浅谈人工智能和神经网络

Water About Artifitial Idiot Intelligence and Neural Network

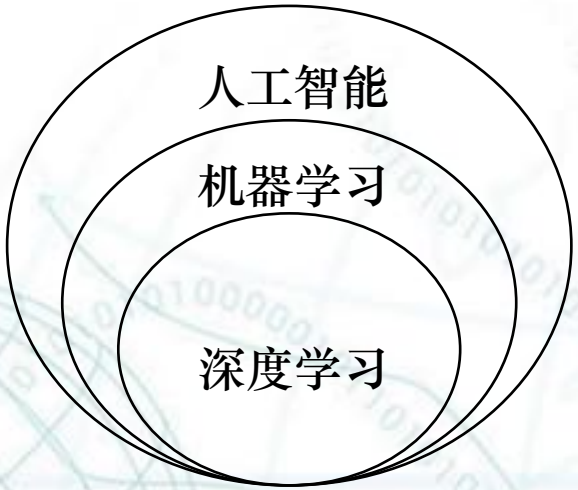
陈旭

AI？

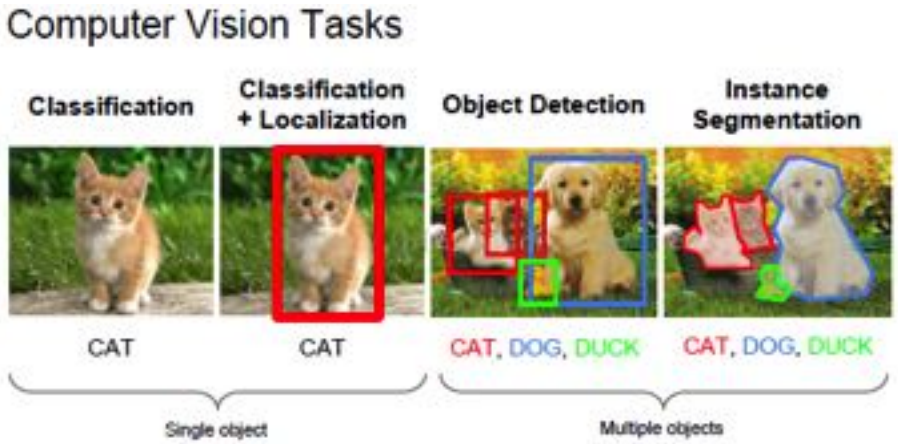
人工智能：用计算机来构造拥有与人类智慧同样本质特性的机器

机器学习：一种实现人工智能的方法

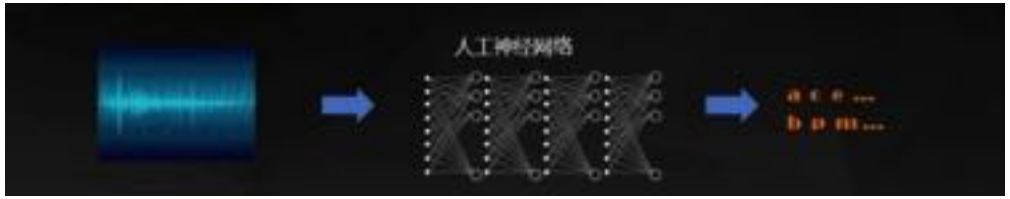
深度学习：一种实现机器学习的技术



深度学习应用场景



