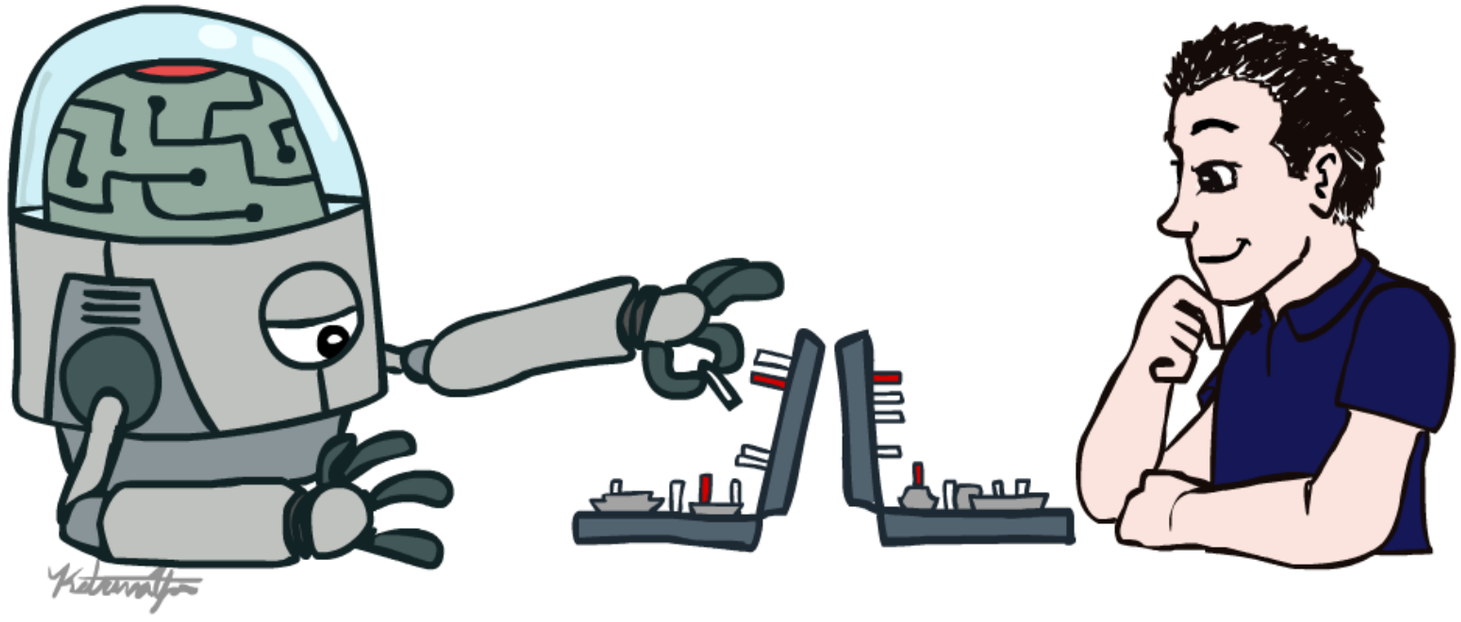
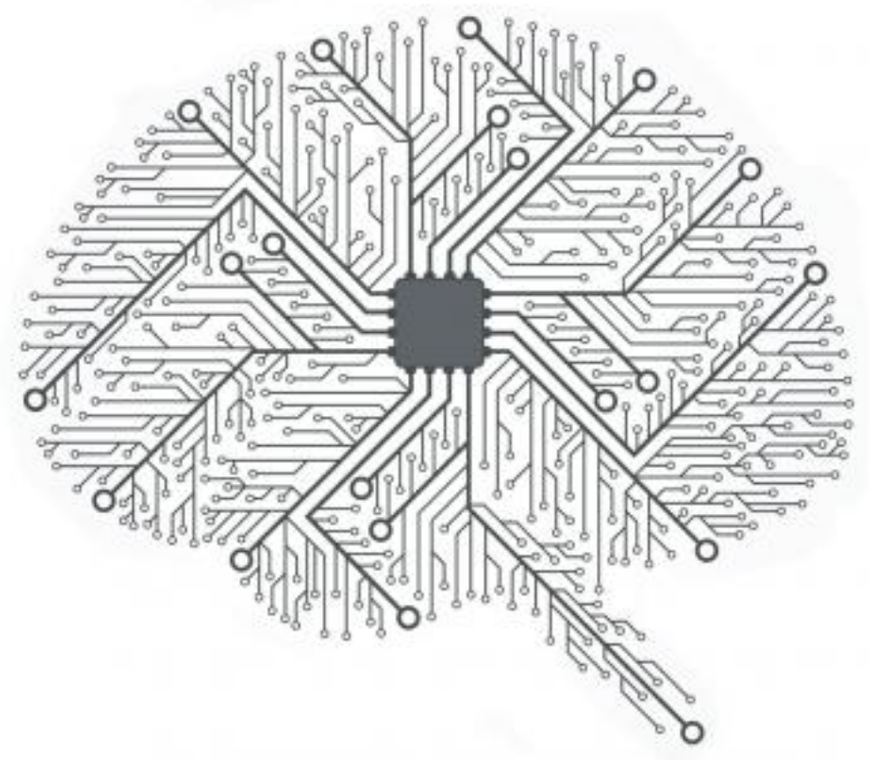
人工智能

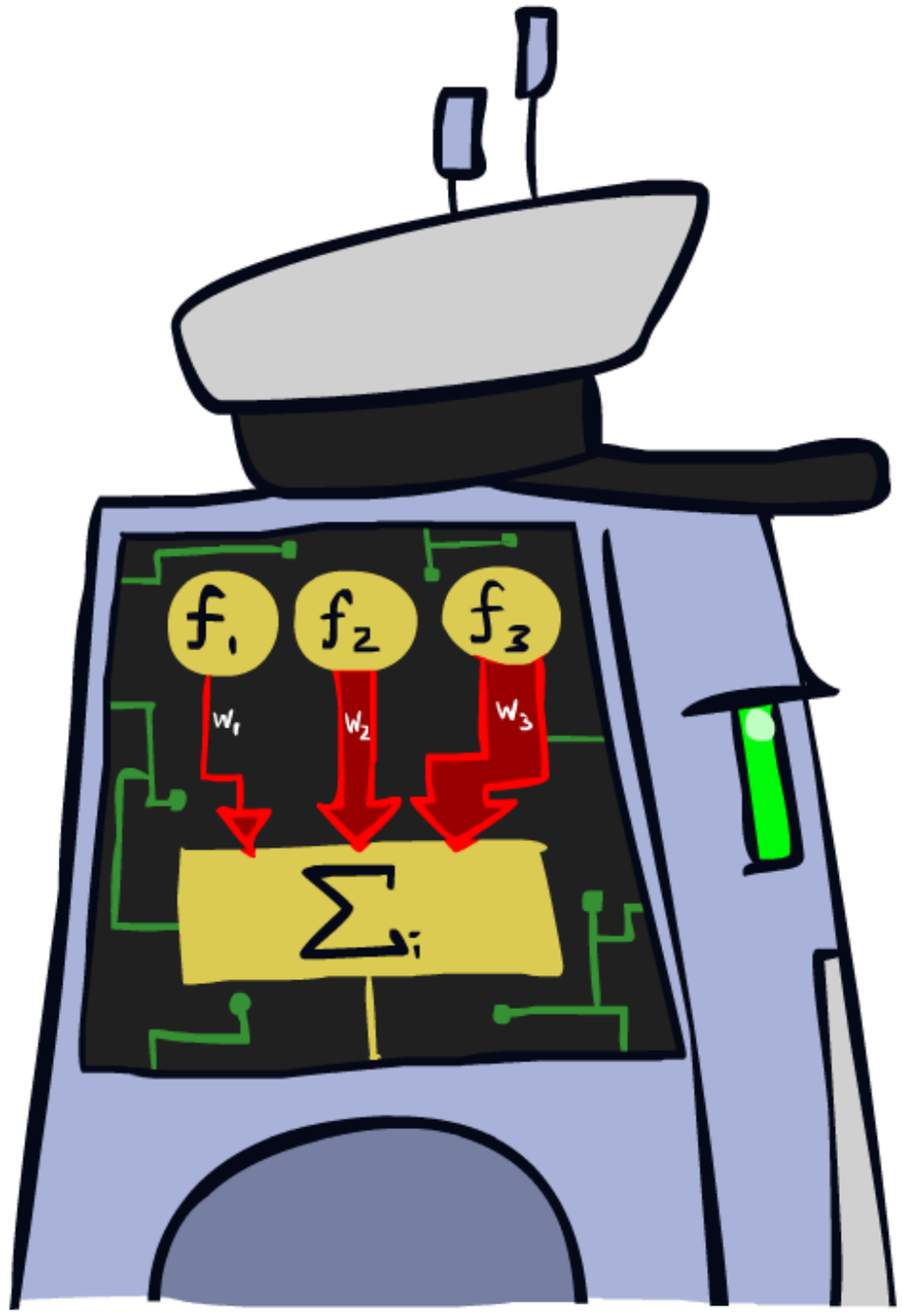


第九章·感知机与神经网络

- 线性分类器
- 感知机
- 逻辑回归
- 神经网络



线性分类器



特征向量

x $f(x)$ y

```
Hello,

Do you want free printr
cartriges?  Why pay more
when you can get them
ABSOLUTELY FREE!  Just
```

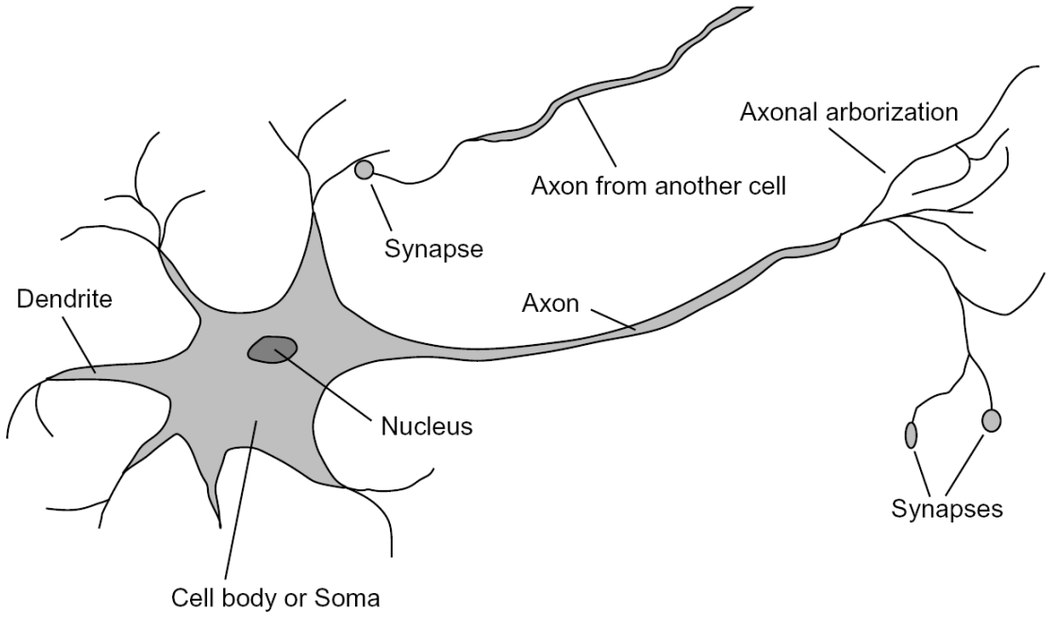
```
# free       : 2
YOUR_NAME    : 0
MISSPELLED   : 2
FROM_FRIEND  : 0
...
```

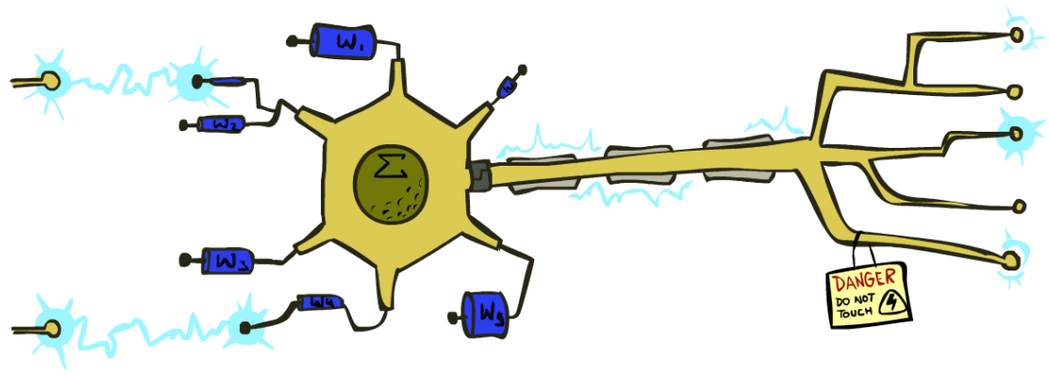
SPAM
or
HAM

```
PIXEL-7,12   : 1
PIXEL-7,13   : 0
...
NUM_LOOPS    : 1
...
```

“2”

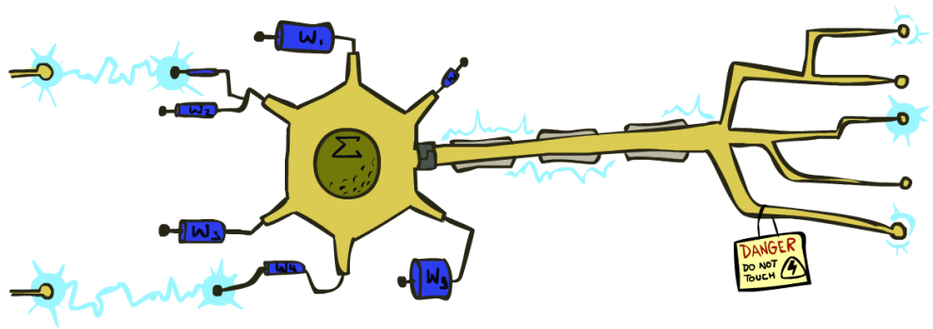
人类神经元


Axonal arborization
Axon from another cell
Synapse
Dendrite
Axon
Nucleus
Synapses
Cell body or Soma


DANGER
DO NOT TOUCH

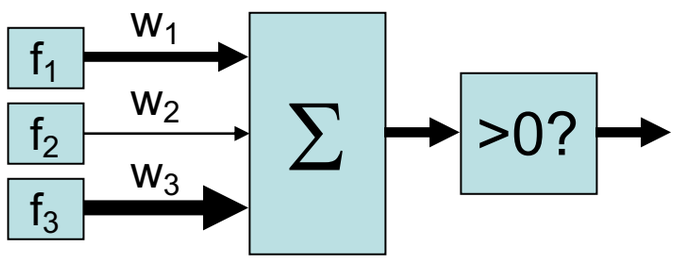
线性分类器

- 输入是特征值
- 每个特征都有权重
- 激活函数是求和

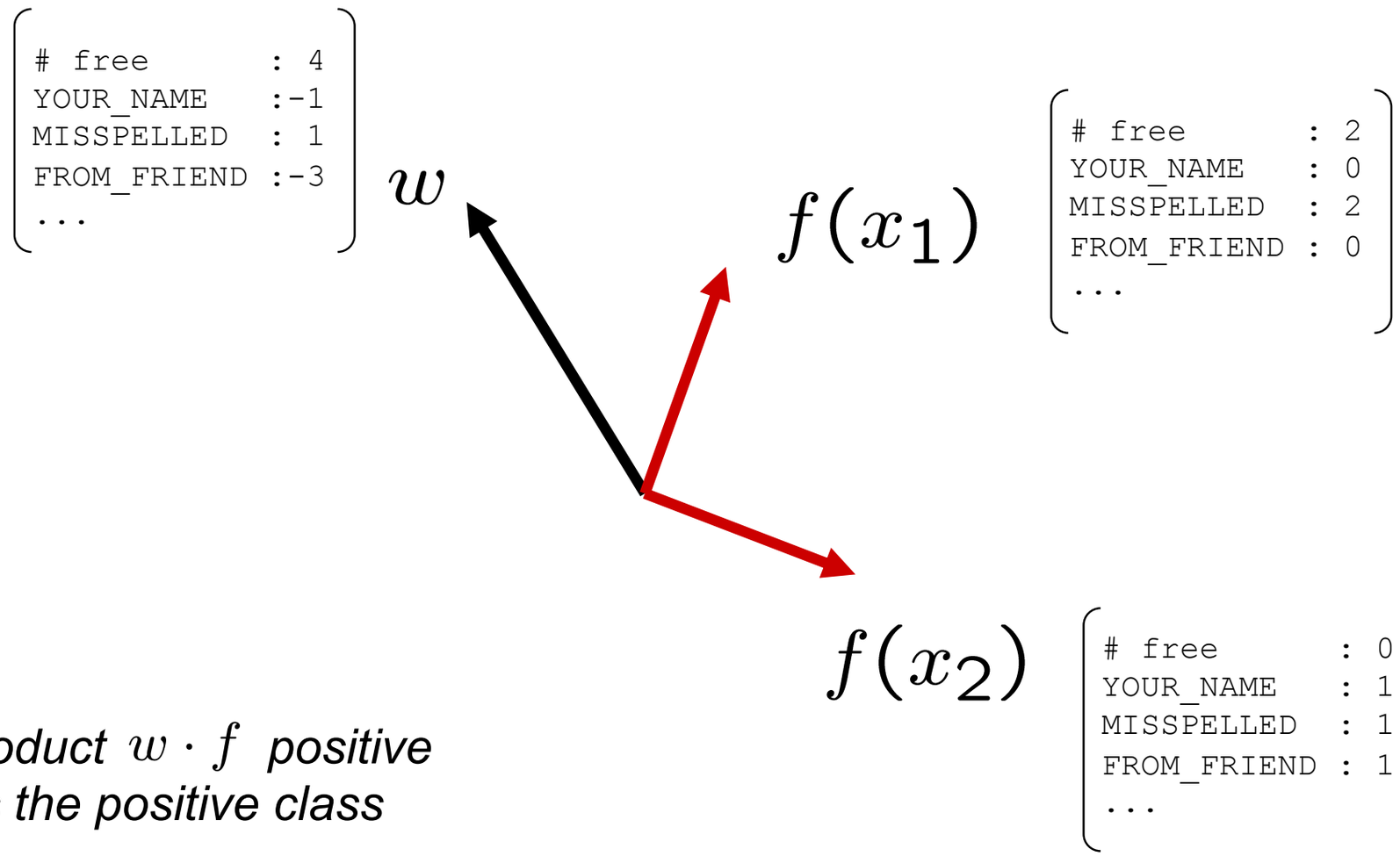


$$\text{activation}_w(x) = \sum_i w_i \cdot f_i(x) = w \cdot f(x)$$

- 如果激活函数的值是:
  - 正数, 那么输出 +1
  - 负数, 那么输出 -1



权重

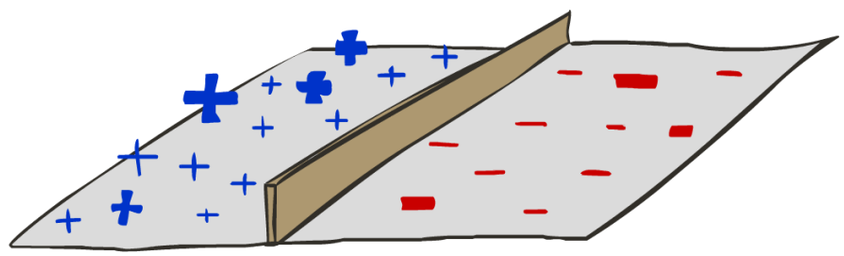


Dot product $w \cdot f$ positive means the positive class

- 学习：从训练样本中得到合适的权重向量

二元决策

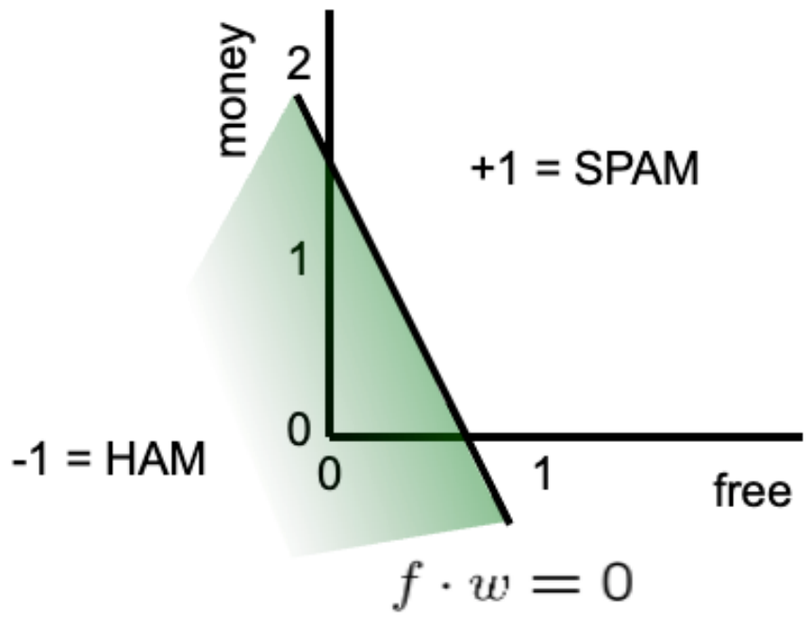
- 在特征向量空间中
  - 学习样本是一些点
  - 权重向量确定了一个超平面$\omega^T \cdot f(x)=0$
  - 这个超平面的一个侧面对应Y=+1



