

Python人工智能

讲师：覃秉丰



深度学习三次热潮

第一次热潮（20世纪50年代-60年代）

- 1950年图灵提出了图灵测试。
- 1958年计算机科学家Rosenblatt提出了一种具有三层网络特性的神经网络结构，成为“感知器”。
- 1969年，人工智能的先驱Minsky出版了一本名为《感知器》的书，书中指出简单的神经网络只能运用于线性问题的求解，能够求解非线性问题的网络应具有隐层，而从理论上还不能证明将感知器扩展到多层网络是有意义的。

第二次热潮（20世纪80年代-90年代）

- 语音识别是当时最具代表性的突破成果之一，语音识别领域最具代表性的人物就是李开复了。
- 1986年，Rumelhart，Hinton，Williams发展了BP算法。（多层感知器的误差反向传播算法）。
- 1987年6月，首届国际神经网络学术会议在美国加州圣地亚哥召开，到会代表有1600余人。之后国际神经网络学会和国际电气工程师与电子工程师学会（IEEE）联合召开每年一次的国际学术会议。

第三次热潮（2006年至今）

- 2006年Hinton在《Science》杂志上的发表了一篇深度学习的论文。
- 2009年李飞飞创立ImageNet。
- 2016年AlphaGo战胜人类顶级围棋选手。



深度学习爆发三要素

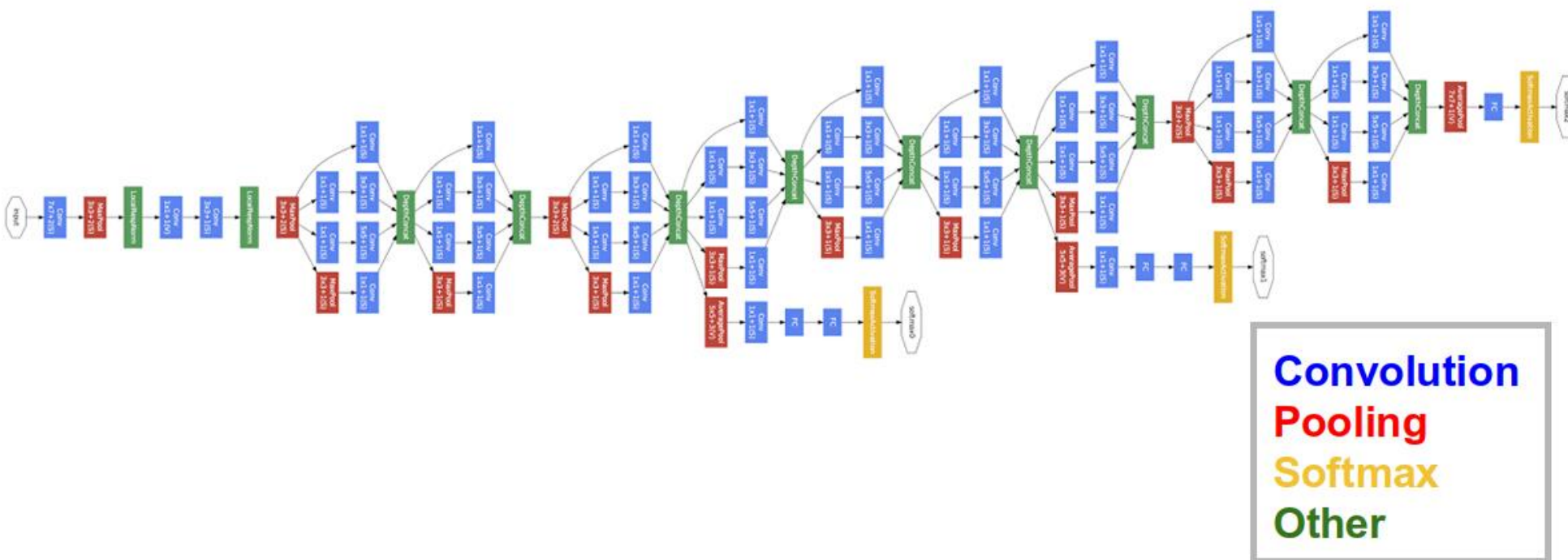
ImageNet Dataset

IM  GENET



Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., ... & Fei-Fei, L. (2015). [Imagenet large scale visual recognition challenge](#). *arXiv preprint arXiv:1409.0575*. [\[web\]](#)







深度学习三巨头

- 英国出生的计算机学家和心理学家，以其在神经网络方面的贡献闻名。Hinton是反向传播算法的发明人之一，也是深度学习的积极推动者。目前担任多伦多大学计算机科学系教授。
- 2013年3月加入Google，领导Google Brain项目。



- 计算机科学家，他最著名的工作是光学字符识别和计算机视觉上使用卷积神经网络（CNN），他也被称为卷积网络之父。
- 多伦多大学跟随Hinton做博士后。1988年，加入贝尔实验室，之后研发出了卷积神经网络，曾广泛用于手写数字识别。
- 2003年去了纽约大学任教。2013年12月加入了Facebook，成为Facebook人工智能实验室的第一任主任。



- 在MIT和贝尔实验室做过博士后研究员，自1993年之后就在蒙特利尔大学任教。在预训练和自动编码器等方面作出过重大贡献。
- “我留在学术圈为全人类作贡献，而不是为某一个公司赚钱”



Andrew Wu (吴恩达)

- 曾经是斯坦福大学计算机科学系和电气工程系的副教授，斯坦福人工智能实验室主任。他创建了在线教育平台Coursera。



- 2011年，吴恩达在Google创建了Google Brain项目，通过分布式集群计算机开发超大规模的人工神经网络。
- 2014年5月，吴恩达加入百度，负责百度大脑计划，并担任百度公司首席科学家。
- 2017年3月，吴恩达从百度离职。

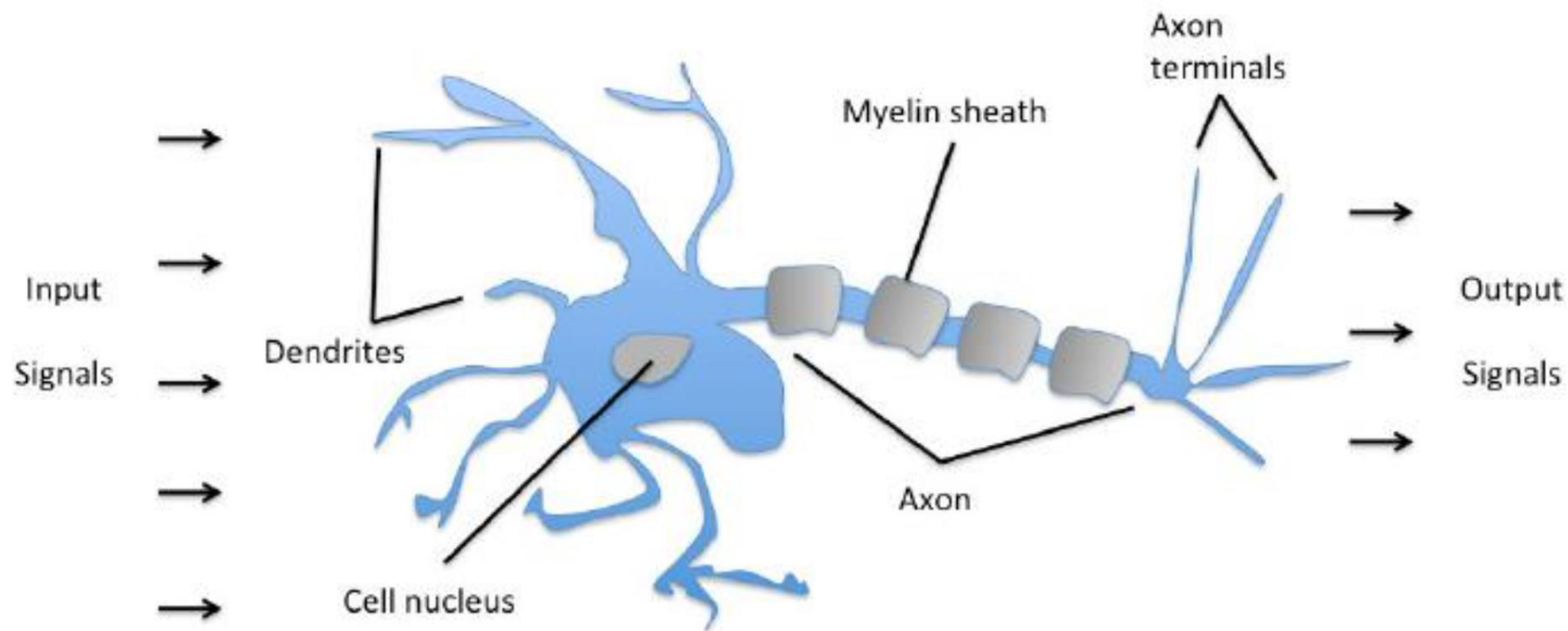
关系图



LeCun Hinton Bengio Andrew Ng



神经网络



Schematic of a biological neuron.

