上海交通大学

计算机视觉

教师：赵旭

班级：AI4701

2024 春

13. 表示学习： 深度网络

内容

- 基本概念
- 网络结构
- 激活函数
- 网络训练
- 正则化

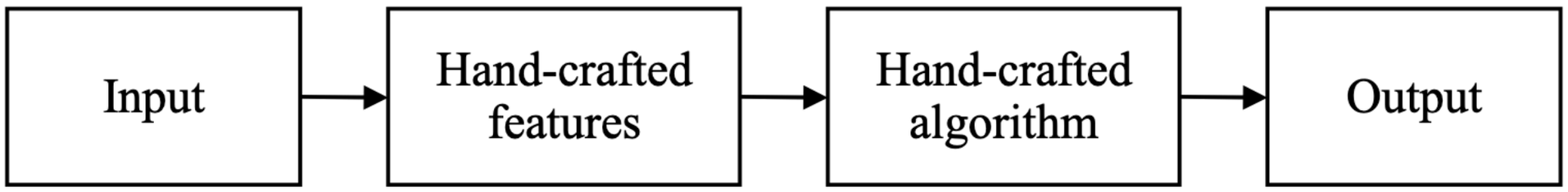
什么是学习?

- 分类：图像分割
- 回归：物体检测
- 结构化输出：图生文...

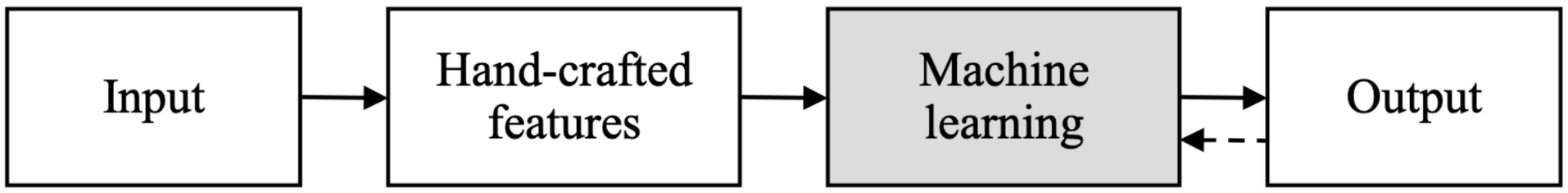
- 数据集
  - 有标签数据
  - 无标签数据
  - 混合数据

- 准确度
- 错误率
- 损失代价
- ...

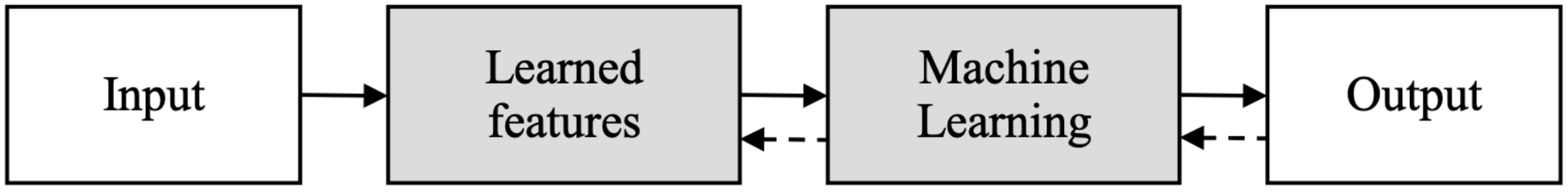
什么是表示学习?



(a) Traditional vision pipeline



(b) Classic machine learning pipeline



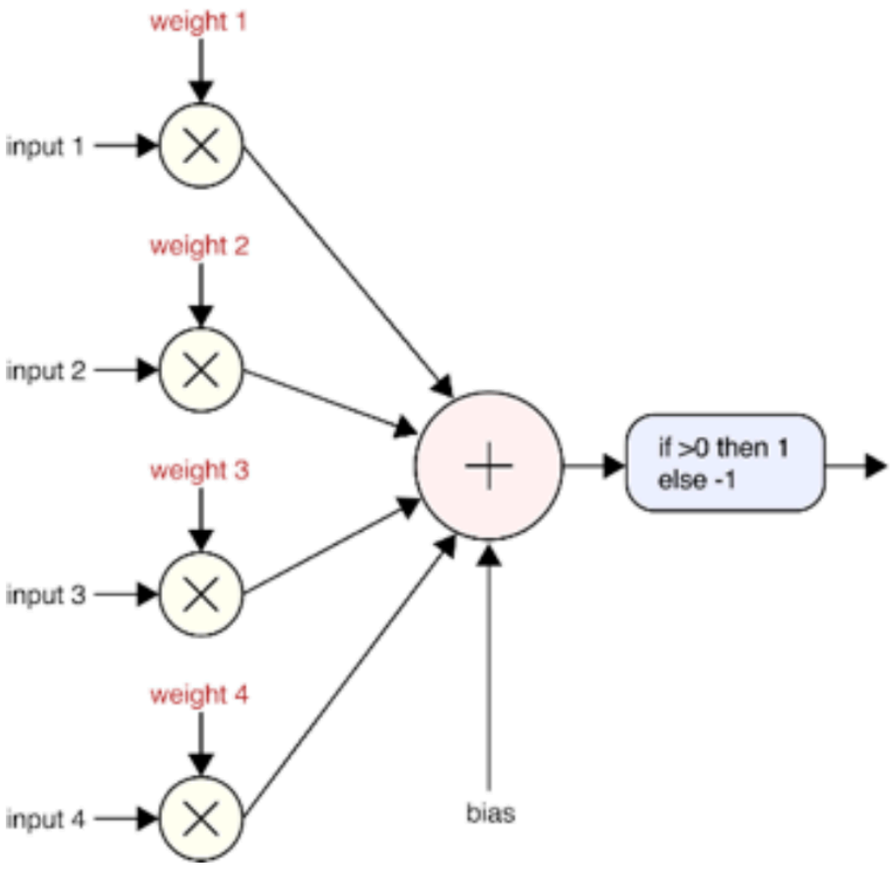
Computer Vision: Algorithms and Applications

欠拟合与过拟合

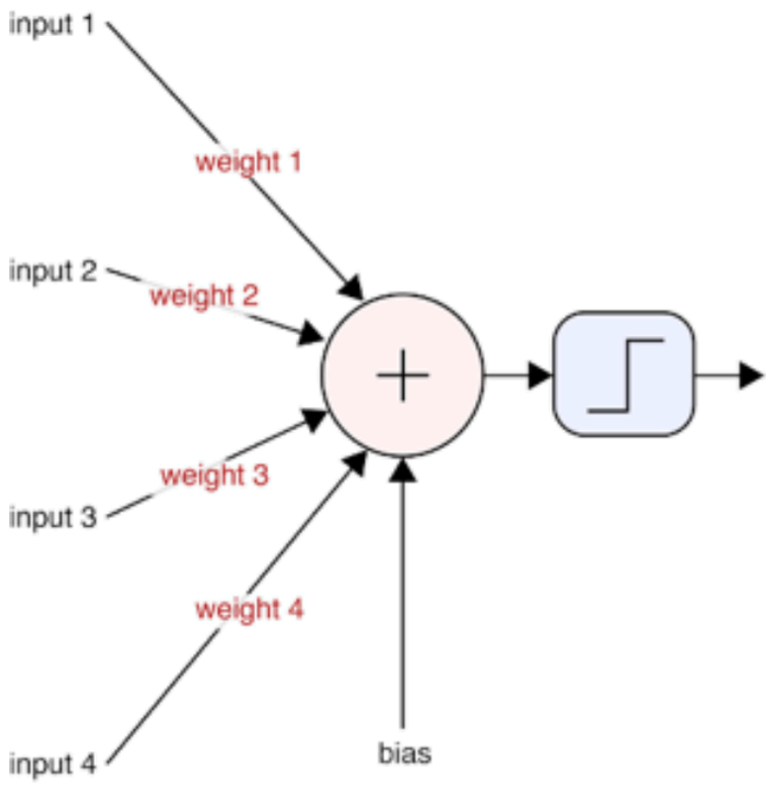
- 学习算法应该具备的能力
  - 使训练误差最小
  - 使训练和测试误差的差距尽量小
- 欠拟合：训练集上的性能不好
- 过拟合：训练集上的性能和测试集上的性能差距较大
- 模型容量：拟合不同函数的能力。欠拟合：容量不足；过拟合：容量过高，过拟合训练集数据的特性。

神经网络: 动机

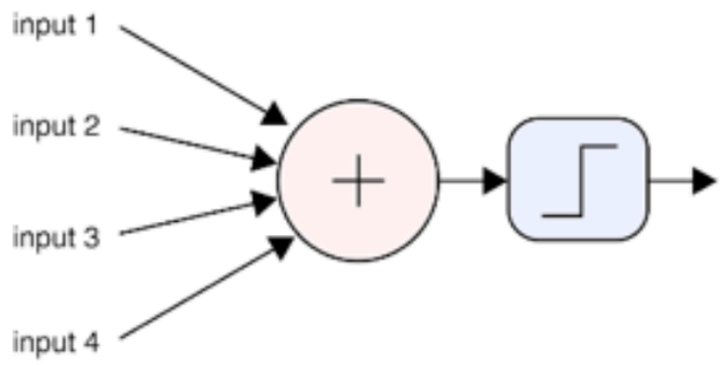
- 生物机制启发的计算模型: 实现像人一样的智能
- 帮助理解人脑如何工作
- 提供一种特别的组合形式，函数近似


weight 1
input 1
weight 2
input 2
weight 3
input 3
weight 4
input 4
bias
if >0 then 1
else -1

(a)


input 1
weight 1
input 2
weight 2
weight 3
input 3
weight 4
input 4
bias

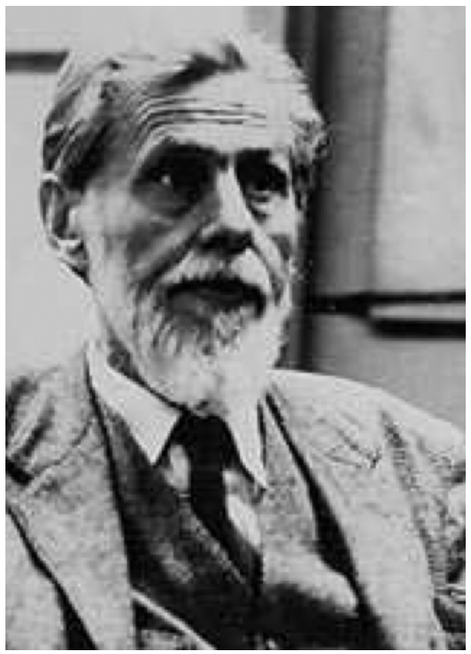
(b)


input 1
input 2
input 3
input 4

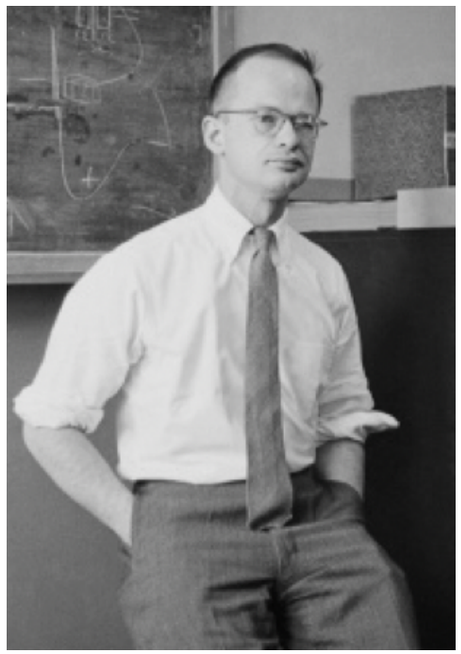
(c)

历史: 里程碑工作

- **McCulloch-Pitts 神经元 （1943)**
  - 2类分类的线性模型
  - 权值手工设定



McCulloch（1898—1969）



Pitts（1923—1969）

尼克. 人工智能简史（第2版）

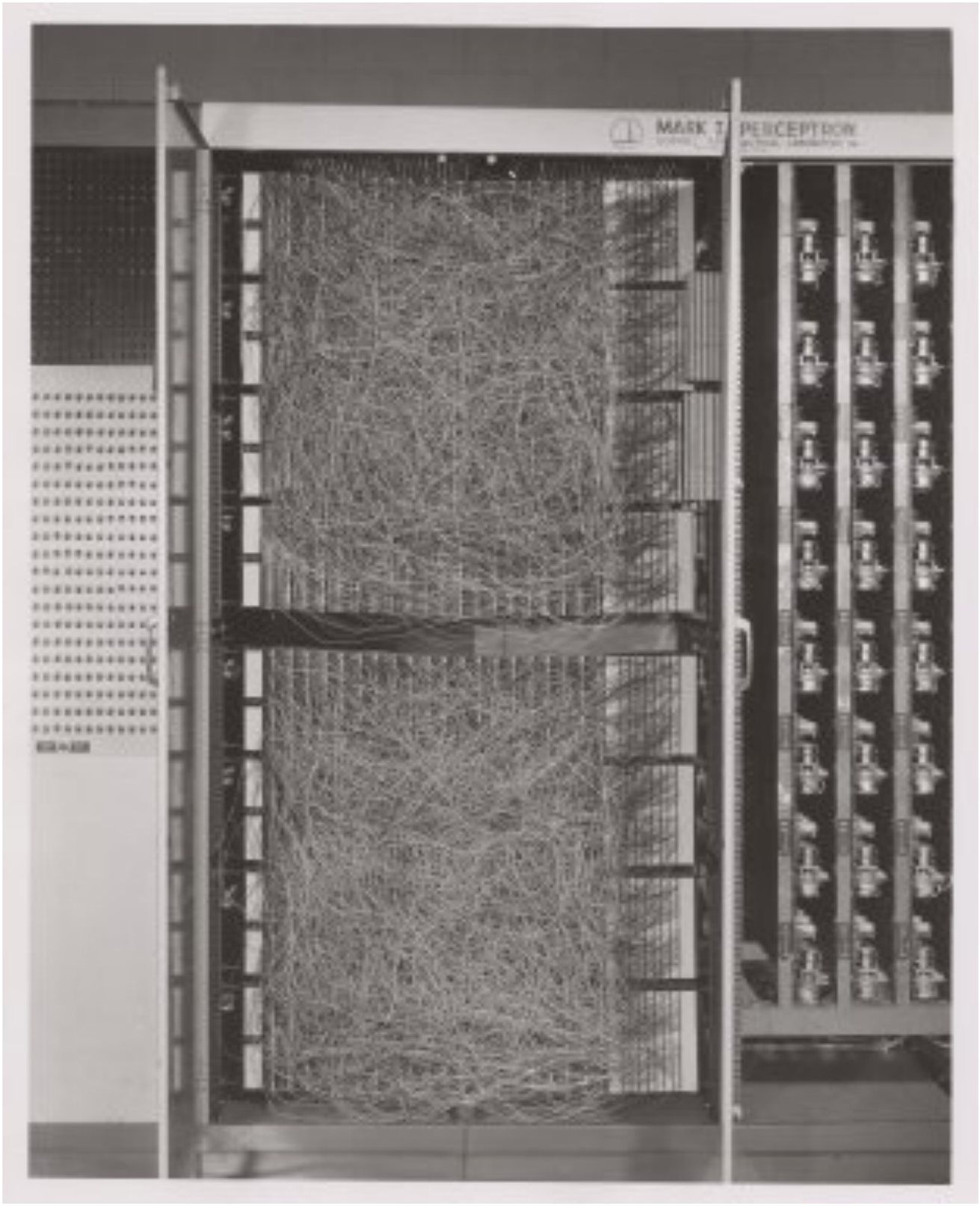
历史: 里程碑工作

- **Rosenblatt: 感知机（1958, 1962)**
  - 第一个看i学习权重的模型
  - 硬件实现：IBM-704

Frank Rosenblatt（1928—1971）

Book (1962): Principles of Neuro-dynamics: Perceptrons and the Theory of Brain

尼克. 人工智能简史（第2版）



Mark 1 Perceptron
c.1960

20x20 pixel
camera feed

Wikipedia

历史: 里程碑工作

- **ADALINE: ADAptive LINEar element (Widrow and Hoff, 1960)**
  - 随机梯度下降的特殊情况
- **Backprogagation (Paul Werbos, 1974)**
  - 通过加入额外1层求解XOR问题
- **Connectionism (1980s)**
  - "A large number of simple computational units can achieve intelligent behavior when networked together"

历史: 里程碑工作

- **Hopfield 网络 (John Hopfield, 1982)**
- **分布式表示 (Hinton, 1986)**
  - “each feature should be involved in the representation of many possible inputs”
- **LSTM (Hochreiter and Schmidhuber, 1997)**
  - 序列模型
- **Deep belief network (Hinton, 2006)**
  - 多层
  - 贪婪逐层训练算法

历史: 三次浪潮

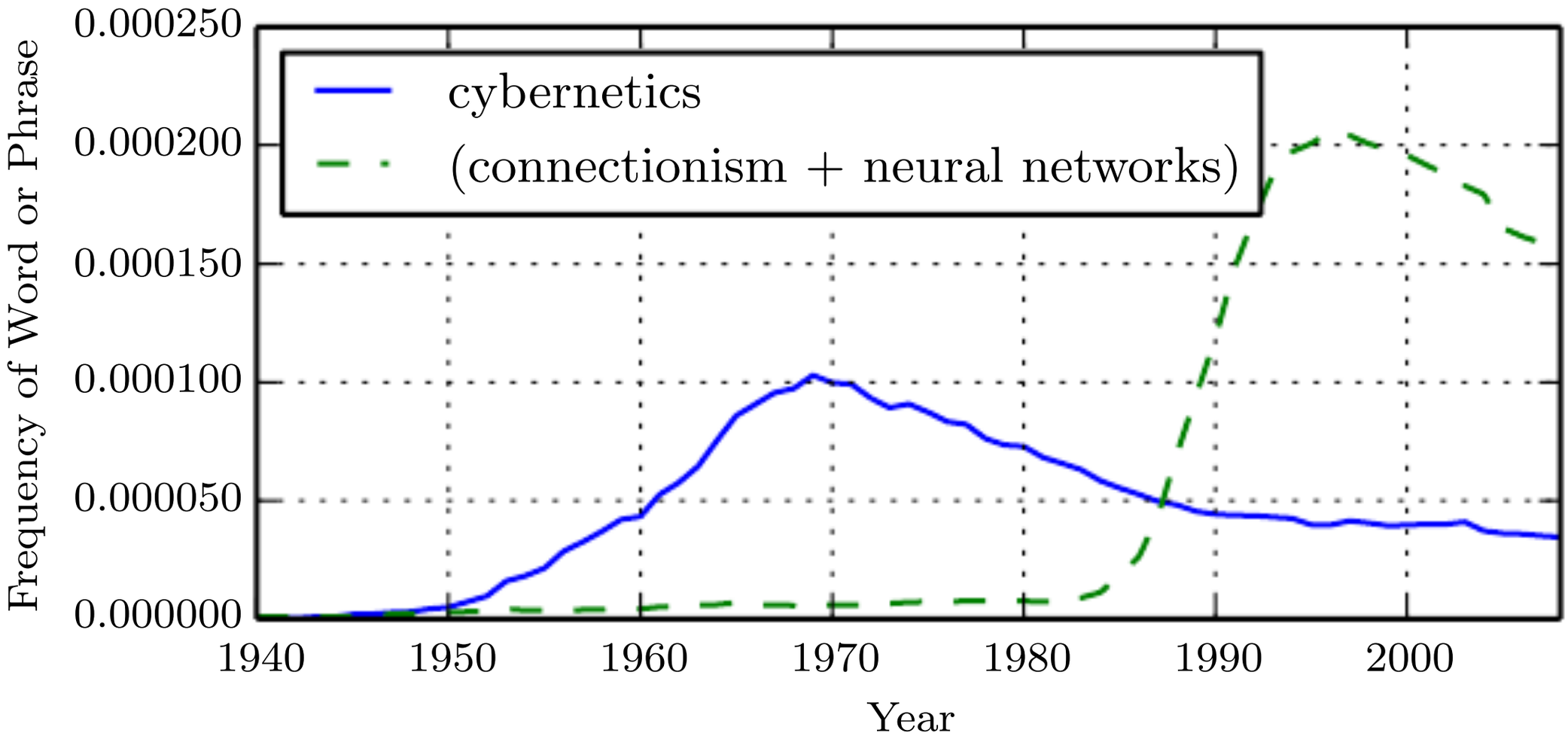


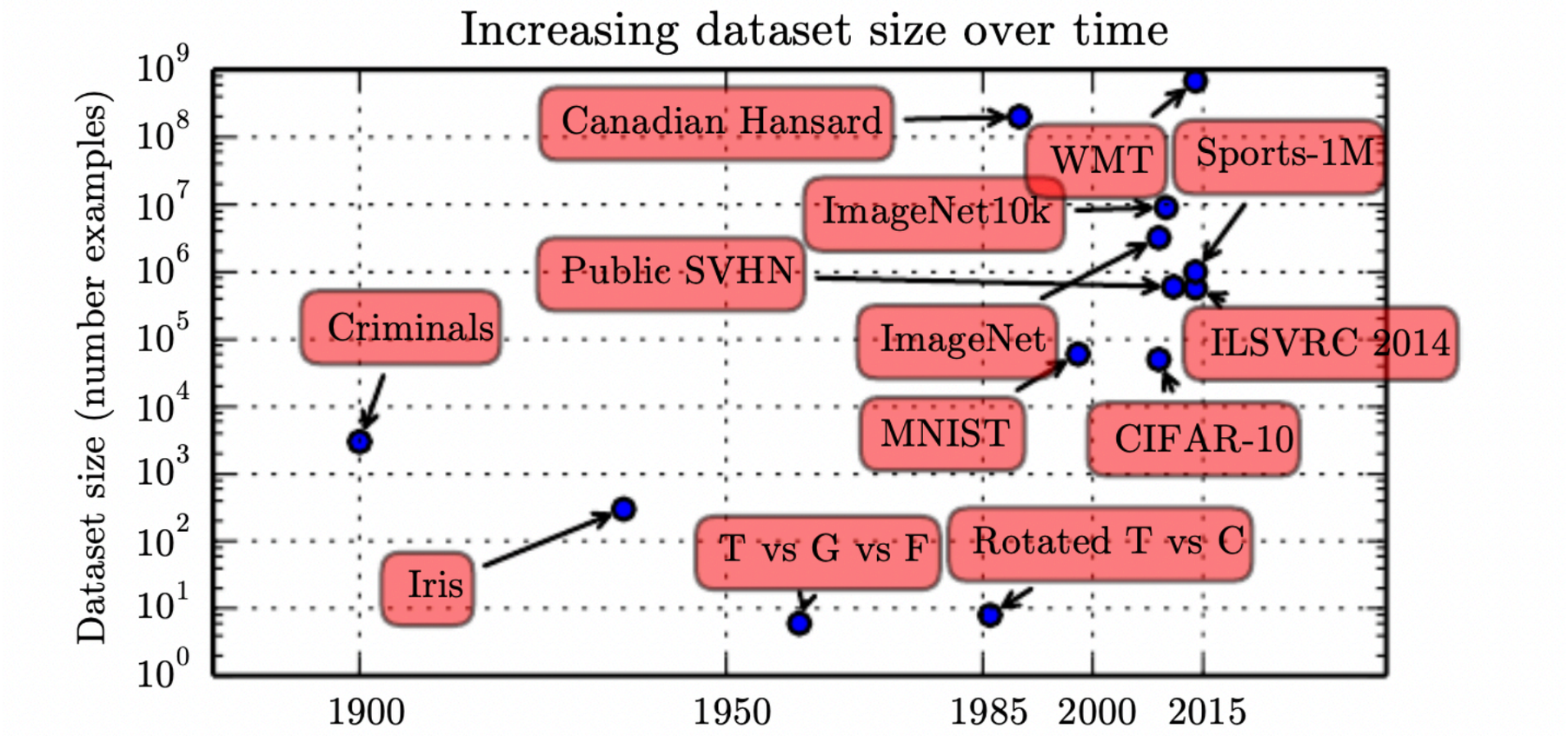
Figure 1.7: The figure shows two of the three historical waves of artificial neural nets research, as measured by the frequency of the phrases "cybernetics" and "connectionism" or "neural networks" according to Google Books (the third wave is too recent to appear). The first wave started with cybernetics in the 1940s–1960s, with the development of theories of biological learning (McCulloch and Pitts, 1943; Hebb, 1949) and implementations of the first models such as the perceptron (Rosenblatt, 1958) allowing the training of a single neuron. The second wave started with the connectionist approach of the 1980–1995 period, with back-propagation (Rumelhart *et al.*, 1986a) to train a neural network with one or two hidden layers. The current and third wave, deep learning, started around 2006 (Hinton *et al.*, 2006; Bengio *et al.*, 2007; Ranzato *et al.*, 2007a), and is just now appearing in book form as of 2016. The other two waves similarly appeared in book form much later than the corresponding scientific activity occurred.

