计算机视觉
Computer Vision

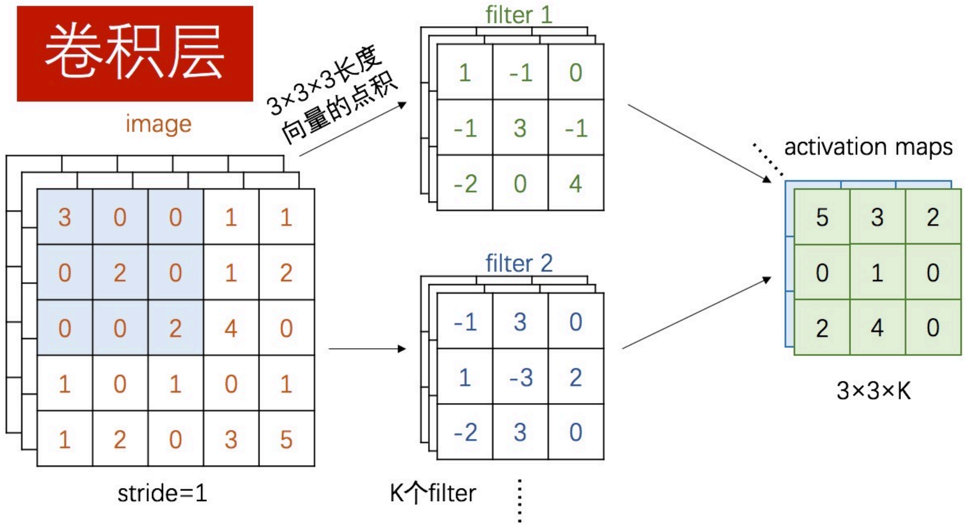
Lecture 6: 神经网络的训练1

DaSE
Data Science
& Engineering

EAST CHINA NORMAL UNIVERSITY
華東師範大學

L05：卷积神经网络

卷积层



image

3	0	0	1	1
0	2	0	1	2
0	0	2	4	0
1	0	1	0	1
1	2	0	3	5

filter 1

1	-1	0
-1	3	-1
-2	0	4

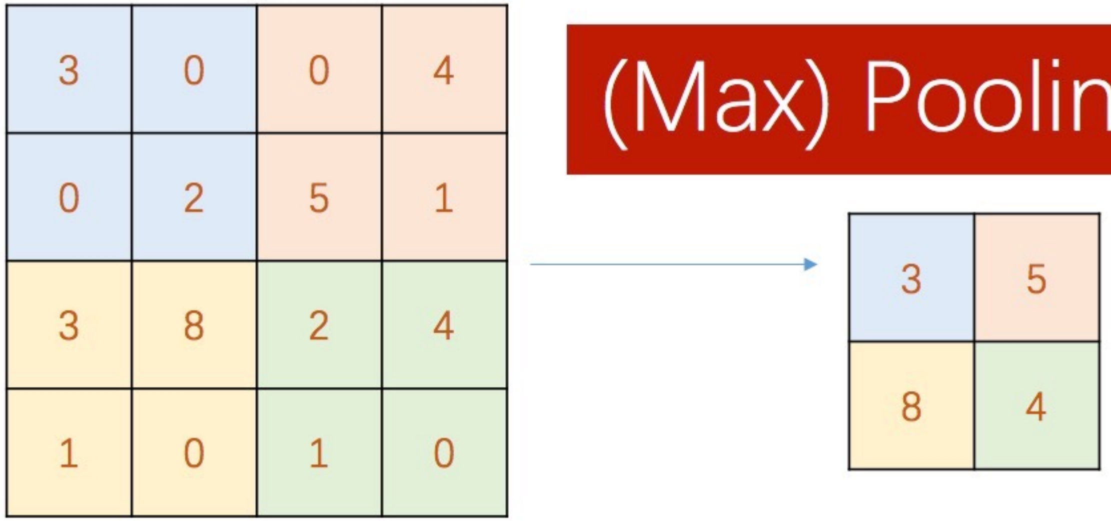
filter 2