计算机视觉



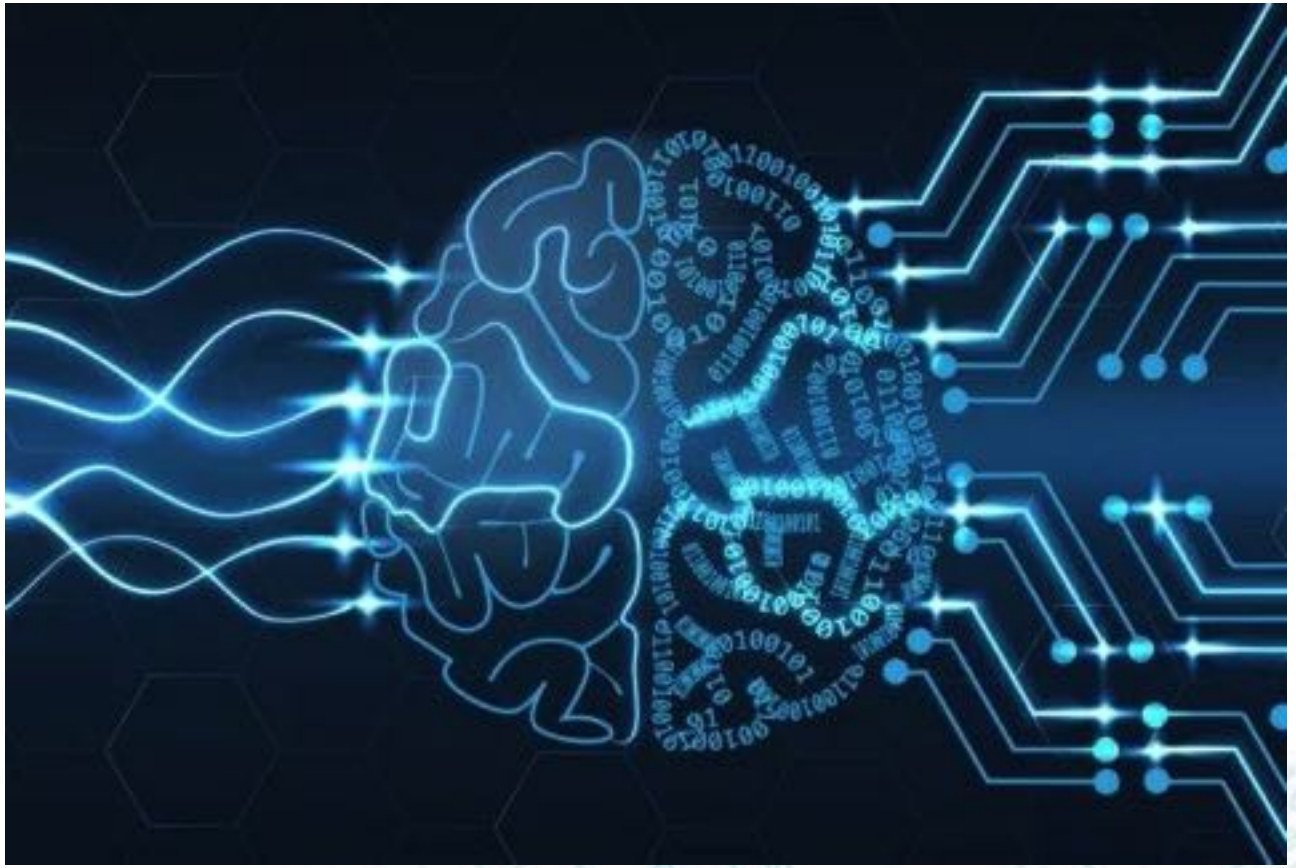
语音识别

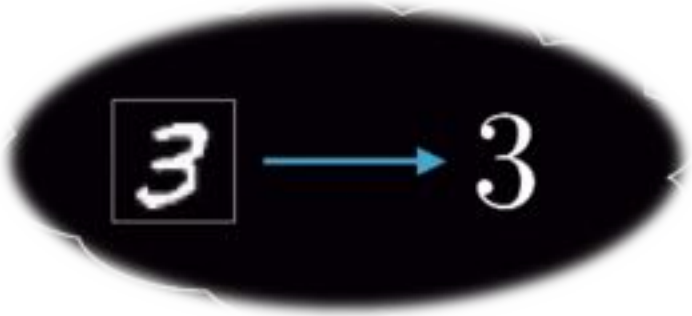
自然语言处理

人机博弈

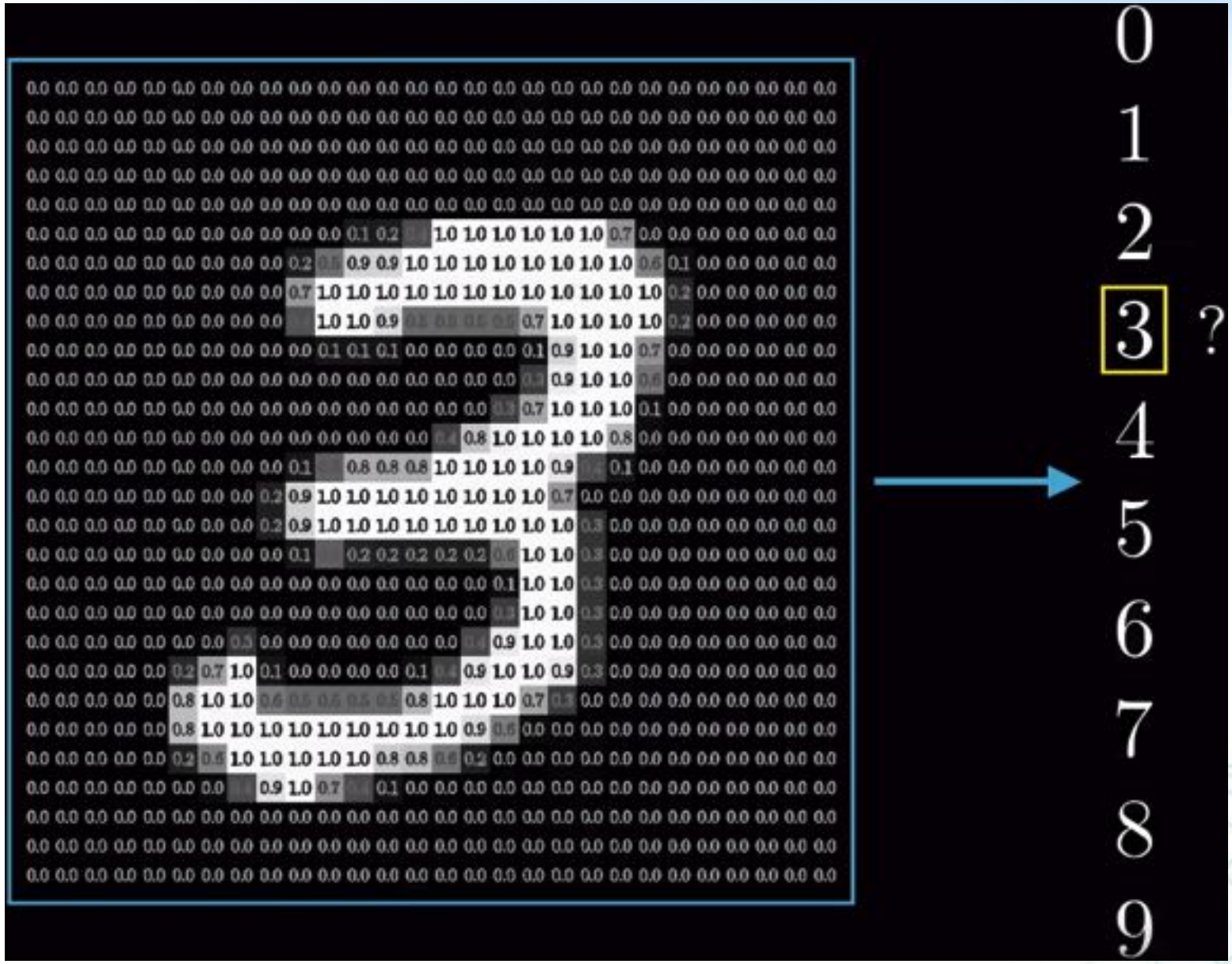
深度学习：深层的神经网络

- 神经元
- 网络结构
- 激活函数
- 损失函数
- 反向传播





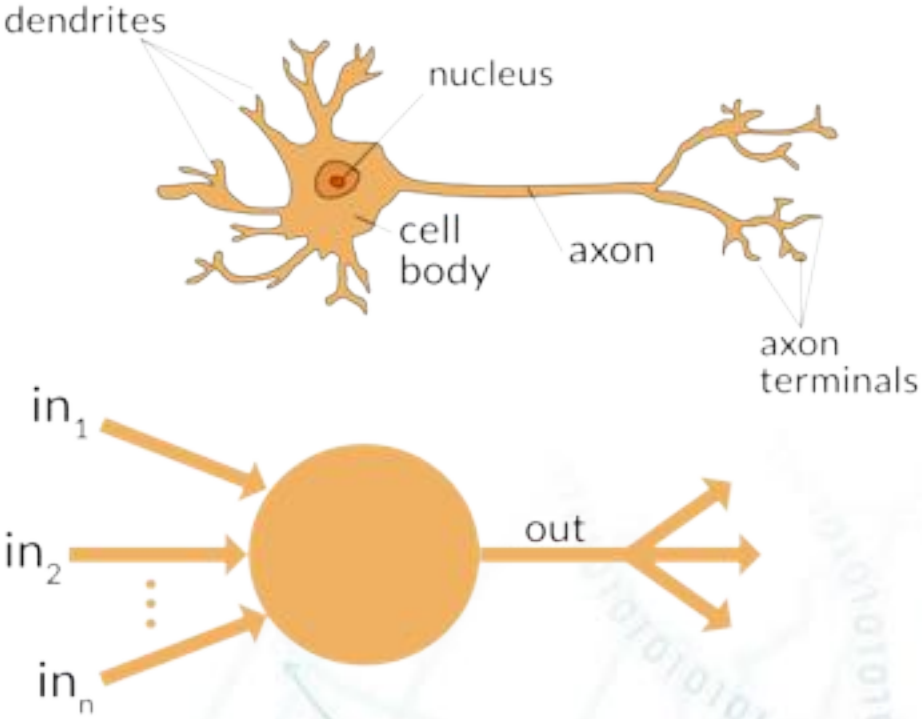
EZ for brain



如何写一个程序
让电脑来识别数字