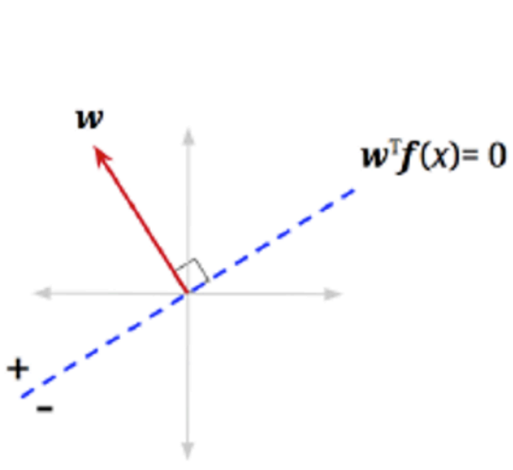
  - 另一个对应于Y=–1

$$w$$

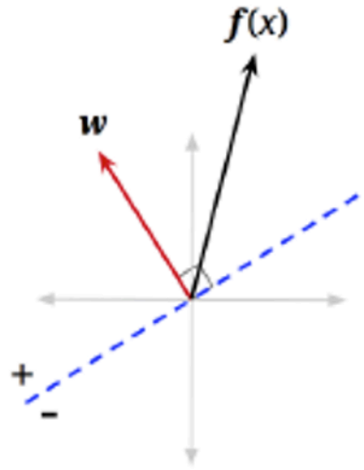
```
BIAS   : -3
free   :  4
money  :  2
...
```



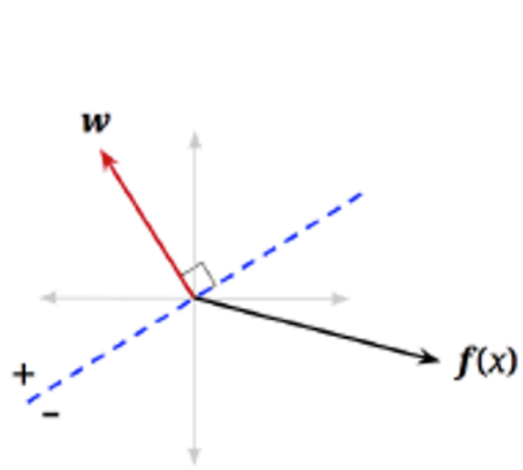
二元决策



Decision Boundary



***x* classified into positive class**

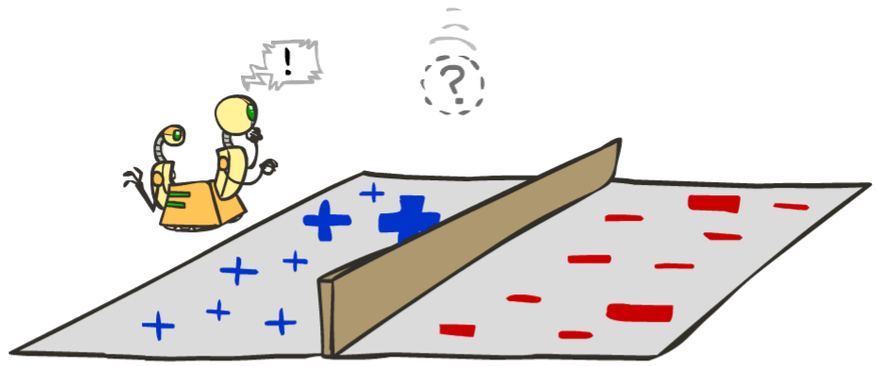


***x* classified into negative class**

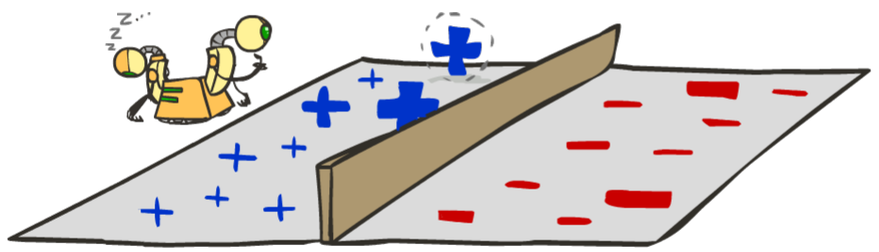
权重学习

学习：二进制感知机

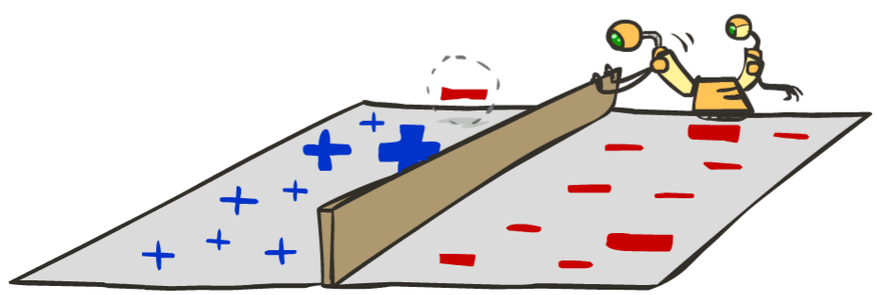
- 开始时权重向量`weights = 0`
- 对每个训练样本:
  - 按当前权重分类



  - 如果正确(y=y*)，不作处理!



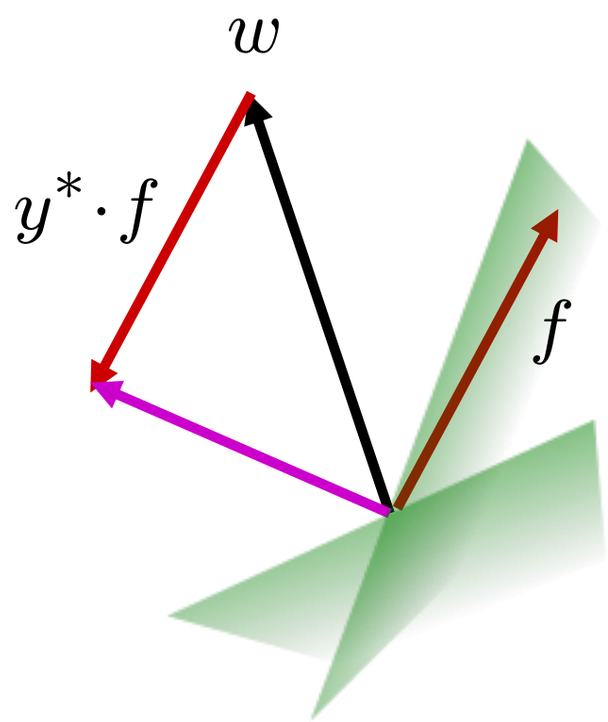
  - 如果错误: 修正权重向量



学习：二进制感知机

- 开始时 权重向量`weights = 0`
- 对每个训练样本:
  - 按当前权重分类

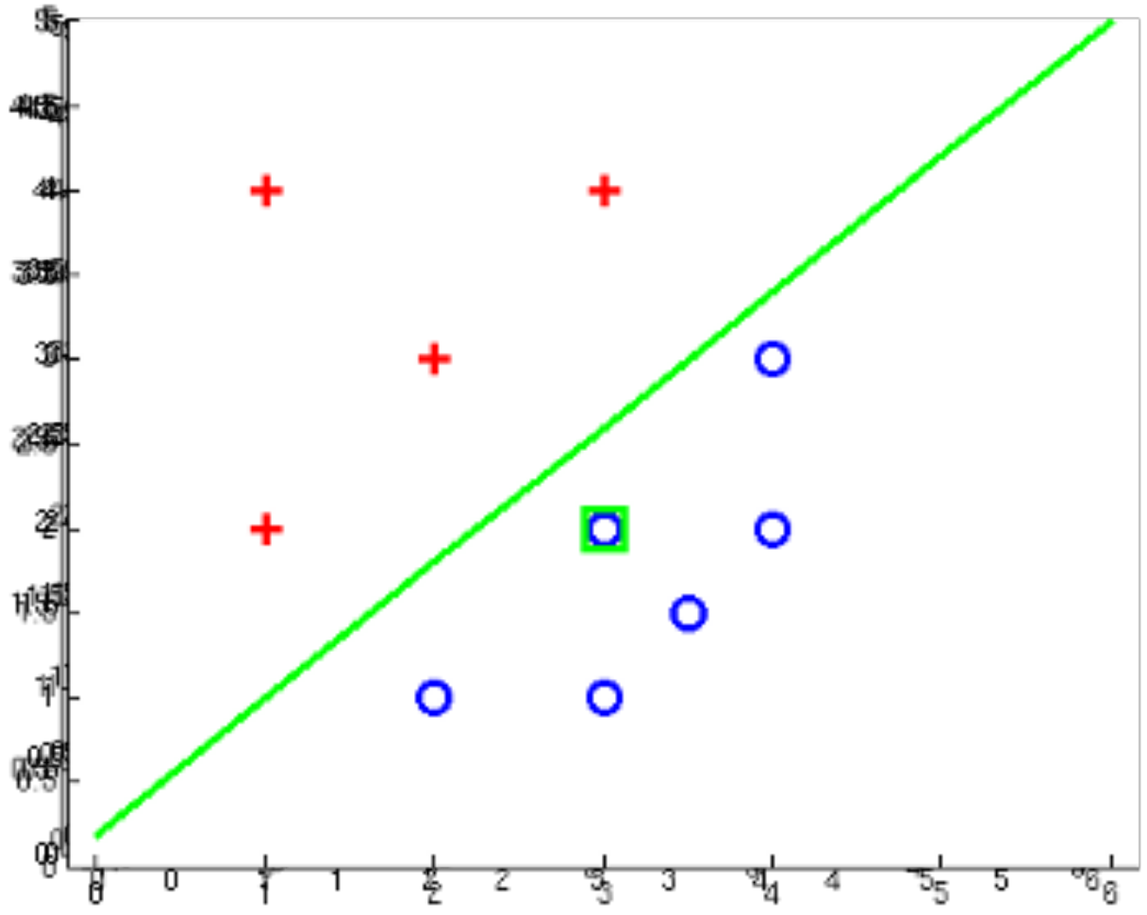
$$y = \begin{cases} +1 & \text{if } w \cdot f(x) \geq 0 \\ -1 & \text{if } w \cdot f(x) < 0 \end{cases}$$



  - 如果正确（y=y*），不作处理!
  - 如果错误: 权重向量加上或减去特征向量

$$w = w + y^* \cdot f$$

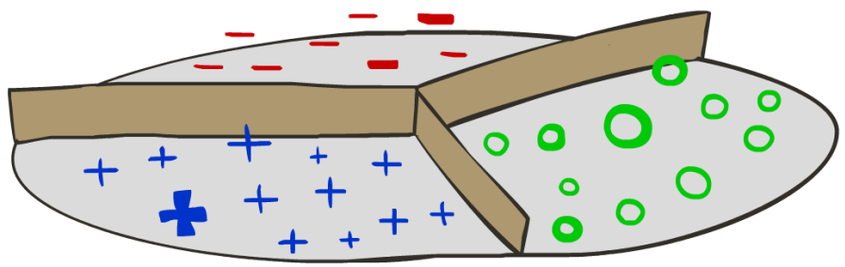
示例：感知机



多类别决策

- 如果有多个类别
  - 每个类的权重向量：

$$w_y$$

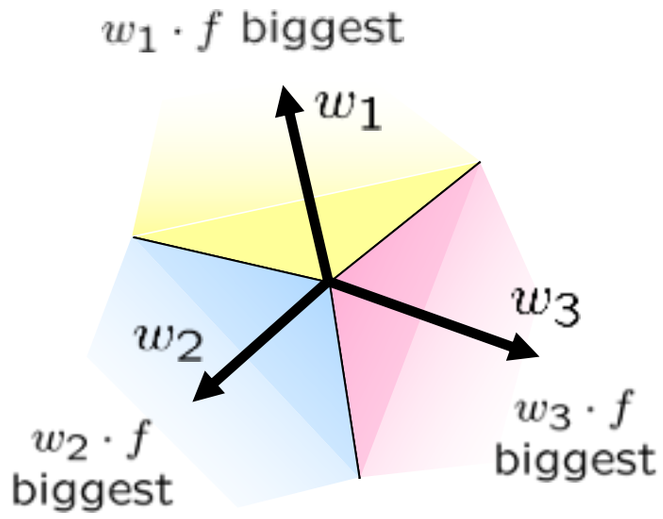


  - 计算样本在每个y类的得分：

$$w_y \cdot f(x)$$

  - 将样本判断为最高得分的类别

$$y = \arg\max_y \ w_y \cdot f(x)$$

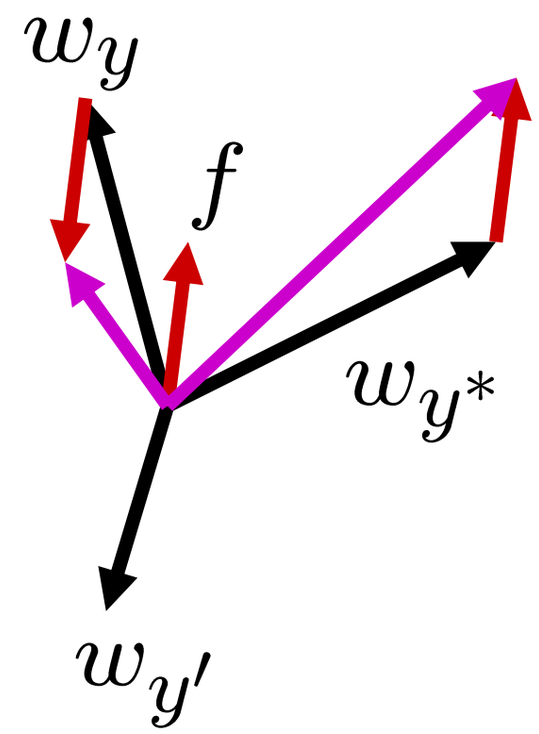


Binary = multiclass where the negative class has weight zero

学习：多类别感知机

- 开始时 所有的权重向量weights=0
- 对每个训练样本:
  - 用当前权重进行预测

$$y = \arg\max_y \; w_y \cdot f(x)$$



  - 如果正确, 不作处理!
  - 如果错误: 降低错误类别的分数，增加正确类别的分数

$$w_y = w_y - f(x)$$

$$w_{y^*} = w_{y^*} + f(x)$$

示例：多类别感知机

“win the vote”

“win the election”

“win the game”

w_{SPORTS}

```
win    : 0
game   : 0
vote   : 0
election : 0
the    : 0
...
```

$w_{POLITICS}$

```
win    : 0
game   : 0
vote   : 0
election : 0
the    : 0
...
```

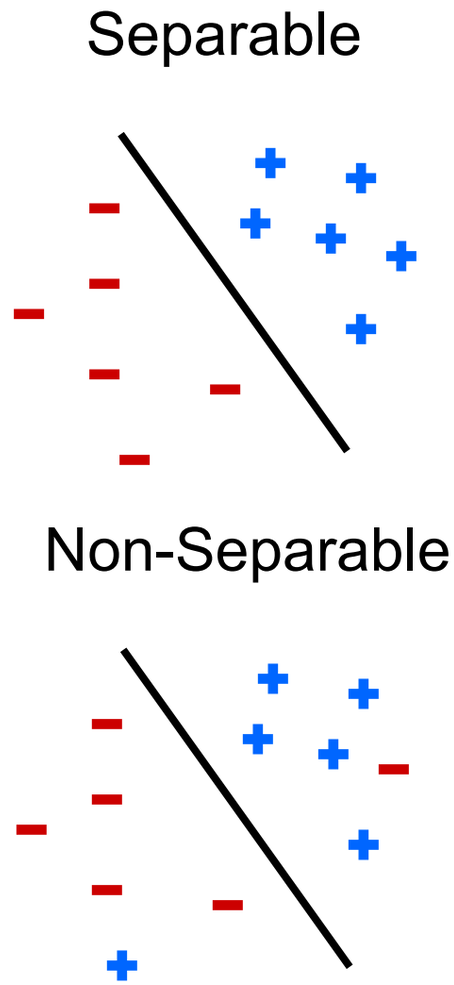
w_{TECH}

```
win    : 0
game   : 0
vote   : 0
election : 0
the    : 0
...
```

感知机的性质

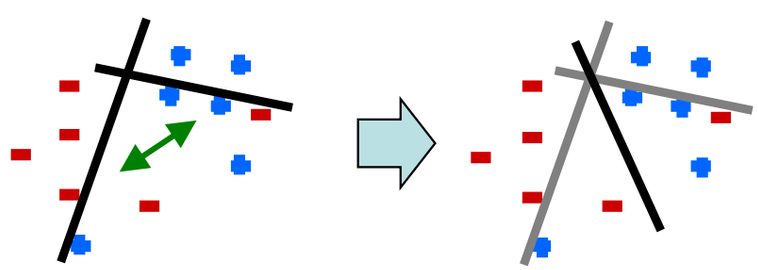
- 可分性：是否存在某些参数使训练集完全正确
- 收敛性：如果训练样本是可分的，则感知机最终会收敛（二分类情况下）
- 错误界限：在训练集上最多会犯多少次错误

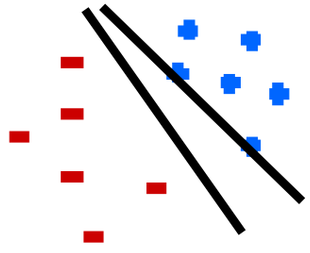
$$\text{mistakes} < \frac{k}{\delta^2}$$

