太原理工大学
TAIYUAN UNIVERSITY OF TECHNOLOGY

太原理工大学
大数据学院
COLLEGE OF DATA SCIENCE
TAIYUAN UNIVERSITY OF TECHNOLOGY

第二章　深度前馈网络

深度前馈网络

主要内容

PART

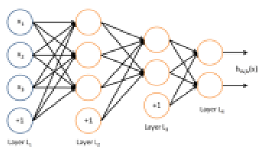
CNN反向传播

ONE

CNN反向传播

全链接：前向传播与反向回馈

$$z^{(l+1)} = W^{(l)}a^{(l)} + b^{(l)}$$
$$a^{(l+1)} = f(z^{(l+1)})$$



Square Euclidean Distance (regression)

$$J = \frac{1}{2}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

$$J(W,b;x,y) = \frac{1}{2}\left\|h_{W,b}(x) - y\right\|^2$$

$$= \left[\frac{1}{m}\sum_{i=1}^{m}\left(\frac{1}{2}\left\|h_{W,b}(x^{(i)}) - y^{(i)}\right\|^2\right)\right] + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(W_{ji}^{(l)}\right)^2$$

更新迭代：

$$W_{ij}^{(l)} = W_{ij}^{(l)} - \alpha\frac{\partial}{\partial W_{ij}^{(l)}}J(W,b)$$

$$b_i^{(l)} = b_i^{(l)} - \alpha\frac{\partial}{\partial b_i^{(l)}}J(W,b)$$

$$\frac{\partial}{\partial W_{ij}^{(l)}}J(W,b) = \left[\frac{1}{m}\sum_{i=1}^{m}\frac{\partial}{\partial W_{ij}^{(l)}}J(W,b;x^{(i)},y^{(i)})\right] + \lambda W_{ij}^{(l)}$$

$$\frac{\partial}{\partial b_i^{(l)}}J(W,b) = \frac{1}{m}\sum_{i=1}^{m}\frac{\partial}{\partial b_i^{(l)}}J(W,b;x^{(i)},y^{(i)})$$

CNN反向传播

全链接：反向推导

sigmoid函数 $f'(z) = f(z)(1-f(z))$

tanh函数 $f'(z) = 1-(f(z))^2$

假设神经网络(NN)总共有 L 层

当第$L-1$层时，权重求导

$$\frac{\partial J}{\partial W_{ij}^{L-1}} = \frac{\partial J}{\partial z_i^L}\frac{\partial z_i^L}{\partial W_{ij}^{L-1}} = \delta_i^L a_j^{L-1}$$

$$\delta_i^L = \frac{\partial J}{\partial z_i^L} = \frac{\partial}{\partial z_i^L}\sum_{i=1}^{s_L}\frac{1}{2}\|y_i - f(z_i^L)\|^2 = -(y_i - f(z_i^L))f'(z_i^L)$$

当第$L-2$层时，权重求导

$$\frac{\partial J}{\partial W_{ij}^{L-2}} = \frac{\partial J}{\partial z_i^{L-1}}\frac{\partial z_i^{L-1}}{\partial W_{ij}^{L-2}} = \delta_i^{L-1} a_j^{L-2}$$

$$\delta_i^{L-1} = \frac{\partial J}{\partial z_i^{L-1}} = \frac{\partial}{\partial z_i^{L-1}}\sum_{b=1}^{s_L}\frac{1}{2}\|y_b - f(z_b^L)\|^2 = \sum_{b=1}^{s_L} -\left(y_b - f(z_b^L)\right) f'(z_b^L)\frac{\partial z_b^L}{\partial z_i^{L-1}}$$

