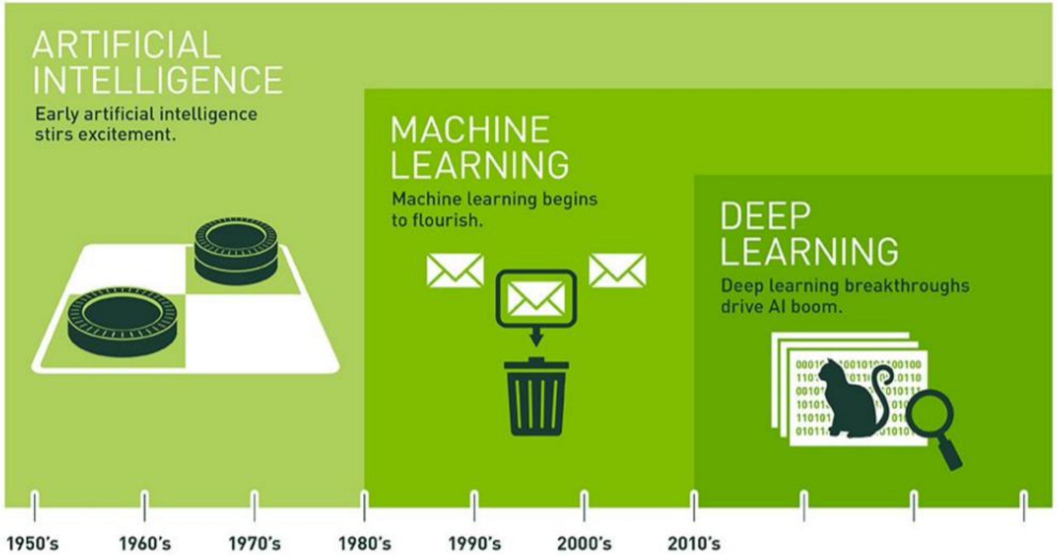
AI, ML & DL

点击此处添加副标题

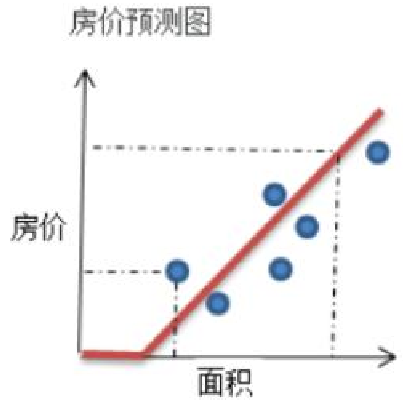
AI，ML，DL

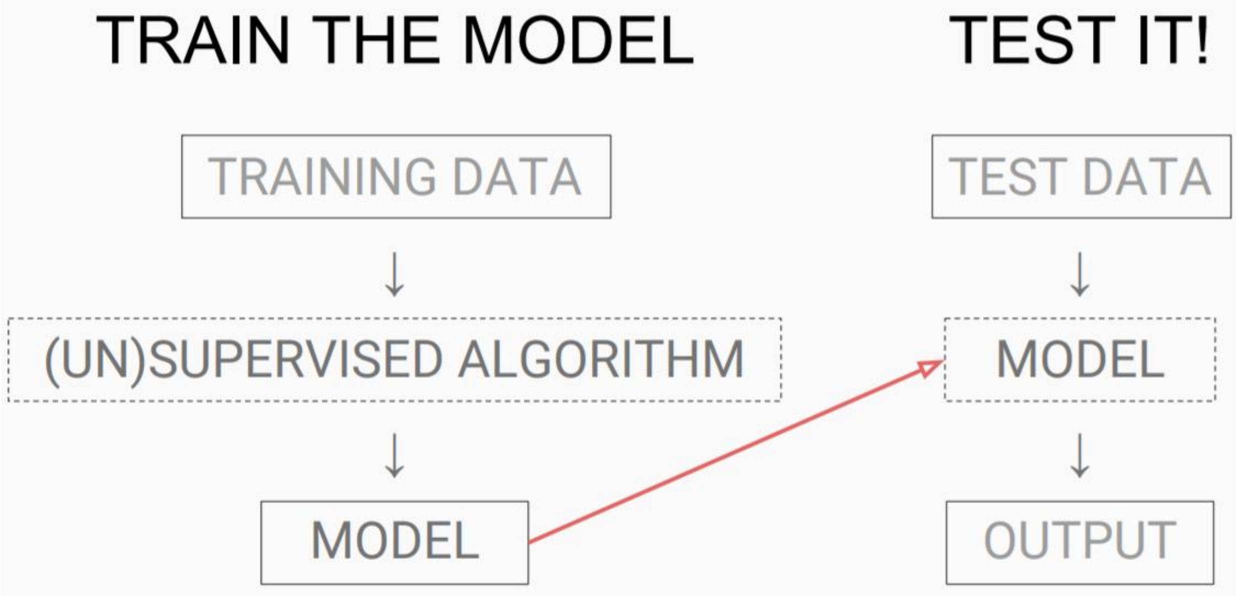


- AI（Artificial Intelligence）人工智能
  - 对人的意识、思维过程的模拟
- ML（Machine Learning）机器学习
  - 实现AI的方法
  - 计算机利用已有数据，得到某个模型，并利用此模型预测未来的一种方法
  - 模式识别、数据挖掘、计算机视觉、语音识别、自然语言处理……
  - 监督学习、无监督学习、强化学习
    - 分类和回归问题
    - SVM 典型的监督学习
    - 神经网络
- DL（Deep Learning）深度学习
  - 机器学习中一种对数据进行表征学习的方法，实质上是层数更多的神经网络学习方法

一个简单问题

- 通过面积预测房价





- 大量的数据通过一个函数进行处理分析，找到相同的规律，然后再根据这个规律分析其他数据

ML

$f($  $)=$ "How are you"

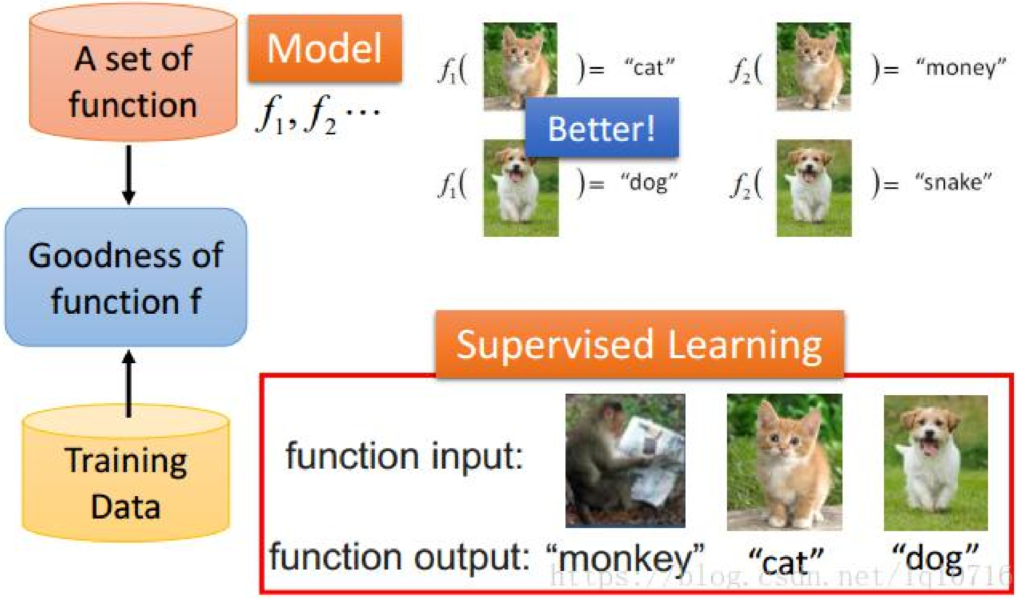
$f($  $)=$ "Cat"

$f($  $)=$ "5-5" (next move)

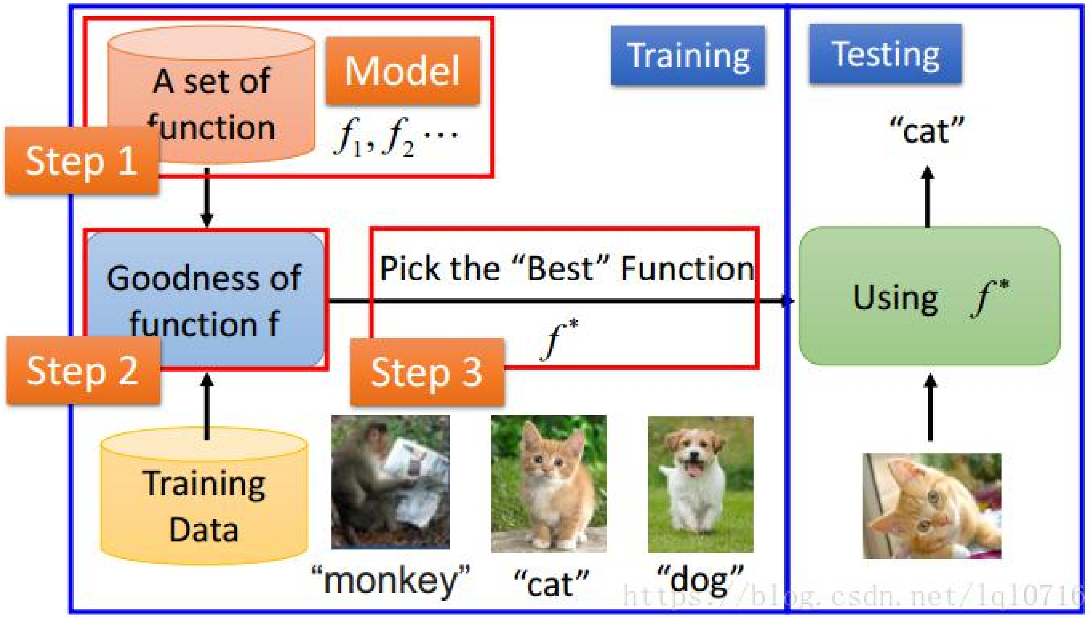
$f($ "Hi" $)=$ "Hello"

(what the user said) (system response)

目标:

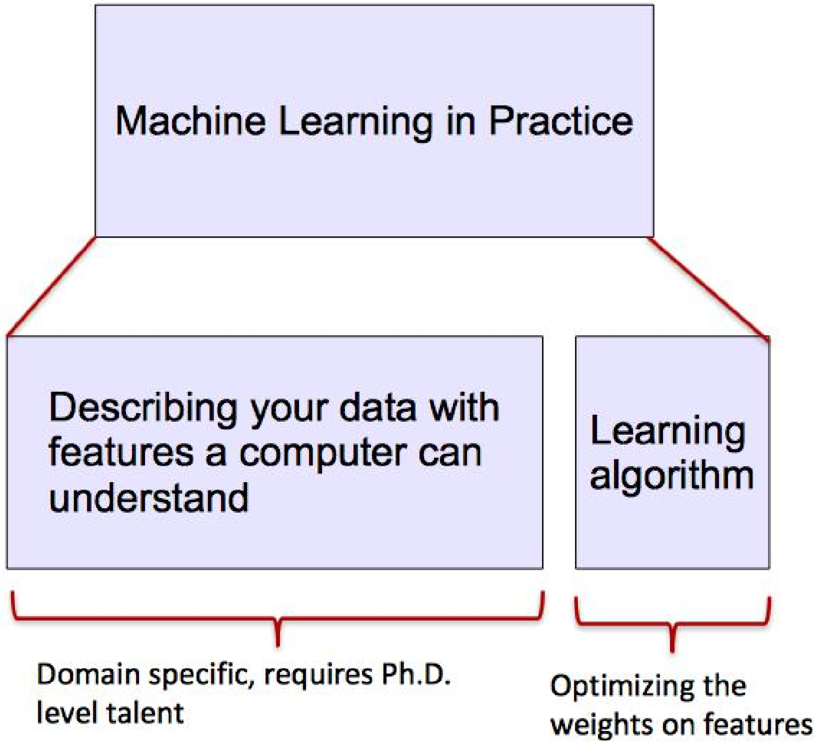


过程:

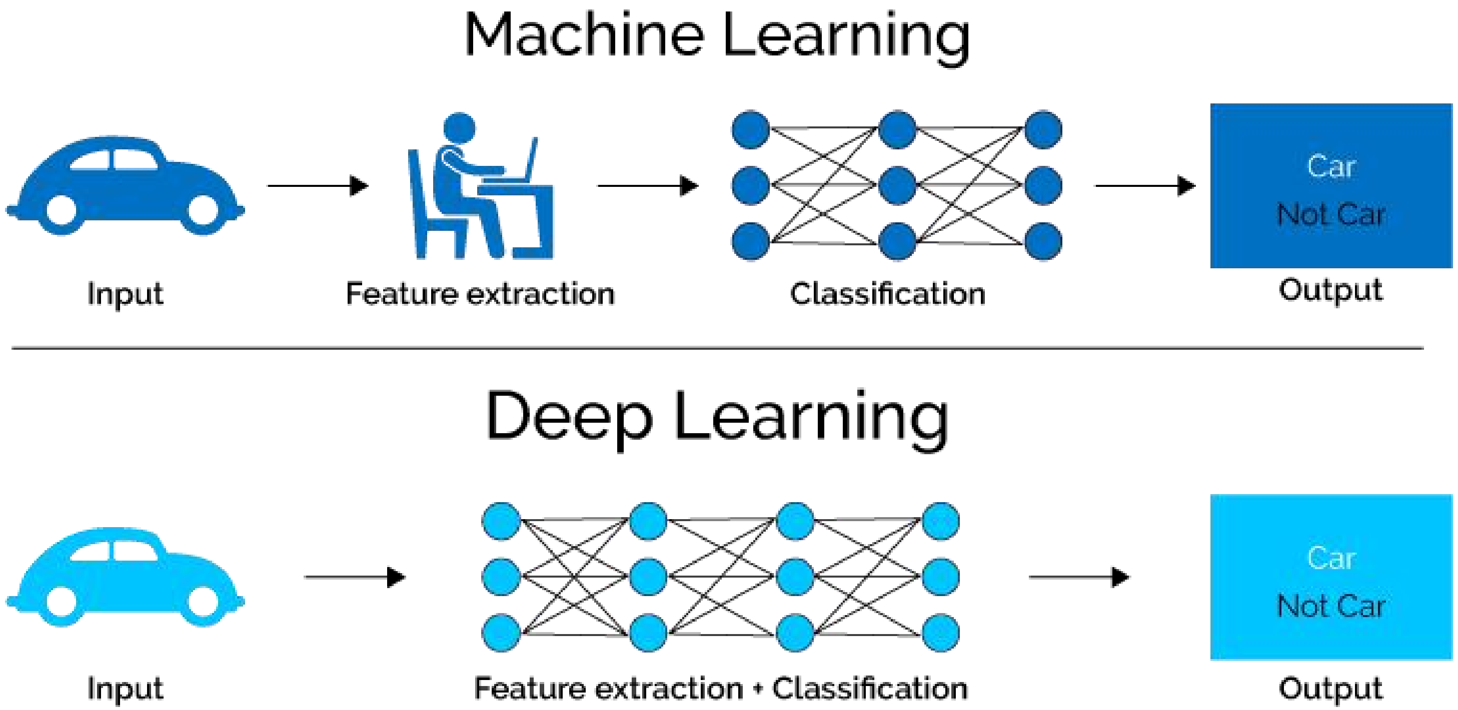


ML -> DL

- s



Machine Learning in Practice
Describing your data with features a computer can understand
Learning algorithm
Domain specific, requires Ph.D. level talent
Optimizing the weights on features



Machine Learning
Input
Feature extraction
Classification
Car
Not Car
Output
Deep Learning
Input
Feature extraction + Classification
Car
Not Car
Output

Why DL?

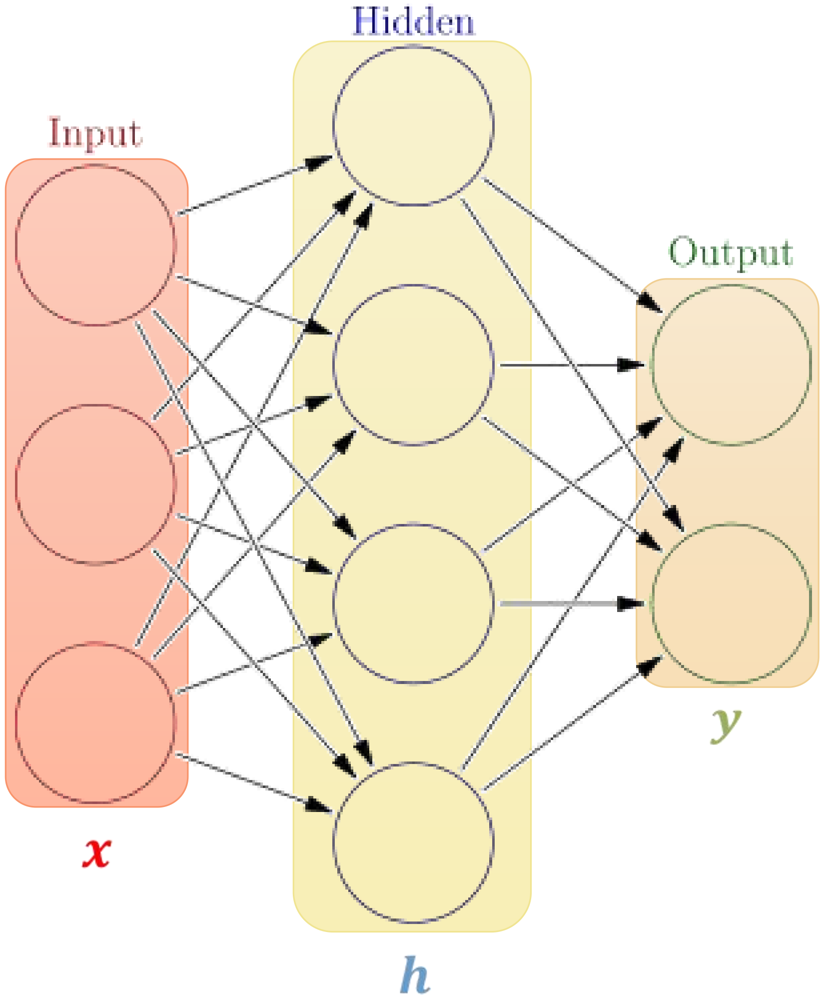
- 人工定义的特征往往是不全面、不完整的，需要消耗大量的时间精力去设计和验证
- DL学习到特征则更灵活和迅速
- End-to-end的学习机制更有效
- DL可以从大量数据中学习

NN

- NN（Neural Networks）神经网络
  - Neurons 神经元、Layers 层
  - Activation Functions 激活函数
  - 学习参数
    - Weights 权重
    - Biases 偏移

$$h = \sigma(\mathbf{W}_1 x + b_1)$$

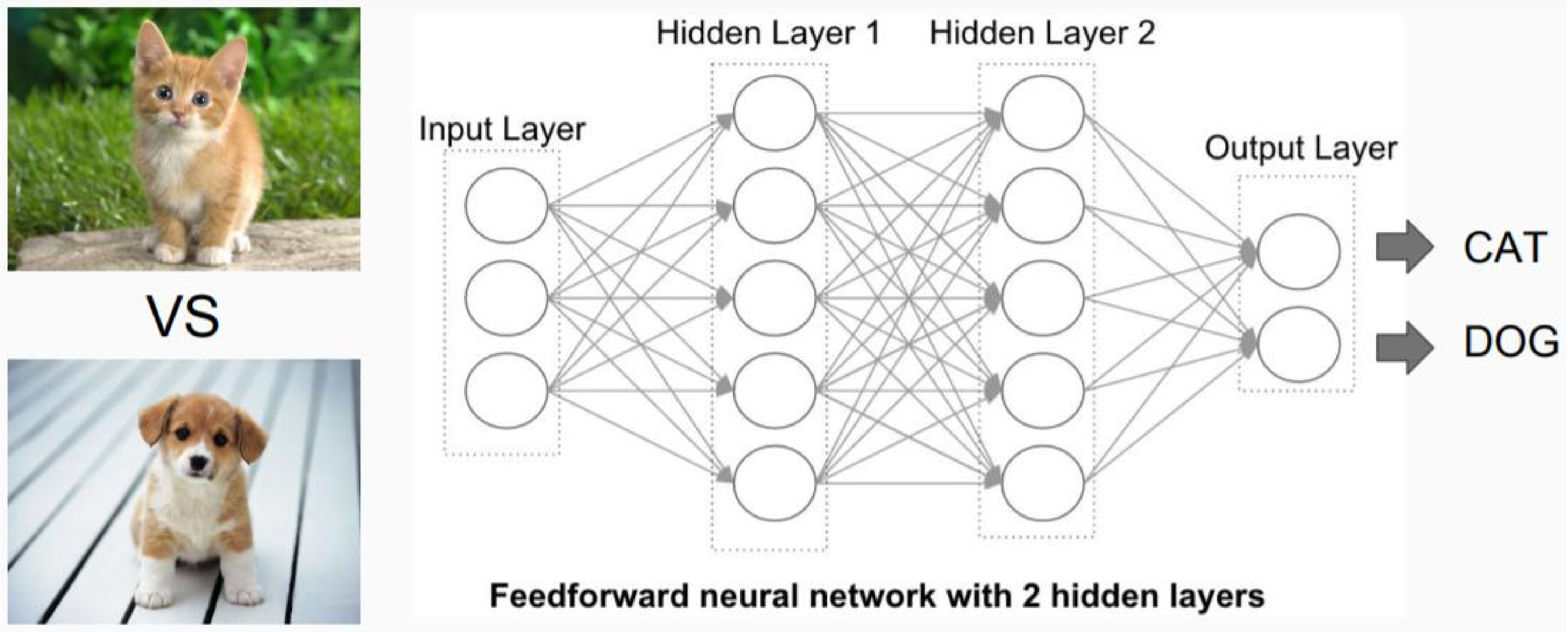
$$y = \sigma(W_2 h + b_2)$$



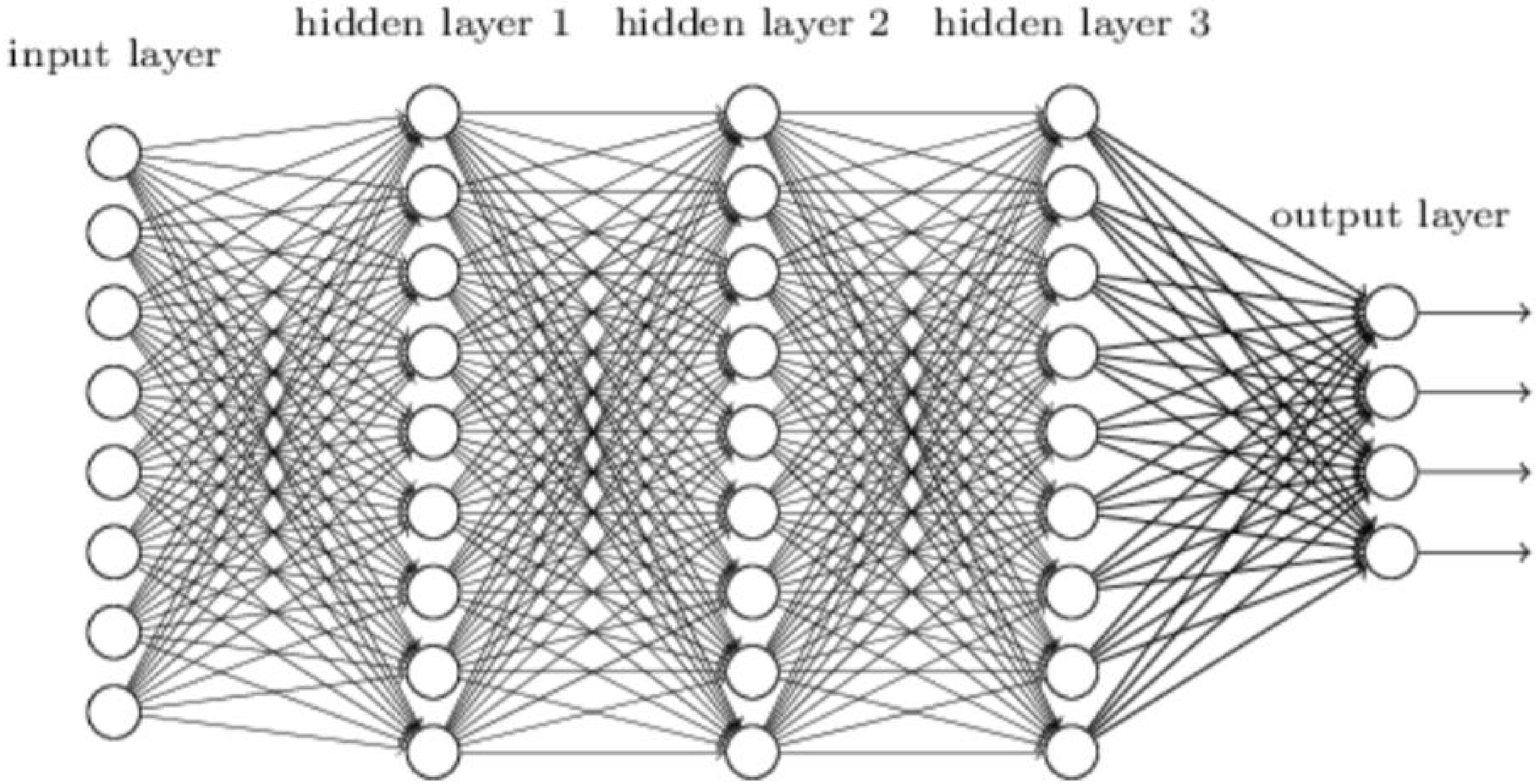
4 + 2 = 6 neurons (不包含输入)
[3 x 4] + [4 x 2] = 20 weights
4 + 2 = 6 biases