为什么深度学习会兴起?

- **性能好**
  - ILSVR C（ImageNet Large Scale Visual Recognition Challenge）2012
  - Speech recognition & Translation (Li Deng, 2009)
  - *“However, they have not succeeded in solving the central problems in AI, such as recognizing speech or recognizing objects. The development of deep learning was motivated in part by the failure of traditional algorithms to generalize well on such AI tasks.” - Deep Learning, Ian Goodfellow et. al*

为什么深度学习会兴起?

- **数据大**



Deep Learning, Ian Goodfellow et. al

为什么深度学习会兴起?

- **硬件强**
  - GPU 的大规模商用
  - “Every break through of Computer Vision needs support from Computer’s break through .”
- **开源**
  - Flexible software platforms with automatic differentiation such as Theano, Torch Caffe, TensorFlow and PyTorch.

为什么要“深”？

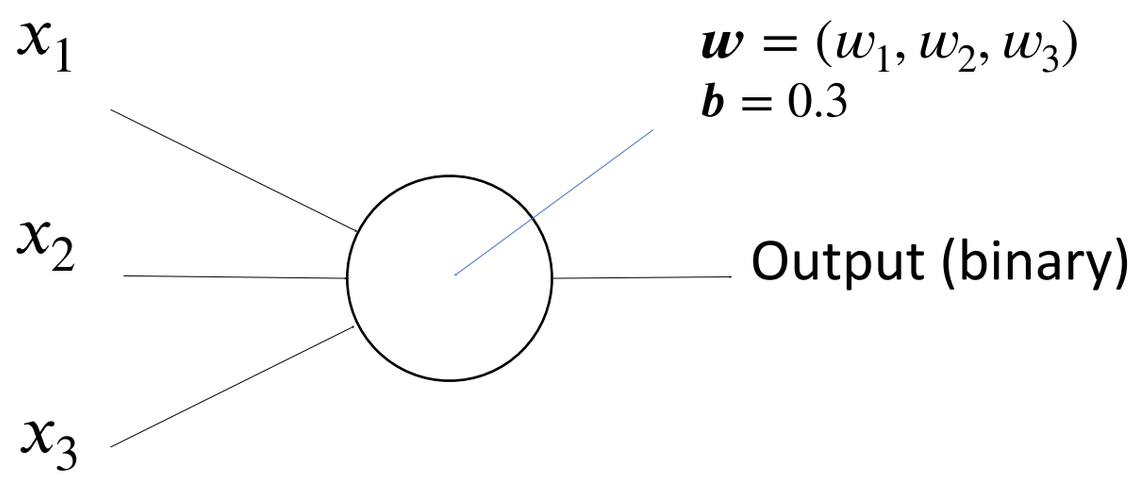
- 更多的参数？不完全是，因为相同的参数量，深层网络比浅层网络的**泛化性**更好。
- 特殊形式的**组合性**：一层的特征以各种不同的方式组合，形成下一层更抽象的特征。

Deep learning for AI, *Communications of the ACM, 2021, Yoshua Bengio Yann Lecun, Geoffrey Hinton*

神经网络-基本结构

Perceptron (Rosenblatt c.1960)

线性分类器：权重向量 w 和 偏差项 b



$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

$$w \cdot x \equiv \sum_j w_j x_j$$

例：图像二分类

每个像素为一个输入：一副28 x 28 的图像可向量化为:

x： 1 x 784

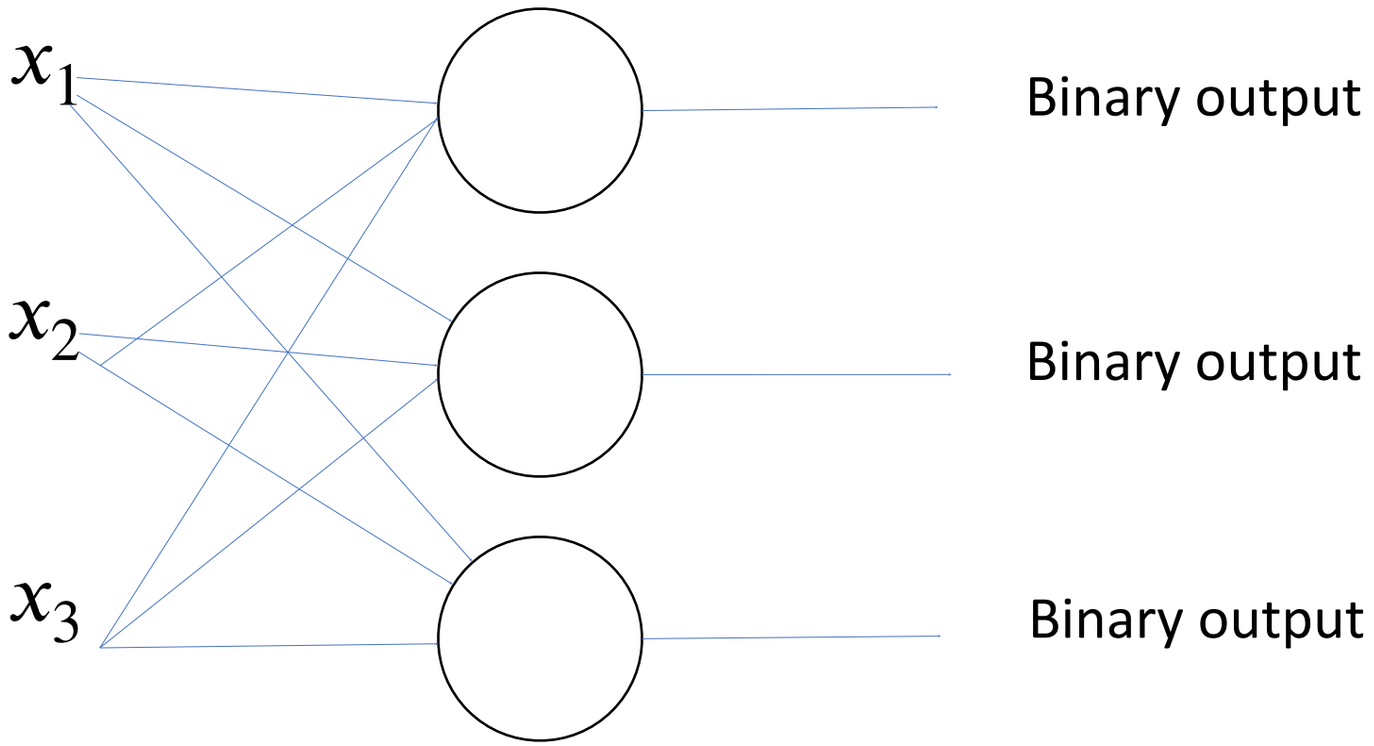
w：784 x 1 （权值向量）

b：标量 (每个神经元)

输出 = **xw** + b -> (1x784) x (784x1) + b = (1x1)+b

多类分类

加入更多的神经元



例：图像多分类

一张 28 x 28 图像：**x** = 1 x 784

W： 10 x 784 **矩阵**，10类分类

b： 1 x 10 向量（每个神经元1个偏差项）

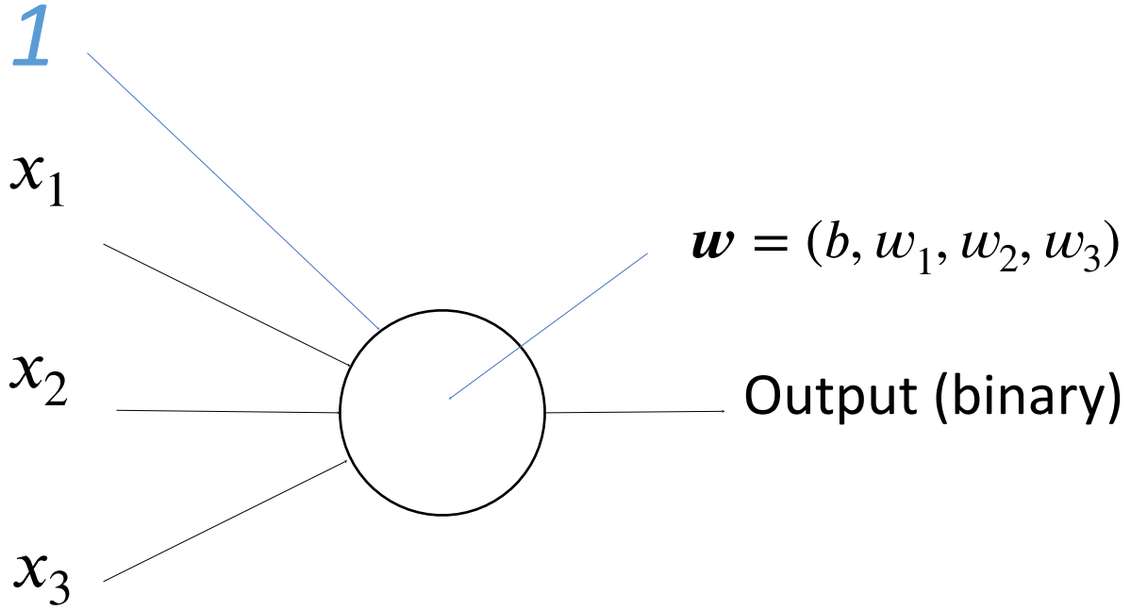
输出： **xW** + **b** -> (1x784) x (784 x 10) + **b**

-> (1 x 10) + (1 x 10) = output vector

偏差项整合到权值中

仅保留“乘”操作

- “假”特征：值为1，对应偏差项
- 权值中加入另外一维：偏差值



$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases} \qquad w \cdot x \equiv \sum_j w_j x_j$$

普遍性

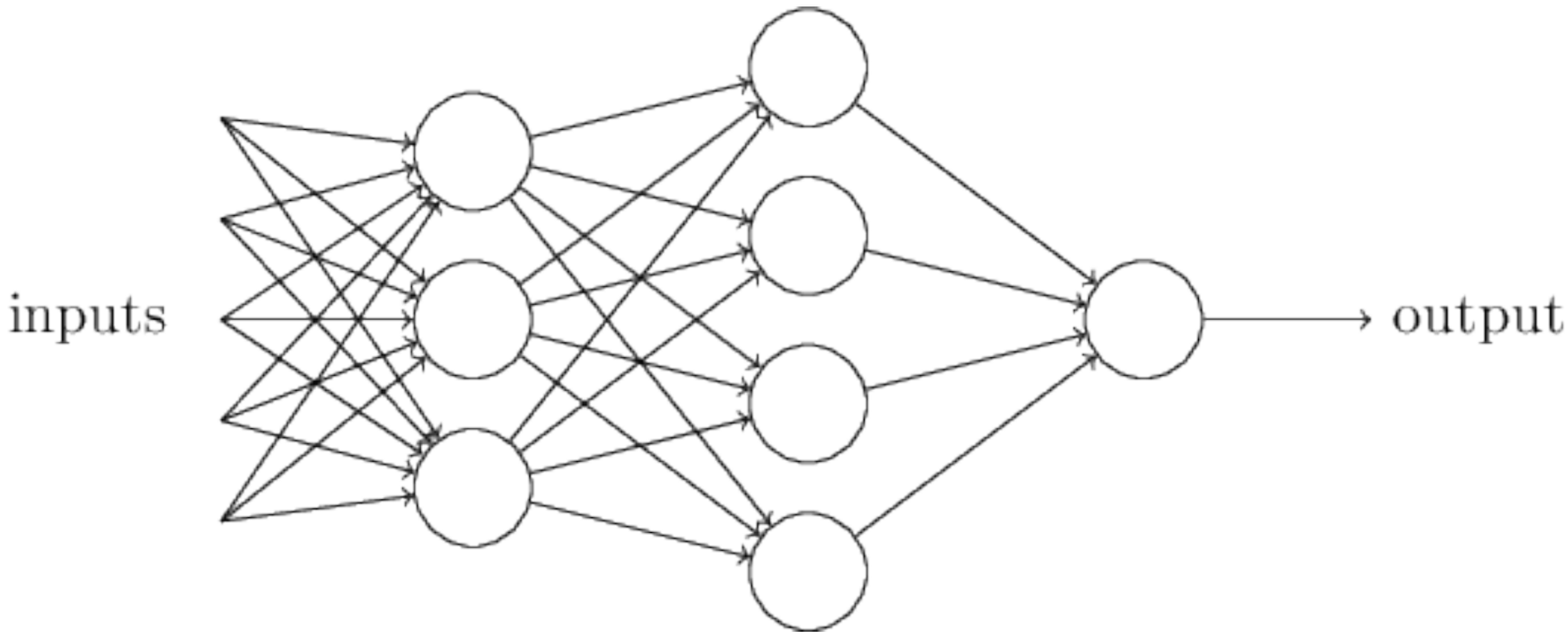
一个单层感知机可以学习任何单变量函数

- 只要是可微的So long as it is differentiable
- 某种程度的近似;
  更多神经元 = 更好的近似

证明(Michael Nielson):

