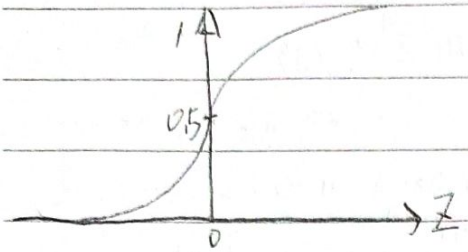
→ Posterior Probability:

$$P(C_1|x) = \frac{P(x|C_1)P(C_1)}{P(x|C_1)P(C_1) + P(x|C_2)P(C_2)} \overset{\text{分子分母同除分子}}{=} \frac{1}{1 + \frac{P(x|C_2)P(C_2)}{P(x|C_1)P(C_1)}} \overset{z = \ln\frac{P(x|C_1)P(C_1)}{P(x|C_2)P(C_2)}}{=} \frac{1}{1+\exp(-z)} = \sigma(z)$$

Sigmoid function

No.
Date : :

* Sigmoid function $\sigma(z)$: if $x \to \infty$, $y \to 1$; $x \to -\infty$, $y \to 0$



$$z = \ln \frac{P(x|C_1)P(C_1)}{P(x|C_2)P(C_2)} = \ln \frac{P(x|C_1)}{P(x|C_2)} + \ln \frac{P(C_1)}{P(C_2)}$$

$$= \ln \frac{\frac{1}{(2\pi)^{D/2}} \frac{1}{|\Sigma^1|^{1/2}} \exp\{-\frac{1}{2}(x-\mu^1)^T(\Sigma^1)^{-1}(x-\mu^1)\}}{\frac{1}{(2\pi)^{D/2}} \frac{1}{|\Sigma^2|^{1/2}} \exp\{-\frac{1}{2}(x-\mu^2)^T(\Sigma^2)^{-1}(x-\mu^2)\}} + \frac{\frac{N_1}{N_1+N_2}}{\frac{N_2}{N_1+N_2}}$$

$$= \ln \frac{|\Sigma^2|^{1/2}}{|\Sigma^1|^{1/2}} - \frac{1}{2}[(x-\mu^1)^T(\Sigma^1)^{-1}(x-\mu^1) - (x-\mu^2)^T(\Sigma^2)^{-1}(x-\mu^2)] + \frac{N_1}{N_2}$$

$$\Rightarrow (x-\mu^1)^T(\Sigma^1)^{-1}(x-\mu^1) = x^T(\Sigma^1)^{-1}x - x^T(\Sigma^1)^{-1}\mu^1 - (\mu^1)^T(\Sigma^1)^{-1}x + (\mu^1)^T(\Sigma^1)^{-1}\mu^1$$

$$= x^T(\Sigma^1)^{-1}x - 2(\mu^1)^T(\Sigma^1)^{-1}x + (\mu^1)^T(\Sigma^1)^{-1}\mu^1$$

$$\Rightarrow (x-\mu^2)^T(\Sigma^2)^{-1}(x-\mu^2) = x^T(\Sigma^2)^{-1}x - 2(\mu^2)^T(\Sigma^2)^{-1}x + (\mu^2)^T(\Sigma^2)^{-1}\mu^2$$

$$\therefore z = \ln \frac{|\Sigma^2|^{1/2}}{|\Sigma^1|^{1/2}} - \frac{1}{2}x^T(\Sigma^1)^{-1}x + (\mu^1)^T(\Sigma^1)^{-1}x - \frac{1}{2}(\mu^1)^T(\Sigma^1)^{-1}\mu^1 + \frac{1}{2}x^T(\Sigma^2)^{-1}x$$
$$- (\mu^2)^T(\Sigma^2)^{-1}x + \frac{1}{2}(\mu^2)^T(\Sigma^2)^{-1}\mu^2 + \ln \frac{N_1}{N_2}$$

if $\Sigma_1 = \Sigma_2 = \Sigma$

$$z = \underbrace{(\mu^1 - \mu^2)^T\Sigma^{-1}}_{w^T}x - \underbrace{\frac{1}{2}\Sigma^{-1}(\mu^{1T}\mu^1 - \mu^{2T}\mu^2) + \ln \frac{N_1}{N_2}}_{b} \Rightarrow P(C_1|x) = \sigma(w \cdot x + b)$$

∴ In generative model, we estimate N_1, N_2, μ^1, μ^2, Σ ⇒ we have w and b

So how about directly finding w and b ?