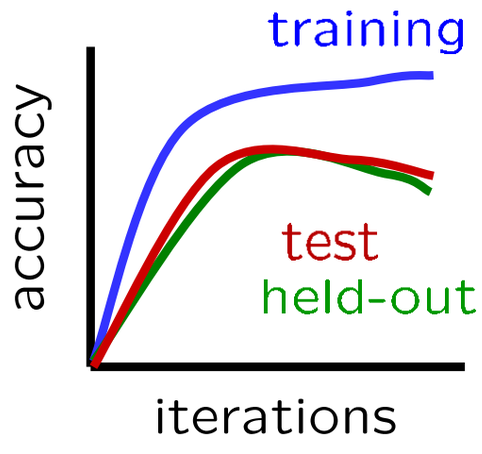


感知机的问题

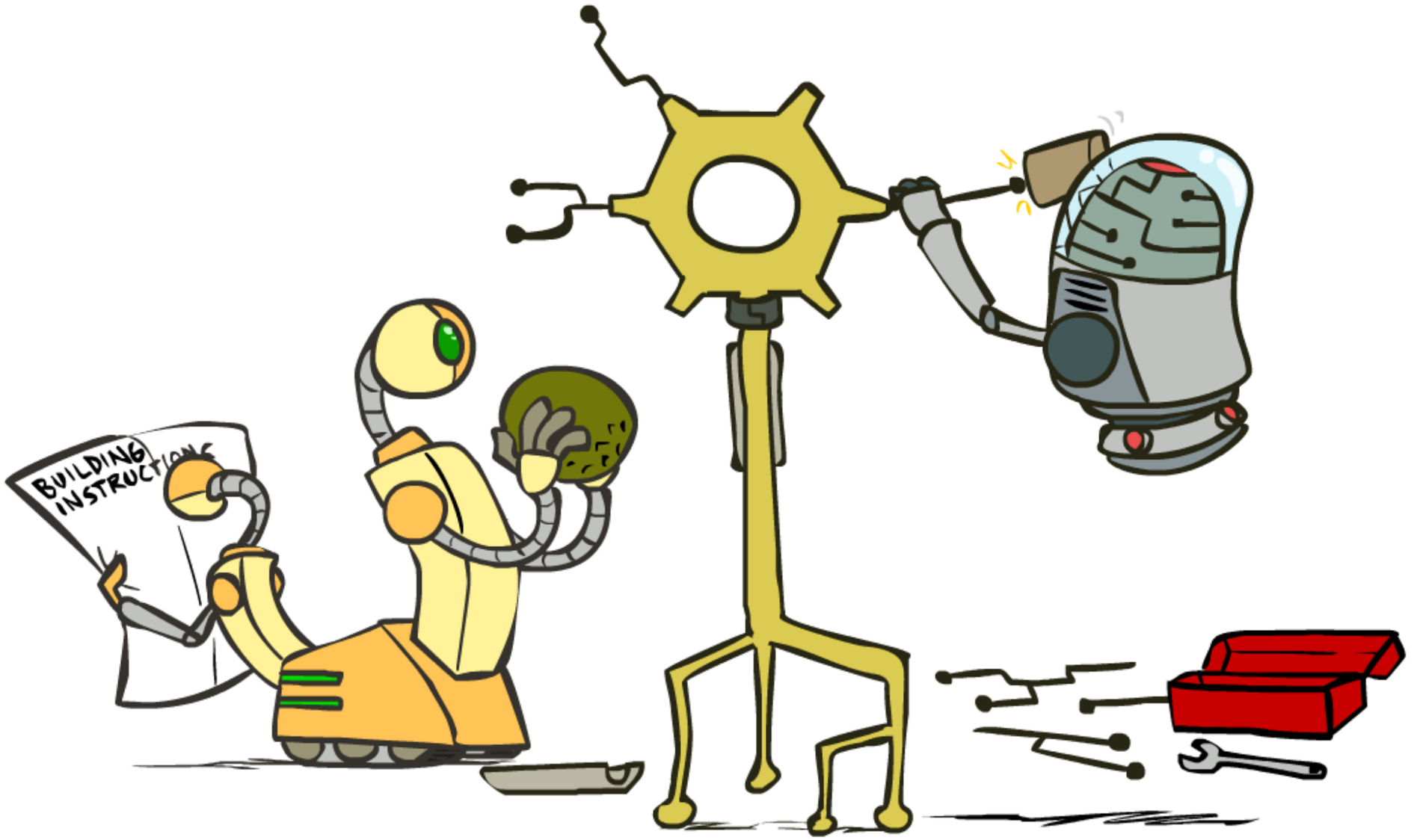
- 噪声：如果数据不可分离，权重可能会剧烈振荡:
  - 时间差分学习会降低权重的振荡
- 泛化能力差：只能找到一个把正负样本“勉强”分开的解决方案
- 过度训练：测试集/留出集的准确性通常先上升后下降
  - 过度训练是一种过拟合



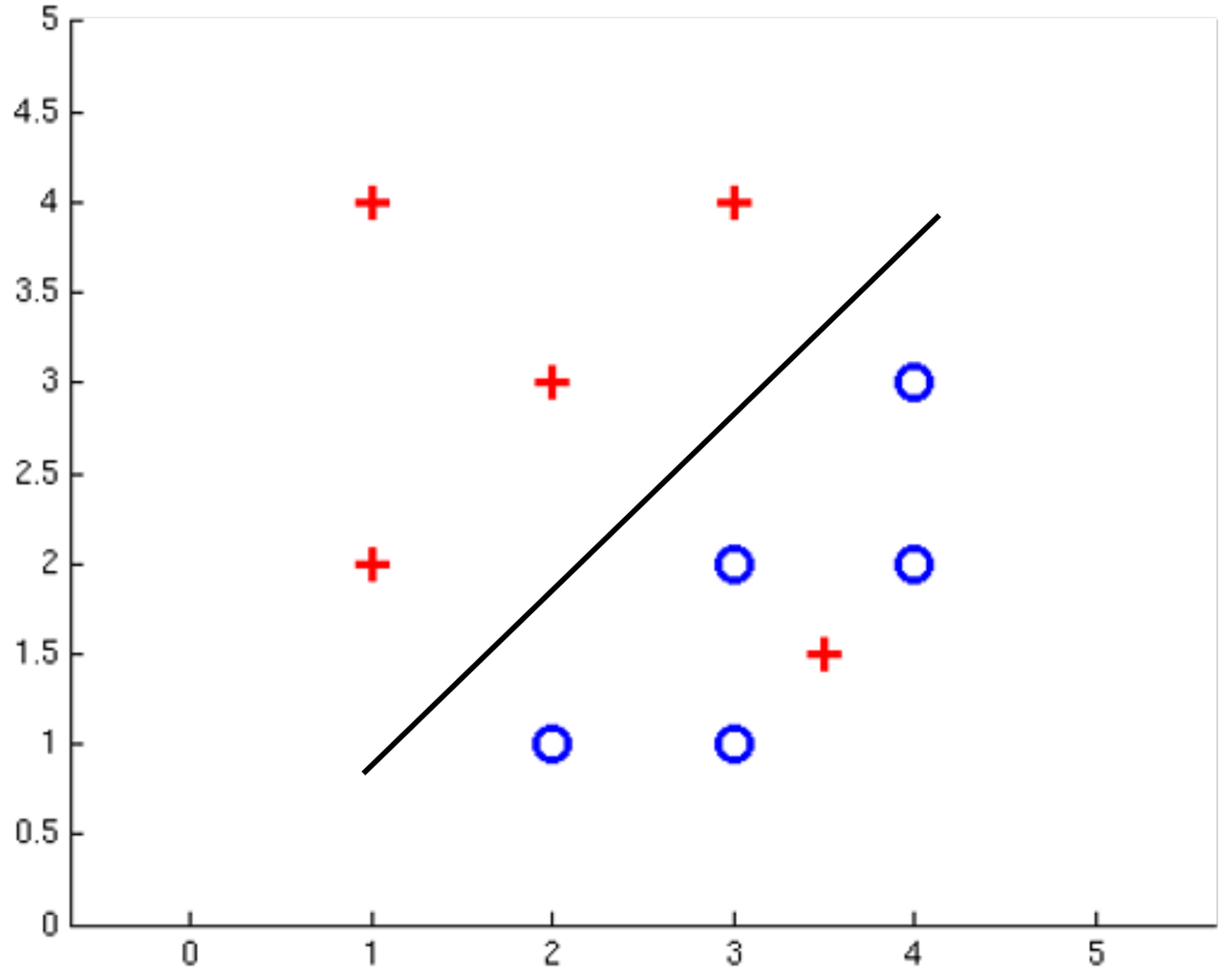




改进感知机

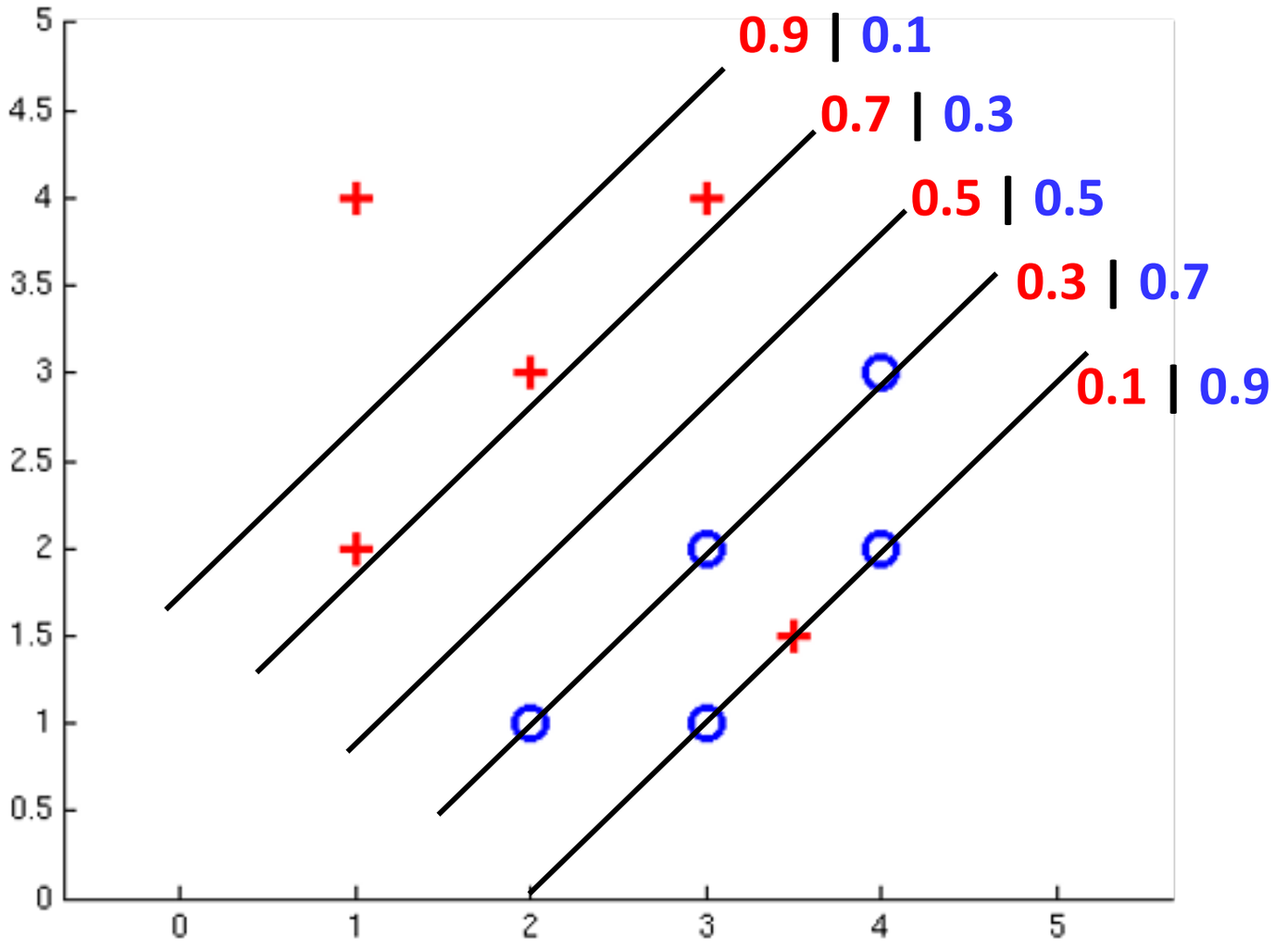


不可分情形：确定性决策



即使是最好的线性边界也至少会犯一个错误

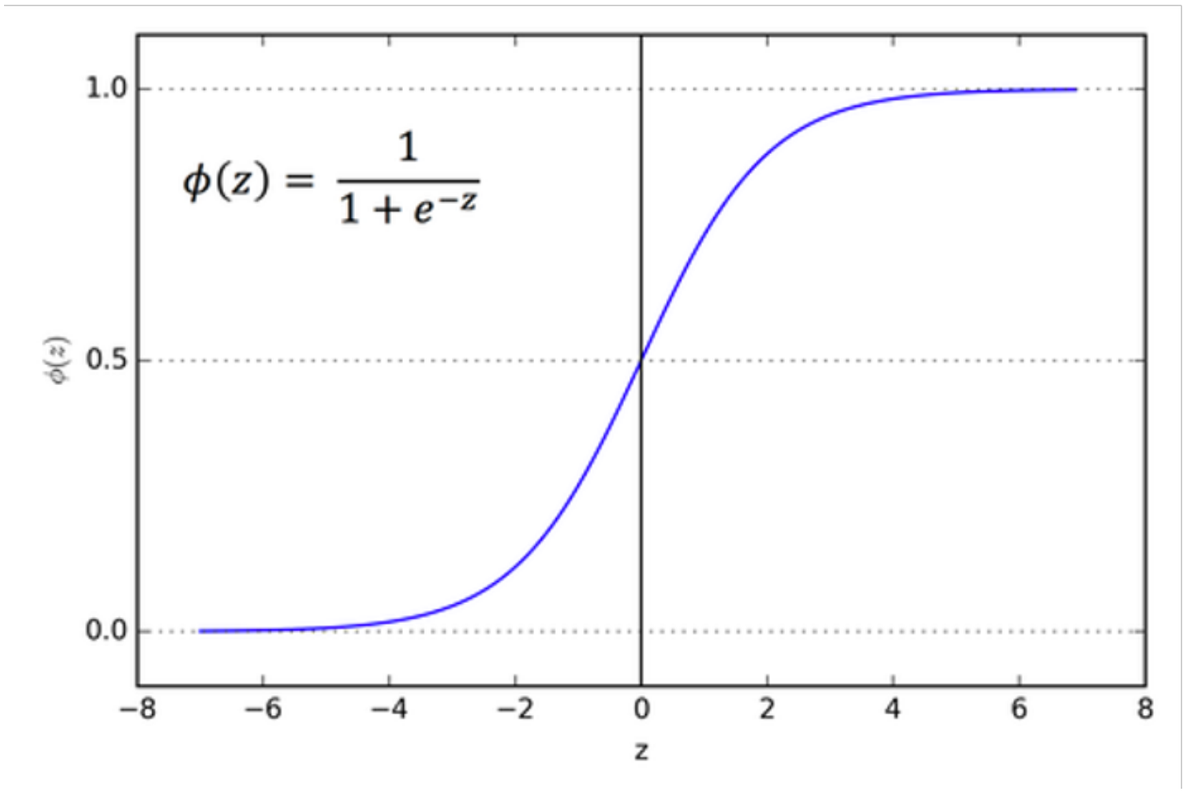
不可分情形：概率决策



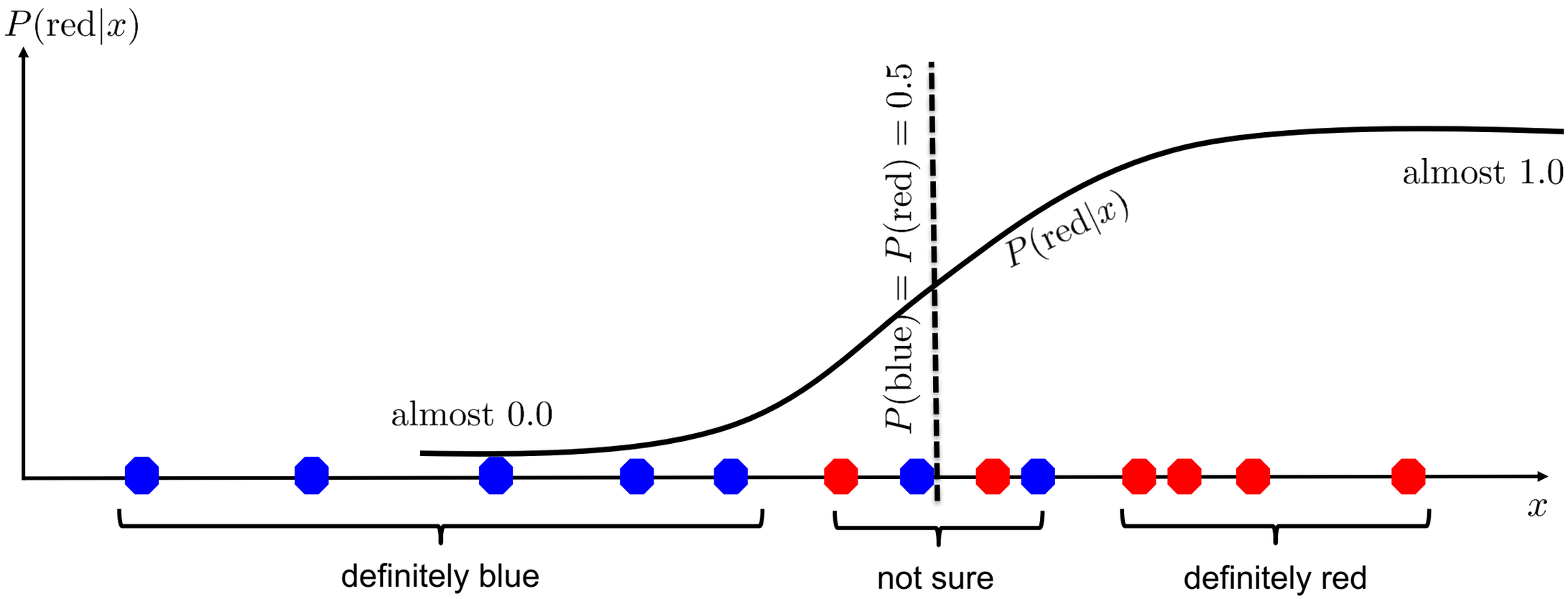
如何获得概率决策?

- 感知机评分: $z = w \cdot f(x)$
  - 如果 $z = w \cdot f(x)$ 是很大的正数，那么希望它的值趋近于1
  - 如果 $z = w \cdot f(x)$ 是很大的正数，那么希望它的值趋近于0
- Sigmoid 函数

$$\phi(z) = \frac{1}{1 + e^{-z}}$$



Sigmoid 二分类的例子

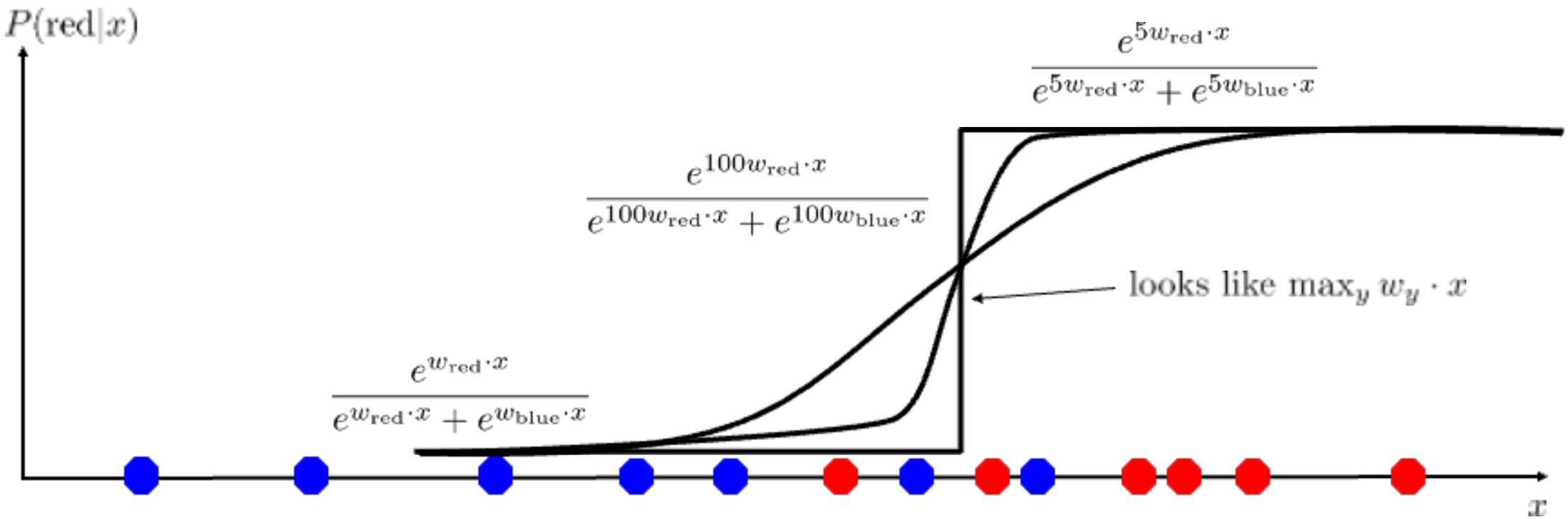


$$P(\text{red}|x) = \frac{e^{w_{\text{red}} \cdot x}}{e^{w_{\text{red}} \cdot x} + e^{w_{\text{blue}} \cdot x}}$$