26 learnable **parameters**

图像分类

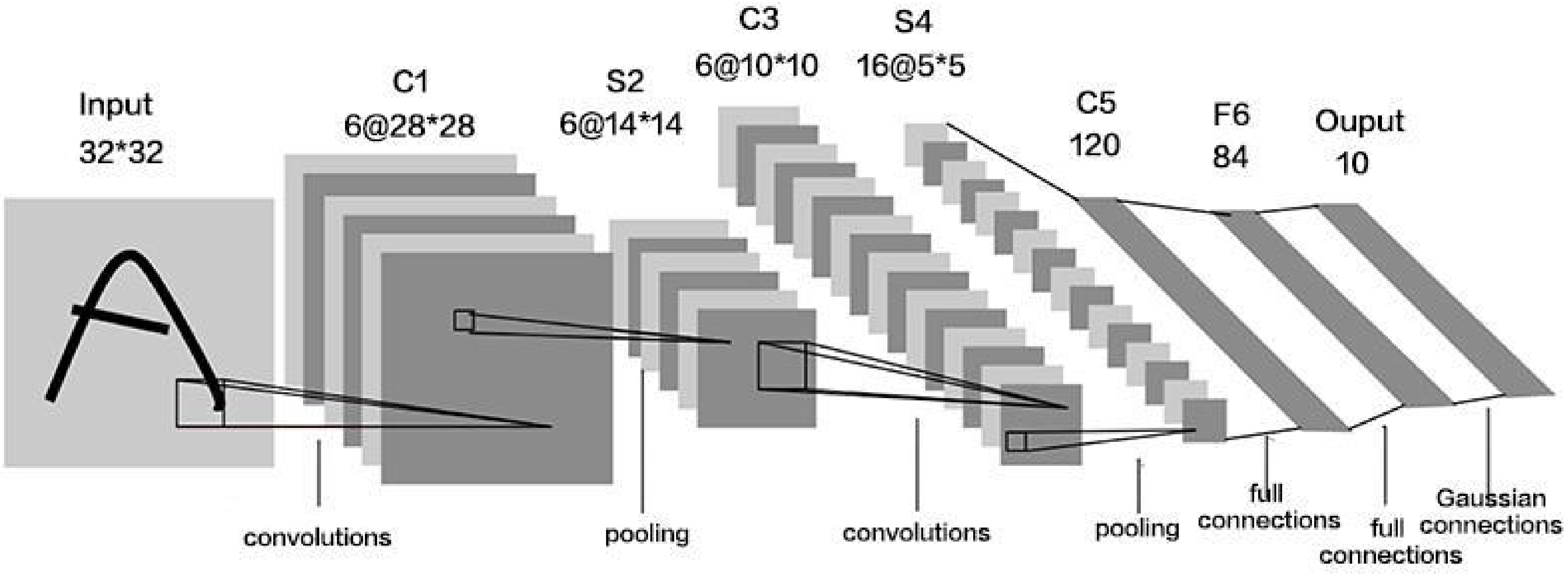

VS
Hidden Layer 1
Hidden Layer 2
Input Layer
Output Layer
CAT
DOG
Feedforward neural network with 2 hidden layers

全连接模型


input layer
hidden layer 1
hidden layer 2
hidden layer 3
output layer

卷积神经网络

- 局部卷积
- 参数共享
- 多卷积核
- 下采样

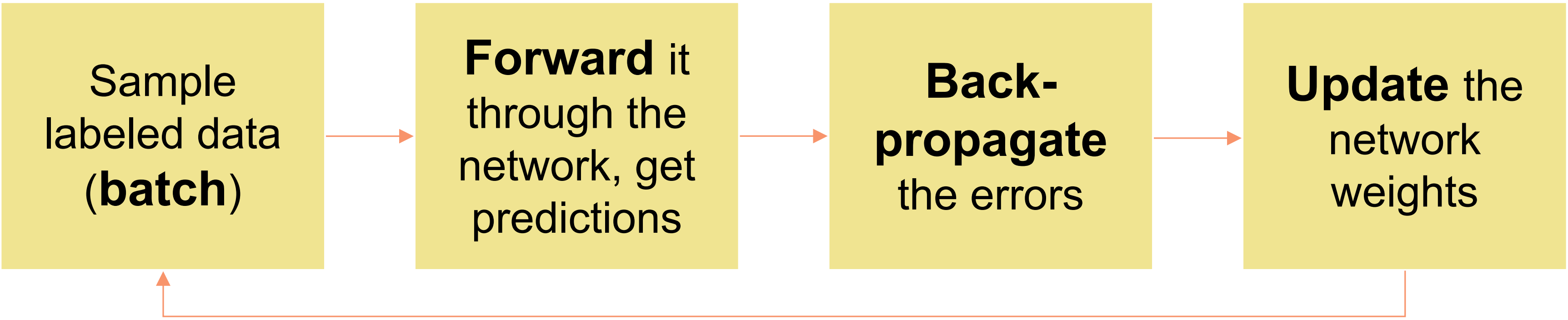


Training（训练）

- 目标：根据训练数据的输入计算输出，根据计算得到的输出与正确输出的误差，调节模型训练参数以减少误差
  - Cost/Loss Functions 代价/损失 函数：量化误差
    - 分类问题：Softmax Loss　　　　　　　　回归问题：MSE Loss 均方差

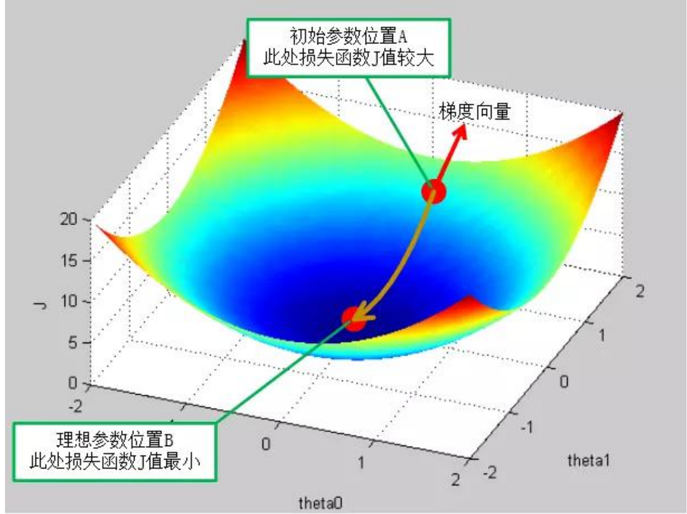
$$P(y=j \mid \mathbf{x}) = \frac{e^{\mathbf{x}^\mathsf{T}\mathbf{w}_j}}{\sum_{k=1}^{K} e^{\mathbf{x}^\mathsf{T}\mathbf{w}_k}}$$

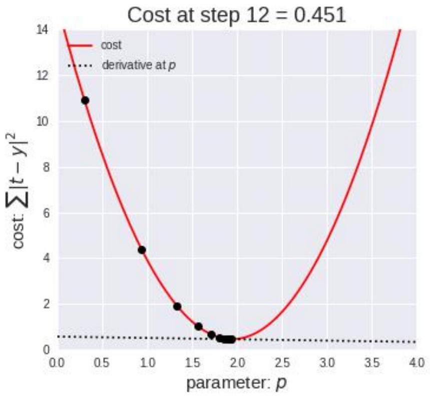
$$J(\theta) = \frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)} - \hat{y}^{(i)}\right)^2$$

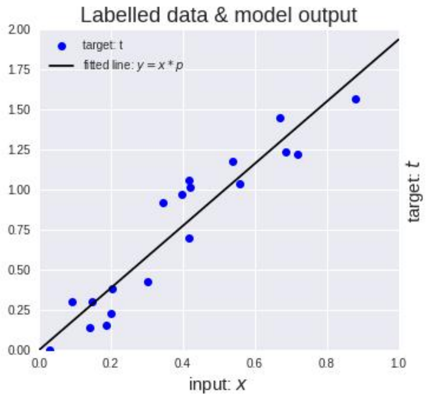


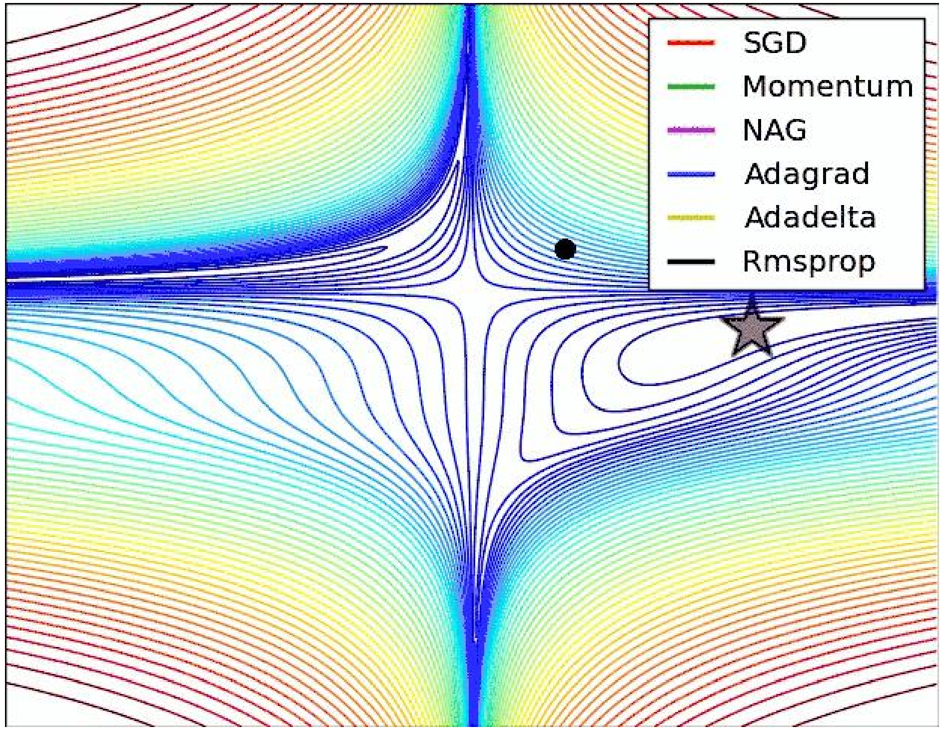
参数更新

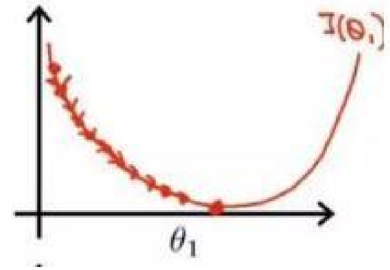
- 梯度下降
  - 梯度决定了努力的方向
  - 步长（Learning rates）决定了脚步的大小