Logistic Regression vs. Linear Regression

Step 1 :

$f_{w,b}(x) = \sigma(\sum_i w_i x_i + b)$ — output : 0 ~ 1

$f_{w,b}(x) = \sum_i w_i x_i + b$ — output : any value

Step 2 : Training data ⇒ $\{x^n, \hat{y}^n\}$

$\hat{y}^n$: 1 for class 1, 0 for class 2

$\hat{y}^n$: a real number

$L(f) = \sum_n l(f(x^n), \hat{y}^n)$

$L(f) = \frac{1}{2}\sum_n (f(x^n) - \hat{y}^n)^2$

⇒ $w^*, b^* = \arg\max_{w,b} L(w,b) = w^*, b^* = \arg\min_{w,b} -\ln L(w,b)$

No.
Date

* Cross entropy: $L(f(x^n), \hat{y}^n) = -[\hat{y}^n \ln f(x^n) + (1-\hat{y}^n) \ln(1-f(x^n))]$

Q: why not also use square error as linear regression?

* $f_{w,b}(x) = \sigma(z) = \frac{1}{1+\exp(-z)}$ $z = w \cdot x + b = \sum_i w_i x_i + b$

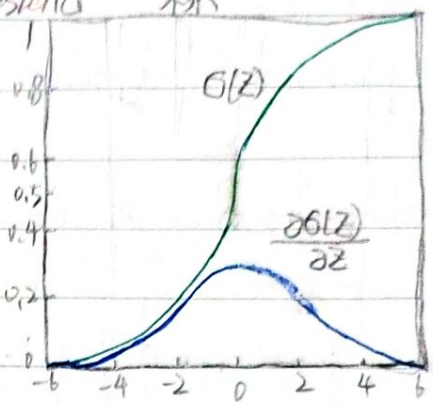
Step 3: 其實跟 linear regression 的 Gradient descend 一樣

$$-\frac{\partial \ln L(w,b)}{\partial w_i} = \sum_n -[\hat{y}^n \frac{\partial \ln f_{w,b}(x^n)}{\partial w_i} + (1-\hat{y}^n)\frac{\partial \ln(1-f_{w,b}(x^n))}{\partial w_i}]$$

$$\frac{\partial \ln f_{w,b}(x)}{\partial w_i} = \frac{\partial \ln f_{w,b}(x)}{\partial z}\frac{\partial z}{\partial w_i} \quad \frac{\partial \ln \sigma(z)}{\partial z} = \frac{1}{\sigma(z)}\frac{\partial \sigma(z)}{\partial z}$$

$$\Rightarrow \frac{\partial \ln(1-f_{w,b}(x))}{\partial w_i} = \frac{\partial \ln(1-f_{w,b}(x))}{\partial z}\frac{\partial z}{\partial w_i} \quad \cdots = \frac{1}{\sigma(z)}\sigma(z)(1-\sigma(z))$$

$$\frac{\partial \ln(1-\sigma(z))}{\partial z} = \frac{-1}{1-\sigma(z)}\frac{\partial \sigma(z)}{\partial z} = \frac{-1}{1-\sigma(z)}\sigma(z)(1-\sigma(z))$$



$$\Rightarrow \sum_n -[\hat{y}^n(1-f_{w,b}(x^n))x_i^n - (1-\hat{y}^n)f_{w,b}(x^n)x_i^n] = \sum_n -[\hat{y}^n - \hat{y}^n f_{w,b}(x^n) - f_{w,b}(x^n) + \hat{y}^n f_{w,b}(x^n)]x_i^n$$

$$= \sum_n -(\hat{y}^n - f_{w,b}(x^n))x_i^n$$ Larger difference, larger update

∴ Step 3 => logistic 跟 linear 一樣，重點在 Step 2

Logistic Regression + Square Error

Step 1: $f_{w,b}(x) = \sigma(\sum_i w_i x_i + b)$

Step 2: Training data: $(x^n, \hat{y}^n)$, $\hat{y}^n$: 1 for class 1, 0 for class 2

$$L(f) = \frac{1}{2}\sum_n (f_{w,b}(x^n) - \hat{y}^n)^2$$

Step 3: $\frac{\partial (f_{w,b}(x)-\hat{y})^2}{\partial w_i} = 2(f_{w,b}(x)-\hat{y})\frac{\partial f_{w,b}(x)}{\partial z}\frac{\partial z}{\partial w_i}$