$$= \sum_{b=1}^{s_L}\delta_b^L \bullet w_{bi}^{L-1} f'(z_i^{L-1})$$

$$= \left(\sum_{b=1}^{s_L}\delta_b^L w_{bi}^{L-1}\right) f'(z_i^{L-1})$$

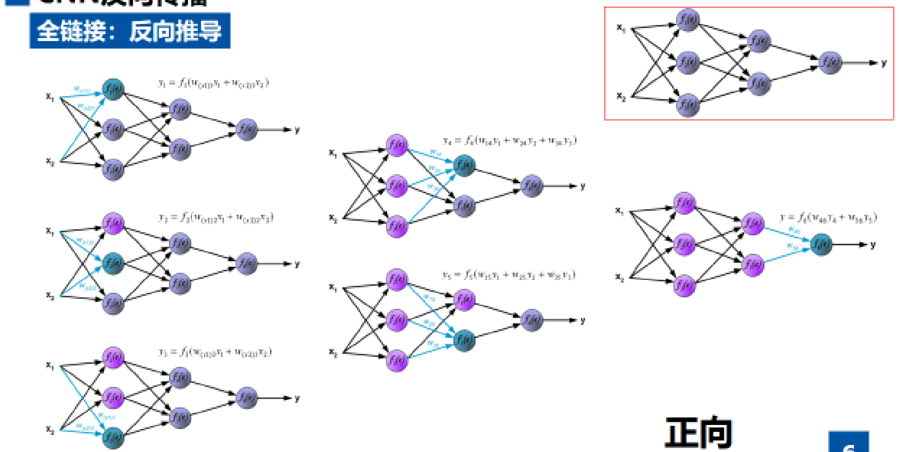
★

5

CNN反向传播

全链接：反向推导

http://galaxy.agh.edu.pl/~vlsi/AI/backp_t_en/backprop.html



正向

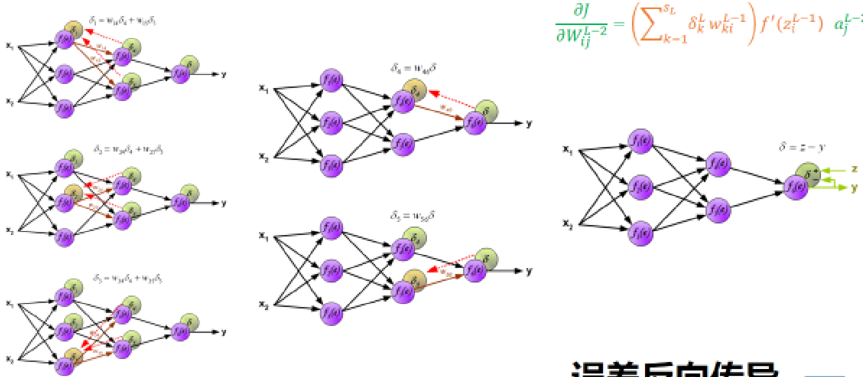
6

CNN反向传播

全链接：反向推导

$$\frac{\partial J}{\partial w_{ij}^{L-1}} = -(y_i - f(z_i^L))f'(z_i^L)a_j^{L-1}$$

$$\frac{\partial J}{\partial W_{ij}^{L-2}} = \left(\sum_{k-1}^{s_L}\delta_k^L w_{ki}^{L-1}\right)f'(z_i^{L-1})\ a_j^{L-2}$$



误差反向传导

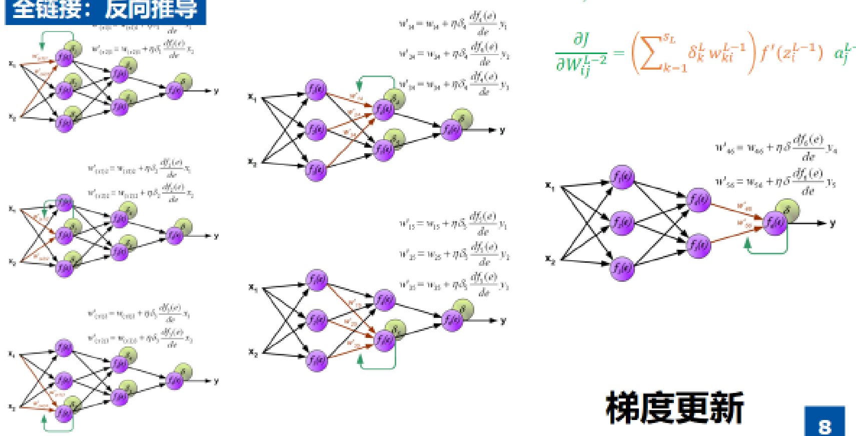
7

CNN反向传播

全链接：反向推导

$$\frac{\partial J}{\partial w_{ij}^{L-1}} = -(y_i - f(z_i^L))f'(z_i^L)a_j^{L-1}$$

$$\frac{\partial J}{\partial W_{ij}^{L-2}} = \left(\sum_{k-1}^{s_L}\delta_k^L w_{ki}^{L-1}\right)f'(z_i^{L-1})\ a_j^{L-2}$$