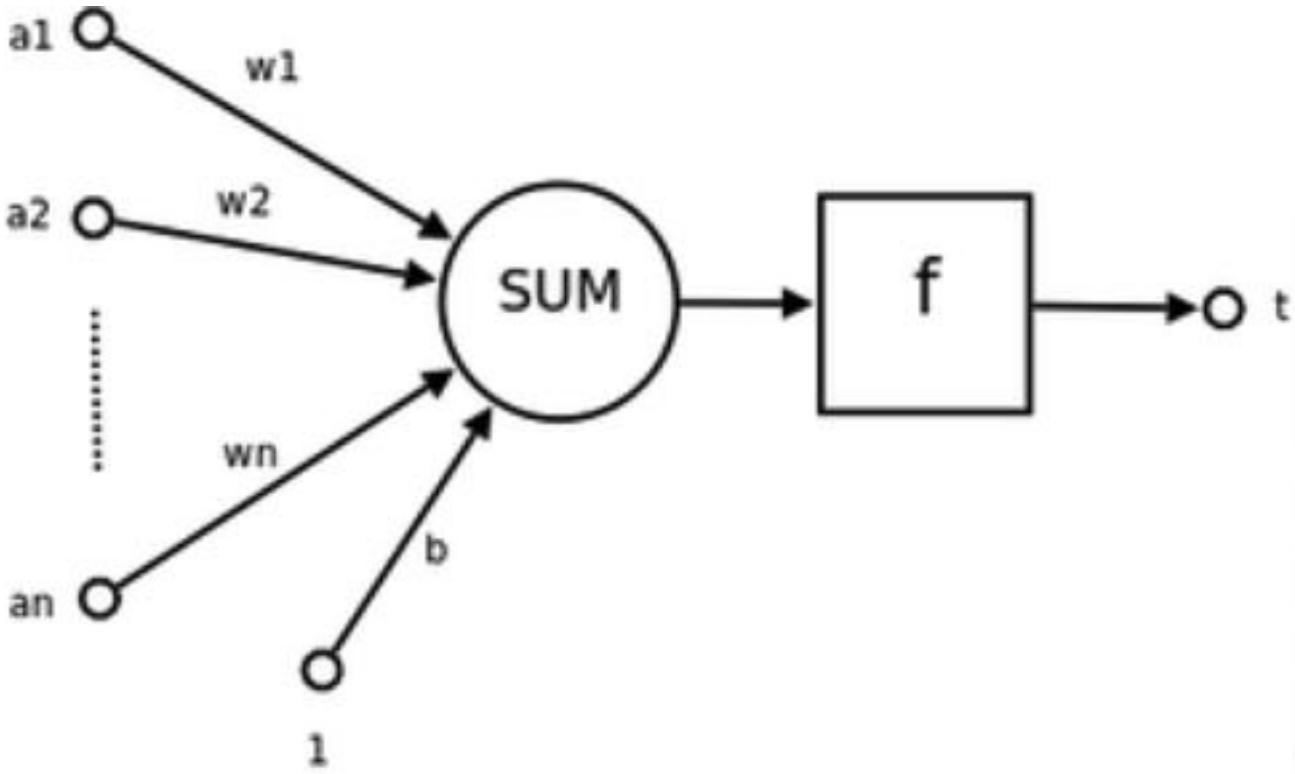
神经元

- 是生物神经细胞的简单抽象。
- 神经细胞结构大致可分为：树突、突触、细胞体及轴突。
- 单个神经细胞可被视为一种只有两种状态的机器——激动时为‘是’，而未激动时为‘否’。



神经元

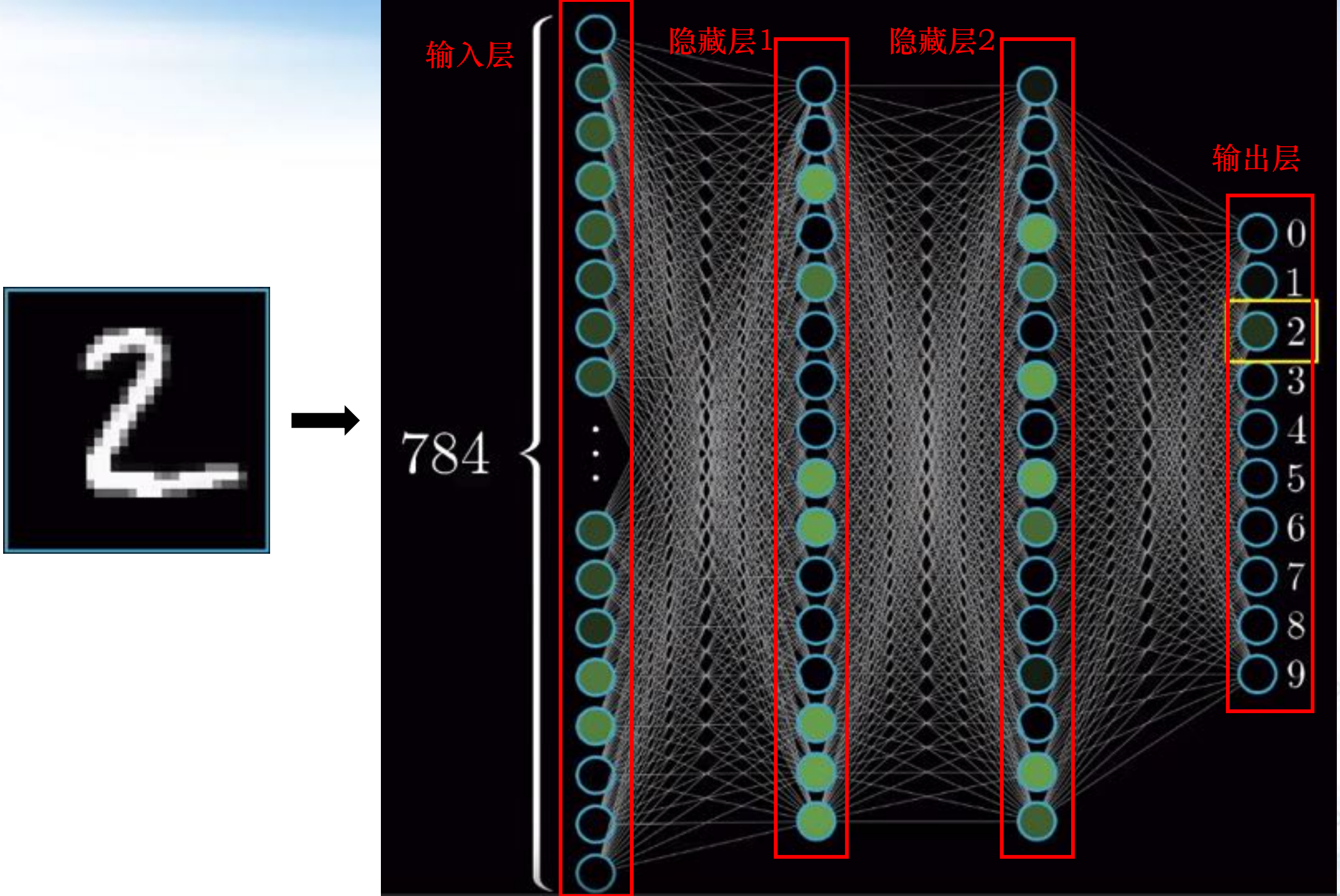
- 设$\vec{a} = (a_1, a_2, \ldots, a_n)$为这个输入的n维分量，$\vec{w} = (w_1, w_2, \ldots, w_n)$为权重，b为偏置量，$f(x)$为激活函数，即

