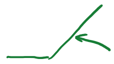
第一周 - Andrew Ng.

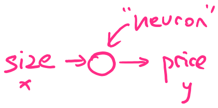
2017年9月17日　　11:06

Welcome to the deep learning specialization

1. Neural networks and deep learning
2. Improve deep neural networks: hyperparameter tuning, regularization, and optimization
3. Structure your machine learning project
4. Convolutional neural networks CNN 卷积神经网络
5. Natural language processing: building sequence models (Recurrent neural network RNN Long short term memory model LSTM) ⇒ NLP

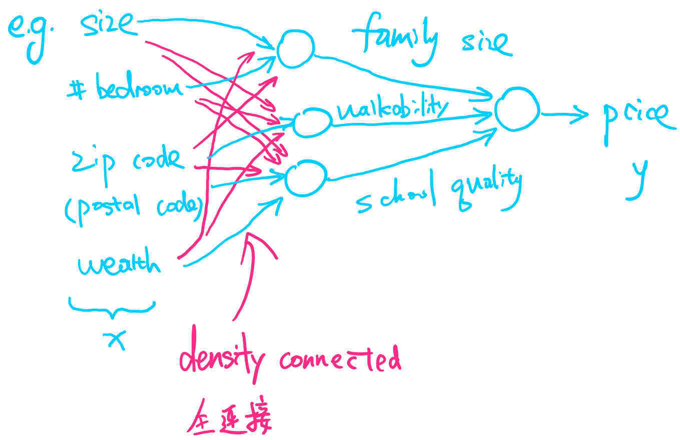
rectified linear regression (ReLu)
线性整流函数　max(0, y)





Supervised learning with neural networks

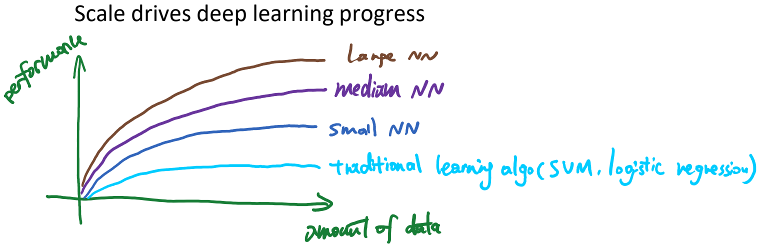
input (x)	output (y)	application	
home features	price	real estate	} standard NN
ad, user info	click on ad? (0/1)	online ads	
(DL) image	object (1, ..., 1000)	photo tagging	CNN 卷积神经网络
audio	text transcript	speech recognition	} RNN (sequence data)
English	Chinese	Machine translation	
image, radar	position of car	autonomous driving	customer hybrid



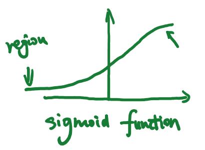
Structured data and unstructured data

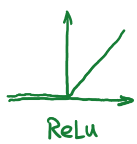
size, # bedrooms, price　　audio, image

Scale drives deep learning progress



Why is deep learning taking off?





gradient descent ≈ 0,
algo will be slow

第二周

2017年10月5日 10:56

Logistic regression as a neural network

Binary classification

RGB. if a image is 64x64 pixel, → 3个 64x64 matrix

unroll all pixel intensity: $x = \begin{bmatrix} \vdots \text{ red} \\ \vdots \text{ green} \\ \vdots \text{ blue} \end{bmatrix}$ — 64x64x3 rows

$x \in \mathbb{R}^{n_x}$, $y \in \{0,1\}$ $n_x = 12288$, $x \to y$

m tr examples $\{(x^{(1)}, y^{(1)}), \cdots (x^{(m)}, y^{(m)})\}$

$m = m_{train}$. m_{test} = # test examples

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix} \uparrow n \text{ rows (features)}$$
$\longleftarrow m \longrightarrow$ colums

GOOD! in this way n×m easier for Neural Network

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

$Y = [y^{(1)} \cdots y^{(m)}]$ $Y \in \mathbb{R}^{1\times m}$, $Y.shape = (1, m)$