单层感知器

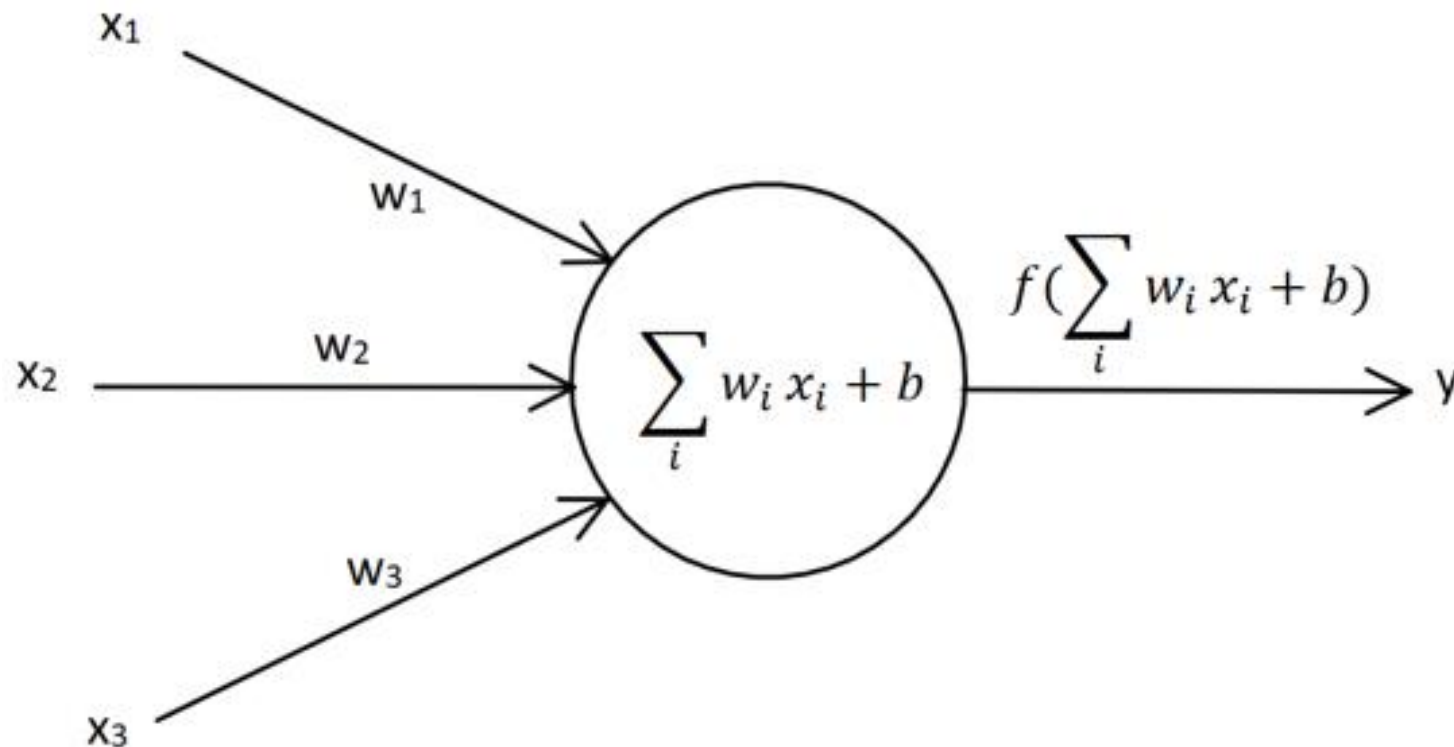
输入节点： x_1, x_2, x_3

输出节点： y

权向量： w_1, w_2, w_3

偏置因子： b

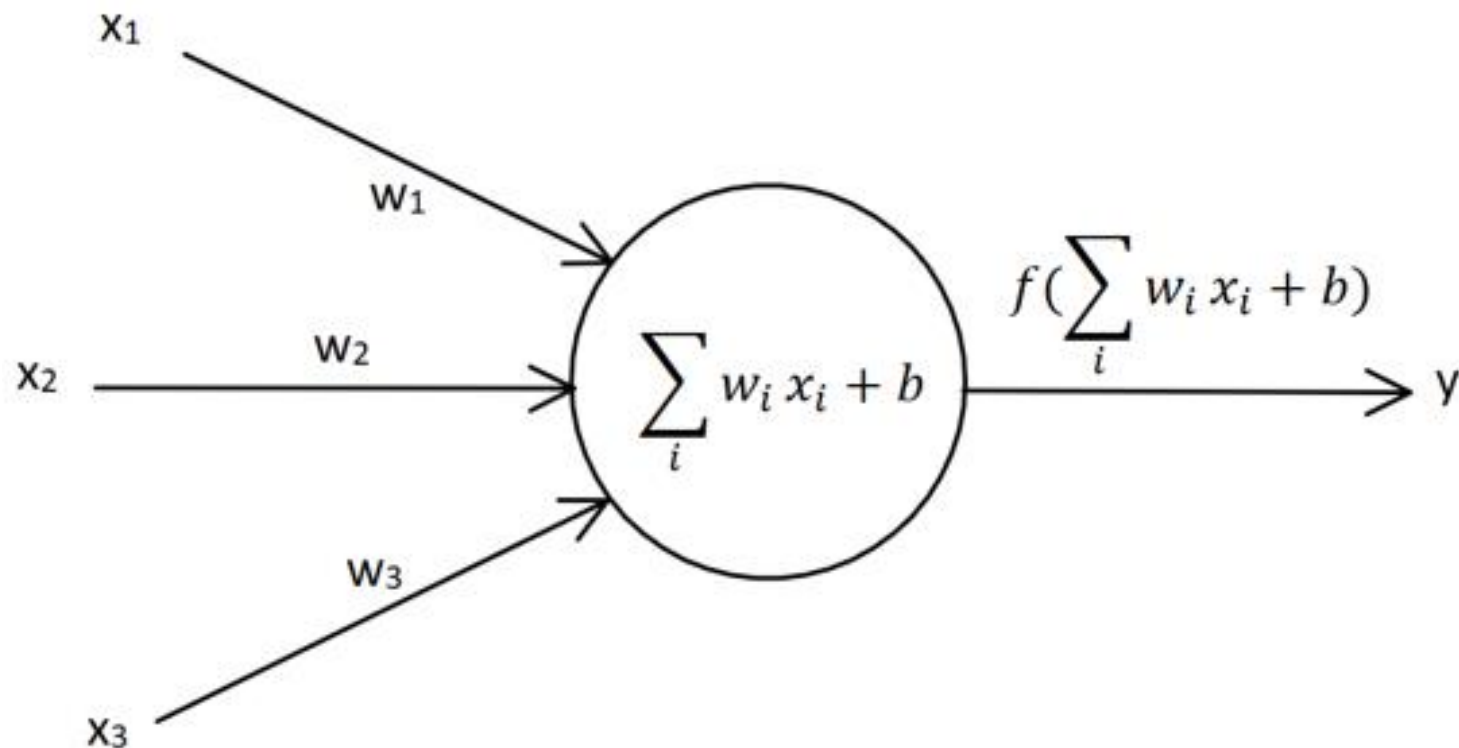
激活函数： $\text{sign}(x) = \begin{matrix} 1 & X \geq 0 \\ -1 & X < 0 \end{matrix}$



单层感知器举例

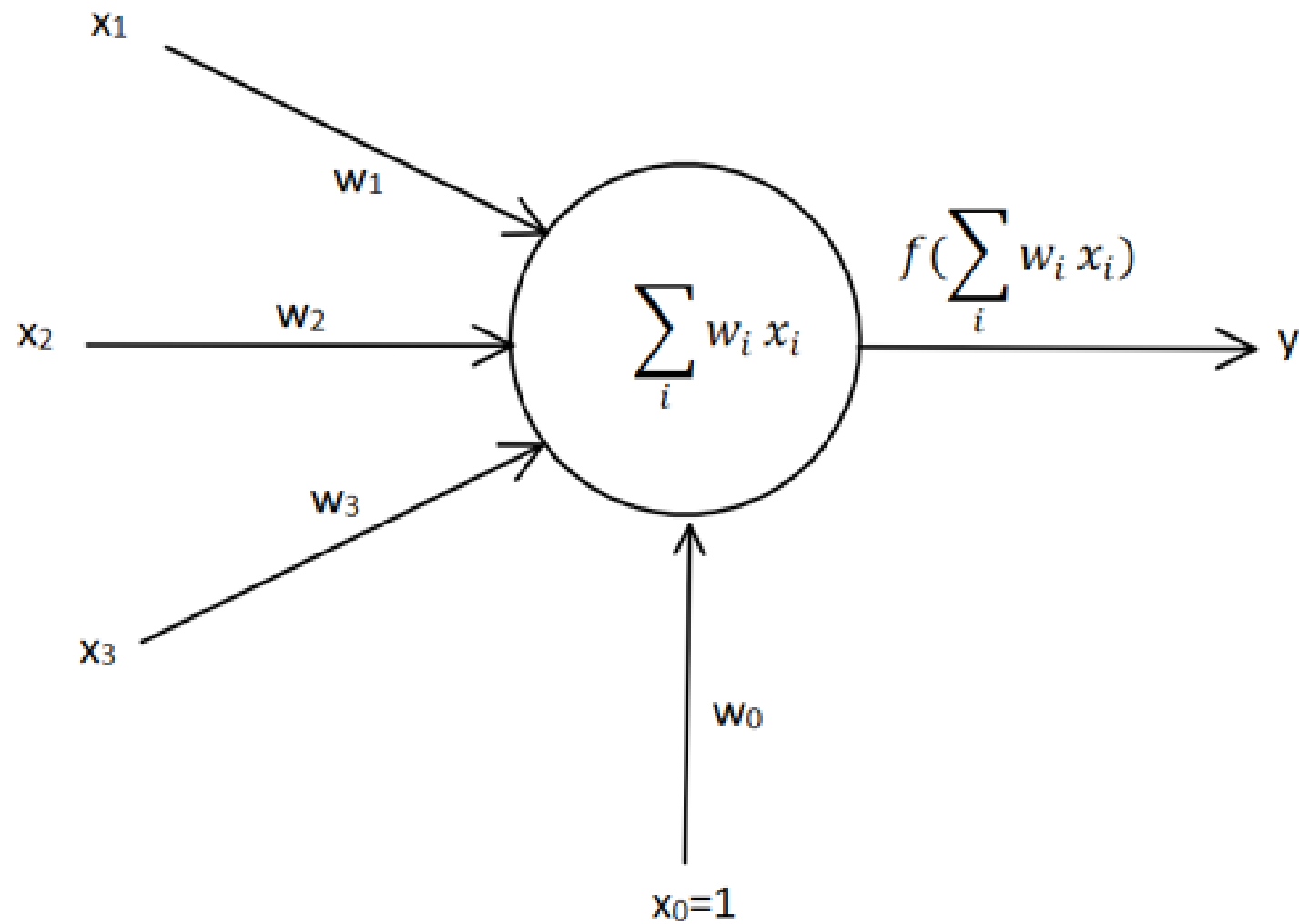
$b = -0.6$

x_1	x_2	x_3	Y
0	0	0	-1
0	0	1	-1
0	1	0	-1
0	1	1	1
1	0	0	-1
1	0	1	1
1	1	0	1
1	1	1	1



$$y = \begin{cases} 1 & (0.5x_1 + 0.5x_2 + 0.5x_3 - 0.6 \geq 0) \\ -1 & (0.5x_1 + 0.5x_2 + 0.5x_3 - 0.6 < 0) \end{cases}$$

单层感知器



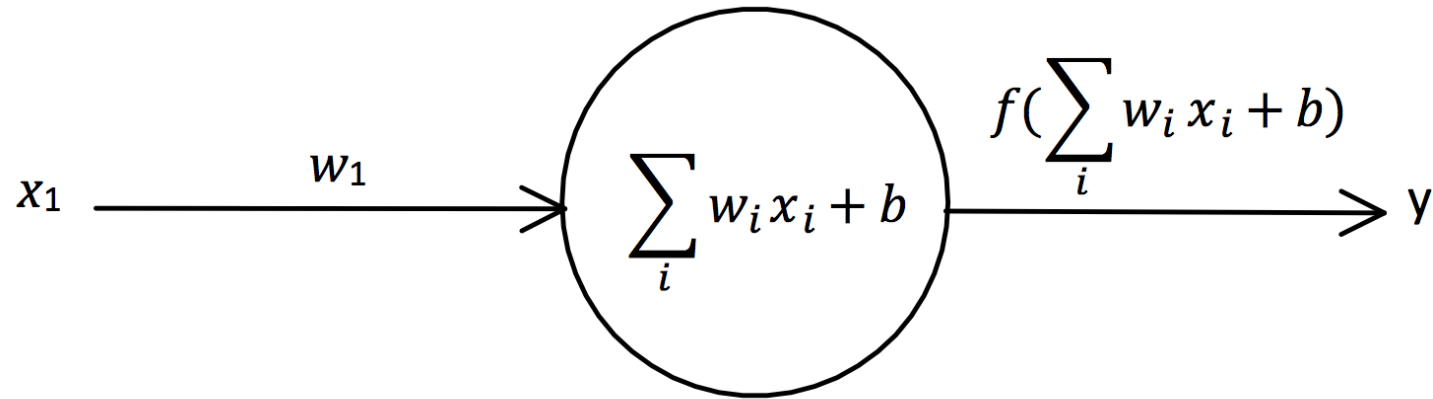
$$y = f \left(\sum_{i=1}^n x_i \cdot w_i \right)$$

$i = 0, 1, 2, \dots$
 y 是网络输出
 f 是sign函数

$$\Delta w_i = \eta(t - y)x_i$$

η 表示学习率
 t 表示正确的标签
 t 和 y 的取值为 ± 1

$$\Delta w_i = \pm 2\eta x_i$$



$$\Delta w_i = \eta(t - y)x_i$$

假设： $t=1$, $\eta=1$, $x_1=1$, $w_1=-5$, $b=0$:

Step1:

$$y = \text{sign}(1 * (-5)) = -1$$

$$\Delta w = 1 * (1 - (-1)) * 1 = 2$$

$$w_1 = w_1 + \Delta w = -3$$

Step2:

$$y = \text{sign}(1 * (-3)) = -1$$

$$\Delta w = 1 * (1 - (-1)) * 1 = 2$$

$$w_1 = w_1 + \Delta w = -1$$

Step3:

$$y = \text{sign}(1 * (-1)) = -1$$

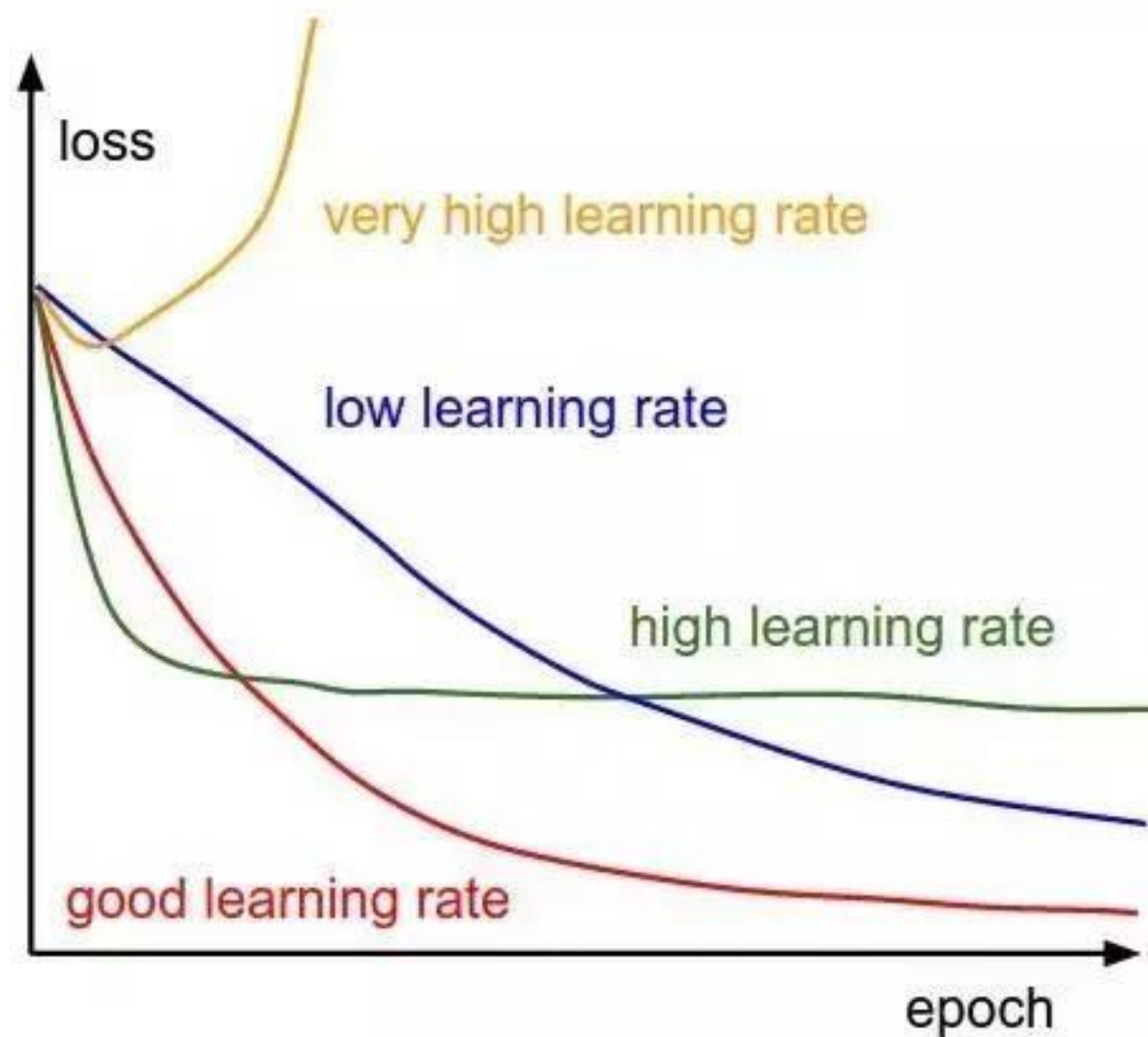
$$\Delta w = 1 * (1 - (-1)) * 1 = 2$$

$$w_1 = w_1 + \Delta w = 1$$

$$y = \text{sign}(1 * 1) = 1 = t$$

- η 取值一般取0-1之间
- 学习率太大容易造成权值调整不稳定
- 学习率太小，权值调整太慢，迭代次数太多

不同学习率

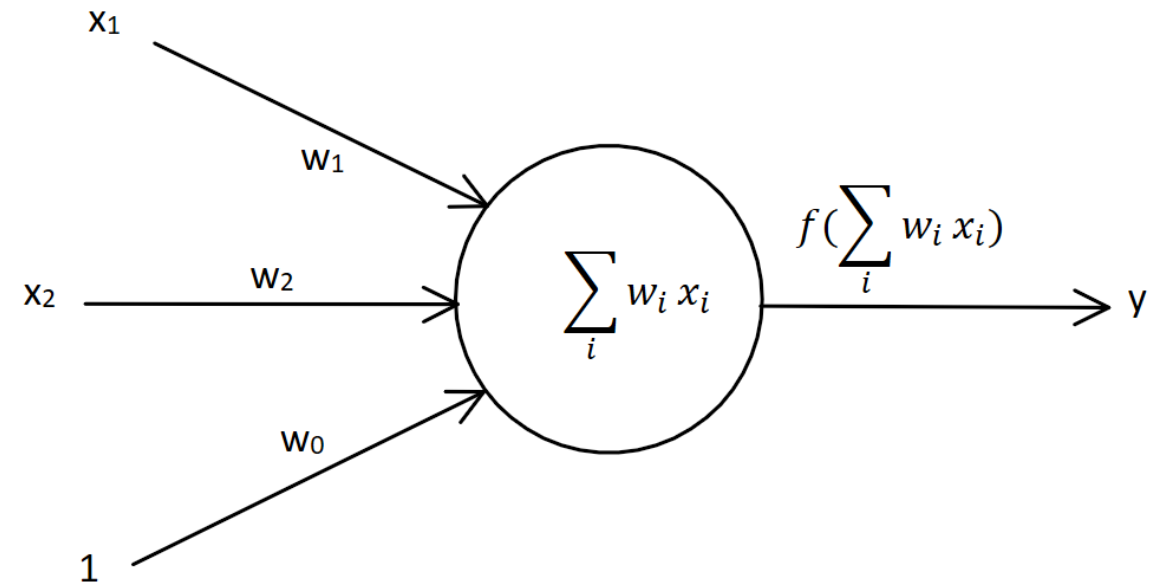


- 误差小于某个预先设定的较小的值
- 两次迭代之间的权值变化已经很小
- 设定最大迭代次数，当迭代超过最大次数就停止

题目：假设平面坐标系上有三个点，(3,3),(4,3)这两个点的标签为1，(1,1)这个点的标签为-1。构建神经网络来分类。

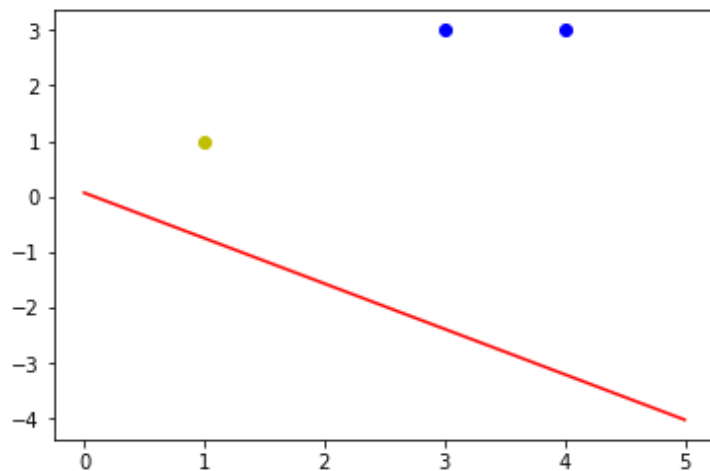
思路：我们要分类的数据是2维数据，所以只需要2个输入节点，我们可以把神经元的偏置值也设置成一个节点，这样我们需要3个输入节点。

输入数据有3个(1,3,3),(1,4,3),(1,1,1)
数据对应的标签为(1,1,-1)
初始化权值 w_0, w_1, w_2 取-1到1的随机数
学习率(learning rate)设置为0.11
激活函数为sign函数