$\hat{y}^n = 0$ $\quad = 2(f_{w,b}(x)-\hat{y})f_{w,b}(x)(1-f_{w,b}(x))x_i$

if $f_{w,b}(x^n) = 1$ (far from target) or $= 0$ (close to target) $\Rightarrow \frac{\partial L}{\partial w_i} = 0$

=> 可能剛好在離目標很遠的地方卡住 => Cross Entropy 無此問題

Generative vs. Discriminative => 當 Data 大量，需直接從 Data 判斷

→ Discriminative 精準度更高 but 若 Data 不夠，有範本還未被列舉出，or 訓練 Data 中有 noise or 訓練 Data 有多種形式

→ Generative 可能精準度更高 (有腦補功能 XD)

No.
Date

∴ Generative ⇒ 機率性的; Discriminative ⇒ 完全根據 Data 訓練

※ Benefit of generative model

- with the assumption of probability distribution, less training data is needed.
- with the assumption of probability distribution, more robust to the noise.
- priors and class-dependent probabilities can be estimated from different sources. ※

Multi-class Classification

$z_i \rightarrow e^{z_i} \rightarrow \sum_{j=1}^{3} e^{z_j} \rightarrow y_i = e^{z_i} / \sum_{j=1}^{3} e^{z_j}$ ⇒ Softmax $\xrightarrow{\text{Cross Entropy}} -\sum_{j=1}^{3} \hat{y}_j \ln y_j \rightarrow \hat{y}$ (target)

⇒ If $x \in$ class1: $\hat{y} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$; $\in$ class2: $\hat{y} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$ ... 以此推類

Limitation of Logistic Regression

有4筆 data ⇒ class1: (0,0), (1,1) ⇒ 無法分類
class2: (0,1), (1,0)

用 Feature Transformation? (矩陣轉換) → Not always easy to find a good one ∴ 不聰明 (X)

∨ Cascading logistic regression models ⇒ 結合多個

⇒ 機器自己 transform (V)

∴ 將每個 logistic regression model 為 neuron

⇒ 多個 neuron 組在一起 = neural network

⇒ 就是 deep learning 阿

No.
Date

Deep learning

— Ups and downs ⇒ History

1958 Perceptron (linear model)

1969 Perception has limitation

1980's Multi-layer perception

⇒ Do not have significant difference from DNN today

1986 Backpropagation

⇒ Usually more than 3 hidden layers is not helpful

1989 1 hidden layer's "good enough", why deep?

2006 RBM initialization

2009 GPU 加速

2011 Start to be popular in speech recognition

2012 win ILSVRS image competition

Three Steps for Deep Learning (Actually ≒ Three Steps for Machine Learning)

Step 1: Neural Network

Neuron = Logistic Regression ⇒ Neuron 為後 content 之 neuron network

⇒ Different connection leads to different network

→ Network parameter θ = all weights and bias in the "neurons"

Ex: fully connected feedforward network

⇒ ≒ a function, input vector, output vector

⇒ Given network structure, a defined fuction set

= input layer + "hidden layers" + output layers ∴ Deep = Many hidden layers

Name (year)	→ AlexNet (2012)	→ VGG (2014)	→ GoogleNet (2014)	→ Residual Net (2015)
layers - error rate	8 - 16.4%	19 - 7.3%	22 - 6.7%	152 - 3.57%

not fully connected ↑ special structure

⇒ 運算其實 = 一連串 matrix operation ∧ 在 hidden layers 最後通過 softmax

Feature extractor replacing Feature engineering ∧ output layer = Multi-class Classifier

