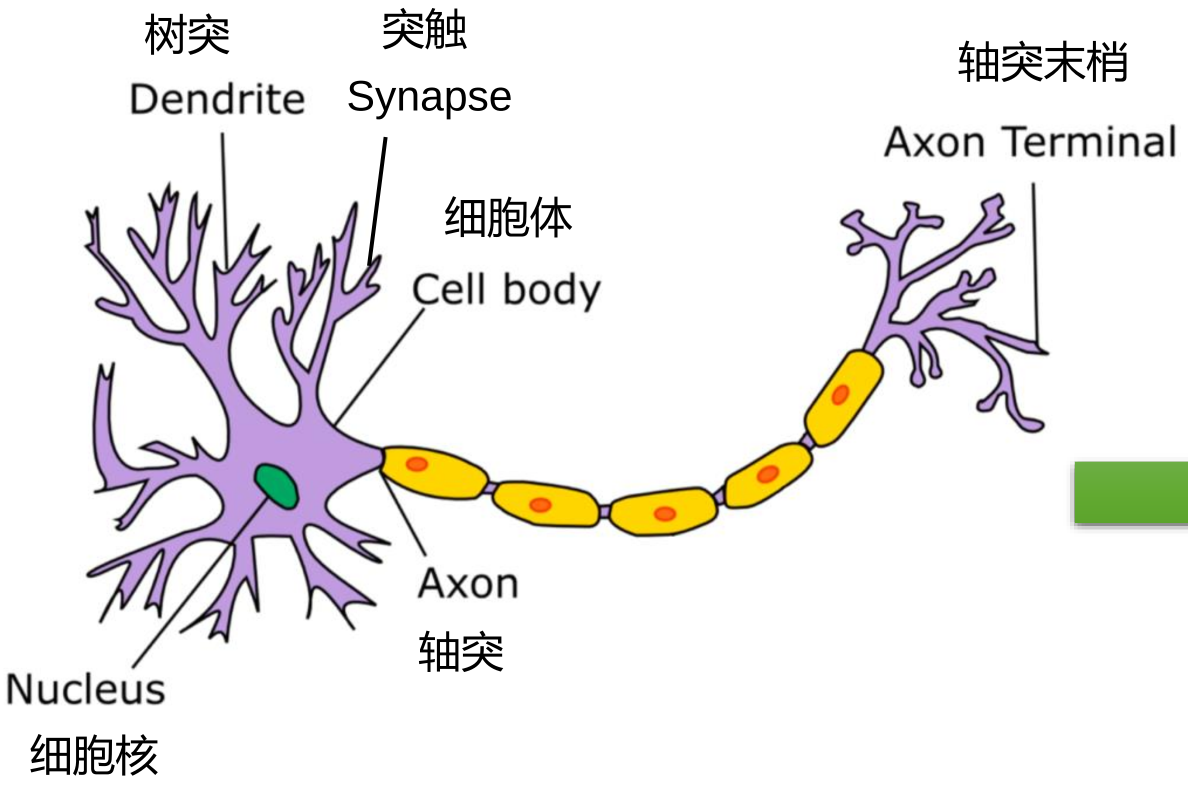