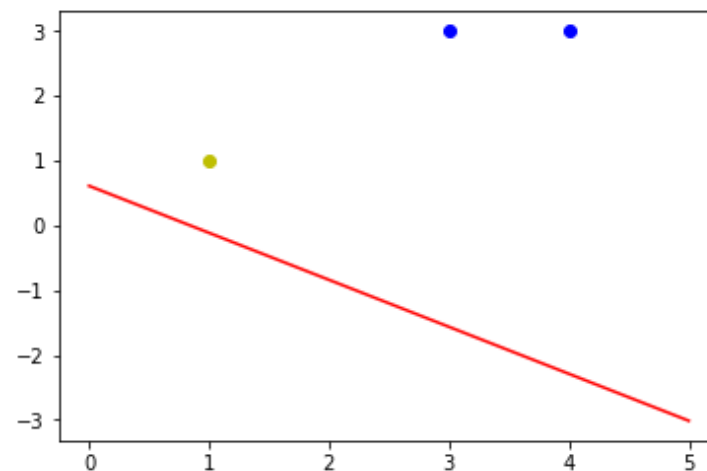


单层感知器分类

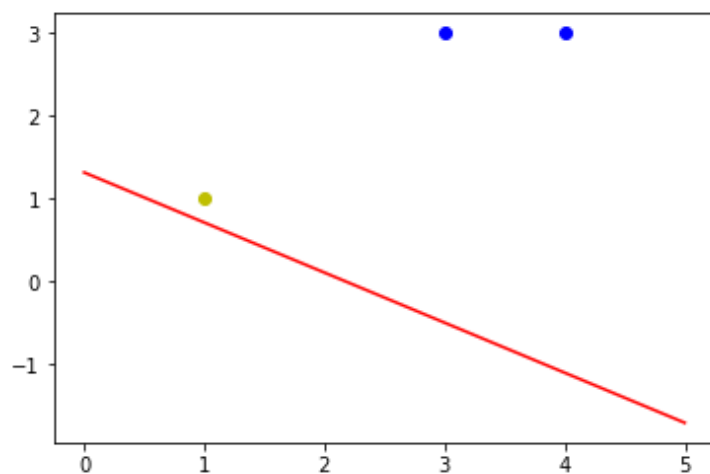
Step1



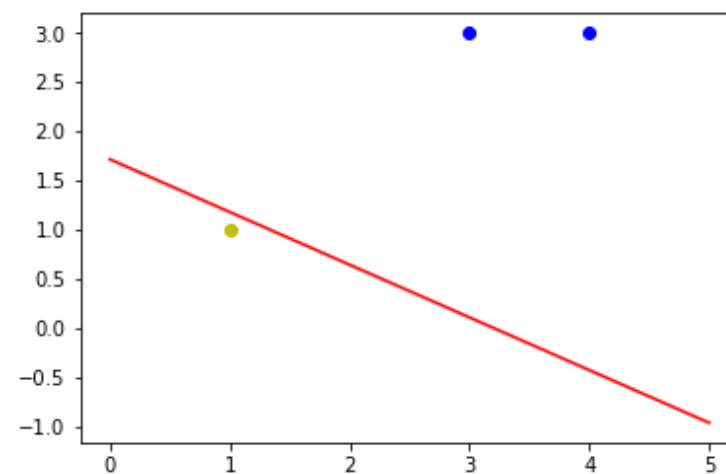
Step2



Step3



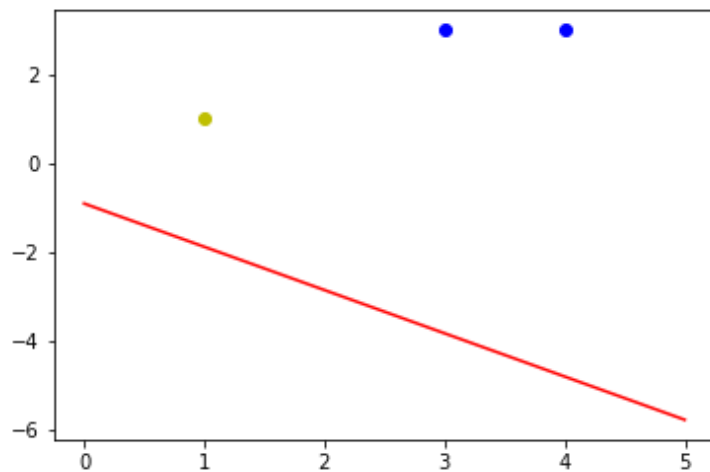
Step4



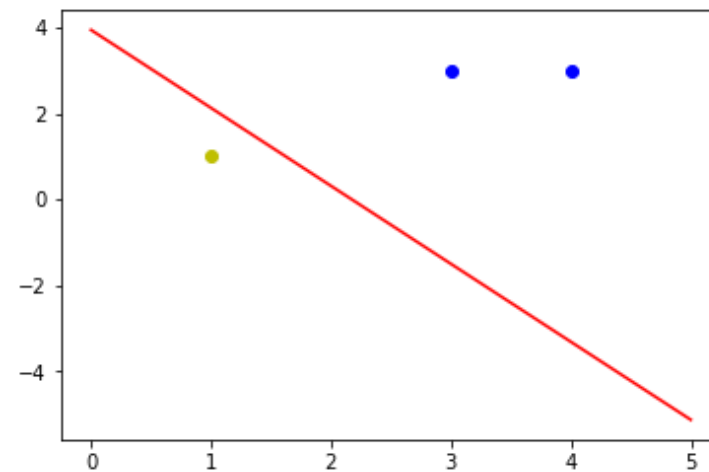
线性神经网络在结构上与感知器非常相似，只是激活函数不同。在模型训练时把原来的sign函数改成了purelin函数： $y = x$

线性神经网络分类

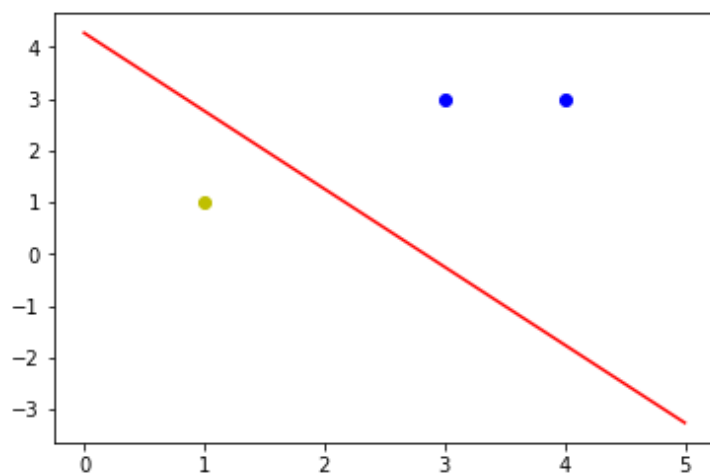
Step1



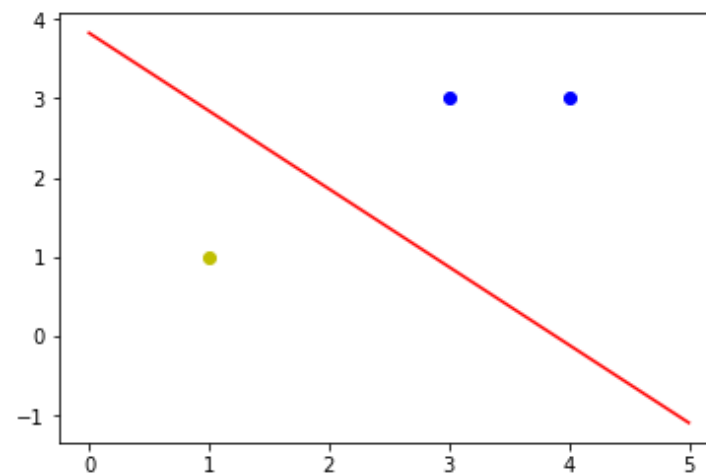
Step2



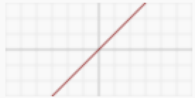

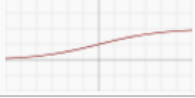
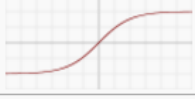
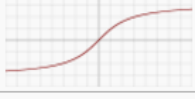




Step3



Step4



线性神经网络异或程序

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

- 1986年，认知心理学家McClelland和Rumelhart在神经网络训练中引入了 δ 规则，该规则也可以称为连续感知器学习规则。
- δ 学习规则是一种利用梯度下降法的一般性的学习规则。

代价函数(损失函数)(Cost Function, Lost Function)

二次代价函数：

$$E = \frac{1}{2}(t - y)^2 = \frac{1}{2}[t - f(WX)]^2$$

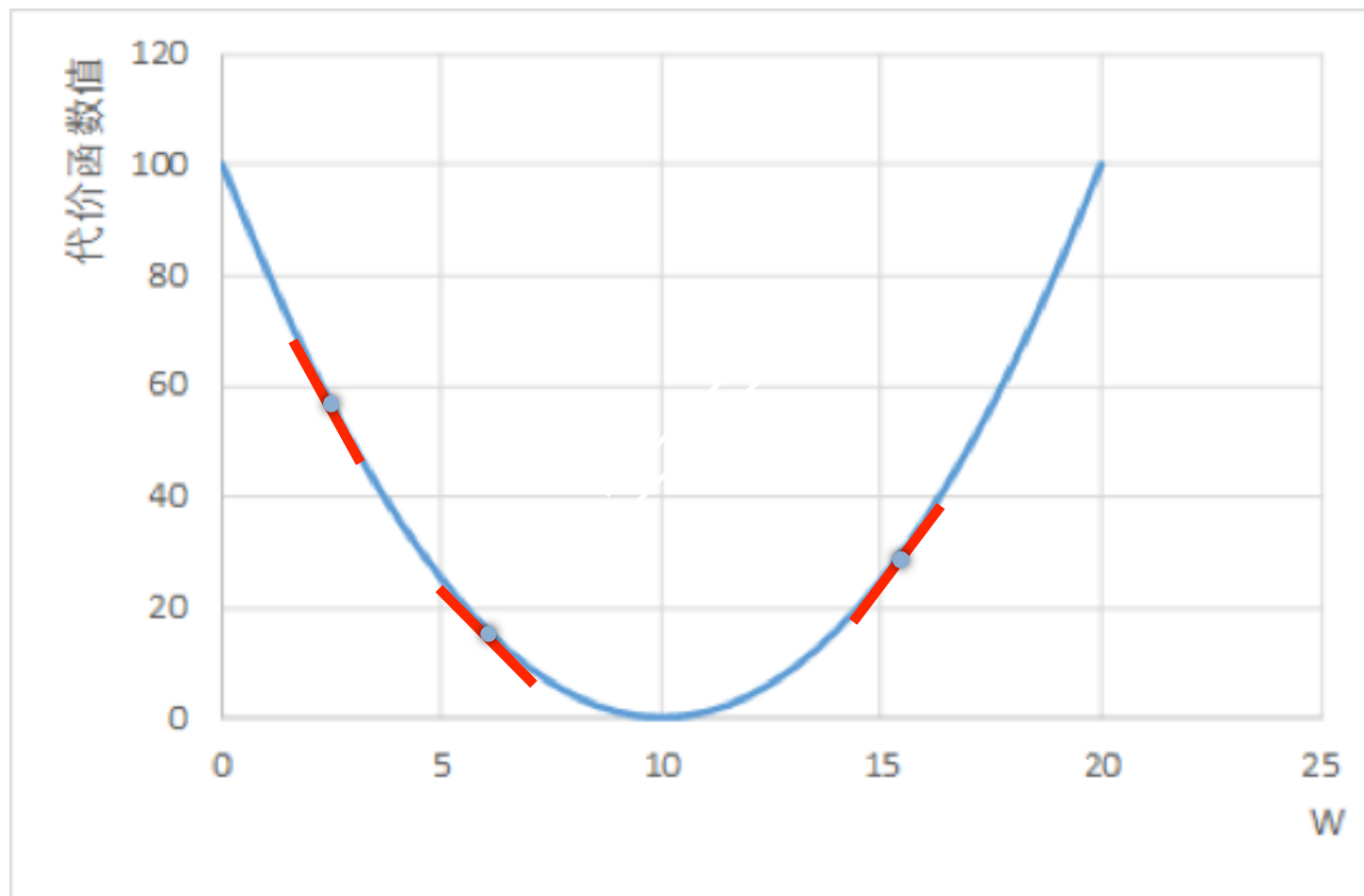
误差E是权向量W的函数，我们可以使用梯度下降法来最小化E的值：

$$\Delta W = -\eta E' = \eta X^T (t - y) f'(WX) = \eta X^T \delta$$

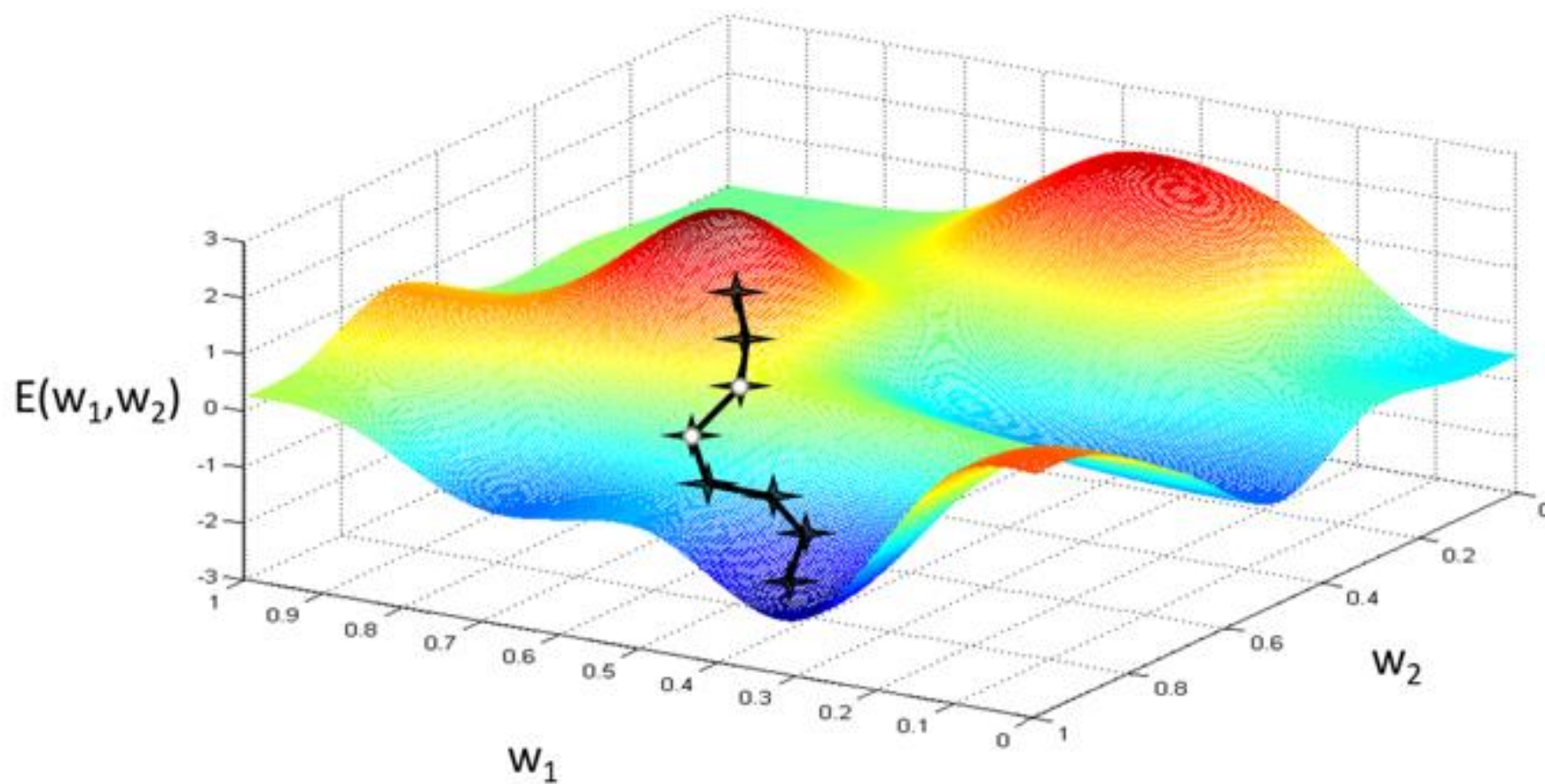
$$\Delta w_i = -\eta E' = \eta x_i (t - y) f'(WX) = \eta x_i \delta$$

