



# Foundation of Deep Learning(1)

Bing Gong  
( 玖冰 )

[gongbing@shnu.edu.cn](mailto:gongbing@shnu.edu.cn)



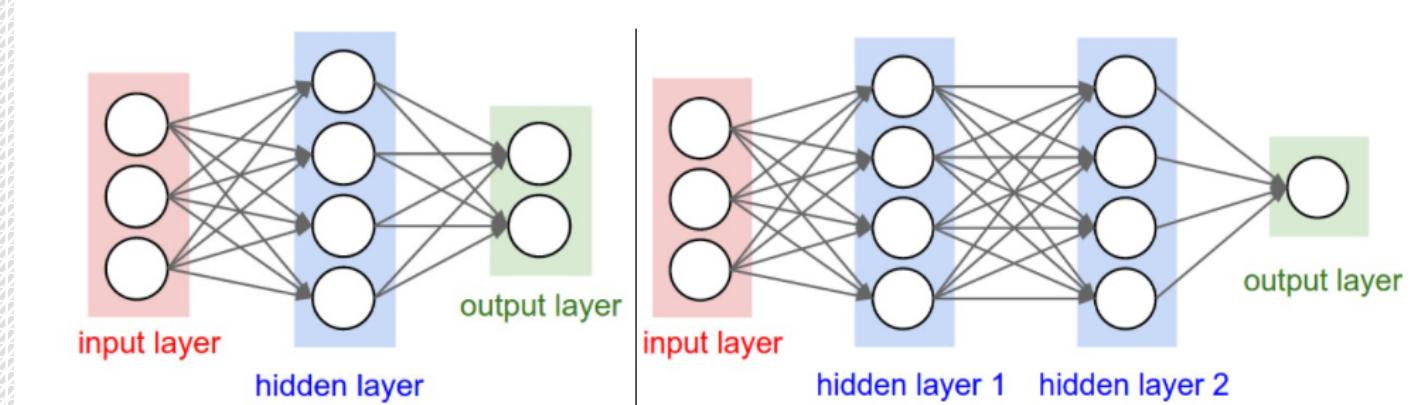
# Objectives:

- Foundation of deep learning 深度学习基础
- Activation functions (激活函数)
- Loss function (损失函数)
- Convolutional Neural Network (卷积神经网络)



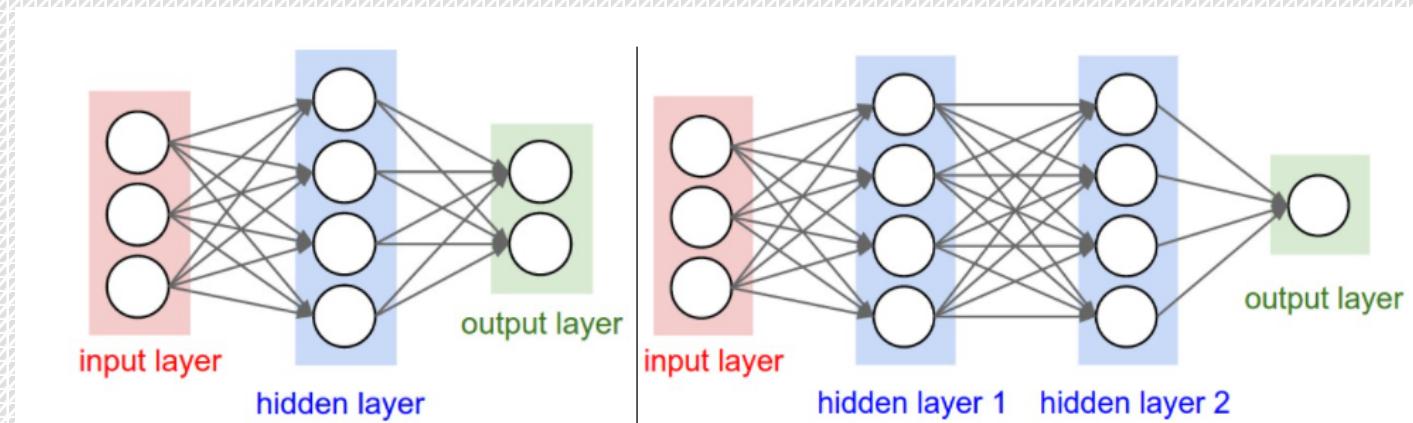
# Deep neural network

- 深度神经网络是一种模仿生物神经系统结构与功能的模型，它由一系列可调节参数的人工神经元组成，经过大量数据训练（调节参数）后，可以用来对函数进行估计或近似。
- 传统机器学习技术在处理图像时，往往依赖于手工设计的特征，但深度学习基于端到端训练，可以直接从原始像素中提取特征并输出任务结果。



# Deep neural network

- 一个简单的深度神经网络如图所示，包括输入层、若干隐藏层和输出层，每一层都由若干神经元组成。输入层用来输入数据，隐藏层可以从数据中逐步提取特征，输出层计算产生网络的输出结果。





# Steps of establishing DL model

设计网  
络模型

选定损  
失函数

参数初  
始化

模型优  
化

使用模  
型

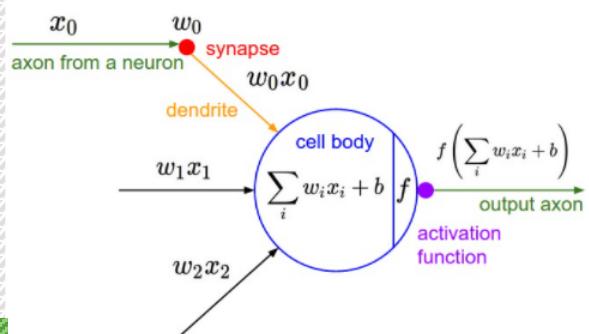
# Deep neural network

## • 基本结构：人工神经元

深度神经网络由成千上万个人工神经元连结而成，而单个人工神经元会对输入数据执行加权求和操作，并通过激活函数进行非线性映射：

$$z = h(\sum_i w_i x_i) = h(\mathbf{w}^T \mathbf{x} + b)$$

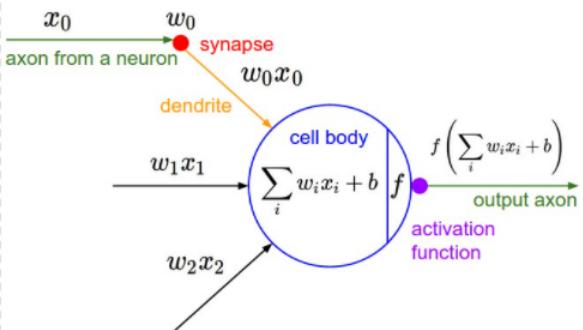
其中  $x$  代表输入数据，  $w$  为该神经元的权重 ( weight )，  $b$  为该神经元的偏置(bias)，  $h(\cdot)$  代表激活函数，  $z$  为该神经元的激活值(activation)。



# Deep neural network

## • 基本结构：人工神经元

单个人工神经元的参数包括其权重与偏置，以上图的神经元为例，其包括4个可调节参数，即 $w_0$ ， $w_1$ ， $w_2$ 和 $b$ 。这些参数可以通过模型训练进行调节。

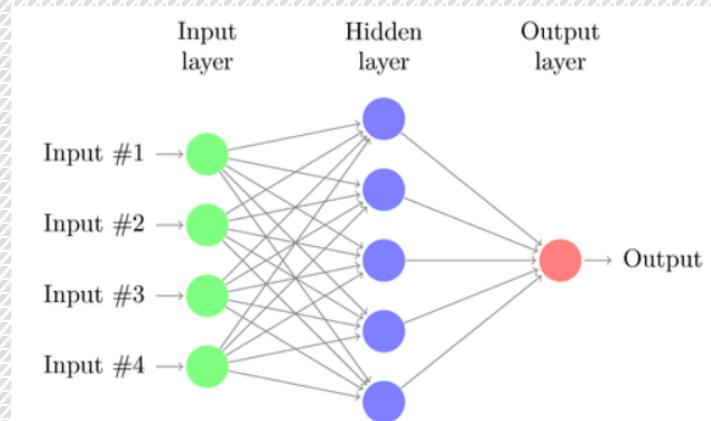


# Deep neural network

神经将网络并不会将神经元胡乱连接，而是将它们组织为连续的层，网络中的层将上一层的结果作为输入，经过本层进一步计算处理，然后将结果传递给下一层：

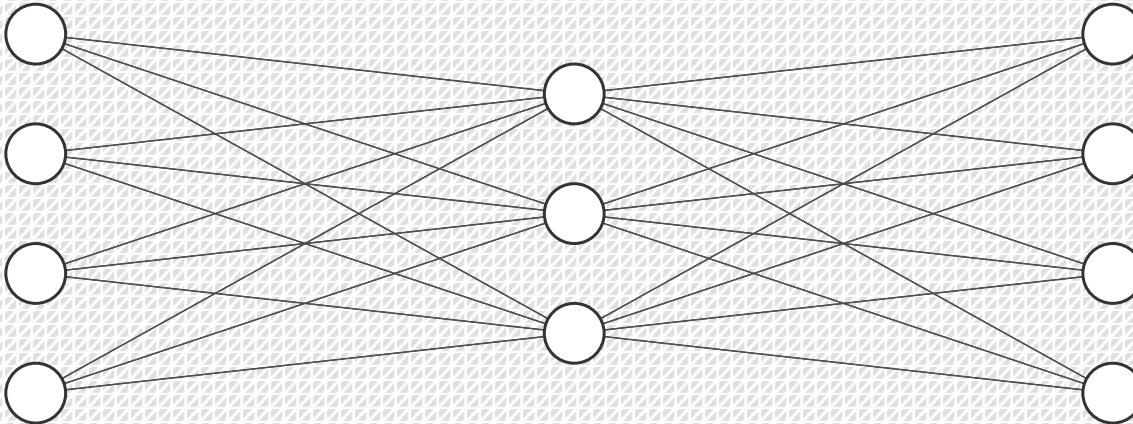
$$\mathbf{z} = \mathbf{h}(\mathbf{Wx} + \mathbf{b})$$

其中 $\mathbf{W}$ 代表权重矩阵。网络中的这些层可以从原始数据中逐步提取特征并用于相关任务。某一层的神经元与前一层的神经元全部两两相连接，这一层为全连接层（Fully Connected Layer）。



# Deep neural network

- ◆ Questions : how many parameters?



