最优化方法

深度学习及其优化算法

张伯雷

南京邮电大学 计算机学院

bolei.zhang@njupt.edu.cn

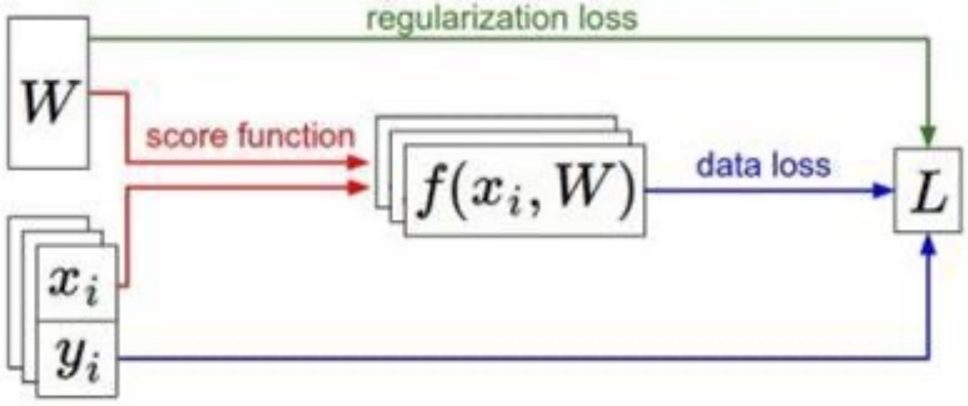
http://bolei-zhang.github.io/course/opt.html

上一节课

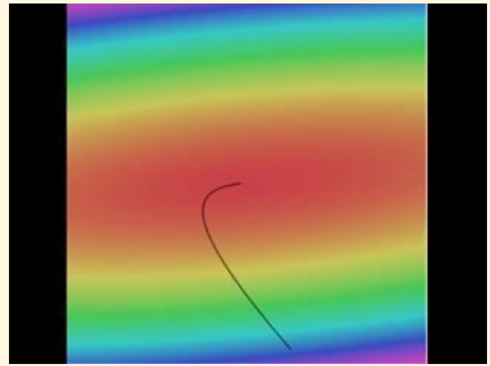
- 数据集 (X, y)
- 模型 $f(x; W)$
- 损失函数

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$ Softmax

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i + R(W)$$ Full loss



梯度下降法



```
# Vanilla Gradient Descent

while True:
  weights_grad = evaluate_gradient(loss_fun, data, weights)
  weights += - step_size * weights_grad # perform parameter update
```

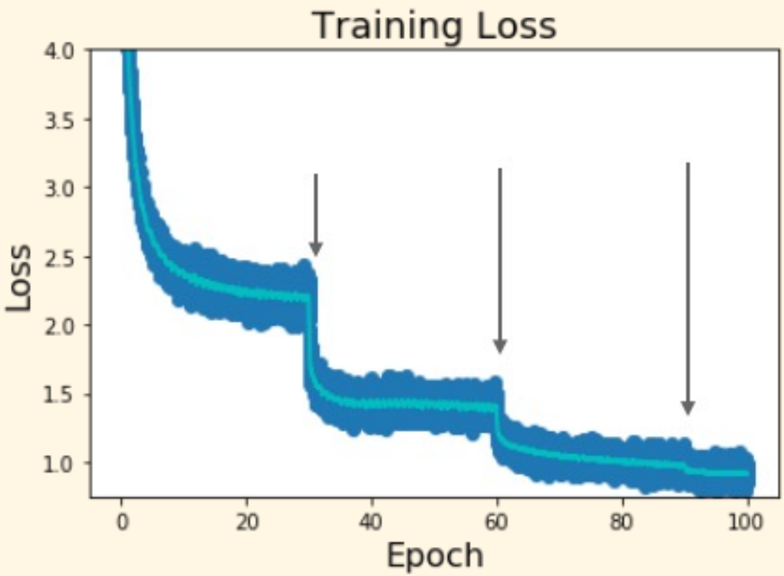
Stochastic Gradient Descent (SGD)

$$L(W) = \frac{1}{N} \sum_{i=1}^{N} L_i(x_i, y_i, W) + \lambda R(W)$$

$$\nabla_W L(W) = \frac{1}{N} \sum_{i=1}^{N} \nabla_W L_i(x_i, y_i, W) + \lambda \nabla_W R(W)$$

```
# Vanilla Minibatch Gradient Descent

while True:
  data_batch = sample_training_data(data, 256) # sample 256 examples
  weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
  weights += - step_size * weights_grad # perform parameter update
```

学习率（步长）



步骤衰减：在几个固定节点降低学习率。

例如，对于 ResNets，在第 30、60 和 90 个 epoch 后将学习率乘以 0.1。

DALL-E 2

“Teddy bears working on new AI research on the moon in the 1980s.”

“Rabbits attending a college seminar on human anatomy.”

“A wise cat meditating in the Himalayas searching for enlightenment.”

Image source: Sam Altman, https://openai.com/dall-e-2/, https://twitter.com/sama/status/1511724264629678084

DALL-E 3

GPT-4

User What is unusual about this image?

Source: Barnorama

GPT-4 The unusual thing about this image is that a man is ironing clothes on an ironing board attached to the roof of a moving taxi.

User Can you explain this meme?

GPT-4 This meme is a joke that combines two unrelated things: pictures of the earth from space and chicken nuggets.
The text of the meme suggests that the image below is a beautiful picture of the earth from space. However, the image is actually of chicken nuggets arranged to vaguely resemble a map of the world.
The humor in this meme comes from the unexpected juxtaposition of the text and the image. The text sets up an expectation of a majestic image of the earth, but the image is actually something mundane and silly.

SAM

Sora

A Shiba Inu dog wearing a beret and black turtleneck.

put the video in space with a rainbow road

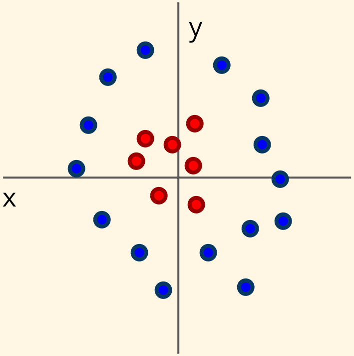
change the video setting to be different than a mountain? perhaps joshua tree

神经网络

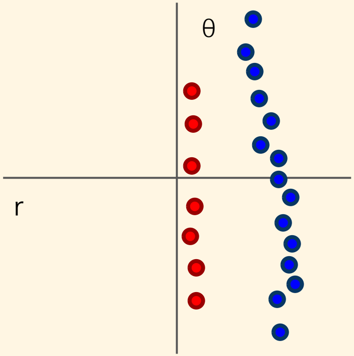
- 线性模型 $f(x;W) = Wx$
- 两层的神经网络

$$f = W_2 \max(0, W_1 x)$$

引入非线性层



$$f(x,y) = (r(x,y), \theta(x,y))$$



神经网络

- 线性模型 $f(x;W) = Wx$
- 两层的神经网络

$$f = W_2 \max(0, W_1 x)$$

- 三层

$$f = W_3 \max(0, W_2 \max(0, W_1 x))$$

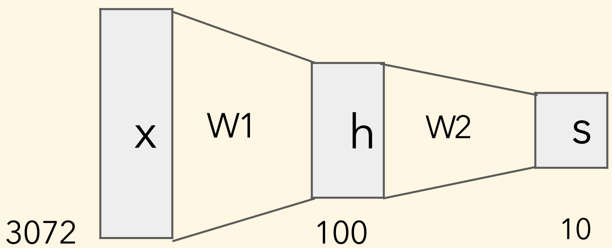
全连接网络（fully connected network）

多层感知机（multi layer perceptrons）

神经网络

- 线性模型 $f(x;W) = Wx$
- 两层的神经网络

$$f = W_2 \max(0, W_1 x)$$



神经网络

- 线性模型 $f(x;W) = Wx$
- 两层的神经网络

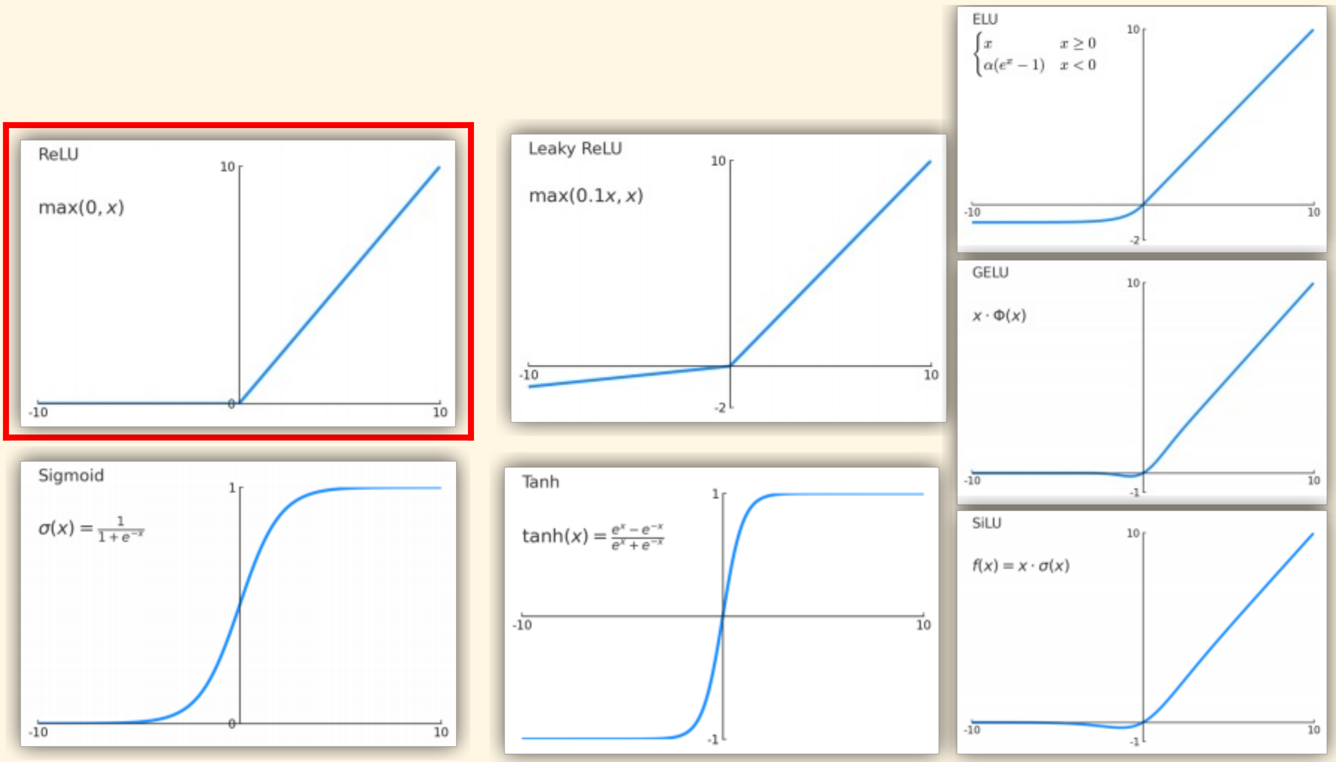
$$f = W_2 \max(0, W_1 x)$$

- 激活层的作用： $\max(0, W_1 x)$
  - 如果没有激活层

$$f = W_2 W_1 x$$

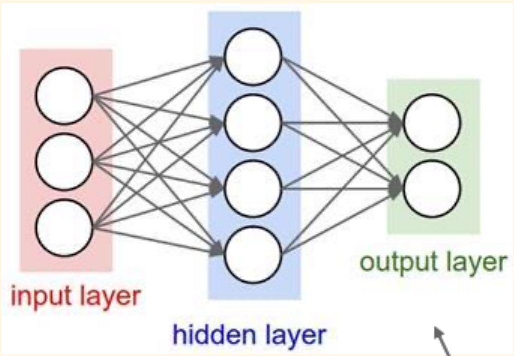
$$W_3 = W_2 W_1 \in \mathbb{R}^{C \times H}, f = W_3 x$$

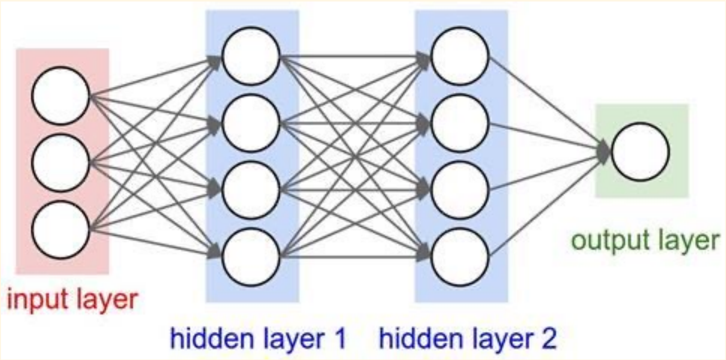
激活函数



ReLU
$\max(0, x)$
Leaky ReLU
$\max(0.1x, x)$
ELU
$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$
GELU
$x \cdot \Phi(x)$
Sigmoid
$\sigma(x) = \frac{1}{1+e^{-x}}$
Tanh
$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
SiLU
$f(x) = x \cdot \sigma(x)$

神经网络结构



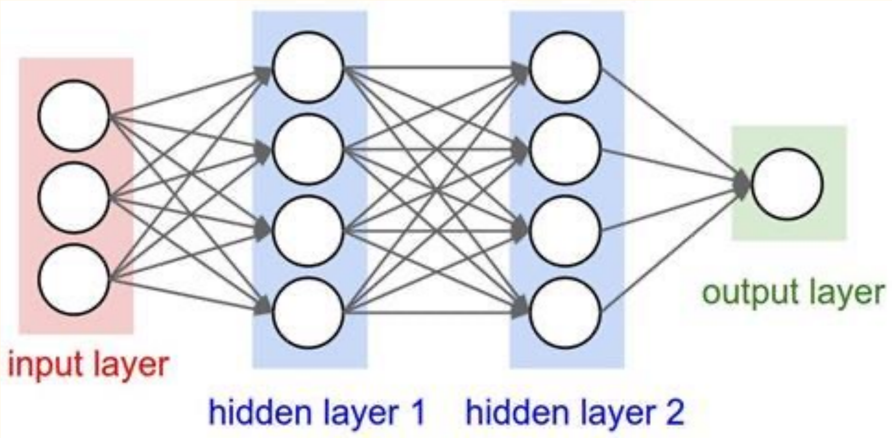


“2-layer Neural Net”, or
“1-hidden-layer Neural Net”

全连接层

“3-layer Neural Net”, or
“2-hidden-layer Neural Net”

feed-forward

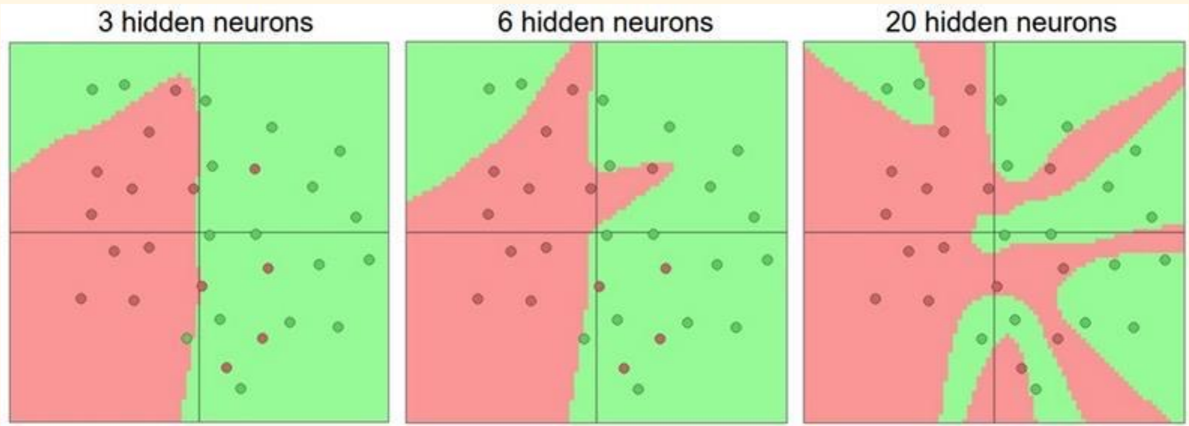



```
# forward-pass of a 3-layer neural network:
f = lambda x: 1.0/(1.0 + np.exp(-x)) # activation function (use sigmoid)
x = np.random.randn(3, 1) # random input vector of three numbers (3x1)
h1 = f(np.dot(W1, x) + b1) # calculate first hidden layer activations (4x1)
h2 = f(np.dot(W2, h1) + b2) # calculate second hidden layer activations (4x1)
out = np.dot(W3, h2) + b3 # output neuron (1x1)
```