概率呈指数增长

归一化

不同的SoftMax函数



求解最佳权重

- 最大似然估计：

$$\max_w \quad ll(w) = \max_w \quad \sum_i \log P(y^{(i)}|x^{(i)}; w)$$

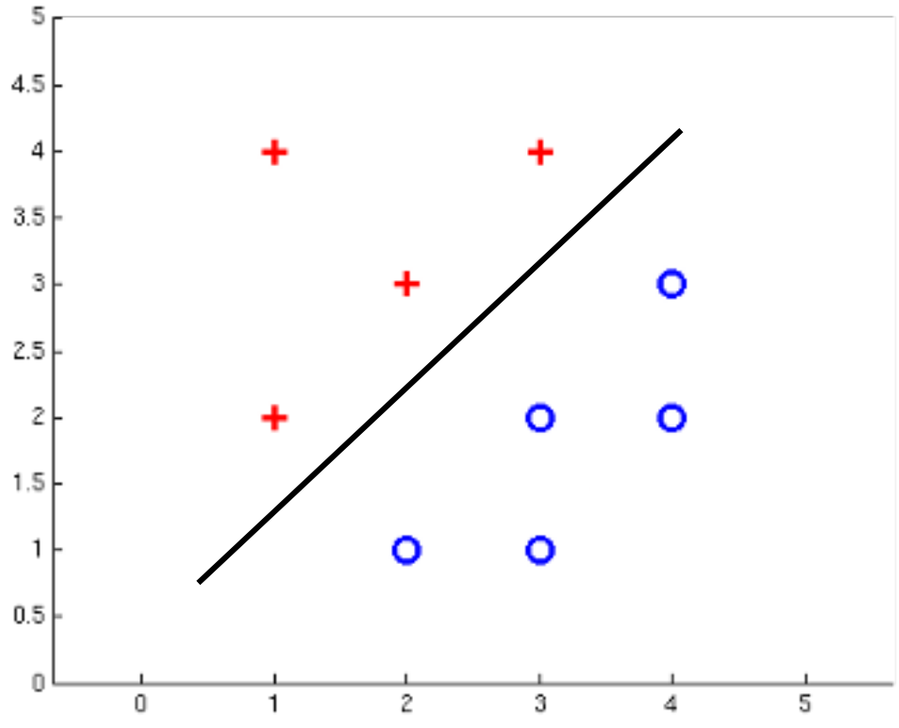
- 其中

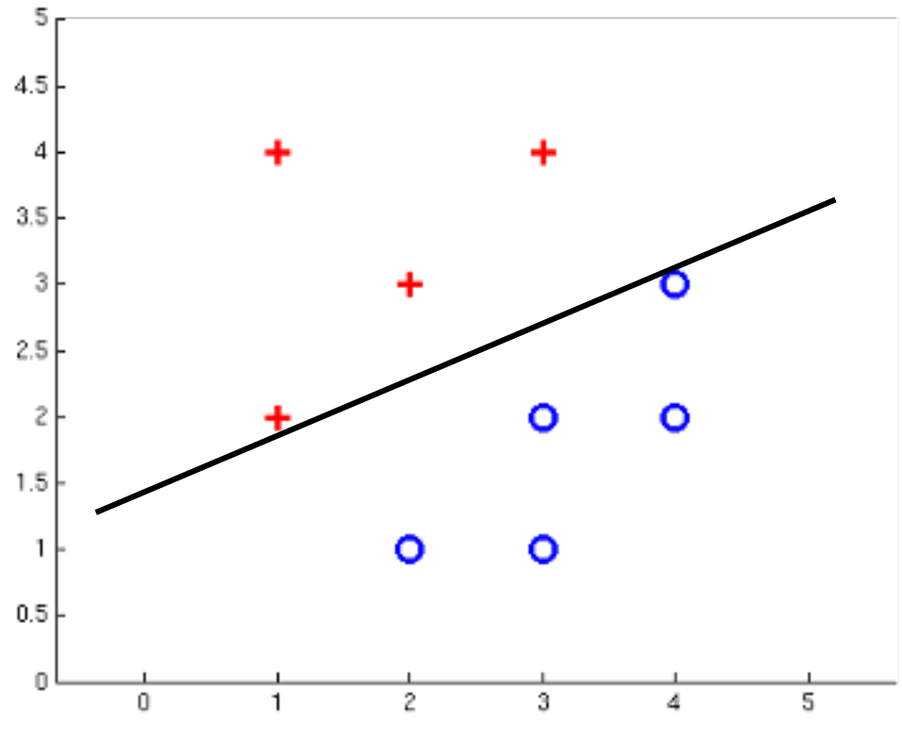
$$P(y^{(i)} = +1|x^{(i)}; w) = \frac{1}{1 + e^{-w \cdot f(x^{(i)})}}$$

$$P(y^{(i)} = -1|x^{(i)}; w) = 1 - \frac{1}{1 + e^{-w \cdot f(x^{(i)})}}$$

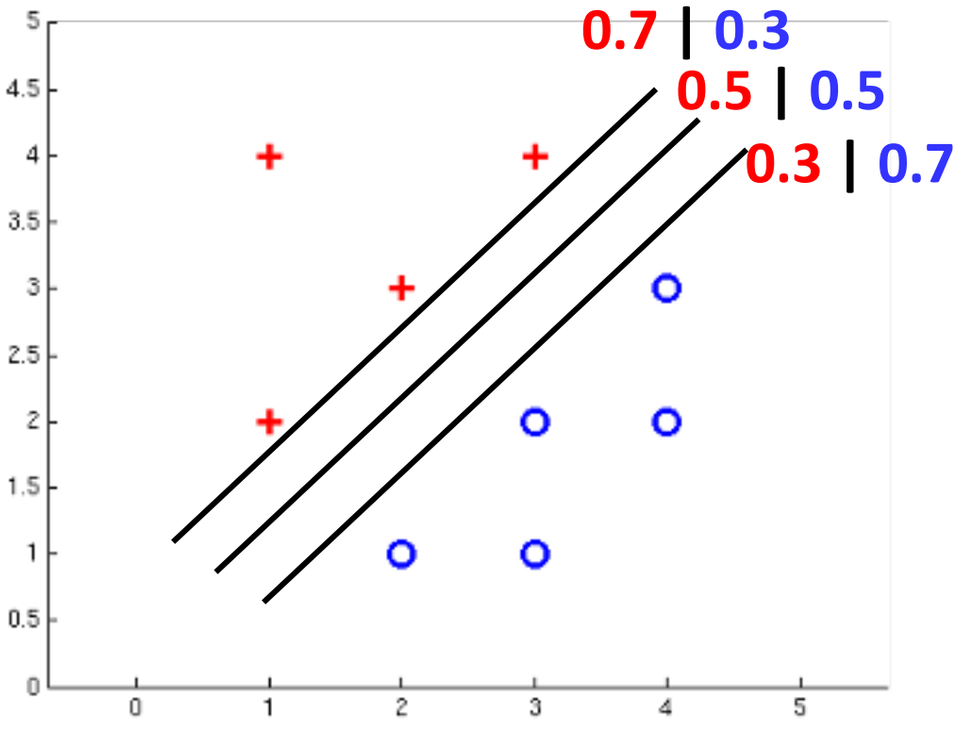
- 这就是逻辑回归 Logistic Regression

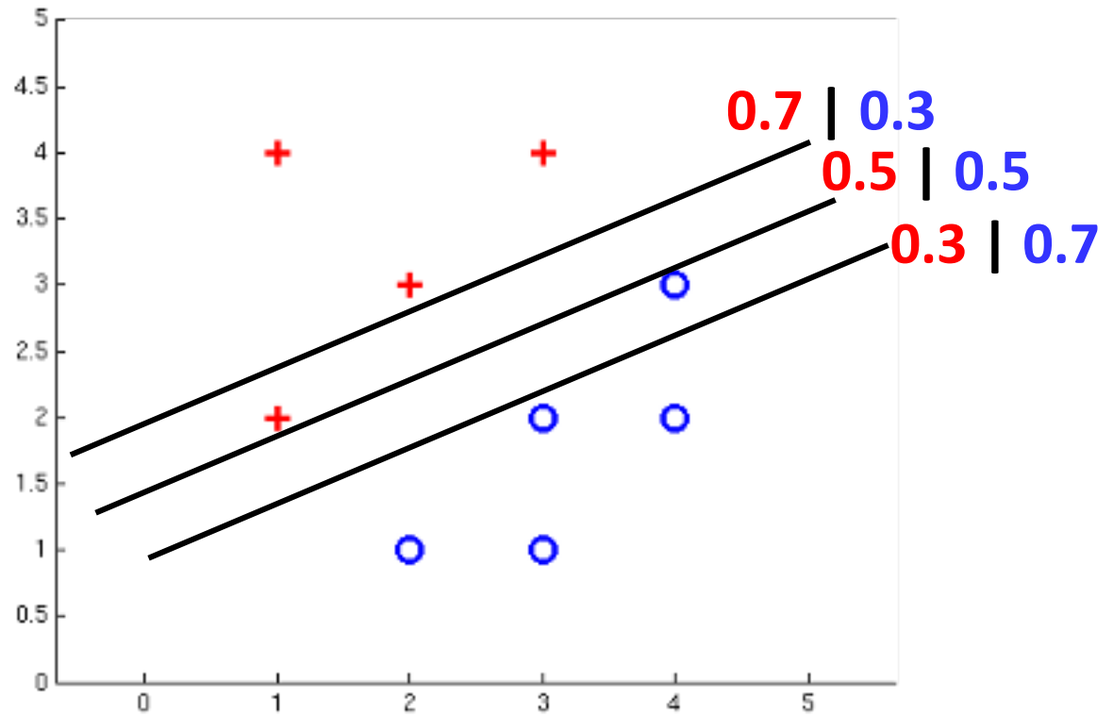
可分离案例：确定性决策-多种可能的选择





可分离案例：概率决策-明确的偏好





多类别的逻辑回归

- 对于感知机:
  - 每个类的权重向量: w_y
  - 计算样本在每个y类的得分: $w_y \cdot f(x)$
  - 将样本判断为最高得分的类别 $y = \arg\max_y \ w_y \cdot f(x)$
- 如何将分数转化为概率?

$$\underbrace{z_1, z_2, z_3}_{\text{original activations}} \rightarrow \underbrace{\frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3}}, \frac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3}}, \frac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3}}}_{\text{softmax activations}}$$

求解最佳权重

- 最大似然估计：

$$\max_w \quad ll(w) = \max_w \quad \sum_i \log P(y^{(i)}|x^{(i)}; w)$$

- 其中

$$P(y^{(i)}|x^{(i)}; w) = \frac{e^{w_{y^{(i)}} \cdot f(x^{(i)})}}{\sum_y e^{w_y \cdot f(x^{(i)})}}$$

- 这就是多分类逻辑回归 Logistic Regression

优化问题

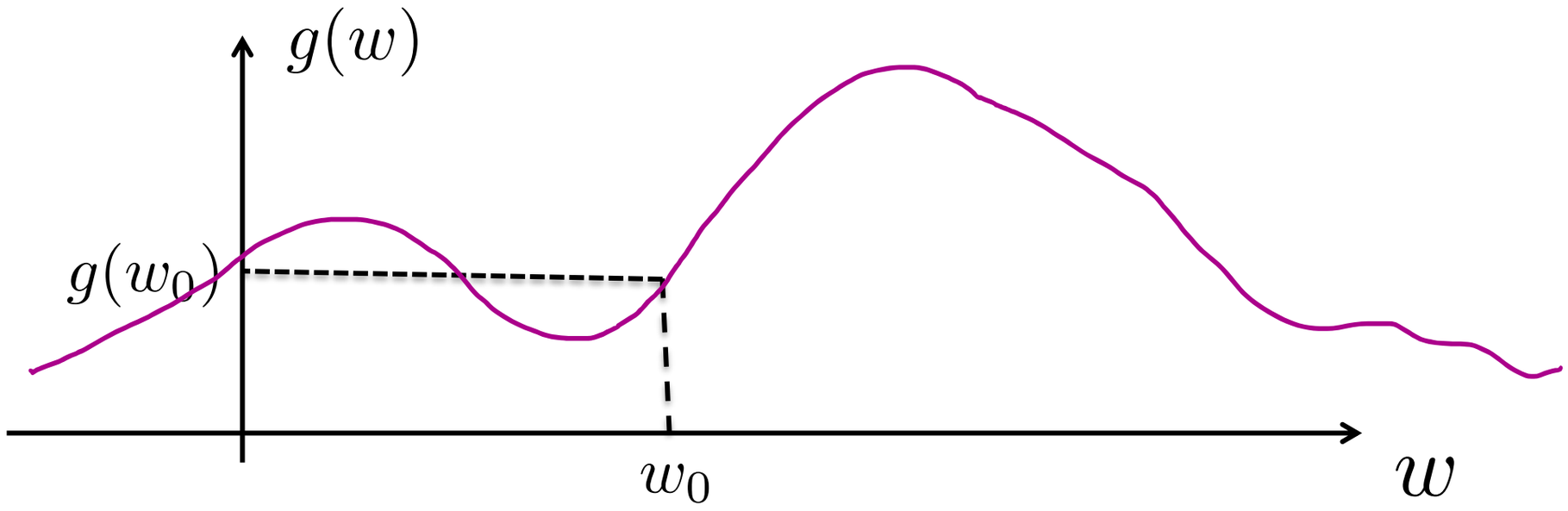
- 如何求解

$$\max_{w} \quad ll(w) = \max_{w} \quad \sum_{i} \log P(y^{(i)}|x^{(i)}; w)$$

爬山算法

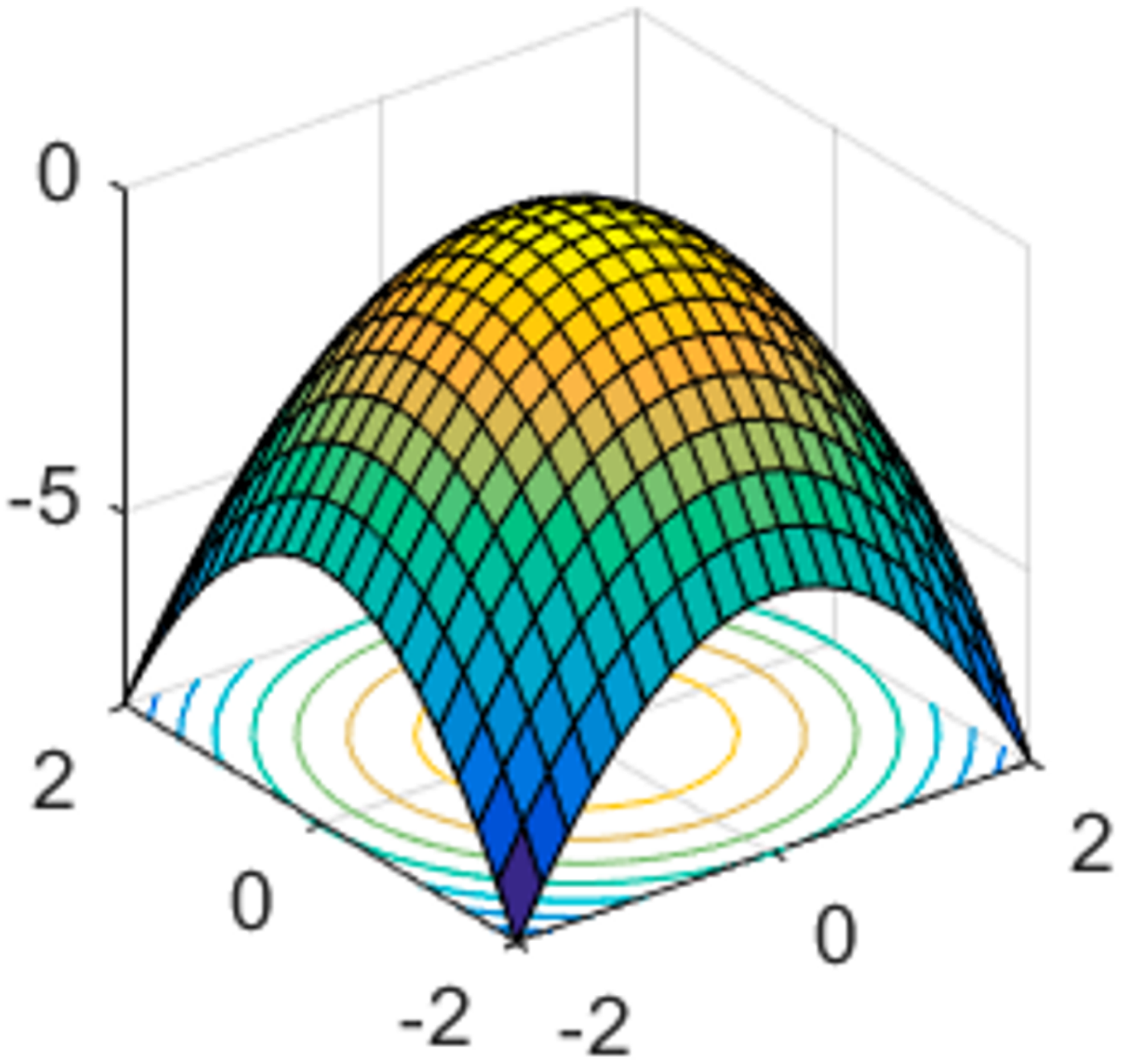
- CSPs中的爬山算法
  - 随机选取开始状态
  - 重复：移动到最佳邻近状态
  - 如果没有比当前更好的邻居，算法结束
- 如何应用到逻辑回归上?
  - 在连续空间上进行优化
  - 无限多个邻居

一维优化



- 可以计算 $g(w_0 + h)$ 和 $g(w_0 - h)$
  - 然后朝最佳方向前进
- 或者，求值导数：$\dfrac{\partial g(w_0)}{\partial w} = \lim_{h \to 0} \dfrac{g(w_0 + h) - g(w_0 - h)}{2h}$
  - 告诉我们要进入哪个方向