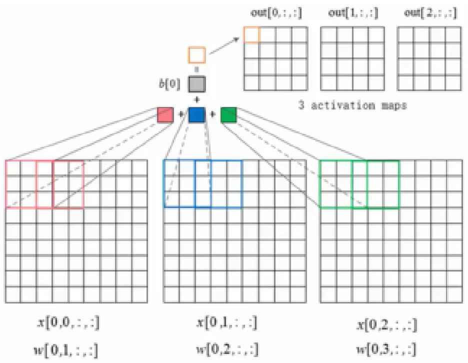


梯度更新

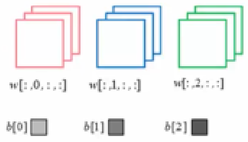
8

CNN神经网络

CNN：训练（前向传播）



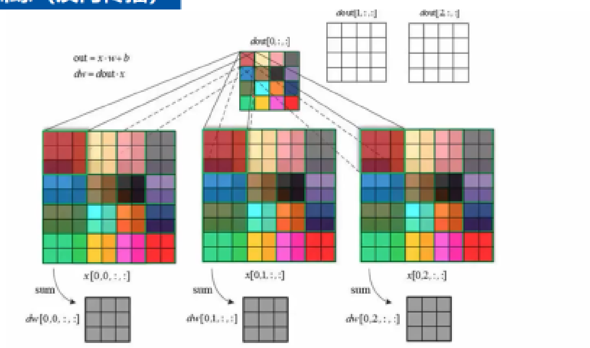
$$out[0,0,0] = np.sum(x[0,:,:3,:3] * w[0]) + b[0]$$
$$out[0,0,1] = np.sum(x[0,:,2:5,:3] * w[0]) + b[0]$$
$$\vdots$$
$$out[1,0,0] = np.sum(x[0,:,:3,:3] * w[1]) + b[1]$$
$$out[1,0,1] = np.sum(x[0,:,2:5,:3] * w[1]) + b[1]$$
$$\vdots$$
$$out[2,0,0] = np.sum(x[0,:,:3,:3] * w[2]) + b[2]$$
$$out[2,0,1] = np.sum(x[0,:,2:5,:3] * w[2]) + b[2]$$
$$\vdots$$



9

CNN神经网络

CNN：训练（反向传播）

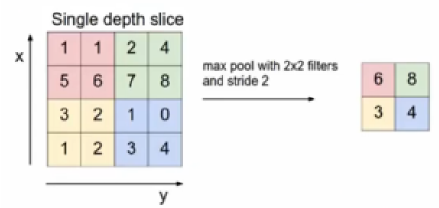


$$dw[0,0,:,:] = x[0,0,:3,:3] * dout[0,0,0] + x[0,0,:3,2:5] * dout[0,0,1] + \cdots + x[0,0,6:9,6:9] * dout[0,3,3]$$
$$dw[0,1,:,:] = x[0,1,:3,:3] * dout[0,0,0] + x[0,1,:3,2:5] * dout[0,0,1] + \cdots + x[0,1,6:9,6:9] * dout[0,3,3]$$
$$dw[0,2,:,:] = x[0,2,:3,:3] * dout[0,0,0] + x[0,2,:3,2:5] * dout[0,0,1] + \cdots + x[0,2,6:9,6:9] * dout[0,3,3]$$

10

CNN反向传播

推导



forward: [1 3; 2 2] -> [2]

backward: [2] -> [0.5 0.5; 0.5 0.5]

forward: [1 3; 2 2] -> 3

backward: [3] -> [0 3; 0 0]

11

PART TWO 深度前馈网络

深度前馈网络

通用近似定理

定理 4.1 – 通用近似定理（Universal Approximation Theorem）[Cybenko, 1989, Hornik et al., 1989]： 令 $\varphi(\cdot)$ 是一个非常数、有界、单调递增的连续函数，$\mathcal{I}_d$ 是一个 d 维的单位超立方体 $[0,1]^d$，$C(\mathcal{I}_d)$ 是定义在 $\mathcal{I}_d$ 上的连续函数集合。对于任何一个函数 $f \in C(\mathcal{I}_d)$，存在一个整数 m，和一组实数 $v_i, b_i \in \mathbb{R}$ 以及实数向量 $\mathbf{w}_i \in \mathbb{R}^d$，$i = 1, \cdots, m$，以至于我们可以定义函数

$$F(\mathbf{x}) = \sum_{i=1}^{m} v_i \varphi(\mathbf{w}_i^{\mathrm{T}} \mathbf{x} + b_i), \tag{4.33}$$

作为函数 f 的近似实现，即

$$|F(\mathbf{x}) - f(\mathbf{x})| < \epsilon, \forall \mathbf{x} \in \mathcal{I}_d. \tag{4.34}$$

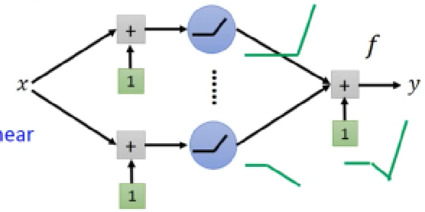
其中 $\epsilon > 0$ 是一个很小的正数。

根据通用近似定理，对于具有线性输出层和至少一个使用“挤压”性质的激活函数的隐藏层组成的前馈神经网络，只要其隐藏层神经元的数量足够，它可以以任意的精度来近似任何从一个定义在实数空间中的有界闭集函数。

深度前馈网络

通用近似定理

- Given a ***shallow*** network structure with one hidden layer with ReLU activation and linear output



A piece-wise linear functions

- Given a L-Lipschitz function f^*
  - How many neurons are needed to approximate f^*?