梯度下降法-一维情况

$$w = w + \Delta w$$

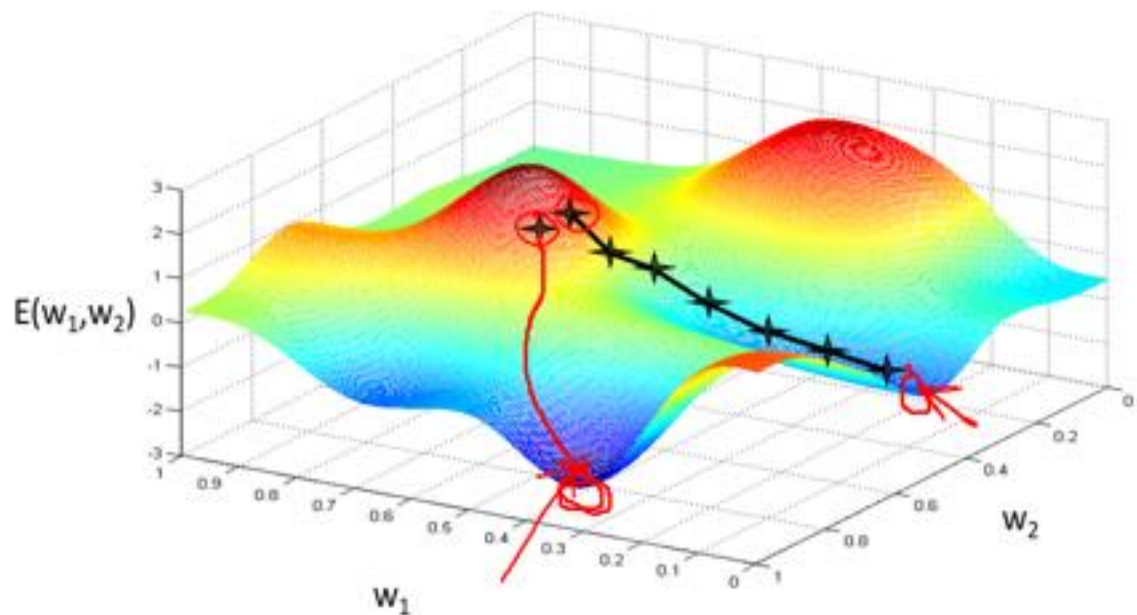
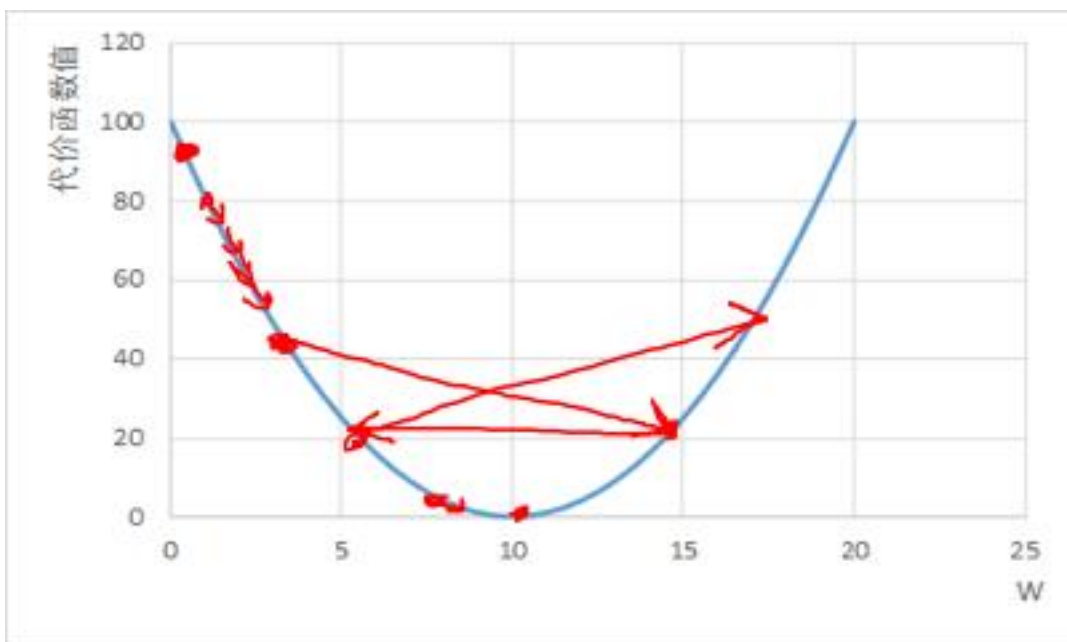


梯度下降法-二维情况



梯度下降法的问题

1. 学习率难以选取，太大会产生震荡，太小收敛缓慢
2. 容易陷入局部最优解(局部极小值)

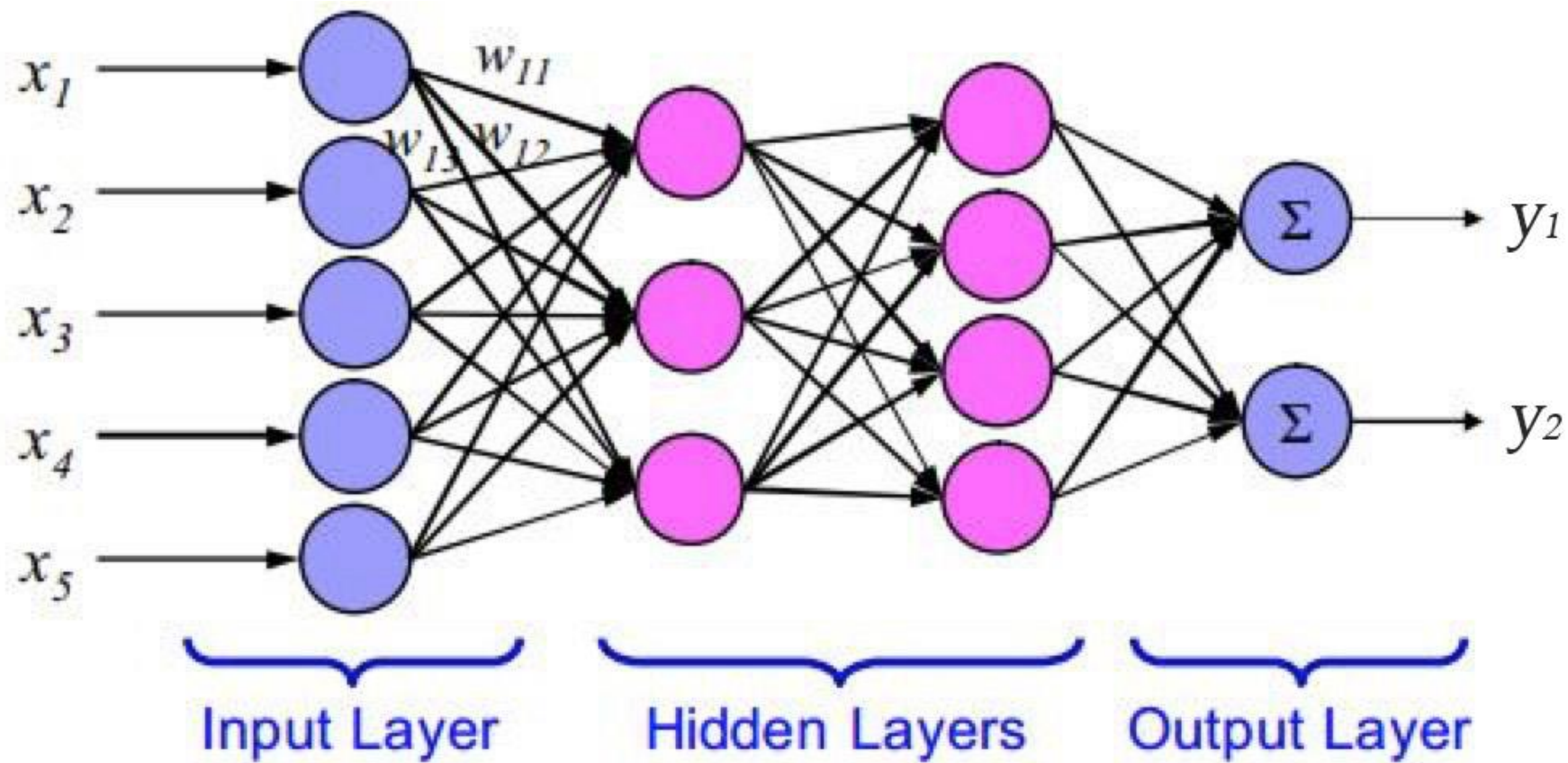




BP神经网络

(Back Propagation Neural Network)

- 1986年，由McClelland和Rumelhart为首的科学家小组提出，解决了多层神经网络的学习问题，极大促进了神经网络的发展。
- BP神经网络也是整个人工神经网络体系中的精华，广泛应用于分类识别，逼近，回归，压缩等领域。在实际应用中，大约80%的神经网络模型都采取了BP网络或BP网络的变化形式。



$$E = \frac{1}{2}(t - y)^2$$

Delta学习规则

$$\frac{\partial E}{\partial W^l} = -(X^l)^T \delta^l$$

$$\Delta W^l = -\eta \frac{\partial E}{\partial W^l} = \eta (X^l)^T \delta^l$$

.....