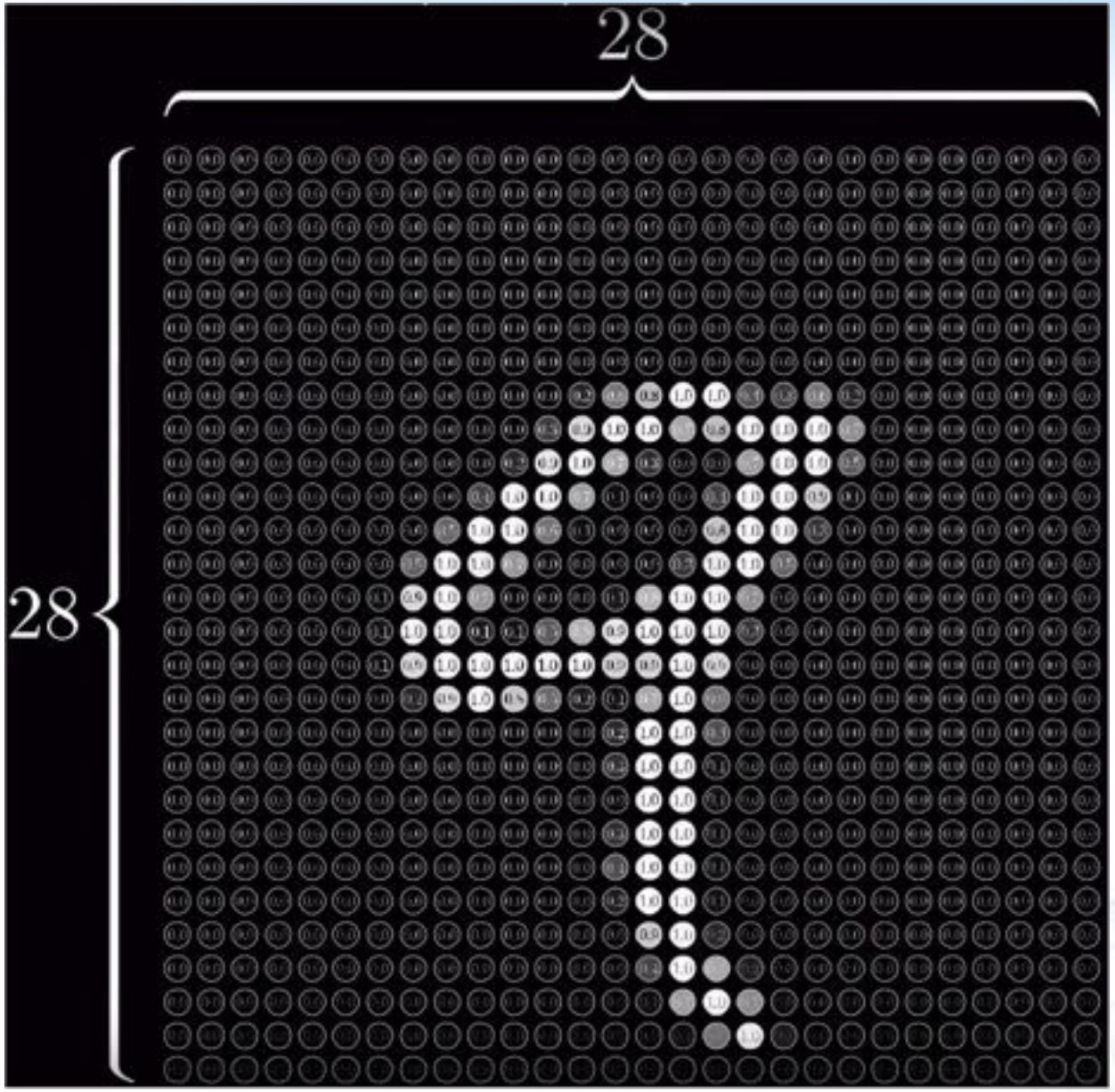


- $f(x) = \begin{cases} 1 \; if \; x > 0 \\ 0 \; if \; x \leq 0 \end{cases}$
- 那么预测值就是$\hat{y} = f(\vec{a} \cdot \vec{w} + b)$

网络结构

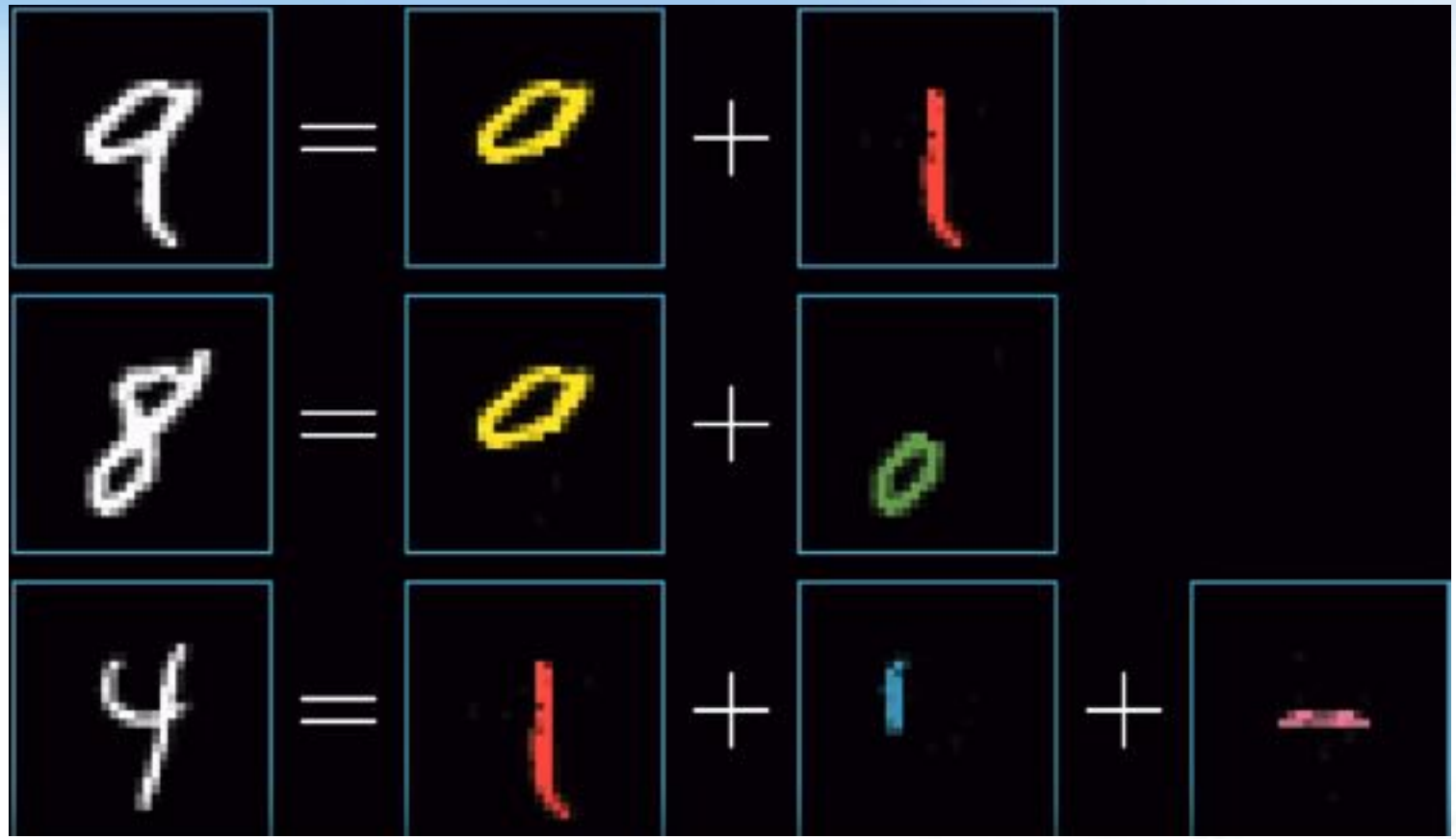
- 卷积神经网络（CNN）
  - 全连接层、卷积层、池化层
- 循环神经网络（RNN）
  - 专门处理序列化数据
  - 长短期记忆网络（LSTM）
- CNN+RNN
  - 根据图像语义自动生成摘要