深度前馈网络

通用近似定理

- Given a L-Lipschitz function f^*
  - How many neurons are needed to approximate f^*?

L-Lipschitz Function (smooth)

$$\|f(x_1) - f(x_2)\| \le L\|x_1 - x_2\|$$

Output change　　Input change

L=1 for "$1 - Lipschitz$"

深度前馈网络

通用近似定理

$$\max_{0 \le x \le 1} |f(x) - f^*(x)| \le \varepsilon$$

$$\sqrt{\int_0^1 |f(x) - f^*(x)|^2 \, dx} \le \varepsilon$$

- Given a L-Lipschitz function f^*
  - How many neurons are needed to approximate f^*?

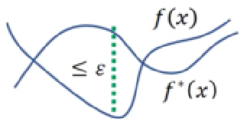
$f \in N(K)$ ➡ The function space defined by the network with K neurons.

Given a small number $\varepsilon > 0$

What is the number of K such that

Exist $f \in N(K)$, $\max_{0 \le x \le 1} |f(x) - f^*(x)| \le \varepsilon$

The difference between $f(x)$ and $f^*(x)$ is smaller than ε.



深度前馈网络

通用近似定理

Universality

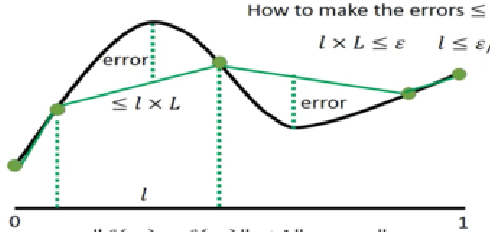
- L-Lipschitz function f^*

All the functions in $N(K)$ are piecewise linear.

Approximate f^* by a piecewise linear function f

How to make the errors $\leq \varepsilon$

$l \times L \leq \varepsilon \quad l \leq \varepsilon / L$



$\|f(x_1) - f(x_2)\| \leq L\|x_1 - x_2\|$

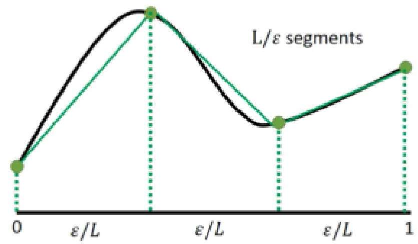
深度前馈网络

通用近似定理

Universality

- L-Lipschitz function f^*

How to make a 1 hidden layer relu network have the output like green curve?



深度前馈网络

通用近似定理

- 1. 实现
  - 使用Numpy实现前馈神经网络
- 2. 函数拟合
  - 理论和实验证明，一个两层的ReLU网络可以模拟任何函数
- https://github.com/nndl/exercise/tree/master/for_chapter_4_%20simple%20neural%20network

深度前馈网络

通用近似定理

Exponential Representation Advantage of Depth

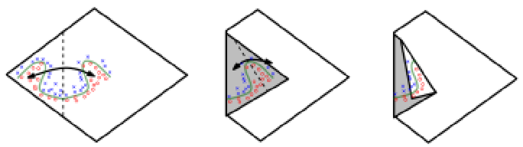


Figure 6.5

PART THREE

TensorFlow

http://playground.tensorflow.org/

Tinker With a **Neural Network** Right Here in Your Browser.
Don't Worry, You Can't Break It. We Promise.

TensorFlow PlayGround

- 选择Sigmoid函数作为激活函数，明显能感觉到训练的时间很长，ReLU函数能大大加快收敛速度
- 当把隐含层数加深后，会发现Sigmoid函数作为激活函数，训练过程loss降不下来
- 隐含层的数量不是越多越好，层数和特征的个数太多，会造成优化的难度和出现过拟合的现象
- **只需要输入最基本的特征x1，x2，只要给予足够多层的神经网络和神经元，神经网络会自己组合出最有用的特征**

23

TensorFlow PlayGround

24

TensorFlow
安装

1. Windows
CPU版：
环境：python 3.5, 3.6(64位)

本地pip安装：
pip3 install --upgrade tensorflow

Anaconda安装：
（1）创建一个名为tensorflow的conda环境
conda create -n tensorflow pip python = 3.5
（2）激活conda
activate tensorflow环境
（3）在conda环境中安装TensorFlow
pip install --ignore-installed --upgrade tensorflow