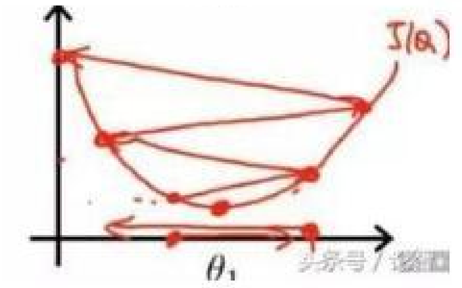






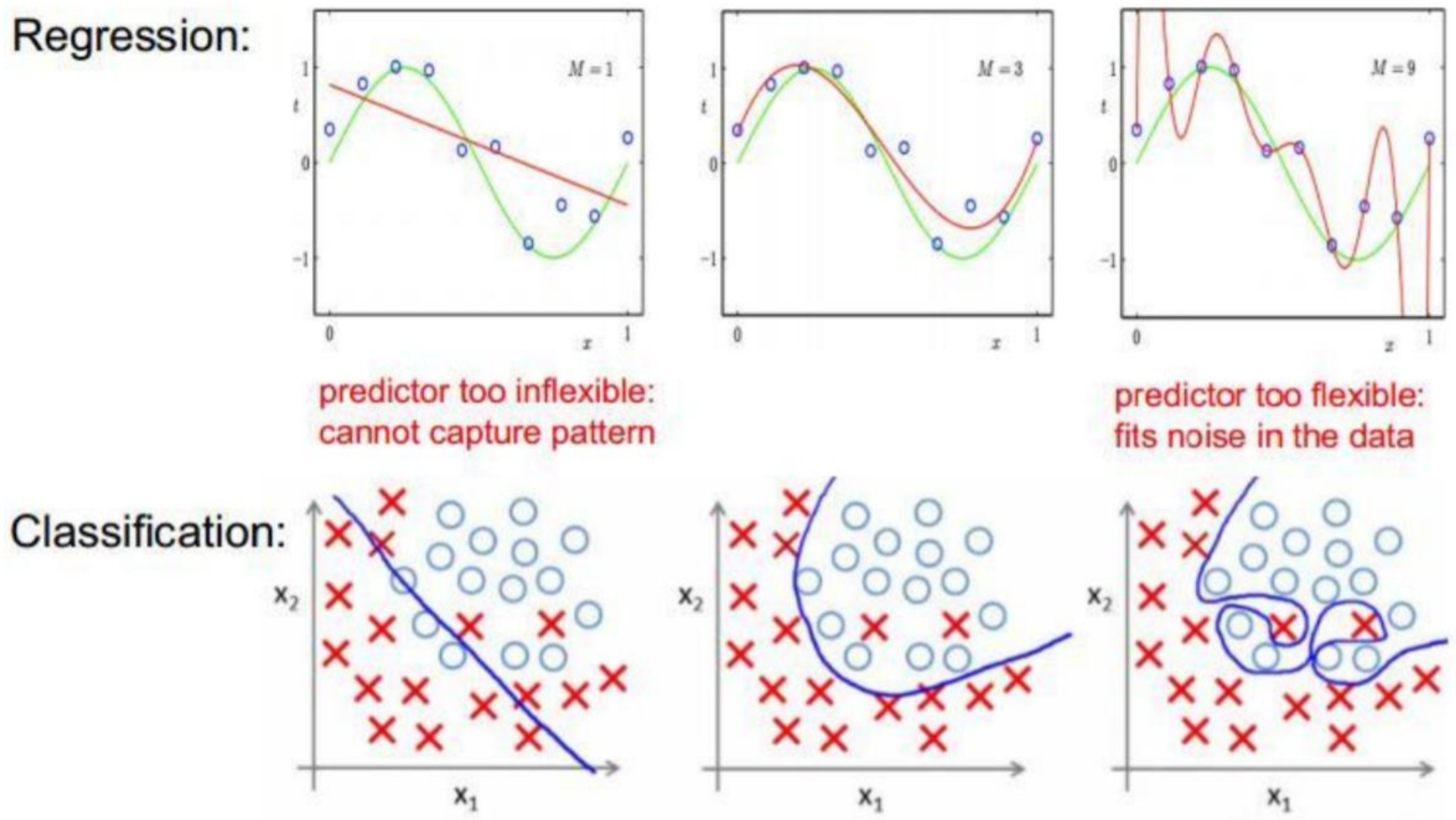






模型的能力评价

- 欠拟合(under-fitting)和过拟合(over-fitting)

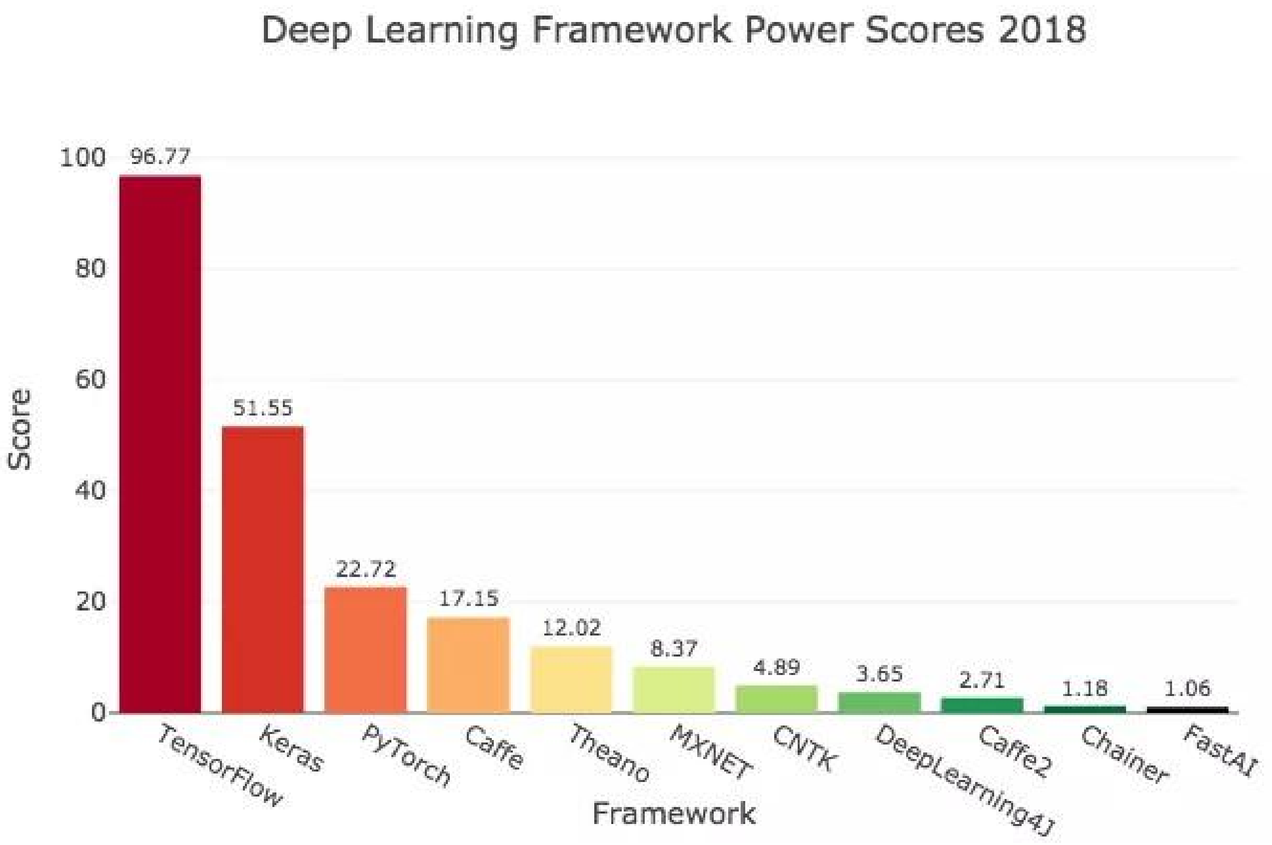


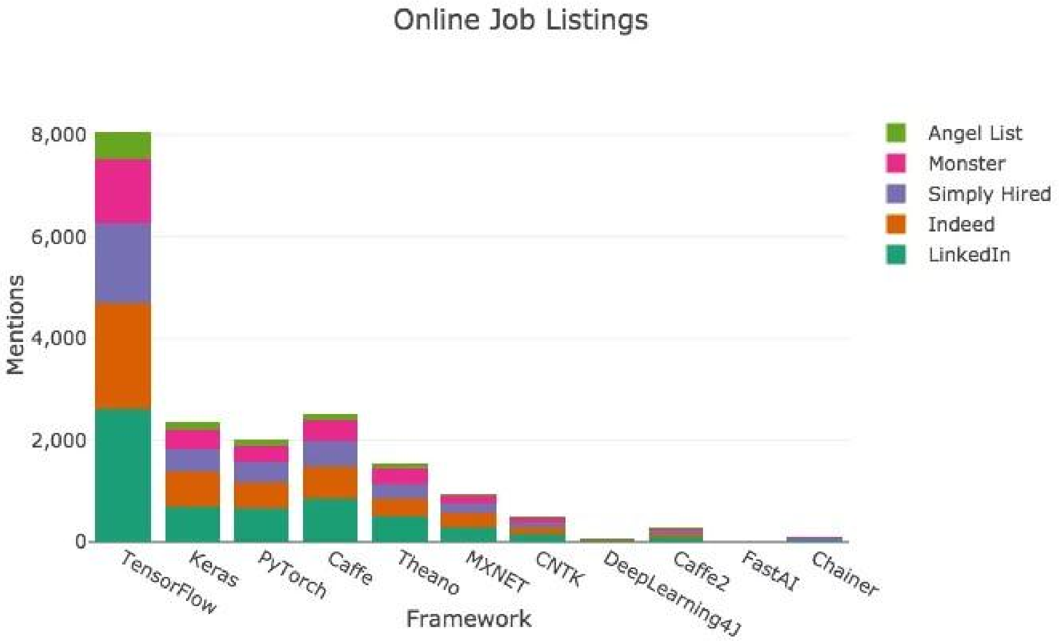
深度学习框架

- Caffe
- Theano
- TensorFlow
- MXNet
- Torch/PyTorch

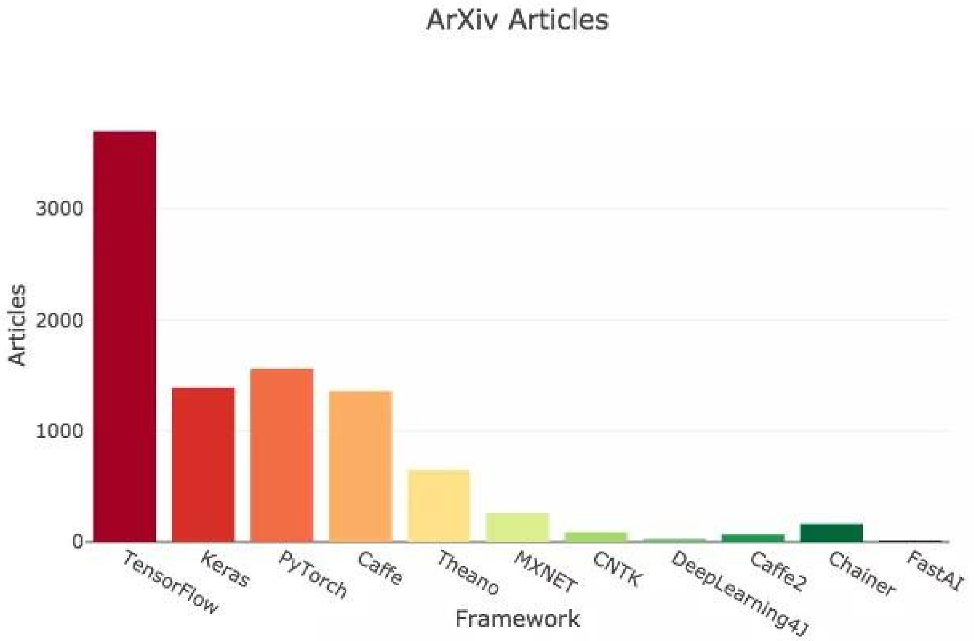
影响力排名

- 基本都使用Python做为接口语言（基本都使用C++做为开发语言）
- Keras，FastAI 为从属框架
- 若干大公司背书：Google、FaceBook、Amazon、Microsoft





Online Job Listings
8,000
6,000
4,000
2,000
0
Mentions
Angel List
Monster
Simply Hired
Indeed
LinkedIn
TensorFlow
Keras
PyTorch
Caffe
Theano
MXNET
CNTK
DeepLearning4J
Caffe2
FastAI
Chainer
Framework



ArXiv Articles
3000
2000
1000
0
Articles
TensorFlow
Keras
PyTorch
Caffe
Theano
MXNET
CNTK
DeepLearning4J
Caffe2
Chainer
FastAI
Framework

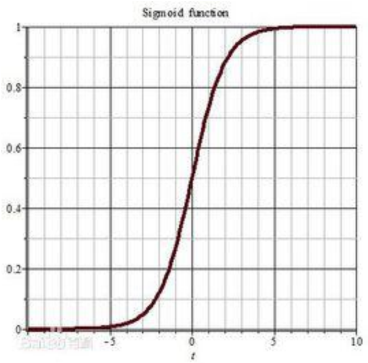
神经网络训练简单示例

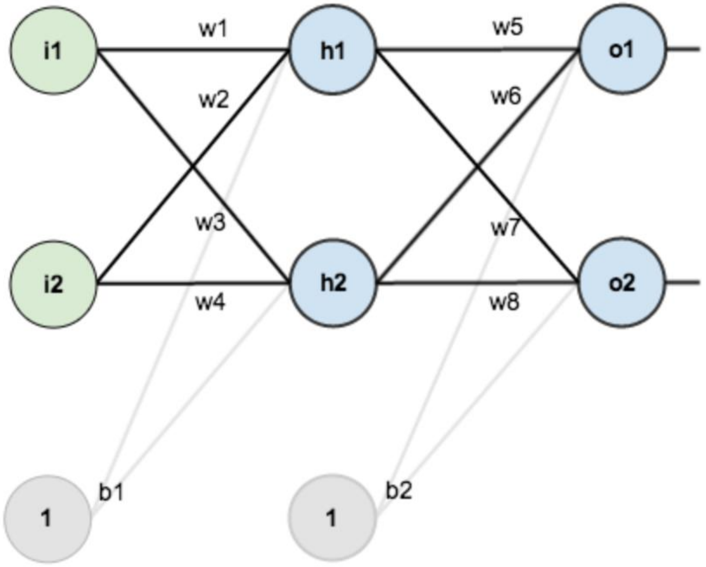
- 三层全连接网络
  - 输入层：两个神经元 i1、i2
  - 隐层：h1、h2
  - 输出层：o1、o2