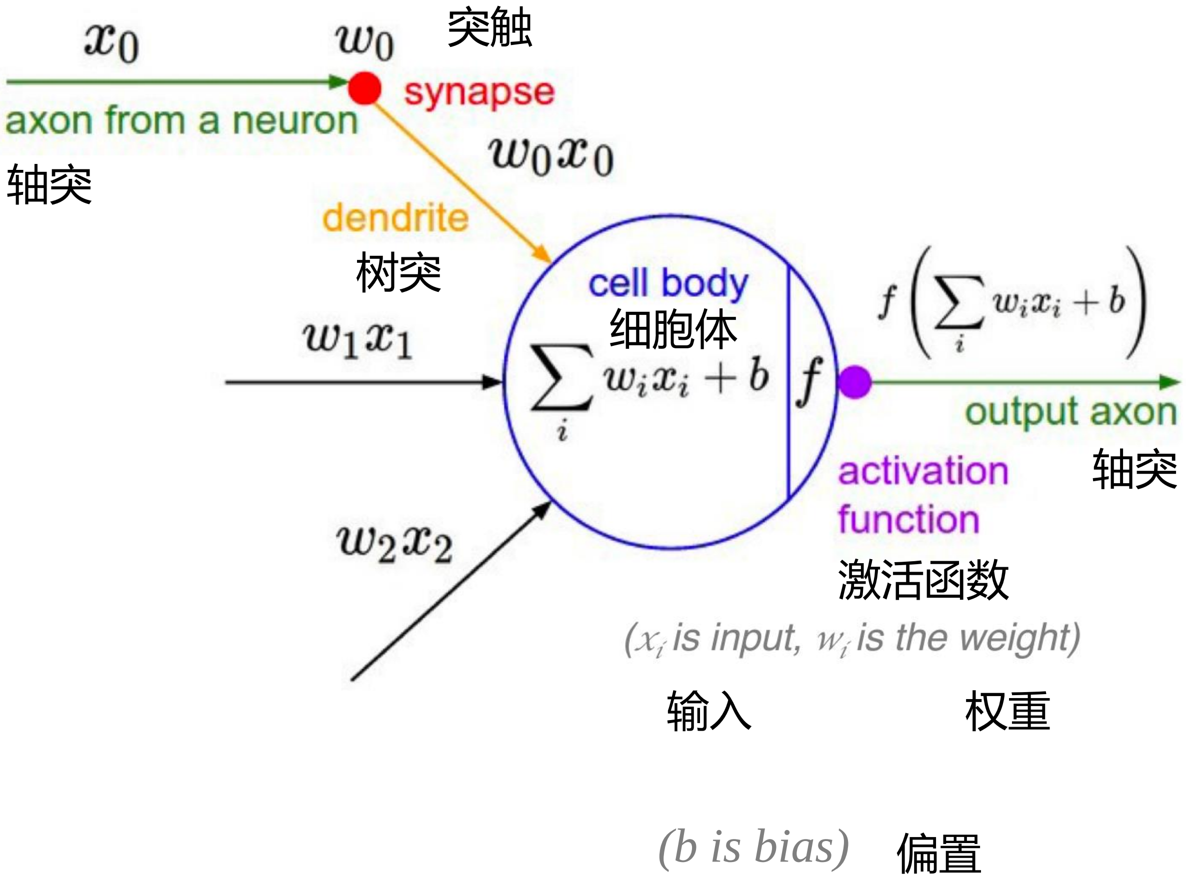
神经元（neuron）