训练一个两层的神经网络

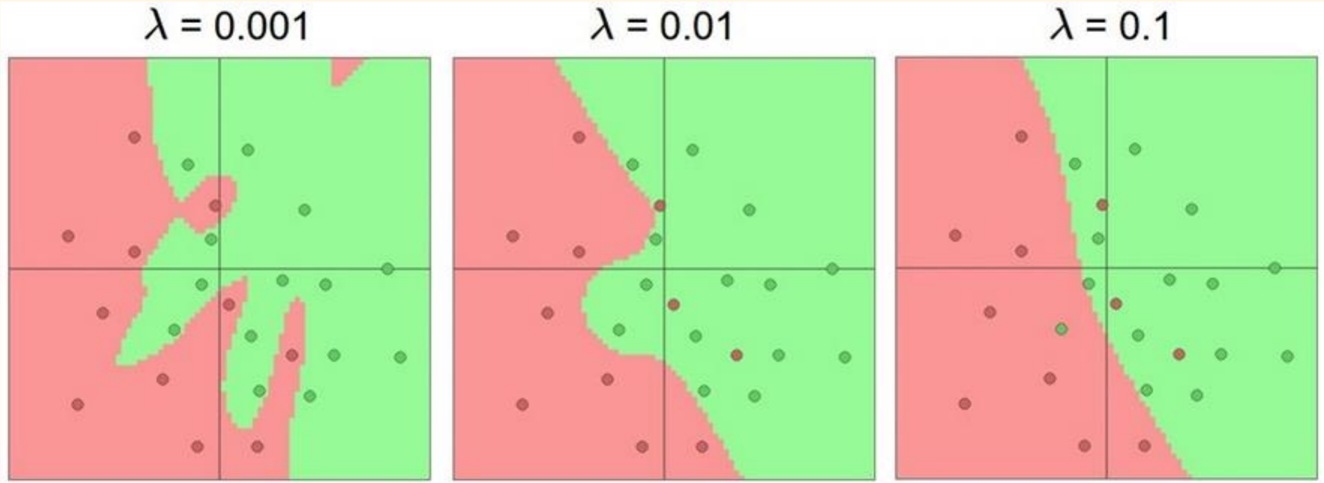
```
import numpy as np
from numpy.random import randn

N, D_in, H, D_out = 64, 1000, 100, 10
x, y = randn(N, D_in), randn(N, D_out)
w1, w2 = randn(D_in, H), randn(H, D_out)

for t in range(2000):
  h = 1 / (1 + np.exp(-x.dot(w1)))
  y_pred = h.dot(w2)
  loss = np.square(y_pred - y).sum()
  print(t, loss)

  grad_y_pred = 2.0 * (y_pred - y)
  grad_w2 = h.T.dot(grad_y_pred)
  grad_h = grad_y_pred.dot(w2.T)
  grad_w1 = x.T.dot(grad_h * h * (1 - h))

  w1 -= 1e-4 * grad_w1
  w2 -= 1e-4 * grad_w2
```

神经网络的层数与神经元的个数



more neurons = more capacity

使用正则化项，而不是神经网络大小



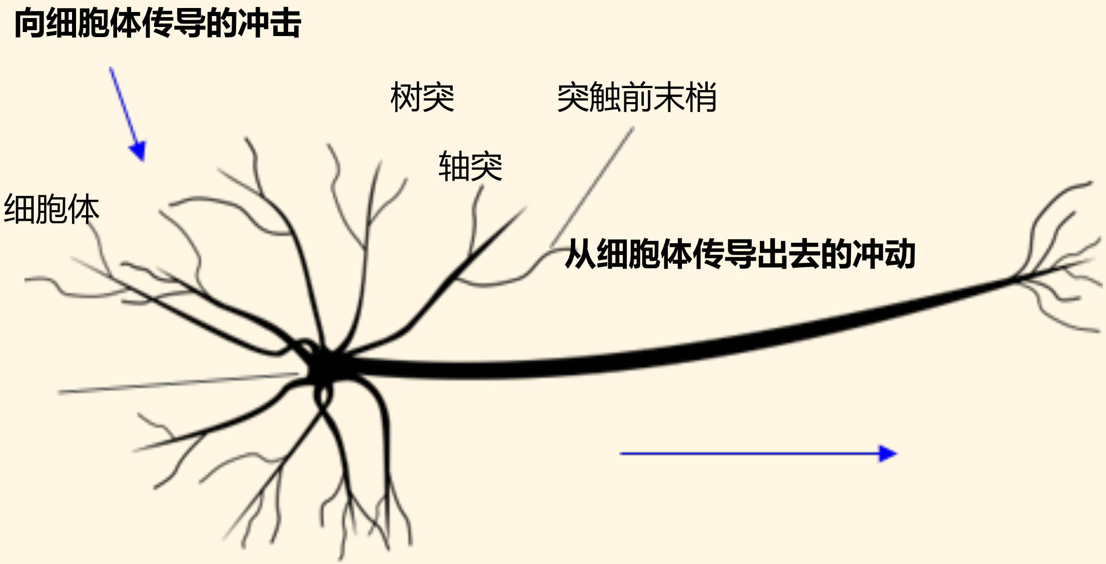
(Web demo with ConvNetJS:
http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html)

TensorFlow Play Ground: https://playground.tensorflow.org/

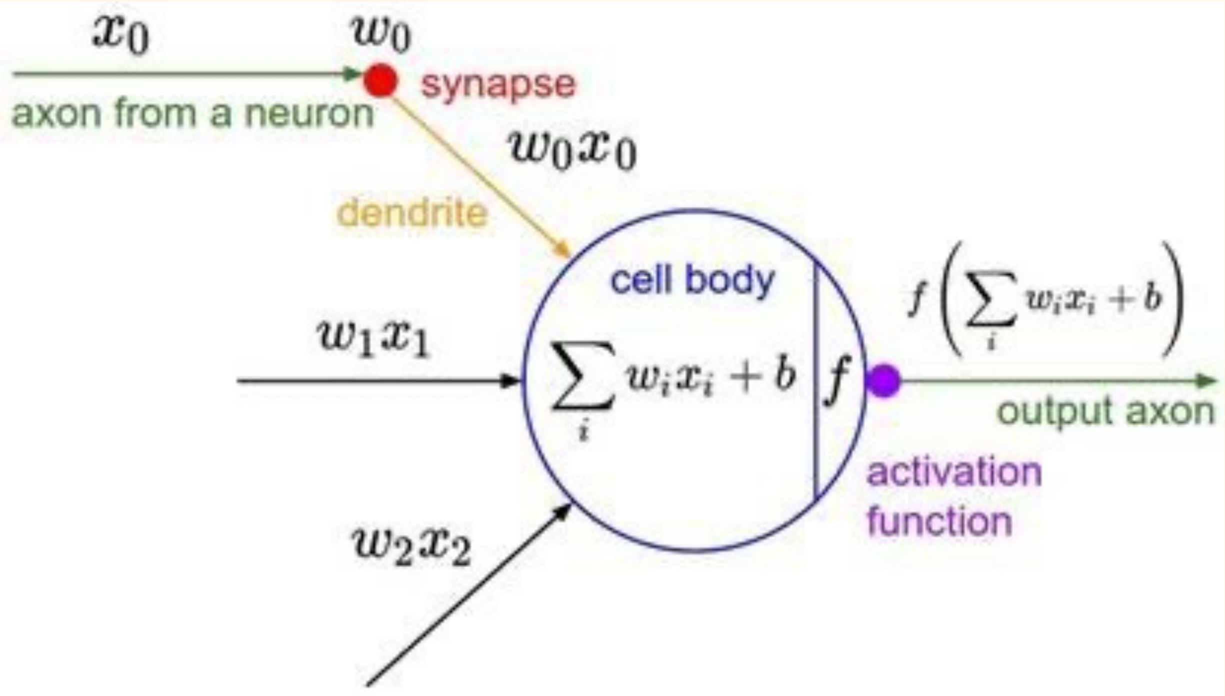
$$L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i) + \lambda R(W)$$

神经元

神经元



神经网络神经元

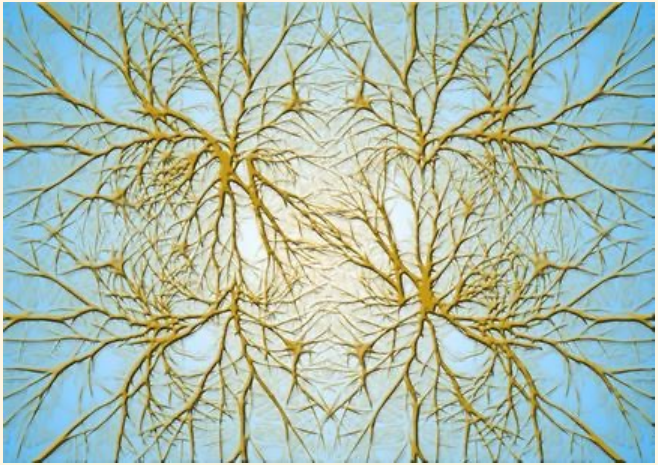


x_0
w_0
synapse
axon from a neuron
$w_0 x_0$
dendrite
cell body
$f\left(\sum_i w_i x_i + b\right)$
$w_1 x_1$
$\sum_i w_i x_i + b$
f
output axon
activation
function
$w_2 x_2$

对比

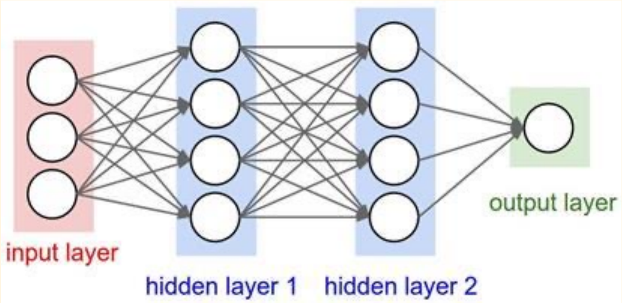
生物神经元：

复杂连接模型



神经网络神经元：

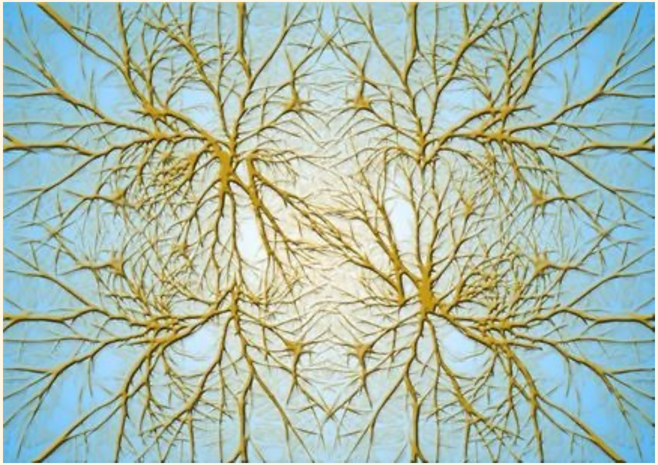
比较规则的连接模式



对比

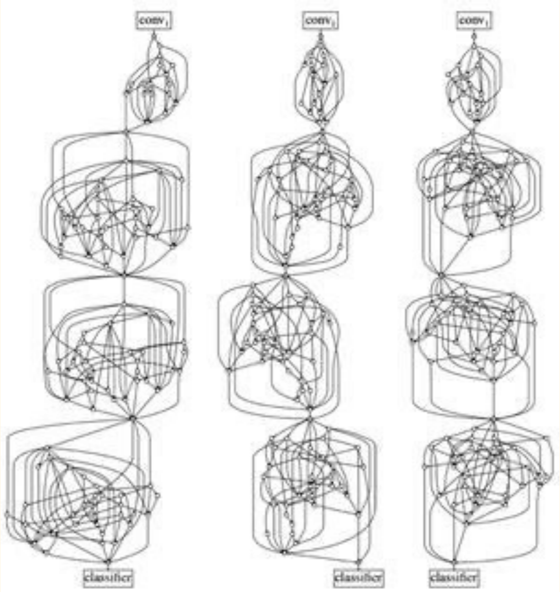
生物神经元:

复杂连接模型



通过激活层，人工神经网络

也会非常复杂



区别

生物神经元的特点:

- 存在多种不同类型
- 树突能够执行复杂的非线性计算
- 突触并非单一权重，而是复杂的非线性动力学系统

损失函数

$$s = f(x; W_1, W_2) = W_2 \max(0, W_1 x)$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$R(W) = \sum_k W_k^2$$ 正则化项

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + \lambda R(W_1) + \lambda R(W_2)$$

非线性损失函数，Hinge Loss

总的损失：Hinge Loss+正则化

如何计算梯度

$$s = f(x; W_1, W_2) = W_2 \max(0, W_1 x)$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

非线性损失函数，Hinge Loss

$$R(W) = \sum_k W_k^2$$

正则化项

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + \lambda R(W_1) + \lambda R(W_2)$$

总的损失：Hinge Loss+正则化

核心是计算损失函数关于权重的梯度$\frac{\partial L}{\partial W_1}$, $\frac{\partial L}{\partial W_2}$

如果直接推导梯度

$$s = f(x;W) = Wx$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$= \sum_{j \neq y_i} \max(0, W_{j,:} \cdot x + W_{y_i,:} \cdot x + 1)$$

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + \lambda \sum_k W_k^2$$

$$= \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, W_{j,:} \cdot x + W_{y_i,:} \cdot x + 1) + \lambda \sum_k W_k^2$$

$$\nabla_W L = \nabla_W \left(\frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, W_{j,:} \cdot x + W_{y_i,:} \cdot x + 1) + \lambda \sum_k W_k^2 \right)$$