http://neuralnetworksanddeeplearning.com/chap4.html

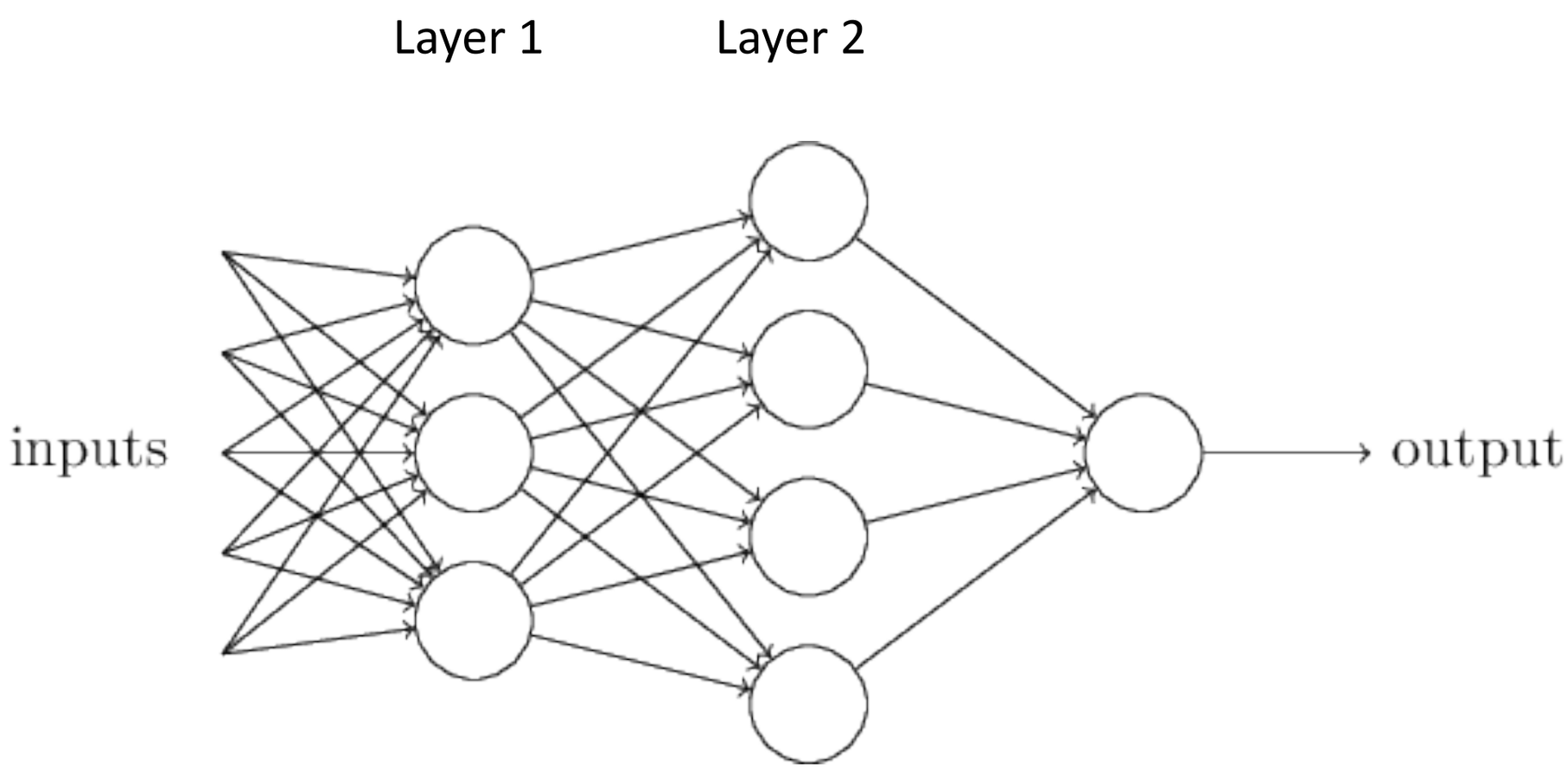
组合性



- 简单函数的组合去表示复杂函数.
- 一个神经元的输出被送入另一个神经元的输入

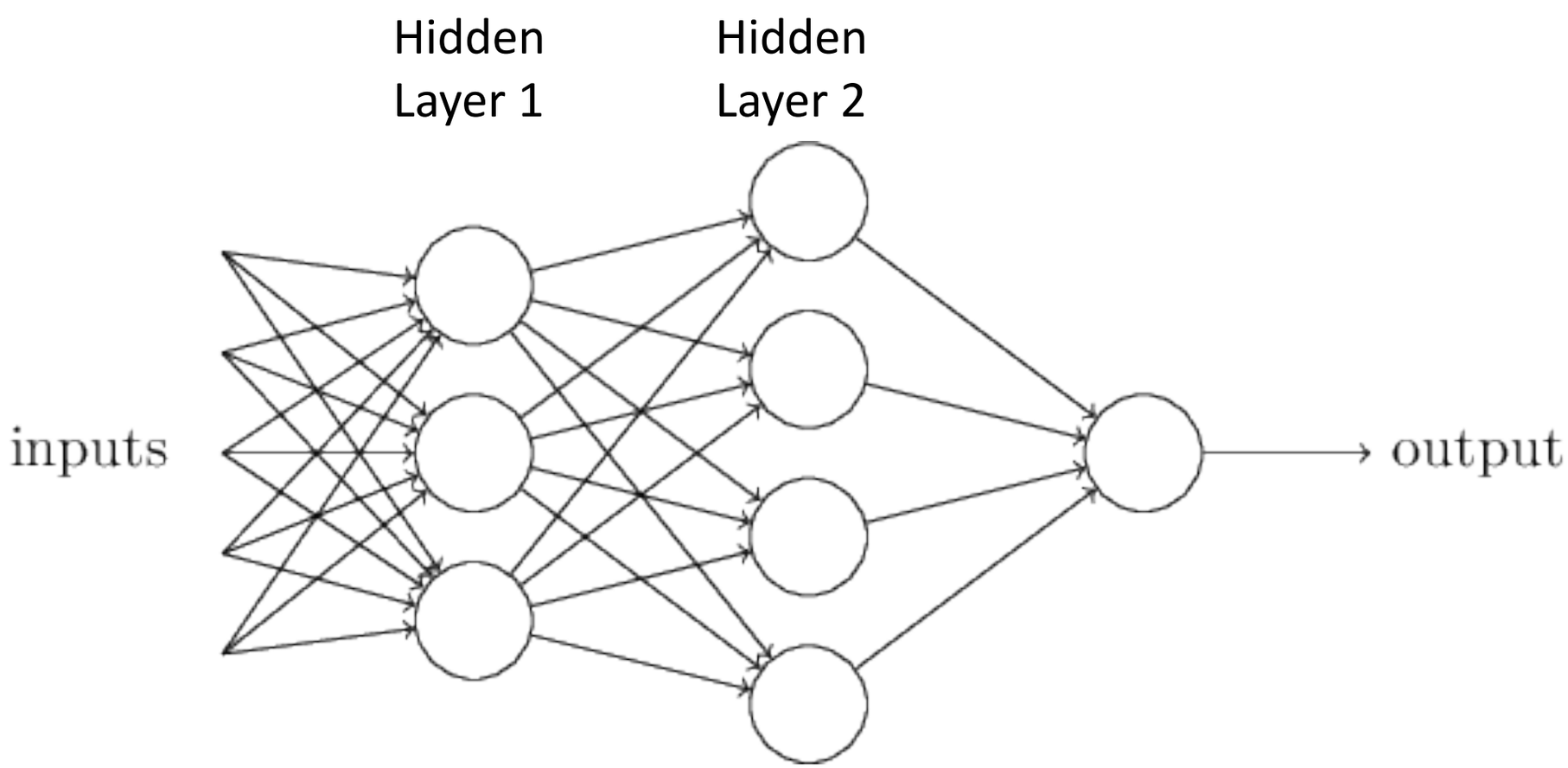
Nielsen

组合性



- 层的集合+连接层与层的权值定义了网络架构

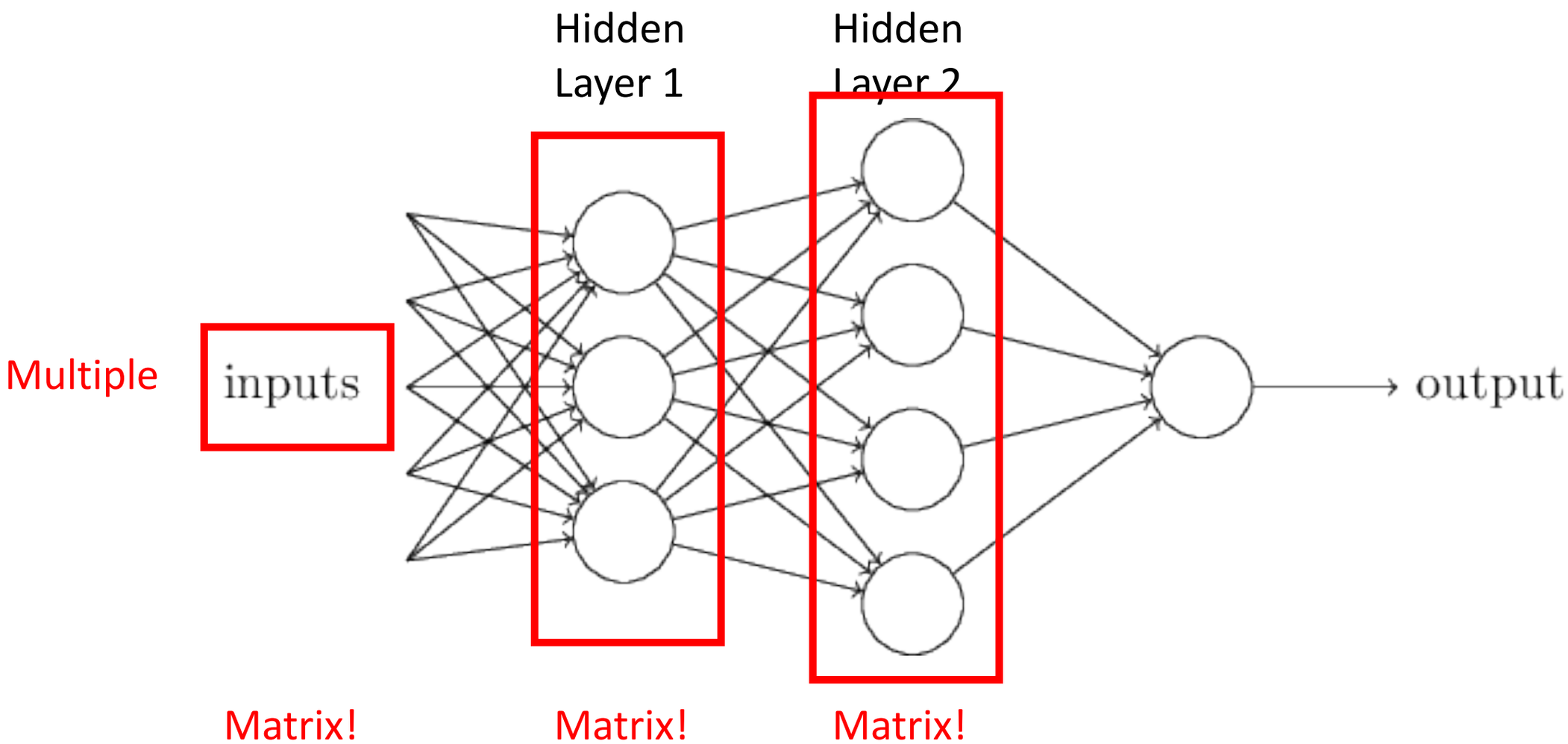
组合性



- 隐层：输入和输出层之间的层，权值通过学习（优化）得到.

Nielsen

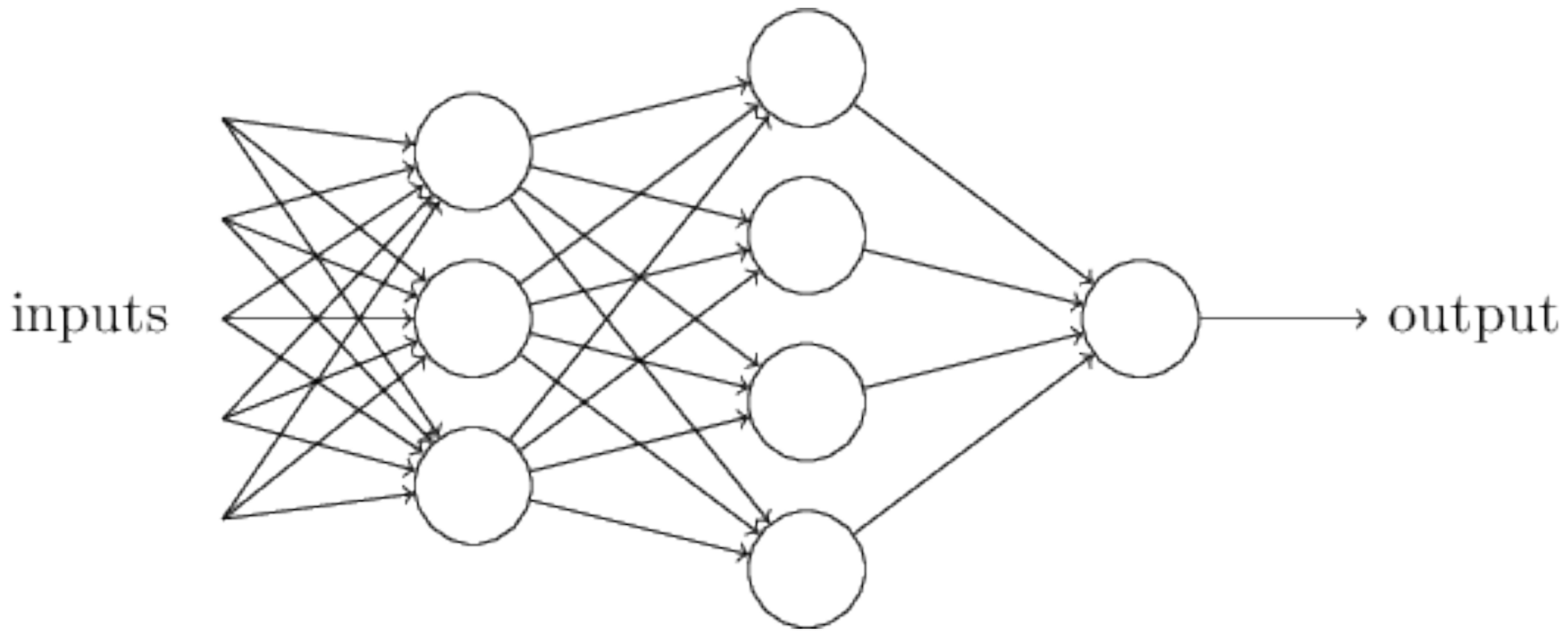
组合性



均为矩阵乘. GPU是特别适合的硬件，适合快速计算大规模矩阵乘法

多层感知机 (MLP)

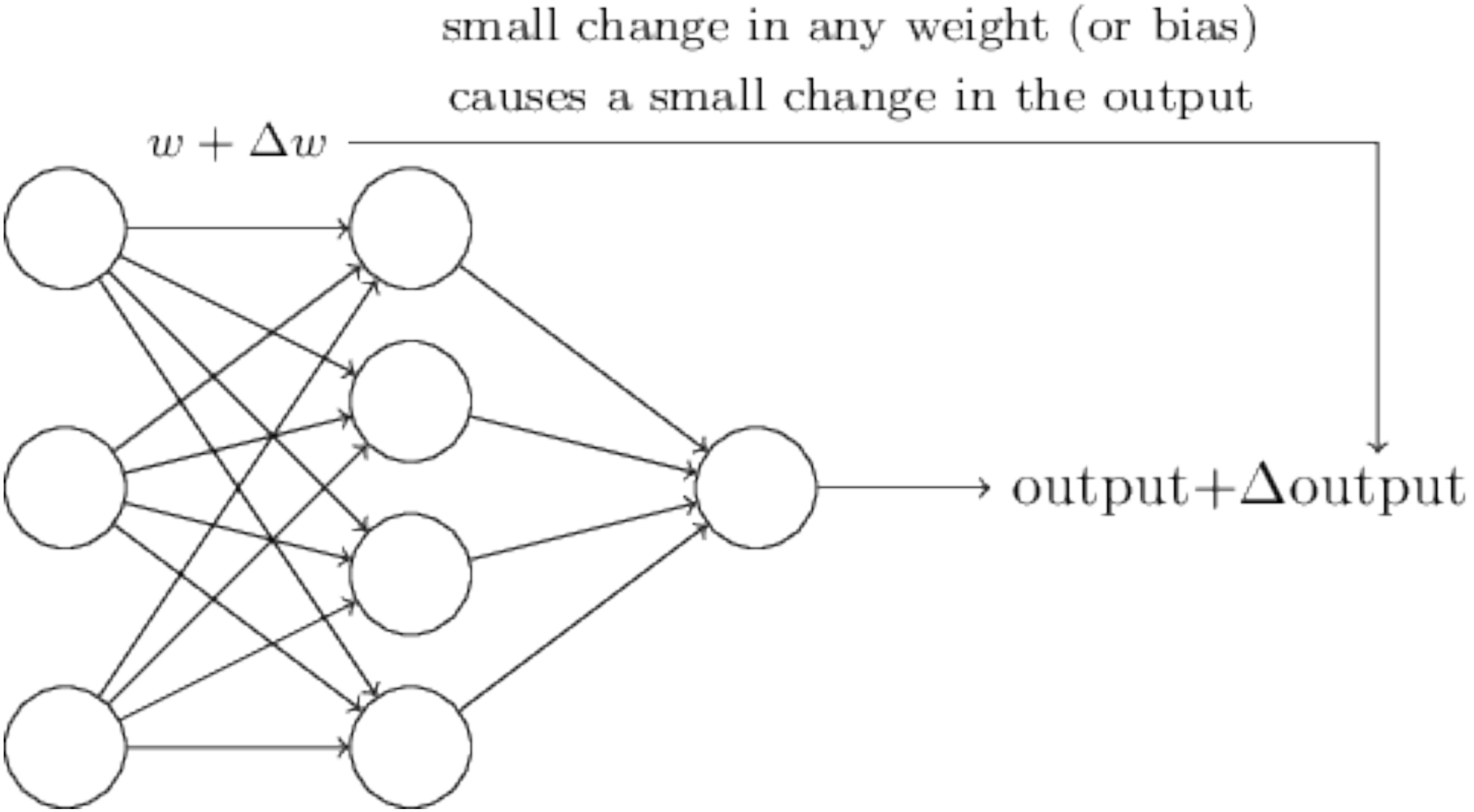
- 全连接+非线性激活函数



- ‘前向’ 神经网络

为什么要加入非线性激活函数?

1. 线性函数链：g = f(h(x))，仍然是线性函数，因此如果没有非线性激活函数，若干线性函数的组合仍然是一个简单线性函数；
2. 线性分类器：小的输入改变会引起二值输出的大的变化

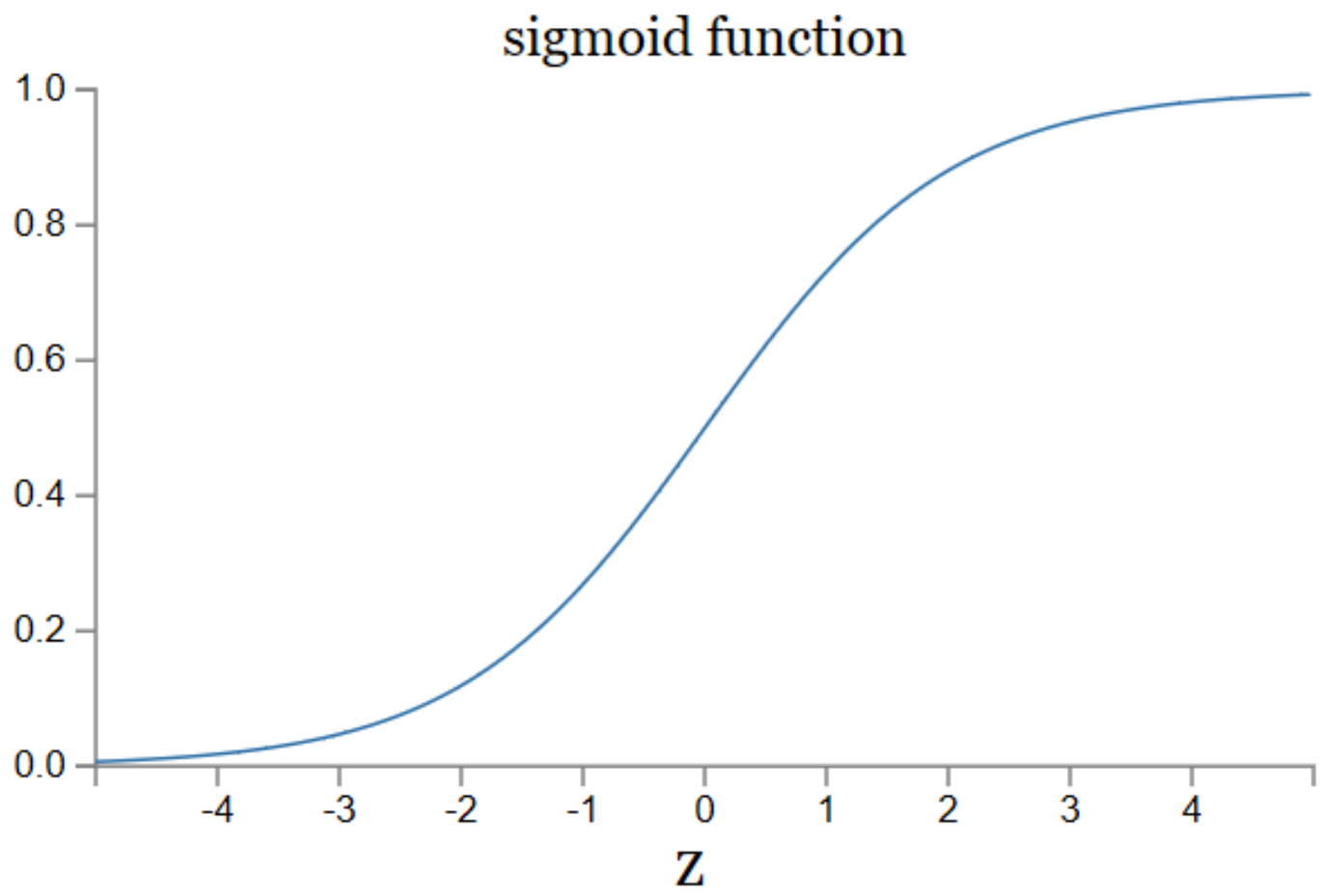

small change in any weight (or bias)
causes a small change in the output
$w + \Delta w$
output+Δoutput

非线性函数

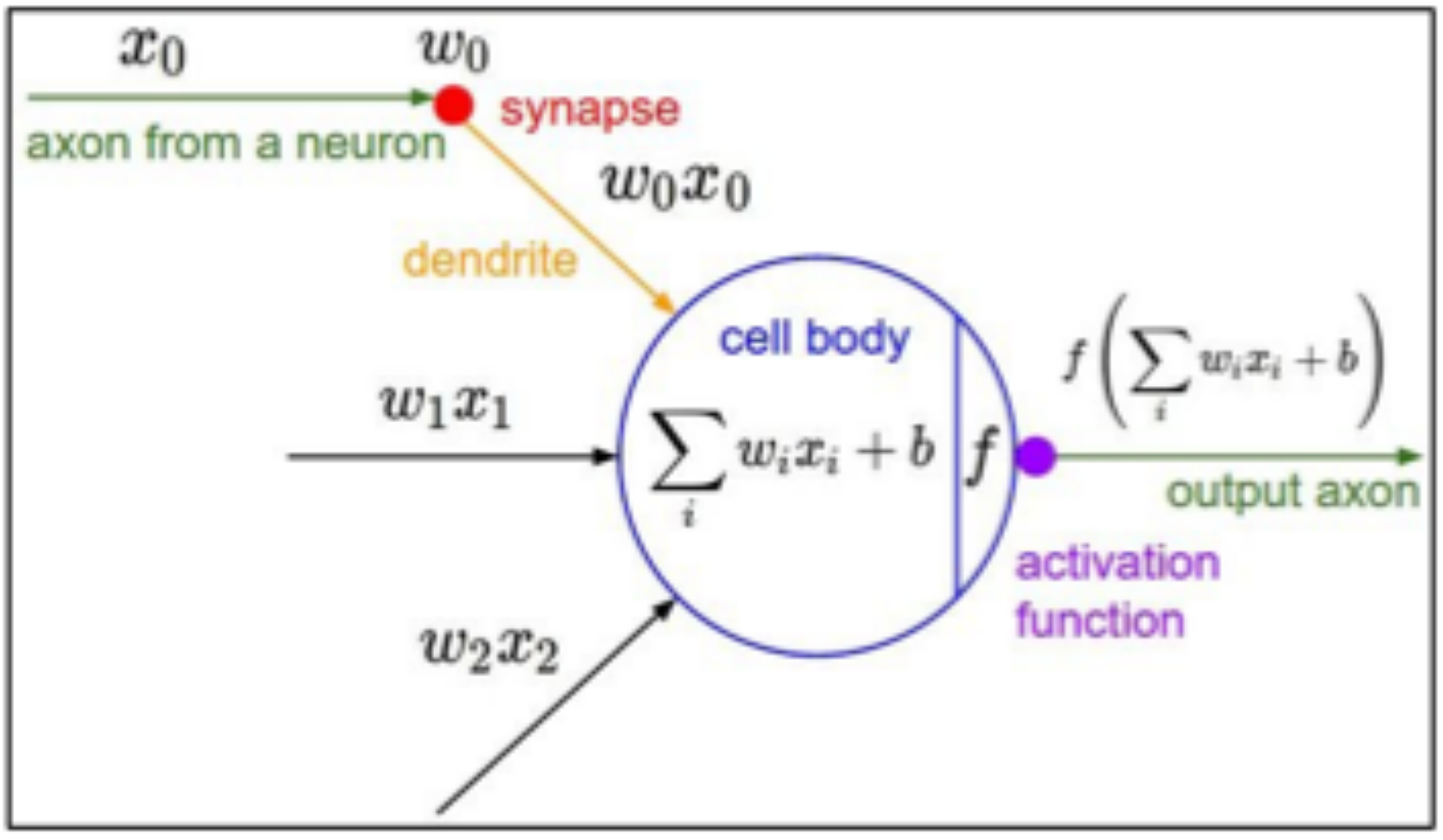
引入非线性函数进行特征变换

$$\sigma(w \cdot x + b)$$

$$\sigma(z) \equiv \frac{1}{1 + e^{-z}}.$$



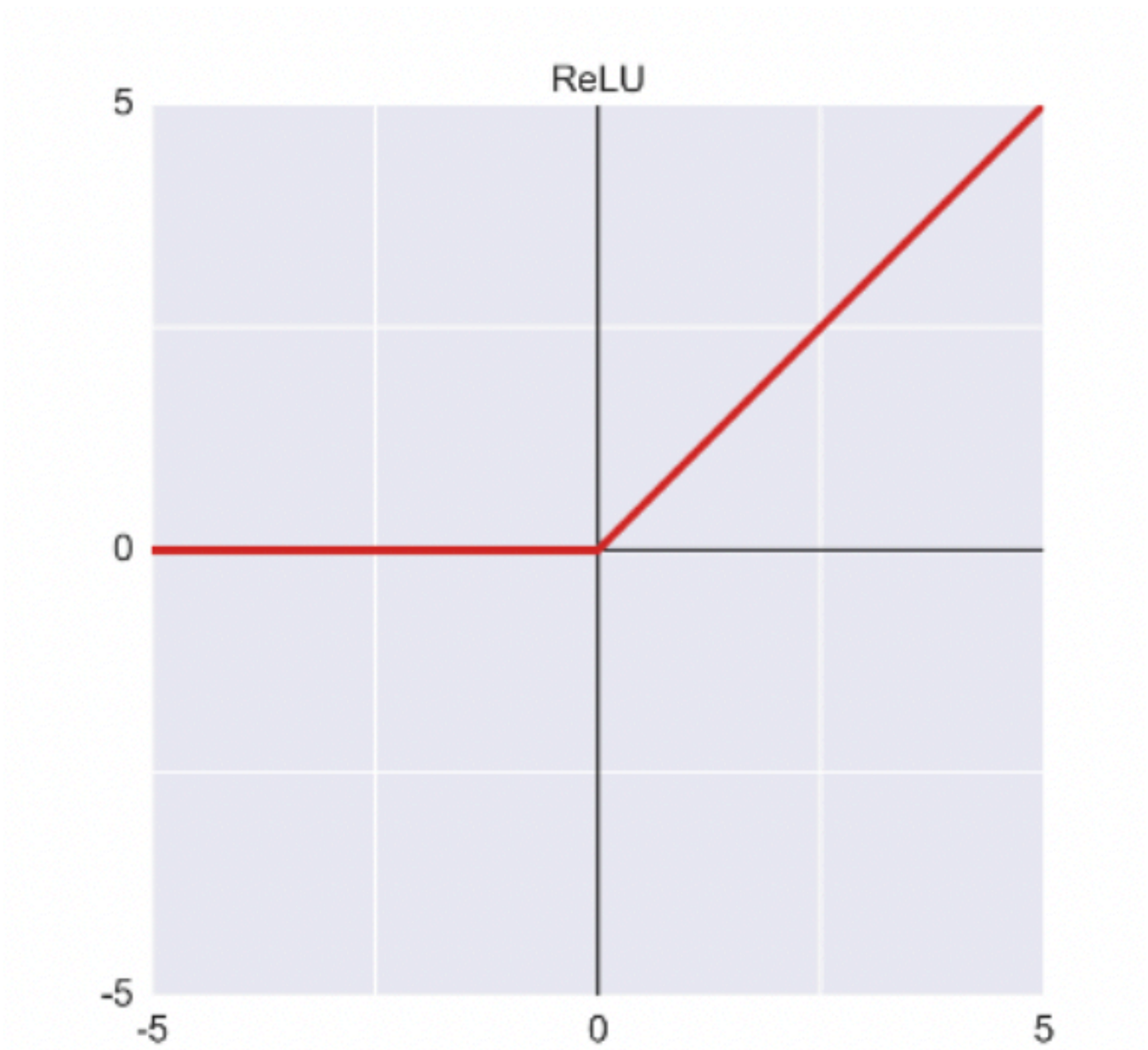
Nielsen

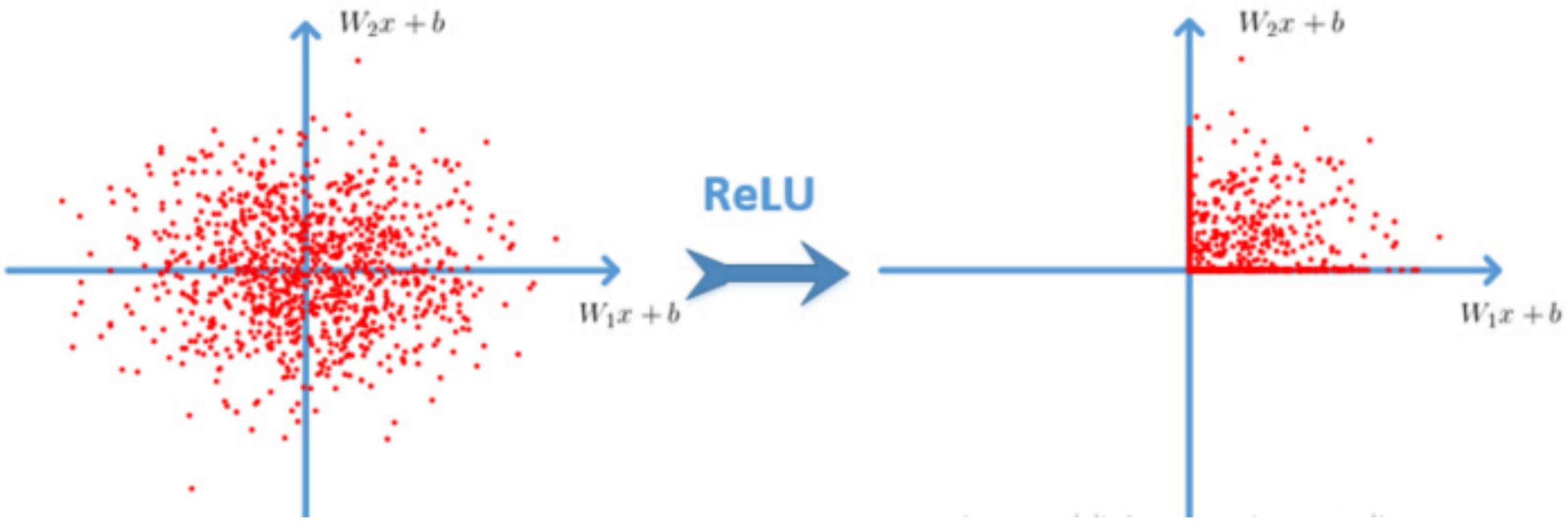

x_0
w_0
synapse
axon from a neuron
$w_0 x_0$
dendrite
cell body
$f\left(\sum_i w_i x_i + b\right)$
$w_1 x_1$
$\sum_i w_i x_i + b$
f
output axon
activation function
$w_2 x_2$