感知机（Perceptron）

一种最简单形式的人工神经网络



三功能：加权，求和，激励

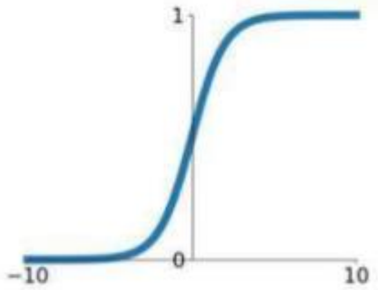
感知机的权重在训练过程中基于训练数据确定

激活函数：非线性处理单元

Activation functions

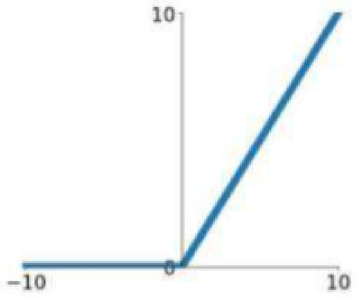
Sigmoid

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



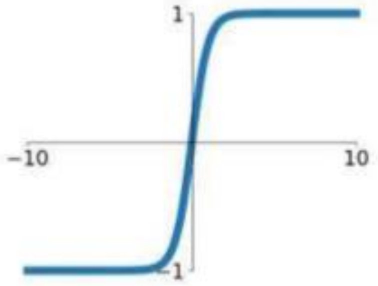
ReLU

$$\max(0, x)$$



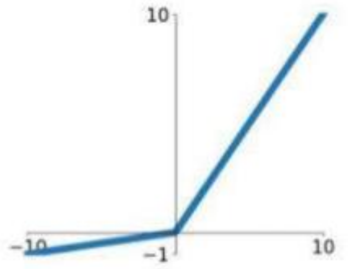
tanh

$$\tanh(x)$$



Leaky ReLU

$$\max(0.1x, x)$$

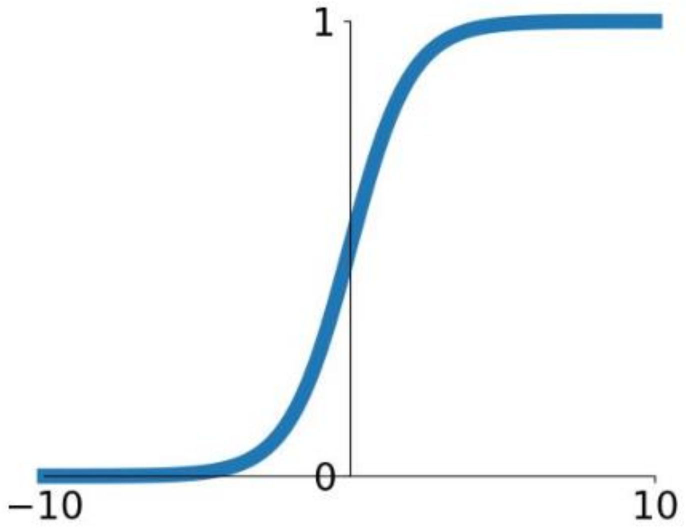


- 激活函数使神经网络具有非线性。它决定感知机是否激发。
- 激活函数的这种非线性赋予了深度网络学习复杂函数的能力。
- 除了在0点的修正单元以外，大多数激活函数都是连续函数和可微函数。

常用激活函数

- Sigmoid函数
- tanh函数
- ReLU函数
- Leaky ReLU函数
- ELU函数
- Softmax函数

Sigmoid函数



Sigmoid

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

- 将变量映射到 [0,1]
- Logistic函数的特例
- 可用于二分类
- 因可解释为神经元的饱和激发率(firing rate)，历史上比较流行

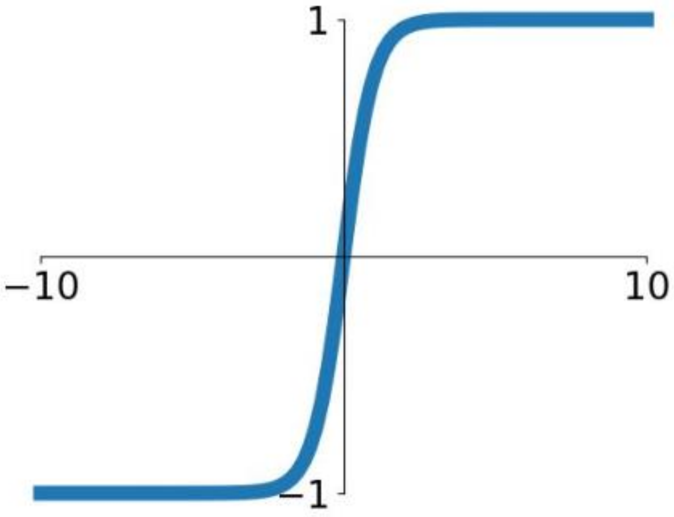
问题:

- 饱和神经元会“kill” 梯度（引起梯度消失）；
- Sigmoid输出不是零中心的;
- exp() 运算导致计算较复杂

activation.h

```
static inline float logistic_activate(float x){return 1./(1. + exp(-x));}
```

tanh函数



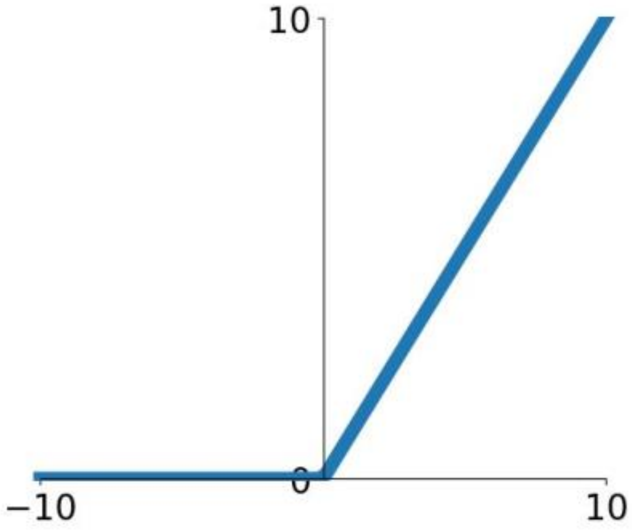
tanh(x)

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

- 将变量映射到 [-1,1]
- 输出零中心
- 饱和神经元仍然会“kill” 梯度

```
static inline float tanh_activate(float x){return (exp(2*x)-1)/(exp(2*x)+1);}
```

ReLU函数



ReLU
(Rectified Linear Unit)