Input Layer  $\in \mathbb{R}^4$

Hidden Layer  $\in \mathbb{R}^3$

Output Layer  $\in \mathbb{R}^4$

# Deep neural network

- Why activation function is needed?



为什么要使用激活函数？若没有激活函数，神经网络仅仅是普通的线性模型，对输入数据进行线性变换，表达能力有限，当加入非线性激活函数后，网络可以拟合更复杂的函数。



# Deep neural network

## • 激活函数性质

- 是连续可导的非线性函数（在少数点上允许不可导）：函数可导，确保可以通过梯度下降调节网络参数；非线性函数增强网络的拟合能力。
- 尽可能简单：计算简单的激活函数可以提高模型的计算效率。
- 激活函数的导函数的值域要落在一个合理的区间内：导函数值过大或过小都会影响模型训练的效率和稳定性。



# Deep neural network

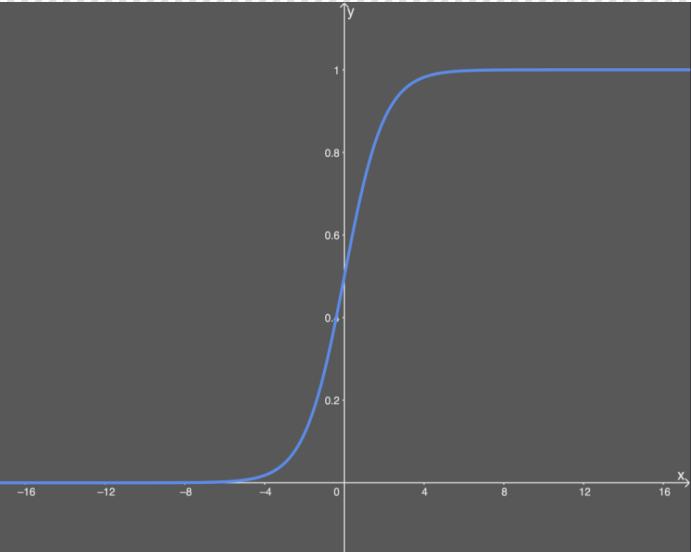
## • 激活函数

Identity	Sigmoid	TanH	ArcTan
ReLU	Leaky ReLU	Randomized ReLU	Parameteric ReLU
Binary	Exponentional Linear Unit	Soft Sign	Inverse Square Root Unit (ISRU)
Inverse Square Root Linear	Square Non-Linearity	Bipolar ReLU	Soft Plus

不同类型的激活函数

# Deep neural network

- Sigmoid激活函数

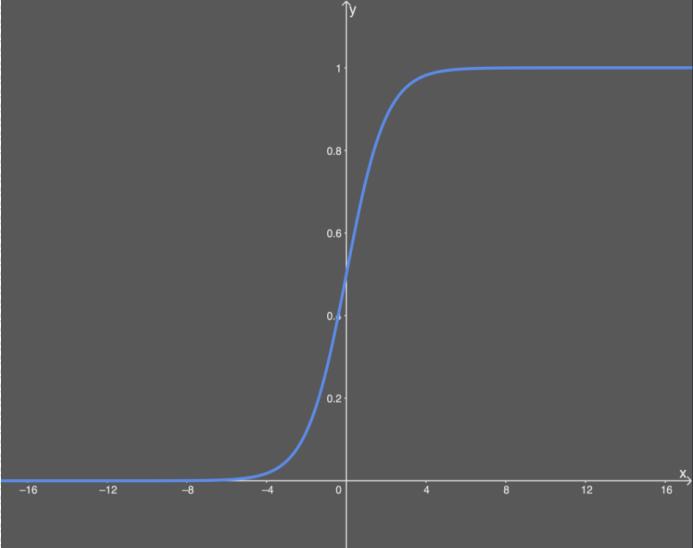


$$h(x) = \frac{1}{1 + e^{-x}}$$

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

# Deep neural network

- Sigmoid激活函数



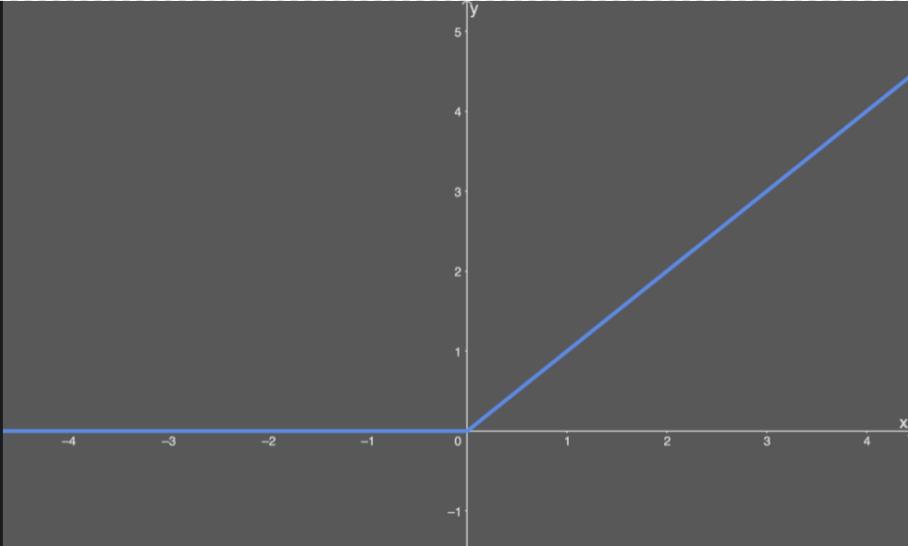
Pros: 具有良好的可解释性。

Cons: Sigmoid激活函数是饱和激活函数，易造成梯度消失；

Sigmoid激活函数中存在 $\exp(\cdot)$ 操作，计算较复杂。

# Deep neural network

- ReLU激活函数

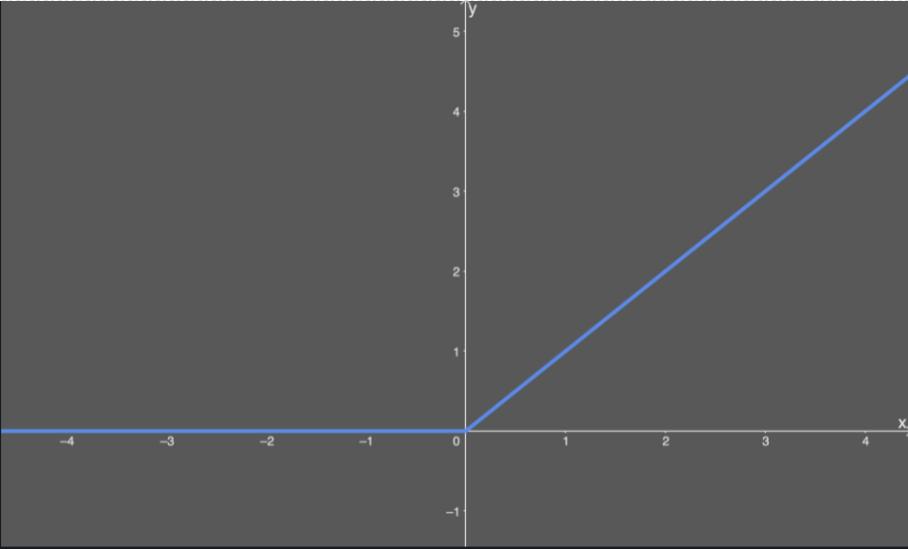


$$h(x) = \max(0, x)$$

$$h'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$$

# Deep neural network

- ReLU激活函数

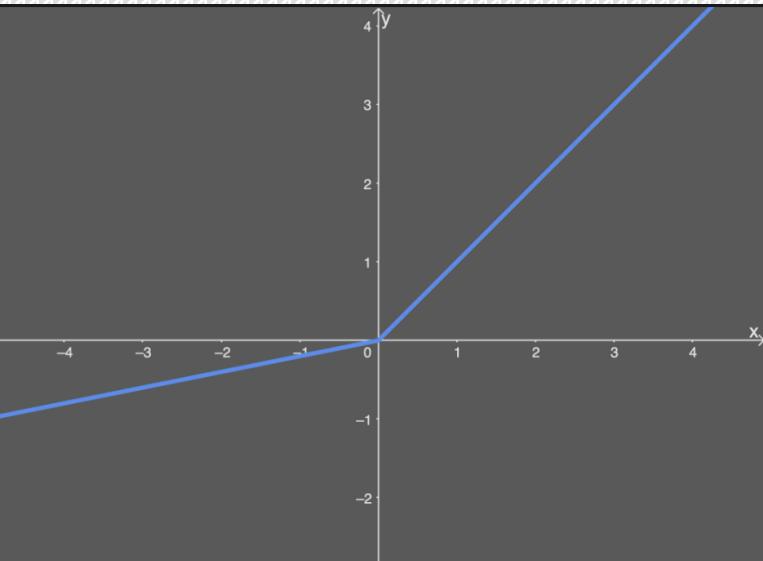


Pros: 计算简单、高效；非饱和激活函数，避免了梯度消失的问题；往往比 Sigmoid/Tanh等激活函数收敛更快。

Cons: 存在dead ReLU问题。

# Deep neural network

- Leaky ReLU激活函数



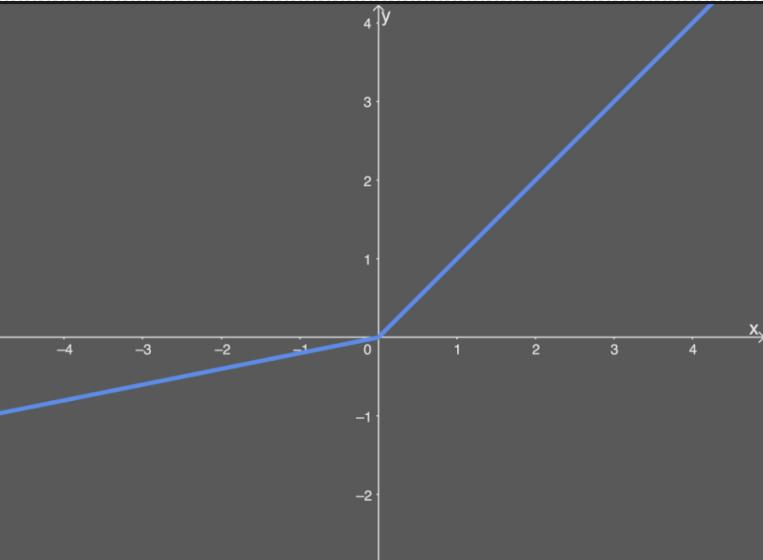
$$h(x) = \begin{cases} x, & x > 0 \\ \alpha x, & x \leq 0 \end{cases}$$