极其繁琐 —— 涉及大量矩阵微积分运算，需要耗费大量纸张。

想更改损失函数该怎么办？例如用 softmax 替代合页损失函数？必须从头重新推导所有内容！

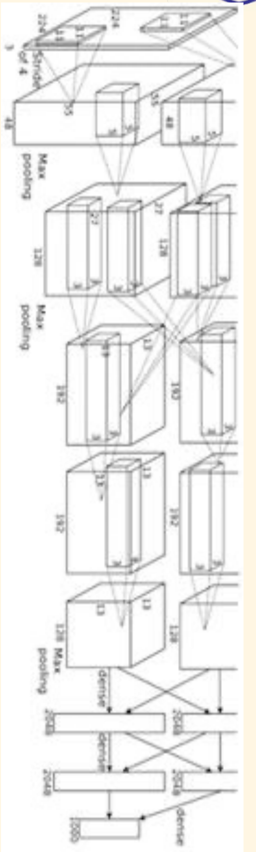
对于极为复杂的模型而言完全不可行！

卷积神经网络（AlexNet）

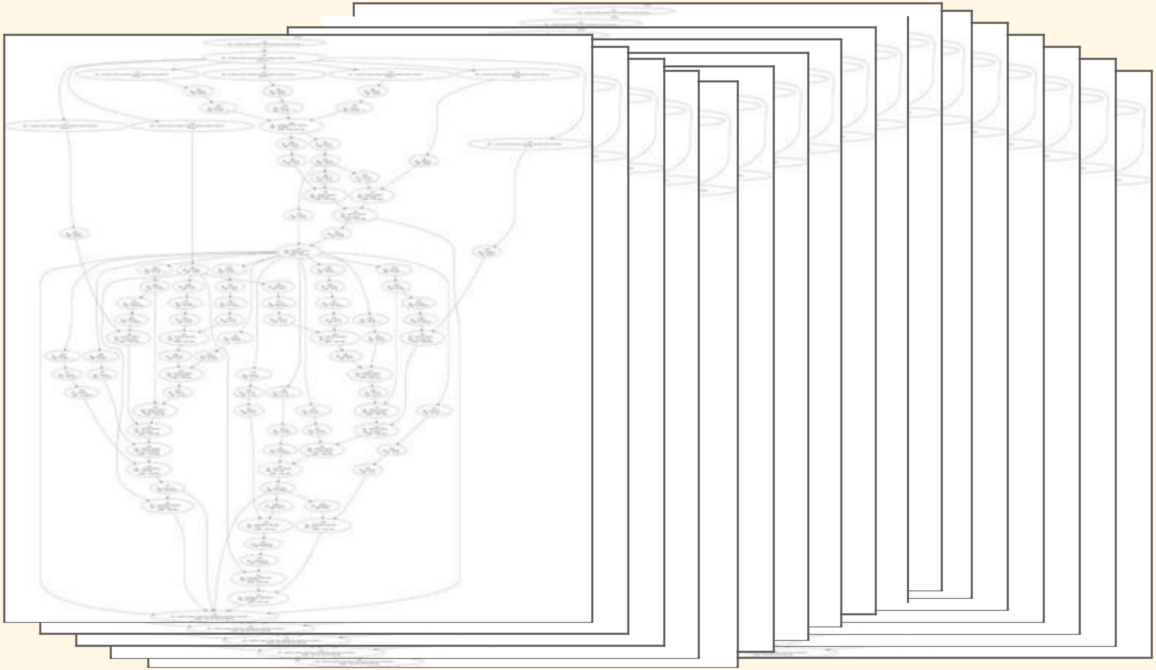
input image

weights

loss

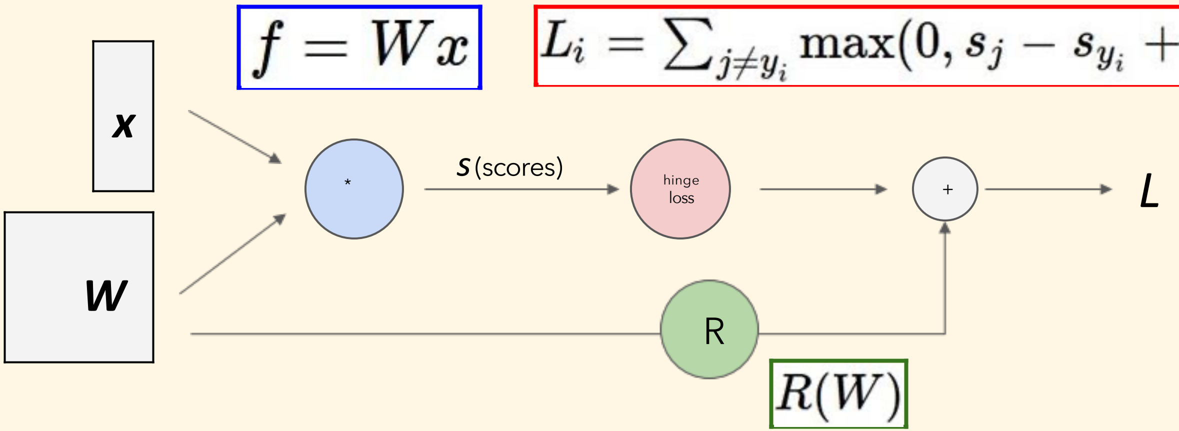


Neural Turing Machine



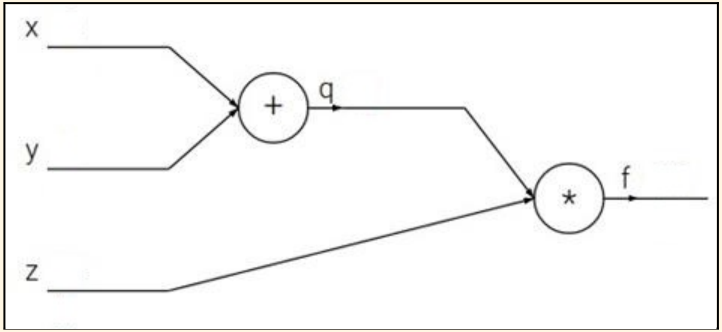
Backpropagation

更好的方法: 计算图+ Backpropagation



Backpropagation

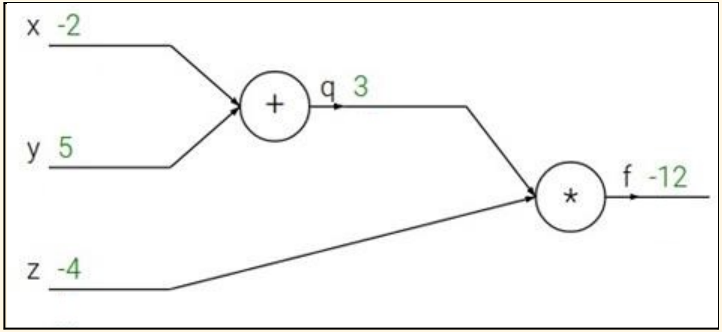
$$f(x, y, z) = (x + y)z$$



Backpropagation

$$f(x,y,z) = (x+y)z$$

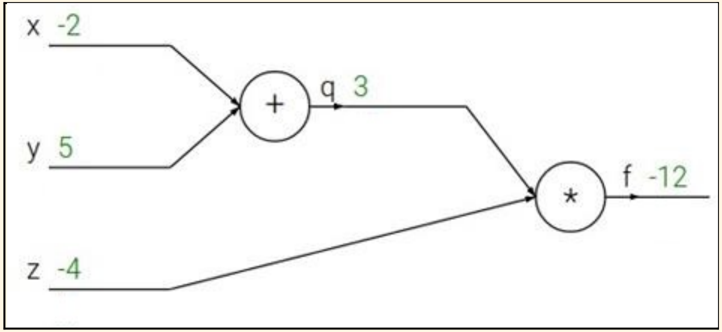
e.g. x = -2, y = 5, z = -4



Backpropagation

$$f(x, y, z) = (x + y)z$$

e.g. $x = -2, y = 5, z = -4$



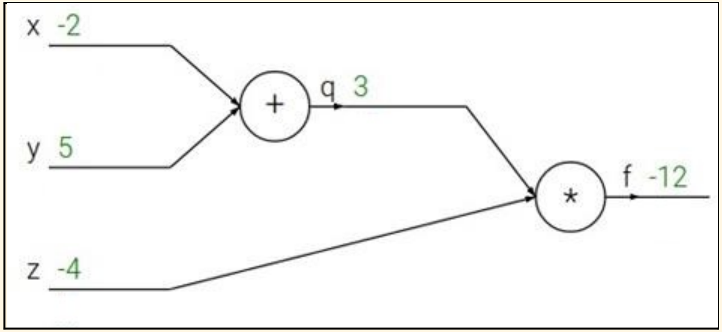
$$\boxed{q = x + y}$$

$$\frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

Backpropagation

$$f(x,y,z) = (x+y)z$$

e.g. $x = -2, y = 5, z = -4$



$$\boxed{q = x + y}$$

$$\frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

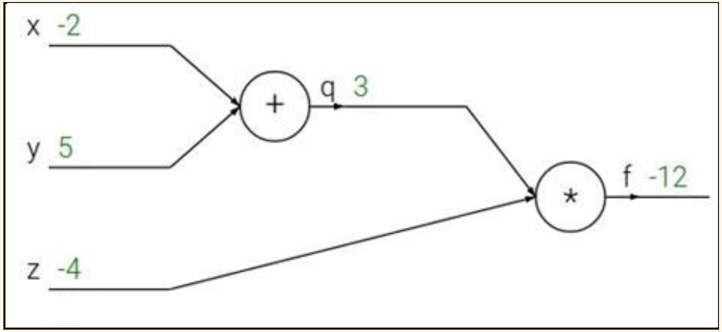
$$\boxed{f = qz}$$

$$\frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Backpropagation

$$f(x, y, z) = (x + y)z$$

e.g. $x = -2, y = 5, z = -4$



$$\boxed{q = x + y}$$

$$\frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$\boxed{f = qz}$$

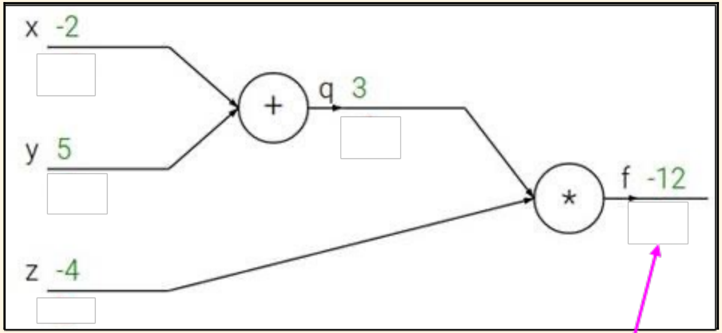
$$\frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

$$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$$

Backpropagation

$$f(x, y, z) = (x + y)z$$

e.g. $x = -2, y = 5, z = -4$



$\frac{\partial f}{\partial f}$

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

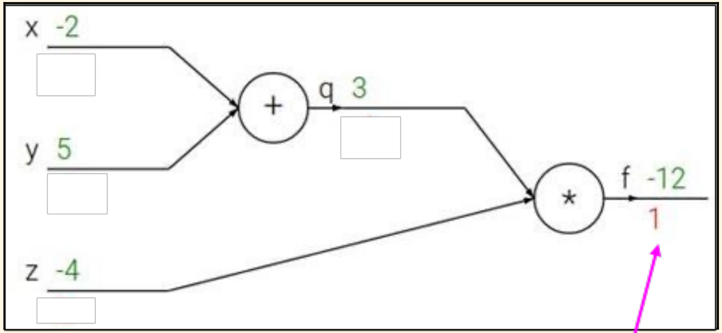
$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

$$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$$

Backpropagation

$$f(x,y,z) = (x+y)z$$

e.g. $x = -2, y = 5, z = -4$



$$\frac{\partial f}{\partial f}$$

$$\boxed{q = x + y} \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

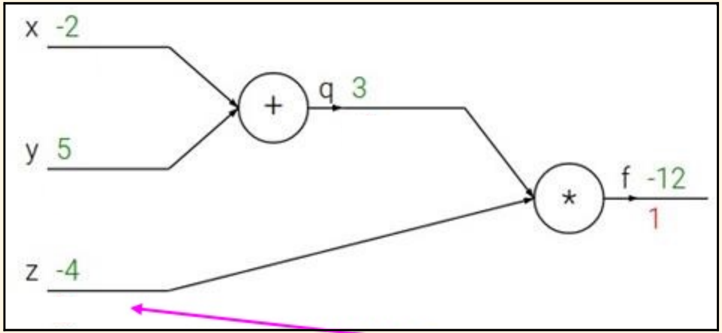
$$\boxed{f = qz} \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

$$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$$

Backpropagation

$$f(x, y, z) = (x + y)z$$

e.g. $x = -2, y = 5, z = -4$



$$\boxed{\frac{\partial f}{\partial z}}$$

$$\boxed{q = x + y}$$ $$\frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

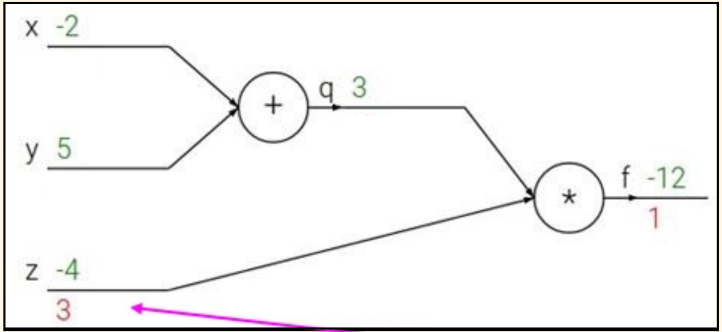
$$\boxed{f = qz}$$ $$\frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

$$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$$

Backpropagation

$$f(x, y, z) = (x + y)z$$

e.g. $x = -2, y = 5, z = -4$



$$\frac{\partial f}{\partial z}$$

$$\boxed{q = x + y}$$ $$\frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

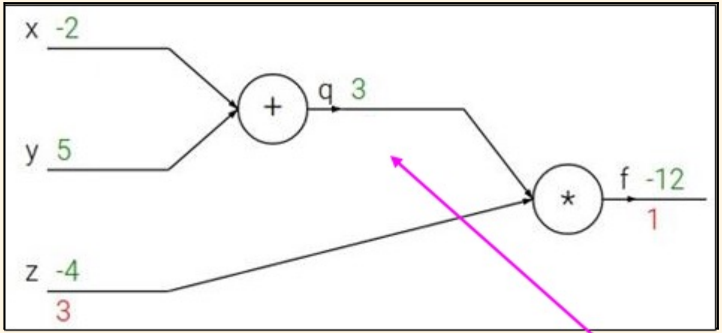
$$\boxed{f = qz}$$ $$\frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

$$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$$

Backpropagation

$$f(x, y, z) = (x + y)z$$

e.g. $x = -2, y = 5, z = -4$



$$\frac{\partial f}{\partial q}$$

$$\boxed{q = x + y}$$

$$\frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$\boxed{f = qz}$$

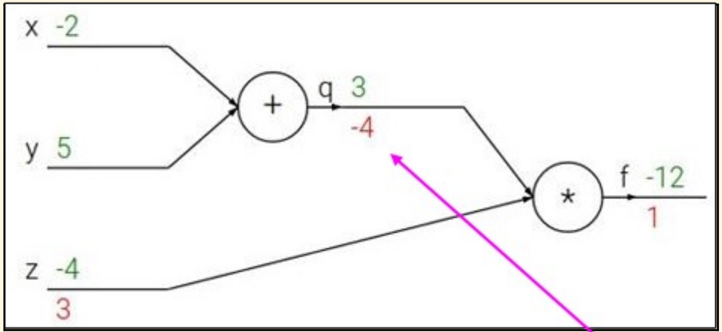
$$\frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

$$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$$

Backpropagation

$$f(x, y, z) = (x + y)z$$

e.g. $x = -2, y = 5, z = -4$



$$\frac{\partial f}{\partial q}$$

$$\boxed{q = x + y}$$

$$\frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$\boxed{f = qz}$$

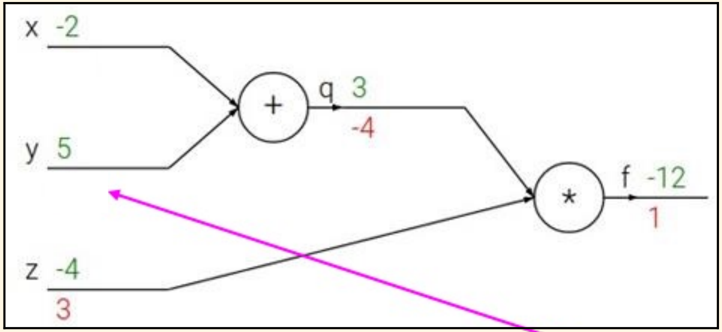
$$\frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

$$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$$

Backpropagation

$$f(x,y,z) = (x+y)z$$

e.g. $x = -2, y = 5, z = -4$



$$\frac{\partial f}{\partial y}$$

$$\boxed{q = x + y} \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$\boxed{f = qz} \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

$$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$$

链式法则

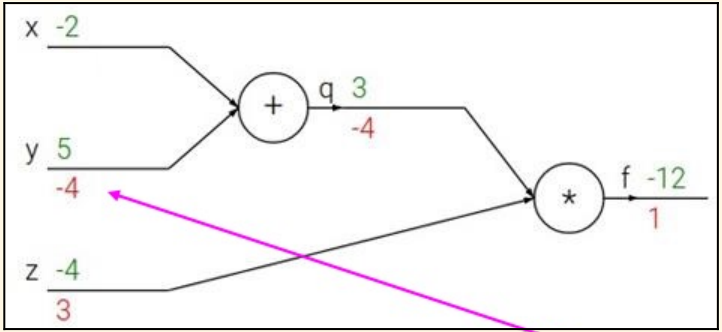
$$\boxed{\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial y}}$$