25

TensorFlow
安装

1. Windows
GPU版：
环境
（1）python3.5及以上（64位）
（2）GPU卡：计算力不小于3.0的NVIDIA显卡
（3）CUDA工具包9.0
（4）cuDNN v7.0

2. Ubuntu（Ubuntu 16.04或更高版本）

环境与window要求一致
https://tensorflow.google.cn/install/install_linux

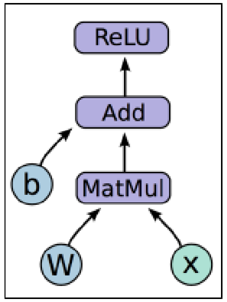
3. macOS（macOS X 10.11（El Capitan）或更高版本）
https://tensorflow.google.cn/install/install_mac

26

TensorFlow
Basic concepts

- Express a numeric computation as a **graph**.
- Graph nodes are **operations** which have any number of inputs and outputs
- Graph edges are **tensors** which flow between nodes

$$h_i = \mathrm{ReLU}(Wx + b)$$



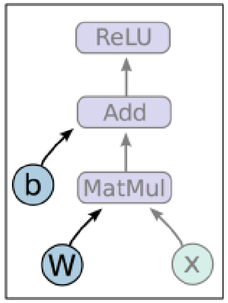
27

TensorFlow
Basic concepts

$$h_i = \mathrm{ReLU}(Wx + b)$$

`Variables` are 0-ary stateful nodes which output their current value. (State is retained across multiple executions of a graph.)

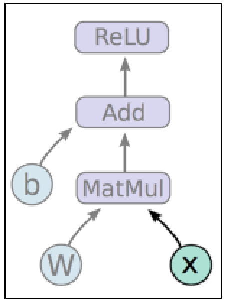
(parameters, gradient stores, eligibility traces, …)



28

TensorFlow
Basic concepts

$h_i = \text{ReLU}(Wx + b)$



Placeholders are 0-ary nodes whose value is fed in at execution time.

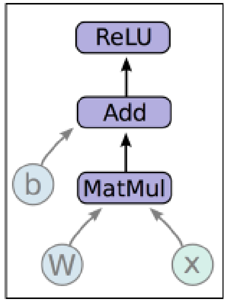
(inputs, variable learning rates, …)

TensorFlow
Basic concepts

$h_i = \text{ReLU}(Wx + b)$

Mathematical operations:

MatMul: Multiply two matrix values.
Add: Add elementwise (with broadcasting).
ReLU: Activate with elementwise rectified linear function.

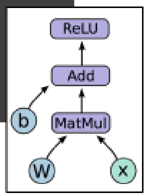


TensorFlow
Basic concepts

1. Create model weights, including initialization
   a. W ~ *Uniform*(-1, 1); b = 0
2. Create input placeholder x
   a. m * 784 input matrix
3. Create computation graph

```
import tensorflow as tf

1  b = tf.Variable(tf.zeros((100,)))
   W = tf.Variable(tf.random_uniform((784, 100),
                   -1, 1))
2  x = tf.placeholder(tf.float32, (None, 784))
3  h_i = tf.nn.relu(tf.matmul(x, W) + b)
```



$h_i = \text{ReLU}(Wx + b)$

Just Run It!

```
sess = tf.Session()
sess.run(tf.initialize_all_variables())
sess.run(h_i, {x: np.random.random(64, 784)})
```


TensorFlow
Basic concepts

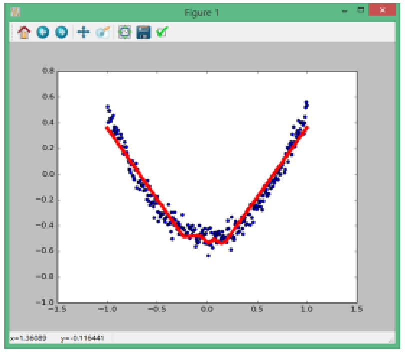
1. Build a graph
   a. Graph contains parameter specifications, model architecture, optimization process, …
   b. Somewhere between 5 and 5000 lines

2. Initialize a session

3. Fetch and feed data with `Session.run`
   a. Compilation, optimization, etc. happens at this step
   — you probably won't notice