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

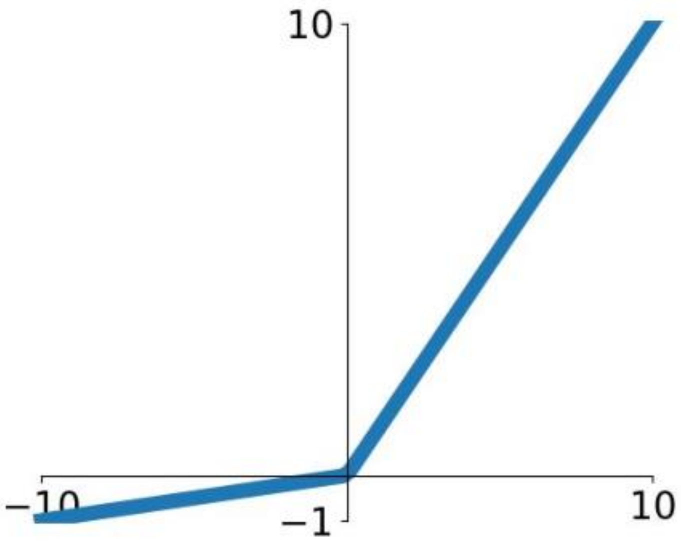
- 在$x > 0$时保持梯度不衰减，从而缓解梯度消失问题
- 计算效率高
- 实际应用中比sigmoid/tanh收敛速度快很多

问题:

- 输出非零中心
- $x < 0$无梯度，会导致权重无法更新

```
static inline float relu_activate(float x){return x*(x>0);}
```

Leaky ReLU函数



$$\sigma(x) = \max(0.01x, x)$$

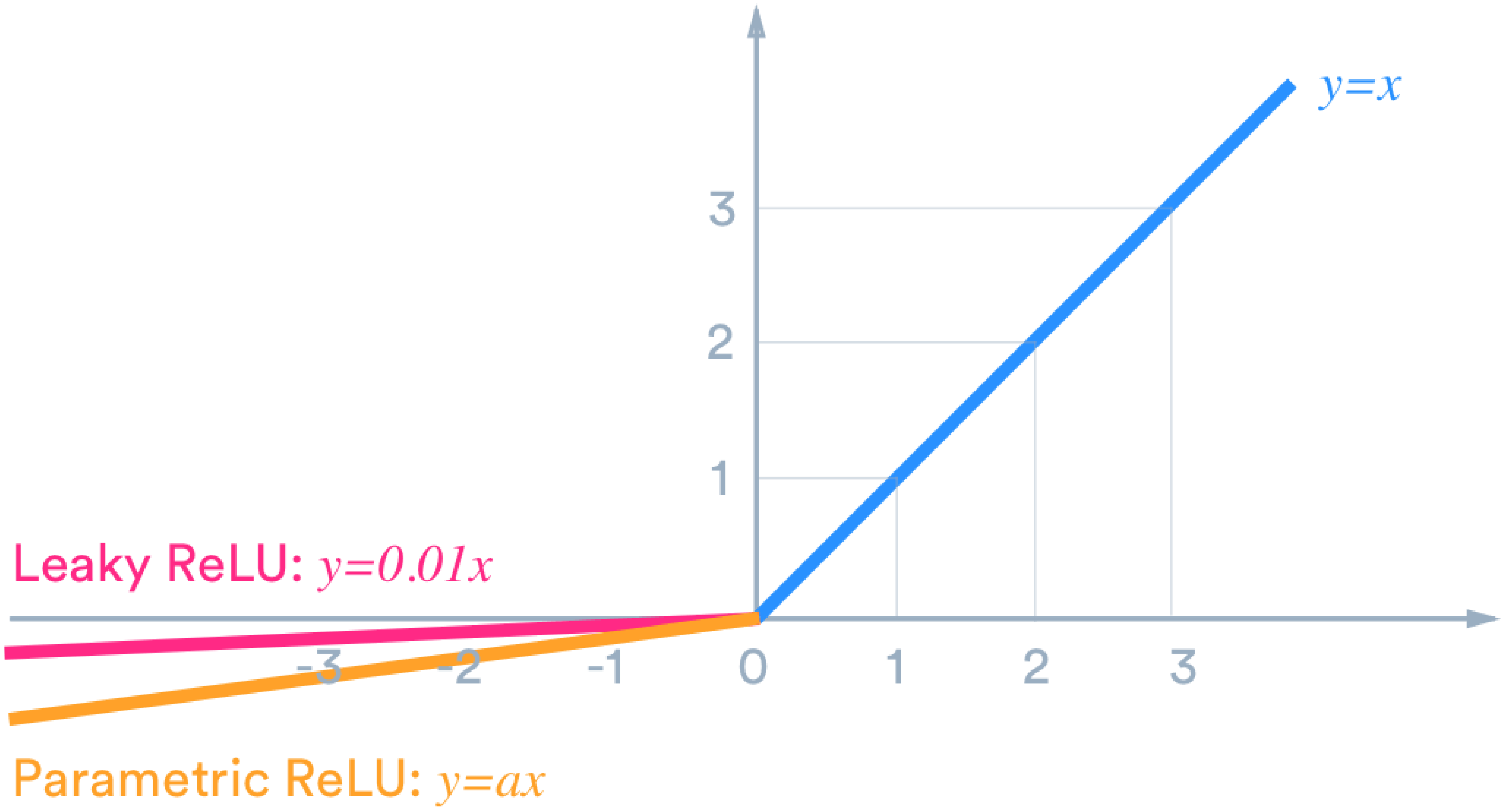
避免ReLU可能出现的神经元“死亡”现象

Leaky ReLU

Parametric Rectifier (PReLU)