-1	3	0
1	-3	2
-2	3	0

activation maps

5	3	2
0	1	0
2	4	0

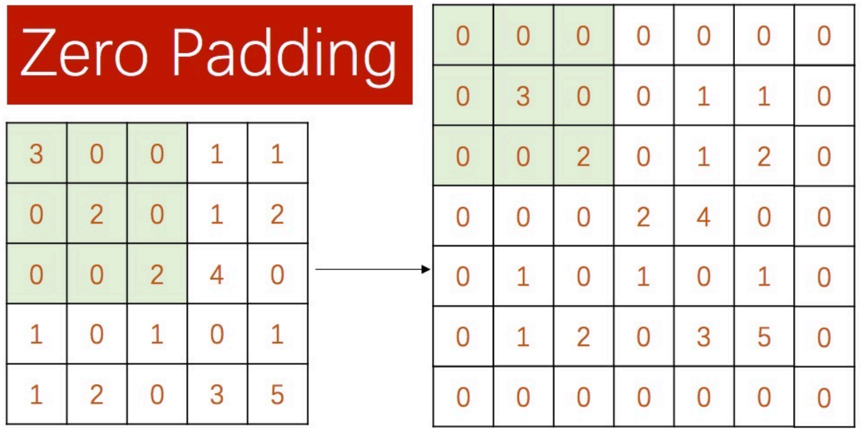
(Max) Pooling



3	0	0	4
0	2	5	1
3	8	2	4
1	0	1	0

3	5
8	4

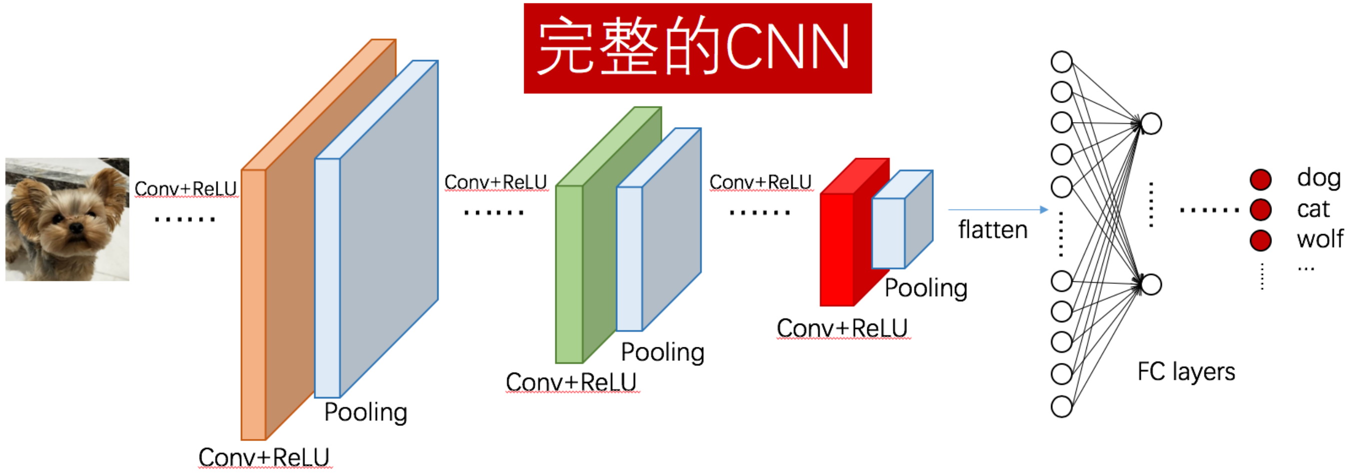
Zero Padding



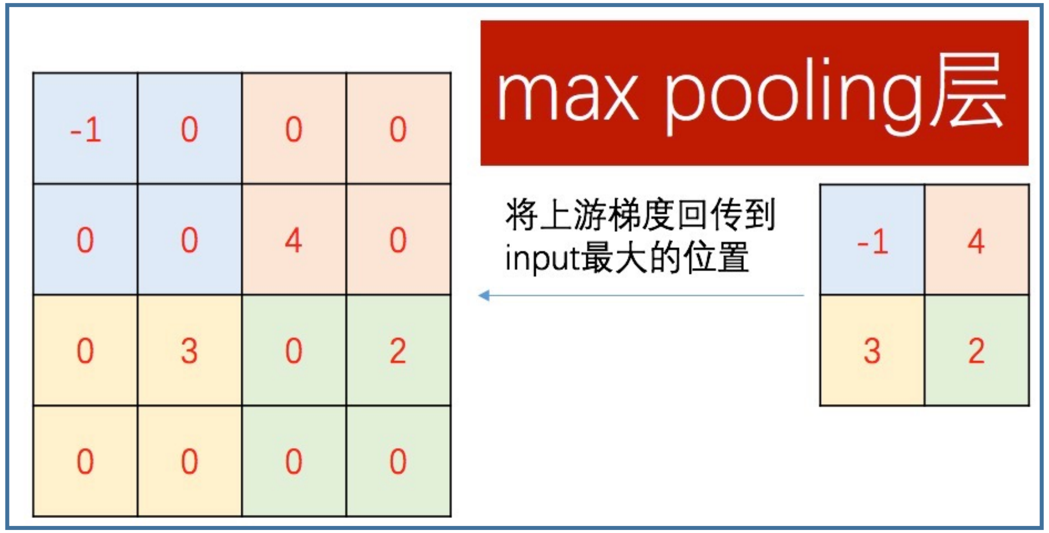
3	0	0	1	1
0	2	0	1	2
0	0	2	4	0
1	0	1	0	1
1	2	0	3	5