TensorFlow

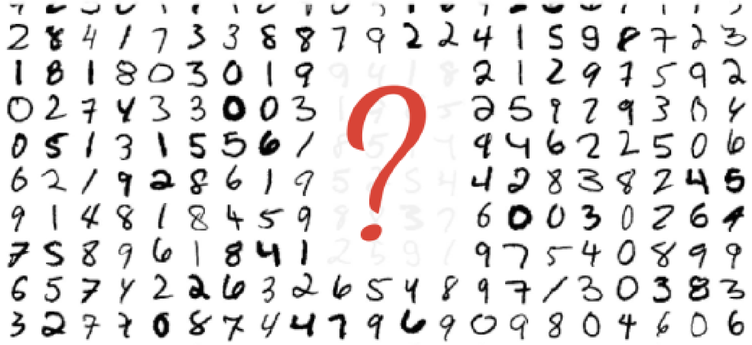
TOY



33

深度前馈网络

单层



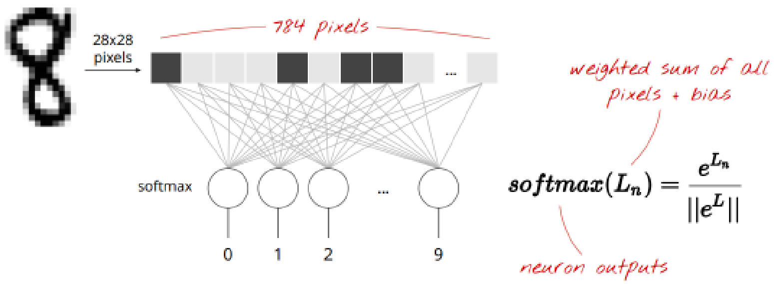
MNIST = Mixed National Institute of Standards and Technology - Download the dataset at http://yann.lecun.com/exdb/mnist/

34

深度前馈网络

单层

$$h_\theta(x) = \begin{bmatrix} P(y=1|x;\theta) \\ P(y=2|x;\theta) \\ \vdots \\ P(y=K|x;\theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^{K} \exp(\theta^{(j)\top} x)} \begin{bmatrix} \exp(\theta^{(1)\top} x) \\ \exp(\theta^{(2)\top} x) \\ \vdots \\ \exp(\theta^{(K)\top} x) \end{bmatrix}$$



$$softmax(L_n) = \frac{e^{L_n}}{||e^L||}$$

http://deeplearning.stanford.edu/tutorial/supervised/SoftmaxRegression/

35

深度前馈网络

单层

$$h_\theta(x) = \begin{bmatrix} P(y=1|x;\theta) \\ P(y=2|x;\theta) \\ \vdots \\ P(y=K|x;\theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^{K} \exp(\theta^{(j)\top} x)} \begin{bmatrix} \exp(\theta^{(1)\top} x) \\ \exp(\theta^{(2)\top} x) \\ \vdots \\ \exp(\theta^{(K)\top} x) \end{bmatrix}$$

当类别数 k = 2 时
softmax 回归退化为 logistic 回归

When K=2, the softmax regression hypothesis outputs

$$h_\theta(x) = \frac{1}{\exp(\theta^{(1)\top} x) + \exp(\theta^{(2)\top} x^{(i)})} \begin{bmatrix} \exp(\theta^{(1)\top} x) \\ \exp(\theta^{(2)\top} x) \end{bmatrix}$$

$$\Rightarrow$$

$$\begin{aligned} h(x) &= \frac{1}{\exp((\theta^{(1)} - \theta^{(2)})^\top x^{(i)}) + \exp(\vec{0}^\top x)} \left[\exp((\theta^{(1)} - \theta^{(2)})^\top x) \exp(\vec{0}^\top x) \right] \\ &= \begin{bmatrix} \frac{1}{1+\exp((\theta^{(1)}-\theta^{(2)})^\top x^{(i)})} \\ \frac{\exp((\theta^{(1)}-\theta^{(2)})^\top x)}{1+\exp((\theta^{(1)}-\theta^{(2)})^\top x^{(i)})} \end{bmatrix} \\ &= \begin{bmatrix} \frac{1}{1+\exp((\theta^{(1)}-\theta^{(2)})^\top x^{(i)})} \\ 1 - \frac{1}{1+\exp((\theta^{(1)}-\theta^{(2)})^\top x^{(i)})} \end{bmatrix} \end{aligned}$$

36

深度前馈网络

单层

tensor shapes: X[100, 748] W[748,10] b[10]

```
Y = tf.nn.softmax(tf.matmul(X, W) + b)
```

matrix multiply

broadcast on all lines

深度前馈网络

单层

0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	1	0	0	0

actual probabilities, "one-hot" encoded

Cross entropy: $-\sum Y_i'.\log(Y_i)$

computed probabilities

this is a "6"

0	1	2	3	4	5	6	7	8	9
0.1	0.2	0.1	0.3	0.2	0.1	0.9	0.2	0.1	0.1

深度前馈网络

单层

this will become the batch size, 100

```
import tensorflow as tf

X = tf.placeholder(tf.float32, [None, 28, 28, 1])
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))

init = tf.initialize_all_variables()
```