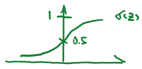
Logistic regression

Given x, want $\hat{y}$ (y的估计值) $\hat{y} = P(y=1|x)$, $0 \le \hat{y} \le 1$

$x \in \mathbb{R}^{n_x}$

parameter of logistic regression: $w \in \mathbb{R}^{n_x}$, $b \in \mathbb{R}$

Output $\hat{y} = \sigma(w^T x + b)$ $\sigma(z) = \frac{1}{1+e^{-z}}$ if z large $\sigma(z) \approx 1$, small $\sigma(z) \approx 0$



$x_0 = 1$, $x \in \mathbb{R}^{n_x+1}$, $\hat{y} = \sigma(\theta^T x)$ $\theta = \begin{bmatrix} \theta_0 \\ \vdots \\ \theta_{n_x} \end{bmatrix}$ ← b, } w

Logistic regression cost function

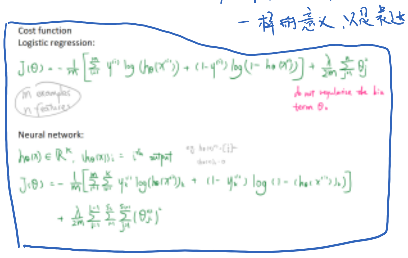
$\hat{y} = \sigma(w^T x + b)$, where $\sigma(z) = \frac{1}{1+e^{-z}}$. $z^{(i)} = w^T x^{(i)} + b$

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

Loss (error) function

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

/ 作个对比, 下面是 machine learning 学到的. 一样的意义, 只是表达方式不同


Cost function
Logistic regression:

$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log(h_\theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$

m examples, n features

do not regularize the bias term θ_0

Neural network:

$h_\Theta(x) \in \mathbb{R}^K$, $(h_\Theta(x))_i = i^{th}$ output

$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + (1-y_k^{(i)})\log(1-(h_\Theta(x^{(i)}))_k)\right]$
$+ \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2$

if $y=1$, $\mathcal{L}(\hat{y}, y) = -\log(\hat{y})$. goal: want $\hat{y}$ large

if $y=0$, $\mathcal{L}(\hat{y}, y) = -\log(1-\hat{y})$. goal: want $\hat{y}$ small

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

The loss function computes the error for a single training examples, and the cost function is the average of the loss function of the

Python and vectorization

Vectorization

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

vectorized:

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

More vectorization examples

Vectors and matrix valued functions

```
Import numpy as np
U = np.exp(v)
U = np.log(v)
U = np.abs(v)
U = np.maximum(v,0)
```

$J=0$; $\boxed{dw_1 = 0; \cancel{dw_2 = 0}}$; $db=0$ ← $dw = np.zeros((n_x, 1))$

For $i = 1$ to m: the first loop
  $z^{(i)} = w^T x^{(i)} + b$
  $a^{(i)} = \sigma(z^{(i)})$
  $J += -[y^{(i)}\log a^{(i)} + (1-y^{(i)})\log(1-a^{(i)})]$
  $dz^{(i)} = a^{(i)} - y^{(i)}$
  $\boxed{\cancel{dw_1 += x_1^{(i)} dz^{(i)}} \quad \cancel{dw_2 += x_2^{(i)} dz^{(i)}}}$ ← $dw += x^{(i)} dz^{(i)}$ $n_x = 2$ (2 feature)
  $db += dz^{(i)}$

second loop: for $j = 1 \cdots n_x$: $dw_j += x_j^{(i)} dz^{(i)}$ $dw_1 \vdots dw_n$

$J /= m$

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

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

Vectorizing logistic regression

$z^{(1)} = w^T x^{(1)} + b$ ... $z^{(m)} = w^T x^{(m)} + b$

$a^{(1)} = \sigma(z^{(1)})$ ... $a^{(m)} = \sigma(z^{(m)})$

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

$Z = [z^{(1)}, \cdots, z^{(m)}] = w^T X + [b, \cdots, b]$ $(1, m)$

in python: $np.dot(w.T, X) + b$ ← a 1x1 real number
expand b to $1 \times m$, call broadcasting

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

Vectorizing logistic regression's gradient output

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

$dZ = [dz^{(1)} \cdots dz^{(m)}]$ $(1\times m)$

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

$dZ = A - Y$

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

$dw /= m$ $db /= m$

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

$dw = \frac{1}{m} X dZ^T$
$= \frac{1}{m}[x^{(1)}dz^{(1)} + \cdots + x^{(m)}dz^{(m)}]$
$n \times 1$

The loss function computes the error for a single training examples, and the cost function is the average of the loss function of the entire training set.