上游梯度 局部梯度

Backpropagation

$$f(x, y, z) = (x + y)z$$

e.g. $x = -2, y = 5, z = -4$



$$\frac{\partial f}{\partial y}$$

$$\boxed{q = x + y}$$

$$\frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$\boxed{f = qz}$$

$$\frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

$$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$$

链式法则

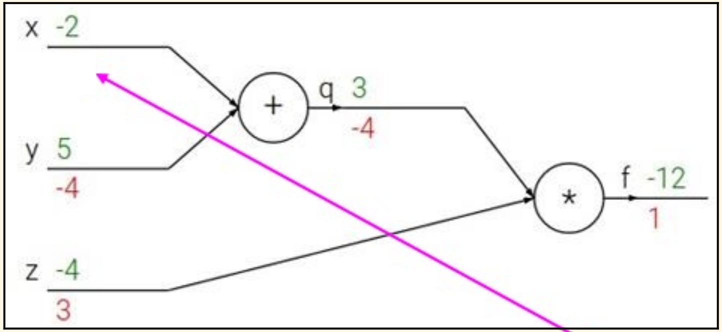
$$\boxed{\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial y}}$$

上游梯度 局部梯度

Backpropagation

$$f(x,y,z) = (x+y)z$$

e.g. $x = -2, y = 5, z = -4$



$$\frac{\partial f}{\partial x}$$

$$\boxed{q = x + y}$$

$$\frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$\boxed{f = qz}$$

$$\frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

$$\boxed{\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial y}}$$

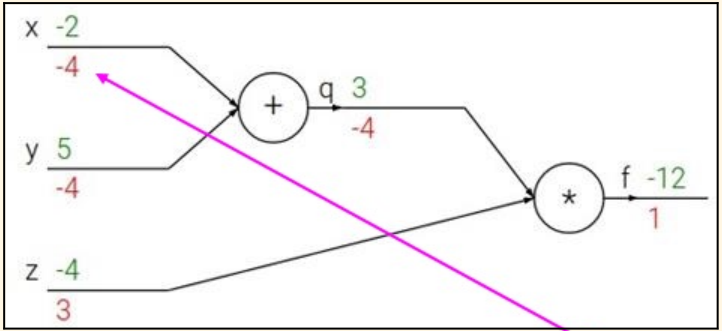
上游梯度 局部梯度

$$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$$

Backpropagation

$$f(x,y,z) = (x+y)z$$

e.g. $x = -2, y = 5, z = -4$



$$\frac{\partial f}{\partial x}$$

$$\boxed{q = x + y}$$ $$\frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

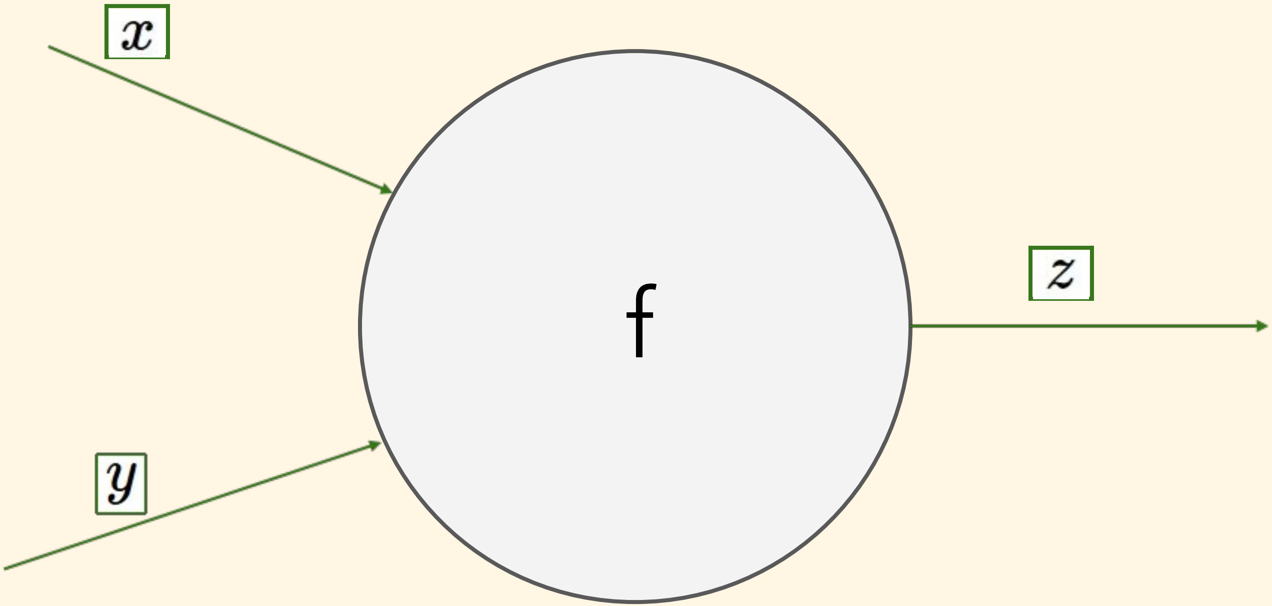
$$\boxed{f = qz}$$ $$\frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

$$\boxed{\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial y}}$$

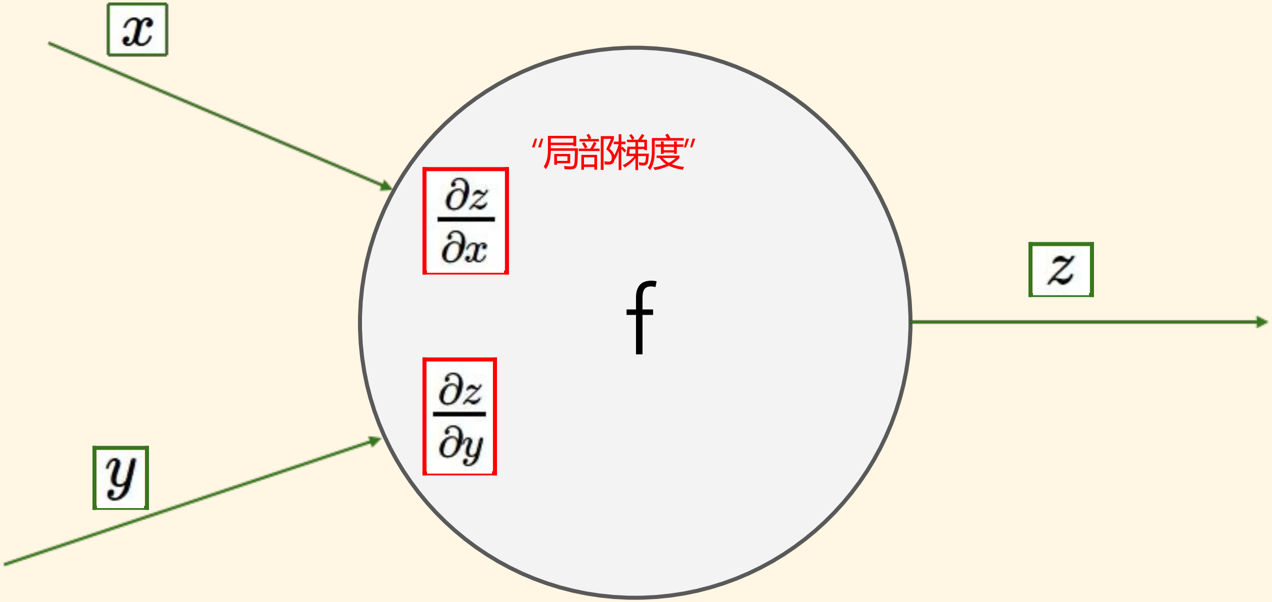
上游梯度 局部梯度

$$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$$

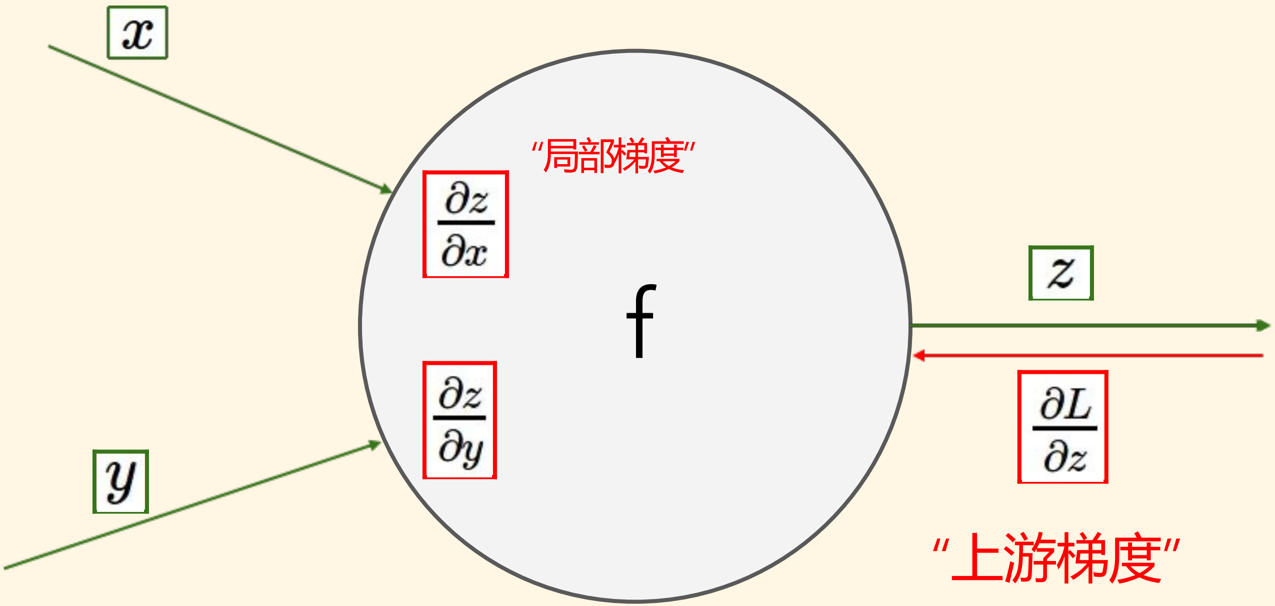
神经元



神经元

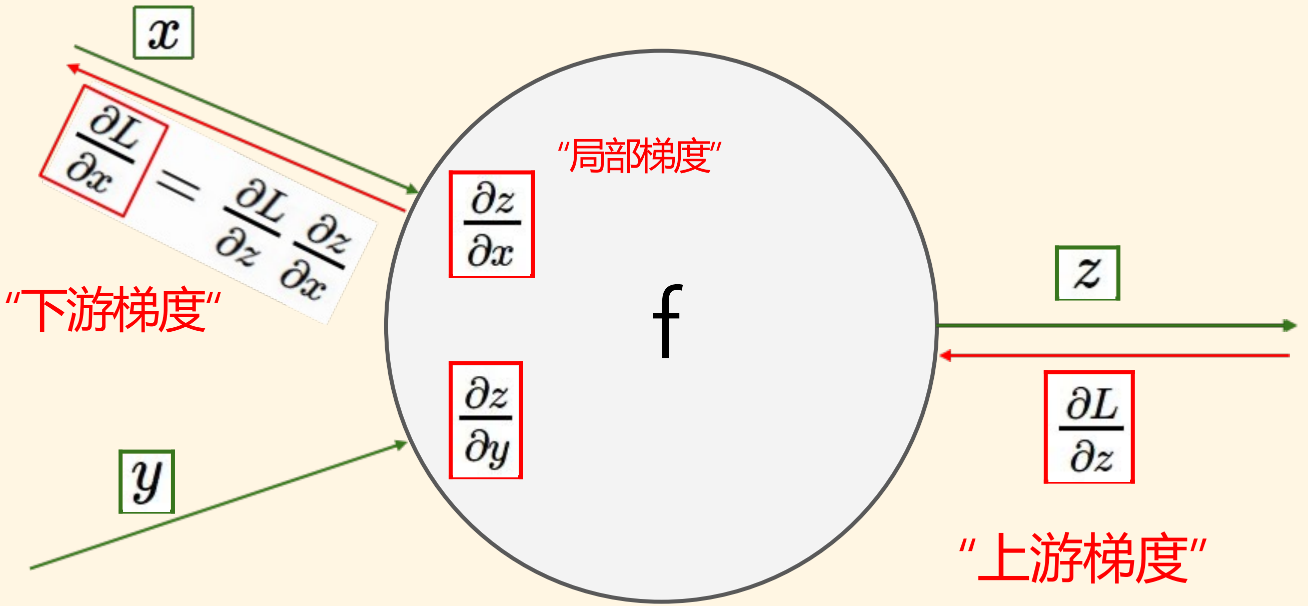


神经元



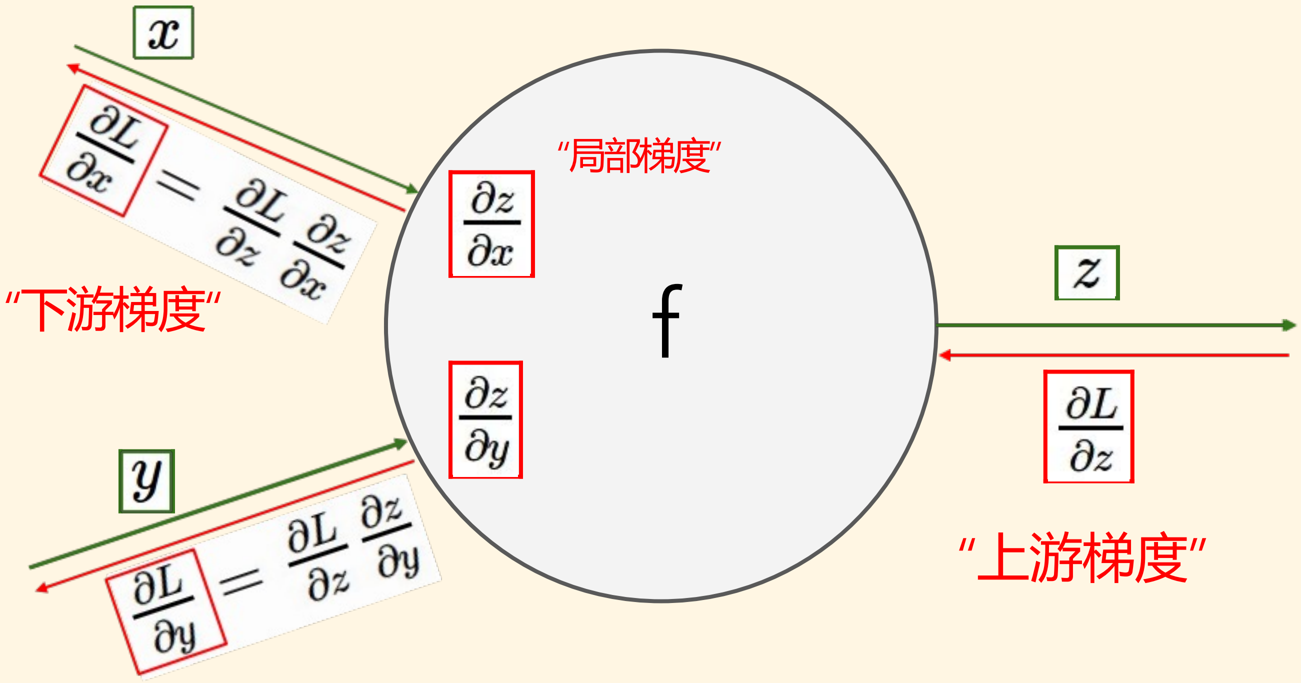
x
"局部梯度"
$\frac{\partial z}{\partial x}$
f
z
$\frac{\partial z}{\partial y}$
$\frac{\partial L}{\partial z}$
y
"上游梯度"