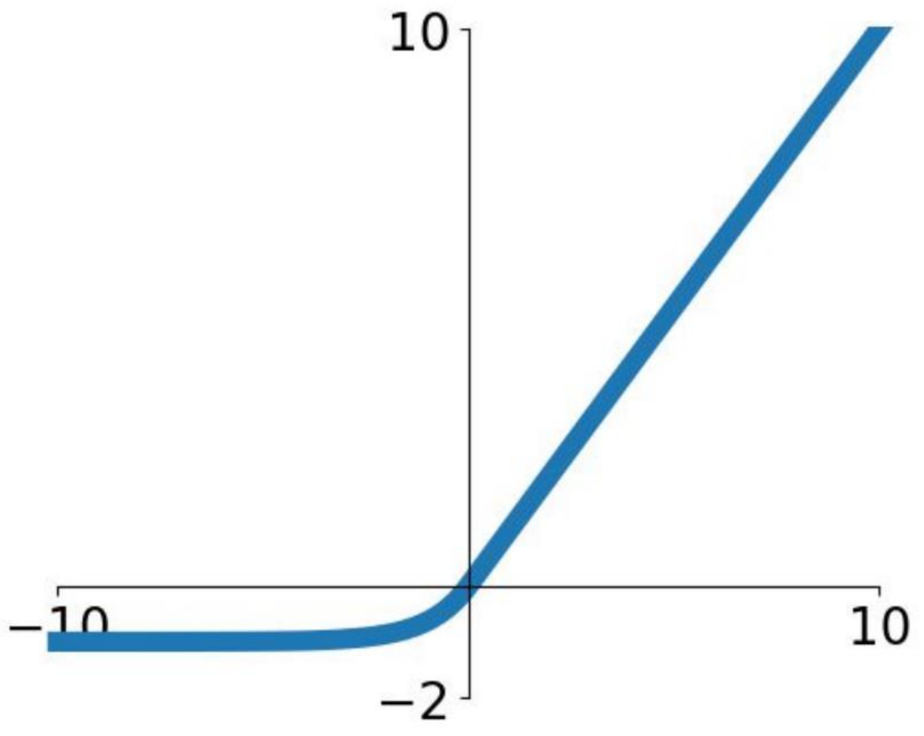
$$\sigma(x) = \max(\alpha x, x)$$

PReLU中的负半轴斜率α可学习而非固定




```
static inline float leaky_activate(float x){return (x>0) ? x : .1*x;}
```

ELU函数



$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(\exp(x) - 1) & \text{if } x \le 0 \end{cases}$$

- 具有ReLU函数的所有优点
- 输出均值接近零
- 负饱和区域相比Leaky ReLU增加了对噪声的鲁棒性

- exp() 运算较复杂

Exponential Linear Units (ELU)

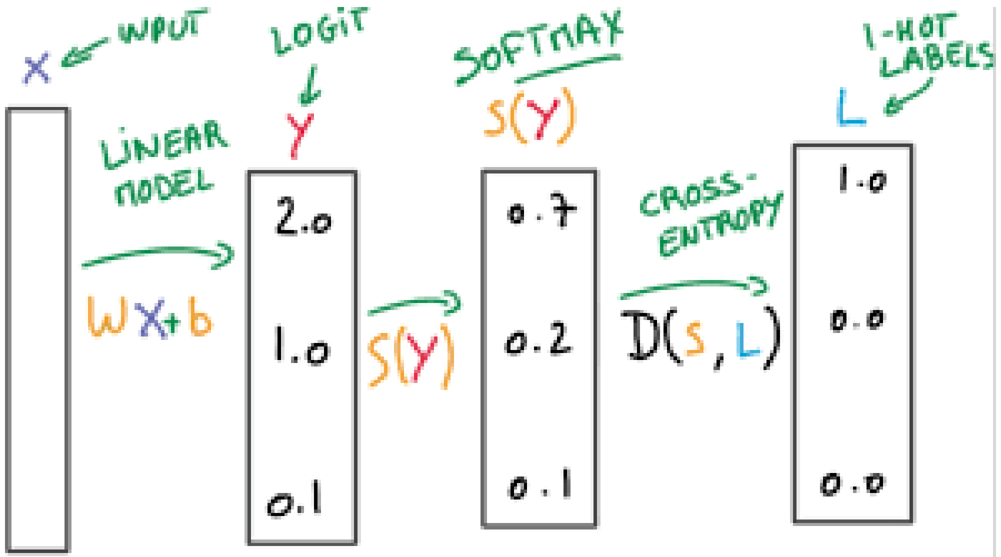
```
static inline float elu_activate(float x){return (x >= 0)*x + (x < 0)*(exp(x)-1);}
```