  - wi表示连接上的weights，bi表示biases
  - 神经元上的激活函数是sigmoid

$$S(x) = \frac{1}{1+e^{-x}}$$

$$S'(x) = \frac{e^{-x}}{(1+e^{-x})^2} = S(x)(1-S(x))$$

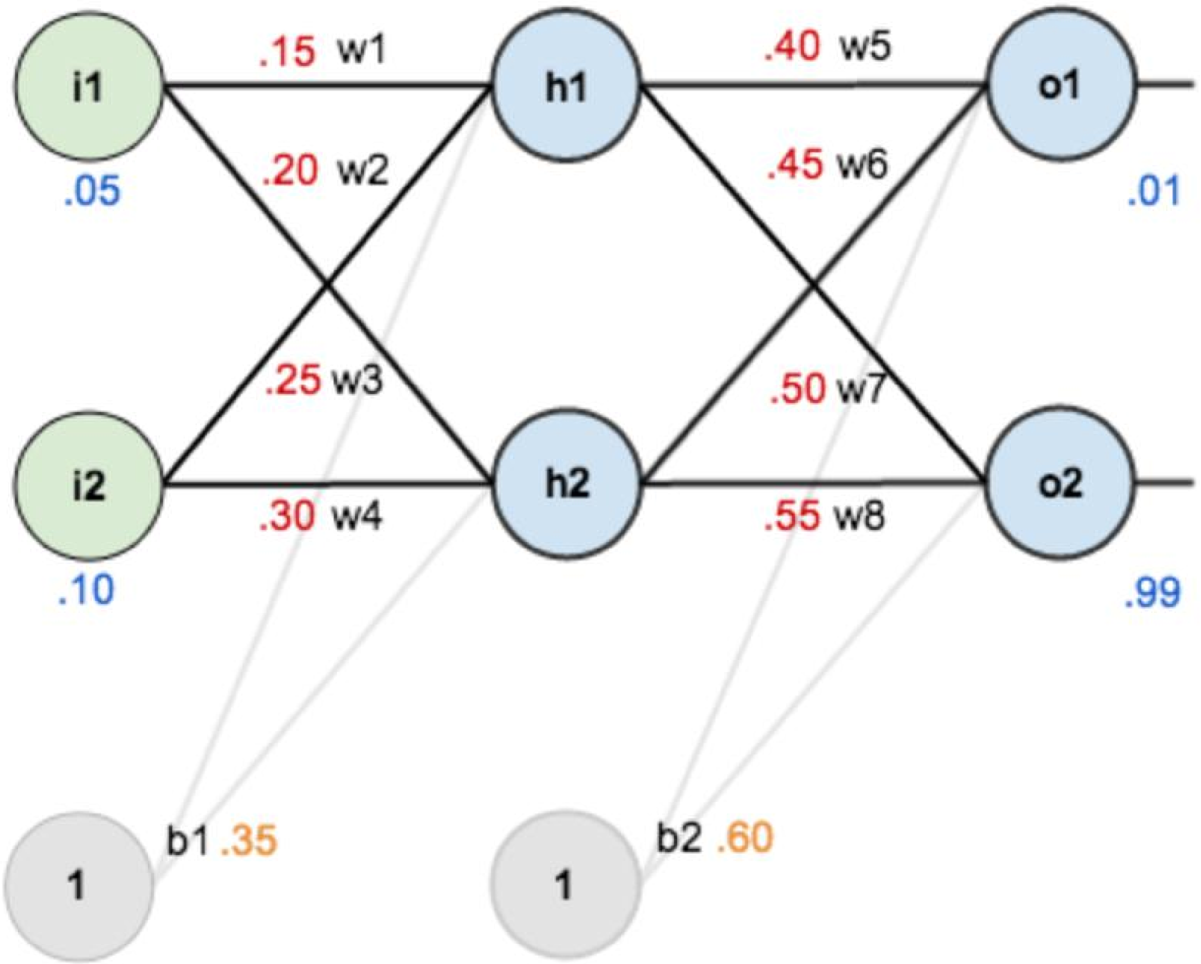




训练过程的初始状态

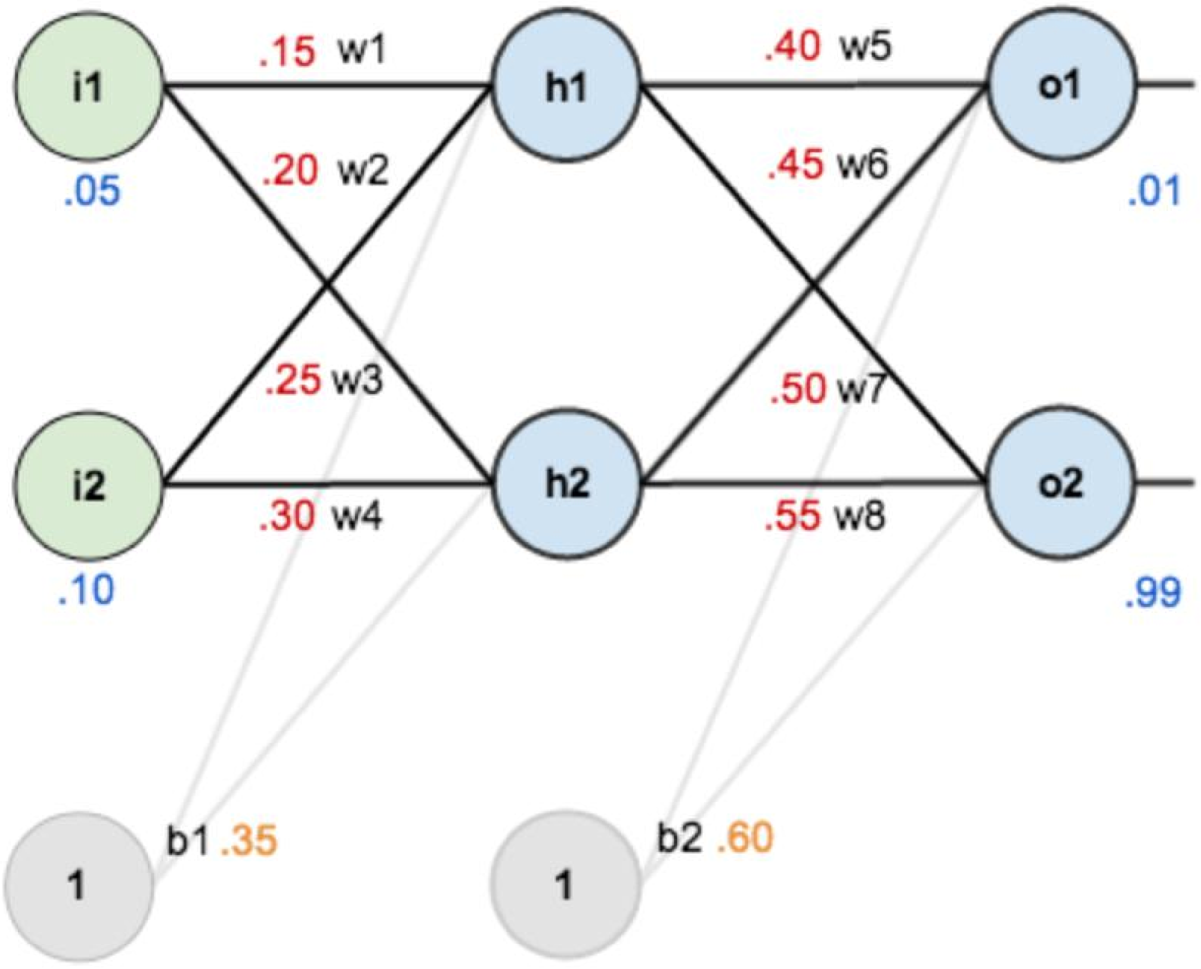
- 当前训练样本
  - 输入 = [0.05, 0.10]
  - 预期输出 = [0.01, 0.99]
  - 当前w、b如图(注意：为简化计算此处同层共享b，且b固定)

- 训练目标：调节w使得真实输出尽量接近预期输出



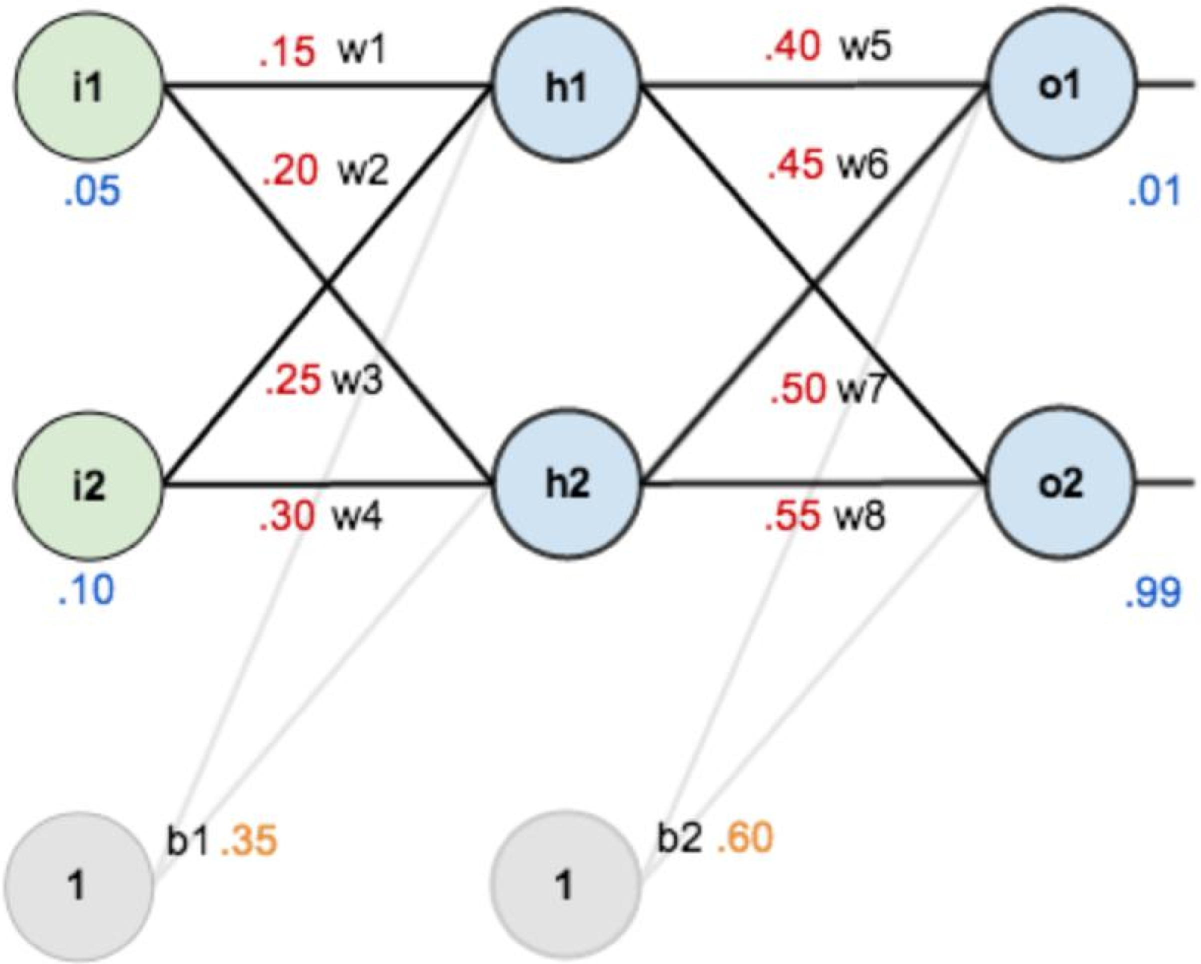
前向计算：输入层->隐层

- h1为例:
  - net_{h1} = w1*i1+w2*i2+b1 = 0.15*0.05+0.2*0.1+0.35=0.3775
  - out_{h1}=1/(1+exp(-0.3775))=0.593269992
- 同理可得
  - out_{h2}=0.596884378