二维优化



梯度上升

- 在上山方向为每个坐标执行更新
- 坡度越陡（即导数越高），该坐标的步长越大
- 例如，考虑： $g(w_1, w_2)$

- 更新：

$$w_1 \leftarrow w_1 + \alpha * \frac{\partial g}{\partial w_1}(w_1, w_2)$$

$$w_2 \leftarrow w_2 + \alpha * \frac{\partial g}{\partial w_2}(w_1, w_2)$$

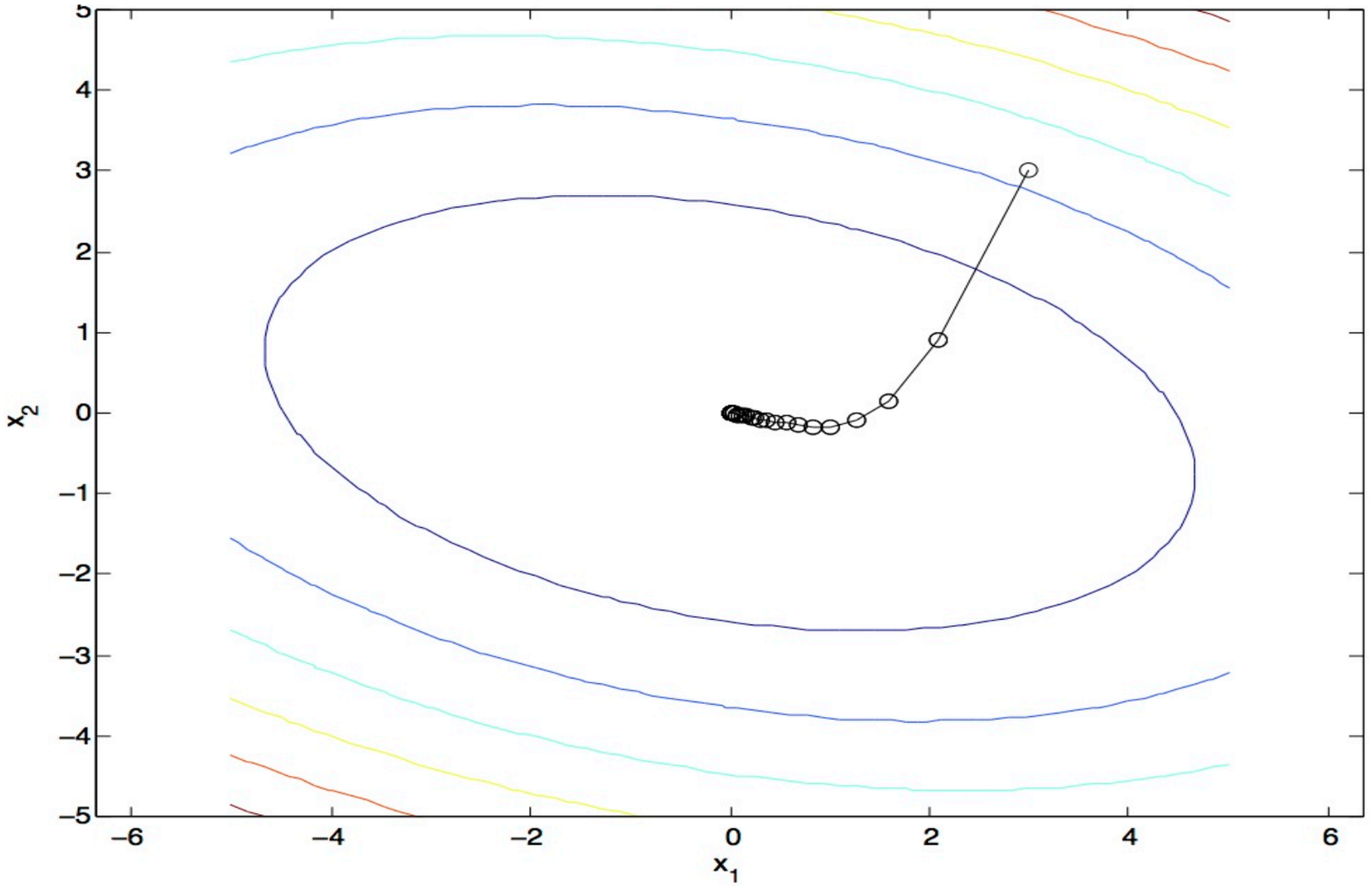
- 矢量表示法的更新：

$$w \leftarrow w + \alpha * \nabla_w g(w)$$

- 其中梯度为 $\nabla_w g(w) = \begin{bmatrix} \frac{\partial g}{\partial w_1}(w) \\ \frac{\partial g}{\partial w_2}(w) \end{bmatrix}$

梯度上升

- 随机选取开始状态
- 重复：向着梯度的方向前进



最陡的方向是什么?

$$\max_{\Delta:\Delta_1^2+\Delta_2^2\leq\varepsilon} \quad g(w+\Delta)$$

- 一阶泰勒展开: $g(w+\Delta) \approx g(w) + \frac{\partial g}{\partial w_1}\Delta_1 + \frac{\partial g}{\partial w_2}\Delta_2$
- 最陡上升方向: $\max_{\Delta:\Delta_1^2+\Delta_2^2\leq\varepsilon} \quad g(w) + \frac{\partial g}{\partial w_1}\Delta_1 + \frac{\partial g}{\partial w_2}\Delta_2$
- 因为: $\max_{\Delta:\|\Delta\|\leq\varepsilon} \Delta^\top a \quad \rightarrow \quad \Delta = \varepsilon\frac{a}{\|a\|}$
- 因此: $\Delta = \varepsilon\frac{\nabla g}{\|\nabla g\|} \quad \nabla g = \begin{bmatrix} \frac{\partial g}{\partial w_1} \\ \frac{\partial g}{\partial w_2} \\ \cdots \\ \frac{\partial g}{\partial w_n} \end{bmatrix}$

优化程序：梯度上升

- init w
- for iter = 1, 2, …

$$w \leftarrow w + \alpha * \nabla g(w)$$

$$\nabla g = \begin{bmatrix} \frac{\partial g}{\partial w_1} \\ \frac{\partial g}{\partial w_2} \\ \cdots \\ \frac{\partial g}{\partial w_n} \end{bmatrix}$$

- α：学习率，梯度上升的调整参数，需要仔细选择
- 如何选择学习率？多多尝试
  - 粗略的经验法则：每次更新使得权重的变化大约0.1–1%

基于对数似然目标的批梯度上升

$$\max_w \quad ll(w) = \max_w \quad \underbrace{\sum_i \log P(y^{(i)}|x^{(i)};w)}_{g(w)}$$

- init w
- for iter = 1, 2, …

$$w \leftarrow w + \alpha * \sum_i \nabla \log P(y^{(i)}|x^{(i)};w)$$

对数似然目标下的随机梯度上升

$$\max_w \quad ll(w) = \max_w \quad \sum_i \log P(y^{(i)}|x^{(i)}; w)$$

- 一旦计算了一个训练样本的梯度，可以在计算下一个之前进行更新

- init w
- for iter = 1, 2, …
  - pick random j

$$w \leftarrow w + \alpha * \nabla \log P(y^{(j)}|x^{(j)}; w)$$

对数似然目标下的小批量梯度上升

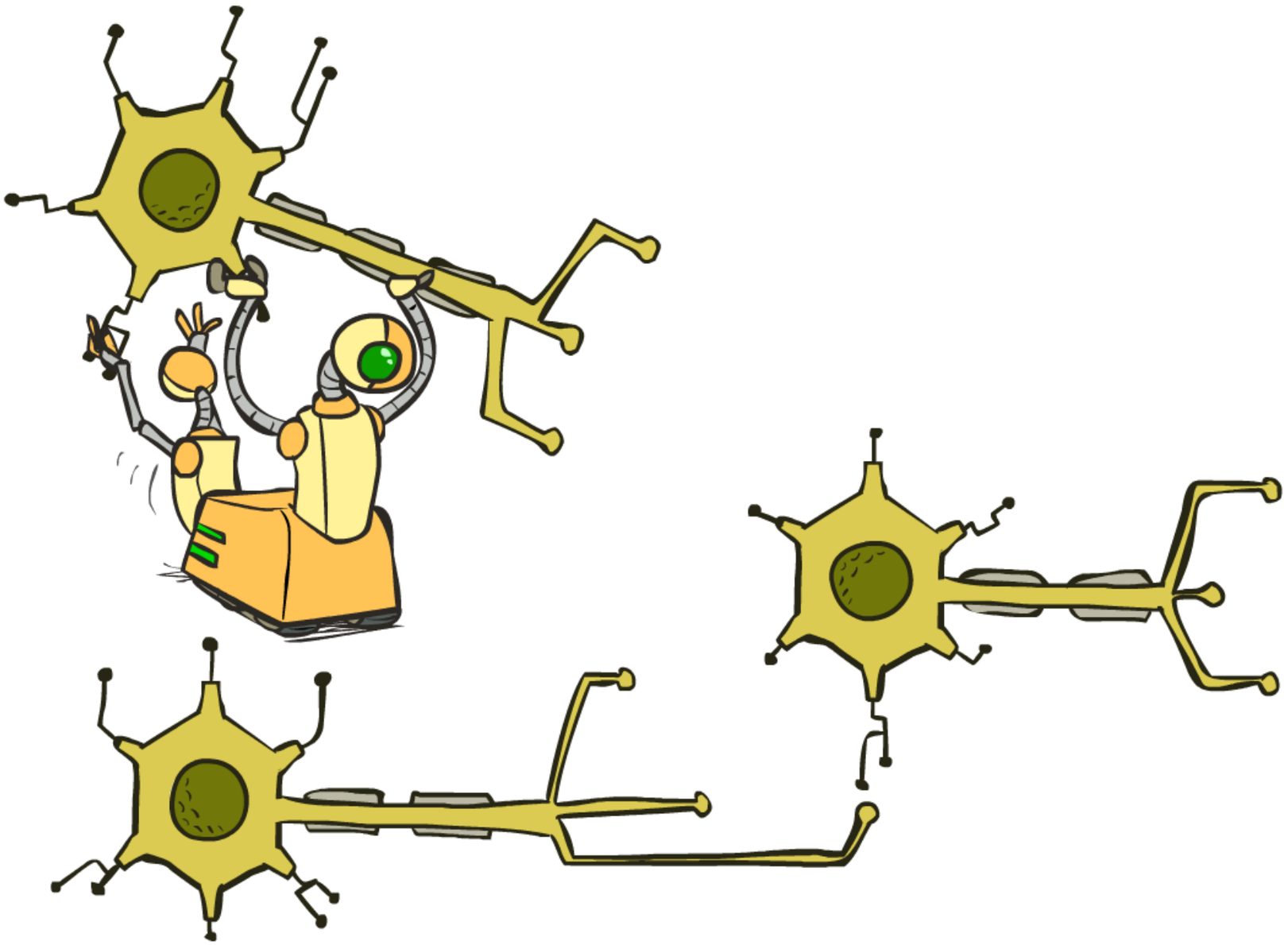
$$\max_w \quad ll(w) = \max_w \quad \sum_i \log P(y^{(i)}|x^{(i)}; w)$$

- 可以用一个小的数据集(mini-batch)上的梯度来代替单一样本上的梯度

- init w
- for iter = 1, 2, …
  - pick random subset of training examples J

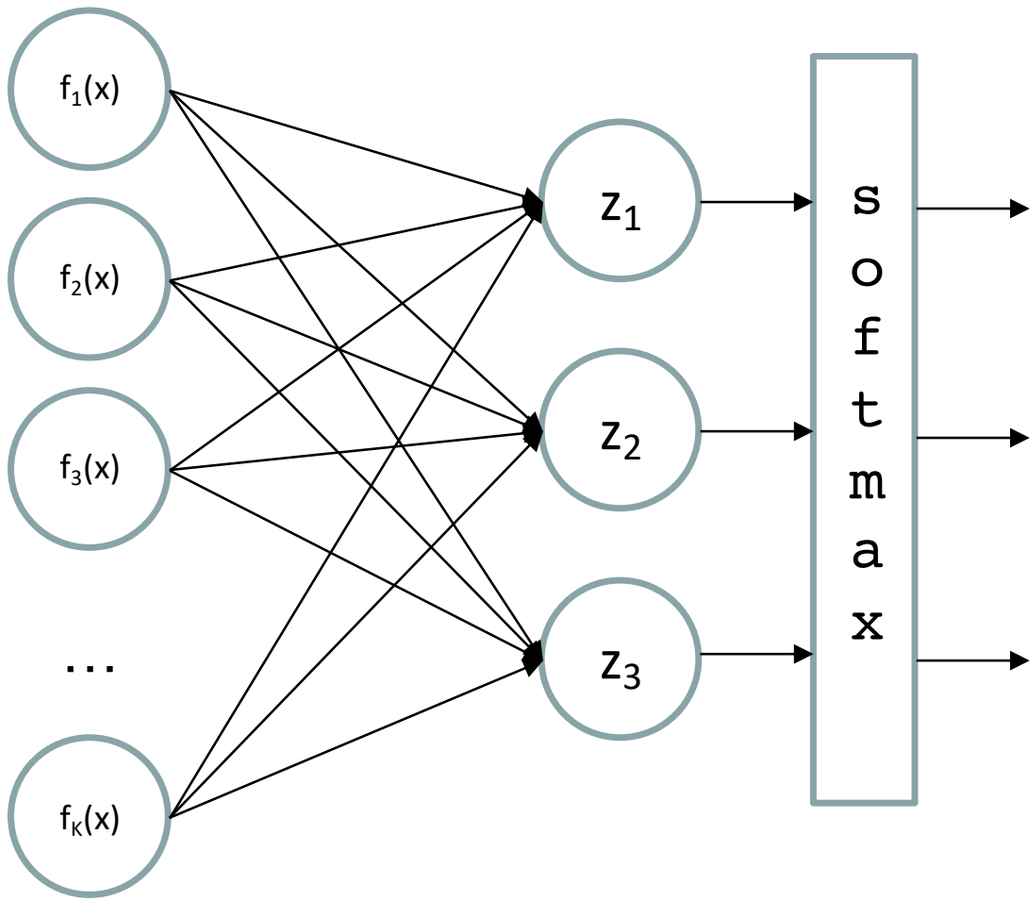
$$w \leftarrow w + \alpha * \sum_{j \in J} \nabla \log P(y^{(j)}|x^{(j)}; w)$$

神经网络



多分类逻辑回归

- 其实是神经网络的一种特殊形式

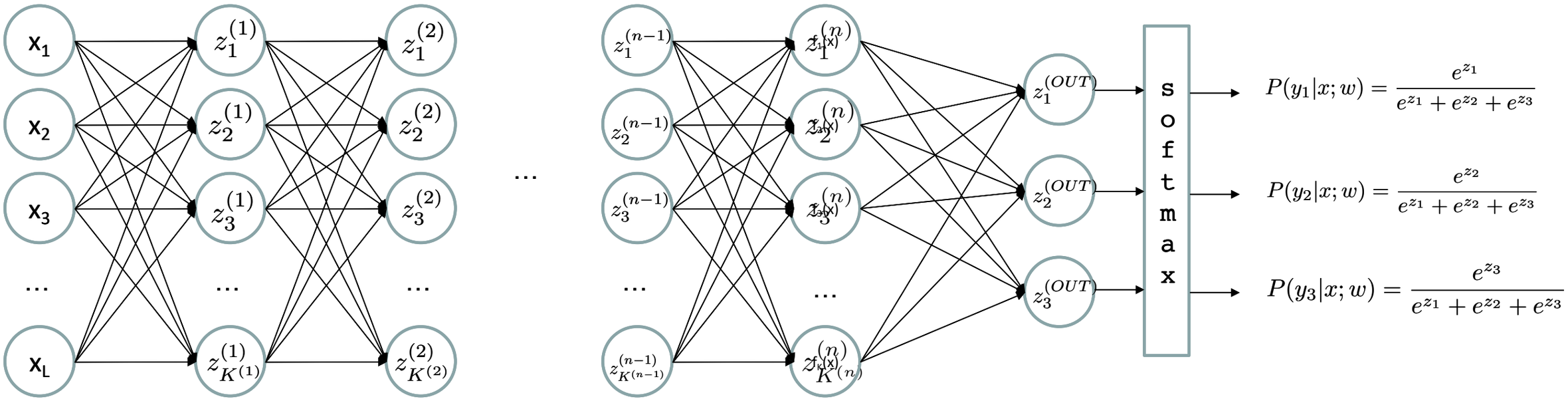


$$P(y_1|x;w) = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

$$P(y_2|x;w) = \frac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

$$P(y_3|x;w) = \frac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

深层神经网络也学习特征

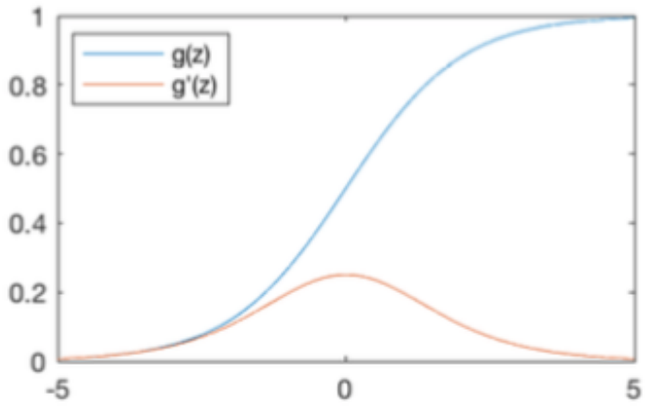


$$z_i^{(k)} = g(\sum_j W_{i,j}^{(k-1,k)} z_j^{(k-1)})$$

g = nonlinear activation function

常用激活函数

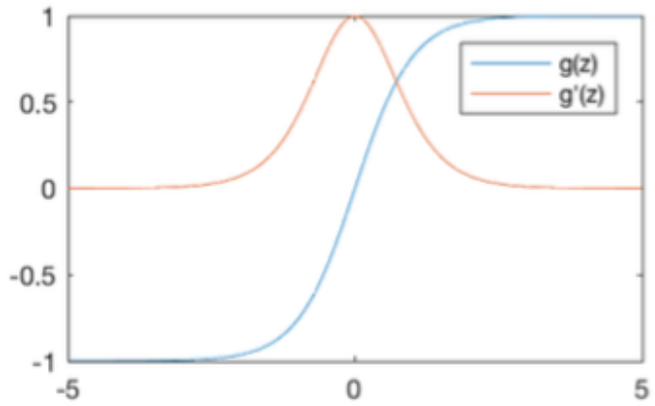
Sigmoid Function



$$g(z) = \frac{1}{1 + e^{-z}}$$

$$g'(z) = g(z)(1 - g(z))$$

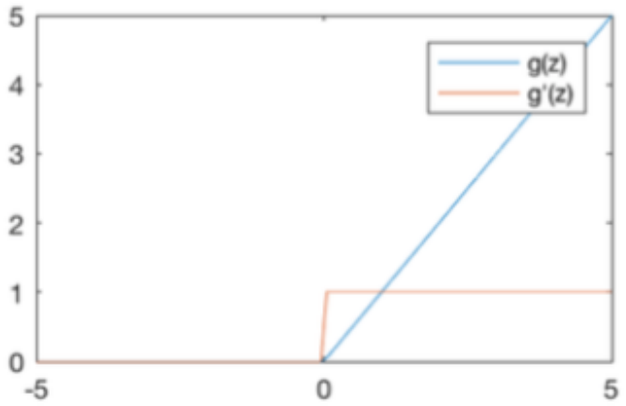
Hyperbolic Tangent



$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

$$g'(z) = 1 - g(z)^2$$

Rectified Linear Unit (ReLU)



$$g(z) = \max(0, z)$$

$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

深层神经网络也学习特征

- 训练深层神经网络和逻辑回归类似：
  - 运行梯度上升
  - 当在留出集上对数似然开始下降时，停止迭代过程

$$\max_{w} \quad ll(w) = \max_{w} \quad \sum_{i} \log P(y^{(i)}|x^{(i)}; w)$$

神经网络特性

- 定理（通用函数逼近器）。一个具有足够数量神经元的双层神经网络可以将任意连续函数近似到任何期望的精度。
- 实际考虑
  - 可以看作是特征的学习
  - 大量神经元会导致较高的过拟合风险

通用函数逼近定理

Hornik theorem 1: Whenever the activation function is *bounded and nonconstant*, then, for any finite measure μ, standard multilayer feedforward networks can approximate any function in $L^p(\mu)$ (the space of all functions on R^k such that $\int_{R^k} |f(x)|^p d\mu(x) < \infty$) arbitrarily well, provided that sufficiently many hidden units are available.

Hornik theorem 2: Whenever the activation function is *continuous, bounded and nonconstant*, then, for arbitrary compact subsets $X \subseteq R^k$, standard multilayer feedforward networks can approximate any continuous function on X arbitrarily well with respect to uniform distance, provided that sufficiently many hidden units are available.

- 说人话：给定任何连续函数f(x)，如果一个2层神经网络有足够的隐藏单元，那么总有一个权值的选择，使它接近f(x)。

Cybenko (1989) “Approximations by superpositions of sigmoidal functions”
Hornik (1991) “Approximation Capabilities of Multilayer Feedforward Networks”
Leshno and Schocken (1991) ”Multilayer Feedforward Networks with Non-Polynomial Activation Functions Can Approximate Any Function”

如何计算导数

$$\frac{d}{dx}(a) = 0$$

$$\frac{d}{dx}(x) = 1$$

$$\frac{d}{dx}(au) = a\frac{du}{dx}$$

$$\frac{d}{dx}(u+v-w) = \frac{du}{dx}+\frac{dv}{dx}-\frac{dw}{dx}$$

$$\frac{d}{dx}(uv) = u\frac{dv}{dx}+v\frac{du}{dx}$$

$$\frac{d}{dx}\left(\frac{u}{v}\right) = \frac{1}{v}\frac{du}{dx}-\frac{u}{v^2}\frac{dv}{dx}$$

$$\frac{d}{dx}(u^n) = nu^{n-1}\frac{du}{dx}$$

$$\frac{d}{dx}(\sqrt{u}) = \frac{1}{2\sqrt{u}}\frac{du}{dx}$$

$$\frac{d}{dx}\left(\frac{1}{u}\right) = -\frac{1}{u^2}\frac{du}{dx}$$

$$\frac{d}{dx}\left(\frac{1}{u^n}\right) = -\frac{n}{u^{n+1}}\frac{du}{dx}$$

$$\frac{d}{dx}[f(u)] = \frac{d}{du}[f(u)]\frac{du}{dx}$$

$$\frac{d}{dx}[\ln u] = \frac{d}{dx}[\log_e u] = \frac{1}{u}\frac{du}{dx}$$

$$\frac{d}{dx}[\log_a u] = \log_a e\frac{1}{u}\frac{du}{dx}$$

$$\frac{d}{dx}e^u = e^u\frac{du}{dx}$$

$$\frac{d}{dx}a^u = a^u\ln a\frac{du}{dx}$$

$$\frac{d}{dx}\left(u^v\right) = vu^{v-1}\frac{du}{dx}+\ln u\ \ u^v\frac{dv}{dx}$$

$$\frac{d}{dx}\sin u = \cos u\frac{du}{dx}$$

$$\frac{d}{dx}\cos u = -\sin u\frac{du}{dx}$$

$$\frac{d}{dx}\tan u = \sec^2 u\frac{du}{dx}$$

$$\frac{d}{dx}\cot u = -\csc^2 u\frac{du}{dx}$$

$$\frac{d}{dx}\sec u = \sec u\tan u\frac{du}{dx}$$

$$\frac{d}{dx}\csc u = -\csc u\cot u\frac{du}{dx}$$

如何计算所有f(x)的导数?