神经元



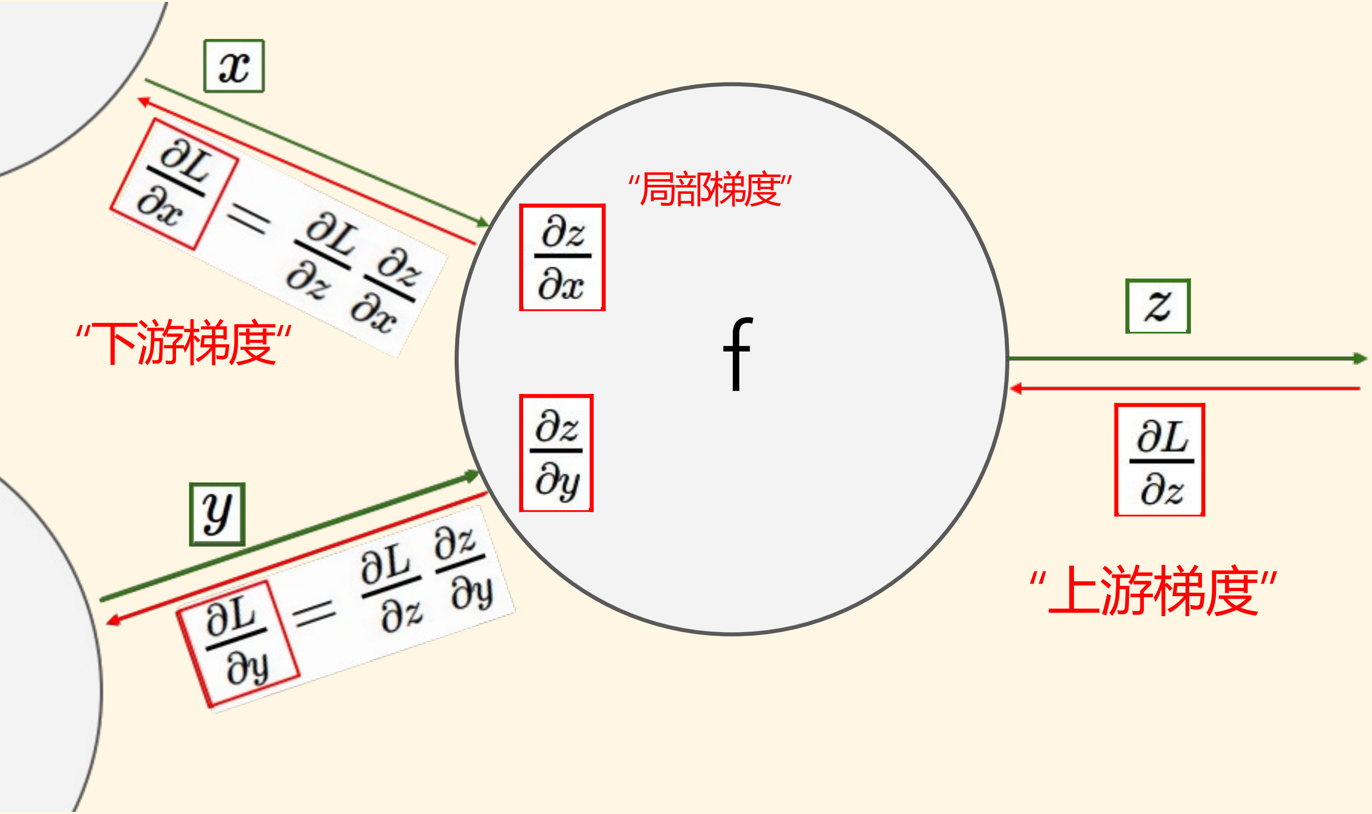
x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
"下游梯度"
"局部梯度"
$\frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial y}$
f
y
z
$\frac{\partial L}{\partial z}$
"上游梯度"

神经元



x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
"局部梯度"
$\frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial y}$
f
z
$\frac{\partial L}{\partial z}$
"下游梯度"
y
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$
"上游梯度"

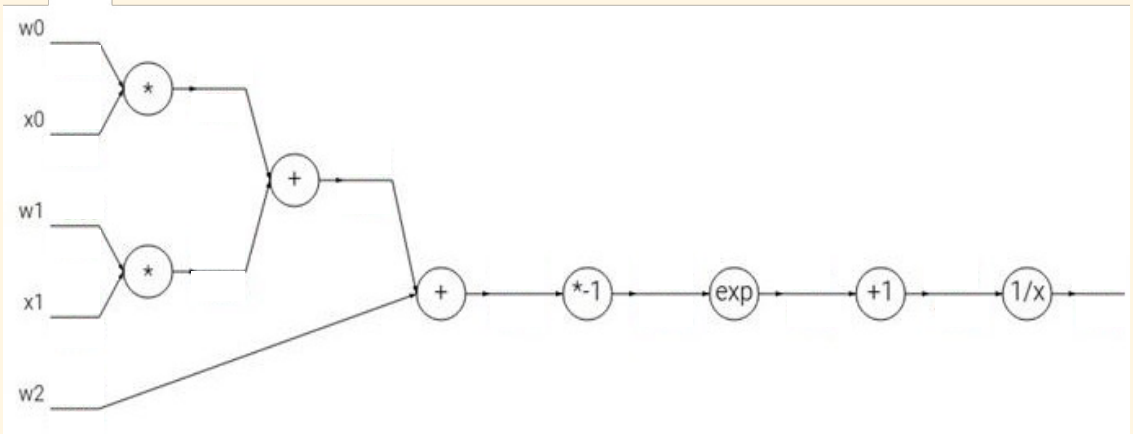
神经元



x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
"下游梯度"
"局部梯度"
$\frac{\partial z}{\partial x}$
f
$\frac{\partial z}{\partial y}$
z
$\frac{\partial L}{\partial z}$
y
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$
"上游梯度"

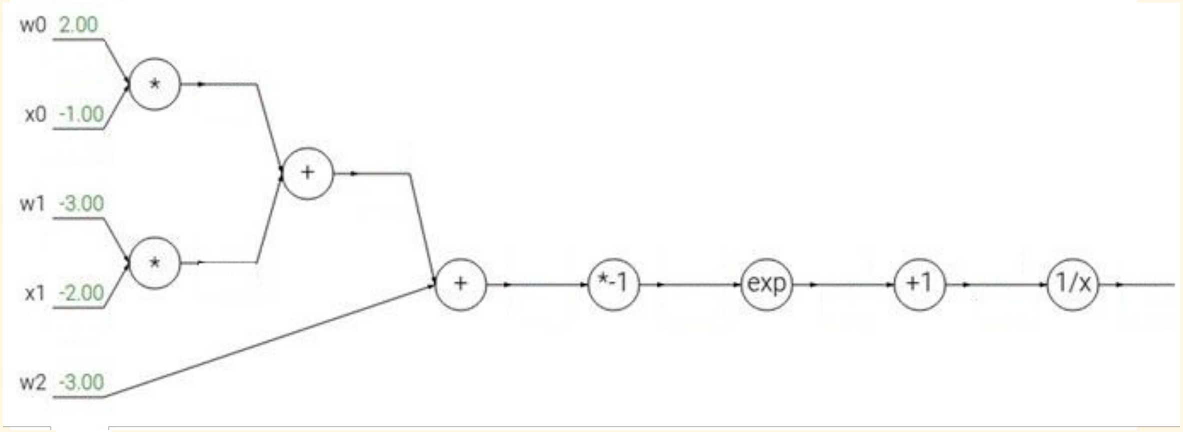
例

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



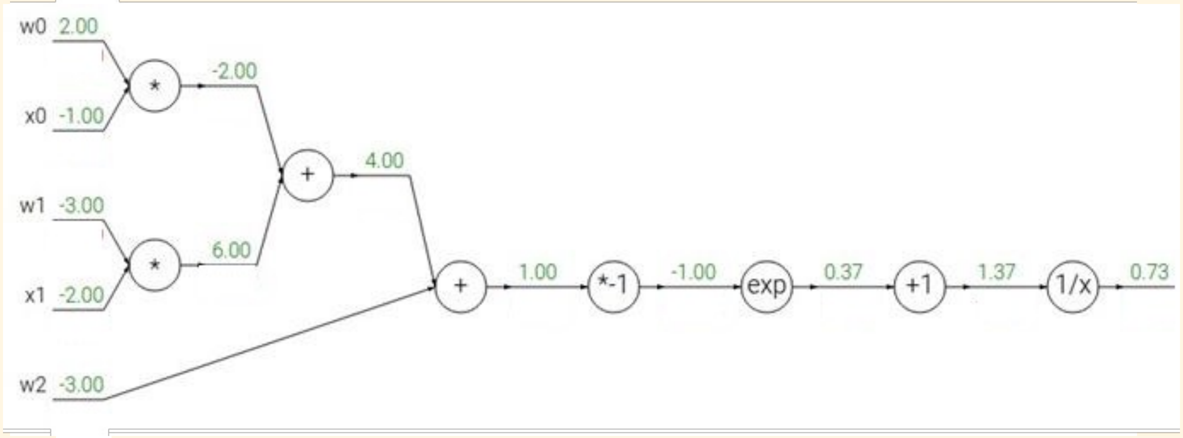
例

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



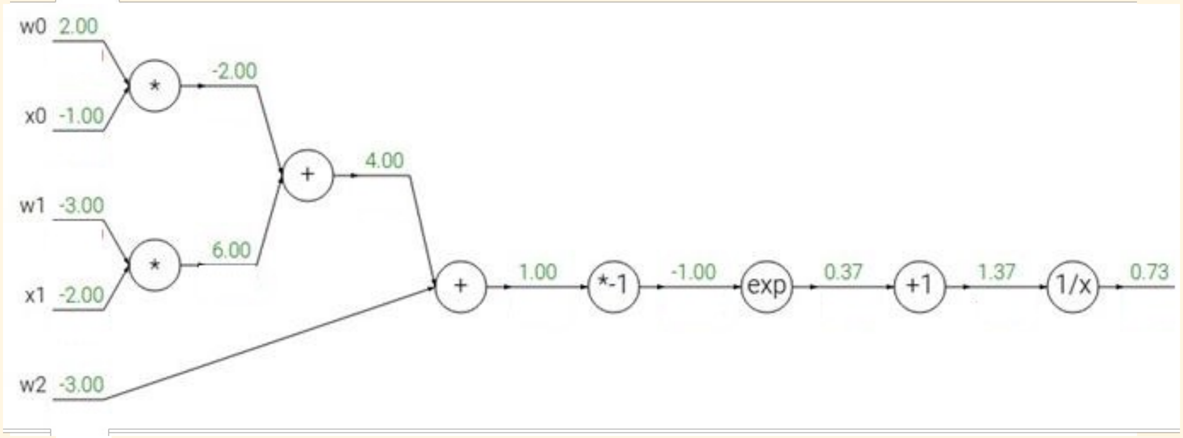
例

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



例

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

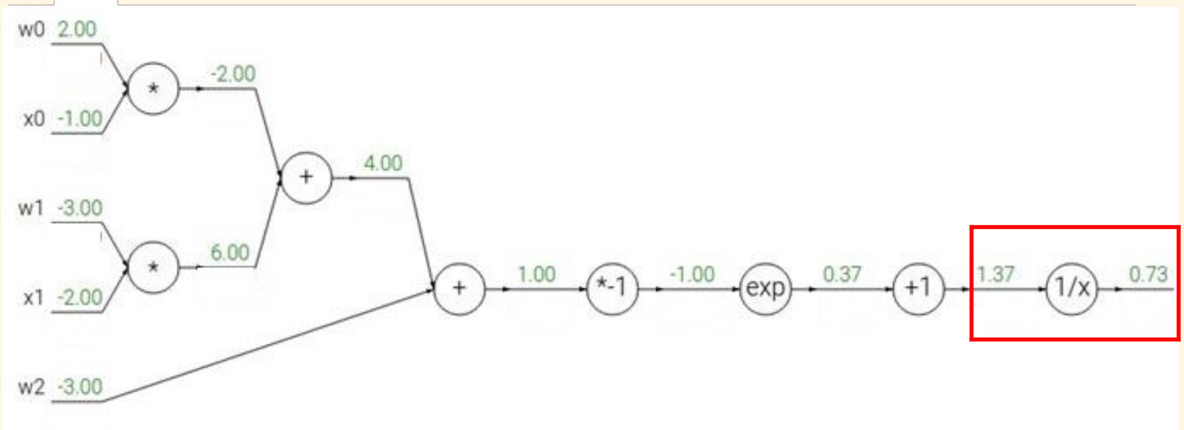
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

例

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

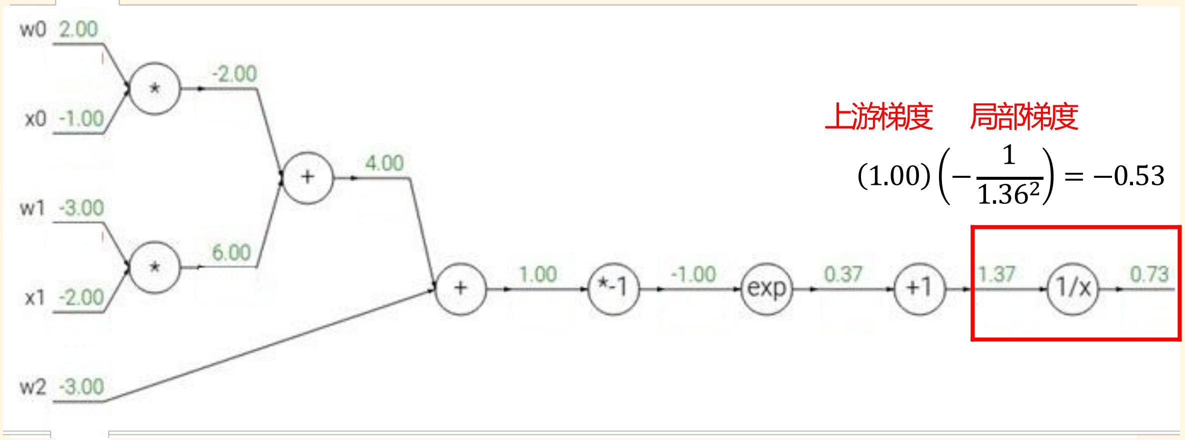
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

例

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

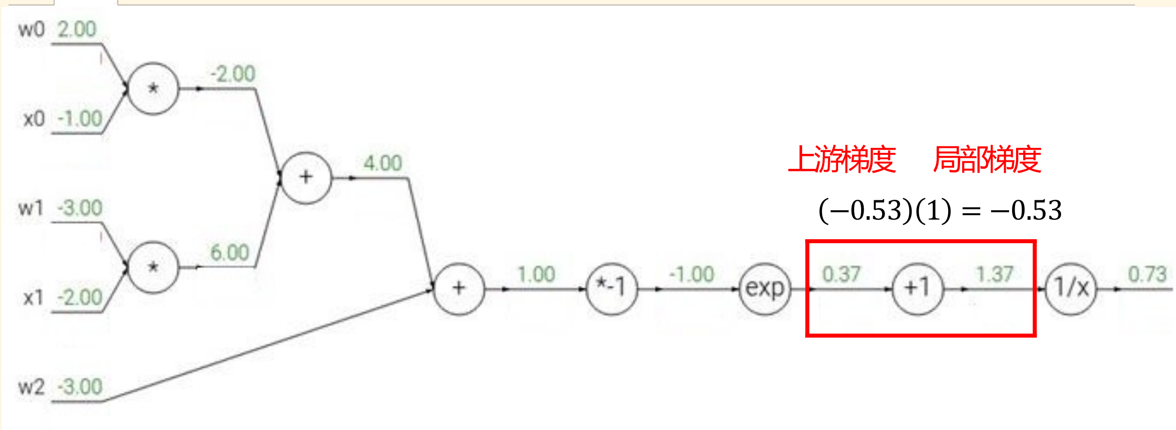
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$\boxed{f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2}$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

例

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

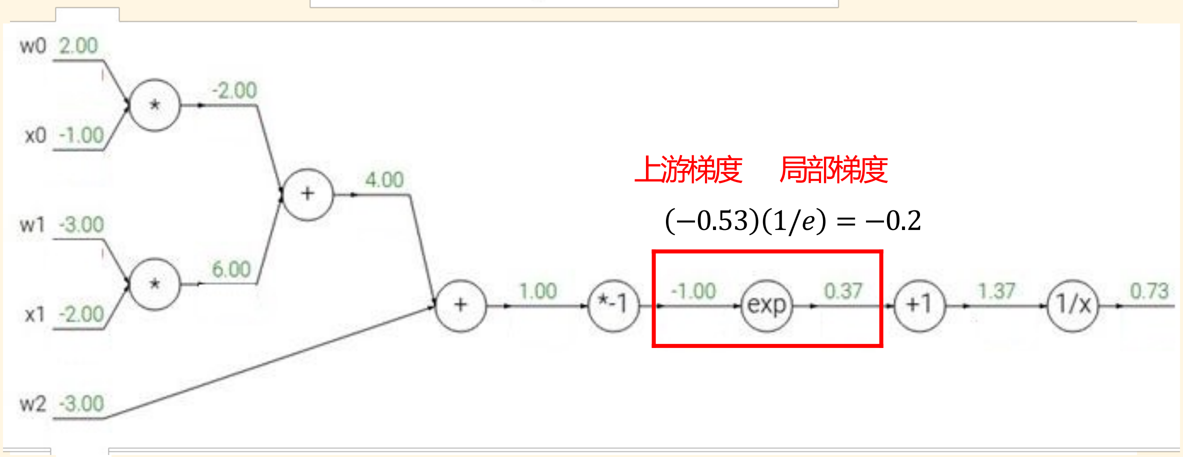
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$\boxed{f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1}$$