前向计算：隐层->输出层

- 以o1为例:
  - $net_{o1}=w5*out_{h1}+w6*out_{h2}+b2=0.4*0.593269992+0.45*0.596884378+0.6=1.105905967$
  - $out_{o1}=1/(1+exp(-1.105905967))=0.75136507$
- 同理可得
  - $out_{o2}=0.772928465$
- 实际输出与预期输出存在差异，需要更新网络参数



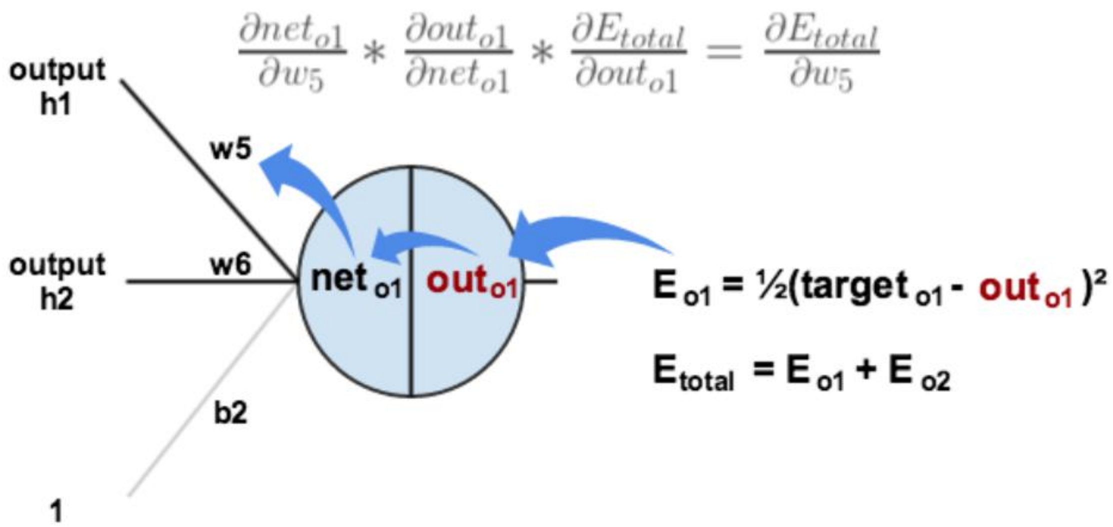
反向传播 Loss函数

- 定义Loss函数：
  - $E_{total}=\sum(\text{target} - \text{output})^2/2$
  - $E_{o1}=(0.01-0.75136507)^2/2=0.274811083$
  - $E_{o2}=0.023560026$
  - $E_{total}=E_{o1}+E_{o2}=0.298371109$

反向传播：输出层参数更新

- 以w5为例：

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$



$$E_{total} = \frac{1}{2}(target_{o1} - out_{o1})^2 + \frac{1}{2}(target_{o2} - out_{o2})^2$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = 2 * \frac{1}{2}(target_{o1} - out_{o1})^{2-1} * -1 + 0$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = -(target_{o1} - out_{o1}) = -(0.01 - 0.75136507) = 0.74136507$$

$$out_{o1} = \frac{1}{1+e^{-net_{o1}}}$$

$$\frac{\partial out_{o1}}{\partial net_{o1}} = out_{o1}(1 - out_{o1}) = 0.75136507(1 - 0.75136507) = 0.186815602$$

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$\frac{\partial net_{o1}}{\partial w_5} = 1 * out_{h1} * w_5^{(1-1)} + 0 + 0 = out_{h1} = 0.593269992$$

更新参数

- 计算梯度，根据梯度和学习率(learning rate)更新参数

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

$$\frac{\partial E_{total}}{\partial w_5} = 0.74136507 * 0.186815602 * 0.593269992 = 0.082167041$$

$$w_5^+ = w_5 - \eta * \frac{\partial E_{total}}{\partial w_5} = 0.4 - 0.5 * 0.082167041 = 0.35891648$$

$$w_6^+ = 0.408666186$$

$$w_7^+ = 0.511301270$$

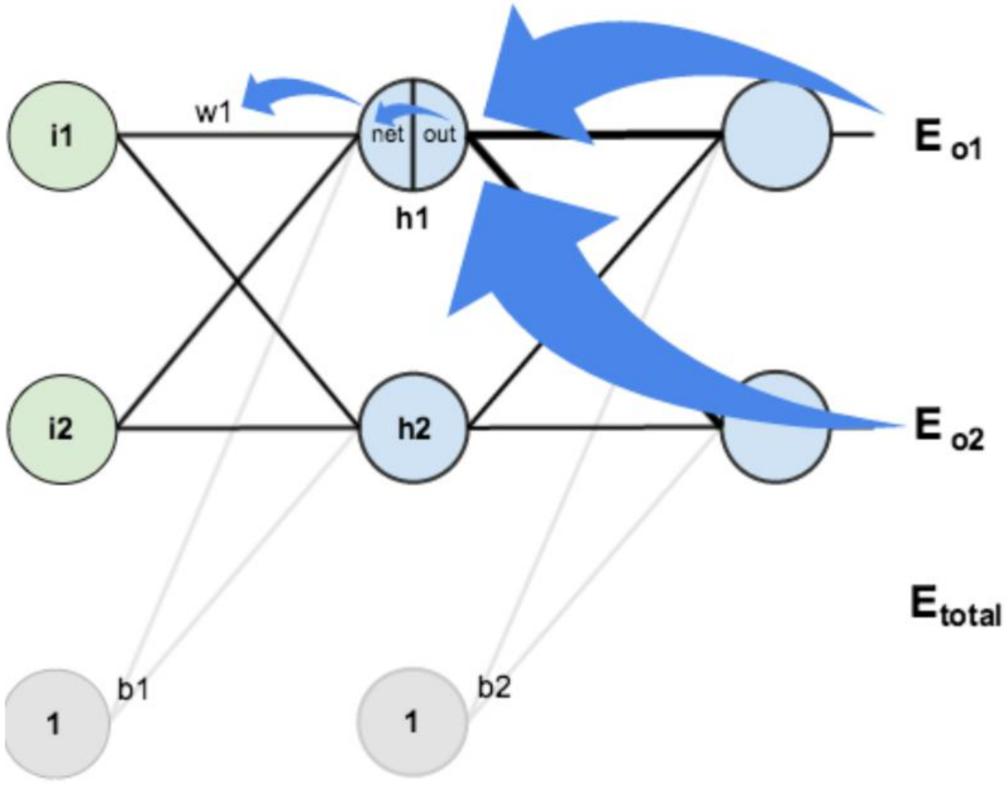
$$w_8^+ = 0.561370121$$

反向传播：隐层参数更新

- 以w1为例：

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$



$E_{total} = E_{o1} + E_{o2}$

$$\frac{\partial E_{o1}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial out_{h1}}$$

$$\frac{\partial E_{o1}}{\partial net_{o1}} = \frac{\partial E_{o1}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} = 0.74136507 * 0.186815602 = 0.138498562$$

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$\frac{\partial net_{o1}}{\partial out_{h1}} = w_5 = 0.40$$

$$\frac{\partial E_{o1}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial out_{h1}} = 0.138498562 * 0.40 = 0.055399425$$

- 同理可得

$$\frac{\partial E_{o2}}{\partial out_{h1}} = -0.019049119$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}} = 0.055399425 + -0.019049119 = 0.036350306$$

$$out_{h1} = \frac{1}{1+e^{-net_{h1}}}$$

$$\frac{\partial out_{h1}}{\partial net_{h1}} = out_{h1}(1 - out_{h1}) = 0.59326999(1 - 0.59326999) = 0.241300709$$

$$net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$$

$$\frac{\partial net_{h1}}{\partial w_1} = i_1 = 0.05$$

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial w_1} = 0.036350306 * 0.241300709 * 0.05 = 0.000438568$$

$$w_1^+ = w_1 - \eta * \frac{\partial E_{total}}{\partial w_1} = 0.15 - 0.5 * 0.000438568 = 0.149780716$$

$$w_2^+ = 0.19956143$$

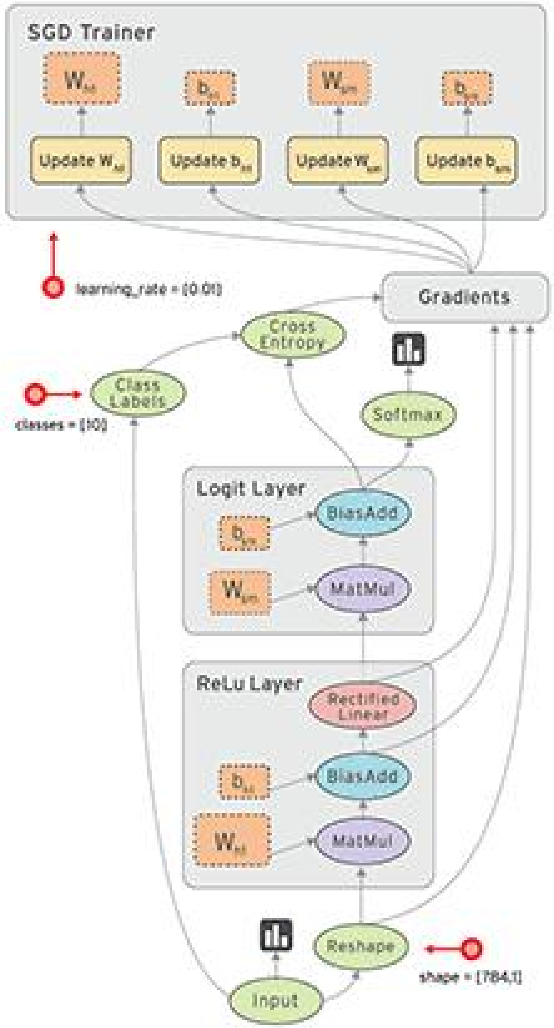
$$w_3^+ = 0.24975114$$

$$w_4^+ = 0.29950229$$

- 一次反向传播完成，再次正向计算可得到新的实际输出
- 迭代上述运算过程10000次后，输出为[0.015912196, 0.984065734]，已经非常接近期望输出

利用张量和TensorFlow处理

- 采用数据流图，用于数值计算的开源软件库
  - 不是一个严格的“神经网络”库，只要可以把计算表示为一个数据流图，就可以使用TF
  - 具有自动求微分的能力（极大降低了机器学习门槛）
  - 很好的可移植性（CPU、GPU、服务器、手机……）
  - 多语言支持，底层实现C++，上层使用Python、GO、Javascript……
  - ……



张量

- 深度学习框架中的核心概念
  - 标量、向量、矩阵分别为：0、1、2阶张量
  - 一张RGB图像可表示为3阶张量（高度、宽度、通道）
  - 深度学习中经常需要使用4阶张量（图像+batchSize）

$h =$

  - 上述示例转换为张量表示：

$$i = \begin{bmatrix} 0.05 \\ 0.10 \end{bmatrix} \quad target = \begin{bmatrix} 0.01 \\ 0.99 \end{bmatrix} \quad w_h = \begin{bmatrix} 0.15 & 0.20 \\ 0.25 & 0.30 \end{bmatrix} \quad w_o = \begin{bmatrix} 0.40 & 0.45 \\ 0.50 & 0.55 \end{bmatrix}$$

$$b_1 = \begin{bmatrix} 0.35 \\ 0.35 \end{bmatrix} \quad b_2 = \begin{bmatrix} 0.60 \\ 0.60 \end{bmatrix}$$

对应到Tensorflow定义

- i = tf.constant([[0.05],[0.10]], name="i")
- target = tf.constant([[0.01],[0.99]], name="target")
- b1 = tf.constant([[0.35], [0.35]], name="b1")
- b2 = tf.constant([[0.60], [0.60]], name="b2")

- w_h = tf.Variable([[0.15, 0.20],[0.25, 0.30]], name="w_h")
- w_o = tf.Variable([[0.40, 0.45],[0.50, 0.55]], name="w_o")