激活函数: ReLU（Rectified Linear Unit）

- ReLU

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

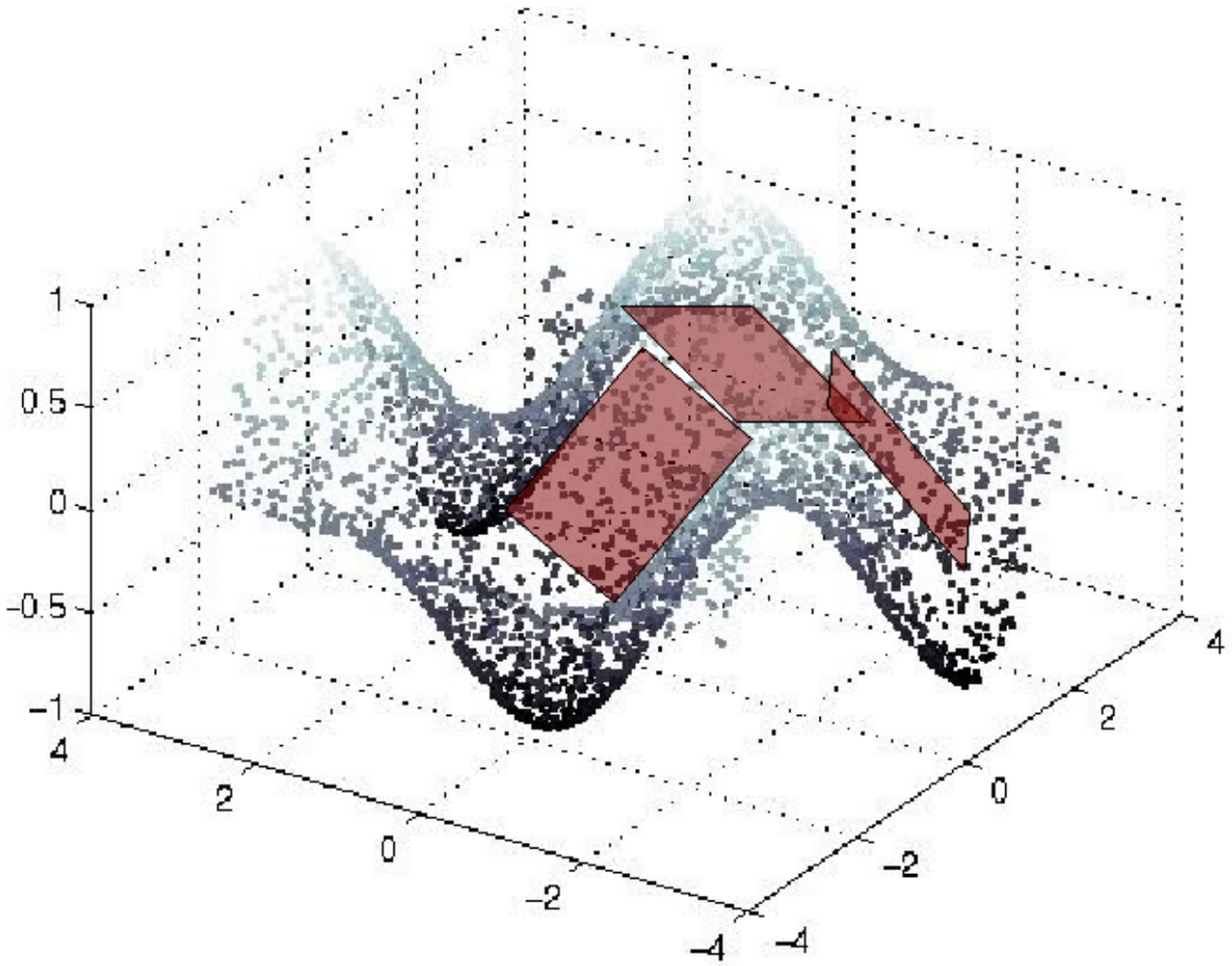



W_2x+b
W_1x+b
ReLU
W_2x+b
W_1x+b

Rectified Linear Unit

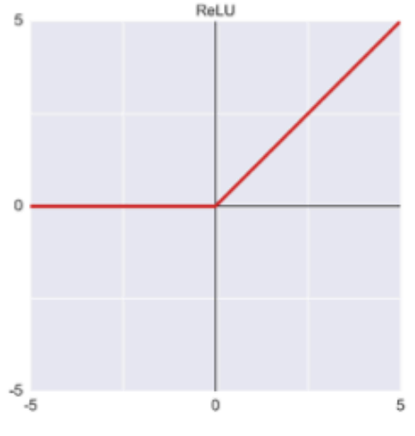
Question: What do ReLU layers accomplish?

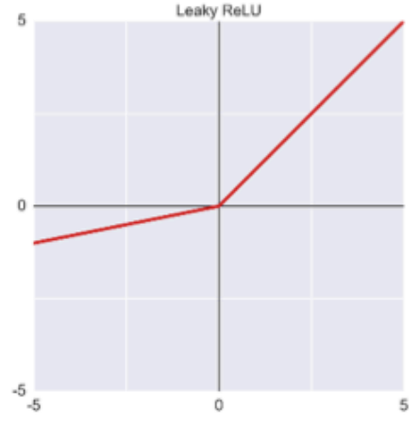
Answer: Piece-wise linear tiling: mapping is locally linear.

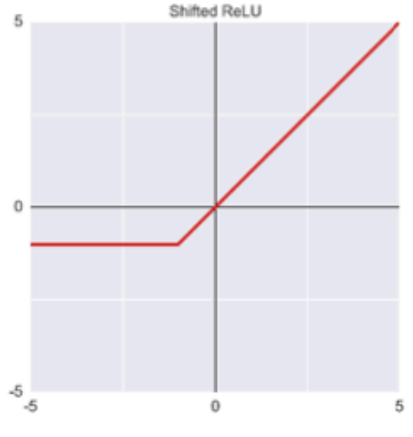


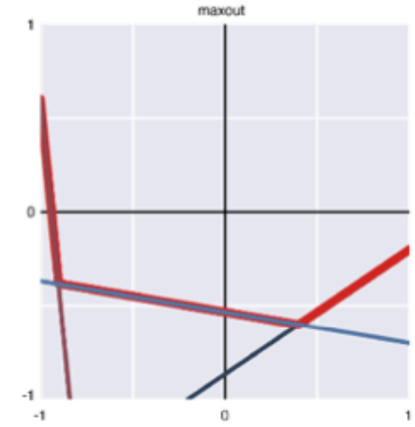
Ranzato

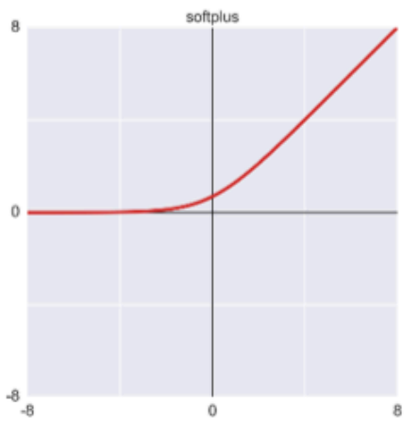
不同的激活函数

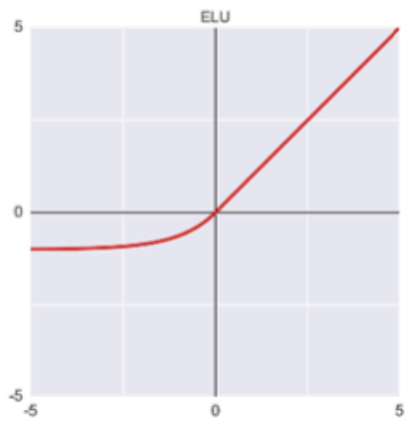


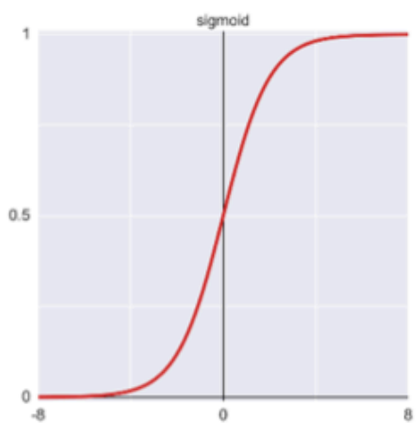


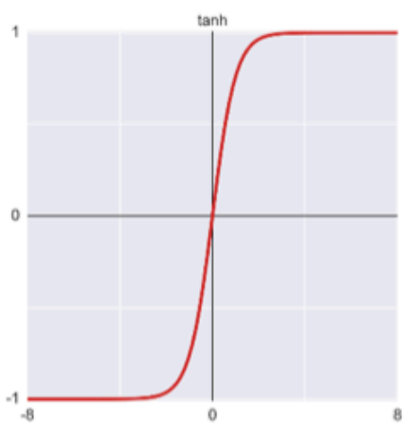


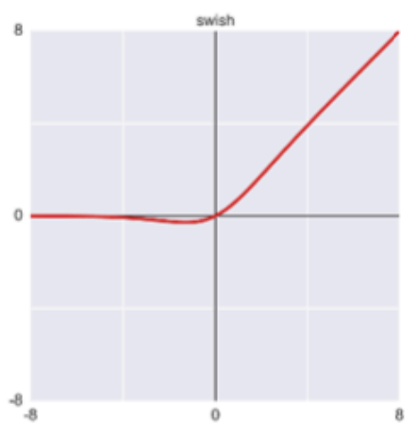






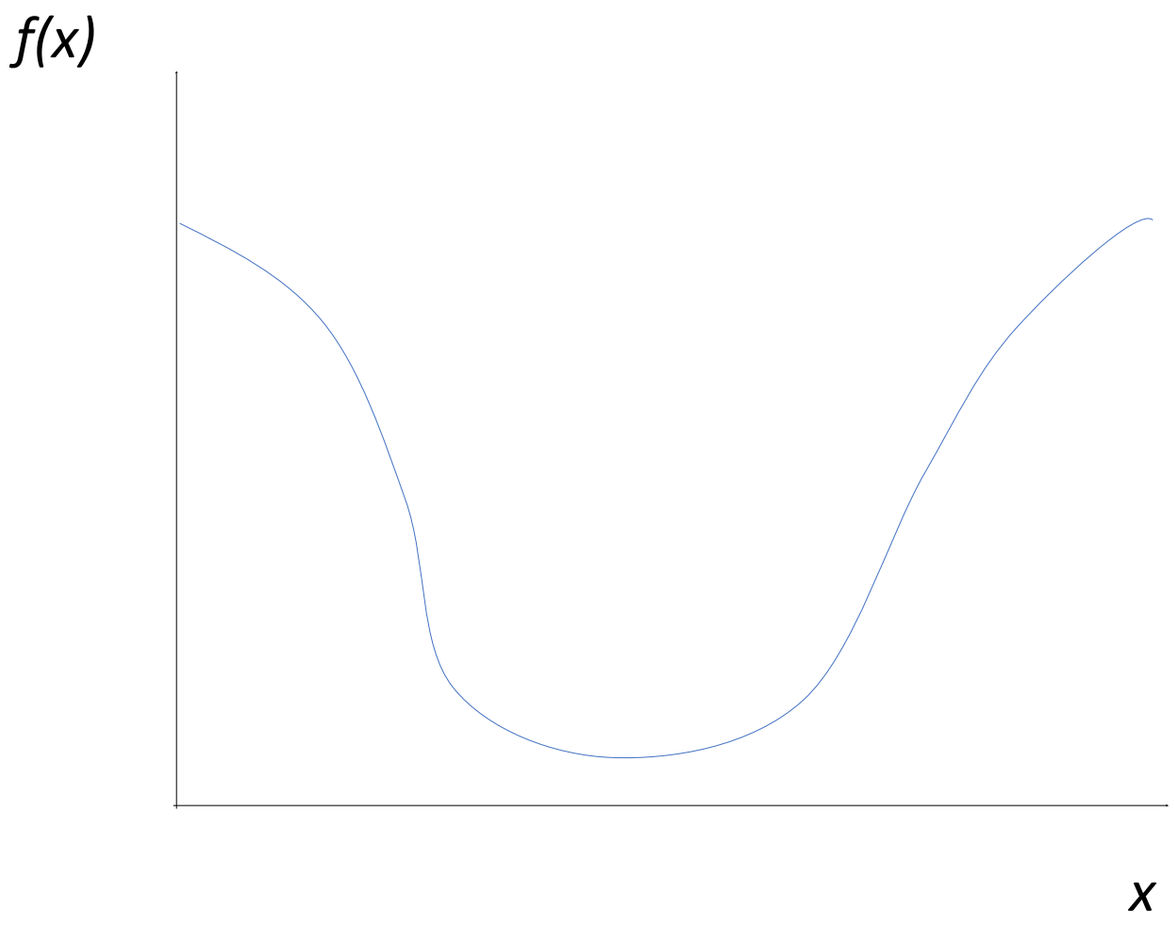






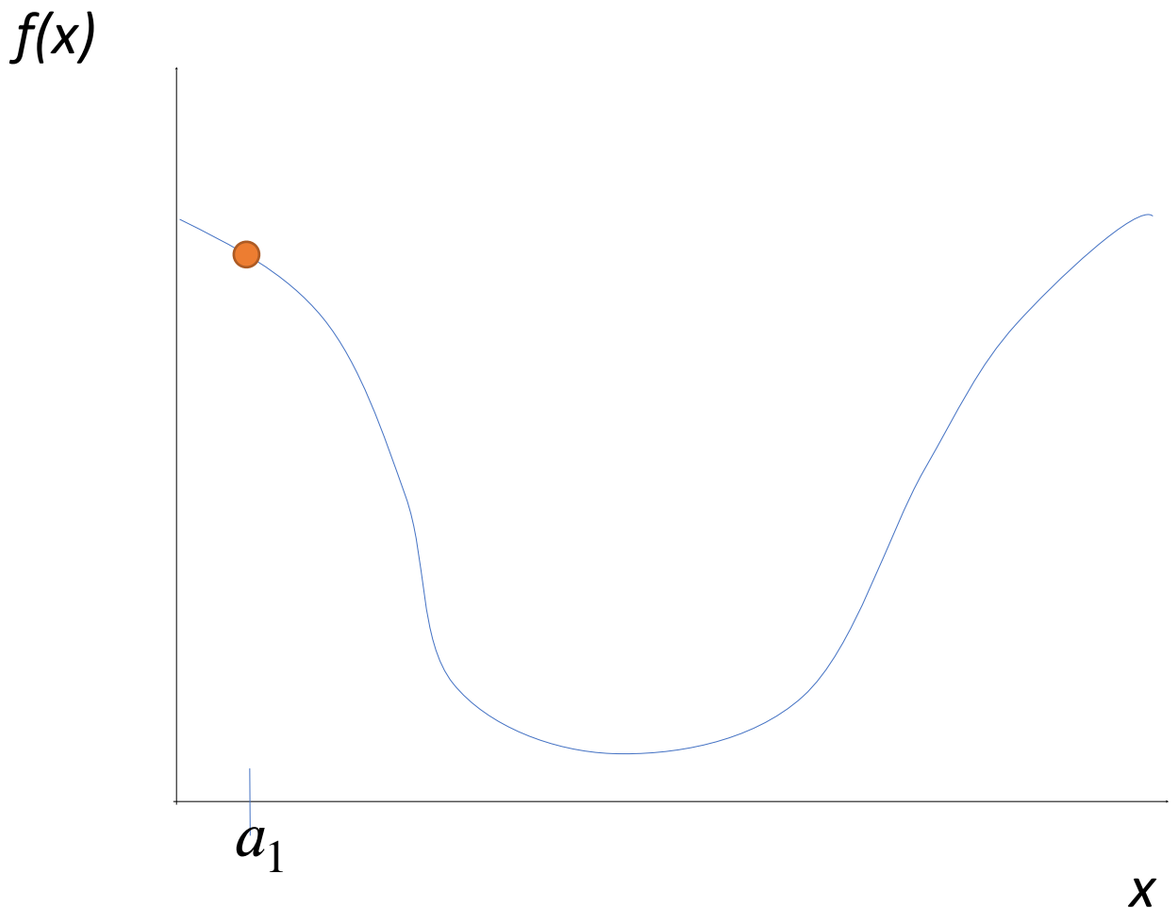
左上到右下: ReLU, leaky ReLU, shifted ReLU, maxout, softplus, ELU, sigmoid, tanh, swish.

梯度下降



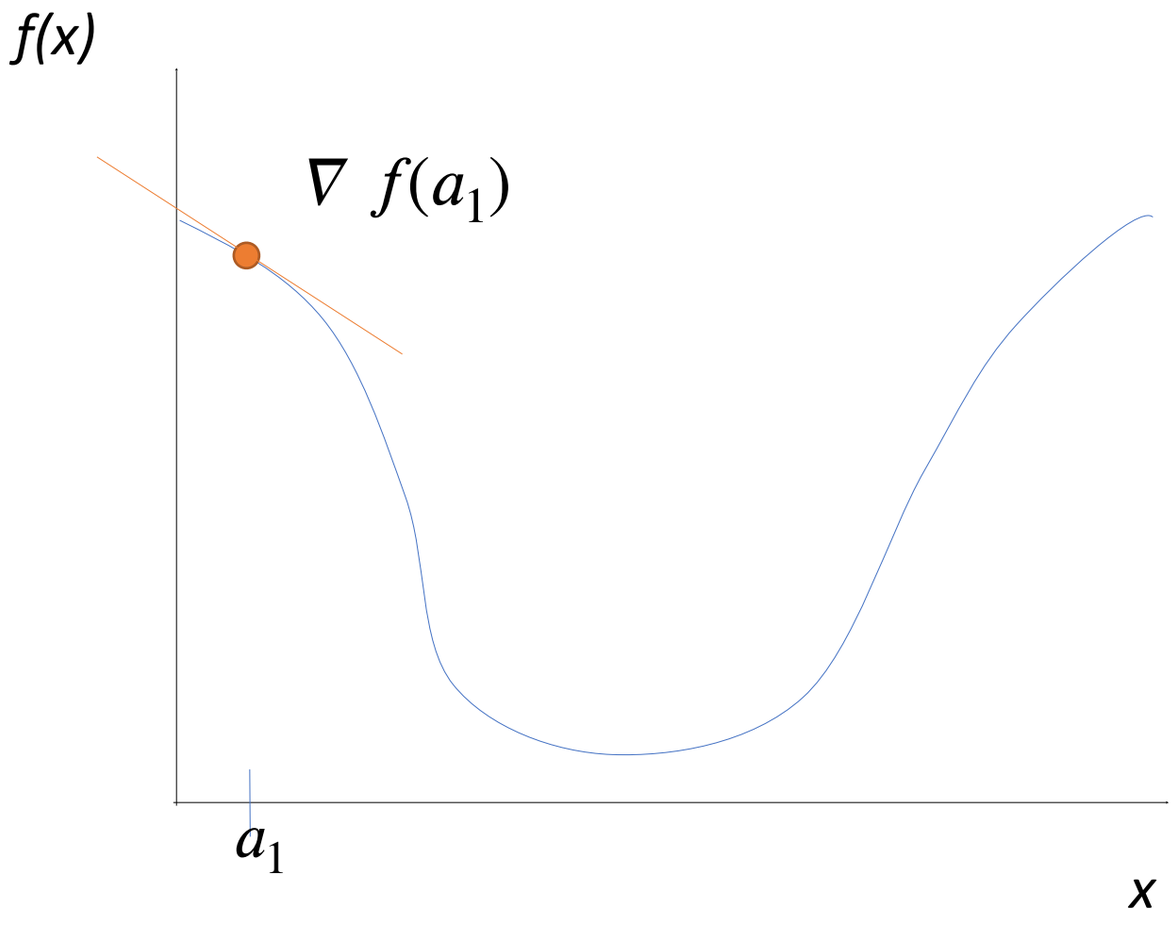
梯度下降

挑选一个随机起始点



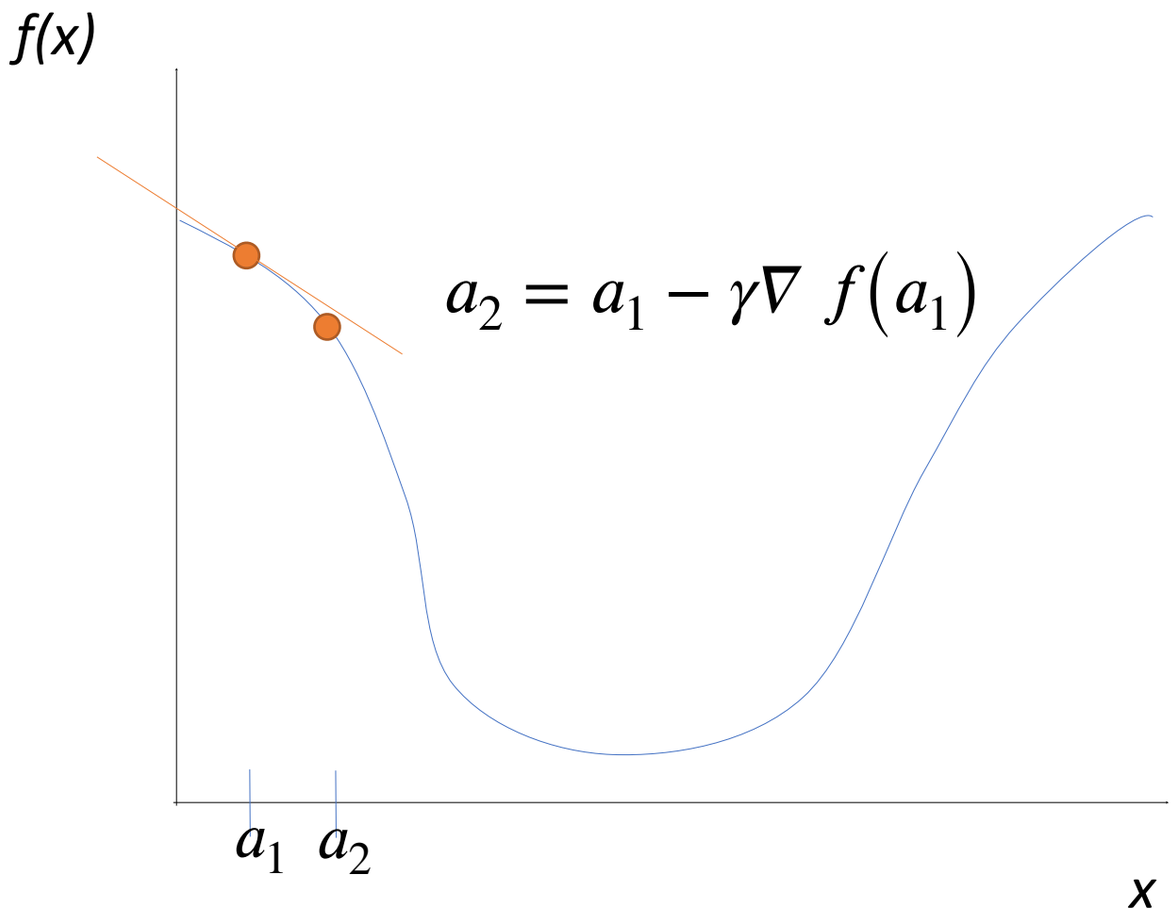
梯度下降

计算改点的梯度 (解析形式或者有限差分)