$$h'(x) = \begin{cases} 1, & x > 0 \\ \alpha, & x \leq 0 \end{cases}$$

其中 $\alpha$ 为一预设值，通常设置为0.01左右。

# Deep neural network

- Leaky ReLU激活函数

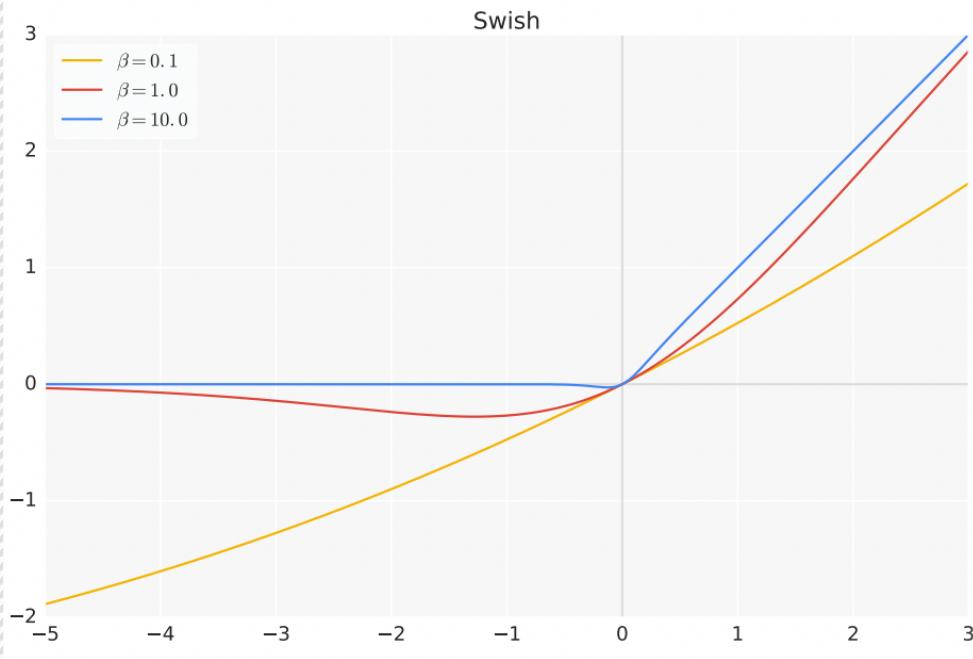


Pros: 与ReLU性质相似，但是可以避免dead ReLU问题。

Cons: 需要调节 $\alpha$ 值，在实际操作中并不能保证总是优于ReLU。

# Deep neural network

- Swish激活函数

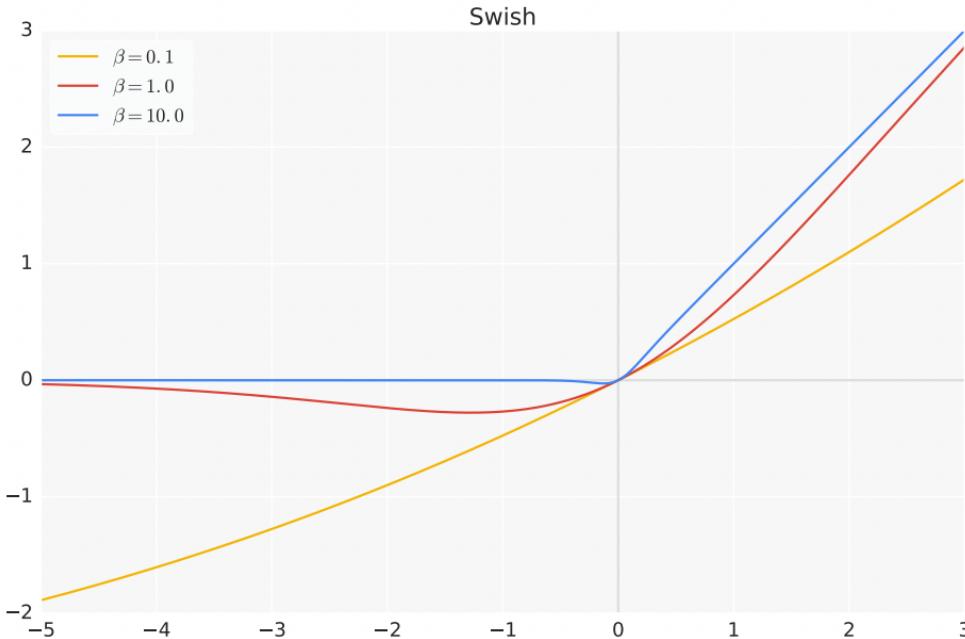


$$h(x) = x \cdot \text{sigmoid}(\beta x)$$

$$h'(x) = \beta h(x) + \text{sigmoid}(\beta x)(1 - \beta h(x))$$

# Deep neural network

- Swish激活函数

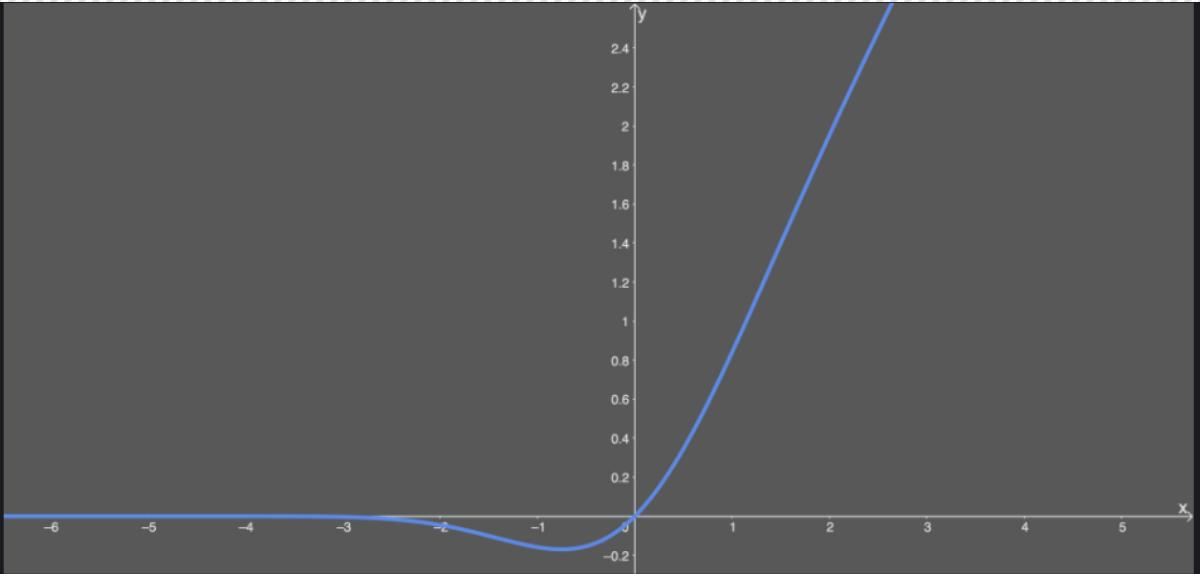


Pros: 在网络较深、batch size较大时效果要比ReLU强。

Cons: 不稳定，在不同任务上效果不同。

# Deep neural network

- GELU激活函数

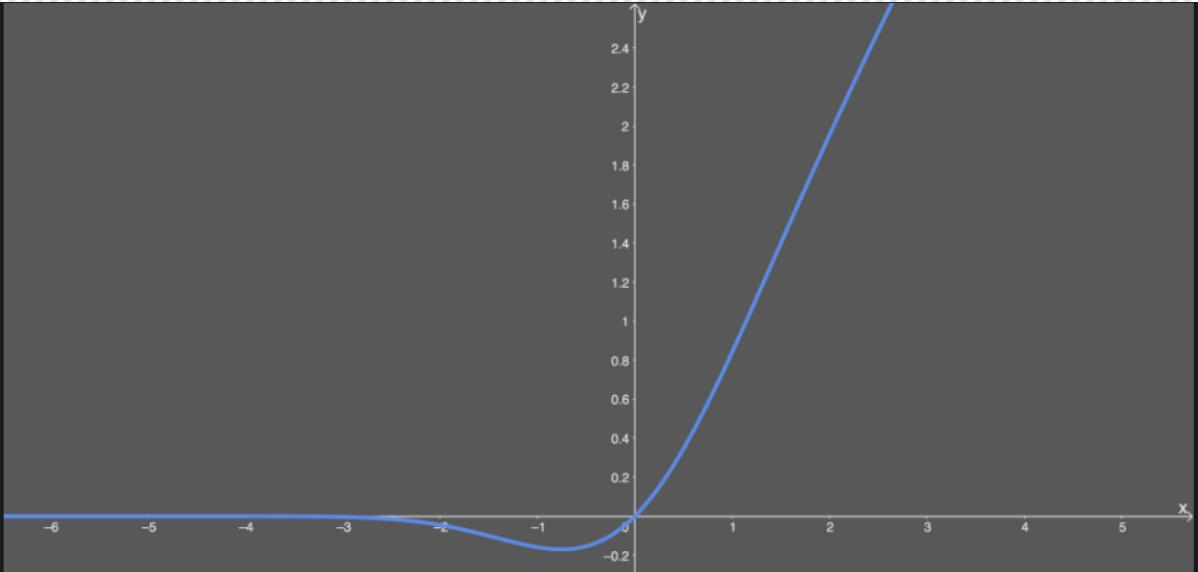


$$h(x) = 0.5x \left( 1 + \tanh \left( \sqrt{2/\pi} (x + 0.044715x^3) \right) \right)$$

$$\begin{aligned} h'(x) &= 0.5 \tanh(0.0356774x^3 + 0.797885x) \\ &\quad + (0.0535161x^3 + 0.398942x) \operatorname{sech}^2(0.0356774x^3 + 0.797885x) + 0.5 \end{aligned}$$