$$\delta^L = (t - y)f'(X^L W^L)$$

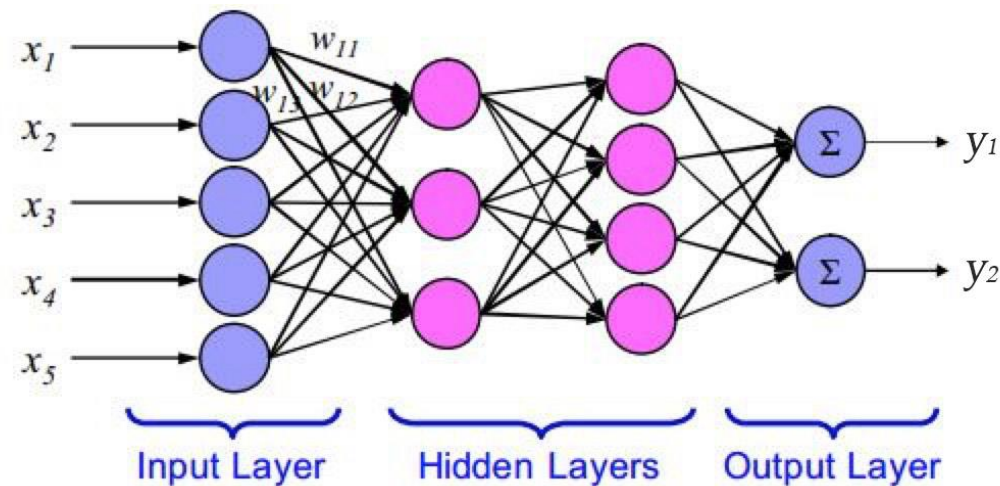
δ^l 第l层学习信号

δ^L 输出层学习信号

W^l 第l-1+1层权值

X^l 第l层的输入信号

$$\delta^l = \delta^{l+1}(W^{l+1})^T f'(X^l W^l)$$



$$\Delta W^{l3} = -\eta \delta^{l3} X^{l3} = \eta (t - y) f'(X^{l3} W^{l3}) X^{l3}$$

$$\delta^{l3} = (t - y) f'(X^{l3} W^{l3})$$

$$\Delta W^{l2} = -\eta \delta^{l2} X^{l2} = \eta \delta^{l3} (W^{l3})^T f'(X^{l2} W^{l2}) X^{l2}$$

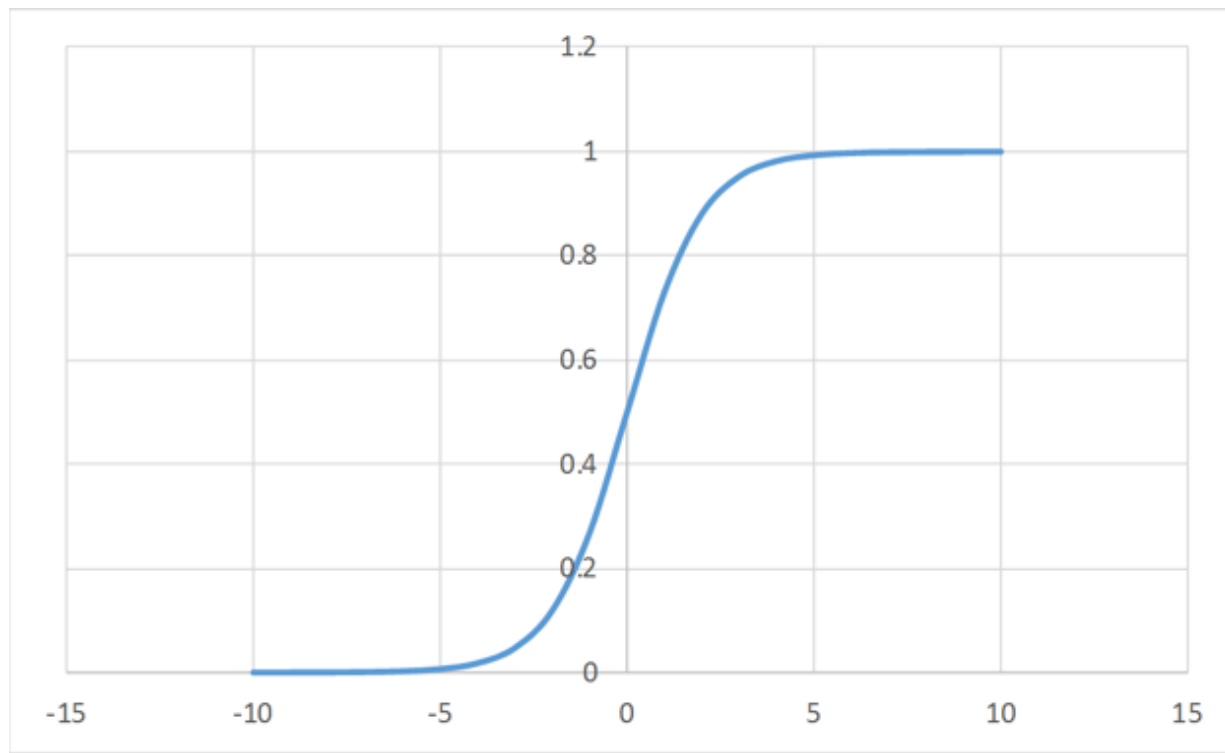
$$\delta^{l2} = \delta^{l3} (W^{l3})^T f'(X^{l2} W^{l2})$$

$$\Delta W^{l1} = -\eta \delta^{l1} X^{l1} = \eta \delta^{l2} (W^{l2})^T f'(X^{l1} W^{l1}) X^{l1}$$

$$\delta^{l1} = \delta^{l2} (W^{l2})^T f'(X^{l1} W^{l1})$$

Sigmoid函数

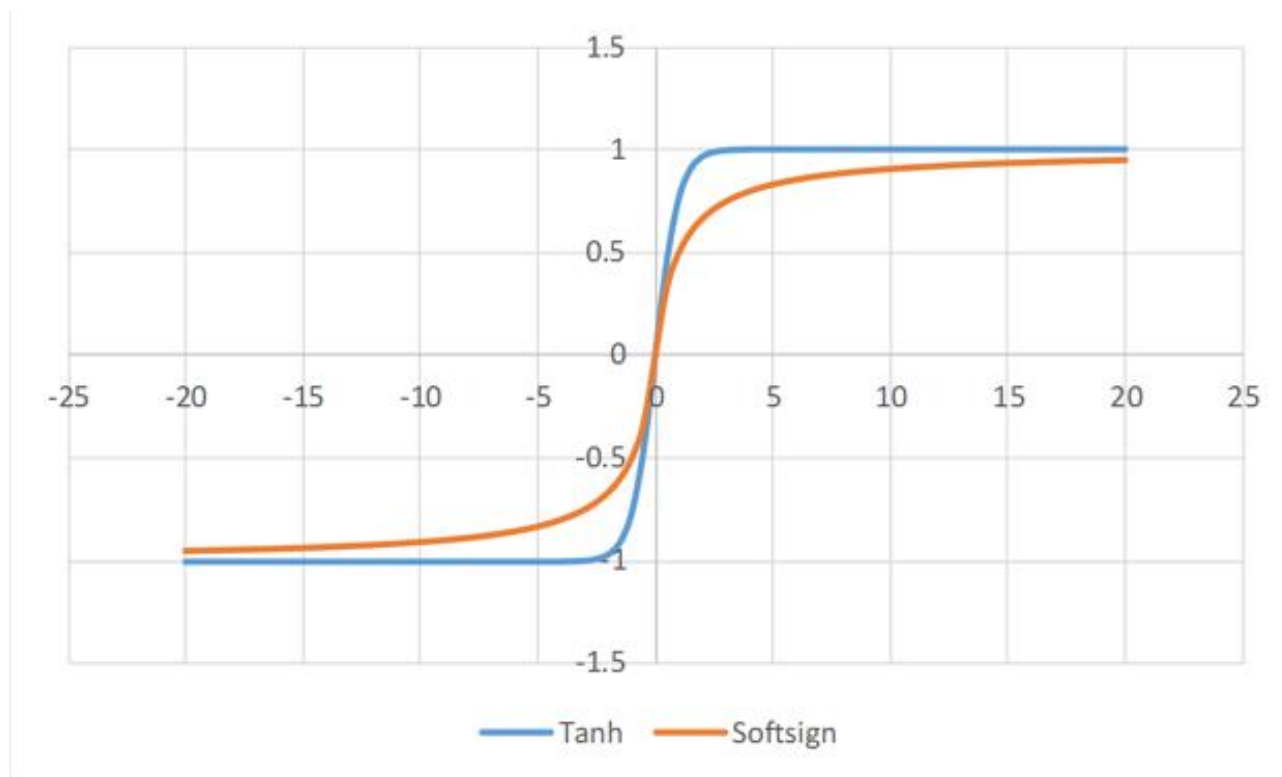
$$f(x) = \frac{1}{1 + e^{-x}}$$



Tanh函数和Softsign函数

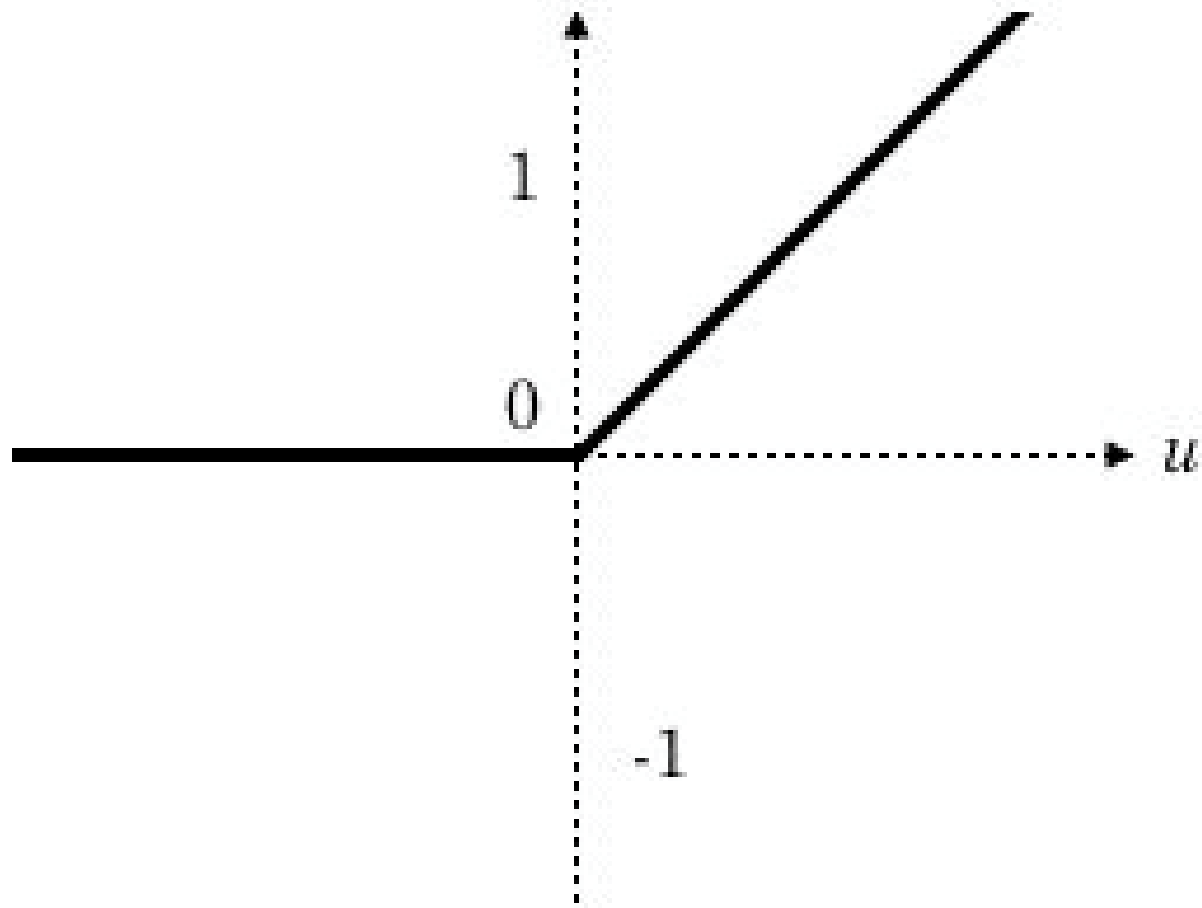
Tanh函数：
$$\frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Softsign函数：
$$\frac{x}{1 + |x|}$$