28 x 28 grayscale images

Training = computing variables W and b

深度前馈网络

单层

flattening images

```
# model
Y = tf.nn.softmax(tf.matmul(tf.reshape(X, [-1, 784]), W) + b)
# placeholder for correct answers
Y_ = tf.placeholder(tf.float32, [None, 10])

# loss function
cross_entropy = -tf.reduce_sum(Y_ * tf.log(Y))

# % of correct answers found in batch
is_correct = tf.equal(tf.argmax(Y,1), tf.argmax(Y_,1))
accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))
```

"one-hot" encoded

"one-hot" decoding

深度前馈网络

单层

learning rate

```
optimizer = tf.train.GradientDescentOptimizer(0.003)
train_step = optimizer.minimize(cross_entropy)
```

loss function

41

深度前馈网络

单层

running a Tensorflow computation, feeding placeholders

```
sess = tf.Session()
sess.run(init)

for i in range(1000):
    # load batch of images and correct answers
    batch_X, batch_Y = mnist.train.next_batch(100)
    train_data={X: batch_X, Y_: batch_Y}

    # train
    sess.run(train_step, feed_dict=train_data)

    # success ?
    a,c = sess.run([accuracy, cross_entropy], feed_dict=train_data)

    # success on test data ?
    test_data={X: mnist.test.images, Y_: mnist.test.labels}
    a,c = sess.run([accuracy, cross_entropy, It], feed=test_data)
```

Tip: do this every 100 iterations

42

深度前馈网络

单层

initialisation

```
import tensorflow as tf

X = tf.placeholder(tf.float32, [None, 28, 28, 1])
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
init = tf.initialize_all_variables()

# model
Y=tf.nn.softmax(tf.matmul(tf.reshape(X,[-1, 784]), W) + b)

# placeholder for correct answers
Y_ = tf.placeholder(tf.float32, [None, 10])

# loss function
cross_entropy = -tf.reduce_sum(Y_ * tf.log(Y))

# % of correct answers found in batch
is_correct = tf.equal(tf.argmax(Y,1), tf.argmax(Y_,1))
accuracy = tf.reduce_mean(tf.cast(is_correct,tf.float32))
```

model

success metrics

training step

```
optimizer = tf.train.GradientDescentOptimizer(0.003)
train_step = optimizer.minimize(cross_entropy)

sess = tf.Session()
sess.run(init)

for i in range(10000):
    # load batch of images and correct answers
    batch_X, batch_Y = mnist.train.next_batch(100)
    train_data={X: batch_X, Y_: batch_Y}

    # train
    sess.run(train_step, feed_dict=train_data)

    # success ? add code to print it
    a,c = sess.run([accuracy, cross_entropy], feed=train_data)

    # success on test data ?
    test_data={X:mnist.test.images, Y_:mnist.test.labels}
    a,c = sess.run([accuracy, cross_entropy], feed=test_data)
```