0	0	0	0	0	0	0
0	3	0	0	1	1	0
0	0	2	0	1	2	0
0	0	0	2	4	0	0
0	1	0	1	0	1	0
0	1	2	0	3	5	0
0	0	0	0	0	0	0

完整的CNN



L05：梯度计算

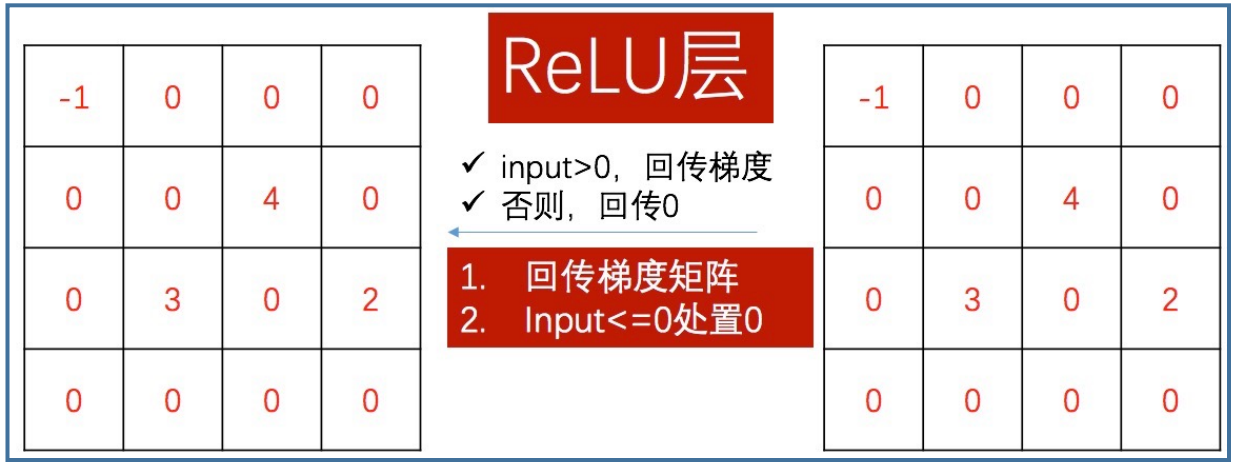


max pooling层

-1	0	0	0
0	0	4	0
0	3	0	2
0	0	0	0

将上游梯度回传到input最大的位置

-1	4
3	2



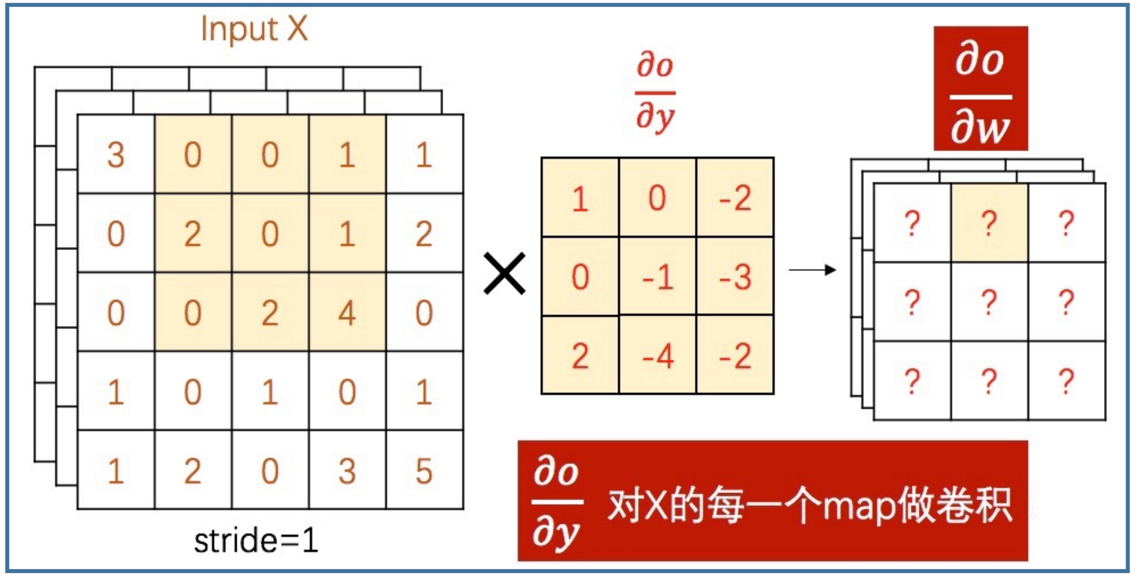
ReLU层

-1	0	0	0
0	0	4	0
0	3	0	2
0	0	0	0

- ✓ input>0，回传梯度
- ✓ 否则，回传0

1. 回传梯度矩阵
2. Input<=0处置0

-1	0	0	0
0	0	4	0
0	3	0	2
0	0	0	0



Input X

3	0	0	1	1
0	2	0	1	2
0	0	2	4	0
1	0	1	0	1