# Deep neural network

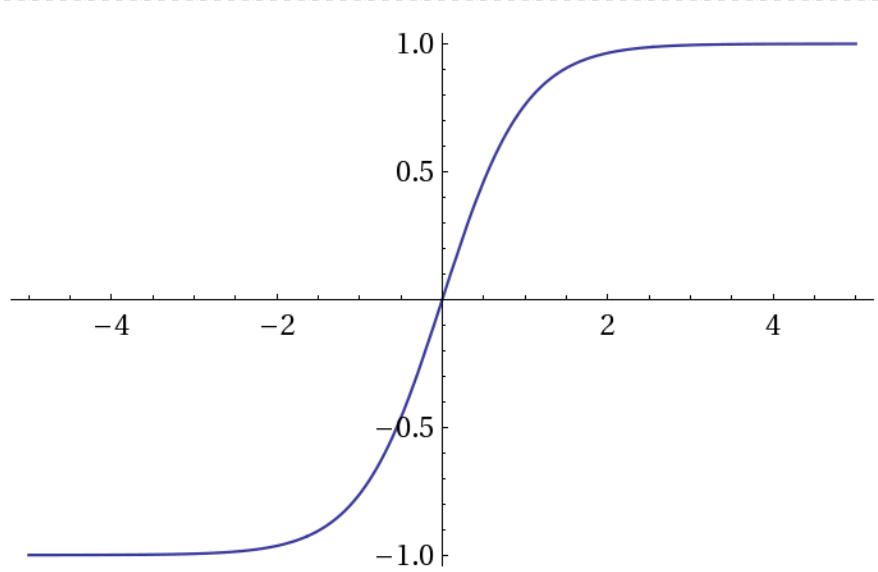
- GELU激活函数



Pros: GELU激活函数可以避免梯度消失问题，目前在Transformer模型中使用较多。

# Deep neural network

- 思考：Tanh激活函数



Tanh激活函数的导函数？

Tanh激活函数的性质如何？



# Deep neural network

## • 激活函数使用建议

- 一般情况下避免使用Sigmoid、Tanh等激活函数。
- 合理使用ReLU激活函数构建网络模型。
- 尝试使用Leaky ReLU、ELU、SELU、Swish等激活函数进一步提升模型性能。



# Deep neural network

- **Loss function(损失函数)**

- 神经网络中的损失函数用来衡量网络模型输出值与真实值之间的差异，损失函数值越小，网络的输出结果越接近真实值。损失函数的优劣会对模型的性能产生影响，而不同的任务应该使用不同的损失函数。

# Deep neural network

- Regression loss function(回归的损失函数)



价格预测



图像超分辨

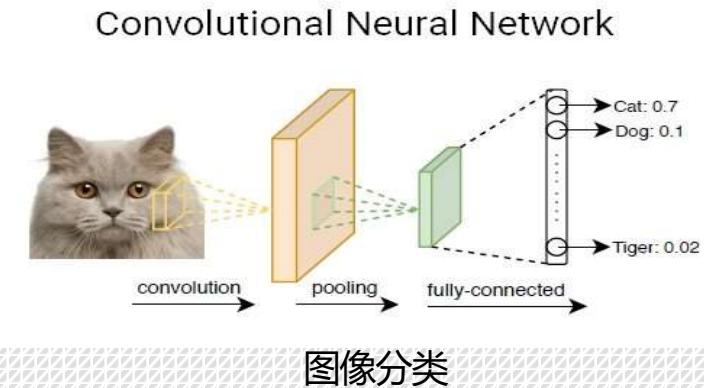
对于价格预测、图像超分辨等回归类型的任务来说，常常使用绝对值损失（L1 Loss）、均方误差损失（MSE Loss or L2 Loss）等损失函数。其中L2 Loss表达式为：

$$L(y, \hat{y}) = \frac{1}{2} (y_i - \hat{y}_i)^2$$

其中 $N$ 为训练样本数， $y_i$ 为真实值， $\hat{y}_i$ 为预测值。

# Deep neural network

## • Classification loss function ( 分类任务常用损失函数 )



图像分类

对于图像分类、图像语义分割等分类类型的任务来说，常使用交叉熵损失函数（Cross Entropy Loss）、Focal Loss等，其中交叉熵损失为： $L(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{c=1}^C y_c \log \hat{y}_c$

其中 $N$ 为训练样本数， $C$ 为样本类别数， $\mathbf{y}$ 为one-hot向量，代表样本所属真实类别， $\hat{\mathbf{y}}$ 为预测结果， $\hat{y}_c$ 为 $\hat{\mathbf{y}}$ 的第 $c$ 个元素，是第 $c$ 个类别的概率，并且 $\sum_{c=0}^C \hat{y}_c = 1$ 。



# Deep neural network

## • 思考：损失函数

- 对于交叉熵损失来说，是否存在一种情况：分类正确时的损失函数值要比分类错误时的损失函数值更高？



# Deep neural network

- 思考：损失函数

为什么会出现  
这种情况？

什么时候更容  
易出现这种情  
况？

y	$\hat{y}_1$	$\hat{y}_2$
1	0.3	0.4
0	0.2	0
0	0.2	0.1
0	0.2	0
0	0.1	0.5

$$L_{CE} = 1.20397$$

$$L_{CE} = 0.91629$$



# Deep neural network

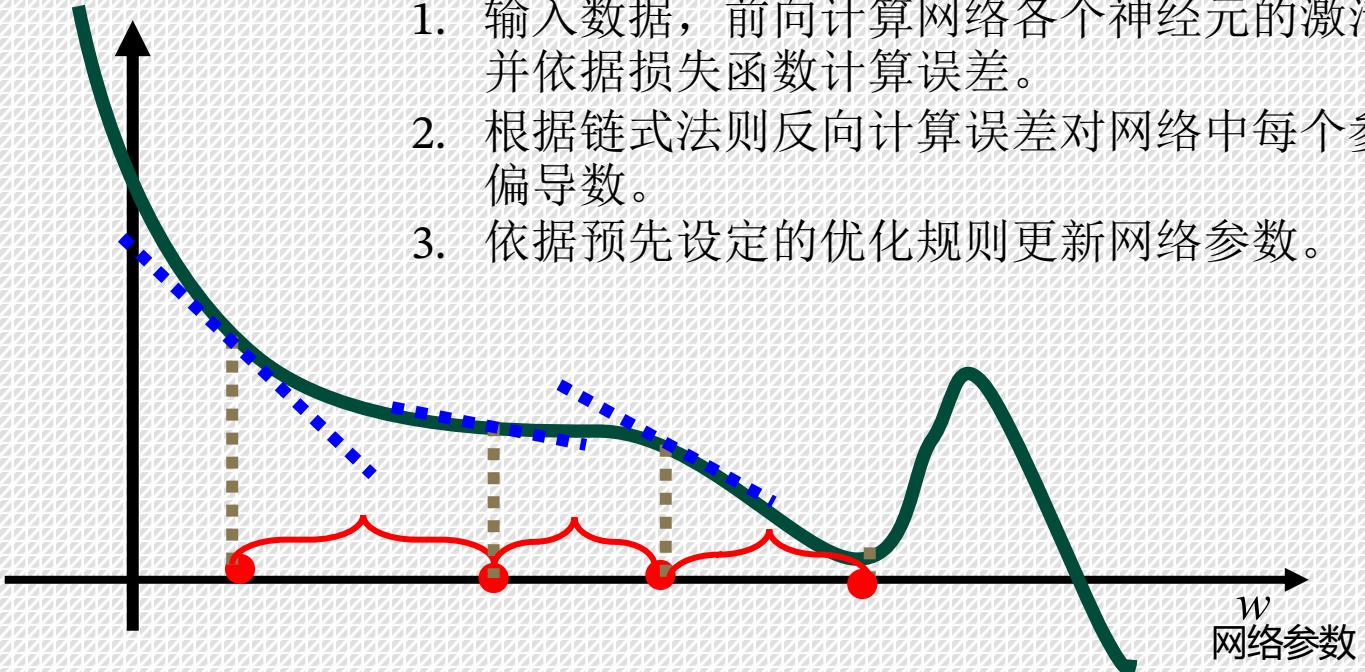
## • 参数初始化(initial weights)

- 在构建好网络模型并选择合适的损失函数后，我们还需要对模型的参数进行初始化，好的参数初始化方法有利于模型的性能。
- 一般来说模型参数需要非对称地初始化，初始化参数完全相同会导致神经元不具有区分性，不利于提取特征；参数既不能过大，也不能过小，过大或过小会造成梯度消失的现象，不利于参数更新；因此为模型参数选择一个合理的随机初始化区间十分重要。
- 常用的随机初始化方法包括Xavier初始化方法、He初始化方法等。在实际应用中，网络模型常选用ReLU激活函数，因此推荐使用He初始化方法。

# Deep neural network

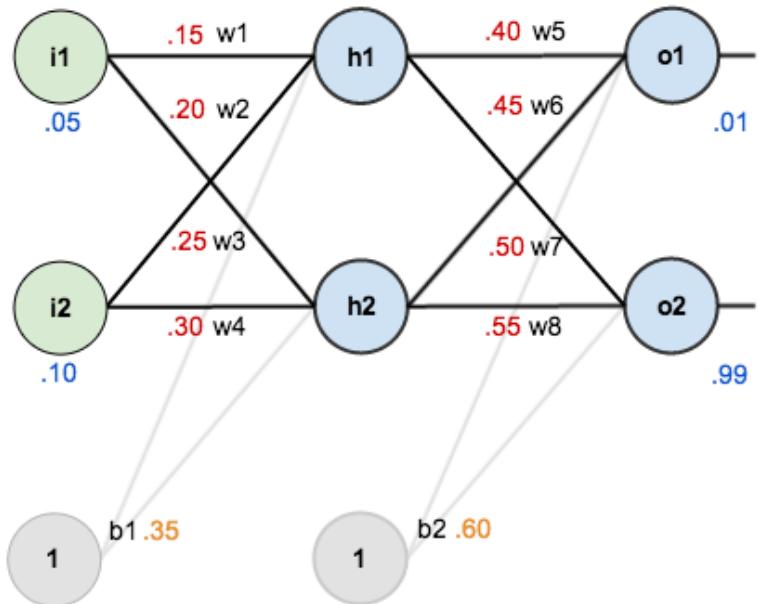
## • 反向传播算法 ( Backpropagation )