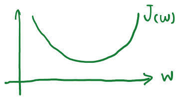
Gradient descent

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

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

convex function

to find w. b that minmize $J(w,b)$



repeat {

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

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

$J(w,b)$ ① $w := w - \alpha \frac{dJ(w,b)}{dw}$ derivative

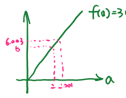
② $b := b - \alpha \frac{dJ(w,b)}{db}$

∂: partial derivative

d: J (cost function)

① update dw ② update db

Derivatives



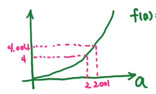
$a=2 \quad f(a)=6$

$a=2.001 \quad f(a)=6.003$

slope (derivative) of f(a) at a=2 is 3

$\frac{df(a)}{da} = 3 = \frac{d}{da} f(a)$

More derivative examples



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

$a=2.001,\ f(a)=4.004$

slope (derivative) of f(a) at a=2 is 4

while $a=5 \quad f(a)=25$

$a=5.001 \quad f(a)=25.010$

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

$(\ln(a))' = \frac{1}{a}$

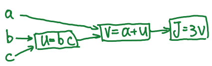
Computation graph

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

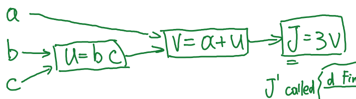
$u = bc$

$v = a+u$

$J = 3v$



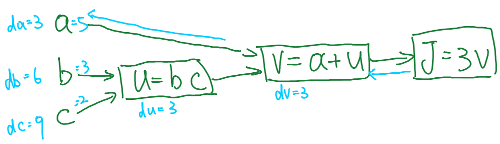
Derivative with a computation graph



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

J' called $\left\{\frac{d\ \text{Final Output Var}}{d\ \text{var}}\right\} \Rightarrow$ "dvar" means the derivative of a final output variables with respect to various intermediate quantities

$\frac{dJ}{da}$ (abbr.) $\to da \qquad \frac{dJ}{dv}$ (abbr.) $\to dv$



Logistic regression gradient descent

$z = w^T x + b$

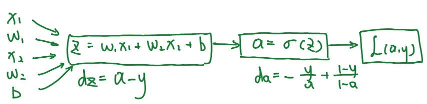
$\hat{y} = a = \sigma(z)$

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

$dw_1 = x_1 \cdot dz$

$dw_2 = x_2\, dz$

$db = dz$



$dz = a - y \qquad da = -\frac{y}{a} + \frac{1-y}{1-a} \qquad \frac{d\mathcal{L}}{da} \cdot \frac{da}{dz}$ denote by dz

since $(\sigma(z))' = \sigma(z)\cdot(1-\sigma(z))$, so $dz = \left(-\frac{y}{a} + \frac{1-y}{1-a}\right) a(1-a)$

$= -y + ya + a - ay$

$= a - y$

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

$dw \mathrel{/}= m \qquad db \mathrel{/}= m$

Implementing logistic regression

$Z = w^T X + b$

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

$A = \sigma(Z)$

$dZ = A - Y$

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

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

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

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

Broadcasting in python

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

$\begin{bmatrix}1\\2\\3\end{bmatrix} + 100 = \begin{bmatrix}101\\102\\103\end{bmatrix} \qquad \begin{bmatrix}1&2&3\\4&5&6\end{bmatrix} + \begin{bmatrix}100&200&300\end{bmatrix} = \begin{bmatrix}101&202&303\\104&205&306\end{bmatrix}$

$\begin{bmatrix}1&2&3\\4&5&6\end{bmatrix} + \begin{bmatrix}100\\200\end{bmatrix} = \begin{bmatrix}101&102&103\\204&205&206\end{bmatrix}$