ReLU函数

$$f(u) = \max(0, u)$$



BP网络解决异或问题

BP网络用于手写数字识别

sklearn-神经网络-手写数字识别

正确率与召回率 (Precision & Recall) 是广泛应用于信息检索和统计学分类领域的两个度量值，用来评价结果的质量。

一般来说，正确率就是检索出来的条目有多少是正确的，召回率就是所有正确的条目有多少被检索出来了。

$$F1 \text{ 值} = 2 * \frac{\text{正确率} * \text{召回率}}{\text{正确率} + \text{召回率}}$$
。是综合上面二个指标的评估指标，用于综合反映整体的指标。

这几个指标的取值都在0-1之间，数值越接近于1，效果越好。

某池塘有1400条鲤鱼，300只虾，300只鳖。现在以捕鲤鱼为目的。撒一大网，逮着了700条鲤鱼，200只虾，100只鳖。那么，这些指标分别如下：

$$\text{正确率} = 700 / (700 + 200 + 100) = 70\%$$

$$\text{召回率} = 700 / 1400 = 50\%$$

$$\text{F值} = 70\% * 50\% * 2 / (70\% + 50\%) = 58.3\%$$

某池塘有1400条鲤鱼，300只虾，300只鳖。现在以捕鲤鱼为目的。撒一大网，逮着了所有的鱼虾鳖：

$$\text{正确率} = 1400 / (1400 + 300 + 300) = 70\%$$

$$\text{召回率} = 1400 / 1400 = 100\%$$

$$\text{F值} = 70\% * 100\% * 2 / (70\% + 100\%) = 82.35\%$$

正确率与召回率指标有时候会出现的矛盾的情况，这样就需要综合考虑他们，最常见的方法就是F-Measure（又称为F-Score）：

$$F_{\beta} = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}}$$

当 $\beta = 1$ 时，就是常见的F1指标：

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

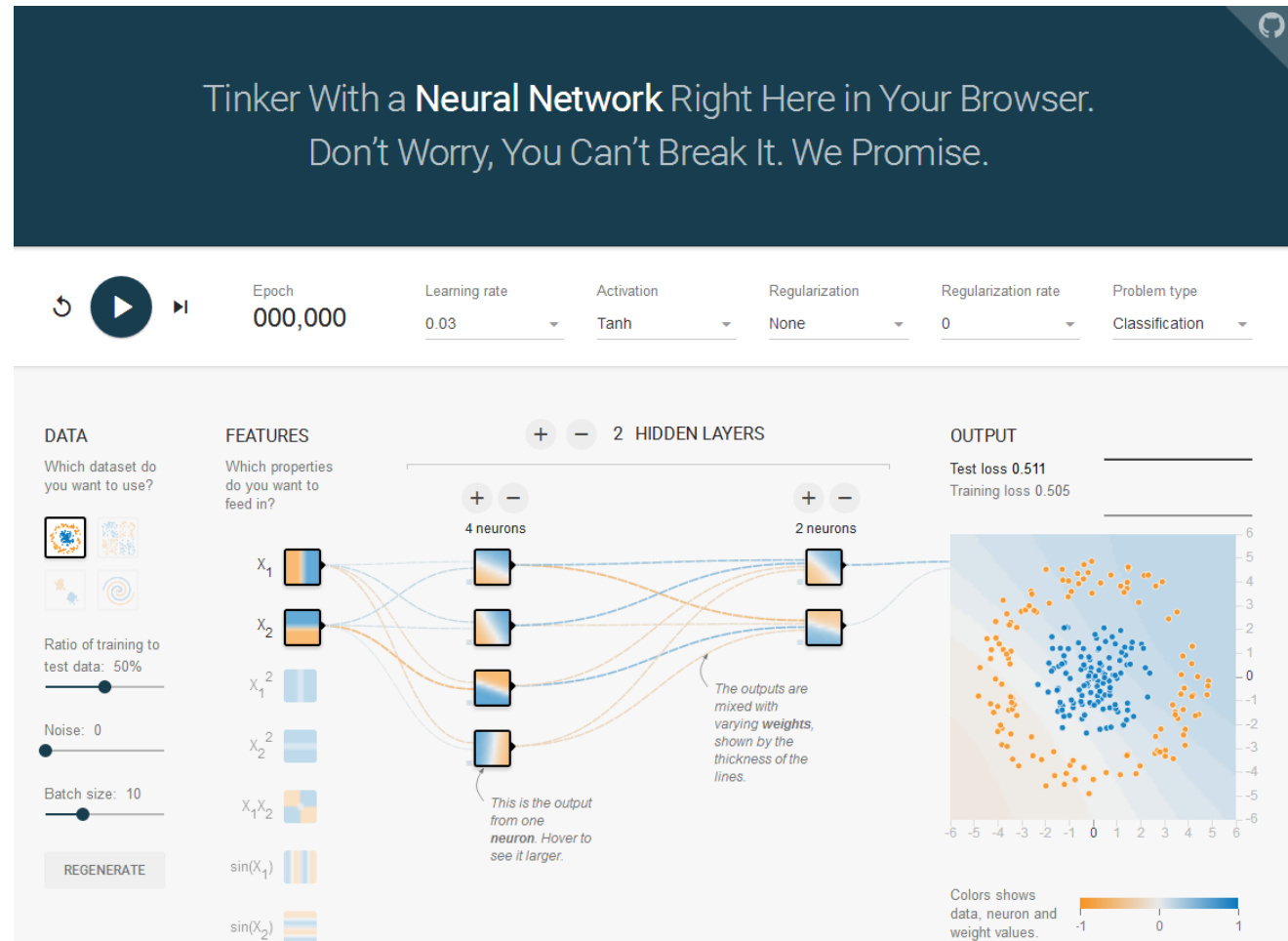
假设有一个用来对猫（cats）、狗（dogs）、兔子（rabbits）进行分类的系统，混淆矩阵就是为了进一步分析性能而对该算法测试结果做出的总结。

		Predicted class		
		Cat	Dog	Rabbit
Actual class	Cat	5	3	0
	Dog	2	3	1
	Rabbit	0	2	11

在这个混淆矩阵中，实际有 8 只猫，但是系统将其中 3 只预测成了狗。对于 6 条狗，其中有 1 条被预测成了兔子，2 条被预测成了猫。

Understanding the difficulty of training deep feedforward neural networks

<http://playground.tensorflow.org/>





Tensorflow

- Windows, MacOS, Linux都已支持Tensorflow
- Windows用户支持python3.5+(64bit)。MacOS和Linux支持python2.7和python3.3+的版本。
- 有GPU可以安装GPU的版本，没有GPU就安装CPU的版本。

- Windows安装Tensorflow

CPU版本：

管理员方式打开命令提示符，输入命令：`pip install tensorflow`

GPU版本：

管理员方式打开命令提示符，输入命令：`pip install tensorflow-gpu`

- 更新Tensorflow

`pip install tensorflow --upgrade`

- NOTE: TensorFlow requires MSVCP140.DLL, which may not be installed on your system. If, when you import tensorflow as tf, you see an error about No module named "_pywrap_tensorflow" and/or DLL load failed, check whether MSVCP140.DLL is in your %PATH% and, if not, you should install the [Visual C++ 2015 redistributable](#) (x64 version).

- Linux和MacOS安装Tensorflow

CPU版本：

Python2.7用户：pip install tensorflow

Python3.3+用户：pip3 install tensorflow

GPU版本：

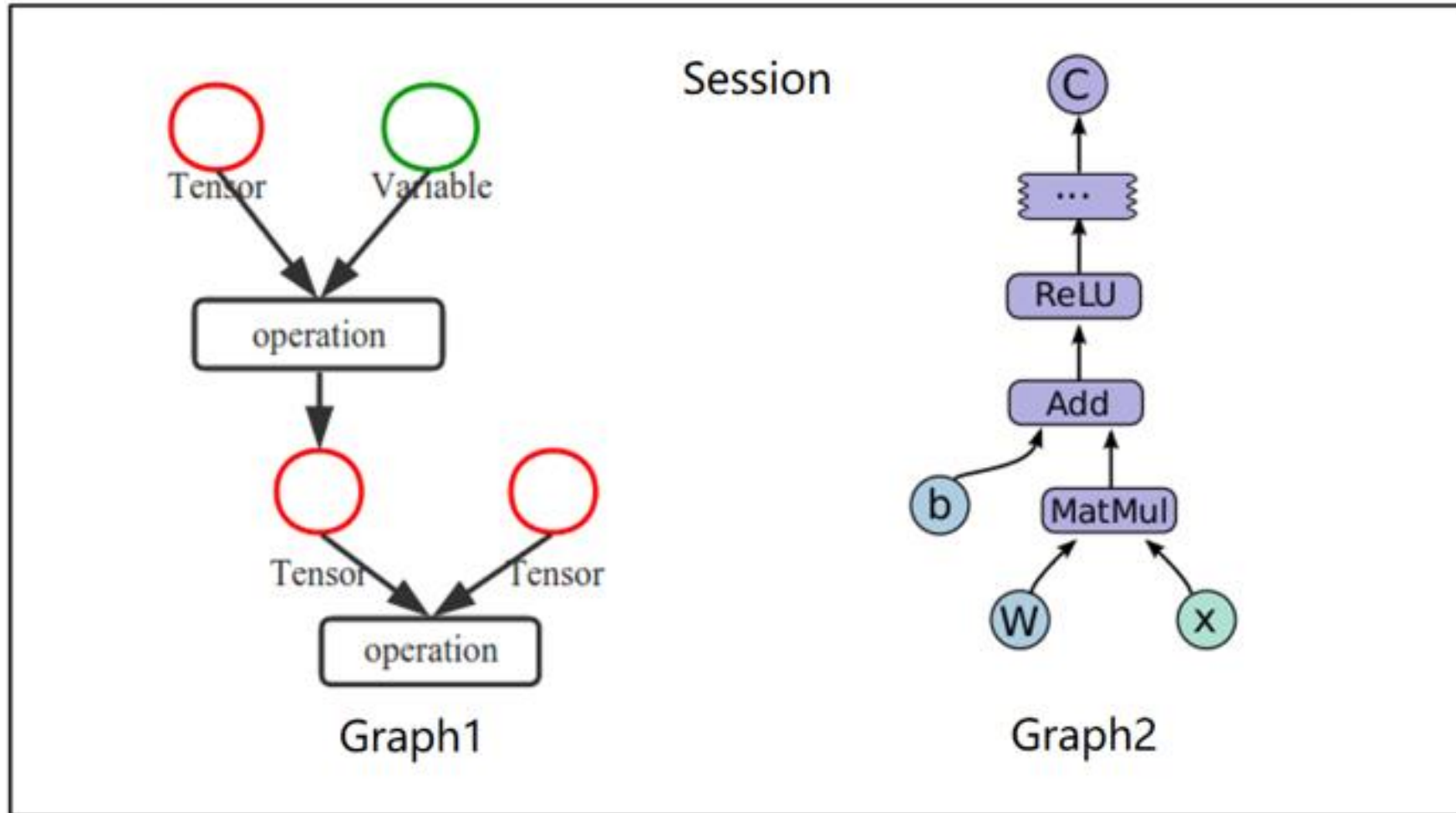
Python2.7用户：pip install tensorflow-gpu

Python3.3+用户：pip3 install tensorflow-gpu

- 使用图(graphs)来表示计算任务
- 在会话(Session)中执行图
- 使用Tensor表示数据
- 通过变量(Variable)维护状态
- 使用feed和fetch可以为任意的操作赋值或取值