例

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



上游梯度　　局部梯度

$$(-0.53)(1/e) = -0.2$$

$$\boxed{f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x}$$

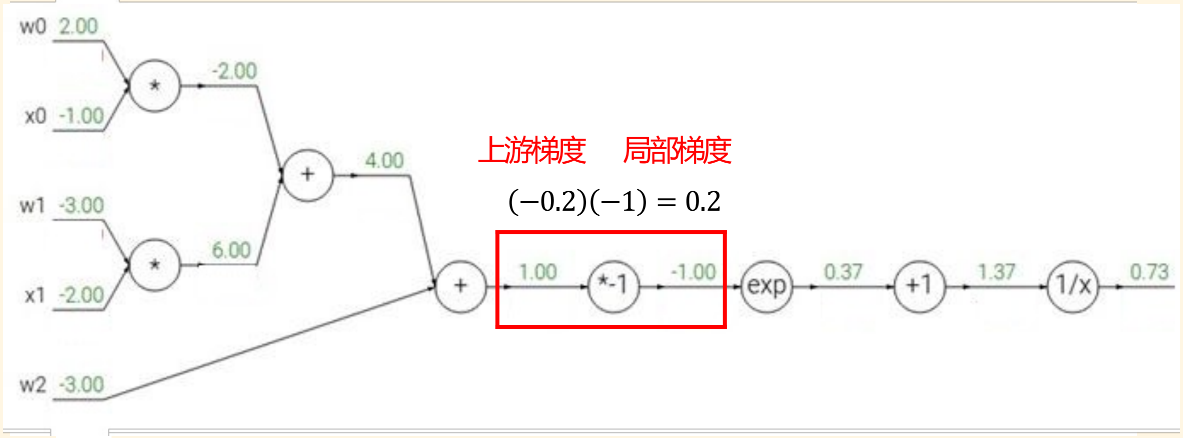
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

例

$$f(w,x)=\frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$f(x)=e^x \quad \rightarrow \quad \frac{df}{dx}=e^x$$

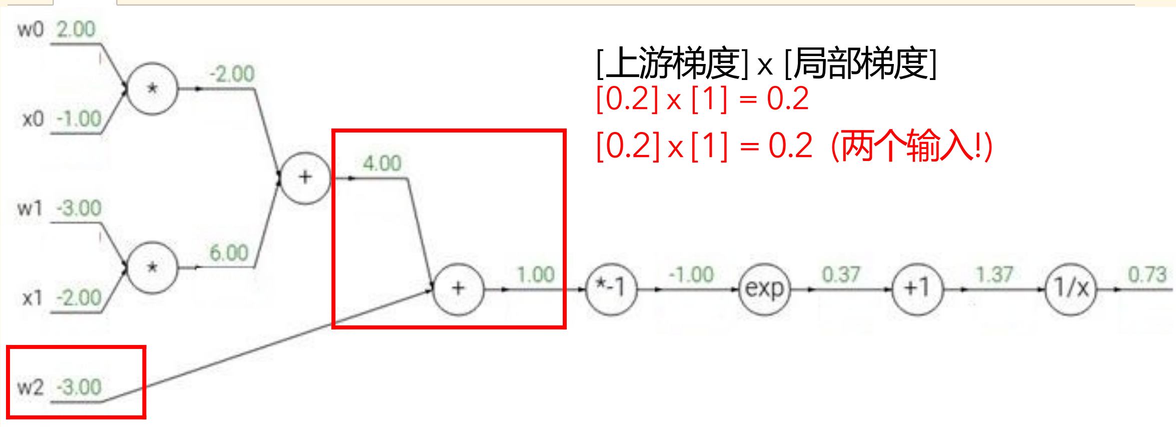
$$\boxed{f_a(x)=ax \quad \rightarrow \quad \frac{df}{dx}=a}$$

$$f(x)=\frac{1}{x} \quad \rightarrow \quad \frac{df}{dx}=-1/x^2$$

$$f_c(x)=c+x \quad \rightarrow \quad \frac{df}{dx}=1$$

例

$$f(w,x)=\frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$f(x)=e^x \quad \rightarrow \quad \frac{df}{dx}=e^x$$

$$f_a(x)=ax \quad \rightarrow \quad \frac{df}{dx}=a$$

$$f(x)=\frac{1}{x} \quad \rightarrow \quad \frac{df}{dx}=-1/x^2$$

$$f_c(x)=c+x \quad \rightarrow \quad \frac{df}{dx}=1$$

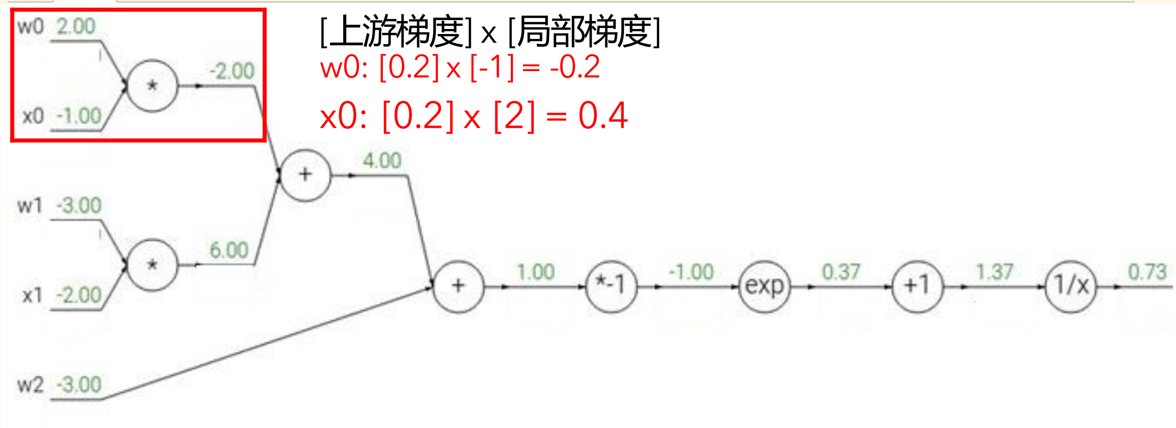
例

$$f(w,x)=\frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

[上游梯度] x [局部梯度]

w0: [0.2] x [-1] = -0.2

x0: [0.2] x [2] = 0.4



$$f(x)=e^x \quad\rightarrow\quad \frac{df}{dx}=e^x$$

$$f_a(x)=ax \quad\rightarrow\quad \frac{df}{dx}=a$$

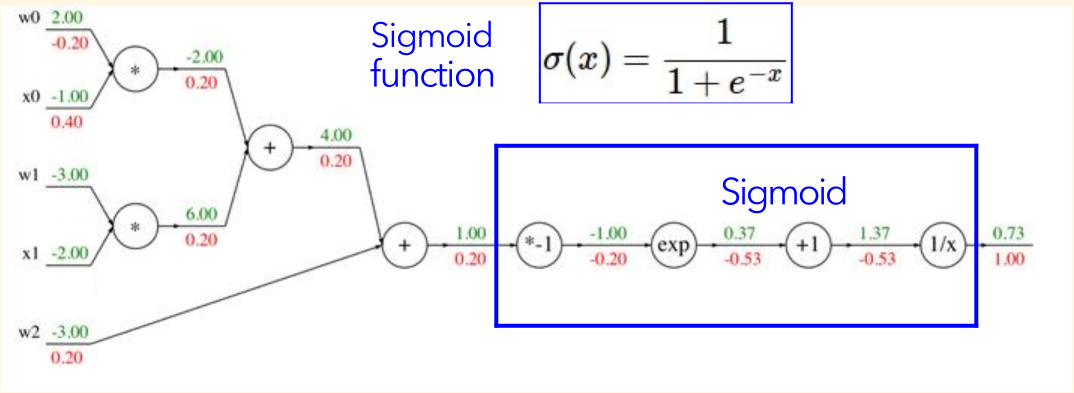
$$f(x)=\frac{1}{x} \quad\rightarrow\quad \frac{df}{dx}=-1/x^2$$

$$f_c(x)=c+x \quad\rightarrow\quad \frac{df}{dx}=1$$

例

$$f(w,x)=\frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

计算图的表示方式并非唯一。选择一种能让每个节点的局部梯度都易于表达的形式即可！



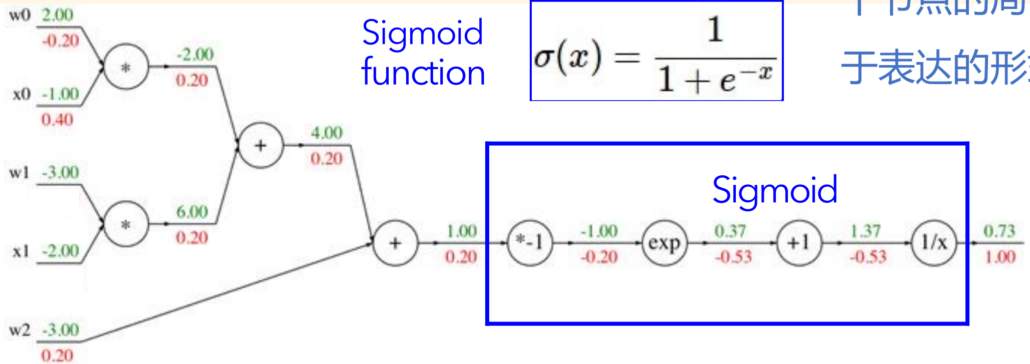
例

$$f(w,x)=\frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

计算图的表示方式并非唯一。选择一种能让每个节点的局部梯度都易于表达的形式即可！

Sigmoid function

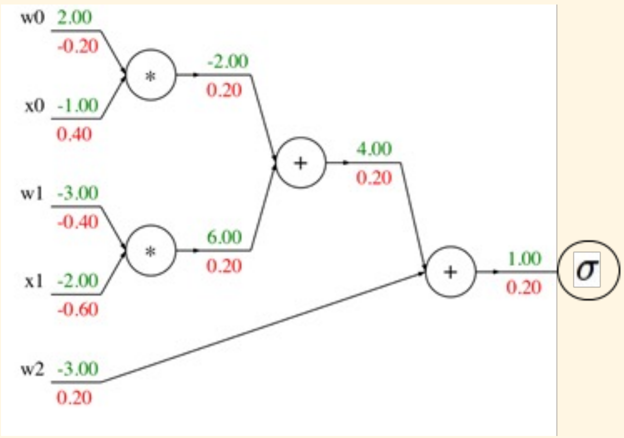
$$\boxed{\sigma(x)=\frac{1}{1+e^{-x}}}$$



Sigmoid local gradient:

$$\frac{d\sigma(x)}{dx}=\frac{e^{-x}}{(1+e^{-x})^2}=\left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right)=(1-\sigma(x))\,\sigma(x)$$

Backpropagation的实现

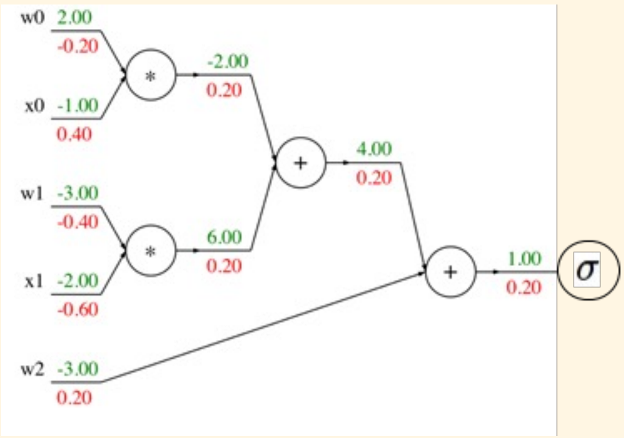


前向计算

反向传播

```
def f(w0, x0, w1, x1, w2):
    s0 = w0 * x0
    s1 = w1 * x1
    s2 = s0 + s1
    s3 = s2 + w2
    L = sigmoid(s3)

    grad_L = 1.0
    grad_s3 = grad_L * (1 - L) * L
    grad_w2 = grad_s3
    grad_s2 = grad_s3
    grad_s0 = grad_s2
    grad_s1 = grad_s2
    grad_w1 = grad_s1 * x1
    grad_x1 = grad_s1 * w1
    grad_w0 = grad_s0 * x0
    grad_x0 = grad_s0 * w0
```

Backpropagation的实现



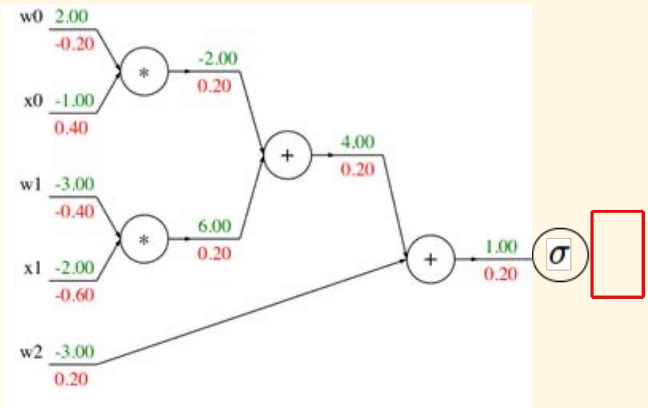
前向计算

```
def f(w0, x0, w1, x1, w2):
    s0 = w0 * x0
    s1 = w1 * x1
    s2 = s0 + s1
    s3 = s2 + w2
    L = sigmoid(s3)
```

反向传播

```
    grad_L = 1.0
    grad_s3 = grad_L * (1 - L) * L
    grad_w2 = grad_s3
    grad_s2 = grad_s3
    grad_s0 = grad_s2
    grad_s1 = grad_s2
    grad_w1 = grad_s1 * x1
    grad_x1 = grad_s1 * w1
    grad_w0 = grad_s0 * x0
    grad_x0 = grad_s0 * w0
```

Backpropagation的实现

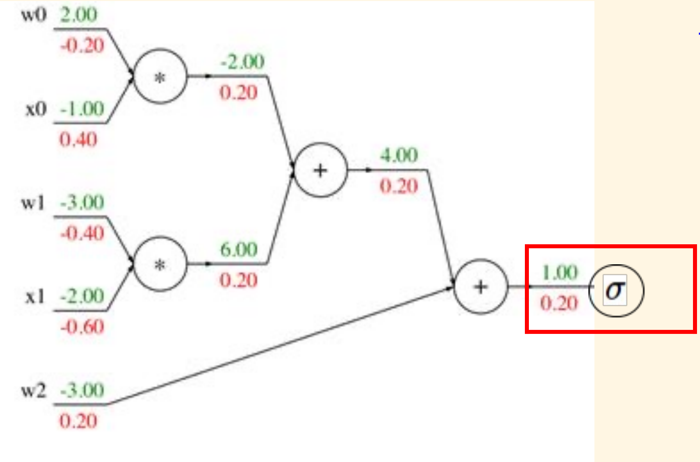


前向计算

反向传播

```
def f(w0, x0, w1, x1, w2):
  s0 = w0 * x0
  s1 = w1 * x1
  s2 = s0 + s1
  s3 = s2 + w2
  L = sigmoid(s3)

  grad_L = 1.0
  grad_s3 = grad_L * (1 - L) * L
  grad_w2 = grad_s3
  grad_s2 = grad_s3
  grad_s0 = grad_s2
  grad_s1 = grad_s2
  grad_w1 = grad_s1 * x1
  grad_x1 = grad_s1 * w1
  grad_w0 = grad_s0 * x0
  grad_x0 = grad_s0 * w0
```