General principle

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

matlab: similar function: bsxfun

A note on python / numpy vectors

a = np.random.randn(5)

a.shape = (5,) "rank 1 array" 秩为1的数组

Explanation of logistic regression cost function

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

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

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

$P(y|x) = \hat{y}^{y}(1-\hat{y})^{(1-y)}$ 两边同取对数 $\log P(y|x) = y\log\hat{y} + (1-y)\log(1-\hat{y})$

if $y=1$, $P(y|x) = \hat{y}$

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

因为 $\log P(y|x)$ 是↑, 所以 $-\log P(y|x)$ 是↓

Cost on m examples

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

$\log P(\cdots) = \sum_{i=1}^{m} \underbrace{\log P(y^{(i)}|x^{(i)})}_{-\mathcal{L}(\hat{y}^{(i)}, y^{(i)})}$ maximum likelihood estimation 最大似然估计

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

Gradient descent on m examples

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

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

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

$J = 0;\ dw_1 = 0;\ dw_2 = 0;\ db = 0$ 假设只有两个变量 w_1, w_2

For $i = 1$ to m: the first loop

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

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

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

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

dw_1 $dw_1^{(i)} \mathrel{+}= x_1^{(i)} \cdot dz^{(i)}$ $n = 2$ (2 feature) $w_1 := w_1 - \alpha\, dw_1$

$\vdots$

dw_n $dw_2^{(i)} \mathrel{+}= x_2^{(i)} \cdot dz^{(i)}$ $w_2 := w_2 - \alpha\, dw_2$

$db \mathrel{+}= dz^{(i)}$ $b := b - \alpha\, db$

$J \mathrel{/}= m$

$dw_1 \mathrel{/}= m$ $dw_2 \mathrel{/}= m$

第三周

2017年10月24日 1:11

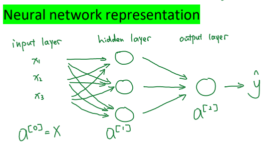
Shallow neural network

Neural networks overview

$x, w, b \rightarrow \boxed{z = w^T x + b} \rightarrow \boxed{a = \sigma(z)} \rightarrow \boxed{\mathcal{L}(a, y)}$

$x, W^{[1]}, b^{[1]} \rightarrow \boxed{z^{[1]} = W^{[1]} x + b^{[1]}} \rightarrow \boxed{a^{[1]} = \sigma(z^{[1]})} \rightarrow \boxed{z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}} \rightarrow \boxed{a^{[2]} = \sigma(z^{[2]})} \rightarrow \boxed{\mathcal{L}(a^{[2]}, y)}$

Neural network representation



e.g. $a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \\ \vdots \\ a_4^{[1]} \end{bmatrix}$

"2 layer NN"

Computing a neural network's output

$z = w^T x + b$

$a = \sigma(z)$

$x_1, x_2, x_3 \rightarrow \boxed{w^T x + b \mid \sigma(z)} \rightarrow a = \hat{y}$

$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}, \quad a_1^{[1]} = \sigma(z_1^{[1]})$

$z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]}, \quad a_4^{[1]} = \sigma(z_4^{[1]})$

$$z^{[1]} = \begin{bmatrix} — w_1^{[1]T} — \\ — w_2^{[1]T} — \\ — w_3^{[1]T} — \\ — w_4^{[1]T} — \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix} = \begin{bmatrix} w_1^{[1]T} x + b_1^{[1]} \\ w_2^{[1]T} x + b_2^{[1]} \\ w_3^{[1]T} x + b_3^{[1]} \\ w_4^{[1]T} x + b_4^{[1]} \end{bmatrix} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix}$$

$W^{[1]}$

Given input x:

$z^{[1]} = W^{[1]} x^{[0]} + b^{[1]}$

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

$z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$

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

Vectoring across multiple examples

$x^{(m)} \rightarrow a^{[2](m)} = \hat{y}^{(m)}$ (layer 2, example i)

for $i = 1$ to m:

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

$a^{[1](i)} = \sigma(z^{[1](i)})$

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

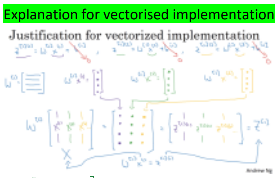
$a^{[2](i)} = \sigma(z^{[2](i)})$

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

X.shape $= (n_x, m)$

$Z^{[1]} = \begin{bmatrix} z^{[1](1)} \cdots z^{[1](m)} \end{bmatrix}$ $\quad A^{[1]} = \begin{bmatrix} a^{[1](1)} \cdots a^{[1](m)} \end{bmatrix}$ → training example number; ↓ hidden units number on the 1st layer