1. 输入数据，前向计算网络各个神经元的激活值，并依据损失函数计算误差。
2. 根据链式法则反向计算误差对网络中每个参数的偏导数。
3. 依据预先设定的优化规则更新网络参数。



# Deep neural network

- Backpropagation(反向传播算法)



# Deep neural network

1. 前向计算：

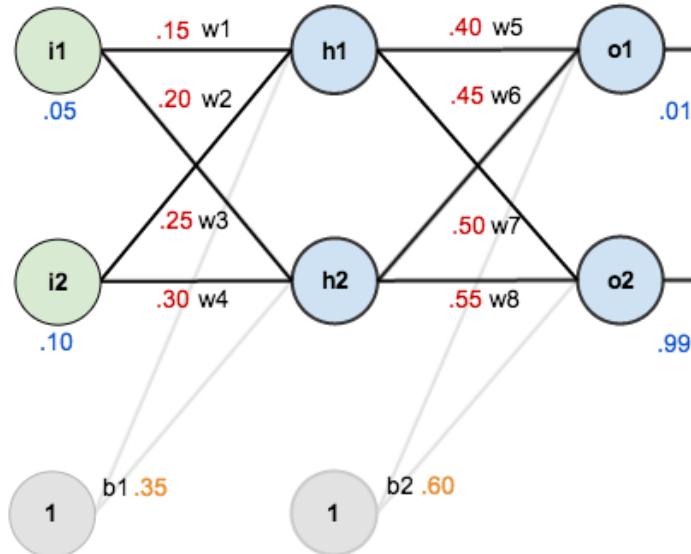
$$\begin{aligned}net_{h1} &= w_1 * i_1 + w_2 * i_2 + b_1 \\&= 0.15 * 0.05 + 0.2 * 0.1 + 0.35 \\&= 0.3775\end{aligned}$$

经过Sigmoid激活函数后：

$$out_{h1} = \frac{1}{1 + e^{-net_{h1}}} = \frac{1}{1 + e^{-0.3775}} = 0.593269992$$

同理：

$$out_{h2} = 0.596884378 \quad out_{o1} = 0.75136507 \quad out_{o2} = 0.772928465$$

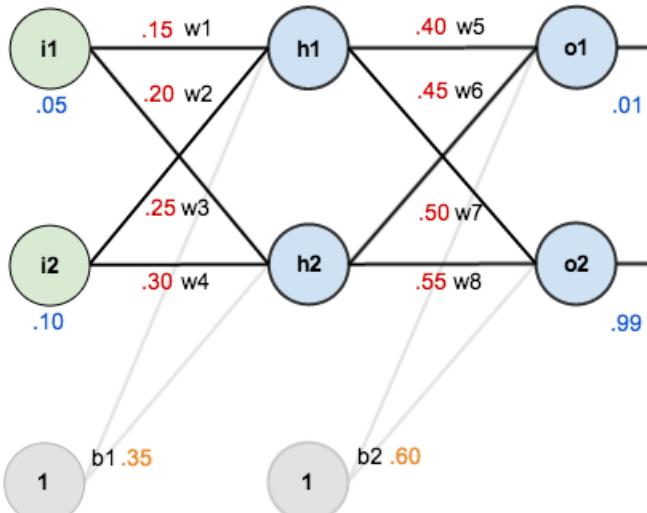


# Deep neural network

## 1. 前向计算：

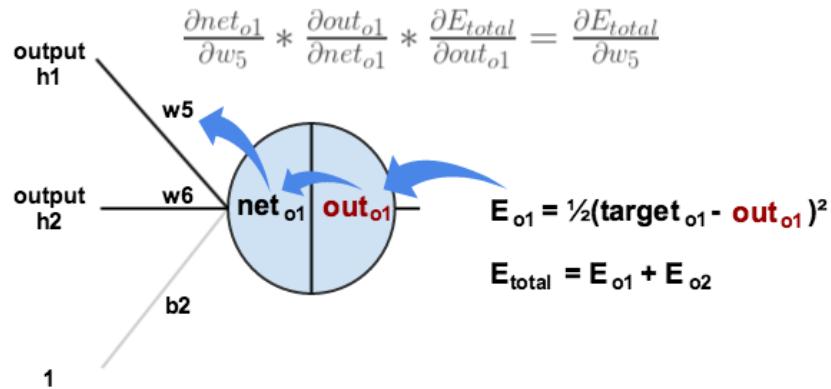
根据MSE Loss计算误差，由于 $o_1$ 和 $o_2$ 的预期输出值分别0.01和0.99，则：

$$E_{total} = E_{o1} + E_{o2} = \frac{1}{2}(0.01 - 0.75136507)^2 + \frac{1}{2}(0.99 - 0.772928465)^2 \\ = 0.274811083 + 0.023560026 = 0.298371109$$



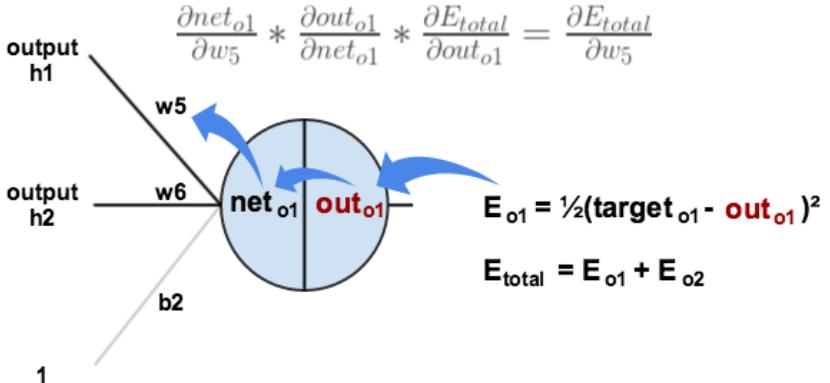
# Deep neural network

- 2. 反向计算：
- 以误差 $E_{total}$ 对参数 $w_5$ 的偏
- 导数为例：
- $\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$
- 其中：
- $\frac{\partial E_{total}}{\partial out_{o1}} = -(target_{o1} - out_{o1}) = -(0.01 - 0.75136507) = 0.74136507$
- $\frac{\partial out_{o1}}{\partial net_{o1}} = out_{o1} * (1 - out_{o1}) = 0.75136507 * (1 - 0.74136507) = 0.186815602$
- $\frac{\partial net_{o1}}{\partial w_5} = out_{h1} = 0.593269992$
- 这样  $\frac{\partial E_{total}}{\partial w_5} = 0.74136507 * 0.186815602 * 0.593269992 = 0.082167041$



# Deep neural network

- 3.参数更新：
- 在计算完误差相对于参数的偏导数后，就可以根据优化规则对参数进行更新，在这里使用梯度下降算法，其中 $\eta$ 为学习率，假设 $\eta = 0.5$ ，则 $w_5$ 更新为：
- $w_5 = w_5 - \eta \frac{\partial E_{total}}{\partial w_5} = 0.4 - 0.5 * 0.082167041 = 0.35891648$





# Deep neural network

## • 优化规则

• 批量梯度下降算法 ( batch gradient descent , BGD ) 使用整个数据集计算梯度，因此计算缓慢，且当数据集过大而无法装入内存时便无法使用：

$$\cdot w_t = w_{t-1} - \eta \nabla_w L(w)$$

• 随机梯度下降 ( Stochastic gradient descent , SGD ) 与之相反，一次仅使用一个样本计算梯度：

$$\cdot w_t = w_{t-1} - \eta \nabla_w L(w; x^{(i)}; y^{(i)}) \quad \text{SGD计算更快，但容易陷入局部最优。}$$

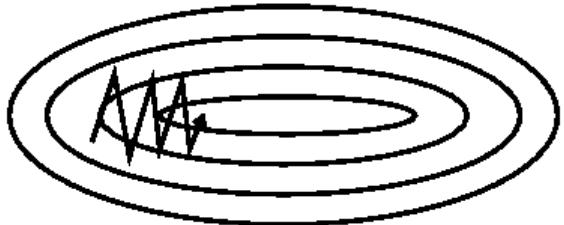
• Mini-Batch梯度下降 ( Mini-Batch gradient descent, MBGD ) 综合了以上两种算法，一次使用数个样本 ( 批量 ) 计算梯度：

$$\cdot w_t = w_{t-1} - \eta \nabla_w L(w; x^{(i:i+n)}; y^{(i:i+n)})$$

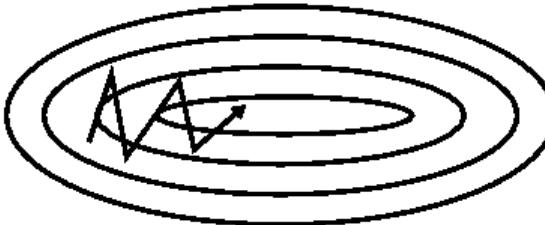
• MBGD一个批量 ( batch ) 使用的样本个数称为批量大小 ( batch size )，通常为2的幂；使用所有训练样本训练一次成为一个回合 ( epoch )。可以看出批量大小、学习率等会影响梯度下降算法。在实际使用中通常使用的是MBGD及其变种。此外，在许多文章中使用“SGD”的说法，但实际上指的是Mini-Batch梯度下降。接下来介绍MBGD的变种算法。