输入层
隐藏层1
隐藏层2
输出层
784
0
1
2
3
4
5
6
7
8
9

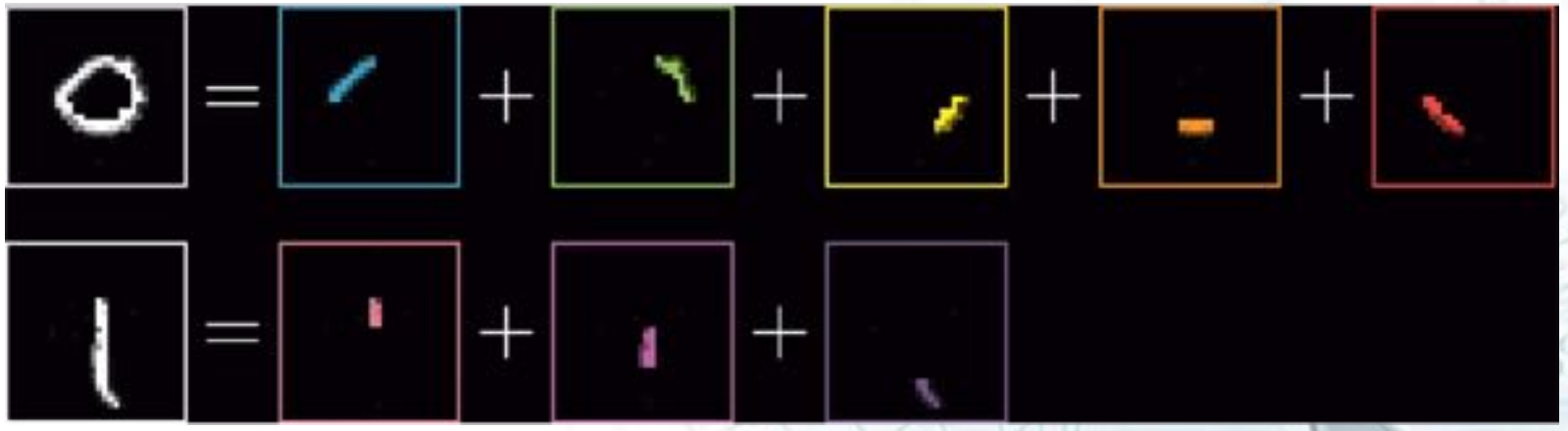


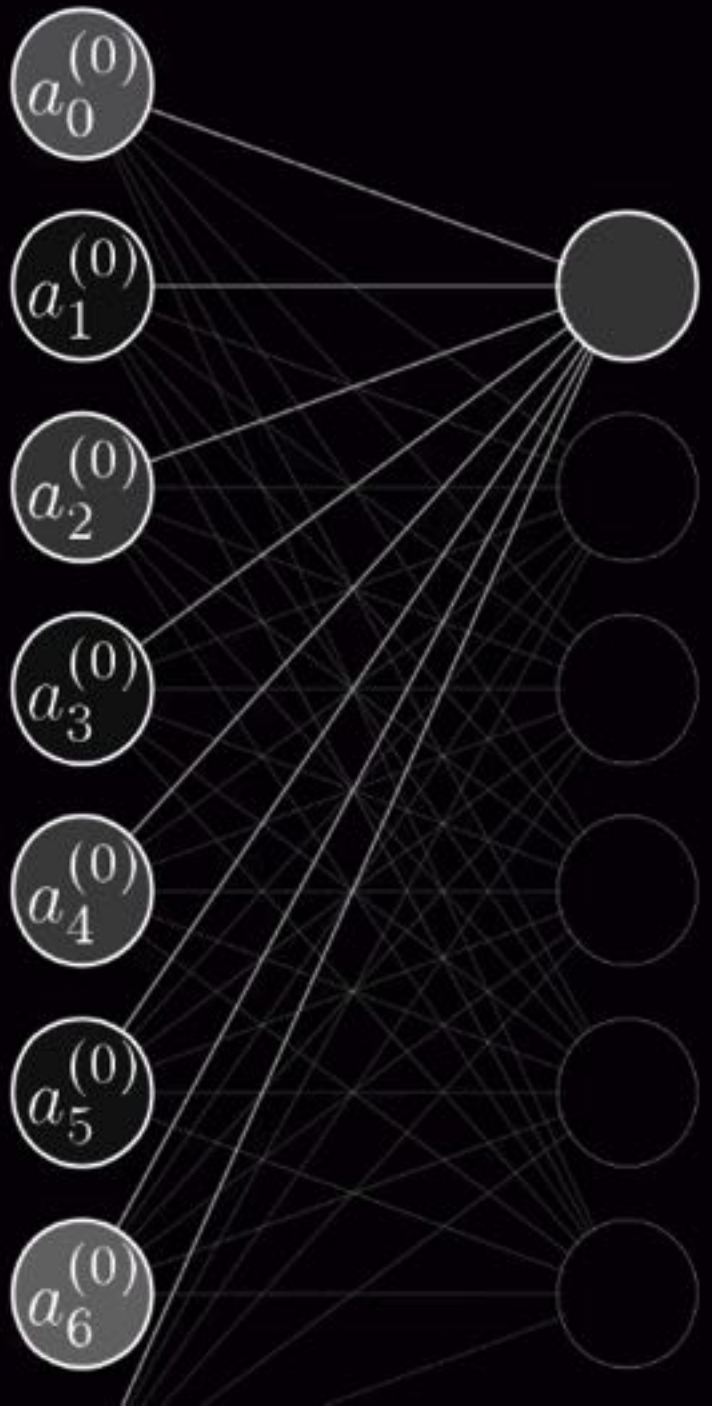
输入层：每个神经元都是一个像素点。根据像素点的灰度值决定每个神经元的activation

隐藏层2：大的结构特征



隐藏层1：小的结构特征





Sigmoid

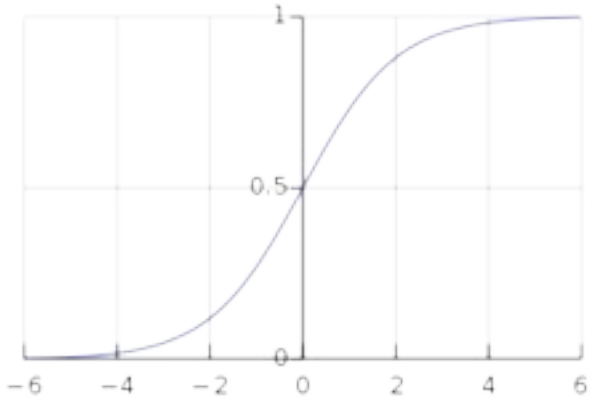
$$a_0^{(1)} = \sigma\left(w_{0,0}\ a_0^{(0)} + w_{0,1}\ a_1^{(0)} + \cdots + w_{0,n}\ a_n^{(0)} + b_0 \right)$$

Bias

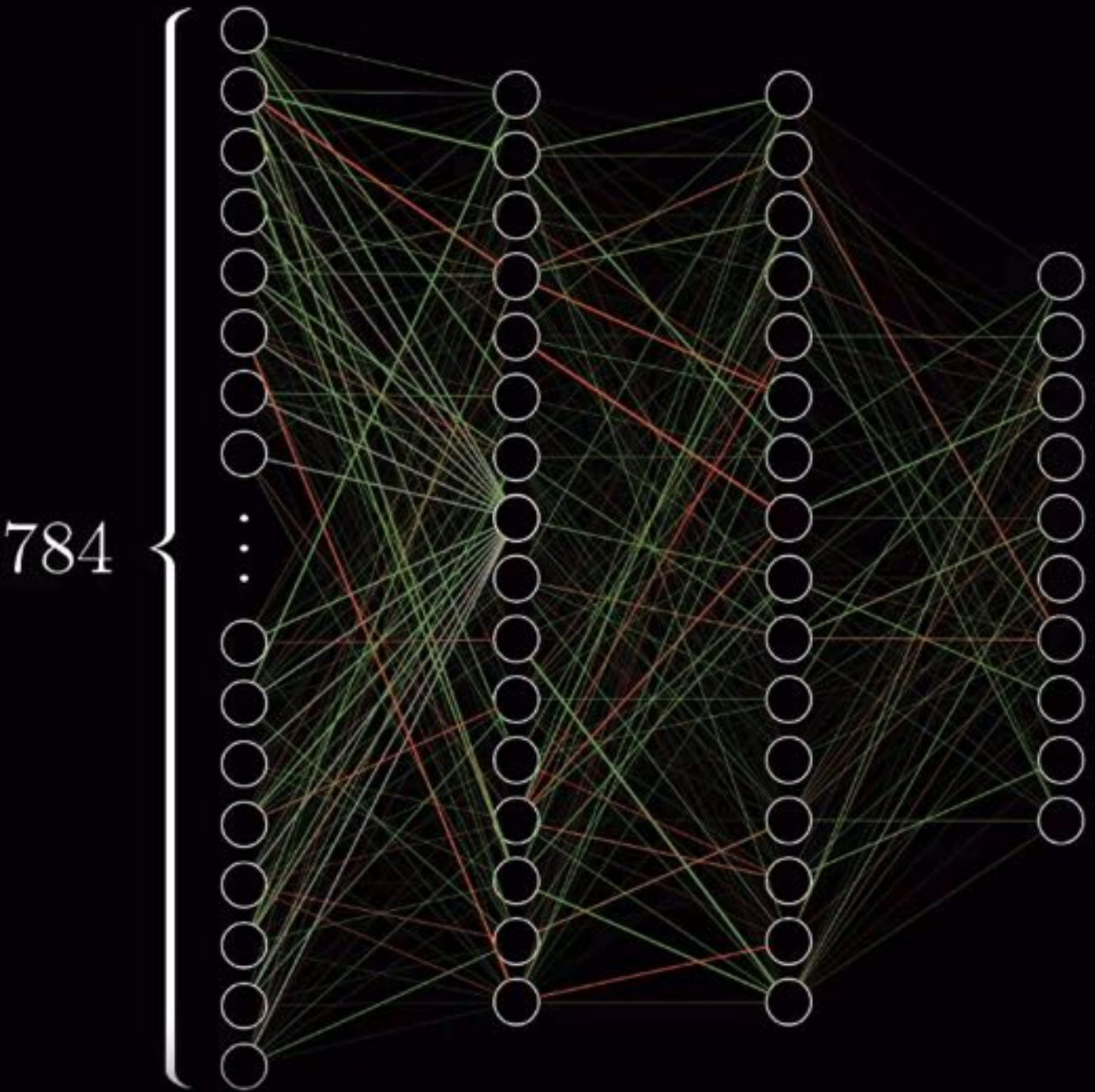
$$\begin{bmatrix} w_{0,0} & w_{0,1} & \cdots & w_{0,n} \\ w_{1,0} & w_{1,1} & \cdots & w_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,0} & w_{k,1} & \cdots & w_{k,n} \end{bmatrix} \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ \vdots \\ a_n^{(0)} \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_n \end{bmatrix}$$