Softmax函数

- Softmax是一种特殊的激活函数，其输出总和为1
- 利用Softmax函数将线性预测值转换为多类别对应的概率

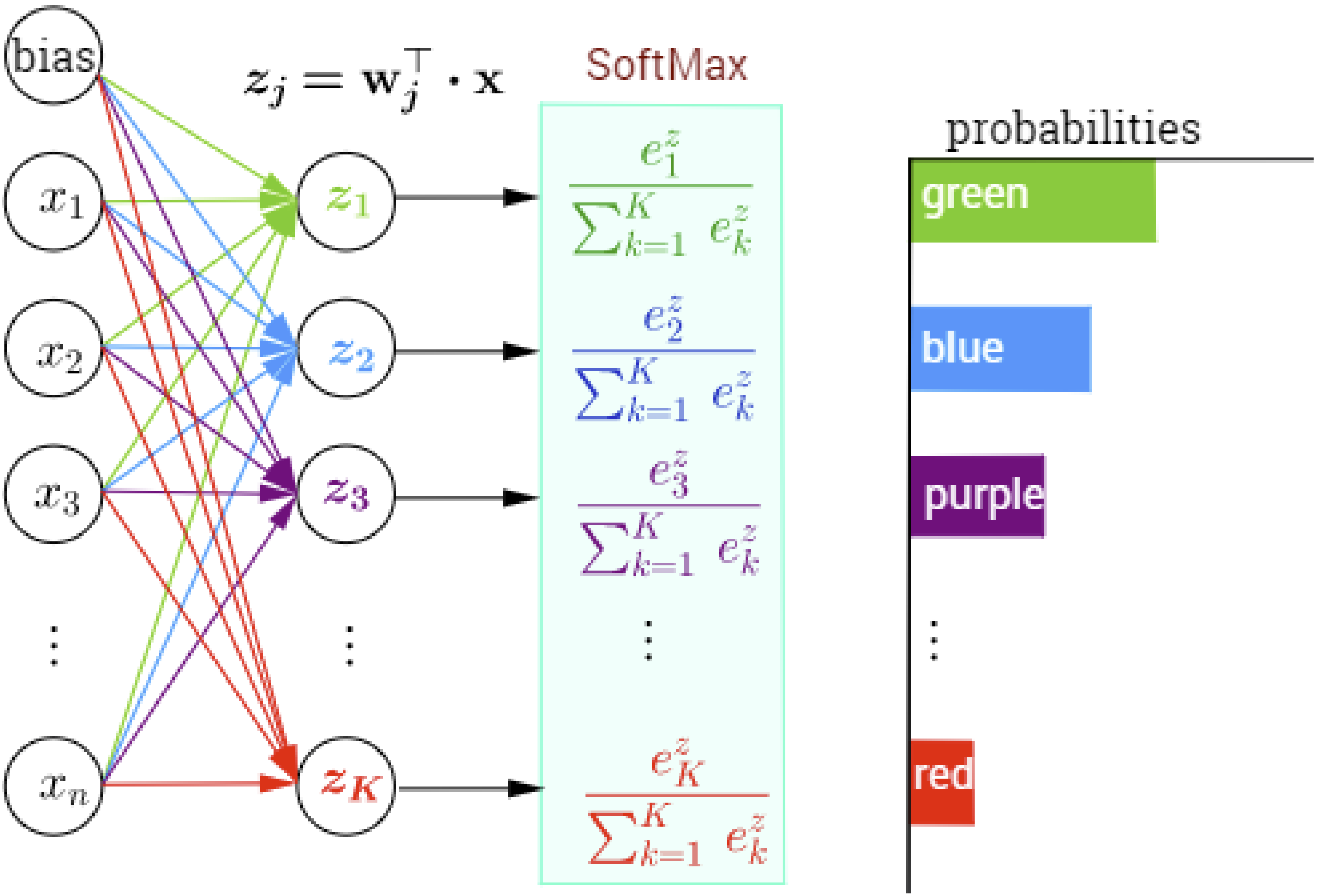
$$\sigma_i(z) = \frac{\exp(z_i)}{\sum_{j=1}^{K} \exp(z_j)}, \quad i = 1, \dots, K$$