Backpropagation的实现

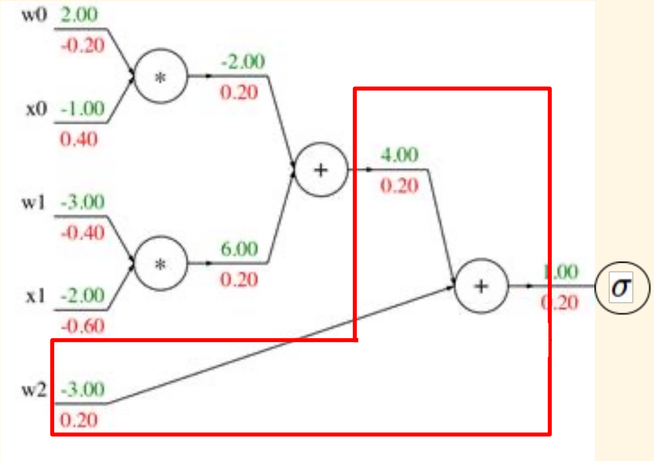


前向计算

反向传播

```
def f(w0, x0, w1, x1, w2):
    s0 = w0 * x0
    s1 = w1 * x1
    s2 = s0 + s1
    s3 = s2 + w2
    L = sigmoid(s3)

    grad_L = 1.0
    grad_s3 = grad_L * (1 - L) * L
    grad_w2 = grad_s3
    grad_s2 = grad_s3
    grad_s0 = grad_s2
    grad_s1 = grad_s2
    grad_w1 = grad_s1 * x1
    grad_x1 = grad_s1 * w1
    grad_w0 = grad_s0 * x0
    grad_x0 = grad_s0 * w0
```

Backpropagation的实现

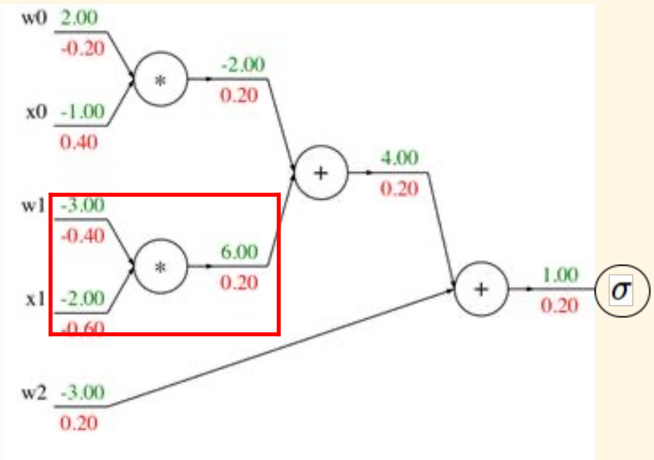


前向计算

反向传播

```
def f(w0, x0, w1, x1, w2):
    s0 = w0 * x0
    s1 = w1 * x1
    s2 = s0 + s1
    s3 = s2 + w2
    L = sigmoid(s3)

    grad_L = 1.0
    grad_s3 = grad_L * (1 - L) * L
    grad_w2 = grad_s3
    grad_s2 = grad_s3
    grad_s0 = grad_s2
    grad_s1 = grad_s2
    grad_w1 = grad_s1 * x1
    grad_x1 = grad_s1 * w1
    grad_w0 = grad_s0 * x0
    grad_x0 = grad_s0 * w0
```

Backpropagation的实现

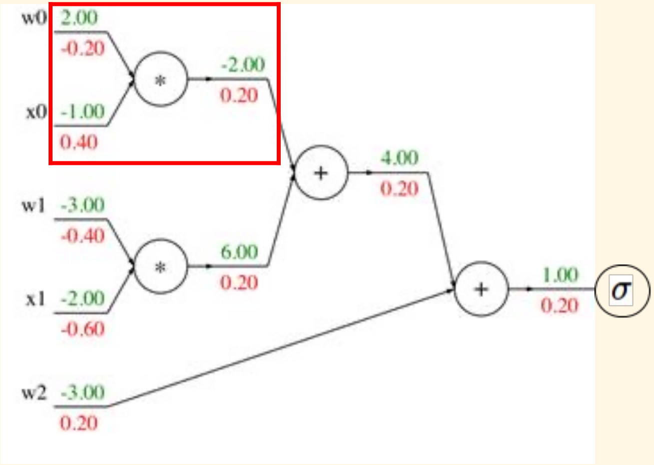


前向计算

反向传播

```python
def f(w0, x0, w1, x1, w2):
    s0 = w0 * x0
    s1 = w1 * x1
    s2 = s0 + s1
    s3 = s2 + w2
    L = sigmoid(s3)

    grad_L = 1.0
    grad_s3 = grad_L * (1 - L) * L
    grad_w2 = grad_s3
    grad_s2 = grad_s3
    grad_s0 = grad_s2
    grad_s1 = grad_s2
    grad_w1 = grad_s1 * x1
    grad_x1 = grad_s1 * w1
    grad_w0 = grad_s0 * x0
    grad_x0 = grad_s0 * w0
```

Backpropagation的实现



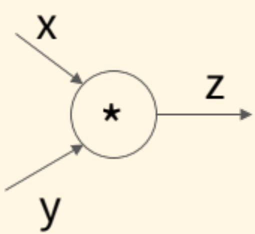
前向计算

反向传播

```
def f(w0, x0, w1, x1, w2):
  s0 = w0 * x0
  s1 = w1 * x1
  s2 = s0 + s1
  s3 = s2 + w2
  L = sigmoid(s3)

  grad_L = 1.0
  grad_s3 = grad_L * (1 - L) * L
  grad_w2 = grad_s3
  grad_s2 = grad_s3
  grad_s0 = grad_s2
  grad_s1 = grad_s2
  grad_w1 = grad_s1 * x1
  grad_x1 = grad_s1 * w1
  grad_w0 = grad_s0 * x0
  grad_x0 = grad_s0 * w0
```

模块化实现

Gate / Node / Function object: Actual PyTorch code



(x,y,z are scalars)

```
class Multiply(torch.autograd.Function):
  @staticmethod
  def forward(ctx, x, y):
    ctx.save_for_backward(x, y)
    z = x * y
    return z
  @staticmethod
  def backward(ctx, grad_z):
    x, y = ctx.saved_tensors
    grad_x = y * grad_z   # dz/dx * dL/dz
    grad_y = x * grad_z   # dz/dy * dL/dz
    return grad_x, grad_y
```

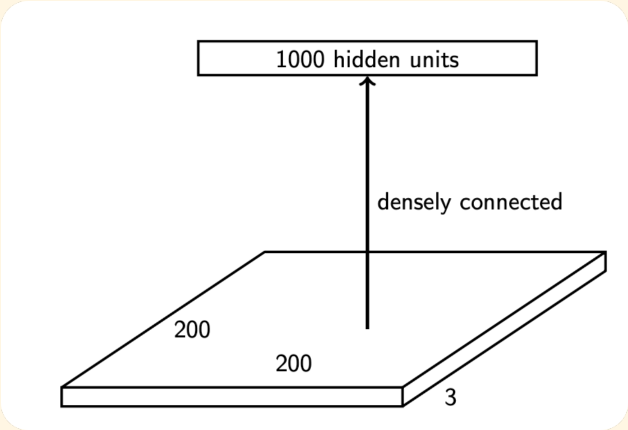
Need to cache some values for use in backward

Upstream gradient

Multiply upstream and local gradients

视觉

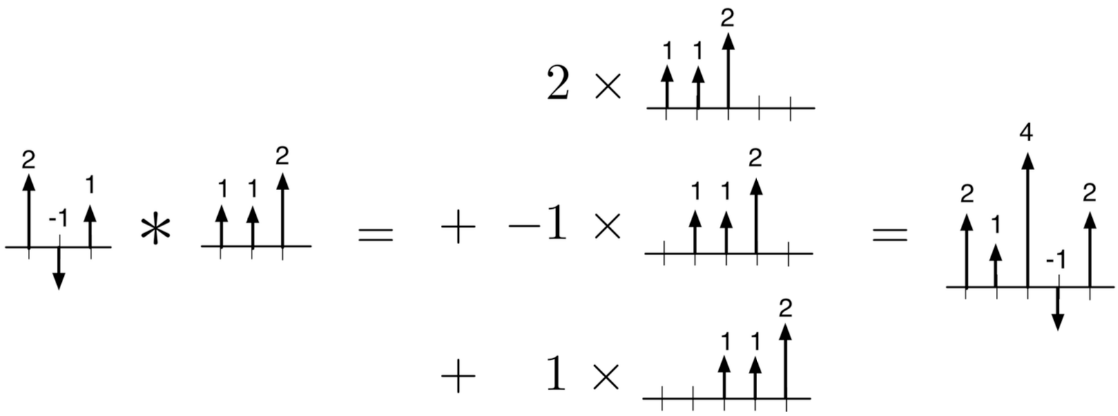
- 假设直接把图像输入到全连接网络中
- 第一层的参数：200*200*3*1000



- 如果图像整体向右移动一个像素

1-D卷积

$$(a * b)_t = \sum_{\tau} a_\tau b_{t-\tau}$$



2-D卷积

$$(A * B)_{ij} = \sum_{s} \sum_{t} A_{st} B_{i-s,j-t}$$

$$\begin{array}{|c|c|c|}\hline 1 & 3 & 1 \\ \hline 0 & -1 & 1 \\ \hline 2 & 2 & -1 \\ \hline \end{array} * \begin{array}{|c|c|}\hline 1 & 2 \\ \hline 0 & -1 \\ \hline \end{array} = 1 \times \begin{array}{|c|c|c|c|}\hline 1 & 3 & 1 & \\ \hline 0 & -1 & 1 & \\ \hline 2 & 2 & -1 & \\ \hline & & & \\ \hline \end{array} + 2 \times \begin{array}{|c|c|c|c|}\hline & 1 & 3 & 1 \\ \hline & 0 & -1 & 1 \\ \hline & 2 & 2 & -1 \\ \hline & & & \\ \hline \end{array} + -1 \times \begin{array}{|c|c|c|c|}\hline & & & \\ \hline & 1 & 3 & 1 \\ \hline & 0 & -1 & 1 \\ \hline & 2 & 2 & -1 \\ \hline \end{array} = \begin{array}{|c|c|c|c|}\hline 1 & 5 & 7 & 2 \\ \hline 0 & -2 & -4 & 1 \\ \hline 2 & 6 & 4 & -3 \\ \hline 0 & -2 & -2 & 1 \\ \hline \end{array}$$

卷积神经网络

$$* \begin{bmatrix} 0 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

$$* \begin{bmatrix} 0 & -1 & 0 \\ -1 & 8 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

卷积神经网络

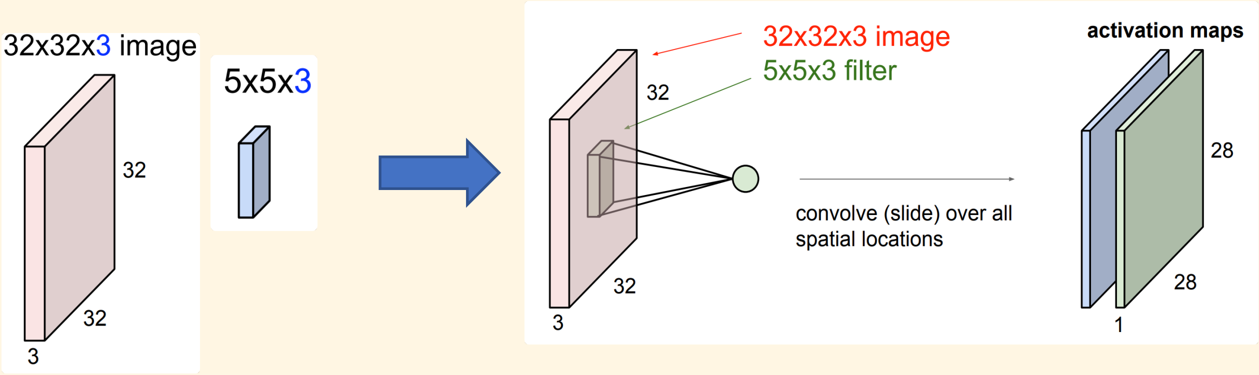
*

0	-1	0
-1	4	-1
0	-1	0

*

1	0	-1
2	0	-2
1	0	-1

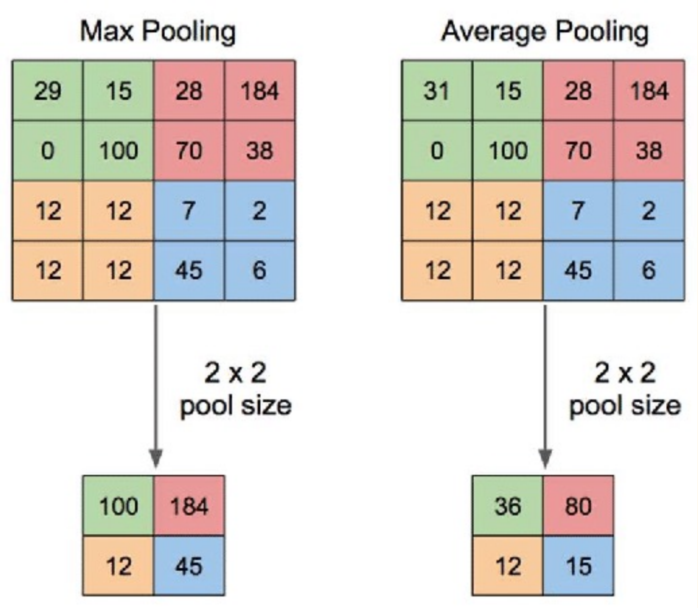
卷积神经网络



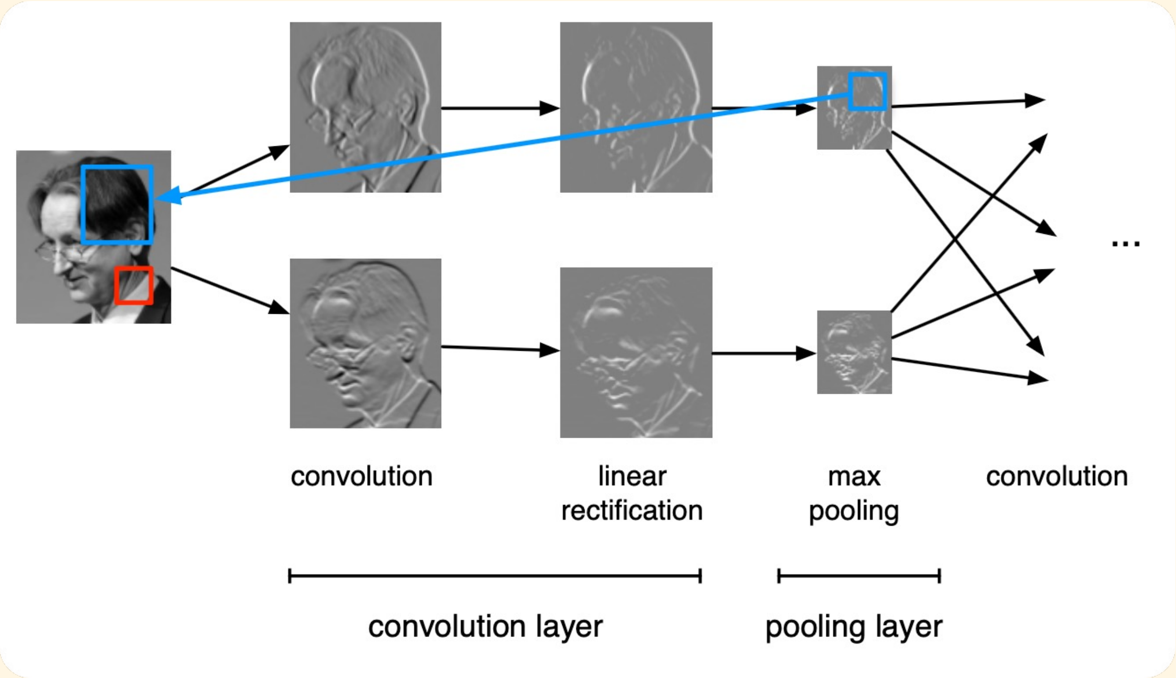
卷积层是线性操作，因此在卷积层之后都要有非线性激活层，比如ReLU

池化层

$$y_i = \max_j z_j$$



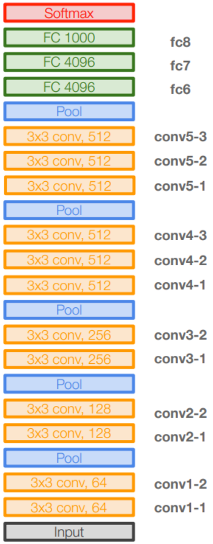
卷积神经网络



...
convolution
linear
rectification
max
pooling
convolution
convolution layer
pooling layer