1	2	0	3	5

stride=1

×

$\frac{\partial o}{\partial y}$

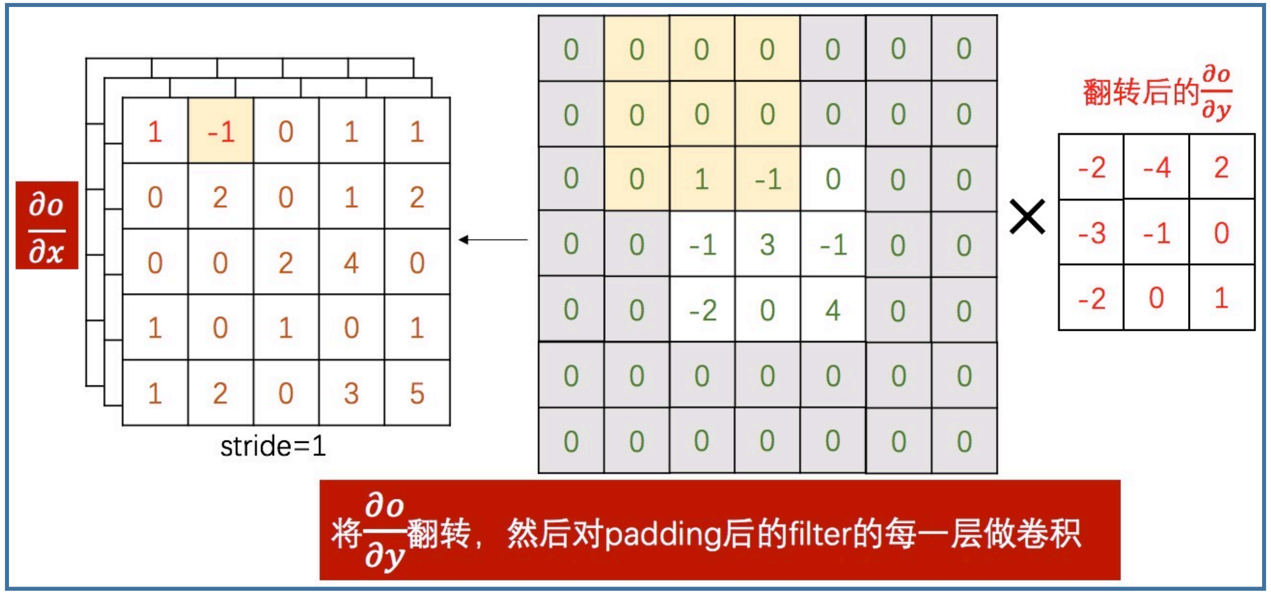
1	0	-2
0	-1	-3
2	-4	-2

→

$\frac{\partial o}{\partial w}$

?	?	?
?	?	?
?	?	?

$\frac{\partial o}{\partial y}$ 对X的每一个map做卷积



$\frac{\partial o}{\partial x}$

1	-1	0	1	1
0	2	0	1	2
0	0	2	4	0
1	0	1	0	1
1	2	0	3	5

stride=1

←

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	1	-1	0	0	0
0	0	-1	3	-1	0	0
0	0	-2	0	4	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