- 链式法则

If $$f(x) = g(h(x))$$

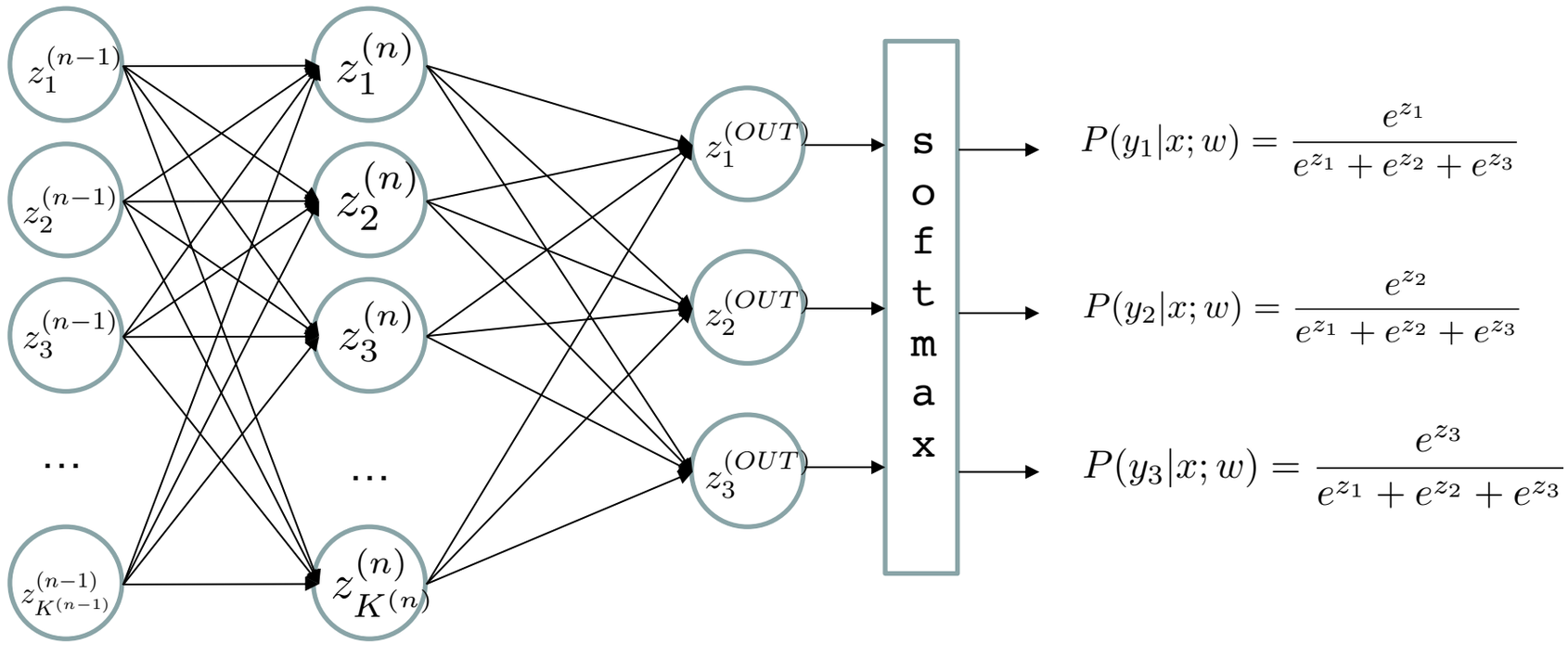
Then $$f'(x) = g'(h(x))h'(x)$$

自动微分

- 自动微分软件
  - Theano, TensorFlow, PyTorch, Chainer
  - 只需通过编程定义函数`g(x,y,w)`
  - 可以自动计算所有导数
  - 这通常是通过在`f`的正向计算过程中缓存信息，然后进行反向传播（Backpropagation, backprop, BP）来完成的
  - 自动微分/反向传播通常可以在计算成本与前向传递相当的情况下完成

训练神经网络

- 关键概念:
  - 前向 (Forward)
  - 反向 (Backwards)
  - 梯度 (Gradient)
  - 反向传播 (Backprop)



g = nonlinear activation function

反向传播示例

- 求解函数 $g(\boldsymbol{w}) = w_1^3 w_2 + 3w_1$ 在 $\boldsymbol{w} = [2, 3]$ 处的梯度

$$g = b + c$$

$$\frac{\partial g}{\partial b} = 1, \frac{\partial g}{\partial c} = 1$$

$$b = a \times w_2$$

$$\frac{\partial g}{\partial a} = \frac{\partial g}{\partial b}\frac{\partial b}{\partial a} = 1\frac{\partial b}{\partial a} = 1 \cdot 3 = 3$$

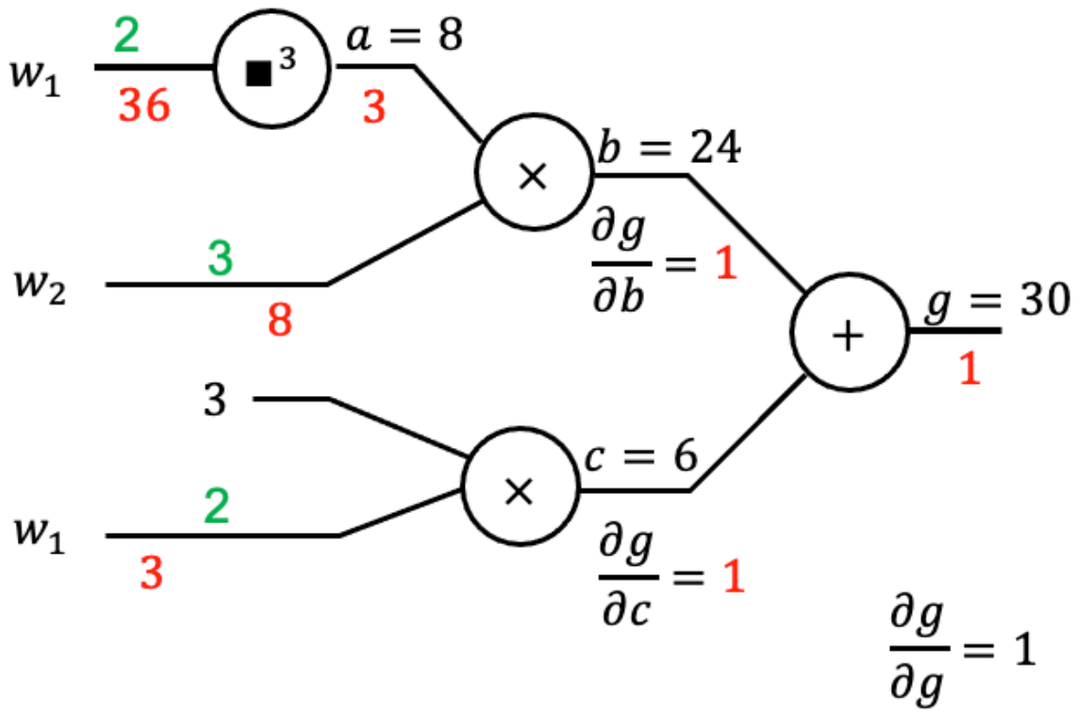
$$\frac{\partial g}{\partial w_2} = \frac{\partial g}{\partial b}\frac{\partial b}{\partial w_2} = 1\frac{\partial b}{\partial w_2} = 1 \cdot 8 = 8$$

$$a = w_1^3$$

$$\frac{\partial g}{\partial w_1} = \frac{\partial g}{\partial a}\frac{\partial a}{\partial w_1} = 3 \cdot 3w_1^2 = 36$$

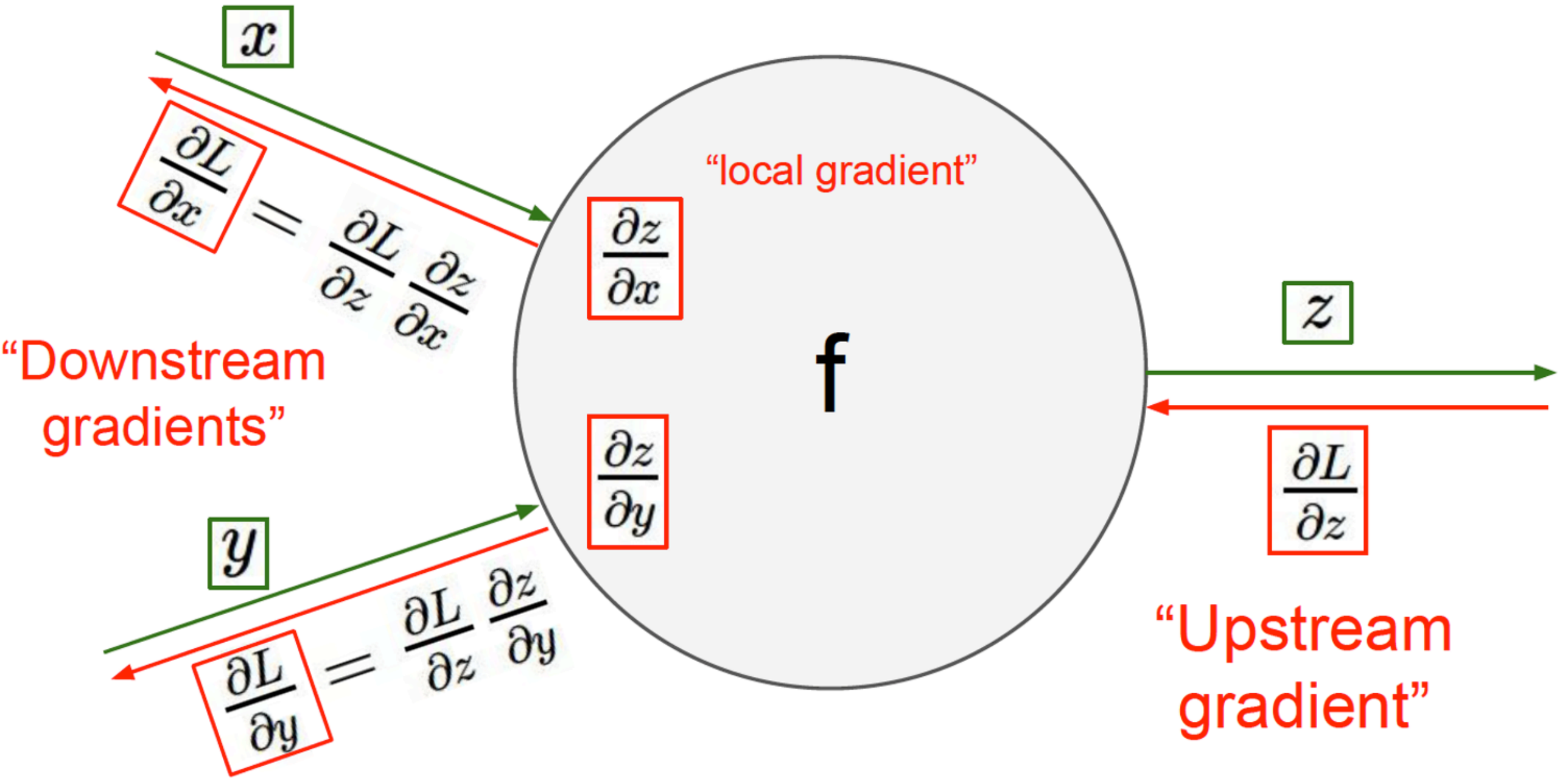
$$c = 3w_1$$

$$\frac{\partial g}{\partial w_1} = \frac{\partial g}{\partial c}\frac{\partial c}{\partial w_1} = 1 \cdot 3 = 3$$

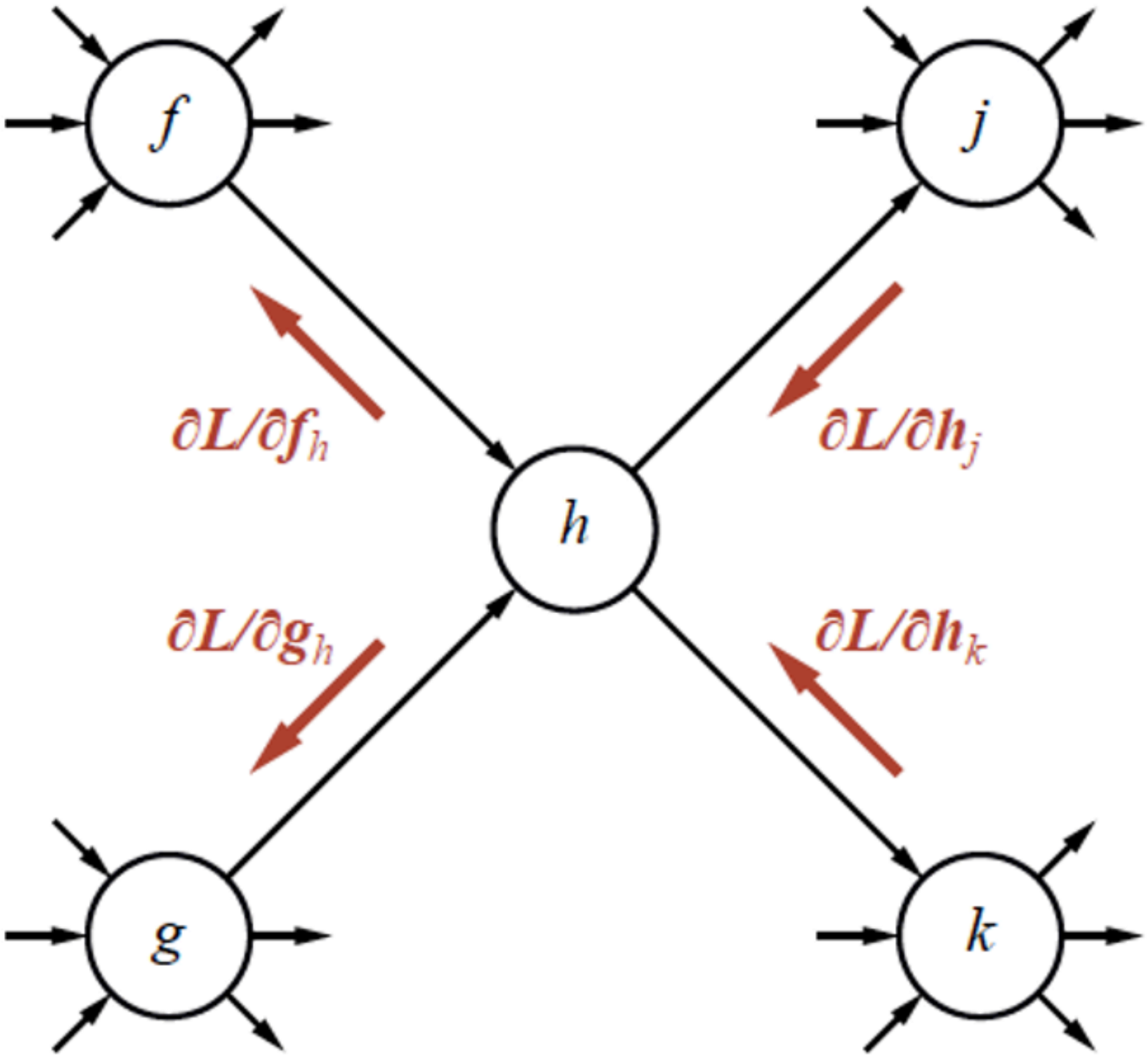


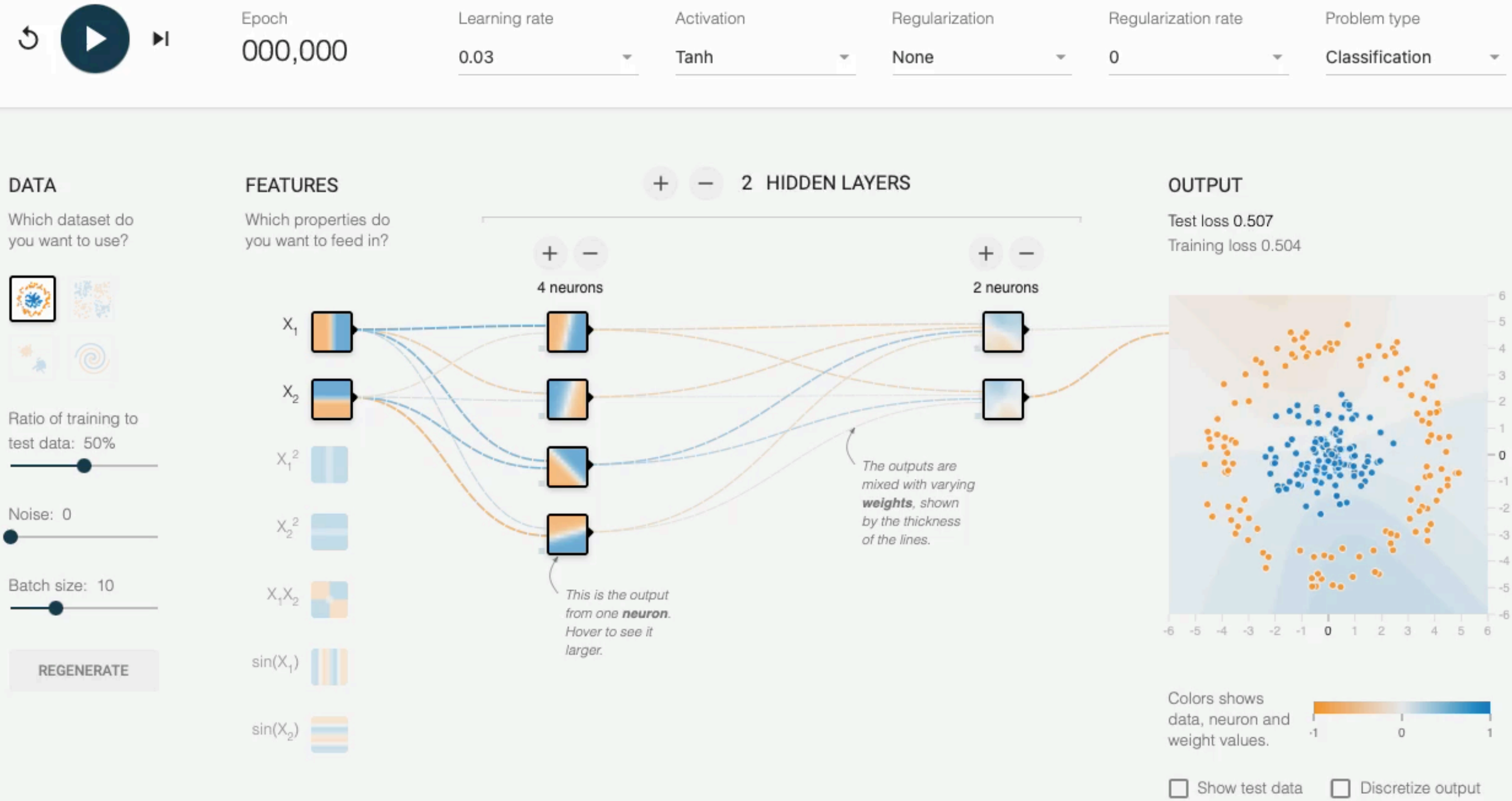
$$\nabla g = \left[\frac{\partial g}{\partial w_1}, \frac{\partial g}{\partial w_2}\right] = [39, 8]$$