激活函数

- Sigmoid function
- $\sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}$



- 线性整流函数（Rectified Linear Unit, ReLU）
- $f(x) = \max(0, x)$



$$784 \times 16 + 16 \times 16 + 16 \times 10$$

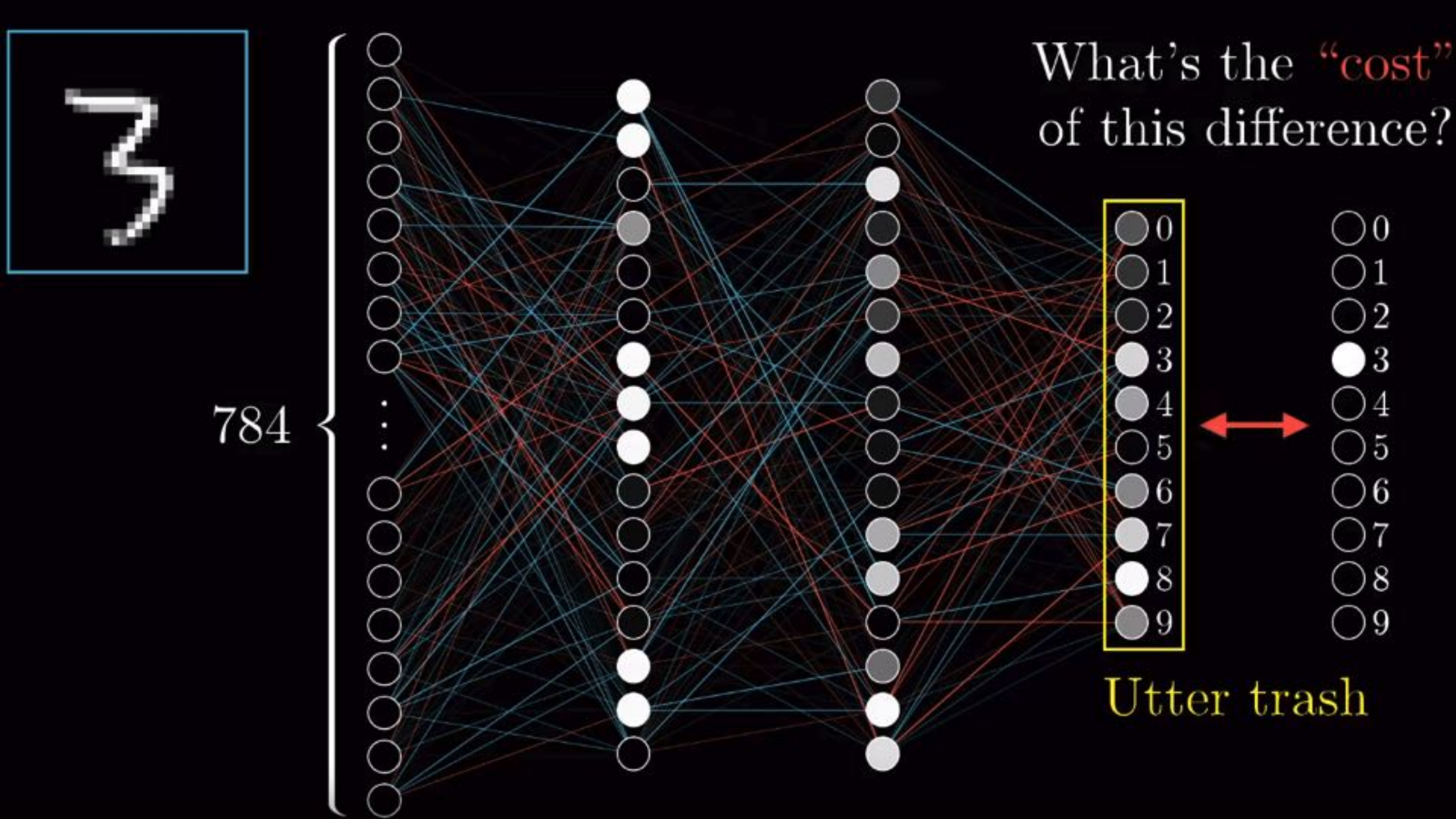
weights

$$16 + 16 + 10$$

biases

$$13{,}002$$

Learning $\rightarrow$ Finding the right weights and biases

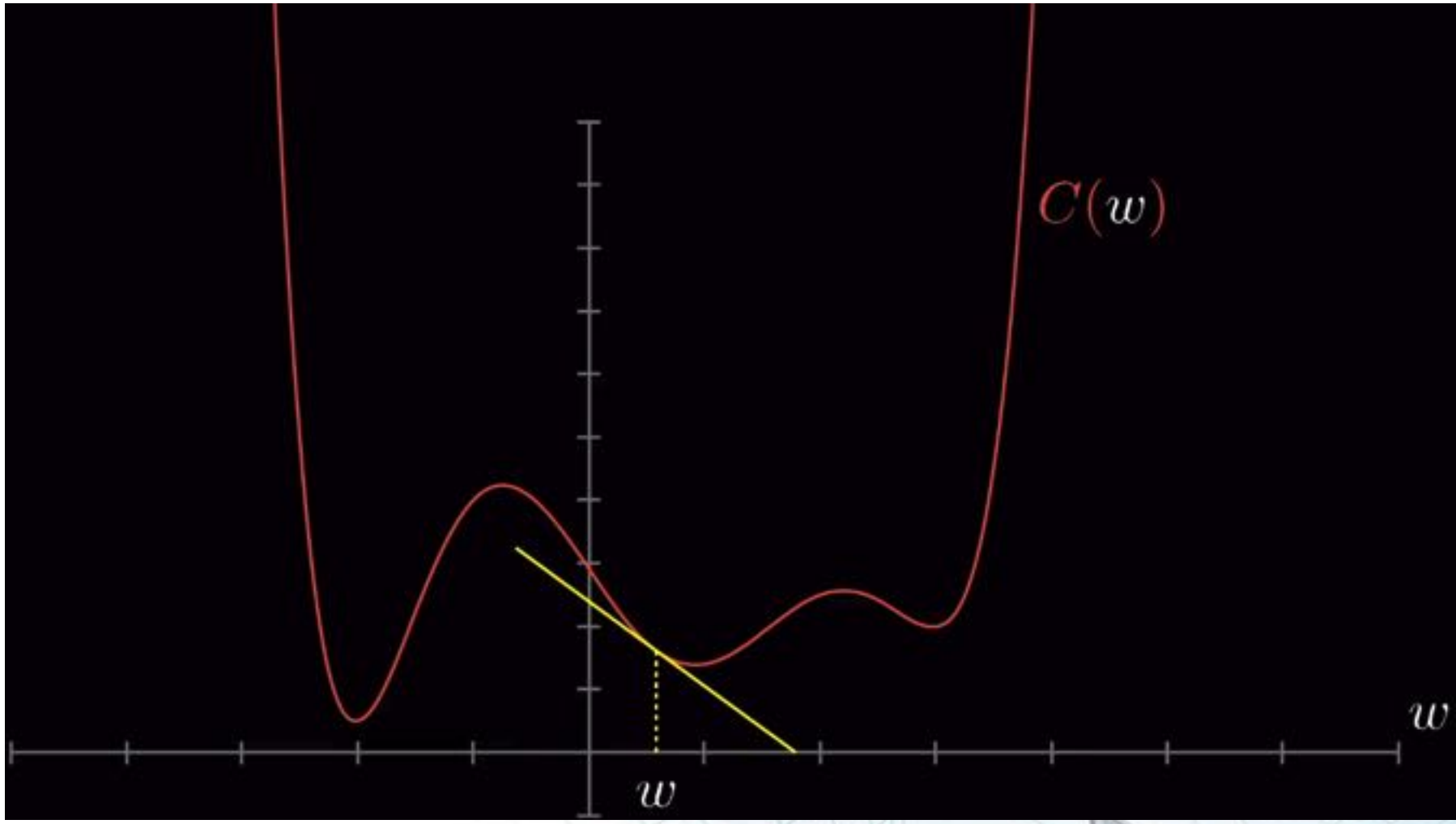

What's the "cost"
of this difference?
784
0
1
2
3
4
5
6
7
8
9
0
1
2
3
4
5
6
7
8
9
Utter trash

损失函数

- 交叉熵（cross entropy）
  - $\mathrm{CE}(Y, f(x)) = -\sum_{x \in X} Y \log f(x)$
- 平方损失函数
  - $L(Y, f(x)) = \sum_{x \in X} (Y - f(x))^2$
- 指数损失函数
  - $L(Y, f(x)) = \sum_{x \in X} e^{-Yf(x)}$

优化算法

- 梯度下降算法
  - 计算梯度
    - $f'(x)$
  - 变量x向梯度反方向移动
    - $x = x - r * f'(x)$
  - 循环直至满足条件
- Adam，RMSProp算法



学习类型

- 监督型
  - Train:
    - data, label
  - Test:
    - data, result
- 非监督型
  - Environment:
    - agent, state, action, reward, policy

深度学习流行框架：tensorflow

基于数据流图的数值计算开源软件库。数据流图中的点表示数学计算操作，边表示操作与操作之间的高维度数组数据，称为tensor。

深度学习的hello world：MNIST手写数字识别

Tensorflow:MNIST识别手写数字

```
from tensorflow.examples.tutorials.mnist import input_data

mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)  # 读取MNIST手写数字识别数据集
import tensorflow as tf