Run

43

深度前馈网络

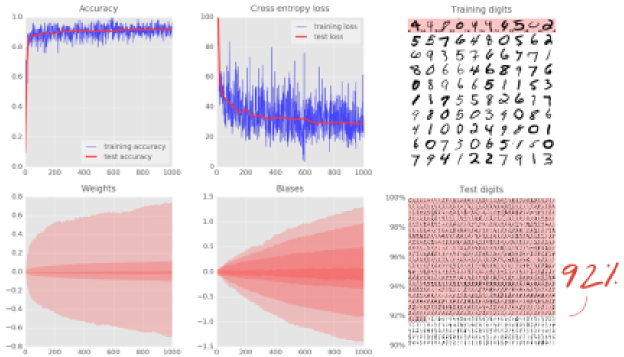
单层

DEMO

44

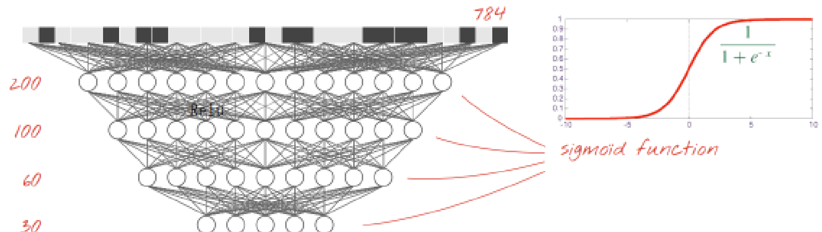
深度前馈网络

单层



深度前馈网络

多层

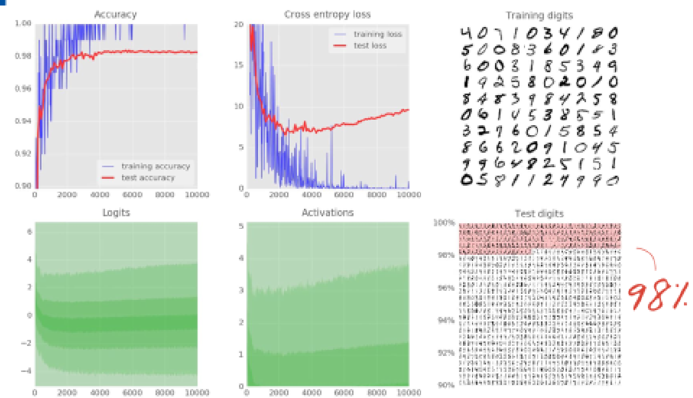


- ☐ Relu
- ☐ Learning rate decay
- ☐ Dropout

深度前馈网络

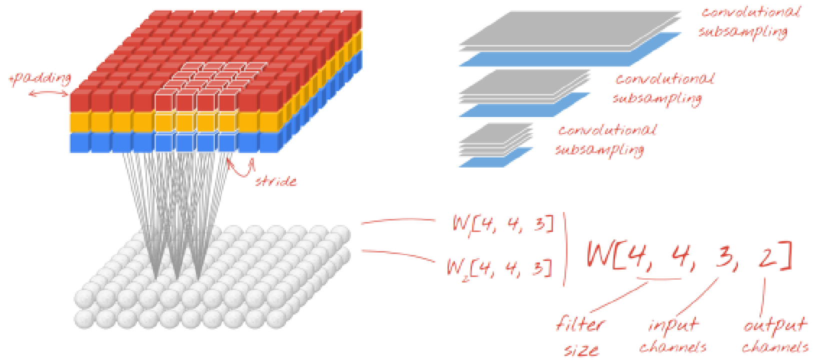
多层

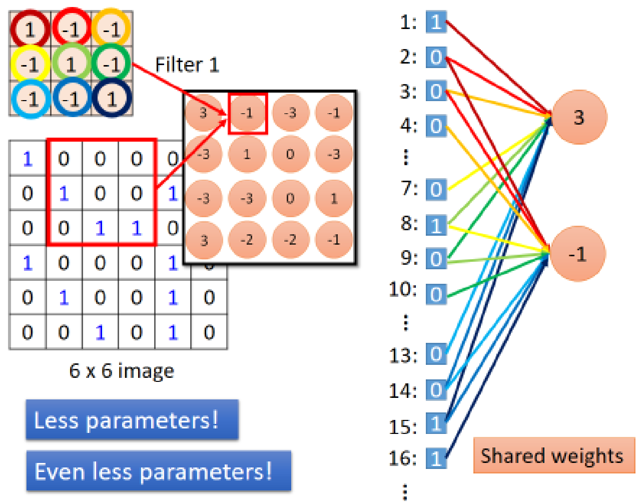
Too many neurons



深度前馈网络

卷积





深度前馈网络

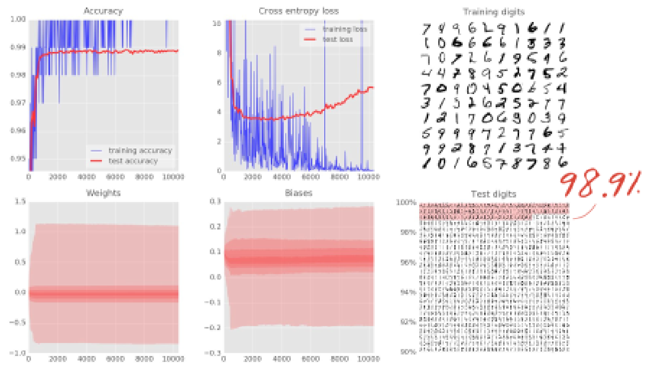
卷积

DEMO

深度前馈网络

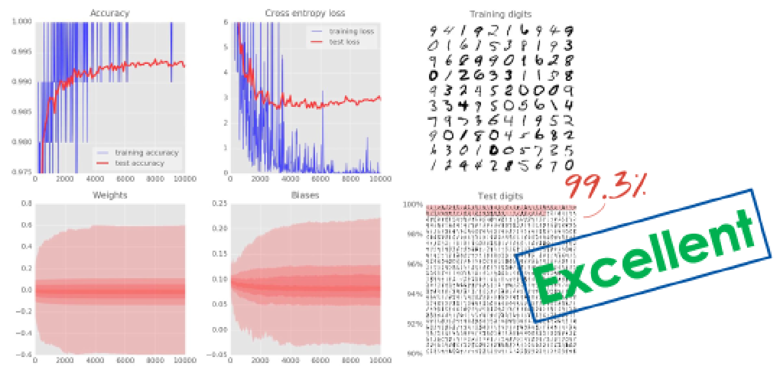
卷积层

Still Too many neurons



深度前馈网络

Bigger卷积+ dropout



THANK YOU

Q&A