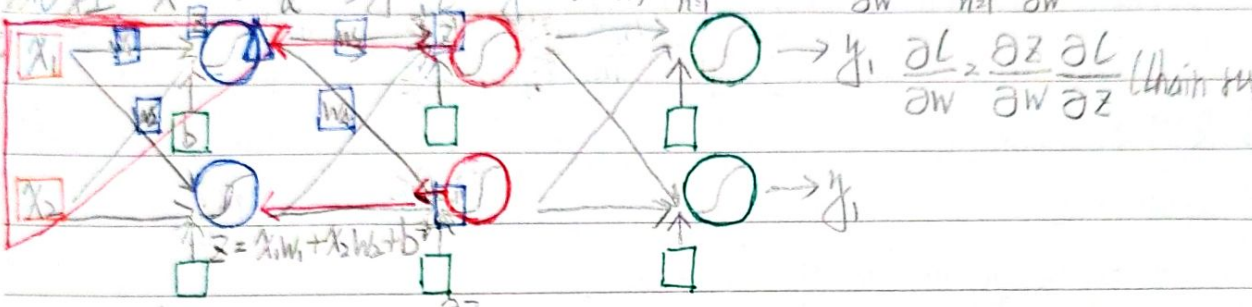
Step 2: Goodness of a function ⇒ Gradient descent

Step 3: Find the best function

No.
Date

(given enough hidden neurons)
but, any continuous fuction f can be realized by a network with one layer
∴ 提問 why "Deep" not "Fat" neural network?

Backpropagation: to compute the gradients efficiently (Same as Gradient Descent)
⇒ Chain Rule: 1. $y=g(x)$, $z=h(y)$ ∴ $\Delta x \to \Delta y \to \Delta z$ ⇒ $\frac{dz}{dx}=\frac{dz}{dy}\frac{dy}{dx}$
2. $x=g(s)$, $y=h(s)$, $z=k(x,y)$ ∴ $\Delta s \to \Delta x, \Delta y \to \Delta z$ ⇒ $\frac{dz}{ds}=\frac{\partial z}{\partial x}\frac{dx}{ds}+\frac{\partial z}{\partial y}\frac{dy}{ds}$
流程: $x^n \to NN \to y^n$ ⇒ $L(\theta)=\sum_{n=1}^{N} C^n(\theta)$ ⇒ $\frac{\partial L(\theta)}{\partial w}=\sum_{n=1}^{N}\frac{\partial C^n(\theta)}{\partial w}$



$\frac{\partial L}{\partial w}=\frac{\partial z}{\partial w}\frac{\partial L}{\partial z}$ (chain rule)

⇒ Forward pass: Compute $\frac{\partial z}{\partial w}$ for all parameters
Backward pass: Compute $\frac{\partial L}{\partial z}$ for all activation function inputs z
Forward pass: $\frac{\partial z}{\partial w_1}=x_1$ ∧ $\frac{\partial z}{\partial w_2}=x_2$ ⇒ The value of the input connected by the weight
= 所有 weight 之偏微分, 即為其前面連接那一值
∴ 要計算每个 weight 之偏微分, 只需 input, 然後計算每个 neuron 之 output

Backward pass: $\frac{\partial L}{\partial z}=\frac{\partial a}{\partial z}\frac{\partial L}{\partial a}=\frac{\partial z'}{\partial a}\frac{\partial L}{\partial z'}+\frac{\partial z''}{\partial a}\frac{\partial L}{\partial z''}$ .... (chain rule)
∴ $\frac{\partial L}{\partial z}=\sigma'(z)\left[w_3\frac{\partial L}{\partial z'}+w_4\frac{\partial L}{\partial z''}\right]$
⇒ 可想為 neuron 之 input = $\frac{\partial L}{\partial z'}$; $\frac{\partial L}{\partial z''}$, 分別乘上 w_3; w_4 後相加, 再乘上 $\sigma'(z)$, 即等於 $\frac{\partial L}{\partial z}$
⇒ $\sigma'(z)$ is a constant because z is already determined in the forward pass.
Case 1. Output layer ⇒ output = network 的 ∴ 計算 $\frac{\partial L}{\partial z}=\frac{\partial y_1}{\partial z}\frac{\partial L}{\partial y_1}$ (chain rule)
⇒ 只需知道 activation function ⇒ y_1 對 L 影響 → cost fuction 定義方式決定 $\frac{\partial L}{\partial z}$ ⇒ 同理
Case 2: Not output layer: Compute $\frac{\partial L}{\partial z}$ recursively → until output layer
= 由 output layer 反向計算至欲求層
⇒ Compute $\frac{\partial L}{\partial z}$ for all activation function inputs z *from the output layer