梯度下降

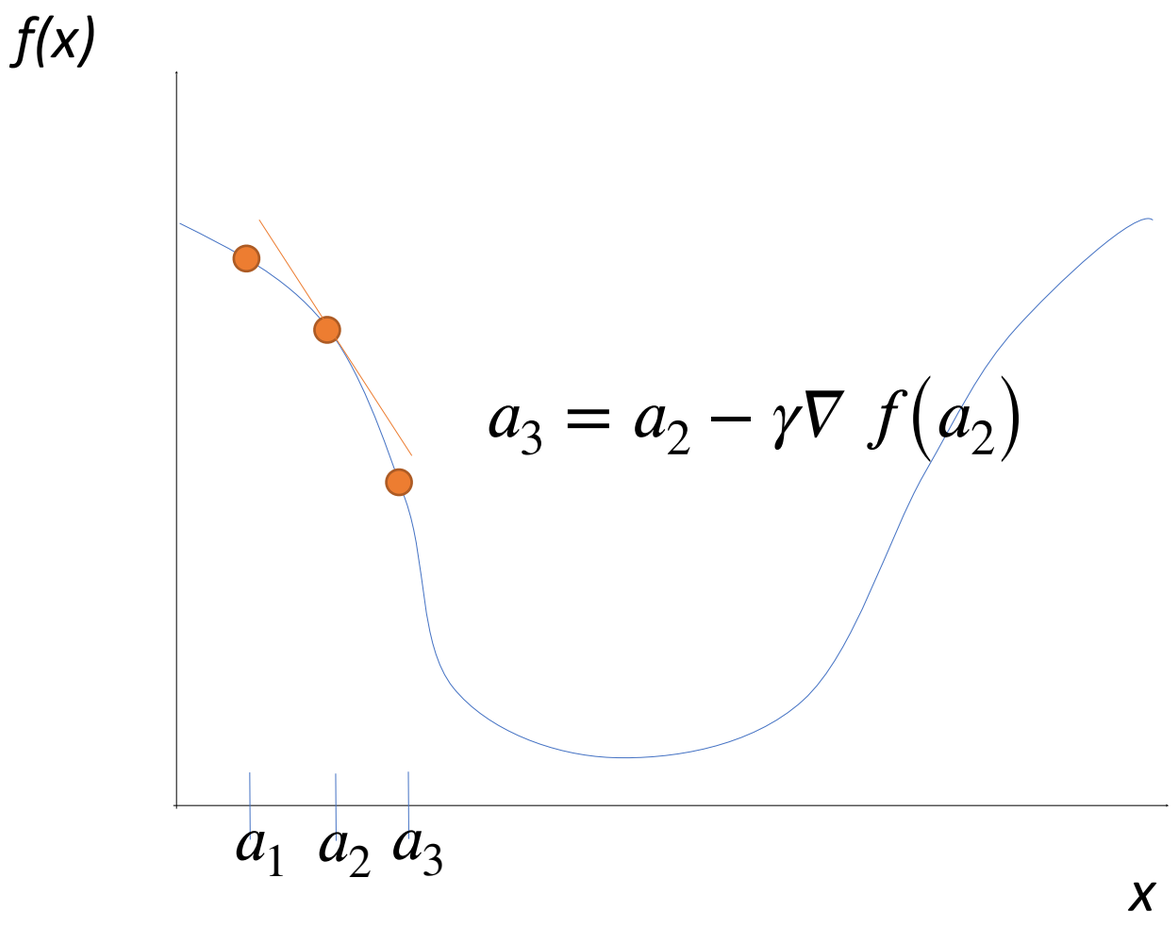
参数空间内沿着负梯度方向移动一定步长



γ = amount to move

= 学习率

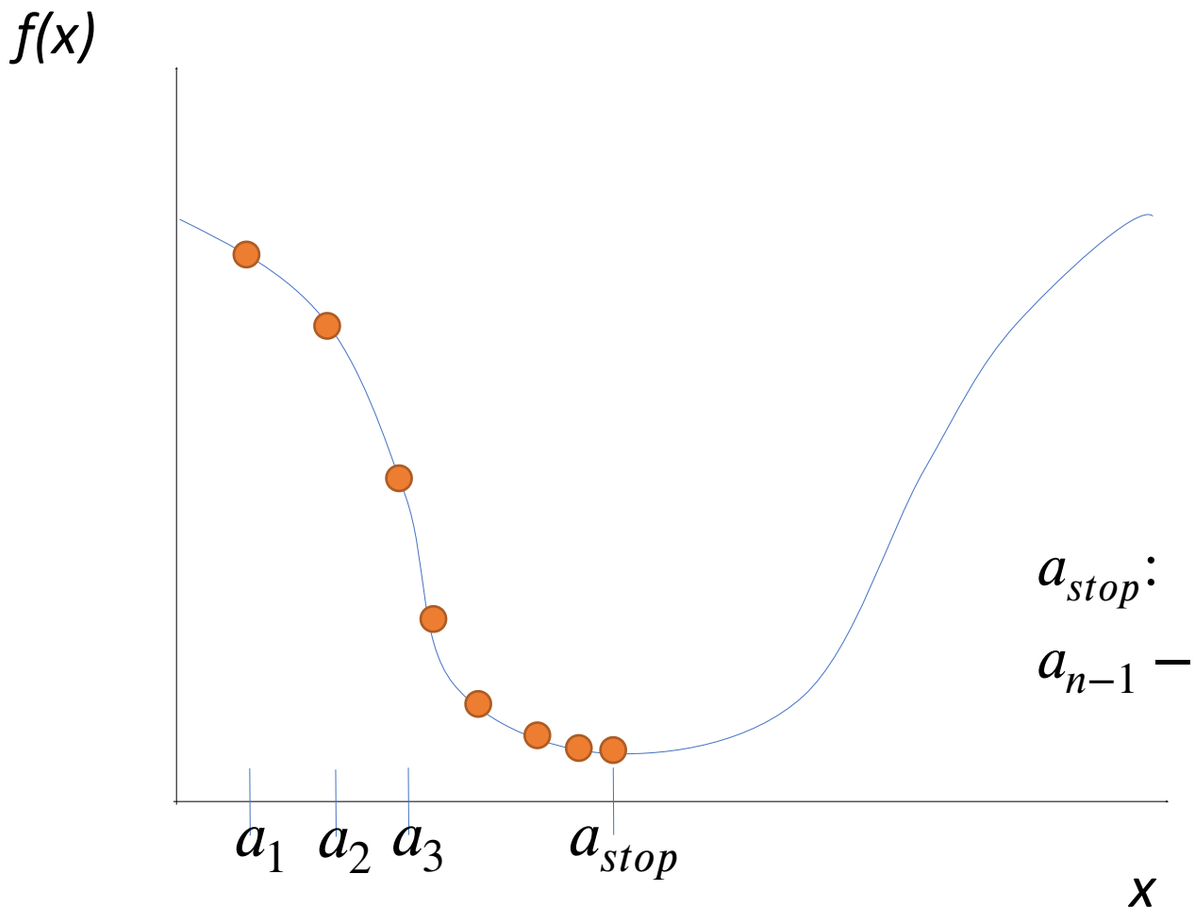
梯度下降



γ = amount to move
= *learning rate*

梯度下降

停止：移动不会引起期望（阈值限定下）的改变

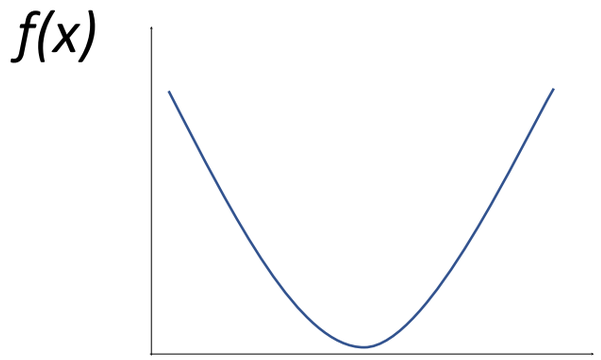


a_{stop}:

$a_{n-1} - \gamma \nabla f(a_{n-1}) = 0$

梯度下降

- 函数优化器
- 对于凸函数：可以找到最优
  - 非凸：局部最优
  - 大部分视觉问题是非凸问题
- 多变量函数
  - 需计算偏导矩阵 *(Jacobian)*



为什么不用最小二乘?

如函数为凸函数

$$A\mathbf{x} - \mathbf{b} = 0$$

$$F(\mathbf{x}) = \|A\mathbf{x} - \mathbf{b}\|^2.$$

$$\nabla F(\mathbf{x}) = 2A^T(A\mathbf{x} - \mathbf{b}).$$

解析解:

$$x = \left(A^T A\right)^{-1} A^T b$$

为什么不用最小二乘?

如果有1,000,000 数据点

大矩阵，计算困难

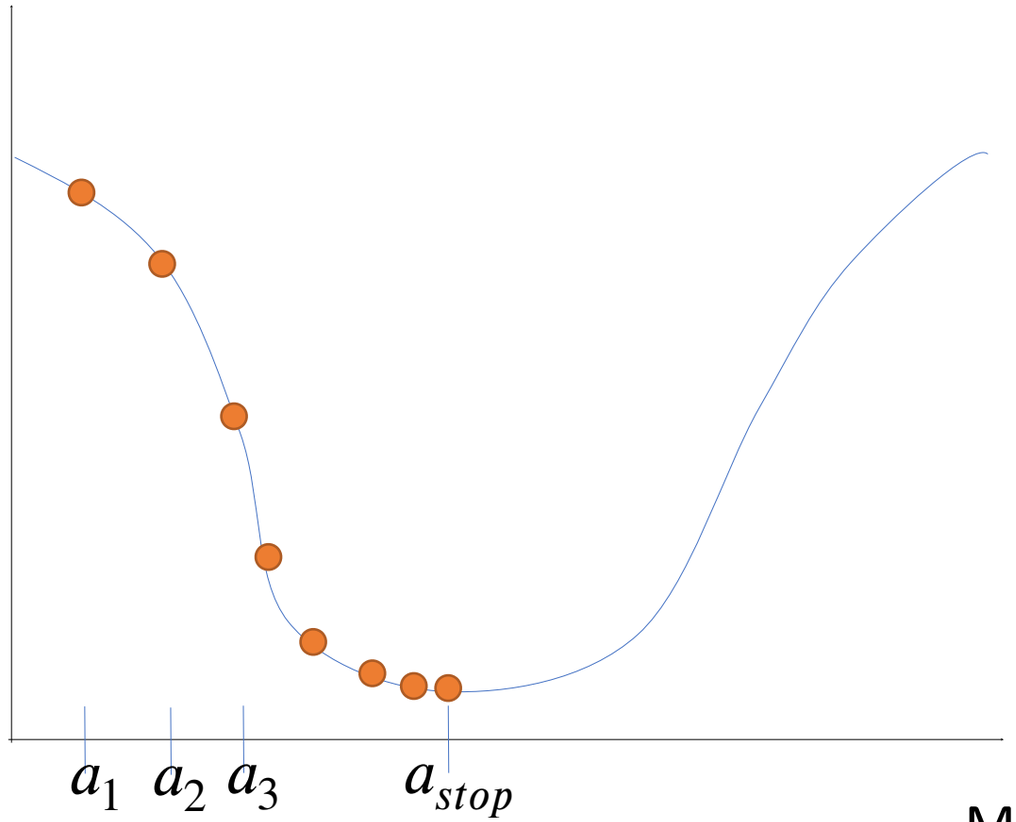
梯度下降法可以迭代求解，不需要很大的矩阵

通过梯度下降训练神经网络

- x^i, y^i = n 训练样本
- $f(x)$ = 前向神经网络
- L(**x**, y; **θ**) = 损失函数
  - 测度网络在模型参数下针对训练样本的的分类能力

通过梯度下降训练神经网络

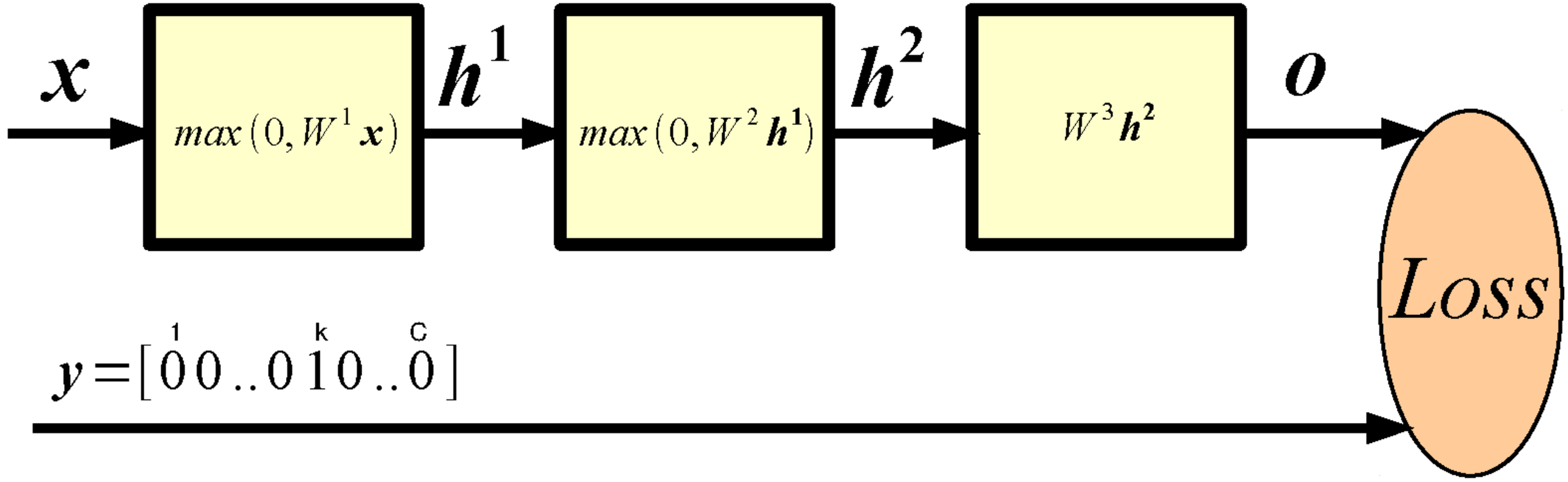
Loss function
(Evaluate NN
on training data)



Model parameters
(perceptron weights)

网络性能的评价

How Good is a Network?



损失函数

- 定义输出阈值
- 分类：比较训练样本的真实类别和预测类别
- 0-1损失：

$$y = true\ label$$

$$\hat{y} = predicted\ label$$

- 不是一个好的损失函数：不可求导

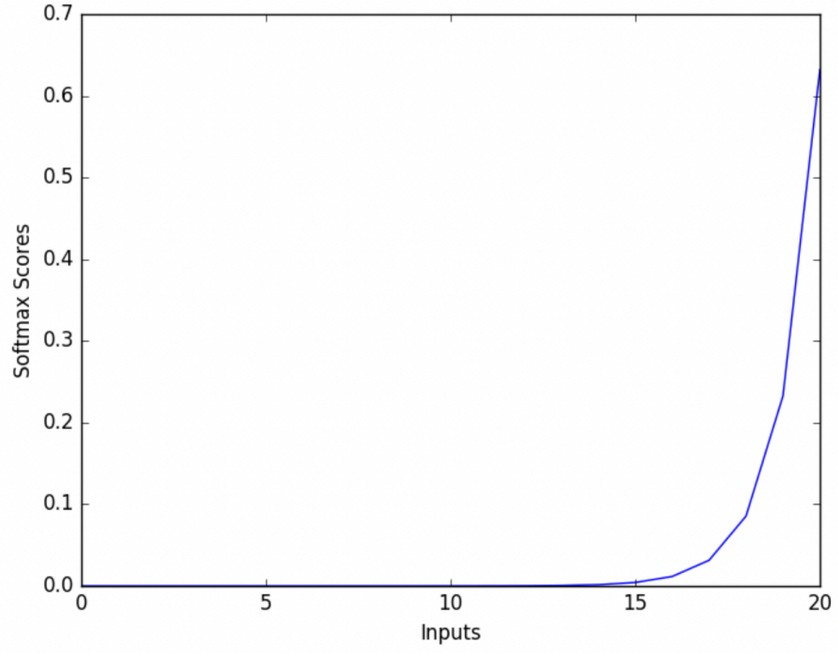
根据概率分类

最后一层：使用特殊函数 - '*Softmax*':

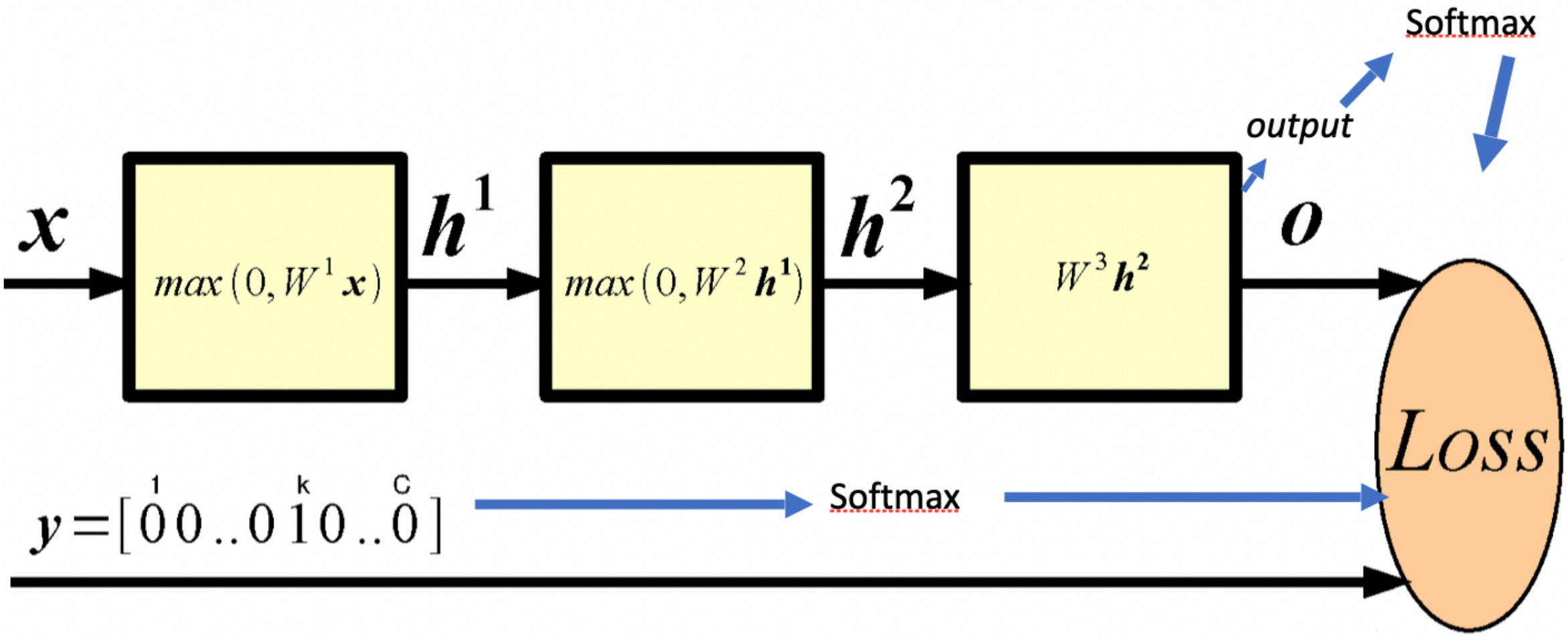
"Squashes" a *C*-dimensional vector $\boldsymbol{o}$ of arbitrary real values to a *C*-dimensional vector $\sigma(\boldsymbol{o})$ of real values in the range (0, 1) that add up to 1.