相关运算

$$out_h = \frac{1}{1+e^{-net_h}}$$

$$net_h = w_h i + b_1$$

$$out_o = \frac{1}{1+e^{-net_o}}$$

$$net_o = w_o out_h + b_2$$

$$E_{total} = \frac{1}{2}\sum \left\| target - out_o \right\|^2$$

对应Tensorflow定义

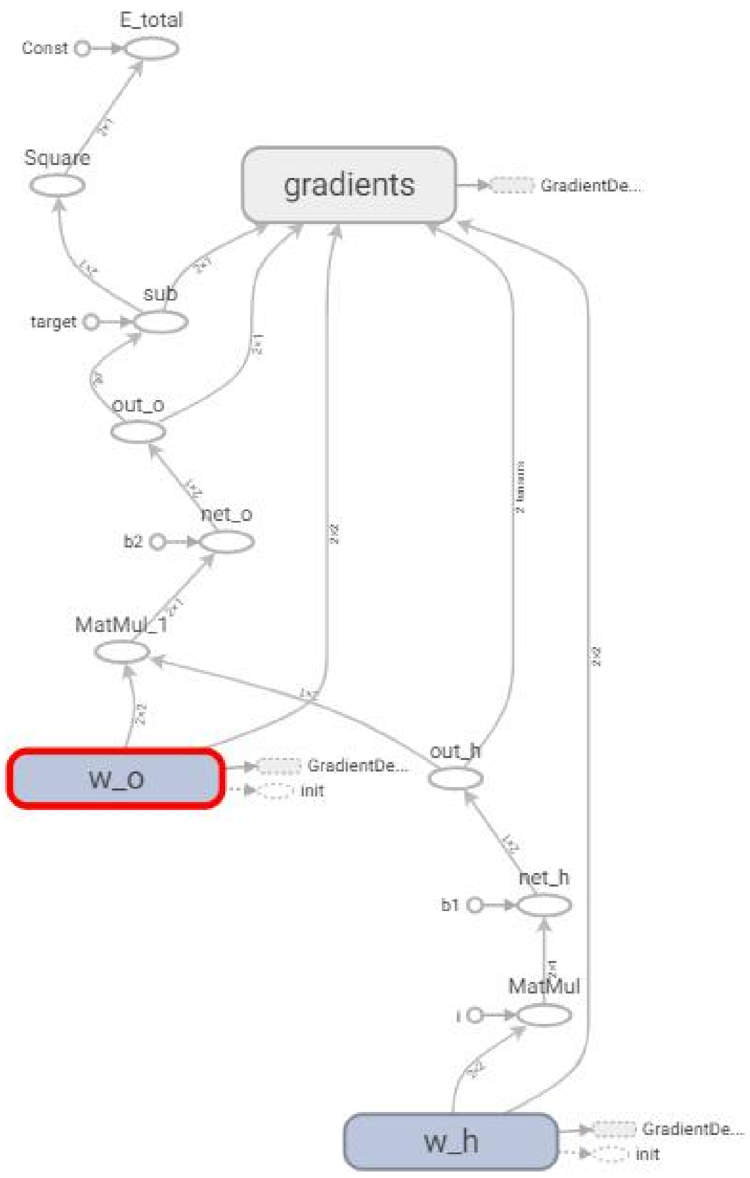
- net_h = tf.add(tf.matmul(w_h, i), b1, name="net_h")
- out_h = tf.nn.sigmoid(net_h, name="out_h")

- net_o = tf.add(tf.matmul(w_o, out_h), b2, name="net_o")
- out_o = tf.nn.sigmoid(net_o, name="out_o")

- E_total = tf.reduce_mean(tf.square(target - out_o), name="E_total")

计算图及其运行

- 计算图（Graph）
  - 定义张量Tensor和操作Operation后构建出计算图
  - 在构建计算图过程中并不真实运算，只是表示计算任务
- 会话（Session）
  - 提供执行Operation和计算Tensor值的环境
  - Session.run(fetches, feed_dict=None)



训练模型

- train_step = tf.train.GradientDescentOptimizer(0.5).minimize(E_total)

- init_op = tf.global_variables_initializer()

```
with tf.Session() as sess:
    sess.run(init_op)
    for i in range(10000):
        sess.run(train_step)
        #train_step.run(session = sess)

    print(sess.run(out_o))
```

可视化显示

- 代码：
  - writer = tf.summary.FileWriter("./graph/nn-3",sess.graph)

- 运行 Tensorboard --logdir=./graph/nn-3

- http://localhost:6006

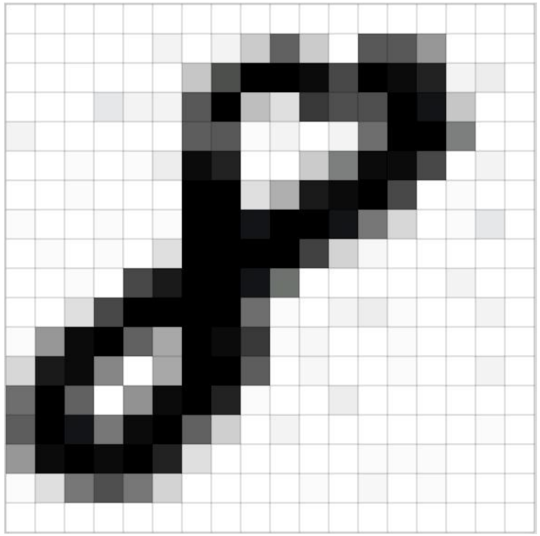
作业

- 完成Tensorflow环境的搭建
- 使用Tensorflow训练一个神经网络，能够拟合函数 sin(x) + cos(x)

- 可使用pytorch、caffe等其他框架完成

利用Tensorflow+CNN完成图像分类

- 卷积神经网络CNN
  - 必要性：全连接网络参数数量爆炸
    - 一个100x100x1的灰度图像输入，相当于输入层包含10000个神经元
    - 假设下一层与输入同样尺寸，则需要10000x10000个权重参数
  - CNN思路
    - 局部感知
    - 参数共享
    - 单层多卷积核
    - 多卷积层
    - 下采样
  - CNN层参数：
    - kernel_size、stride、padding、kernel_num



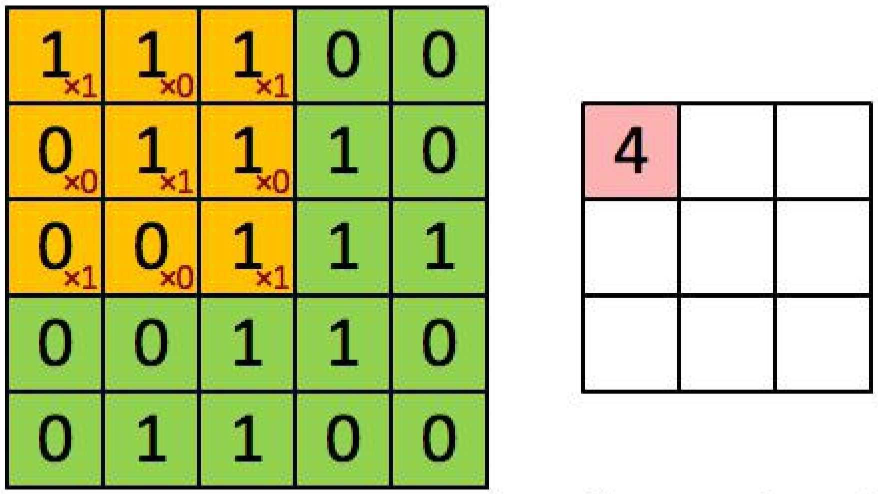
局部感知和卷积核运算

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

输入

1	0	1
0	1	0
1	0	1

卷积核



Image

Convolved Feature

卷积步长stride和边界padding

- stride：卷积每次移动的像素数量
  - 对输出尺寸有较大影响

INPUT IMAGE

18	54	51	239	244	188
55	121	75	78	95	88
35	24	204	113	109	221
3	154	104	235	25	130
15	253	225	159	78	233
68	85	180	214	245	0

WEIGHT

1	0	1
0	1	0
1	0	1

429

INPUT IMAGE

18	54	51	239	244
55	121	75	78	95
35	24	204	113	109
3	154	104	235	25
15	253	225	159	78

WEIGHT

1	0	1
0	1	0
1	0	1

429

- padding：图像四周填充
  - 对输出尺寸产生影响

0	0	0	0	0	0	0	0
0	18	54	51	239	244	188	0
0	55	121	75	78	95	88	0
0	35	24	204	113	109	221	0
0	3	154	104	235	25	130	0
0	15	253	225	159	78	233	0
0	68	85	180	214	245	0	0
0	0	0	0	0	0	0	0