Explanation for vectorised implementation



$X = \begin{bmatrix} x^{(1)} \cdots x^{(m)} \end{bmatrix} \quad A^{[1]} = \begin{bmatrix} a^{[1](1)} \cdots a^{[1](m)} \end{bmatrix}$

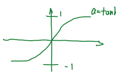
$Z^{[1]} = W^{[1]} X + b^{[1]}$

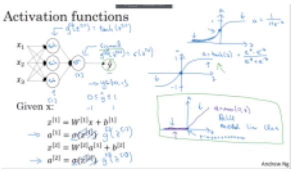
$A^{[1]} = \sigma(Z^{[1]})$

$Z^{[2]} = W^{[2]} A^{[1]} + b^{[2]}$

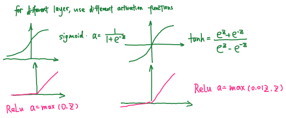
$A^{[2]} = \sigma(Z^{[2]})$

Activation functions





for different layer, use different activation functions



Why do you need non-linear activation functions?

Given x

$z^{[1]} = W^{[1]} x + b^{[1]}$

$a^{[1]} = g^{[1]}(z^{[1]})$ 若这些是线性函数，那么神经网络只是在计算线性函数，

$z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$ 不论有多少隐藏层，都是在做线性函数组合，没有意义

$a^{[2]} = g^{[2]}(z^{[2]})$

Derivatives of activation functions

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

$g'(z) = \frac{d}{dz} g(z) = g(z) \times (1 - g(z))$

$z = 10, \ g(z) \approx 1 \qquad z = -10, \ g(z) \approx 0$

$g'(z) \approx 0 \qquad g'(z) \approx 0$

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

$g'(z) = \frac{1}{4}$

→ Tanh $\quad g(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$

$g'(z) = \frac{d}{dz} g(z) = 1 - (\tanh(z))^2$

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

$g'(z) \approx 0 \qquad g'(z) \approx 0$

$z = 0, \ \tanh z = 0$

$g'(z) = 1$

→ Relu

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

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

→ Leaky ReLu

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

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

Gradient descent for neural networks

parameter: $W^{[1]}_{(n^{[1]}, n^{[0]})}, b^{[1]}_{(n^{[1]}, 1)}, W^{[2]}_{(n^{[2]}, n^{[1]})}, b^{[2]}_{(n^{[2]}, 1)}$

$n_x = n^{[0]}$ input units

$n^{[1]}$ hidden units

$n^{[2]}$ output units

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

Gradient descent:

Repeat { compute predicts ($\hat{y}^{(i)}$, $i = 1, \dots, m$)

$dW^{[1]} = \frac{dJ}{dW^{[1]}}, \ db^{[1]} = \frac{dJ}{db^{[1]}}, \dots$

$W^{[1]} = W^{[1]} - \alpha \, dW^{[1]}$

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

⋮ }

Formulas for computing derivatives

Forward propagation:

$Z^{[1]} = W^{[1]} X + b^{[1]}$

$A^{[1]} = g^{[1]}(Z^{[1]})$

$Z^{[2]} = W^{[2]} A^{[1]} + b^{[2]}$

$A^{[2]} = g^{[2]}(Z^{[2]}) = \sigma(Z^{[2]})$

Backpropagation:

1. $dZ^{[2]} = A^{[2]} - Y \qquad Y = [y^{(1)} \dots y^{(m)}]$
2. $dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]T}$
3. $db^{[2]} = \frac{1}{m}$ np.sum($dZ^{[2]}$, axis=1, keepdims=True) $\quad (n^{[2]}, 1)$
4. $dZ^{[1]} = \underbrace{W^{[2]T} dZ^{[2]}}_{(n^{[1]}, m)} * \underbrace{g^{[1]\prime}(Z^{[1]})}_{(n^{[1]}, m)}$
5. $dW^{[1]} = \frac{1}{m} dZ^{[1]} X^T$
6. $db^{[1]} = \frac{1}{m}$ np.sum($dZ^{[1]}$, axis=1, keepdims=True) $\quad (n^{[1]}, 1)$