# Deep neural network

- 优化规则



SGD without Momentum



SGD with Momentum

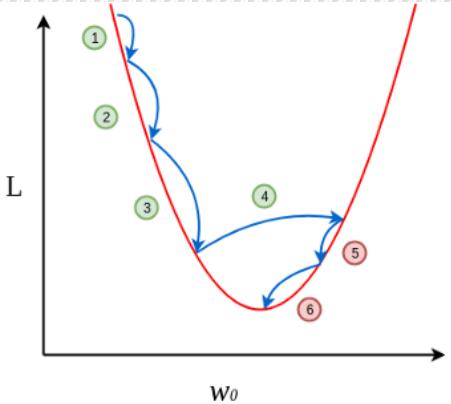
Momentum SGD:

$$v_t = \gamma v_{t-1} + \eta \nabla_w L(w_{t-1})$$

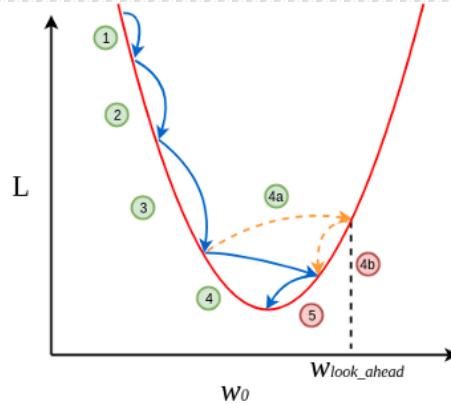
$$w_t = w_{t-1} - v_t$$

# Deep neural network

- 优化规则



(a) Momentum-Based Gradient Descent



(b) Nesterov Accelerated Gradient Descent

$$\text{Green circle} \Rightarrow \frac{\partial L}{\partial w_0} = \frac{\text{Negative}(-)}{\text{Positive}(+)} \quad \text{Red circle} \Rightarrow \frac{\partial L}{\partial w_0} = \frac{\text{Negative}(-)}{\text{Negative}(-)}$$

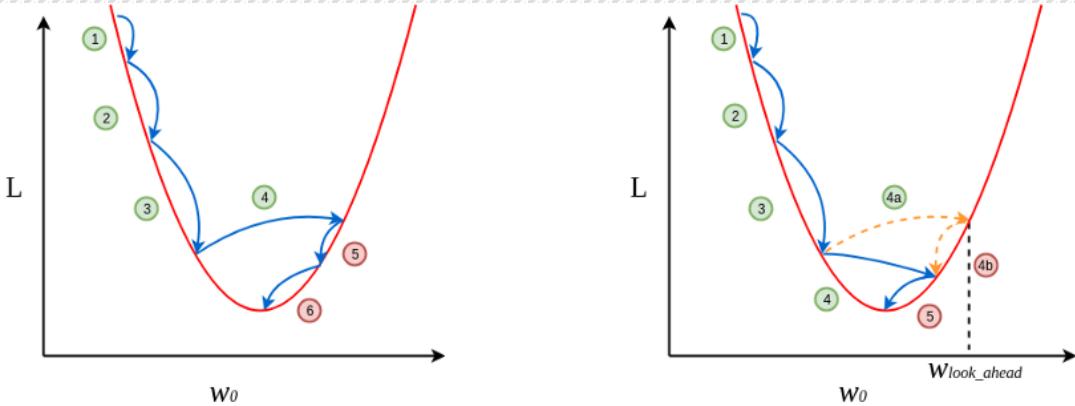
Nesterov accelerated gradient(NAG) :

$$v_t = \gamma v_{t-1} + \eta \nabla_w L(w - \gamma v_{t-1})$$

$$w_t = w_{t-1} - v_t$$

# Deep learning neural network

- 优化规则



(a) Momentum-Based Gradient Descent

(b) Nesterov Accelerated Gradient Descent

$$\text{Green circle} \Rightarrow \frac{\partial L}{\partial w_0} = \frac{\text{Negative}(-)}{\text{Positive}(+)} \quad \text{Red circle} \Rightarrow \frac{\partial L}{\partial w_0} = \frac{\text{Negative}(-)}{\text{Negative}(-)}$$

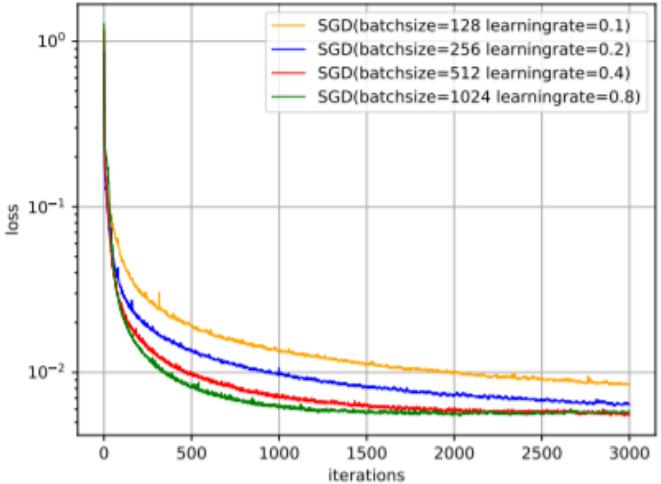
Nesterov accelerated gradient(NAG) :