X = tf.placeholder(tf.float32, [None, 784], name='x-input')  # tensorflow变量占位符，在执行运算时才传入数据
Y = tf.placeholder(tf.float32, [None, 10], name='y-input')


def init_weights(shape):  # 使用服从正态分布的随机值来初始化参数矩阵
    return tf.Variable(tf.random_normal(shape, stddev=0.01))


w_h = init_weights([784, 16])  # 从输入层到hidden layer1，28x28=784维转化为16维
w_h2 = init_weights([16, 16])  # 从hidden layer1到hidden layer2，16维转化为16维
w_o = init_weights([16, 10])  # 从hidden layer2到输出层，16维转化为10维
```

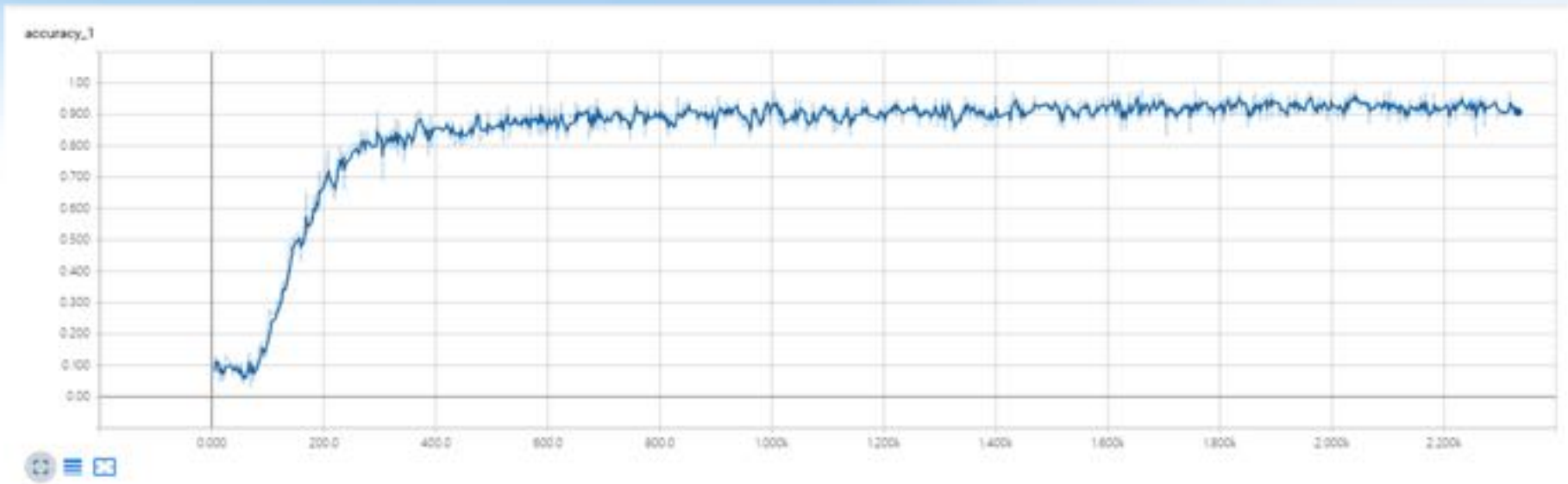
Tensorflow:MNIST识别手写数字

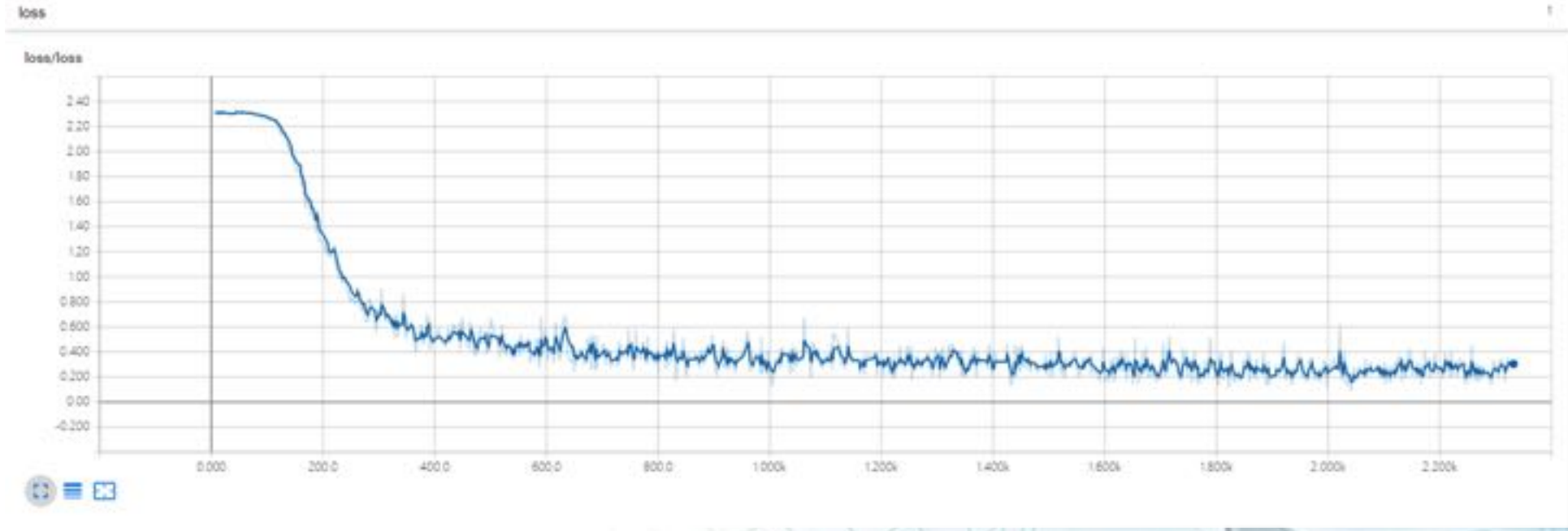
```
def model(X, w_h, w_h2, w_o):  # 构建神经网络训练模型
    h = tf.nn.relu(tf.matmul(X, w_h))  # hidden layer1是由输入层和权重矩阵1进行矩阵乘法后经过激活函数得到
    h2 = tf.nn.relu(tf.matmul(h, w_h2))  # hidden layer2是由hidden layer1和权重矩阵2进行矩阵乘法后经过激活函数得到
    return tf.matmul(h2, w_o)  # 返回hidden layer2和权重矩阵3的矩阵乘法，即输出层


py_x = model(X, w_h, w_h2, w_o)  # 初始化一个神经网络训练模型
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=py_x, labels=Y))  # 设置改模型的代价函数
train_op = tf.train.RMSPropOptimizer(0.001, 0.9).minimize(cost)  # 设置优化算法，这里是改进后的梯度下降算法
predict_acc = tf.reduce_mean(tf.cast(tf.equal(tf.argmax(py_x, 1), tf.argmax(Y, 1)), tf.float32))  # 计算预测的准确率
epoch_count = 2333  # 需要计算的次数
batch_size = 64  # 每一批的图片数量
```