把输出变成关于类别的概率分布

$$p(c_k = 1|\boldsymbol{x}) = \frac{e^{o_k}}{\sum_{j=1}^{C} e^{o_j}}$$



网络性能的评价



Probability of class k given input (softmax):

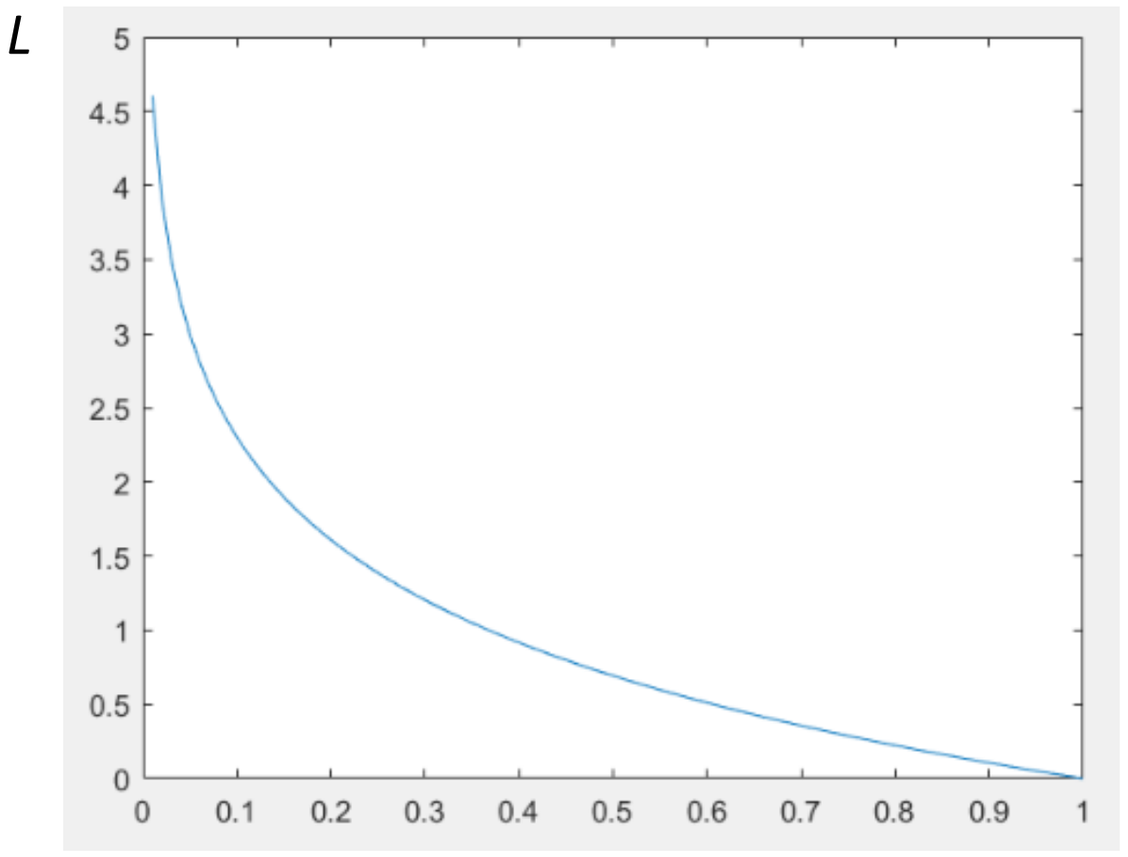
$$p(c_k = 1|\boldsymbol{x}) = \frac{e^{o_k}}{\sum_{j=1}^{C} e^{o_j}}$$

交叉熵损失函数

- 负对数似然函数

$$L\left(\boldsymbol{x}, y; \boldsymbol{\theta}\right) = -\sum_{j} y_j \log p(c_j \mid \boldsymbol{x})$$

- 最小化该损失函数等价于最小化预测与目标分布的KL散度
- 是否好的损失函数?
  - 可导
  - 随着概率的增加代价也降低



训练

Learning consists of minimizing the loss (plus some regularization term) w.r.t. parameters over the whole training set.

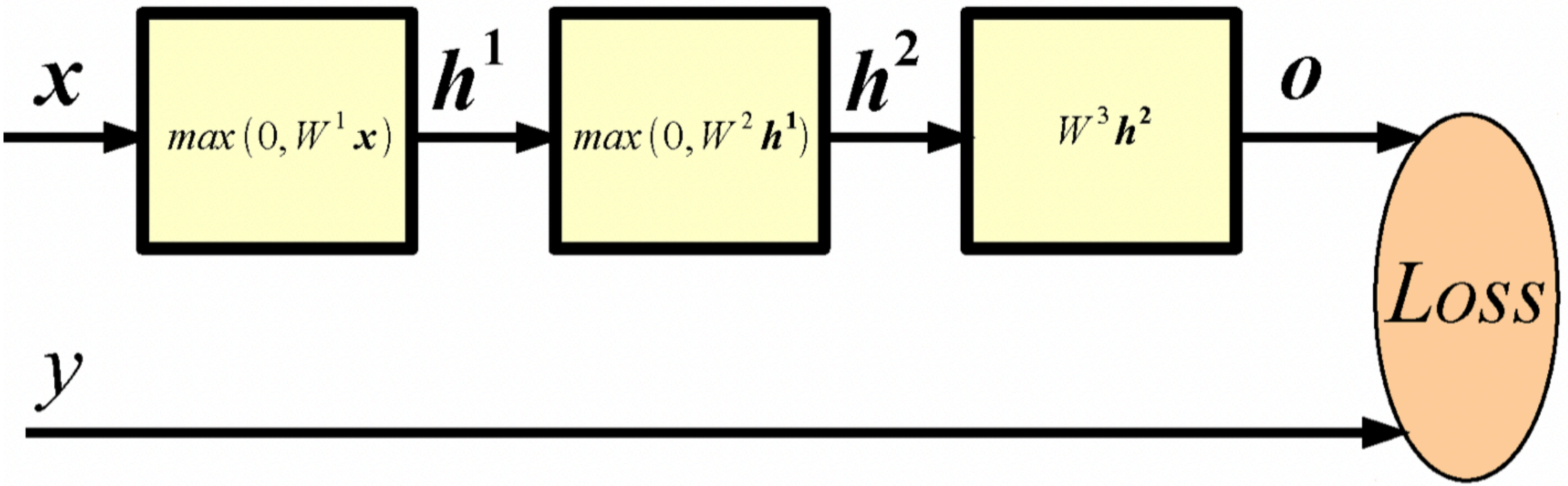
$$\boldsymbol{\theta}^* = arg\,min_{\boldsymbol{\theta}} \sum_{n=1}^{P} L\left(\boldsymbol{x}^n, y^n; \boldsymbol{\theta}\right)$$

Question: How to minimize a complicated function of the parameters?

Answer: Chain rule, a.k.a. **Backpropagation**! That is the procedure to compute gradients of the loss w.r.t. parameters in a multi-layer neural network.

Rumelhart et al. "Learning internal representations by back-propagating.." Nature 1986

核心思想：逐渐降低损失



Let's say we want to decrease the loss by adjusting $W^1_{i,j}$.
We could consider a very small $\epsilon = 1\text{e-}6$ and compute:

$$L(\boldsymbol{x}, y; \boldsymbol{\theta})$$

$$L(\boldsymbol{x}, y; \boldsymbol{\theta} \setminus W^1_{i,j}, W^1_{i,j} + \epsilon)$$

Then, update:

$$W^1_{i,j} \leftarrow W^1_{i,j} + \epsilon \, sgn(L(\boldsymbol{x}, y; \boldsymbol{\theta}) - L(\boldsymbol{x}, y; \boldsymbol{\theta} \setminus W^1_{i,j}, W^1_{i,j} + \epsilon))$$

Derivative w.r.t. Input of Softmax

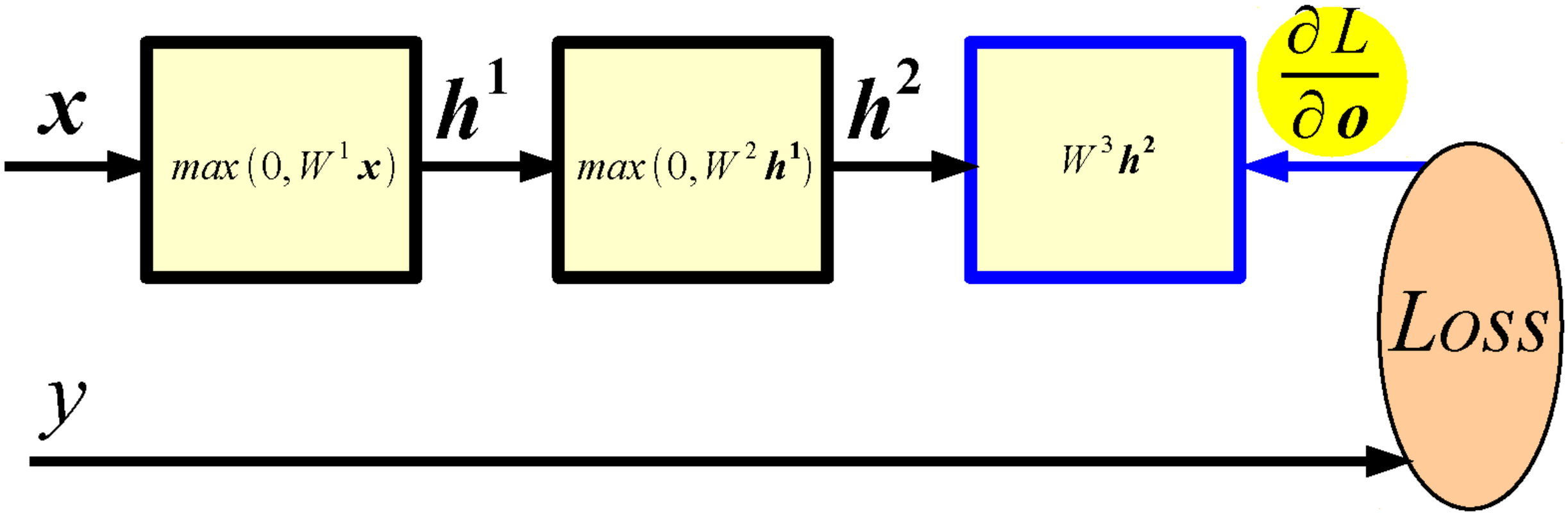
$$p(c_k=1|\boldsymbol{x})=\frac{e^{o_k}}{\sum_j e^{o_j}}$$

$$L(\boldsymbol{x},y;\boldsymbol{\theta})=-\sum_j y_j \log p(c_j|\boldsymbol{x}) \qquad \boldsymbol{y}=[\overset{1}{0}0..0\overset{k}{1}0..\overset{C}{0}]$$

By substituting the fist formula in the second, and taking the derivative w.r.t. $\boldsymbol{o}$ we get:

$$\frac{\partial L}{\partial o}=p(c|\boldsymbol{x})-\boldsymbol{y}$$

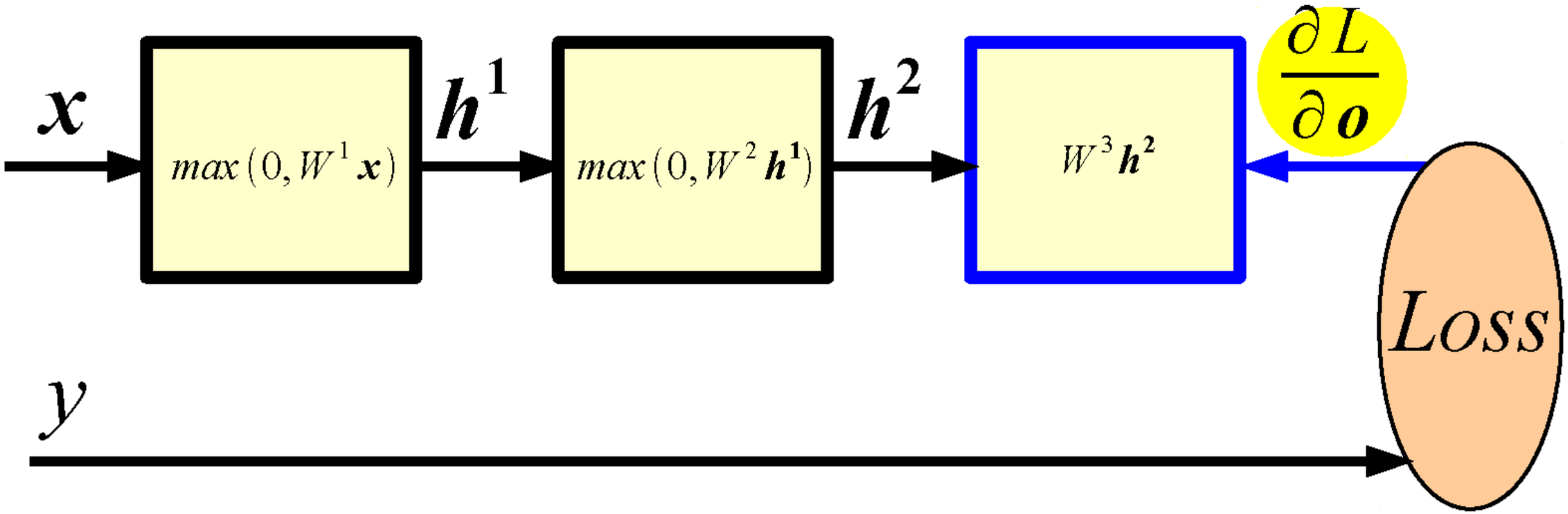
Backward Propagation



Given $\partial L / \partial \boldsymbol{o}$ and assuming we can easily compute the Jacobian of each module, we have:

$$\frac{\partial L}{\partial W^3} = \frac{\partial L}{\partial \boldsymbol{o}} \frac{\partial \boldsymbol{o}}{\partial W^3}$$

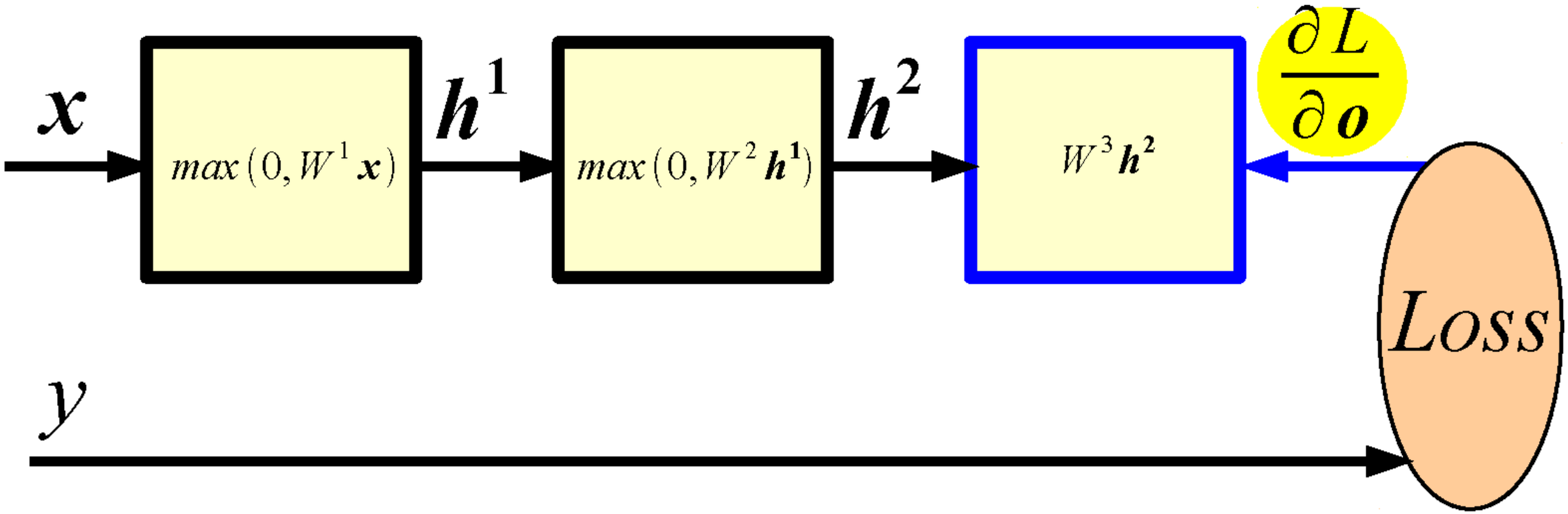
Backward Propagation



Given $\partial L / \partial \boldsymbol{o}$ and assuming we can easily compute the Jacobian of each module, we have:

$$\frac{\partial L}{\partial W^3} = \frac{\partial L}{\partial \boldsymbol{o}} \frac{\partial \boldsymbol{o}}{\partial W^3} \qquad \frac{\partial L}{\partial \boldsymbol{h}^2} = \frac{\partial L}{\partial \boldsymbol{o}} \frac{\partial \boldsymbol{o}}{\partial \boldsymbol{h}^2}$$

Backward Propagation

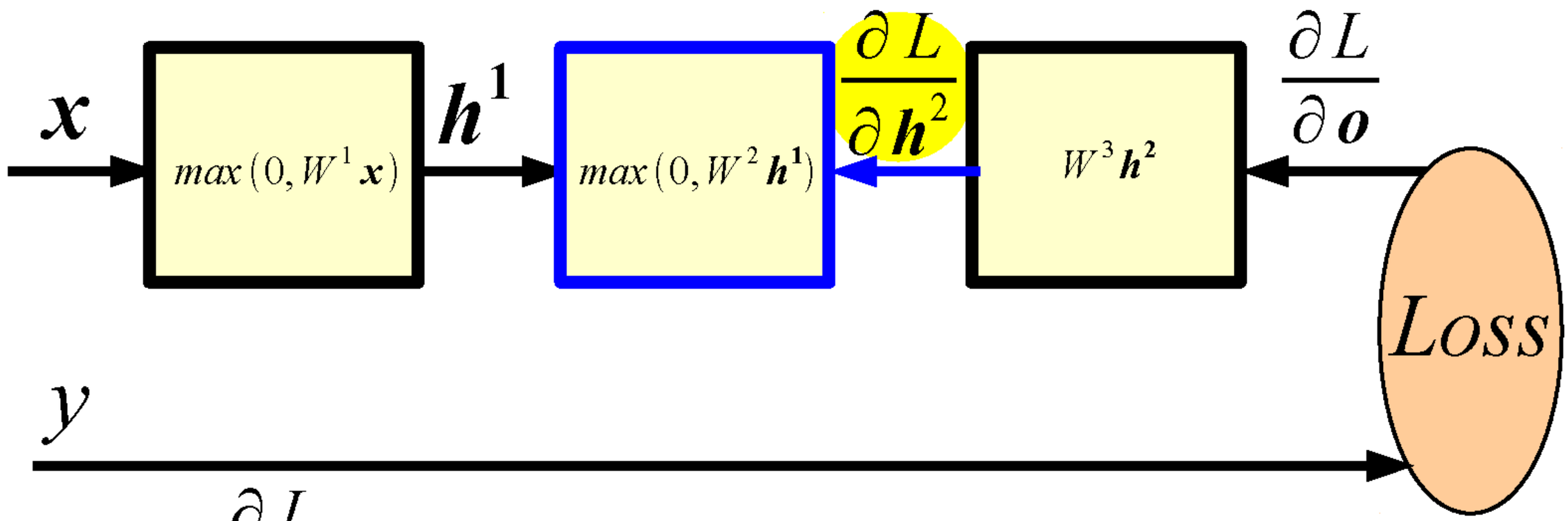


Given $\partial L / \partial \boldsymbol{o}$ and assuming we can easily compute the Jacobian of each module, we have:

$$\frac{\partial L}{\partial W^3} = \frac{\partial L}{\partial \boldsymbol{o}} \frac{\partial \boldsymbol{o}}{\partial W^3} \qquad \frac{\partial L}{\partial \boldsymbol{h}^2} = \frac{\partial L}{\partial \boldsymbol{o}} \frac{\partial \boldsymbol{o}}{\partial \boldsymbol{h}^2}$$

$$\frac{\partial L}{\partial W^3} = (p(c|\boldsymbol{x}) - \boldsymbol{y})\ \boldsymbol{h}^{2\,T} \qquad \frac{\partial L}{\partial \boldsymbol{h}^2} = W^{3\,T}(p(c|\boldsymbol{x}) - \boldsymbol{y})$$

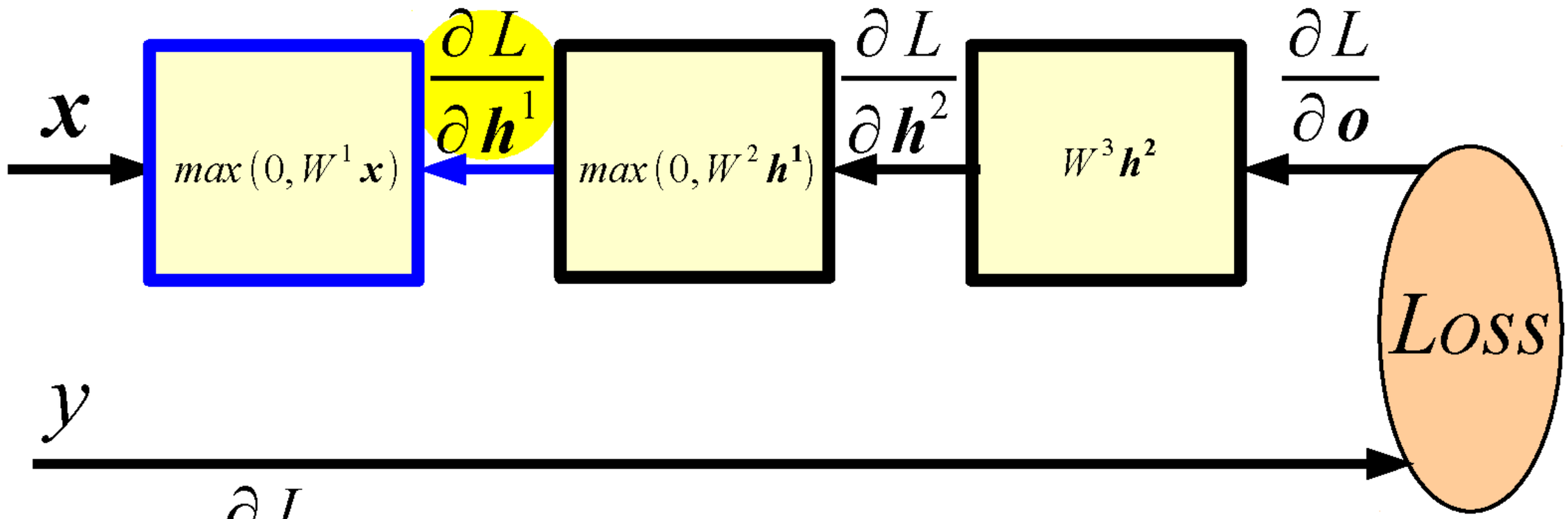
Backward Propagation



Given $\frac{\partial L}{\partial h^2}$ we can compute now:

$$\frac{\partial L}{\partial W^2} = \frac{\partial L}{\partial h^2} \frac{\partial h^2}{\partial W^2} \qquad \frac{\partial L}{\partial h^1} = \frac{\partial L}{\partial h^2} \frac{\partial h^2}{\partial h^1}$$

Backward Propagation



Given $\frac{\partial L}{\partial h^1}$ we can compute now:

$$\frac{\partial L}{\partial W^1} = \frac{\partial L}{\partial h^1} \frac{\partial h^1}{\partial W^1}$$

Backward Propagation

Question: Does BPROP work with ReLU layers only?