反向传播


x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
"local gradient"
$\frac{\partial z}{\partial x}$
f
$\frac{\partial z}{\partial y}$
"Downstream gradients"
z
$\frac{\partial L}{\partial z}$
y
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$
"Upstream gradient"

反向传播

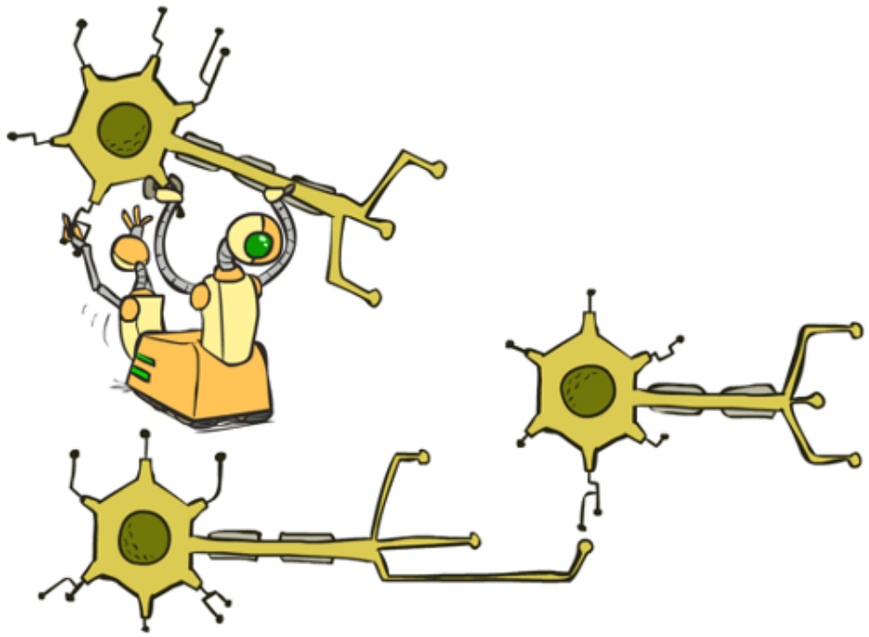




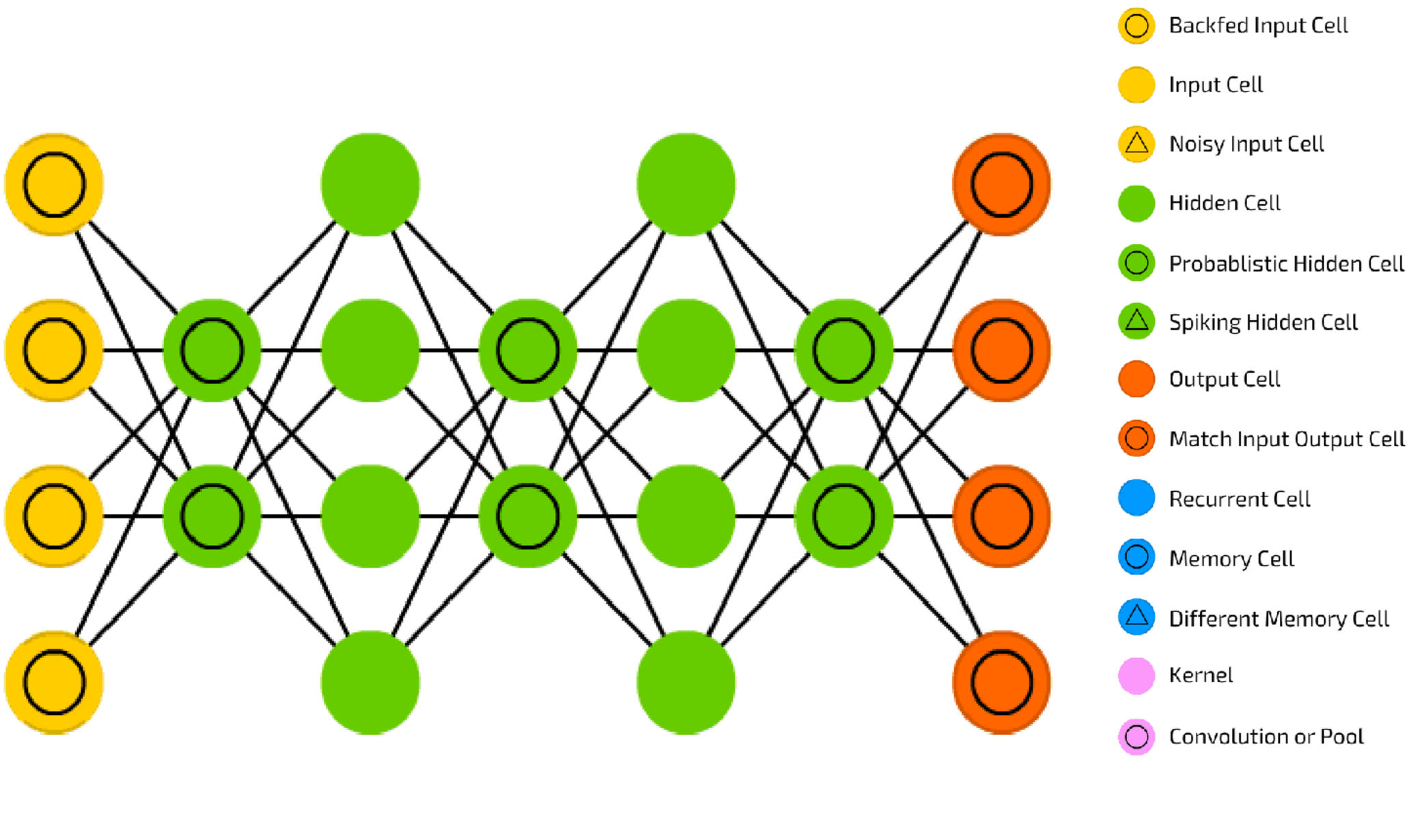
http://playground.tensorflow.org

常见网络结构

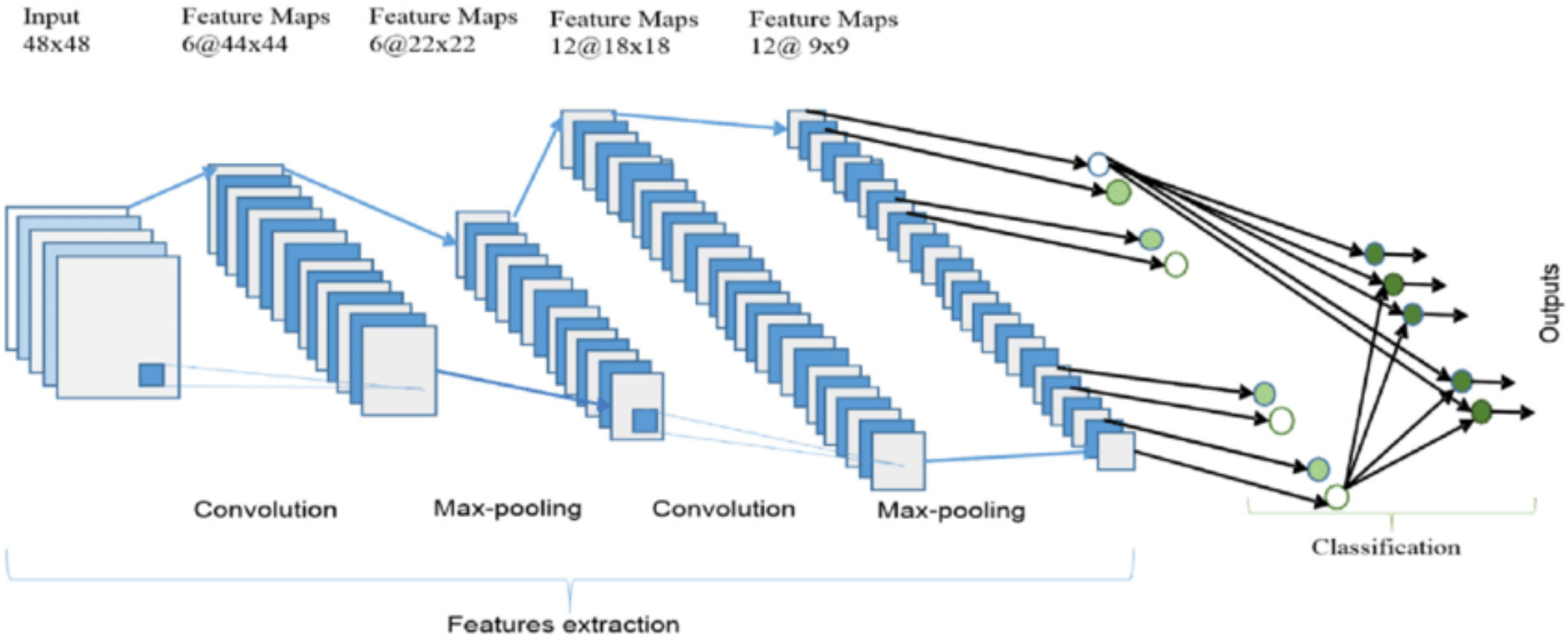
- 深度置信网络
- 卷积神经网络
- 循环神经网络
- 自注意力神经网络
- 图神经网络



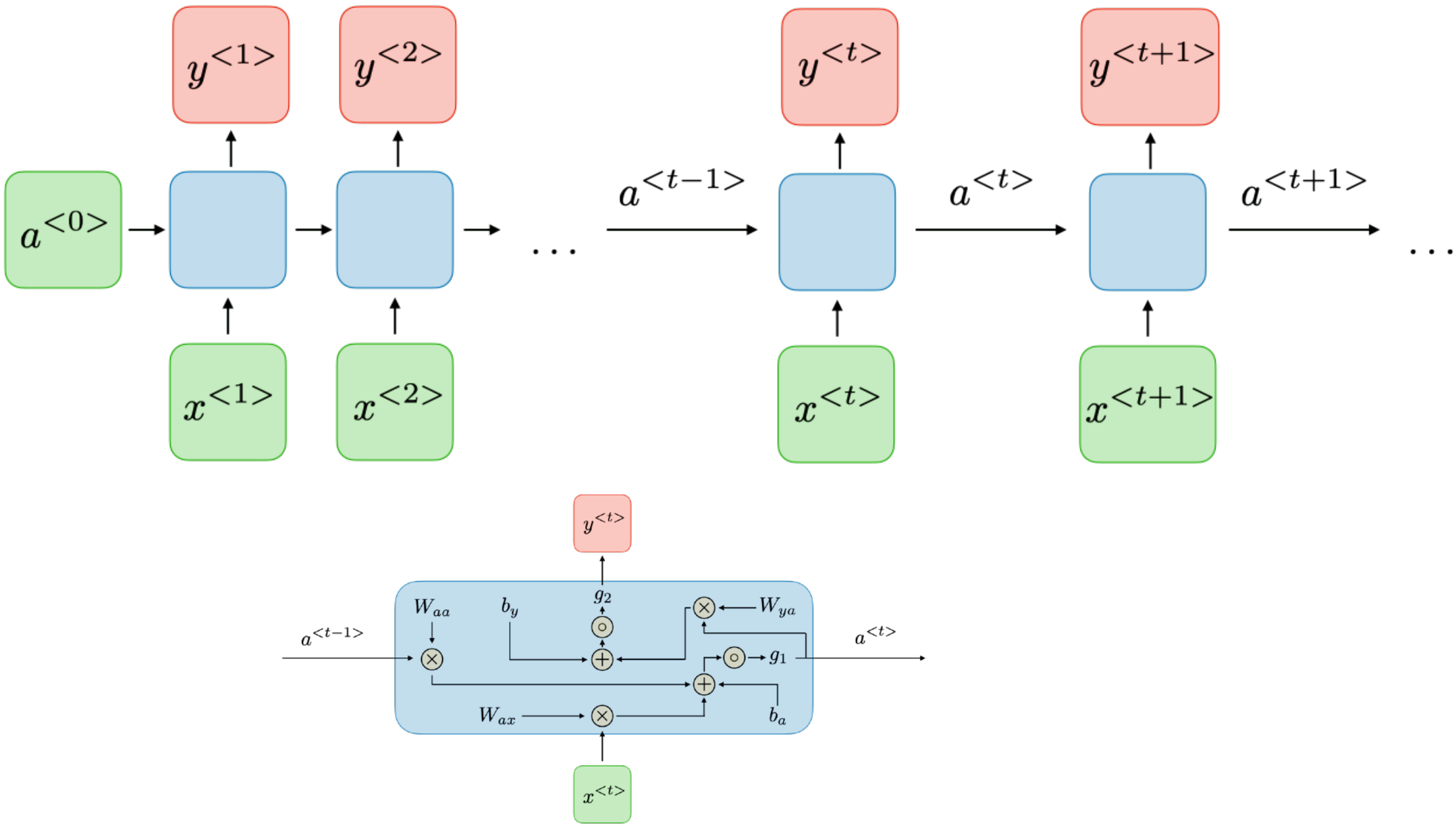
深度置信网络


Backfed Input Cell
Input Cell
Noisy Input Cell
Hidden Cell
Probablistic Hidden Cell
Spiking Hidden Cell
Output Cell
Match Input Output Cell
Recurrent Cell
Memory Cell
Different Memory Cell
Kernel
Convolution or Pool

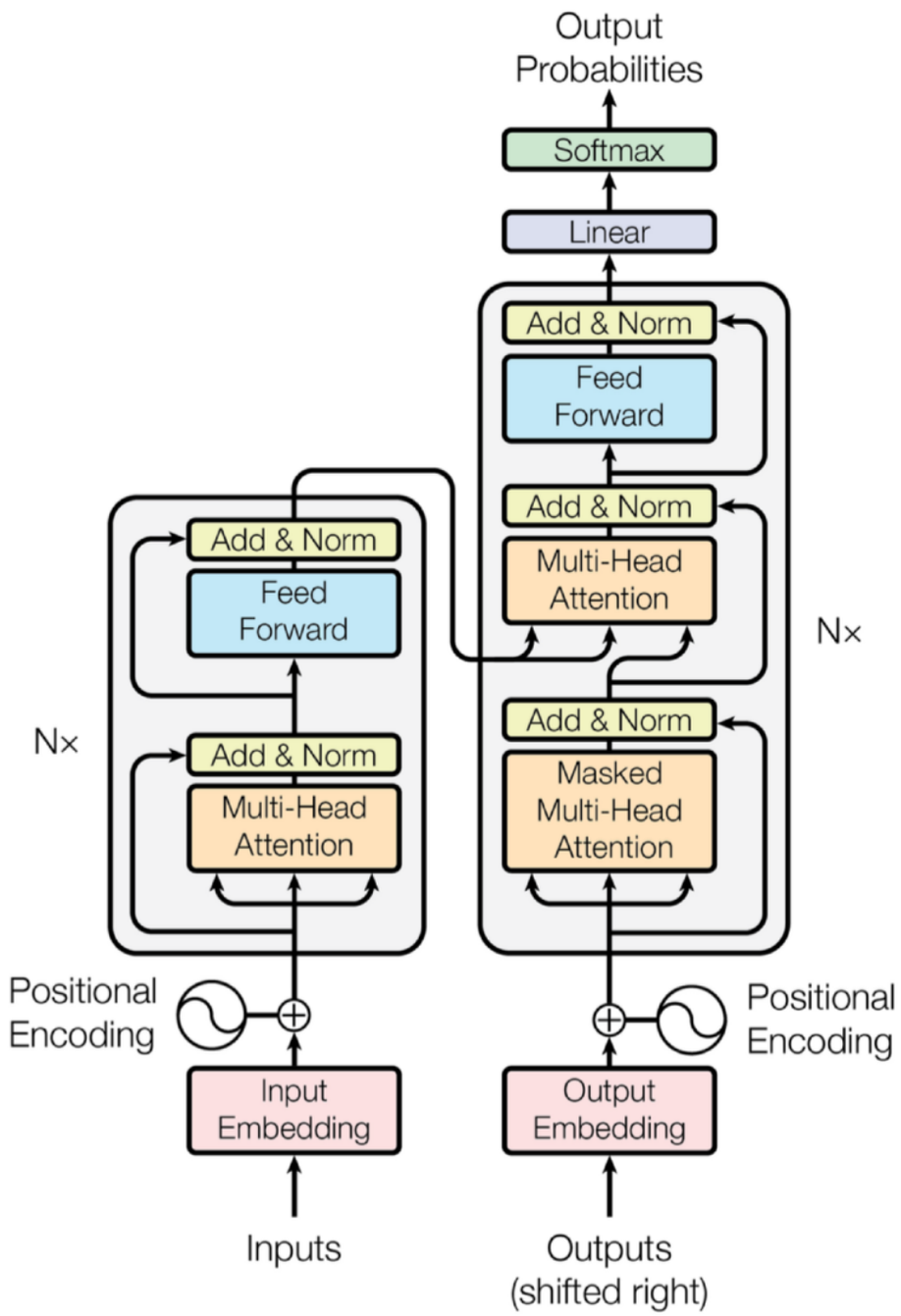
卷积神经网络

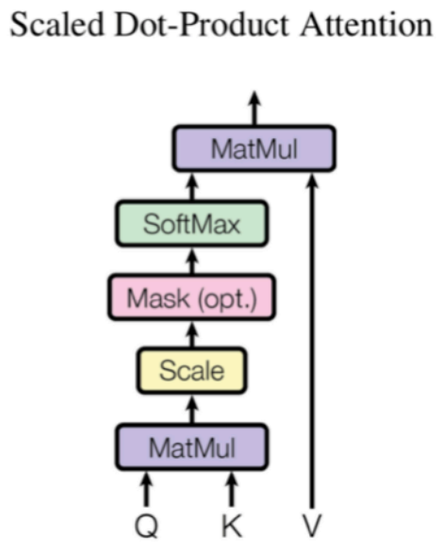

Input
48x48
Feature Maps
6@44x44
Feature Maps
6@22x22
Feature Maps
12@18x18
Feature Maps
12@ 9x9
Outputs
Convolution
Max-pooling
Convolution
Max-pooling
Classification
Features extraction

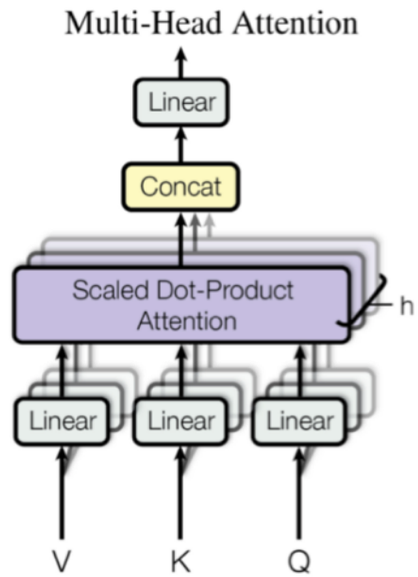
循环神经网络


$y^{<1>}$
$y^{<2>}$
$y^{<t>}$
$y^{<t+1>}$
$a^{<0>}$
$a^{<t-1>}$
$a^{<t>}$
$a^{<t+1>}$
$x^{<1>}$
$x^{<2>}$
$x^{<t>}$
$x^{<t+1>}$
$y^{<t>}$
W_{aa}
b_y
g_2
W_{ya}
$a^{<t-1>}$
$a^{<t>}$
g_1
W_{ax}
b_a
$x^{<t>}$