Tensorflow:MNIST识别手写数字

```
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())  # 初始化各种变量
    step = 0
    for i in range(epoch_count):
        step += 1
        batch_x, batch_y = mnist.train.next_batch(batch_size)  # 读取相应一批的图片和标签数据
        sess.run(train_op, feed_dict={X: batch_x, Y: batch_y})  # 通过神经网络训练
        if step % 128 == 0:  # 输出训练记录
            loss, acc = sess.run([cost, predict_acc],
                                 feed_dict={X: batch_x, Y: batch_y})
            print("Epoch: {}".format(step), "\tLoss: {:.6f}".format(loss), "\tTraining Accuracy: {:.5f}".format(acc))
    print("Testing Accuracy: {:0.5f}".format(sess.run(predict_acc,
                                                      feed_dict={X: mnist.test.images, Y: mnist.test.labels})))
```

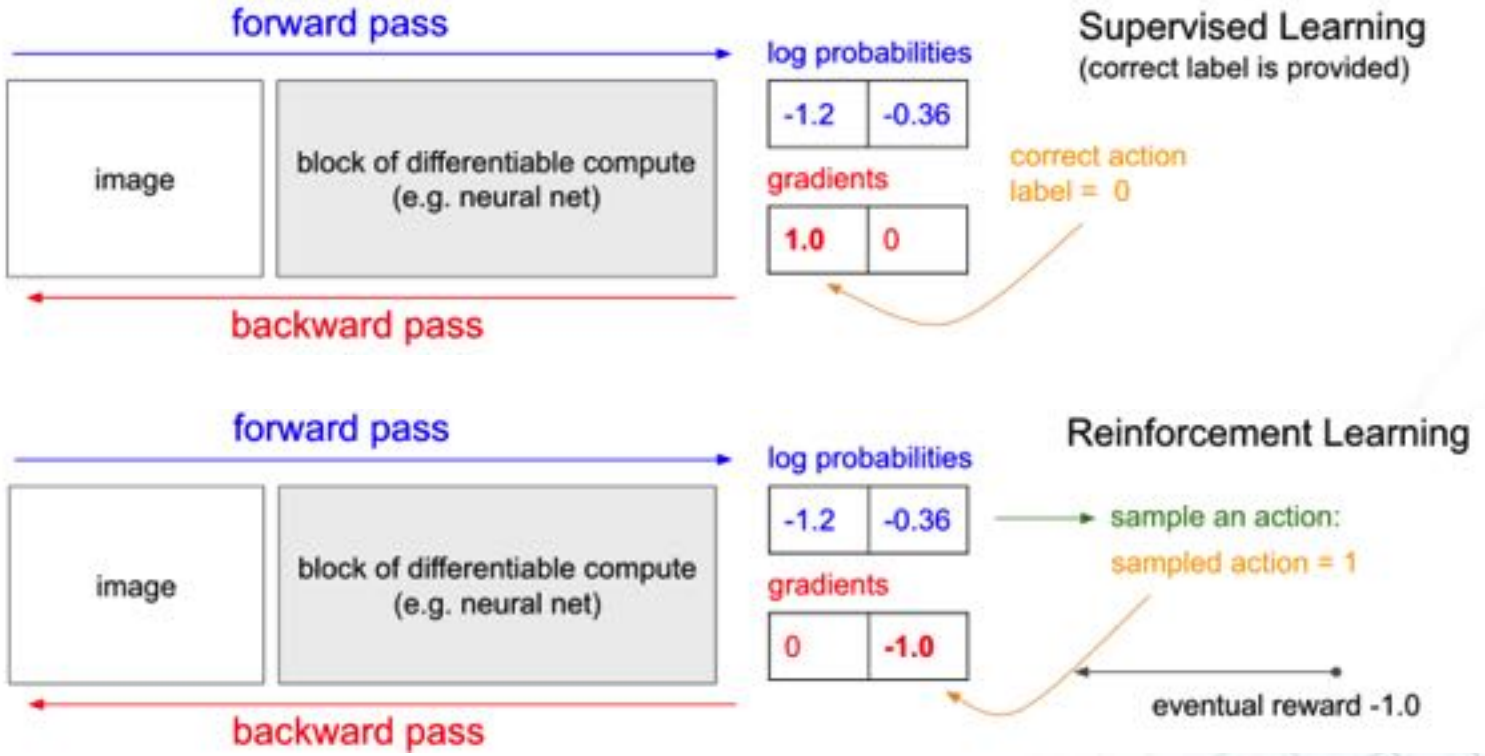
```
Accuracy at step 0: 0.0819
Accuracy at step 128: 0.3761
Accuracy at step 256: 0.7901
Accuracy at step 384: 0.8468
Accuracy at step 512: 0.8702
Accuracy at step 640: 0.886
Accuracy at step 768: 0.8964
Accuracy at step 896: 0.9032
Accuracy at step 1024: 0.9078
Accuracy at step 1152: 0.9036
Accuracy at step 1280: 0.9111
Accuracy at step 1408: 0.9116
Accuracy at step 1536: 0.9141
Accuracy at step 1664: 0.9194
Accuracy at step 1792: 0.9227
Accuracy at step 1920: 0.9265
Accuracy at step 2048: 0.9275
Accuracy at step 2176: 0.9284
Accuracy at step 2304: 0.9313
```





非监督型：强化学习

- Trial and error



OpenAI Gym

Gym is a toolkit for developing and comparing reinforcement learning algorithms.

Pong-v0

Maximize your score in the Atari 2600 game Pong. In this environment, the observation is an RGB image of the screen, which is an array of shape (210, 160, 3) Each action is repeatedly performed for a duration of k frames, where k is uniformly sampled from $\{2, 3, 4\}$.

The game is simulated through the Arcade Learning Environment [ALE], which uses the Stella [Stella] Atari emulator.

[ALE] *MG Bellemare, Y Naddaf, J Veness, and M Bowling. "The arcade learning environment: An evaluation platform for general agents." Journal of Artificial Intelligence Research (2012).*

[Stella] *Stella: A Multi-Platform Atari 2600 VCS emulator http://stella.sourceforge.net/*

VIEW SOURCE ON GITHUB

RandomAgent on Pong-v0

纯随机打球&策略梯度算法打球

谢谢

Thanks for watching