Backpropagation intuition

logistic regression

$x, w, b \rightarrow \boxed{z = w^T x + b} \rightarrow \boxed{a = \sigma(z)} \rightarrow \boxed{\mathcal{L}(a, y)}$ $\quad -y \log a - (1-y) \log(1-a)$

$dz = a - y \qquad da = -\frac{y}{a} + \frac{1-y}{1-a}$

$\frac{dL}{dz^{[2]}} = \frac{dL}{da^{[2]}} \cdot \frac{da^{[2]}}{dz^{[2]}}$

$= W^{[2]T} dz^{[2]} \cdot \frac{da^{[1]}}{dz^{[1]}}$

$= W^{[2]T} dz^{[2]} \cdot \frac{d}{dz^{[1]}} g^{[1]}(z^{[1]})$

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

$\boxed{\frac{dL}{da^{[2]}} \frac{da^{[2]}}{dz^{[2]}} \frac{dz^{[2]}}{da^{[1]}}} \frac{da^{[1]}}{dz^{[1]}}$

$x, W^{[1]}, b^{[1]} \rightarrow \boxed{z^{[1]} = W^{[1]} x + b^{[1]}} \rightarrow \boxed{a^{[1]} = \sigma(z^{[1]})} \rightarrow \boxed{z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}} \rightarrow \boxed{a^{[2]} = \sigma(z^{[2]})} \rightarrow \boxed{\mathcal{L}(a^{[2]}, y)}$

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

$z^{[2]}, dz^{[2]} (n^{[2]}, 1)$

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

$W^{[2]} (n^{[2]}, n^{[1]})$

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

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

$dz^{[2]} = da^{[2]} - Y$ 这两个式子是在激活函数是sigmoid函数时候

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

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

Summary of gradient descent

1. $dz^{[2]} = a^{[2]} - y$
2. $dW^{[2]} = dz^{[2]} a^{[1]T}$
3. $db^{[2]} = dz^{[2]}$
4. $dz^{[1]} = W^{[2]T} dz^{[2]} \cdot g^{[1]\prime}(z^{[1]})$
5. $dW^{[1]} = dz^{[1]} x^T$
6. $db^{[1]} = dz^{[1]}$

Vectorization

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

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

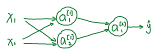
$db^{[2]} = \frac{1}{m}$ np.sum($dZ^{[2]}$, axis=1, keepdims=True)

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

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

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

Random initialization



$W^{[1]}$ = np.random.randn((2,2)) * 0.01

$b^{[1]}$ = np.zeros((2,1))

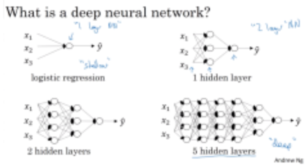
$W^{[2]}$ = np.random.randn((1,2)) * 0.01

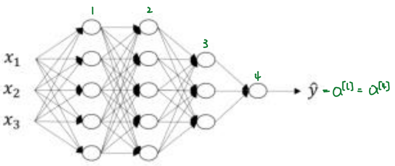
$b^{[2]}$ = np.zeros((2,1))

第四周

2017年10月26日 10:18

Deep-layer neural network





$L = 4$ (# layers)

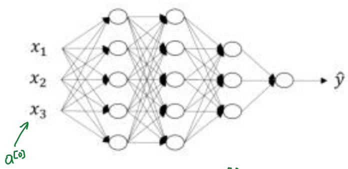
$n^{[l]}$ = # units in layer l, $n^{[1]} = 5$, $n^{[2]} = 5$, $n^{[3]} = 3$, $n^{[4]} = 1$, $n^{[0]} = n_x = 3$

$a^{[l]}$ = activations in layer l

$a^{[l]} = g^{l}(z^{[l]})$, $W^{[l]}$ = weight for $z^{[l]}$

$X = a^{[0]}$

Forward propagation in a deep network



$z^{[1]} = W^{[1]}x + b^{[1]}$

$a^{[1]} = g^{[1]}(z^{[1]})$

$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$

$a^{[2]} = g^{[2]}(z^{[2]})$

$\vdots$

$z^{[4]} = W^{[4]}a^{[3]} + b^{[4]}$

$a^{[4]} = g^{[4]}(z^{[4]}) = \hat{y}$

$z^{[l]} = W^{[l]}a^{[l-1]} + b^{[l]}$

$a^{[l]} = g^{[l]}(z^{[l]})$

Vectorized:

$z^{[1]} = W^{[1]}A^{[0]} + b^{[1]}$

$A^{[1]} = g^{[1]}(z^{[1]})$

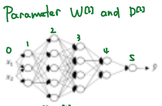
$\vdots$

$\hat{y} = g^{[4]}(z^{[4]}) = A^{[4]}$

for loop $l = 1, \dots, 4$

Getting your matrix dimensions right

Parameter $W^{[l]}$ and $b^{[l]}$



$dW^{[l]}, W^{[l]} : (n^{[l]}, n^{[l-1]})$

$db^{[l]}, b^{[l]} : (n^{[l]}, 1)$

$(n^{[1]}, n^{[0]})$

$z^{[1]} = W^{[1]} \cdot x + b^{[1]}$ the vector of activations

$(n^{[1]}, 1)$ $(n^{[0]}, 1)$

Vectorized implementation

$z^{[1]} = W^{[1]}x + b^{[1]}$ $(n^{[1]}, m)$ m is the number of dataset

by broadcasting . $b^{[1]}$ $(n^{[1]}, 1) \Rightarrow (n^{[1]}, m)$

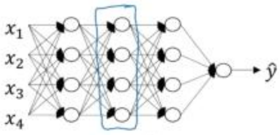
$Z^{[l]}, A^{[l]} : (n^{[l]}, m)$

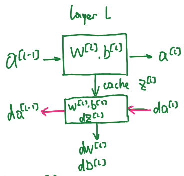
Why deep representation?

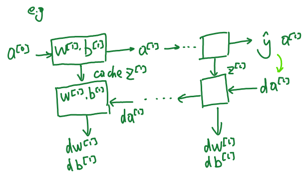
Circuit theory and deep learning

Informally, there are functions you can compute with a small L-layer deep neural network that shallower networks require exponentially more hidden units to compute.

Building blocks of deep neural networks







layer l : $W^{[l]}, b^{[l]}$

Forward : input $a^{[l-1]}$, output $a^{[l]}$

Backward : input $da^{[l]}$, output $da^{[l-1]}$ $dW^{[l]}, db^{[l]}$

$z^{[l]} = W^{[l]}A^{[l-1]} + b^{[l]}$

$a^{[l]} = g^{[l]}(z^{[l]})$

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:

$z^{[l]} = W^{[l]} \cdot A^{[l-1]} + b^{[l]}$

$A^{[l]} = g^{[l]}(z^{[l]})$

Backward propagation for layer l

input $da^{[l]}$

output $da^{[l-1]}$, $dW^{[l]}$, $db^{[l]}$

$dz^{[l]} = da^{[l]} * g^{[l]\prime}(z^{[l]})$

$dW^{[l]} = dz^{[l]} \cdot a^{[l-1]}$

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

$da^{[l-1]} = W^{[l]T} \cdot dz^{[l]}$

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

Vectorized:

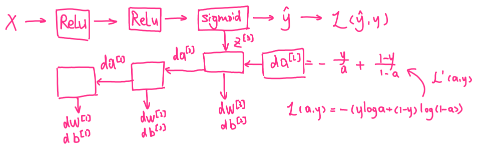
$dZ^{[l]} = dA^{[l]} * g^{[l]\prime}(z^{[l]})$

$dW^{[l]} = \frac{1}{m} dZ^{[l]} \cdot A^{[l-1]T}$

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

$dA^{[l-1]} = W^{[l]T} dZ^{[l]}$

Summary



Parameters vs hyperparameters

parameters : $W^{[l]}, b^{[l]}$ ...

hyperparameters : learning rate α

iterations

hidden layer L

hidden units $n^{[1]}, n^{[2]}$ ...

choice of activation function

other : momentum, mini-batch size, regularization

Applied deep learning is a very empirical process

Application: vision, speech recognition, NLP, online advertising, recommendation

What does this have to do with human brain?

forward propagation

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

backward propagation

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

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

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

$$dZ^{[L-1]} = dW^{[L]^T} dZ^{[L]} g'^{[L]}(Z^{[L-1]})$$

$$\vdots$$

$$dZ^{[1]} = dW^{[L]^T} dZ^{[2]} g'^{[1]}(Z^{[1]})$$

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

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