Answer: Nope, any a.e. differentiable transformation works.

Backward Propagation

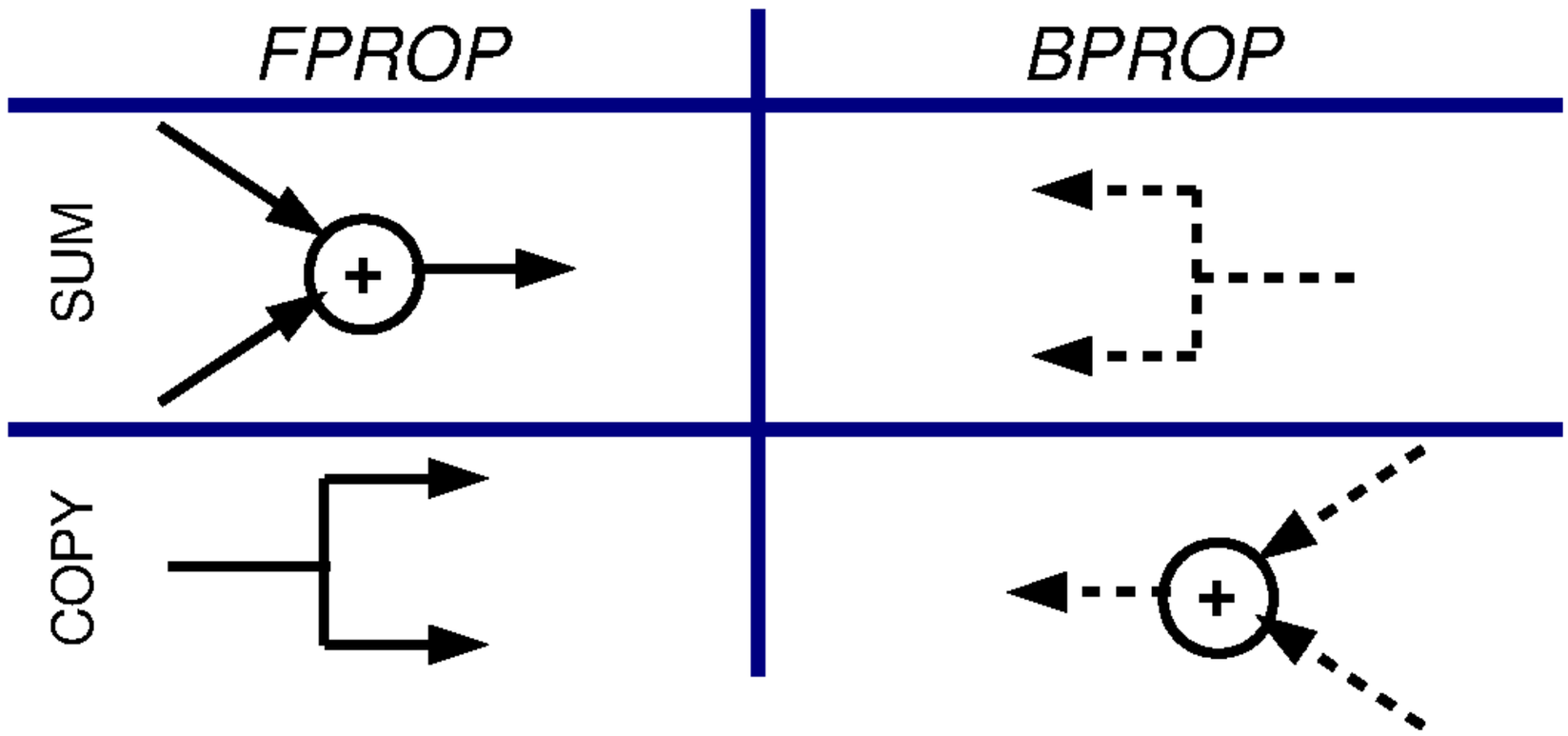
Question: Does BPROP work with ReLU layers only?

Answer: Nope, any a.e. differentiable transformation works.

Question: What's the computational cost of BPROP?

Answer: About twice FPROP (need to compute gradients w.r.t. input and parameters at every layer).

Note: FPROP and BPROP are dual of each other. E.g.,:



Toy Code (Matlab): Neural Net Trainer

```
% F-PROP
for i = 1 : nr_layers - 1
  [h{i}  jac{i}]  =  nonlinearity(W{i} * h{i-1} +  b{i});
end
h{nr_layers-1}  =  W{nr_layers-1} * h{nr_layers-2}  +  b{nr_layers-1};
prediction  =  softmax(h{l-1});


% CROSS ENTROPY LOSS
loss  =  -  sum(sum(log(prediction)  .*  target)) / batch_size;
```

```
% B-PROP
dh{l-1}  =  prediction  -  target;
for i = nr_layers - 1 : -1 : 1
  Wgrad{i}  =  dh{i} * h{i-1}';
  bgrad{i}  =  sum(dh{i}, 2);
  dh{i-1}  =  (W{i}' * dh{i})  .*  jac{i-1};
end


% UPDATE
for i = 1 : nr_layers - 1
  W{i}  =  W{i}  -  (lr / batch_size)  *  Wgrad{i};
  b{i}  =  b{i}  -  (lr / batch_size)  *  bgrad{i};
end
```

随机梯度下降（Stochastic Gradient Descent）

- Dataset can be too large to strictly apply gradient descent.
- Instead, randomly sample a data point, perform gradient descent per point, and iterate.
  - True gradient is approximated only
  - Picking a subset of points: *"mini-batch"*

Pick starting W and learning rate γ

While not at minimum:

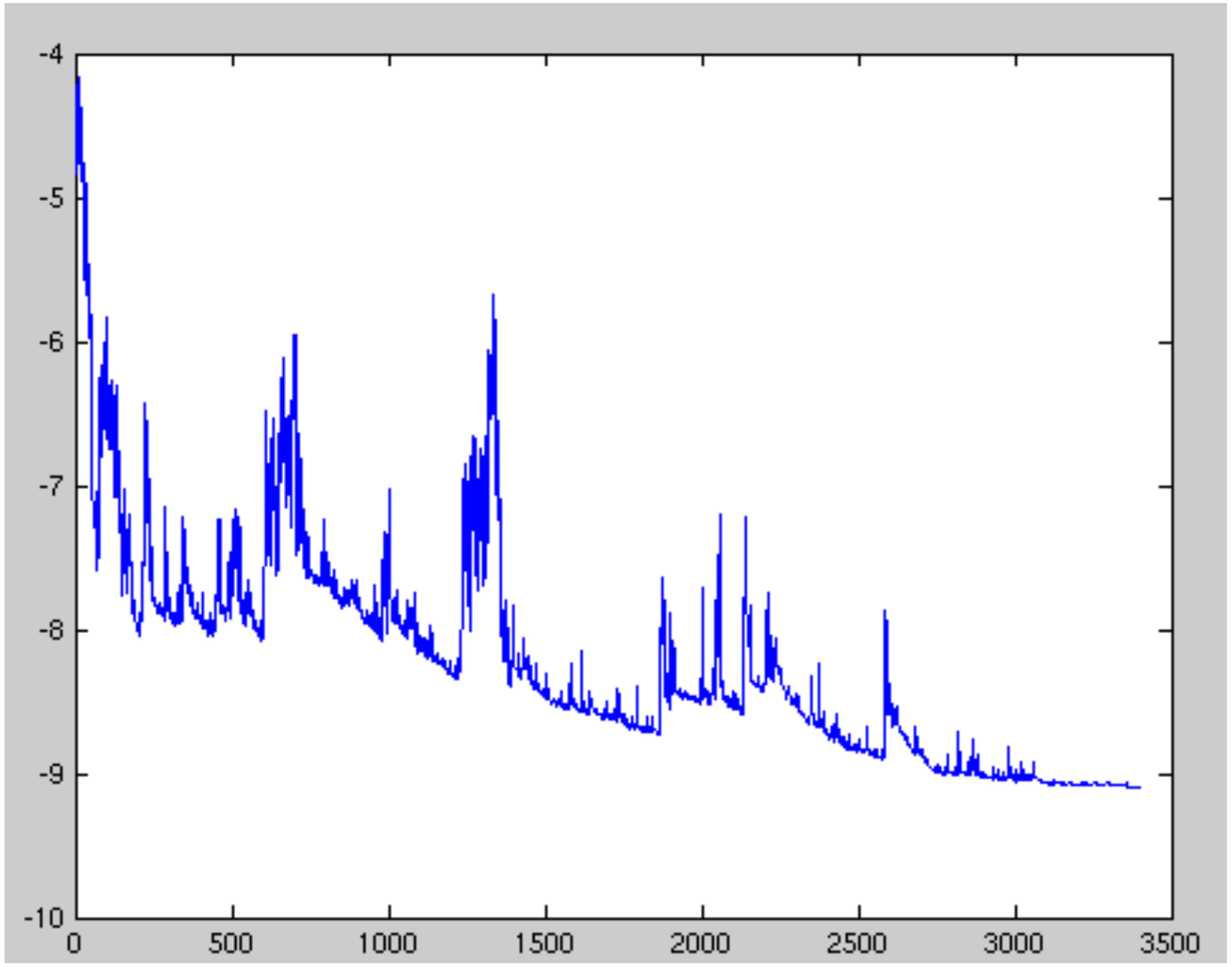
- Shuffle training set
- For each data point *i=1…n (maybe as mini-batch)*
  - *Gradient descent*

} *"Epoch"*

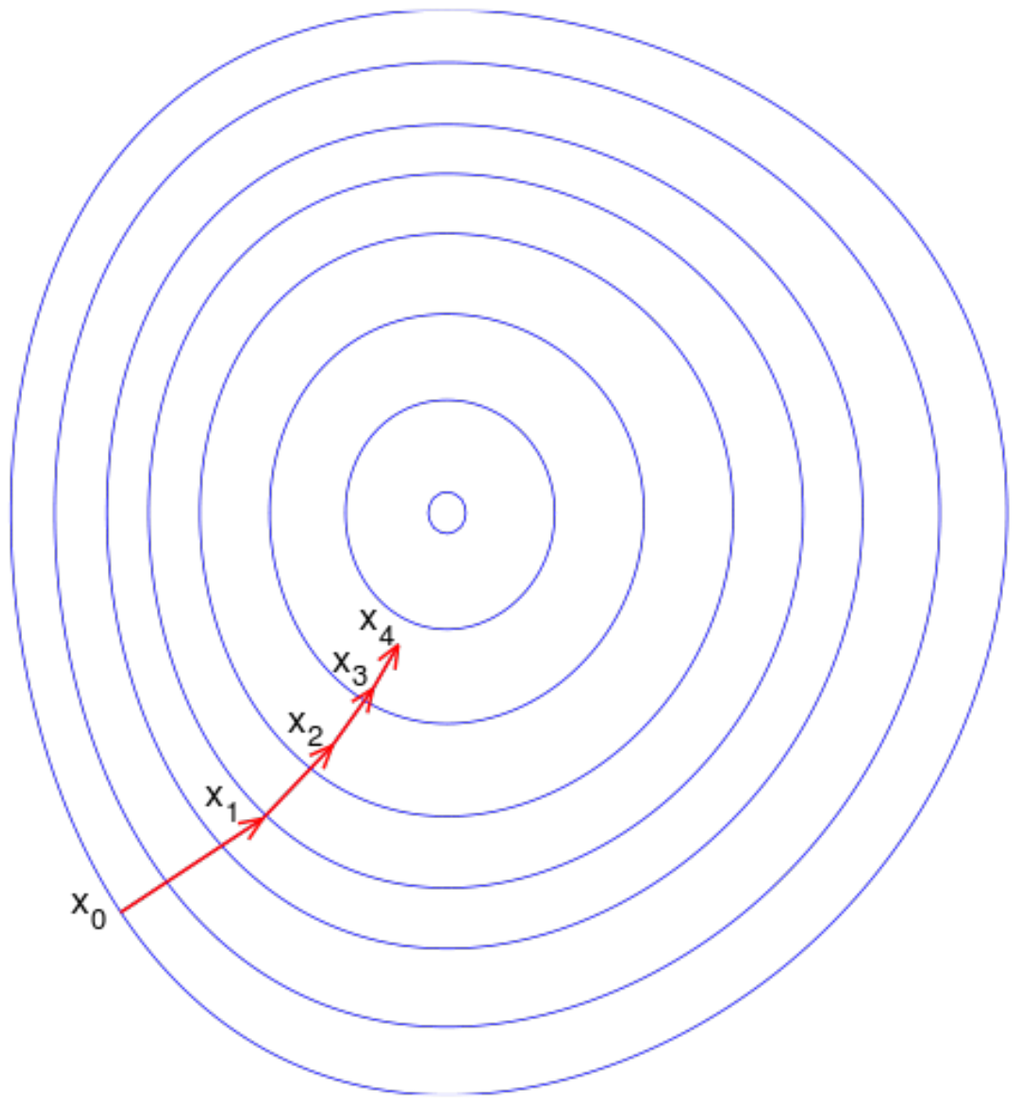
随机梯度下降（Stochastic Gradient Descent）

Loss will not always decrease (locally) as training data point is random.

Still converges over time.

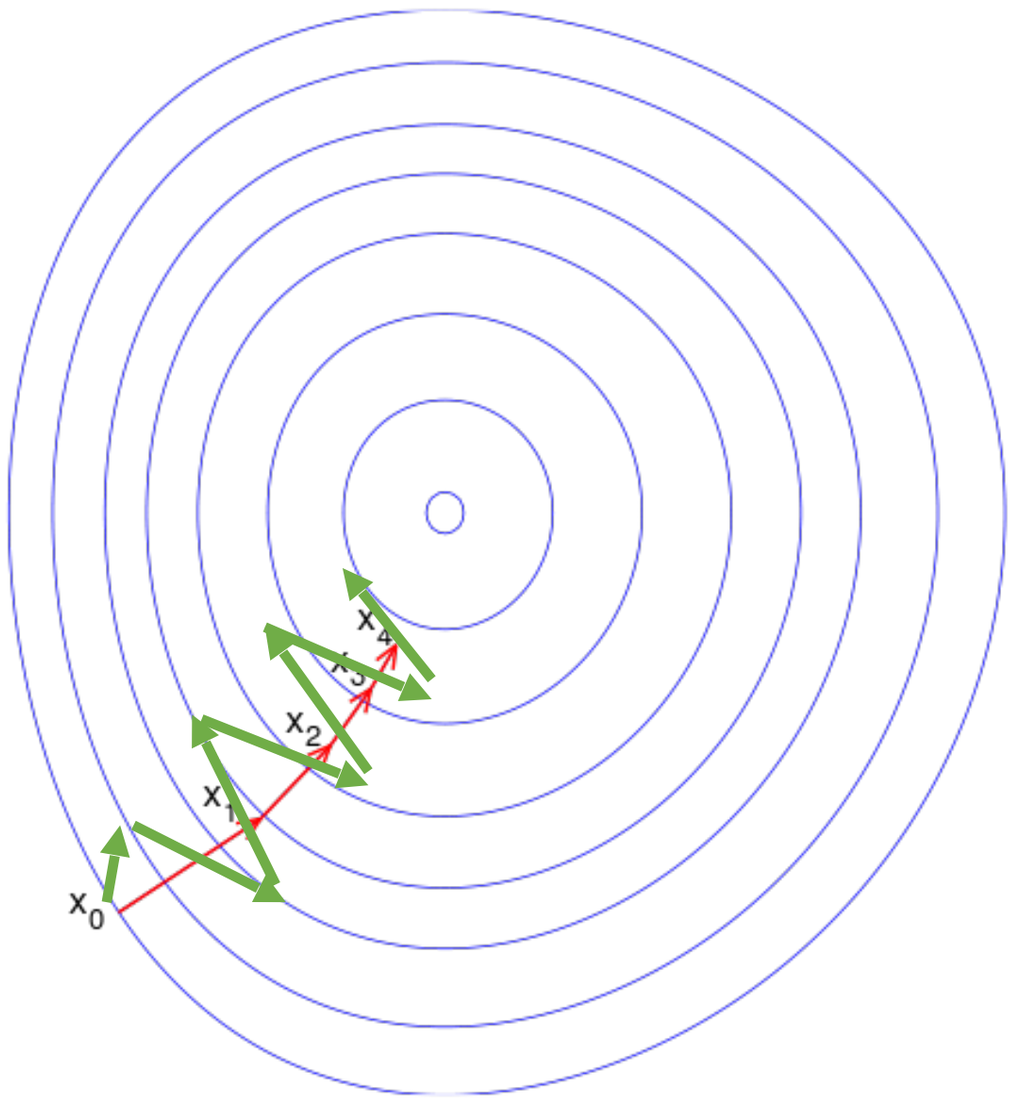


Gradient descent oscillations



Wikipedia

Gradient descent oscillations



Slow to converge to the (local) optimum

Wikipedia

Momentum

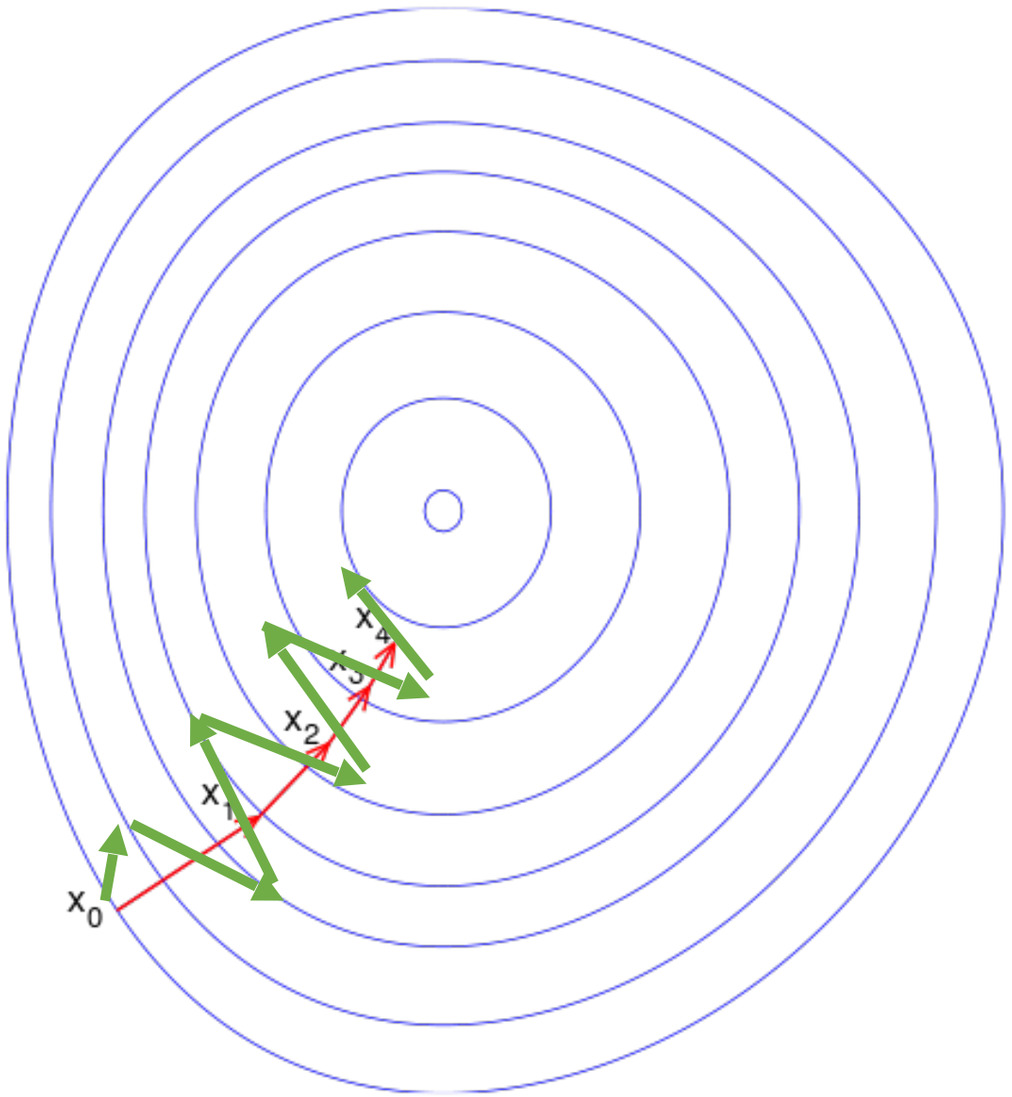
- Adjust the gradient by a weighted sum of the previous amount plus the current amount.

