

# 深度学习之神经网络的结构 Part 1

卷积神经网络 → 摄像图像识别

长短时记忆网络 → 摄像语音识别

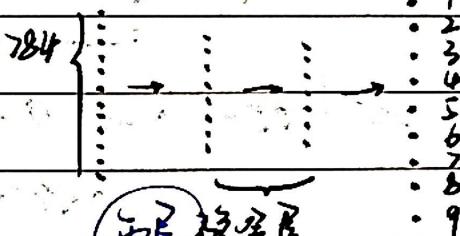
第一层

神经元 → 得到对应像素灰度值(激活值) (0 → 1)

张数

9

$$28 \times 28 = 784 \text{ (个神经元)} \Rightarrow$$



随机  
设置

(进行处理识别数字的具体工作)

每层10个神经元

→ 为什么要分层?

(注:实际上神经元不会这样做)

将识别工作拆分成小块。例如“9” → 第一隐含层识别“竖边并点亮”

→ 第二层识别“圆”和“边”并点亮

↓  
(上层影响下层) → 拼成数字

· 设计让第一隐含层中这一神经元

能够正确识别出图像中的这块区域,否则在一侧

三 → 周围点负权重

给每条线都赋上一个权重值 weight

Activation  $w_1a_1 + w_2a_2 + \dots + w_na_n$

激活值

→ 现在如果把关注区域的权重设为正值, 其区域一律输出

0 激活值 对所有像系取加权和, 就只会累加关注区域像素值了。

→ 识别偏在一侧?

只需要给周围一圈的像素赋予负的权重

这样当中间像亮, 周围像暗暗时, 加权就能达到最大值

激活值 0

⇒ 这些神经元乘以负权重得到的值也小。

$w_1a_1 + w_2a_2 + \dots + w_na_n \rightarrow$  需要激活值处在0和1之间

Sigmoid 函数:  $\sigma(x) = \frac{1}{1+e^{-x}}$ . 能把非常大的负值变成接近0.  
非常大正值变为1

所以这个神经元中激活值, 实际上是一个对加权和到底有多正的打分

$$T \sigma(w_1a_1 + w_2a_2 + \dots + w_na_n)$$

$\downarrow$   
Sigmoid

有时激活值>0, 也不想激发,  $\Rightarrow$  可以设置一个偏置值  
保证不能随便激发

例如:

当 weight sum > 10, 激发才有意义

$$\Rightarrow \sigma(w_1a_1 + \dots + w_na_n - 10) \quad \uparrow \text{偏置 bias}$$

第一层(隐藏)每个神经元各带 784 个权重, 各带 16 个偏置  
(总计  $784 \times 16$  个权重和 16 个偏置)

$\Rightarrow$  整个网络

权重  $784 \times 16 + 16 \times 16 + 16 \times 10 = 13002$

偏置  $16 + 16 + 10$

Learning  $\rightarrow$  找到正确的权重和偏置

bias

$$\therefore a_0^{(l)} = \sigma(w_{0,0}a_0^{(l)} + w_{0,1}a_1^{(l)} + \dots + w_{0,n}a_n^{(l)} + b_0)$$

$$\downarrow$$

$$\sigma \left( \begin{bmatrix} w_{0,0} & w_{0,1} & \dots & w_{0,n} \\ w_{1,0} & w_{1,1} & \dots & w_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,0} & w_{k,1} & \dots & w_{k,n} \end{bmatrix} \begin{bmatrix} a_0^{(l)} \\ a_1^{(l)} \\ \vdots \\ a_n^{(l)} \end{bmatrix} \right) = \begin{bmatrix} a_0^{(l+1)} \\ a_1^{(l+1)} \\ \vdots \\ a_n^{(l+1)} \end{bmatrix}$$

神经元

function

权重

Function

$$\Rightarrow a^{(l)} = \sigma(wa^{(l)} + b)$$

$$\quad + \begin{bmatrix} b_0 \\ \vdots \\ b_k \end{bmatrix}$$

Function

## 深度学习之梯度下降法 part2 Training Set

→ 机器学习部分：监督学习，代价函数 → 找到最小值，↑准确率

计算梯度的算法是神经网络的核心 → 反向传播法(BP)

Neural network function

Input: 784 pixels

Output: 10 number

Parameters: 13.002 weights/biases

further

→

Cost function

Input: 13.002 weights/biases

Output: 1 number (the cost)

Parameters: Many, many, many

training examples

代价函数的梯度 → 如何微调权重 偏置的值

## 深度学习之反向传播算法 Part3 (上理解反向传播)

$$-\nabla C(\dots) = \begin{bmatrix} \vdots \\ \vdots \end{bmatrix}$$

梯度向量每一项的大小 → 代价函数对于每个参数

有多敏感

例 → 当输入为  $\boxed{2}$  时，可能得到

(神经网络如何学习的一个理论)

“输出理论” → “同一激活的神经元关联在

→ 让“看到一个 $\boxed{2}$ ”时激发的神经一起

和“想到一个 $\boxed{2}$ ”时激发的神经元

联系地更紧密

⑥.5 0

⑤.8 1

④.2 2

需要让“ $\boxed{2}$ ”的激活值变大

③.0 3

$0.2 = \sigma(w_0a_0 + w_1a_1 + \dots + w_{n-1}a_{n-1} + b)$

②.4 4

↓  
increase  $b$  → in proportion to  $a_i$

①.6 5

↑ Increase  $w_i$  选择性降低

①.0 6

Change  $a_i$  → in proportion to  $w_i$

①.0 7

↓ 在所有权重连接的  
神经元更弱

①.2 8

所有负权重连接的神经元更强。

其他的  
神经元  
的权重变化的

⇒ 所以，我们都会把数第 2 神经元的期待和别的输出神经元的期待全部加起来作为如何改变倒数第二层神经元的指示

⇒ 错到了一半对倒数第二层改动的~~变化量~~ } 反向传播

⇒ 把以上过程循环到第一层

- 对所有 training data 进行反向传播，并记录  $w_0, \dots, w_{13,001}$  再对每一个对应的  $w_0, \dots, w_{13,001}$  分别取平均值
- (→  $w_0 = \frac{1}{n} (w_0^{(1)} + \dots + w_0^{(n)})$ ) n 个样本