$$v_t = \gamma v_{t-1} + \eta \nabla_w L(w - \gamma v_{t-1})$$

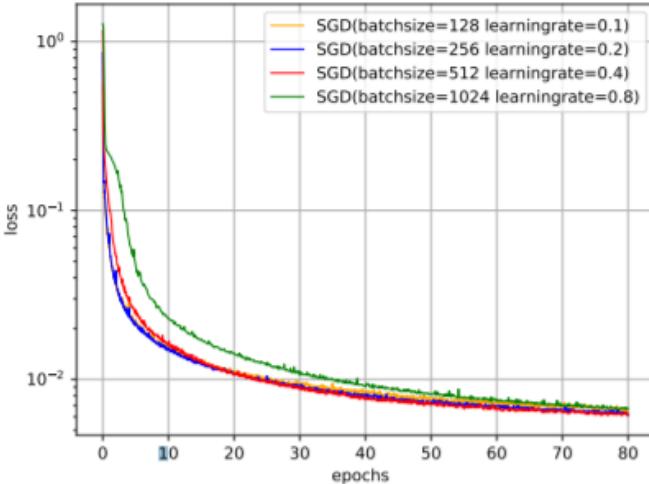
$$w_t = w_{t-1} - v_t$$

# Deep neural network

## • 批量大小的选择



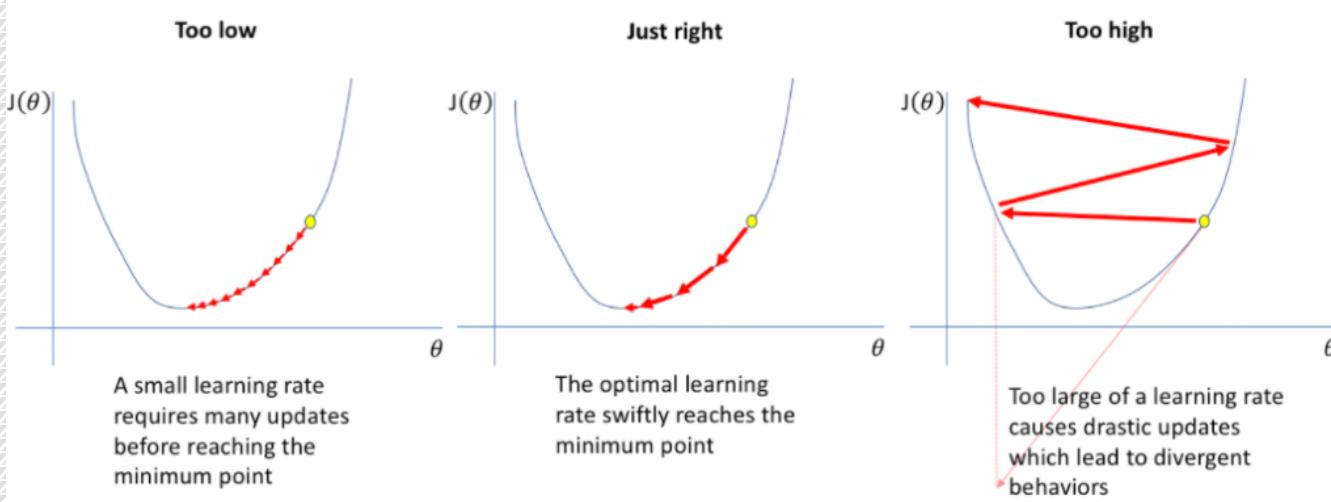
(a) 按迭代 ( Iteration ) 的损失变化



(b) 按回合 ( Epoch ) 的损失变化

# Deep neural network

## • 调节学习率





# Deep neural network

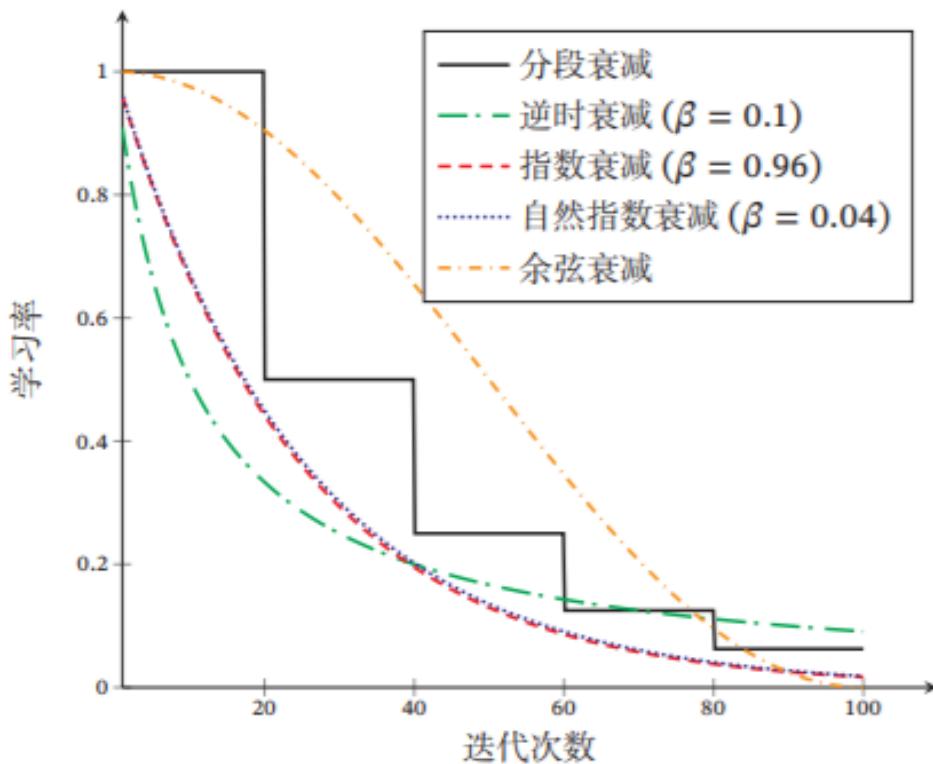
## • 调节学习率：学习率衰减

• 依照经验，在训练开始宜选择较大的学习率，以加快收敛速度，但在接近最优点时应选择较小学习率，以保证训练的稳定和收敛，这可以通过学习率衰减实现。

- 分段常数衰减：即每经过 $T_1, T_2, \dots, T_m$ 次迭代学习率衰减为原来的 $\beta_1, \beta_2, \dots, \beta_m$ 倍。
- 逆时衰减： $\eta_t = \eta_0 \frac{1}{1+\beta t}$
- 指数衰减： $\eta_t = \eta_0 \beta^t$
- 自然指数衰减： $\eta_t = \eta_0 e^{-\beta t}$
- 余弦衰减： $\eta_t = \frac{1}{2} \eta_0 (1 + \cos(\frac{t\pi}{T}))$

# Deep neural network

- 调节学习率：学习率衰减





# Deep neural network

## • 调节学习率：学习率预热

- 对于MBGD算法，若batch size较大，一般会选择较大学习率。但在训练开始时，梯度较大，较大的学习率会导致训练不稳定。因此，为了训练的稳定，可以在开始训练时使用一种成为学习率预热的手段从较小学习率开始学习，并随迭代逐步增大学习率，在到达目标学习率后再使用学习率衰减方法。
- 一个常用的学习率预热方法是逐渐预热：

$$\bullet \eta'_t = \frac{t}{T} \eta_0$$

# Deep neural network



- Adaptive Gradient(Adagrad) :
  - $v_t^w = v_{t-1}^w + (\nabla_w L(w_{t-1}))^2$
  - $w_t = w_{t-1} - \frac{\mu}{\sqrt{v_t^w + \varepsilon}} \nabla_w L(w_{t-1})$

Root Mean Squared Backpropagation(RMSProp) :

$$v_t^w = \beta v_{t-1}^w + (1 - \beta)(\nabla_w L(w_{t-1}))^2$$
$$w_t = w_{t-1} - \frac{\mu}{\sqrt{v_t^w + \varepsilon}} \nabla_w L(w_{t-1})$$



# Deep neural network

## • 优化规则

Adaptive Momentum(Adam) :

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla_w L(w_{t-1})$$

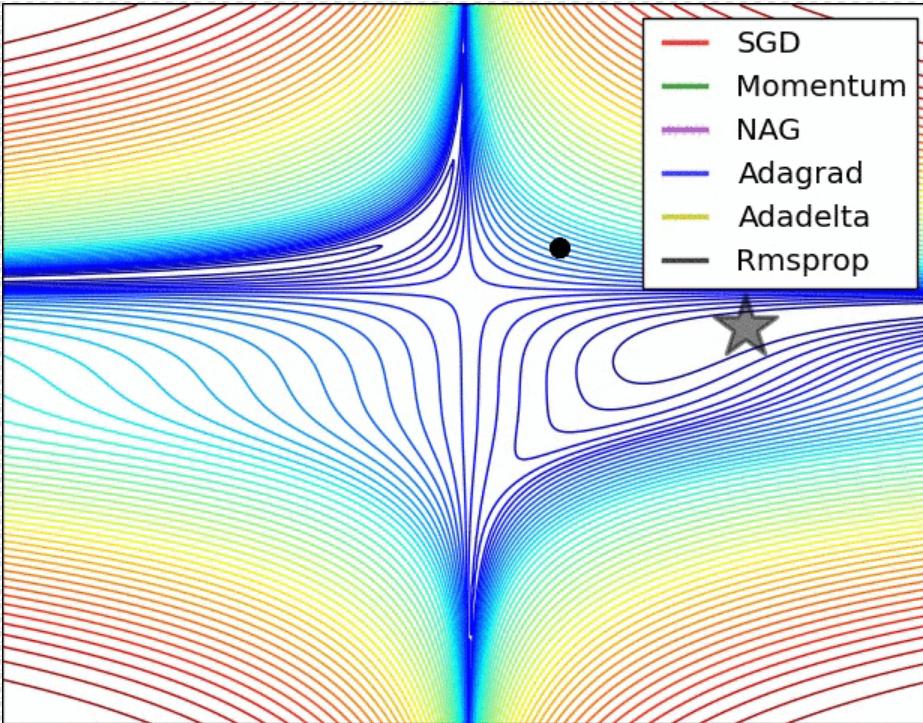
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) (\nabla_w L(w_{t-1}))^2$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1} \quad \hat{v}_t = \frac{v_t}{1 - \beta_2}$$

$$w_t = w_{t-1} - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

# Deep neural network

## • 优化规则





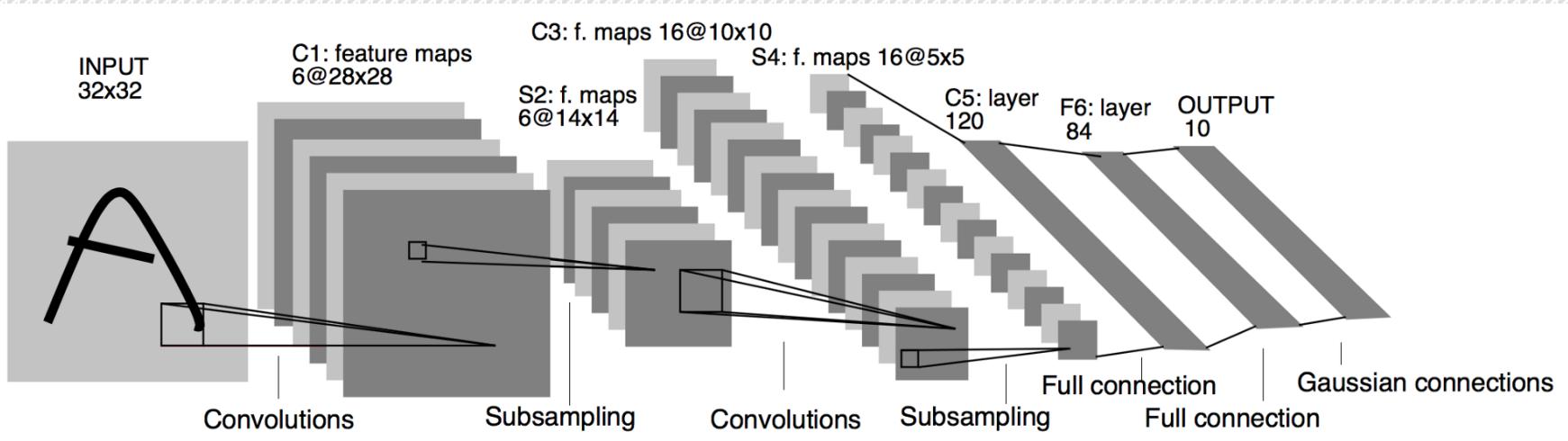
# Deep neural network

## • 选择优化规则的建议

- 对于稀疏数据，推荐使用自适应学习率方法，如RMSProp、Adam。
- 训练更为复杂的网络时使用自适应学习率方法。
- 总的来说RMSProp、AdaDelta、Adam的性能相近，但在一些情况下Adam的表现可能更好。
- 在有较好初始参数和学习率调节方案情况下可以选择SGD。

# Convolutional Neural Network

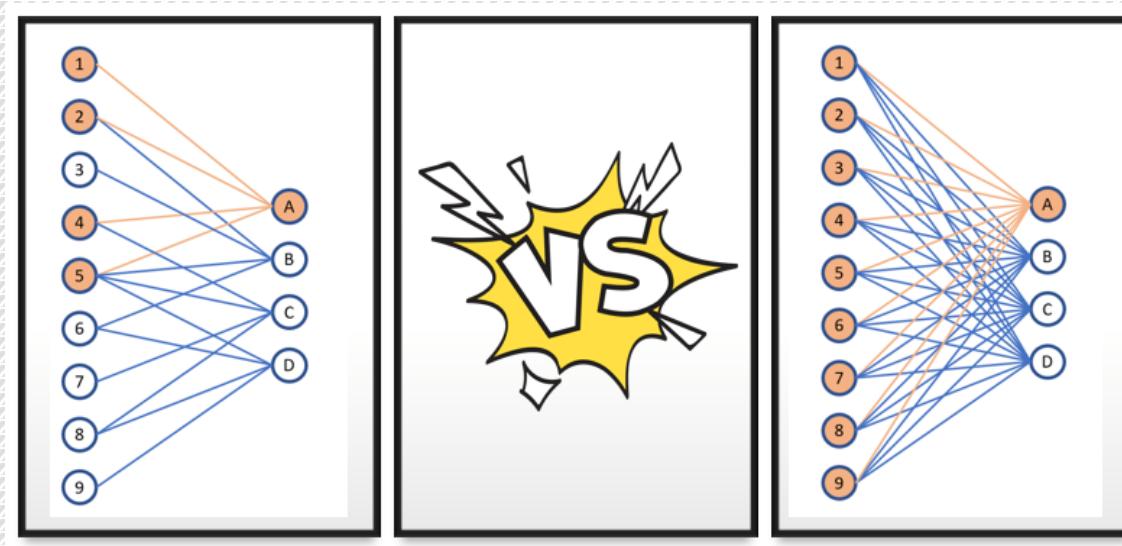
## • 卷积神经网络 ( Convolutional Neural Network , CNN )