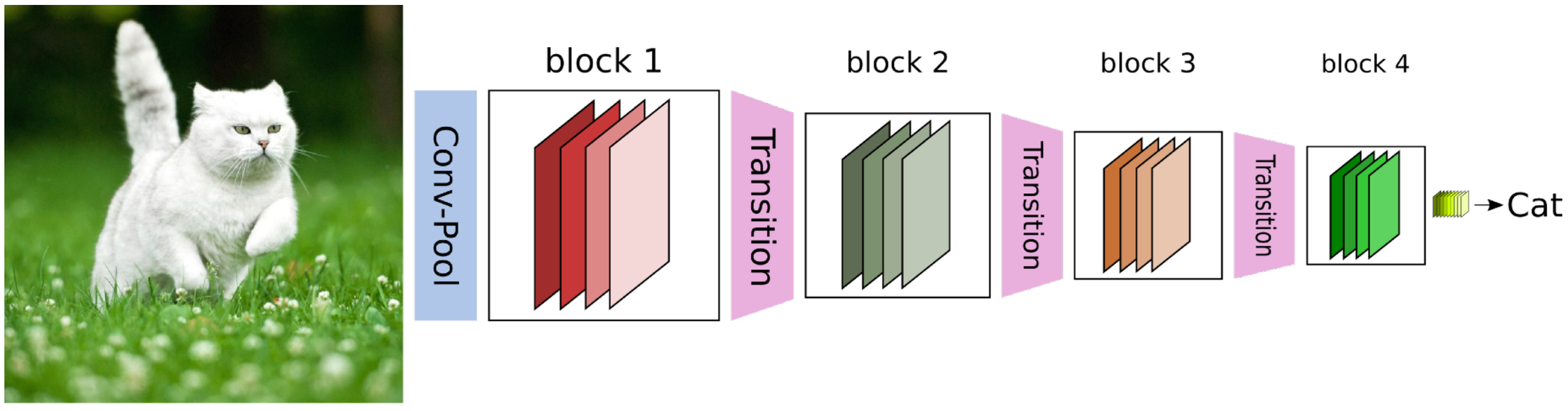
Transformer



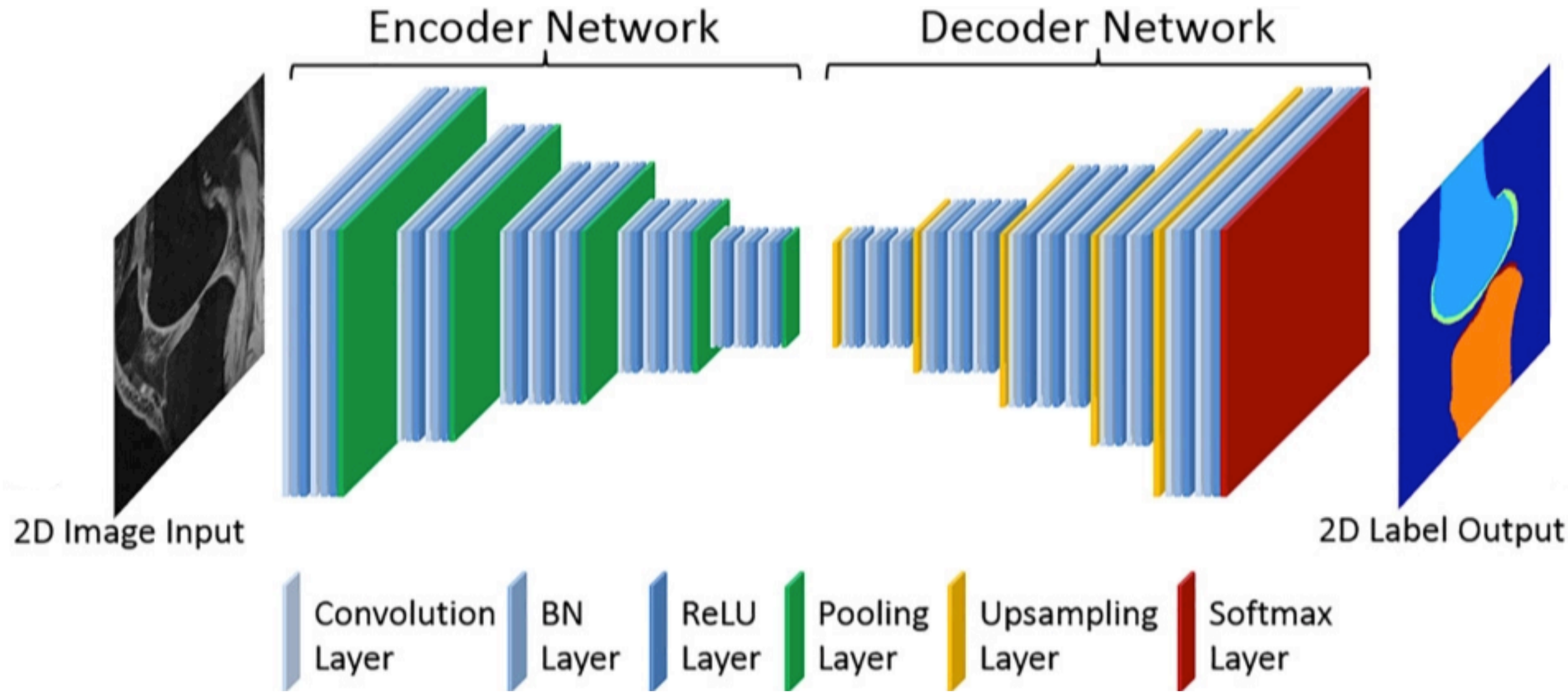




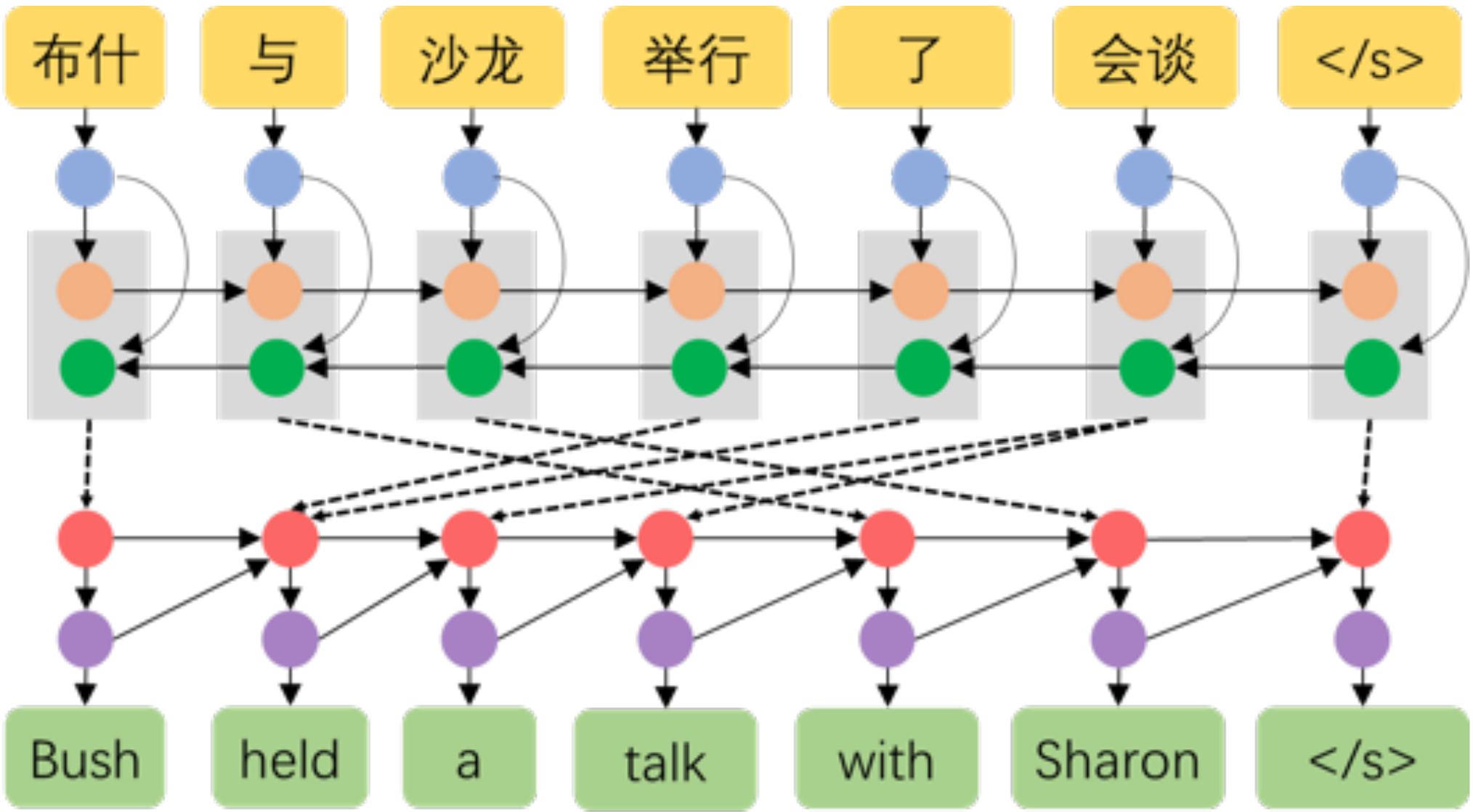
图像分类


block 1
block 2
block 3
block 4
Conv-Pool
Transition
Transition
Transition
Cat

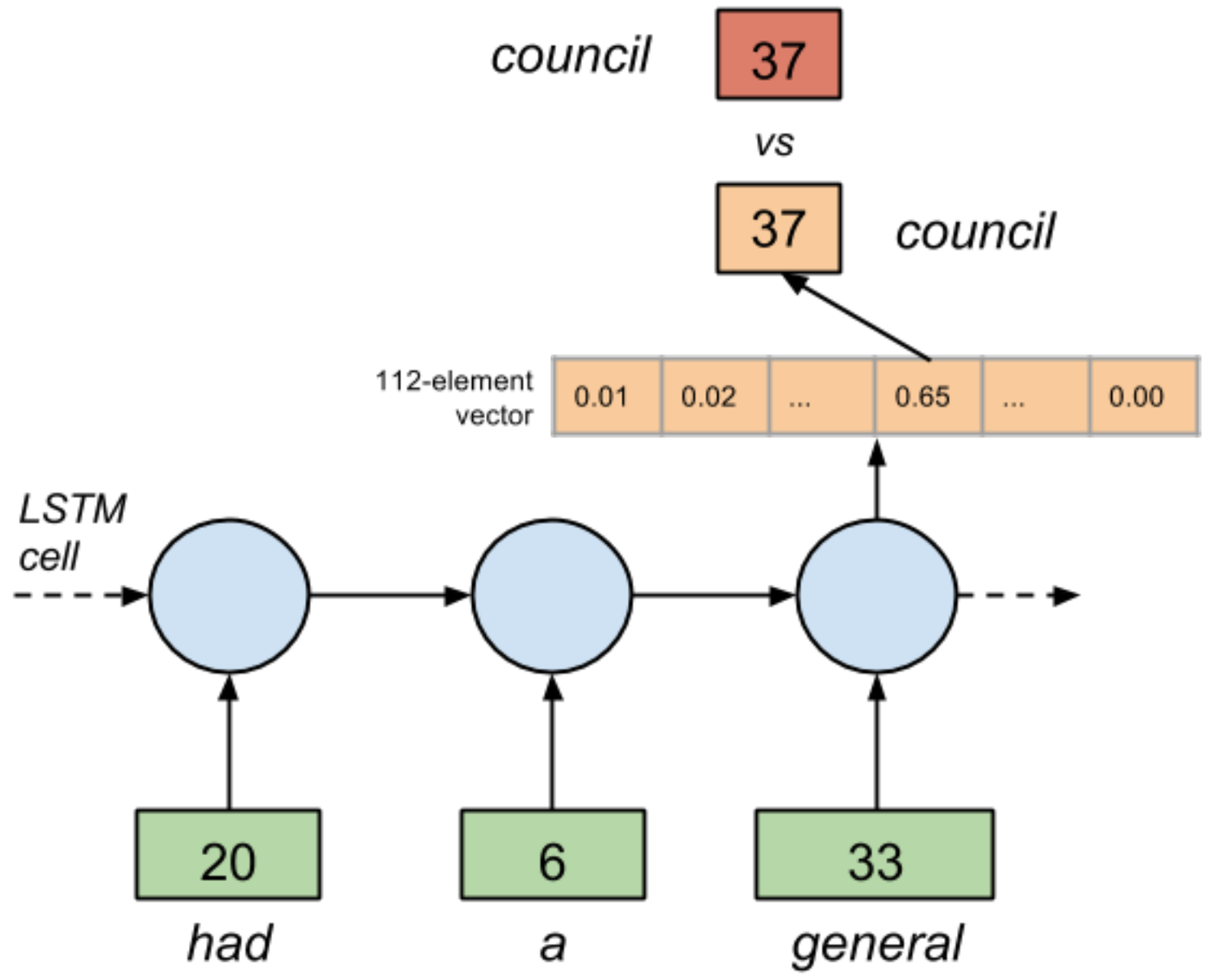
图像标注


Encoder Network
Decoder Network
2D Image Input
2D Label Output
Convolution Layer
BN Layer
ReLU Layer
Pooling Layer
Upsampling Layer
Softmax Layer

机器翻译



文本写作


council
37
vs
37
council
112-element vector
0.01
0.02
...
0.65
...
0.00
LSTM cell
20
6
33
had
a
general

多模态理解

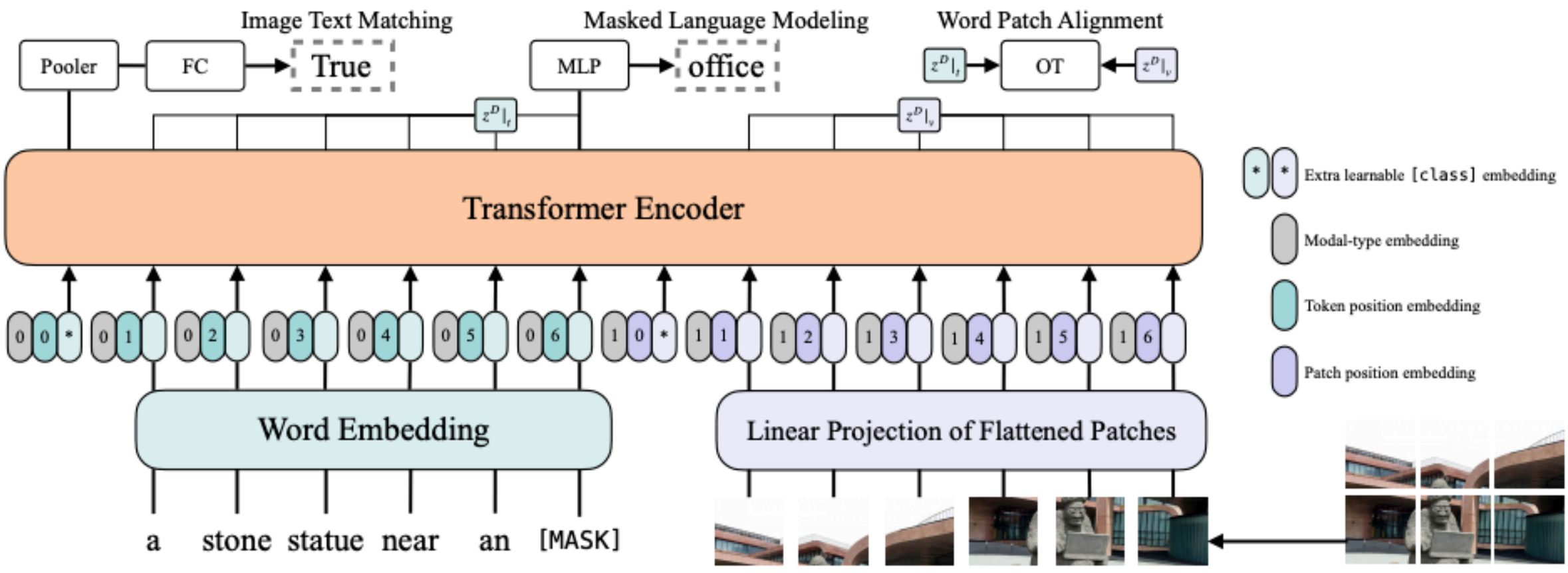
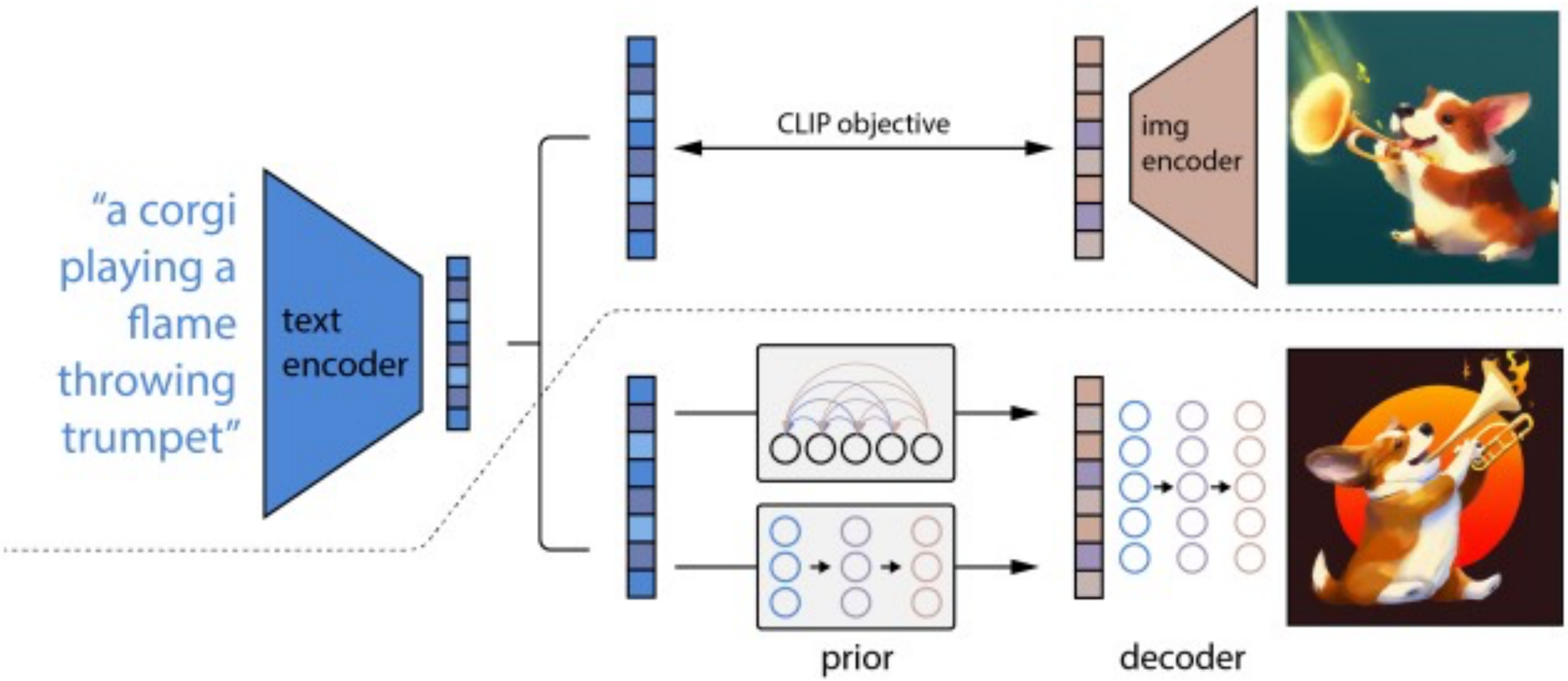


Image Text Matching
Masked Language Modeling
Word Patch Alignment
Pooler
FC
True
MLP
office
OT
Transformer Encoder
Extra learnable [class] embedding
Modal-type embedding
Token position embedding
Patch position embedding
Word Embedding
Linear Projection of Flattened Patches
a stone statue near an [MASK]

多模态生成


"a corgi playing a flame throwing trumpet"
text encoder
CLIP objective
img encoder
prior
decoder

作业

- 假设给定以下的 5 个编号的数据向量点，每个数据向量由 A 和 B 两个维度构成，标签列给出的是对应数据向量的分类，假设是二分类的情况。

数据索引	A	B	分类标签
1	1	1	-
2	3	2	+
3	2	4	+
4	3	4	+
5	2	3	-

- 请回答：

1.检查上面这些不同类别的数据点是否可以被线性分割辨别出来?

2.训练一个感知机分类器。假设初始权值向量是 [0,0]。依次使用上面给出的数据训练一轮该感知机模型，得到的权值向量是什么?

3.在2中得到的感知机是否可以正确分类以上数据?