Softmax其实就是先对每一个 z_i 取指数变成非负，然后除以所有项之和进行归一化

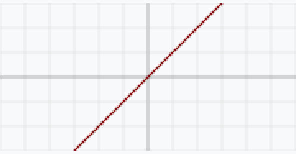
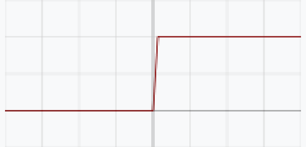
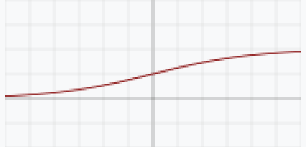
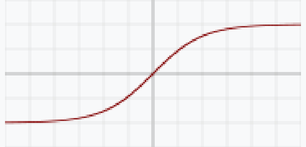
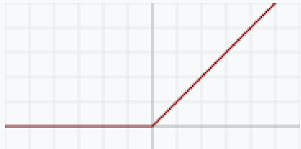
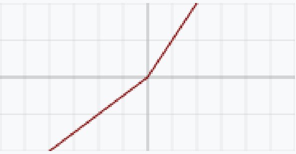


“logit”

$$\mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_K \end{bmatrix} = \begin{bmatrix} \mathbf{w}_1^\top \\ \mathbf{w}_2^\top \\ \mathbf{w}_3^\top \\ \vdots \\ \mathbf{w}_K^\top \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$$



激活函数的导数

Name	Plot	Equation	Derivative (with respect to *x*)
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. Sigmoid or Soft step)		$f(x)=\sigma(x)=\frac{1}{1+e^{-x}}$ [1]	$f'(x)=f(x)(1-f(x))$
TanH		$f(x)=\tanh(x)=\frac{(e^{x}-e^{-x})}{(e^{x}+e^{-x})}$	$f'(x)=1-f(x)^2$
Rectified linear unit (ReLU)[15]		$f(x)=\begin{cases}0 & \text{for } x<0\\ x & \text{for } x\geq 0\end{cases}$	$f'(x)=\begin{cases}0 & \text{for } x<0\\ 1 & \text{for } x\geq 0\end{cases}$
Leaky rectified linear unit (Leaky ReLU)[17]		$f(x)=\begin{cases}0.01x & \text{for } x<0\\ x & \text{for } x\geq 0\end{cases}$	$f'(x)=\begin{cases}0.01 & \text{for } x<0\\ 1 & \text{for } x\geq 0\end{cases}$

Name	Equation	Derivatives	Range
Softmax	$f_i(\vec{x}) = \frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}}$ for i = 1, …, J	$\frac{\partial f_i(\vec{x})}{\partial x_j} = f_i(\vec{x})(\delta_{ij} - f_j(\vec{x}))$[7]	$(0,1)$
Maxout[31]	$f(\vec{x}) = \max_i x_i$	$\frac{\partial f}{\partial x_j} = \begin{cases} 1 & \text{for } j = \underset{i}{\operatorname{argmax}}\, x_i \\ 0 & \text{for } j \neq \underset{i}{\operatorname{argmax}}\, x_i \end{cases}$	$(-\infty,\infty)$

Here, δ_{ij} is the Kronecker delta.

$$\delta_{ij} = \begin{cases} 0 & \text{if } i \neq j, \\ 1 & \text{if } i = j. \end{cases}$$

activation.h

```c
// 返回线性激活函数（就是f(x)=x）关于输入x的导数值
static inline float linear_gradient(float x){return 1;}
// 返回logistic (sigmoid) 函数关于输入x的导数值
static inline float logistic_gradient(float x){return (1-x)*x;}
```

```c
// 返回ReLU非线性激活函数关于输入x的导数值
static inline float relu_gradient(float x){return (x>0);}
// 返回指数线性单元（Exponential Linear Unit, ELU）非线性激活函数关于输入x的导数值
static inline float elu_gradient(float x){return (x >= 0) + (x < 0)*(x + 1);}
```

```c
// 返回leaky ReLU非线性激活函数关于输入x的导数值
static inline float leaky_gradient(float x){return (x>0) ? 1 : .1;}
// 返回tanh非线性激活函数关于输入x的导数值
static inline float tanh_gradient(float x){return 1-x*x;}
```

Softmax temperature

$$q_i = \frac{\exp\left(z_i / T\right)}{\sum_j \exp\left(z_j / T\right)}$$

$$\mathbf{z} = \left(z_1, \ldots, z_n\right)$$

$$\mathbf{q} = \left(q_1, \ldots, q_n\right)$$

Temperature increases the sensitivity to low probability candidates.