- Without momentum: $\theta_{t+1} = \theta_t - \gamma \frac{\partial L}{\partial \theta}$

- With momentum (new α parameter):

$$\theta_{t+1} = \theta_t - \gamma \left(\alpha \left[\frac{\partial L}{\partial \theta} \right]_{t-1} + \left[\frac{\partial L}{\partial \theta} \right]_t \right)$$

Lowering the learning rate = smaller steps in SGD

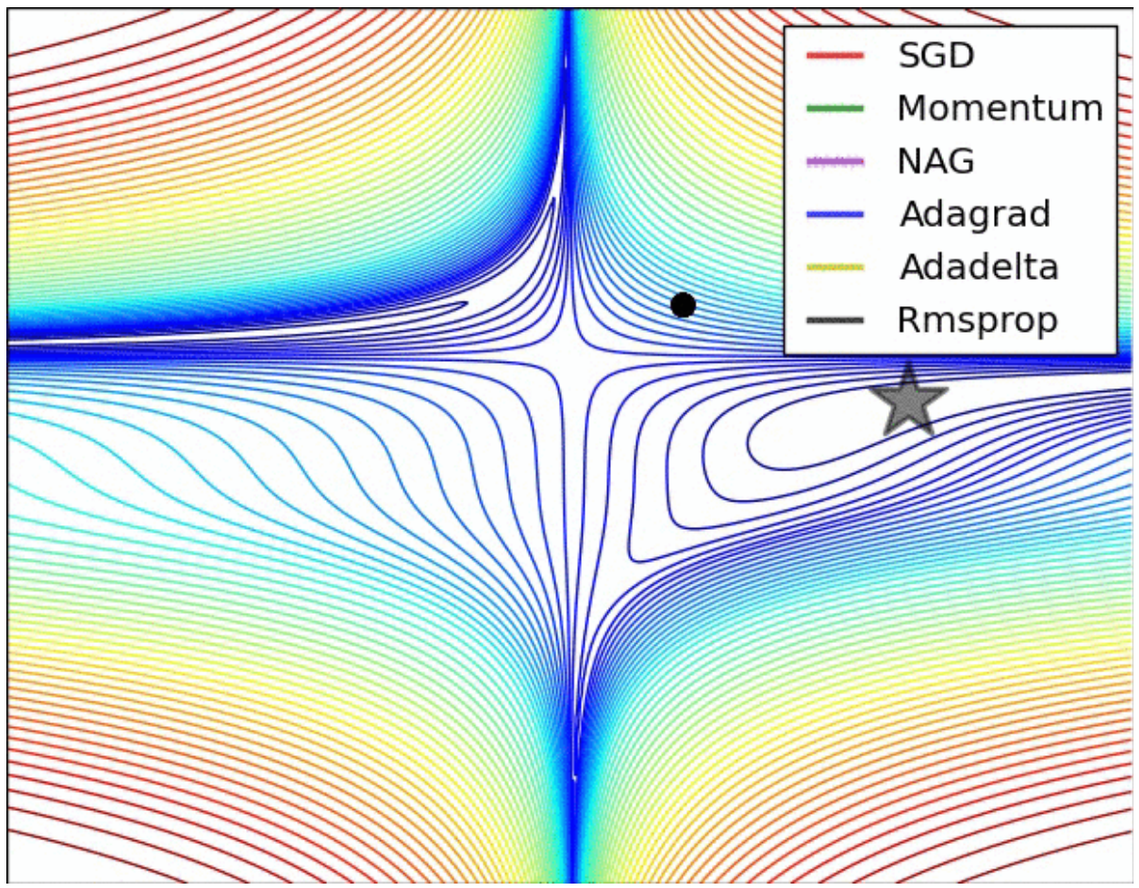


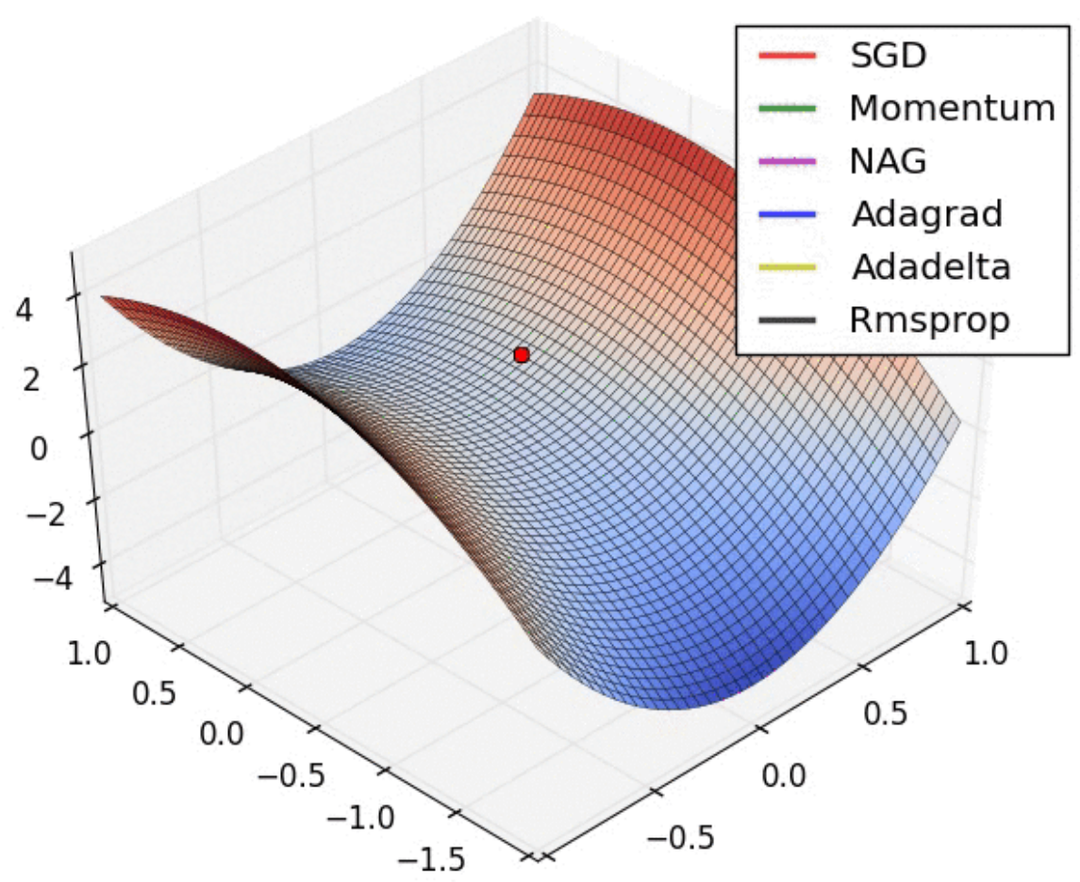
-Less ‘ping pong’

-Takes longer to get to the optimum

Wikipedia

Flat regions in energy landscape





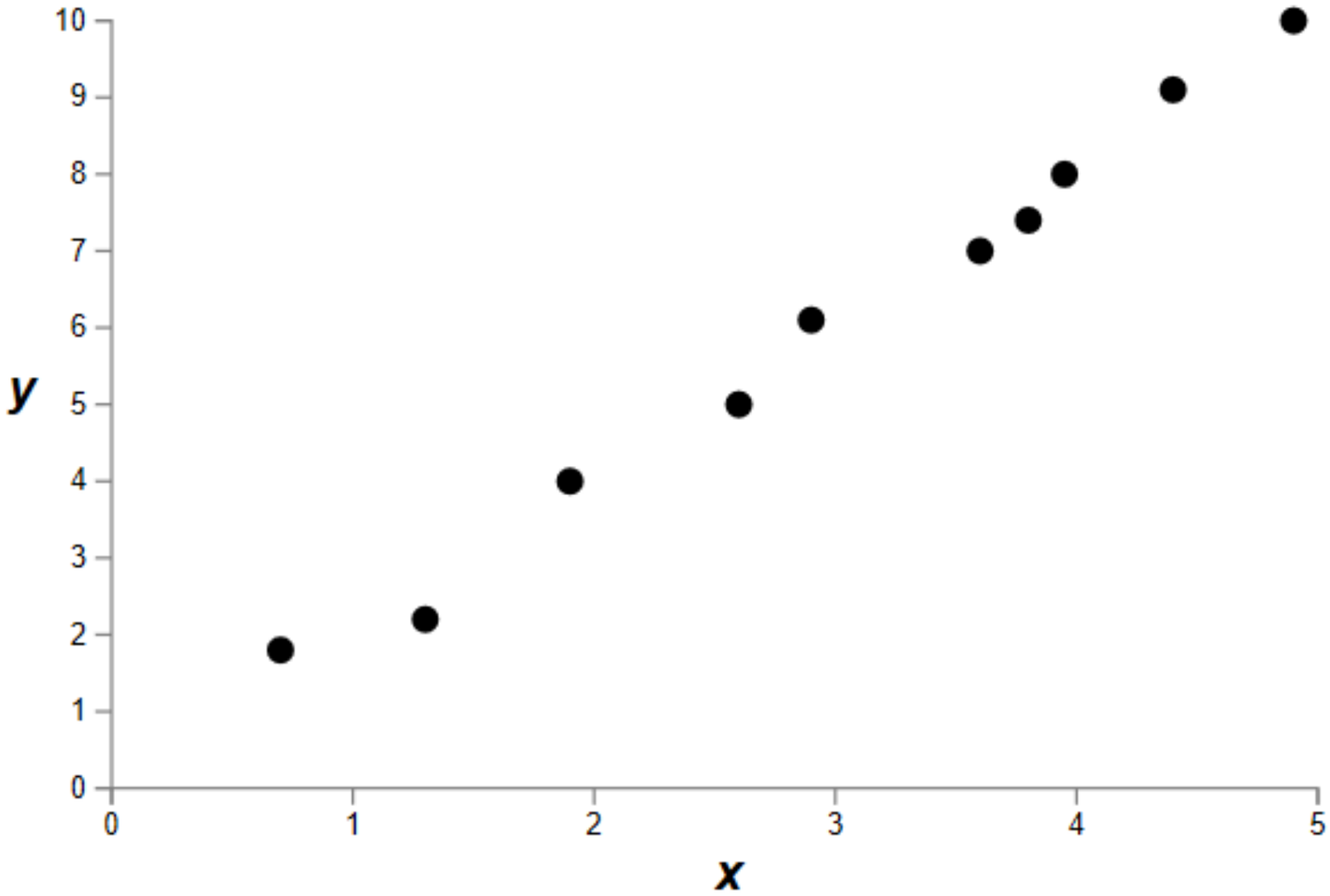
Problem of fitting

- Too many parameters = overfitting
- Not enough parameters = underfitting
- More data = less chance to overfit
- How do we know what is required?

Regularization

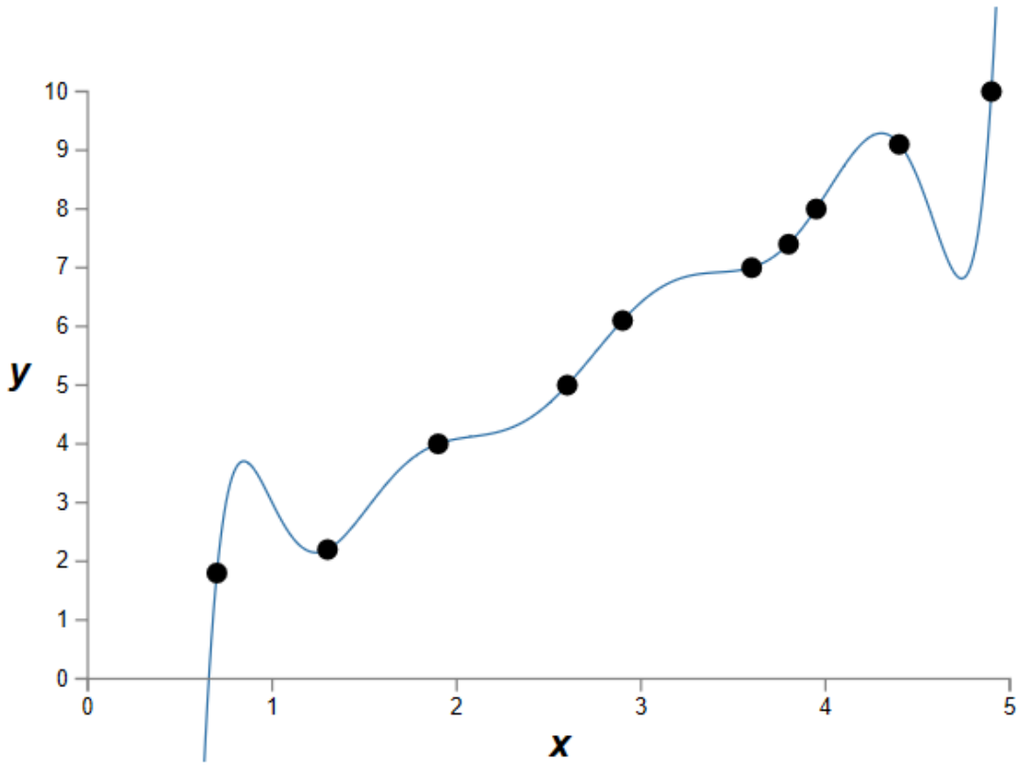
- Attempt to guide solution to *not overfit*
- But still give freedom with many parameters

Data fitting problem

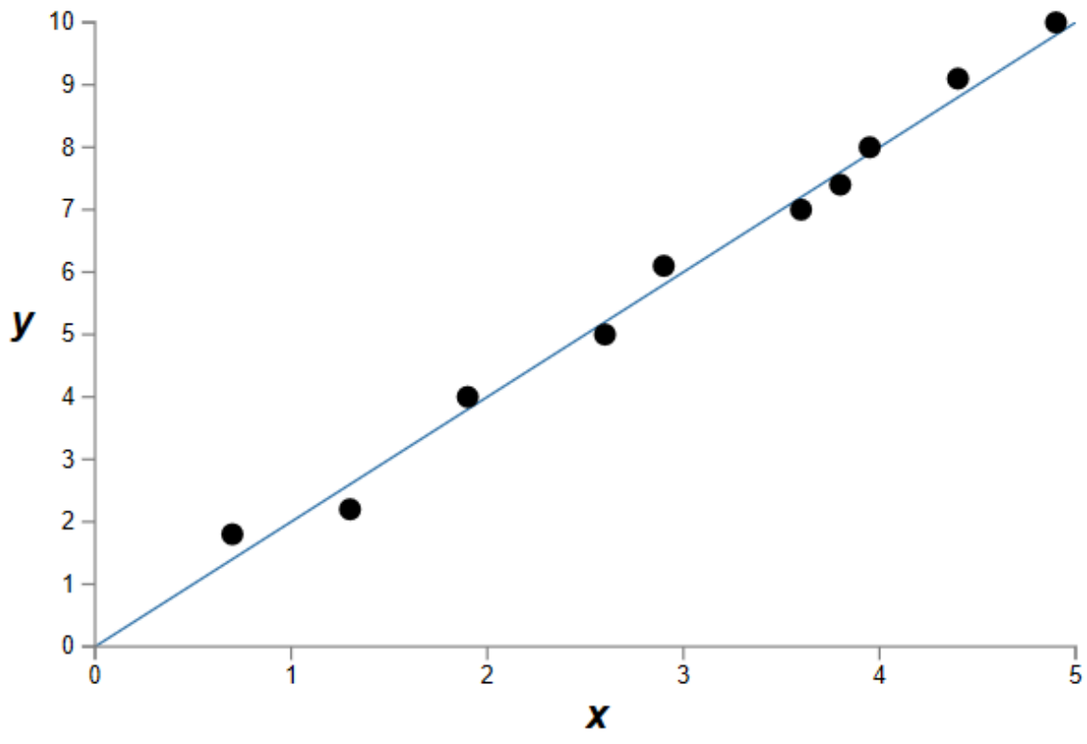


[Nielson]

Which is better?
Which is better a priori?



9th order polynomial



1st order polynomial

[Nielson]

Regularization

- Attempt to guide solution to *not overfit*
- But still give freedom with many parameters

- Idea:
  Penalize the use of parameters to prefer small weights.

Regularization:

- Idea: add a cost to having high weights
- λ = regularization parameter

$$C = C_0 + \lambda \sum_w w^2,$$

[Nielson]

Both can describe the data…

- …but one is simpler.

- Occam's razor:

  "Among competing hypotheses, the one with the fewest assumptions should be selected"

For us:

Large weights cause large changes in behavior in response to small changes in the input.

Simpler models (or smaller changes) are more robust to noise.

Regularization

- Idea: add a cost to having high weights
- λ = regularization parameter

$$C = C_0 + \lambda \sum_w w^2,$$

$$C = -\frac{1}{n} \sum_{xj} \left[y_j \ln a_j^L + (1 - y_j) \ln(1 - a_j^L) \right] + \lambda \sum_w w^2.$$

Normal cross-entropy loss (binary classes)

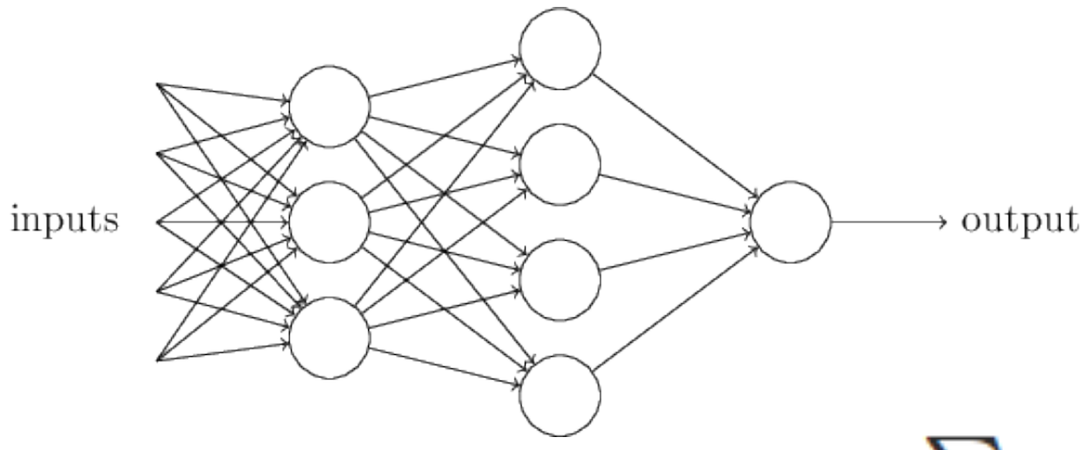
Regularization term

[Nielson]

Regularization: Dropout

- Our networks typically start with random weights.
- Every time we train = slightly different outcome.

- Why random weights?
- If weights are all equal, response across filters will be equivalent.
  - Network doesn't train.



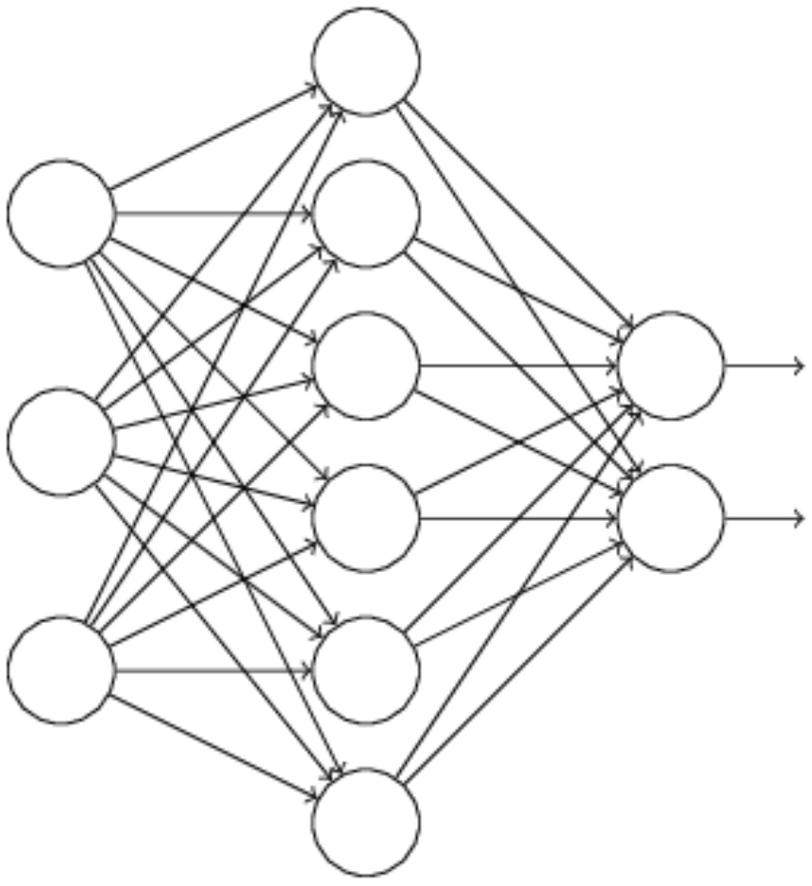
$$w \cdot x \equiv \sum_j w_j x_j$$

[Nielson]

Regularization

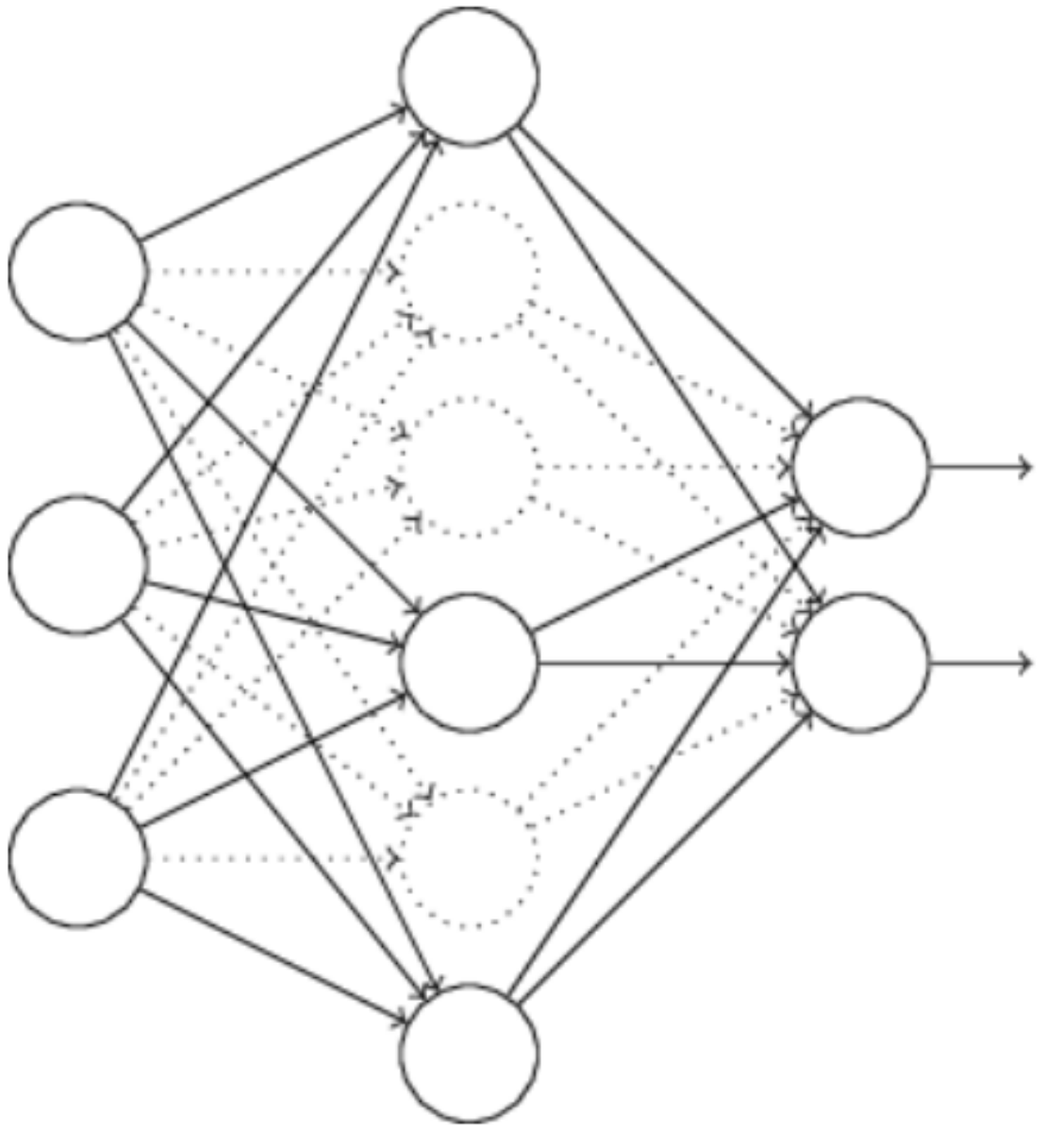
- Our networks typically start with random weights.
- Every time we train = slightly different outcome.
- Why not train 5 different networks with random starts and vote on their outcome?
  - Works fine!
  - Helps generalization because error is averaged.

Regularization: Dropout



[Nielson]

Regularization: Dropout



At each mini-batch:

- Randomly select a subset of neurons.
- Ignore them.

Effect:

- Neurons become less dependent on output of connected neurons.
- Forces network to learn more robust features that are useful to more subsets of neurons.
- Like averaging over many different trained networks with different random initializations.
- Except cheaper to train.

[Nielson]

Many forms of ‘regularization’

- Adding more data is a kind of regularization
- Pooling is a kind of regularization
- Data augmentation is a kind of regularization