VGG模型

INPUT: [224x224x3] memory: 224*224*3=150K params: 0 (not counting biases)
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

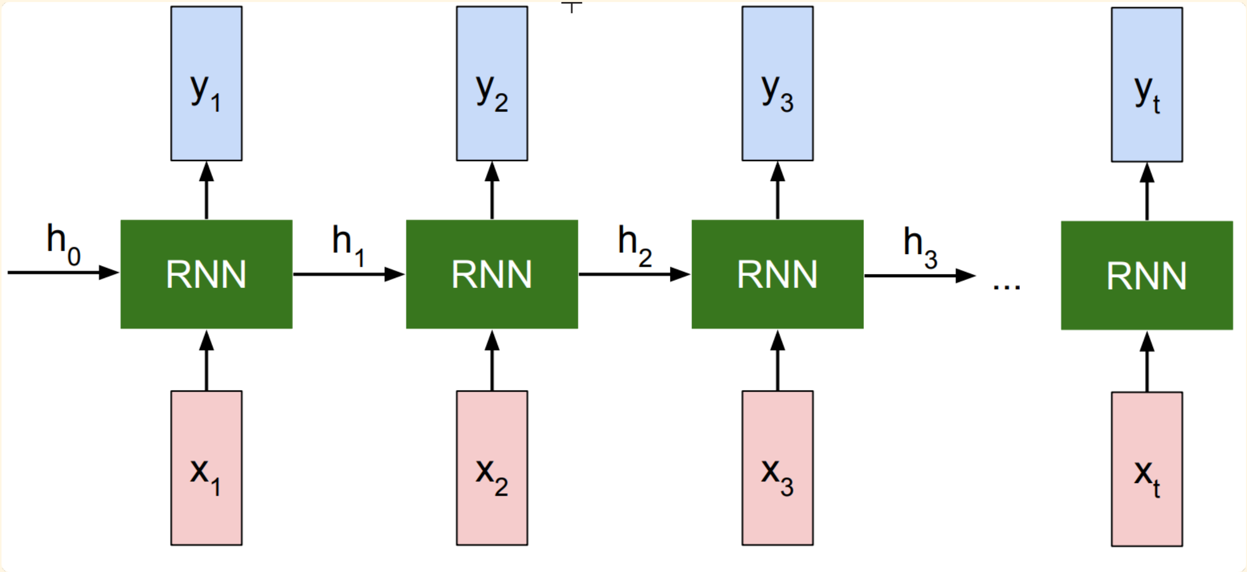


VGG16

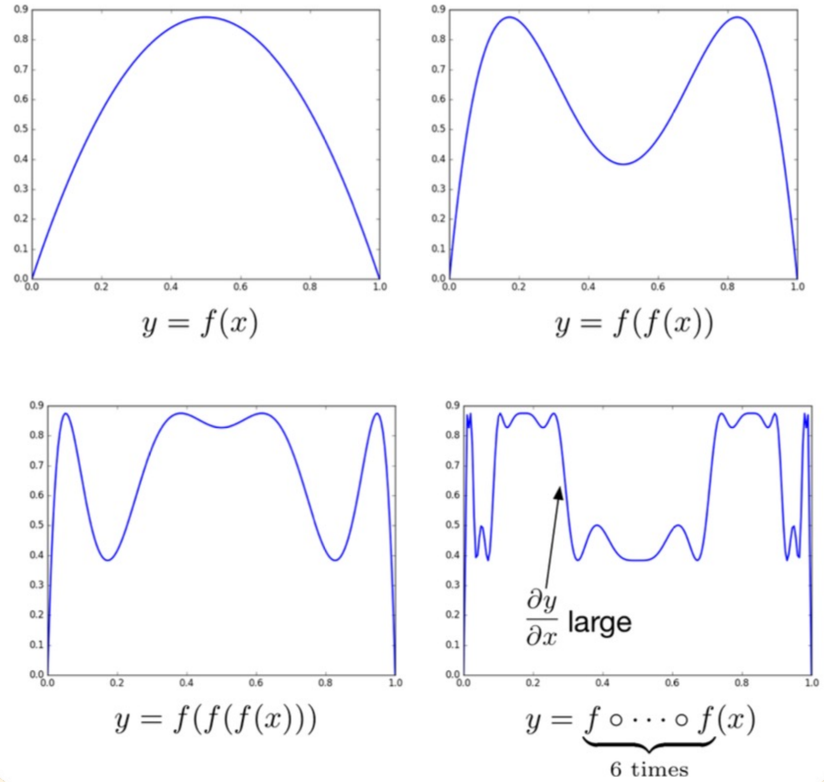
Common names

TOTAL memory: 24M * 4 bytes ~= 96MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters

循环神经网络



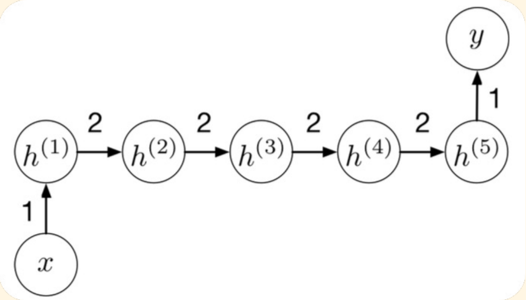
梯度消失与梯度爆炸

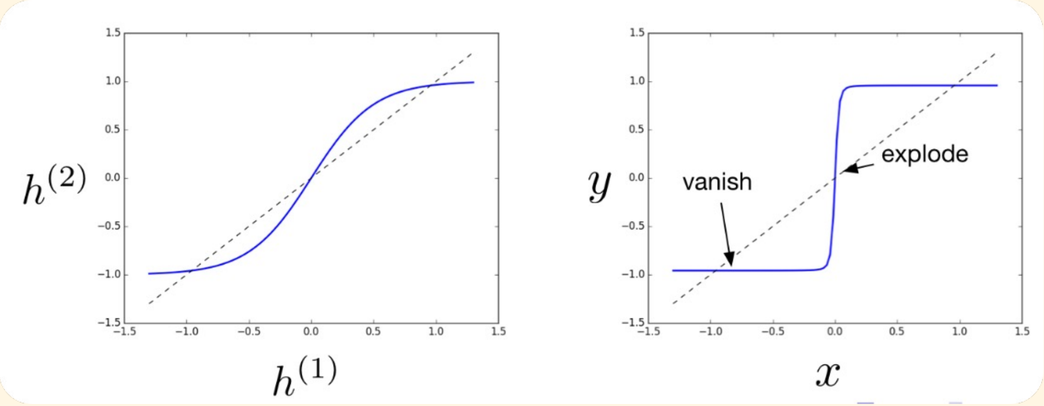


$y = f(x)$
$y = f(f(x))$
$y = f(f(f(x)))$
$\frac{\partial y}{\partial x}$ large
$y = \underbrace{f \circ \cdots \circ f}_{6\ \text{times}}(x)$

梯度消失与梯度爆炸

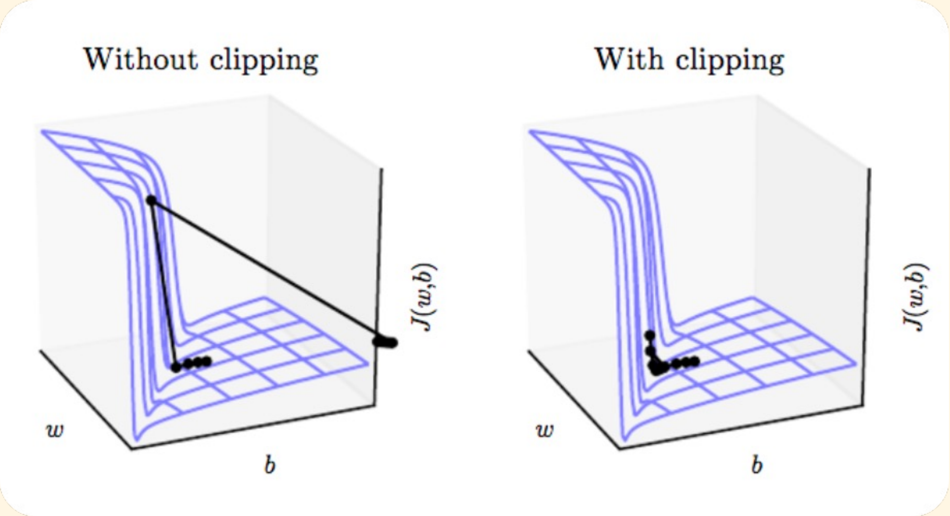
考虑一个多层的循环神经网络



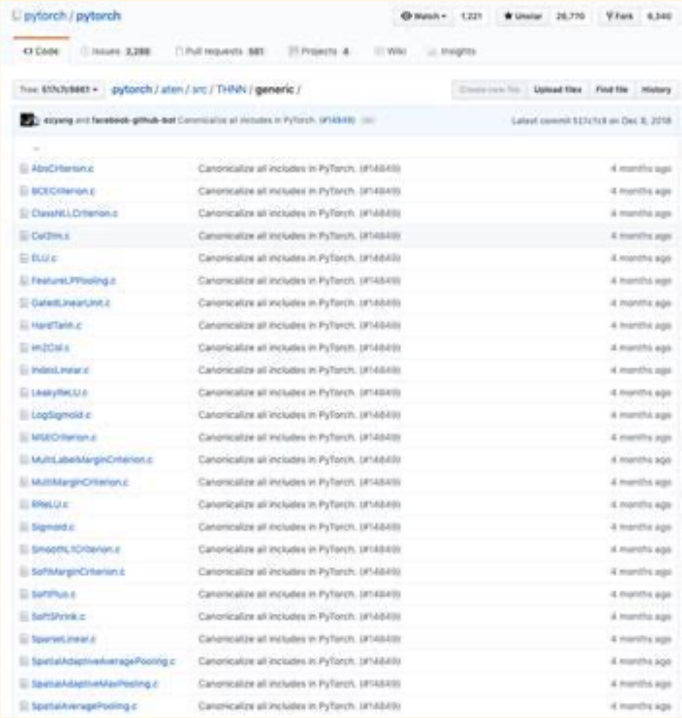


梯度裁剪

$$\text{if } \left\| g \right\| > \eta, \qquad g = \frac{\eta g}{||g||}$$



PyTorch算子



PyTorch Sigmoid层

```c
#ifndef TH_GENERIC_FILE
#define TH_GENERIC_FILE "THNN/generic/Sigmoid.c"
#else

void THNN_(Sigmoid_updateOutput)(
          THNNState *state,
          THTensor *input,
          THTensor *output)
{
  THTensor_(sigmoid)(output, input);
}

void THNN_(Sigmoid_updateGradInput)(
          THNNState *state,
          THTensor *gradOutput,
          THTensor *gradInput,
          THTensor *output)
{
  THNN_CHECK_NELEMENT(output, gradOutput);
  THTensor_(resizeAs)(gradInput, output);
  TH_TENSOR_APPLY3(scalar_t, gradInput, scalar_t, gradOutput, scalar_t, output,
    scalar_t z = *output_data;
    *gradInput_data = *gradOutput_data * (1. - z) * z;
  );
}

#endif
```

Forward

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

PyTorch Sigmoid层

```c
#ifndef TH_GENERIC_FILE
#define TH_GENERIC_FILE "THNN/generic/Sigmoid.c"
#else

void THNN_(Sigmoid_updateOutput)(
          THNNState *state,
          THTensor *input,
          THTensor *output)
{
  THTensor_(sigmoid)(output, input);
}

void THNN_(Sigmoid_updateGradInput)(
          THNNState *state,
          THTensor *gradOutput,
          THTensor *gradInput,
          THTensor *output)
{
  THNN_CHECK_NELEMENT(output, gradOutput);
  THTensor_(resizeAs)(gradInput, output);
  TH_TENSOR_APPLY3(scalar_t, gradInput, scalar_t, gradOutput, scalar_t, output,
    scalar_t z = *output_data;
    *gradInput_data = *gradOutput_data * (1. - z) * z;
  );
}

#endif
```

Forward

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

```cpp
static void sigmoid_kernel(TensorIterator& iter) {
  AT_DISPATCH_FLOATING_TYPES(iter.dtype(), "sigmoid_cpu", [&]() {
    unary_kernel_vec(
        iter,
        [=](scalar_t a) -> scalar_t { return (1 / (1 + std::exp((-a)))); },
        [=](Vec256<scalar_t> a) {
          a = Vec256<scalar_t>((scalar_t)(0)) - a;
          a = a.exp();
          a = Vec256<scalar_t>((scalar_t)(1)) + a;
          a = a.reciprocal();
          return a;
        });
  });
}
```

```cpp
return (1 / (1 + std::exp((-a))));
```

PyTorch Sigmoid层

```c
#ifndef TH_GENERIC_FILE
#define TH_GENERIC_FILE "THNN/generic/Sigmoid.c"
#else

void THNN_(Sigmoid_updateOutput)(
          THNNState *state,
          THTensor *input,
          THTensor *output)
{
  THTensor_(sigmoid)(output, input);
}

void THNN_(Sigmoid_updateGradInput)(
          THNNState *state,
          THTensor *gradOutput,
          THTensor *gradInput,
          THTensor *output)
{
  THNN_CHECK_NELEMENT(output, gradOutput);
  THTensor_(resizeAs)(gradInput, output);
  TH_TENSOR_APPLY3(scalar_t, gradInput, scalar_t, gradOutput, scalar_t, output,
    scalar_t z = *output_data;
    *gradInput_data = *gradOutput_data * (1. - z) * z;
  );
}

#endif
```

Forward

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

```cpp
static void sigmoid_kernel(TensorIterator& iter) {
  AT_DISPATCH_FLOATING_TYPES(iter.dtype(), "sigmoid_cpu", [&]() {
    unary_kernel_vec(
        iter,
        [=](scalar_t a) -> scalar_t { return (1 / (1 + std::exp((-a)))); },
        [=](Vec256<scalar_t> a) {
          a = Vec256<scalar_t>((scalar_t)(0)) - a;
          a = a.exp();
          a = Vec256<scalar_t>((scalar_t)(1)) + a;
          a = a.reciprocal();
          return a;
        });
  });
}
```

Backward

$$(1-\sigma(x))\,\sigma(x)$$

总结

- 神经网络由线性函数和非线性激活函数堆叠而成，其表征能力远超线性分类器。
- 反向传播是沿计算图递归应用链式法则，以计算所有输入、参数和中间变量的梯度。
- 实现过程会维护一个图结构，图中的节点实现前向传播（forward ()）/ 反向传播（backward ()）编程接口。
- 前向传播（forward）：计算运算结果，并将梯度计算所需的所有中间变量存储在内存中。
- 反向传播（backward）：应用链式法则，计算损失函数对输入的梯度。

作业

1.给定单样本输入向量 $x = \begin{bmatrix} 0.5 \\ -1.5 \end{bmatrix}$。两层前馈网络结构：

- 隐藏层（2 个神经元），权重矩阵 $W^{(1)} \in \mathbb{R}^{2\times 2}$、偏置 $b^{(1)} \in \mathbb{R}^2$，激活函数为 sigmoid：

$$W^{(1)} = \begin{bmatrix} 0.4 & -0.2 \\ -0.1 & 0.3 \end{bmatrix}, b^{(1)} = \begin{bmatrix} 0.1 \\ -0.3 \end{bmatrix}$$

隐含激活 $h = \sigma\left(z^{(1)}\right)$，其中 $z^{(1)} = W^{(1)}x + b^{(1)}$。

- 输出层为单输出（线性），权重 $W^{(2)} = [0.2 \quad -0.5]$，偏置 $b^{(2)} = 0.05$，输出 $y = W^{(2)}h + b^{(2)}$。

求：$z^{(1)}$，h，y。

2. 同样的网络与输入，令目标值 $t = 0.3$。使用均方误差损失（带 1/2 因子）：

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

求损失对参数的梯度：$\frac{\partial L}{\partial W^{(2)}}$，$\frac{\partial L}{\partial b^{(2)}}$，$\frac{\partial L}{\partial W^{(1)}}$，$\frac{\partial L}{\partial b^{(1)}}$。

（提示：sigmoid 导数 $\sigma'(u) = \sigma(u)\left(1 - \sigma(u)\right)$）

谢谢！