# 反向传播算法算的是单个训练样本怎样修改权重与偏置，不仅说每个参数应该变大变小，还包括了这些变化比例是多大，才能最快地降低代价。

↓  
# 真正的梯度下降要对 all training data 都这么操作，然后对变化值取平均。

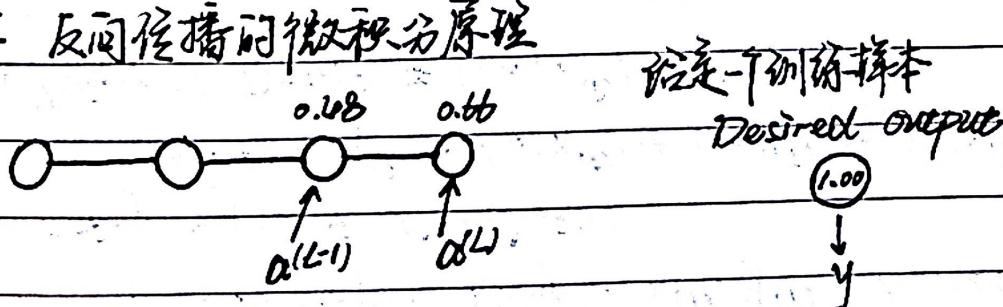
↓  
# (算起来太慢) ⇒ 把所有样本分别各个 minibatch 中去

计算一个 minibatch 来作为梯度下降的第一步

计算每个 minibatch 的梯度，调整参数，不断循环

最终会收敛到代价函数的一个局部最小值上。

### Part 3 下 反向传播的微积分原理



$$Cost \rightarrow Cost(\dots) = (a^{(L)} - y)^2$$

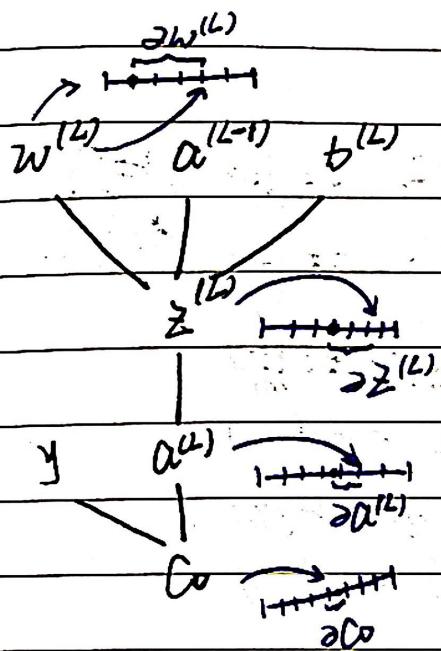
$$z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)}$$

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

$$\frac{\partial C}{\partial w^{(l)}} = \alpha^{(L-1)} \sigma'(\sum^{(L)}) \sum (\alpha^{(L)} - y)$$

Chain rule  
rule

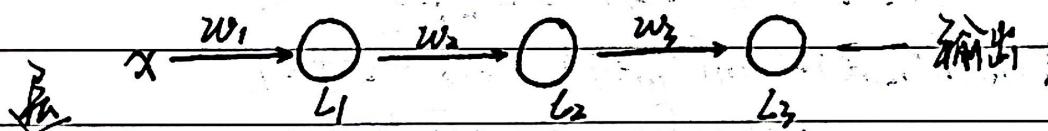
$$= \frac{\partial \sum^{(L)}}{\partial w^{(l)}} \cdot \frac{\partial \alpha^{(L)}}{\partial \sum^{(L)}} \cdot \frac{\partial C}{\partial \alpha^{(L)}}$$



$$\frac{\partial C}{\partial w^{(l)}} = \frac{1}{n} \sum_{k=0}^{n-1} \frac{\partial C_k}{\partial w^{(l)}}$$

Average of All training examples.

## 正向传播和反向传播



各输出定义:  $l_1 = \text{sigmoid}(w_1, x)$ ,  $l_2 = \text{sig}(w_2, l_1)$ ,  $l_3 = \text{sig}(w_3, l_2)$

定义整个网络最终的损失函数:  $\text{loss} = \text{loss}(l_3, y_{\text{expect}})$

对损失函数求  $w_3$  偏导数, 得到:

$$\frac{\partial \text{loss}}{\partial w_3} = \text{loss}'(l_3, y_{\text{expect}}) \text{sig}'(w_3, l_2) l_2$$

同理:

$$\frac{\partial \text{loss}}{\partial w_2} = \text{loss}'(l_3, y_{\text{expect}}) \text{sig}'(w_3, l_2) \text{sig}'(w_2, l_1) l_1$$

$$\frac{\partial \text{loss}}{\partial w_1} = \text{loss}'(l_3, y_{\text{expect}}) \text{sig}'(w_3, l_2) \text{sig}'(w_2, l_1) \text{sig}'(w_1, x) x$$

综上所述

$$\frac{\partial \text{loss}}{\partial w_3} = \text{loss}'(l_3) l_2$$

$$\frac{\partial \text{loss}}{\partial w_2} = \text{loss}'(l_3) l_2 l_1$$

$$\frac{\partial \text{loss}}{\partial w_1} = \text{loss}'(l_3) l_2 l_1 x$$