卷积神经网络是一种常用于处理图像与视频数据的神经网络，其主要使用卷积层从图像中提取特征。

# Convolutional Neural Network

- 为什么不使用全连接层处理图像？



- 图像数据维度较高，使用全连接层处理会导致网络模型参数过多，难以训练和部署。
- 自然图像具备局部不变特性，即对图像进行平移、旋转等操作并不影响图像的语义信息而全连接层难以捕获图像的这种特性。

# Convolutional Neural Network

- Convolution operation(卷积操作)

1 <small><math>\times 1</math></small>	1 <small><math>\times 0</math></small>	1 <small><math>\times 1</math></small>	0	0
0 <small><math>\times 0</math></small>	1 <small><math>\times 1</math></small>	1 <small><math>\times 0</math></small>	1	0
0 <small><math>\times 1</math></small>	0 <small><math>\times 0</math></small>	1 <small><math>\times 1</math></small>	1	1
0	0	1	1	0
0	1	1	0	0

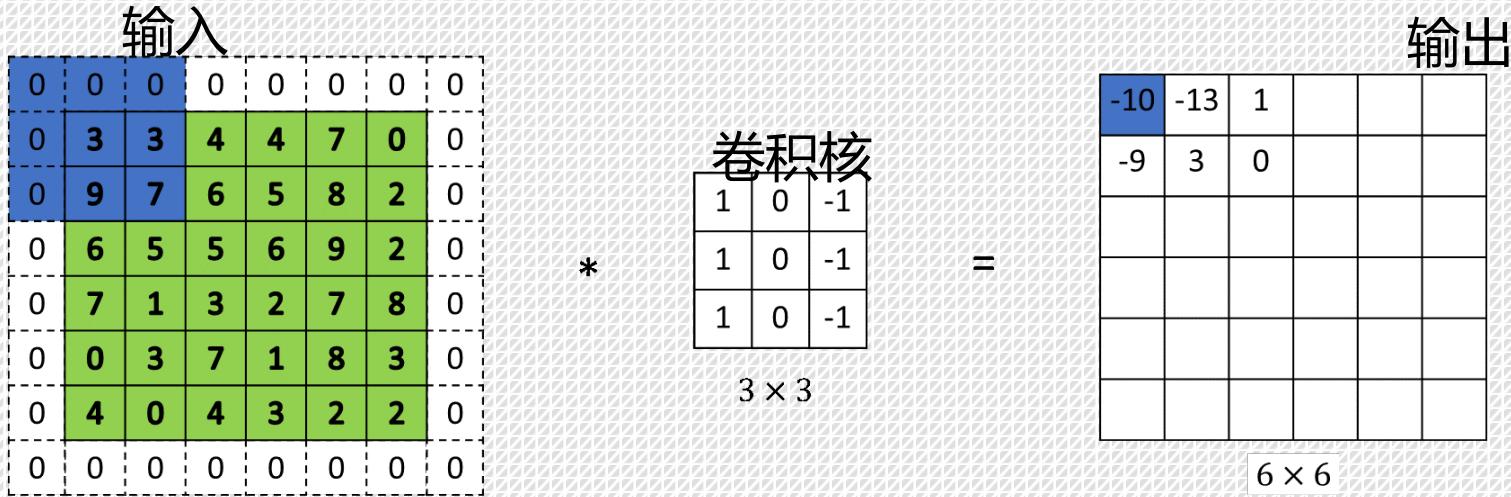
Image

4		

Convolved  
Feature

# Convolutional Neural Network

- Convolution operation(卷积操作)



特征图 ( Feature Map ) : 如上图右侧经过卷积核处理的图像即为特征图。  
 $6 \times 6 \rightarrow 8 \times 8$

卷积核 ( Convolutional Kernel ) : 如上图 , 为一大小为  $3 \times 3$  矩阵 , 与输入图像作滑动点积。

填充 ( Padding ) : 卷积会造成图像边缘像素损失 , 但有时我们希望输入与输出大小相同。为解决这一问题 , 我们在输入图像边缘填充一些像素。如上图我们在每一边额外填充了一行 “0” 。

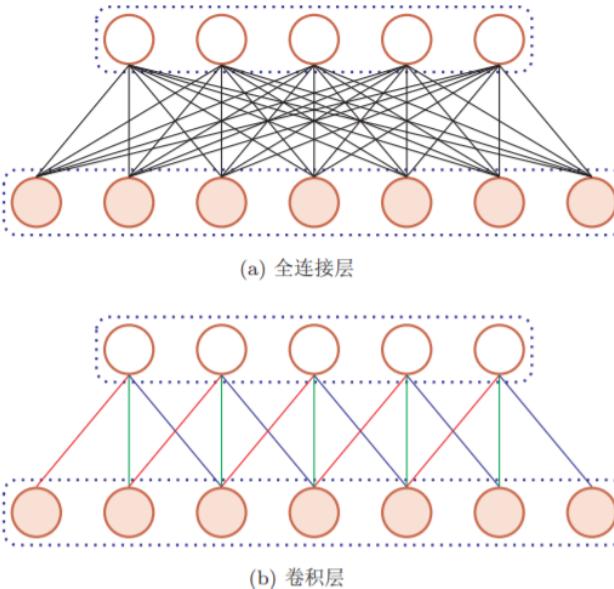
# Convolutional Neural Network

- 卷积层

Input Volume (+pad 1) (7x7x3)	Filter W0 (3x3x3)	Filter W1 (3x3x3)	Output Volume (3x3x2)
x[:, :, 0]	w0[:, :, 0]	w1[:, :, 0]	o[:, :, 0]
0 0 0 0 0 0 0 0 0 1 1 2 2 0 0 0 1 1 0 0 0	1 1 -1 -1 0 1 -1 -1 0	-1 -1 0 -1 1 0 -1 1 0	1 0 -3 -6 1 1 4 -3 1
0 1 1 0 1 0 0 0 1 0 1 1 1 0 0 0 2 0 1 0 0	w0[:, :, 1]	w1[:, :, 1]	o[:, :, 1]
0 0 0 0 0 0 0	-1 0 -1 0 0 -1 1 -1 0	1 -1 0 -1 0 -1 -1 0 0	-1 -6 -4 -2 -3 -4 -1 -3 -3
x[:, :, 1]	w0[:, :, 2]	w1[:, :, 2]	
0 0 0 0 0 0 0 0 1 1 1 2 0 0 0 0 2 1 -1 2 0	0 1 0 1 0 1 0 -1 1	-1 0 1 1 0 1 0 -1 0	
0 1 2 0 0 2 0 0 0 2 1 2 1 0 0 2 0 1 2 0 0	Bias b0 (1x1x1)	b1[:, :, 0]	
0 0 0 0 0 0 0	b0	0	
x[:, :, 2]		toggle movement	
0 0 0 0 0 0 0 0 2 0 2 0 2 0 0 0 0 1 1 2 0			
0 0 0 0 0 0 0			

# Convolutional Neural Network

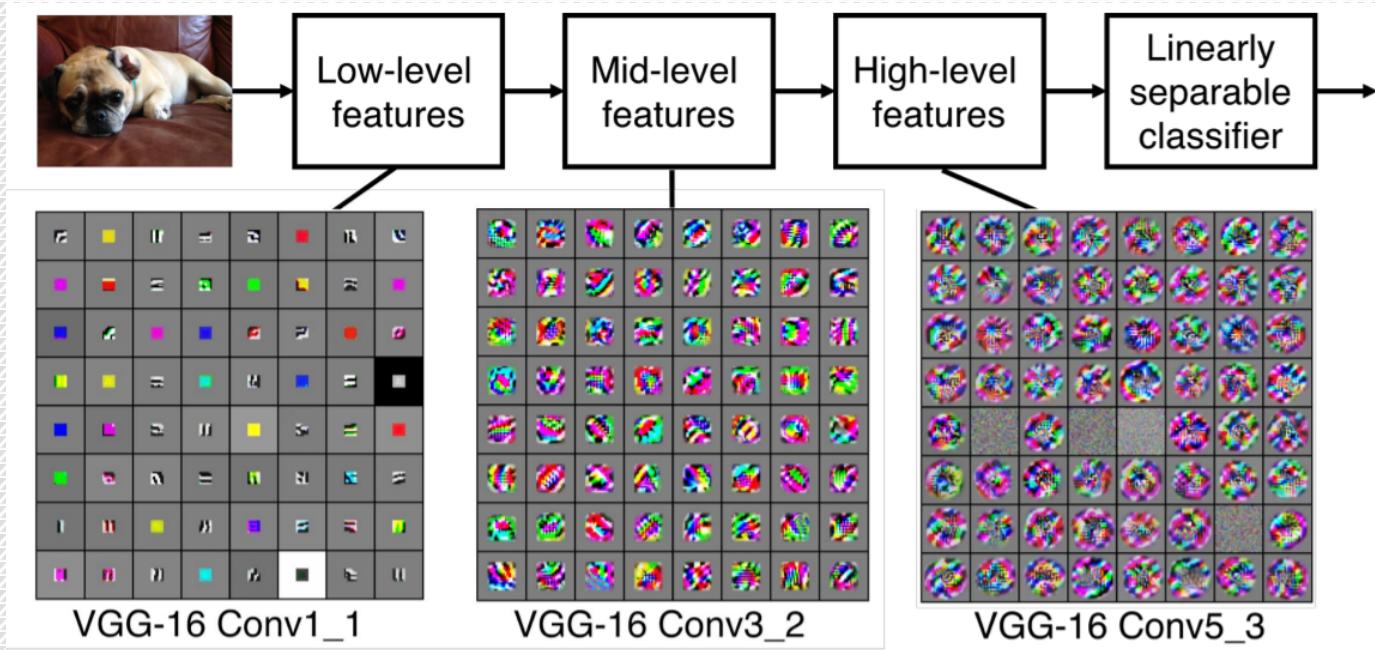
- 卷积层



局部连接：与全连接层不同，卷积层的每一个神经元只和上一层局部窗口中的神经元连接，形成一种局部连接。  
权重共享：卷积核的权重对于所有的神经元都是相同的。

# Convolutional Neural Network

- 卷积层





# Andrew NG

- <https://www.bilibili.com/video/BV1244y1S79G/>