当T很大时，即趋于正无穷时，所有的激活值对应的激活概率趋近于相同（激活概率差异性较小）；而当T很低时，即趋于0时，不同的激活值对应的激活概率差异也就越大。

spatial softmax

Spatial softmax is defined in End-to-End Training of Deep Visuomotor Policies (https://arxiv.org/abs/1504.00702)

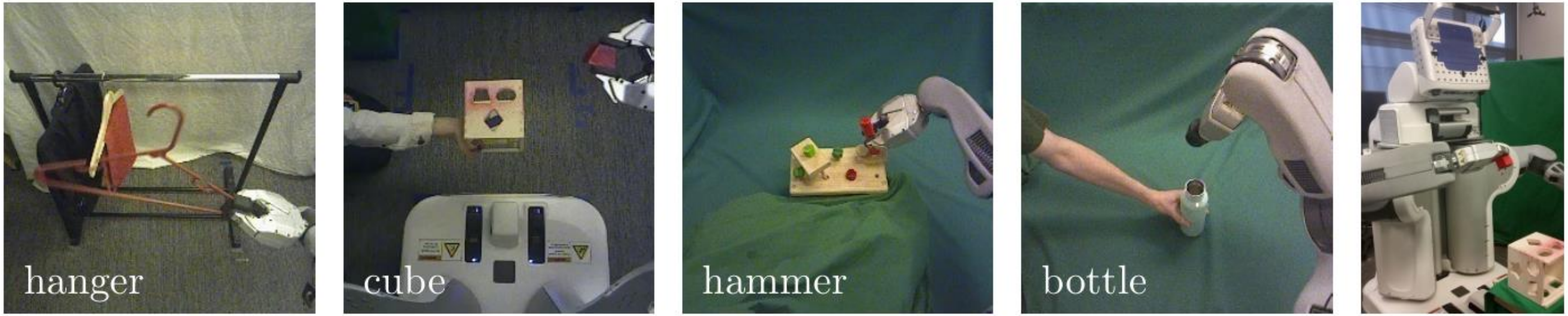
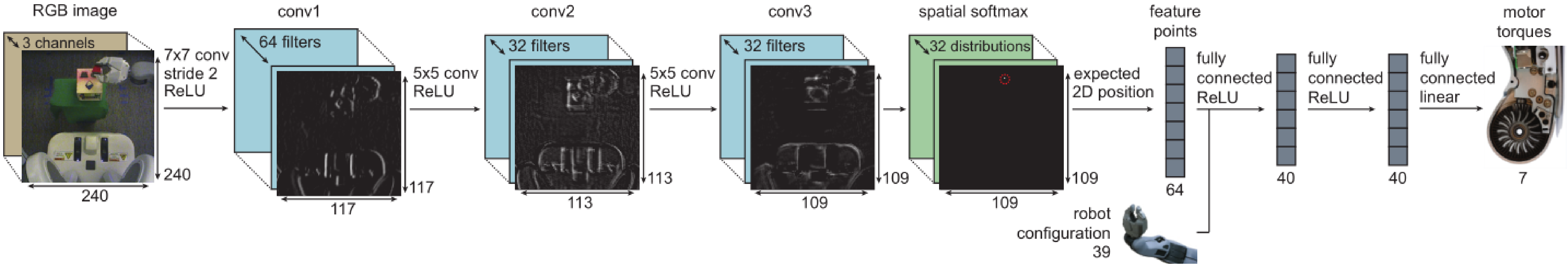


Figure 1: Our method learns visuomotor policies that directly use camera image observations (left) to set motor torques on a PR2 robot (right).



spatial softmax

The response maps are passed through a spatial softmax function of the form

$$s_{cij} = e^{a_{cij}} / \sum_{i'j'} e^{a_{ci'j'}}$$

$a_{cij} = \max(0, z_{cij})$ for each channel c and each pixel coordinate (i, j).

Each output channel of the softmax is a probability distribution over the location of a feature in the image.

To convert from this distribution to a coordinate representation $\left(f_{cx}, f_{cy}\right)$

The network calculates the expected image position of each feature, yielding a 2D coordinate for each channel:

$$f_{cx} = \sum_{ij} s_{cij} x_{ij} \qquad f_{cy} = \sum_{ij} s_{cij} y_{ij}$$

where $\left(x_{ij}, y_{ij}\right)$ is the image-space position of the point (i,j) in the response map.

The combination of the spatial softmax and expectation operator implement a kind of soft-argmax.