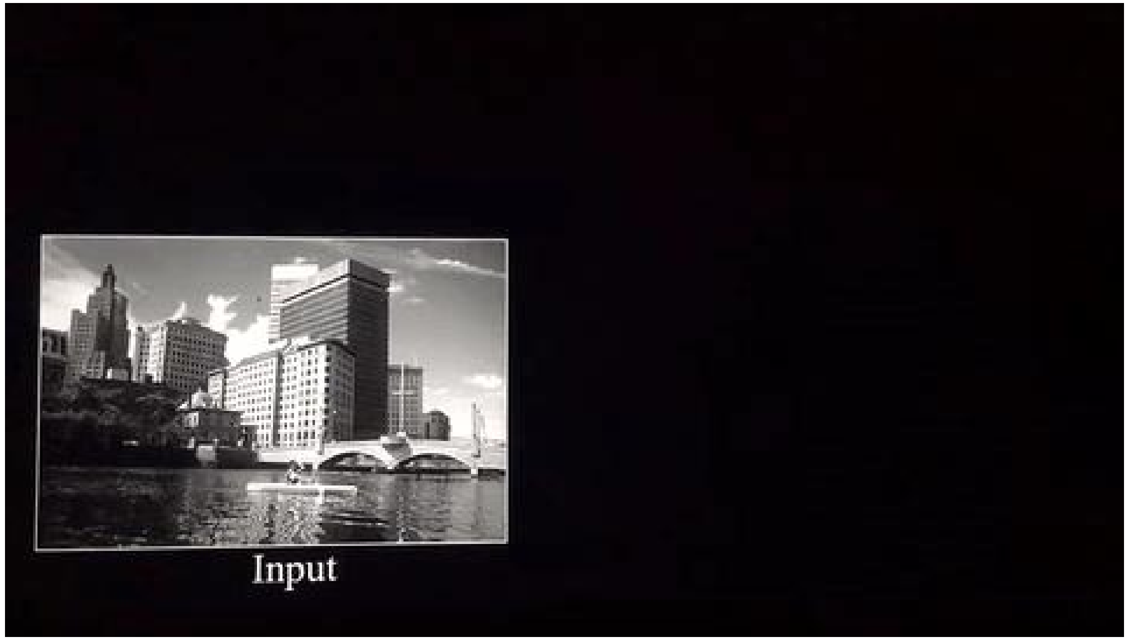
WEIGHT

1	0	1
0	1	0
1	0	1

139

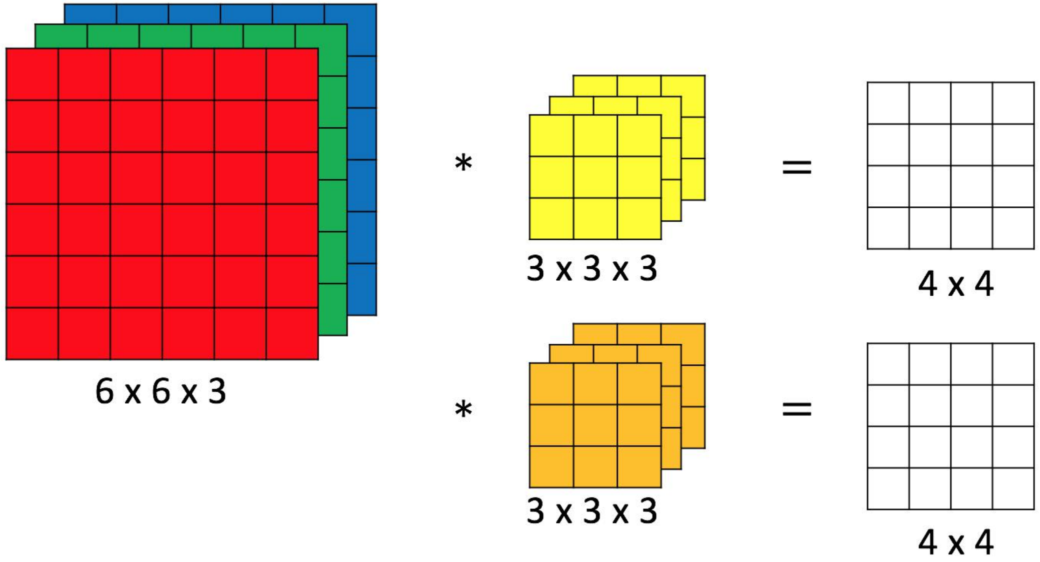
多卷积核

- 一个卷积层包含多个卷积核，实现特征的多样化



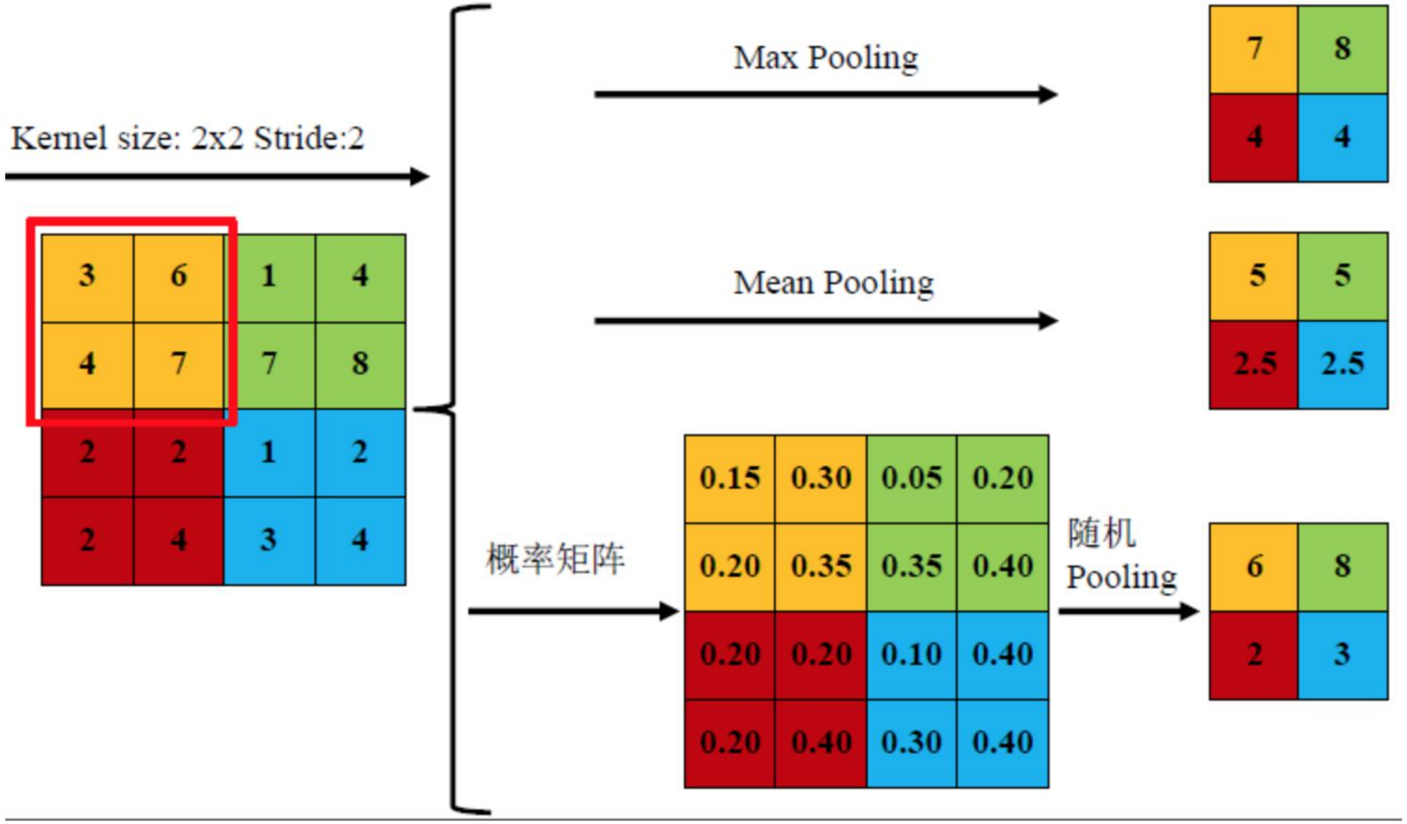
卷积层参数

- 假设卷积层输入为[w1, h1, c1]，输出为[w2, h2, c2]
  - 全连接的话需要 w1 x h1 x c1 x w2 x h2 x c2个参数
  - 卷积层需要c2个卷积核，每个卷积核尺寸[f1, f2, c1]，f1, f2为相对较小的数
- 示例：
  - [6, 6, 3] -> [4, 4, 2]
  - 一共只需2个3x3x3卷积



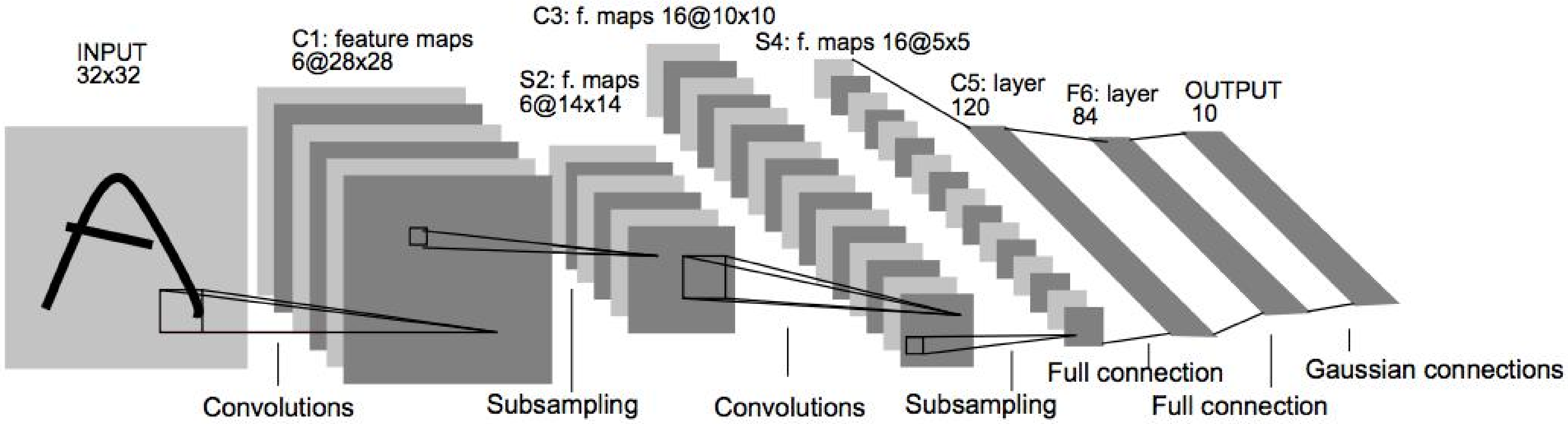
下采样（池化）

- 卷积后的图像特征仍然数量巨大，尤其是网络层数较多时，池化层进一步减少特征数量

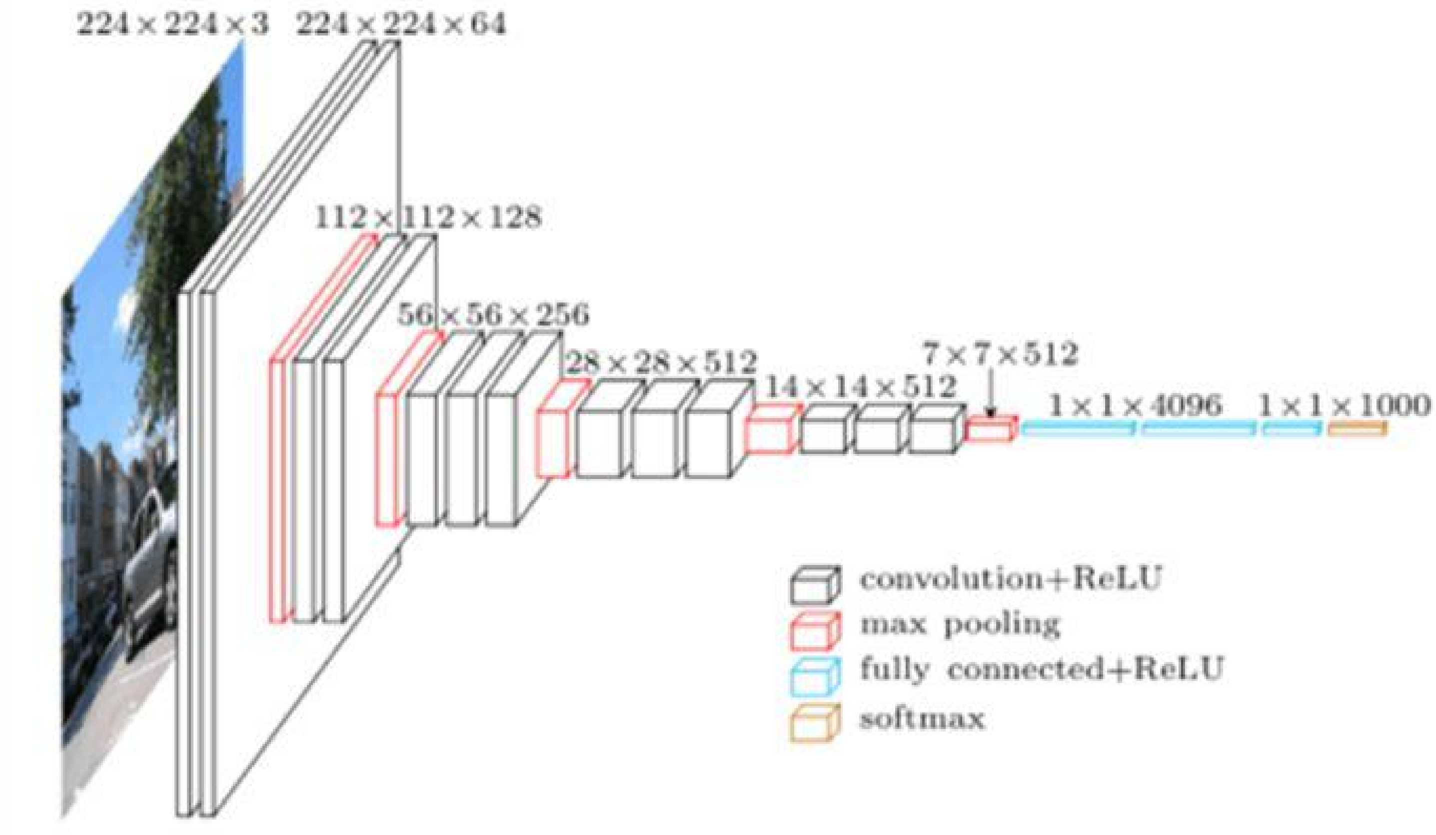


多卷积层实现深度网络

- LeNet5网络



- VGG-16网络结构


224×224×3
224×224×64
112×112×128
56×56×256
28×28×512
14×14×512
7×7×512
1×1×4096
1×1×1000
convolution+ReLU
max pooling
fully connected+ReLU
softmax

MNIST手写数字图像分类

- 输入图像 -> 10类
- 网络结构
  - 输入层 -> 卷积层1 -> 池化层1 -> 卷积层2 -> 池化层2 -> 全连接层 -> softmaxLoss
  - 28x28x1 -> 28x28x32 -> 14x14x32 -> 14x14x64 -> 7x7x64 -> 1024 -> 10

  - softmaxLoss = softmax + cross_entropy（交叉熵）

输入层和卷积层1

- x = tf.placeholder(tf.float32,[None, 784])
- x_image = tf.reshape(x, [-1, 28, 28, 1]) #最后一维代表通道数目，如果是rgb则为3

- W_conv1 = weight_variable([5, 5, 1, 32])
- b_conv1 = bias_variable([32])

- h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
- h_pool1 = max_pool_2x2(h_conv1)

卷积层2

- W_conv2 = weight_variable([5, 5, 32, 64])
- b_conv2 = bias_variable([64])

- h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)
- h_pool2 = max_pool_2x2(h_conv2)

全连接和Softmax

- W_fc1 = weight_variable([7 * 7 * 64, 1024])
- b_fc1 = bias_variable([1024])

- h_pool2_flat = tf.reshape(h_pool2, [-1, 7 * 7 * 64])
- h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)

- W_fc2 = weight_variable([1024, 10])
- b_fc2 = bias_variable([10])

- y_conv = tf.nn.softmax(tf.matmul(h_fc1, W_fc2) + b_fc2)

训练loss和准确率定义

- y_ = tf.placeholder("float", [None, 10])
- cross_entropy = -tf.reduce_sum(y_ * tf.log(y_conv)) #计算交叉熵
- train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)
- correct_prediction = tf.equal(tf.argmax(y_conv,1), tf.argmax(y_,1))
- accuracy = tf.reduce_mean(tf.cast(correct_prediction,"float"))

开启会话训练模型

- init = tf.global_variables_initializer();
- with tf.Session() as sess:
-     sess.run(init)
-     for i in range(5000):
-         batch = mnist.train.next_batch(50)
-         sess.run(train_step, feed_dict = {x:batch[0], y_:batch[1]})
-         if i % 100 == 0:
-             test_accuracy = accuracy.eval(session = sess, feed_dict = {x:mnist.test.images, y_:mnist.test.labels})
-             print("step %d, test accuracy %g" %(i, test_accuracy))

作业

- 完成一个简单的图像分类任务，具体内容自拟