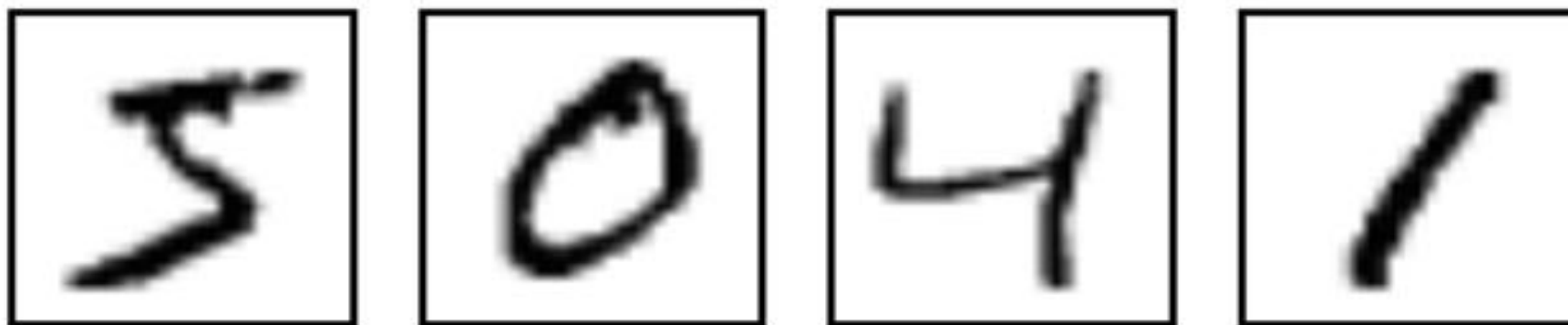
Tensorflow是一个编程系统，使用图(graphs)来表示计算任务，图(graphs)中的节点称之为op(operation)，一个op获得0个或多个Tensor，执行计算，产生0个或多个Tensor。Tensor看作是一个n维的数组或列表，图必须在会话(Session)里被启动。

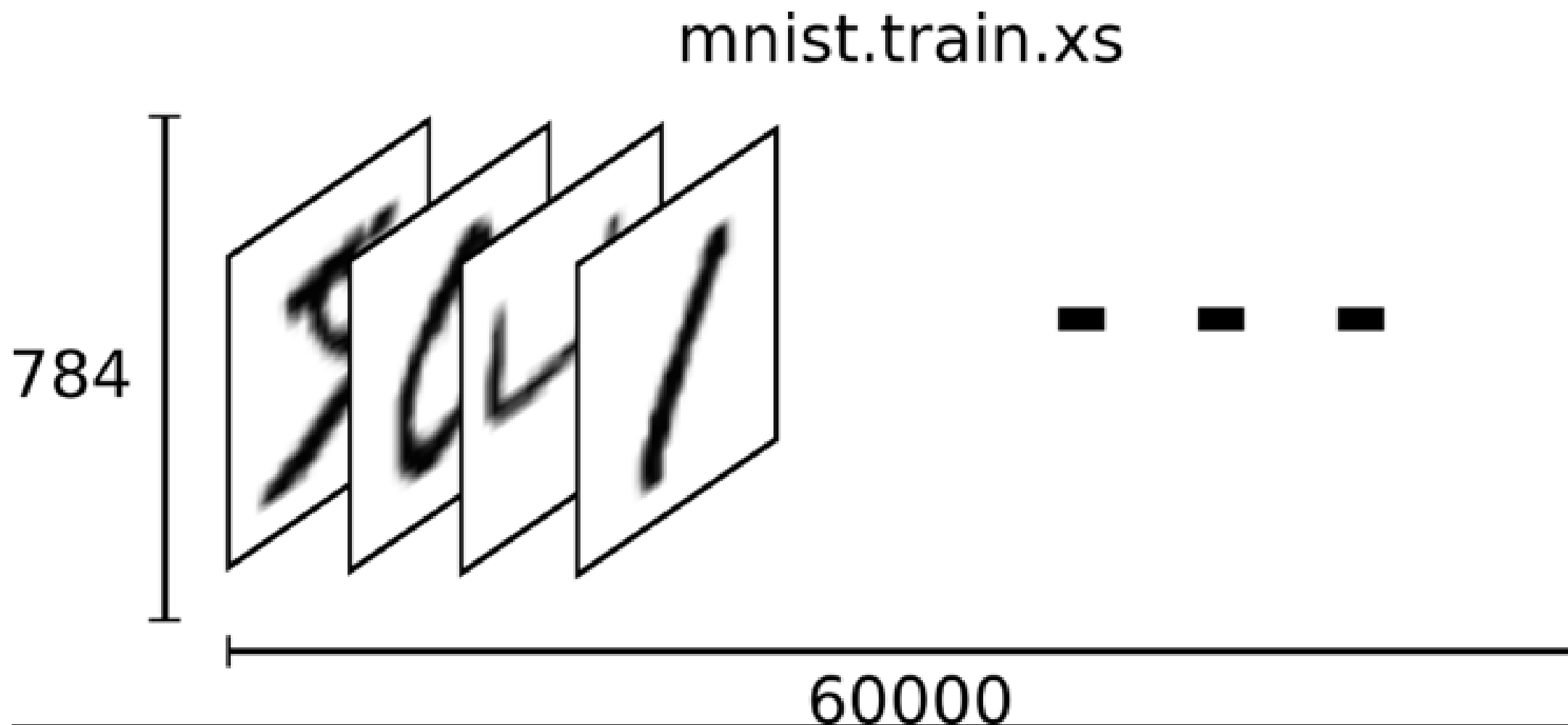
Session



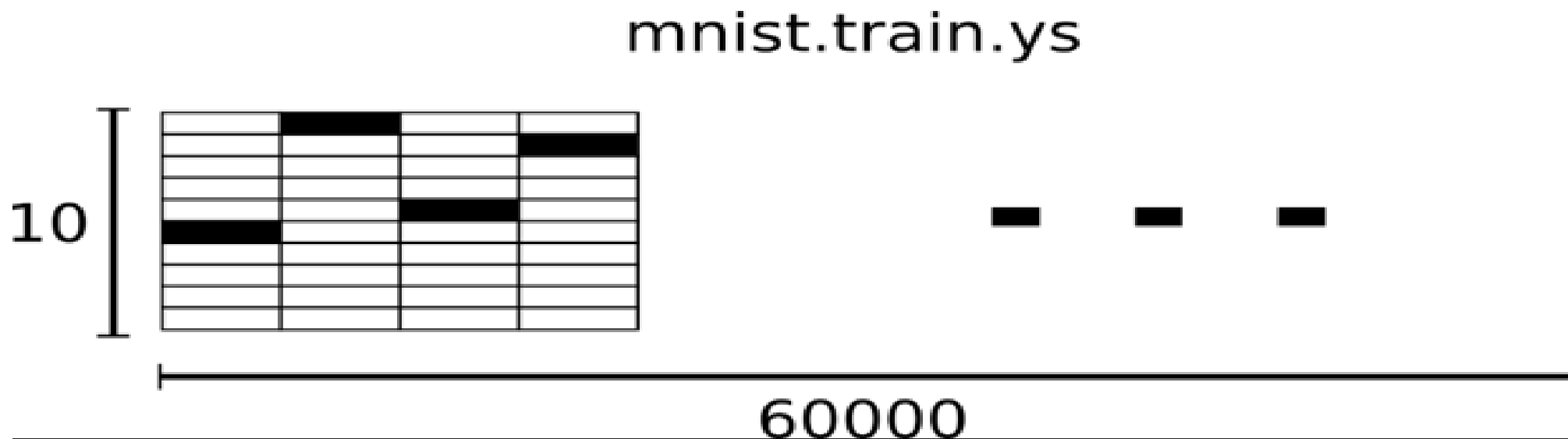
- 1.创建图，启动图
- 2.变量
- 3.Fetch and Feed
- 4.Tensorflow线性回归示例
- 5.Tensorflow非线性回归示例

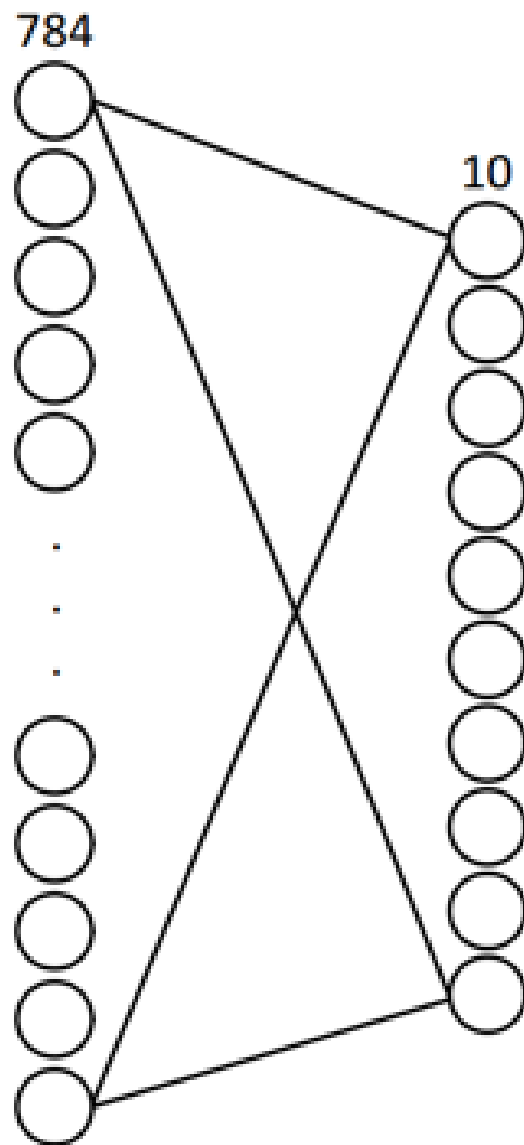
- MNIST数据集官网：[Yann LeCun's Website](http://yann.lecun.com/ex/ex2/mnist/)
- 下载下来的数据集被分成两部分：60000行的训练数据集（mnist.train）和10000行的测试数据集（mnist.test）





- MNIST数据集的标签是介于0-9的数字，我们要把标签转化为“one-hot vectors”。一个one-hot向量除了某一位数字是1以外，其余维度数字都是0，比如标签0将表示为([1,0,0,0,0,0,0,0,0,0])，标签3将表示为([0,0,0,1,0,0,0,0,0,0])。
- 因此，mnist.train.labels 是一个 [60000, 10] 的数字矩阵。





- 我们知道MNIST的结果是0-9，我们的模型可能推测出一张图片是数字9的概率是80%，是数字8的概率是10%，然后其他数字的概率更小，总体概率加起来等于1。这是一个使用softmax回归模型的经典案例。softmax模型可以用来给不同的对象分配概率。

$$\text{softmax}(x)_i = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

某个神经网络有3个输出值：[1,5,3]

$$e^1 = 2.718$$

$$e^5 = 148.413$$

$$e^3 = 20.086$$

$$e^1 + e^5 + e^3 = 171.217$$

$$p1 = \frac{e^1}{e^1 + e^5 + e^3} = 0.016$$

$$p2 = \frac{e^5}{e^1 + e^5 + e^3} = 0.867$$

$$p3 = \frac{e^3}{e^1 + e^5 + e^3} = 0.117$$

- 6.MNIST数据集分类简单版本

Thanks!