×

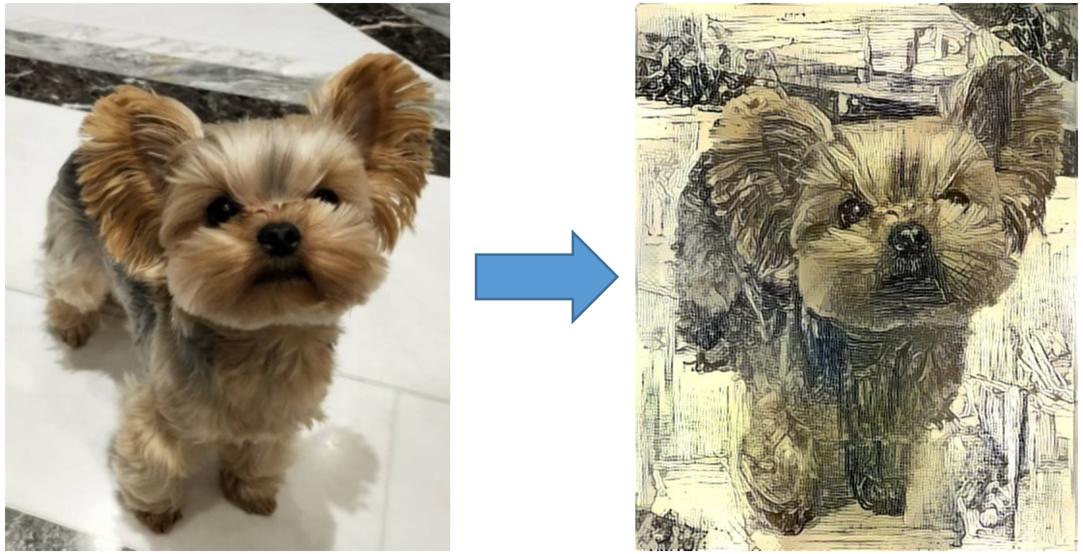
翻转后的$\frac{\partial o}{\partial y}$

-2	-4	2
-3	-1	0
-2	0	1

将$\frac{\partial o}{\partial y}$翻转，然后对padding后的filter的每一层做卷积

除了分类，CNN还可以做什么？

- 风格迁移



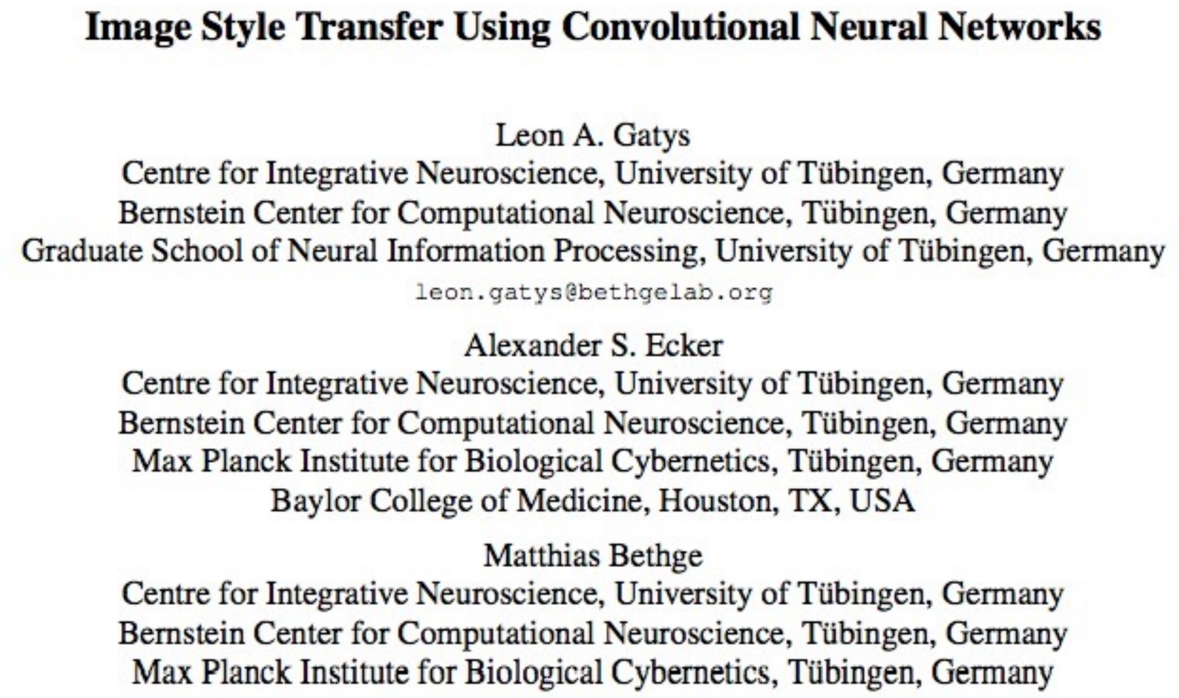


Image Style Transfer Using Convolutional Neural Networks

Leon A. Gatys
Centre for Integrative Neuroscience, University of Tübingen, Germany
Bernstein Center for Computational Neuroscience, Tübingen, Germany
Graduate School of Neural Information Processing, University of Tübingen, Germany
leon.gatys@bethgelab.org

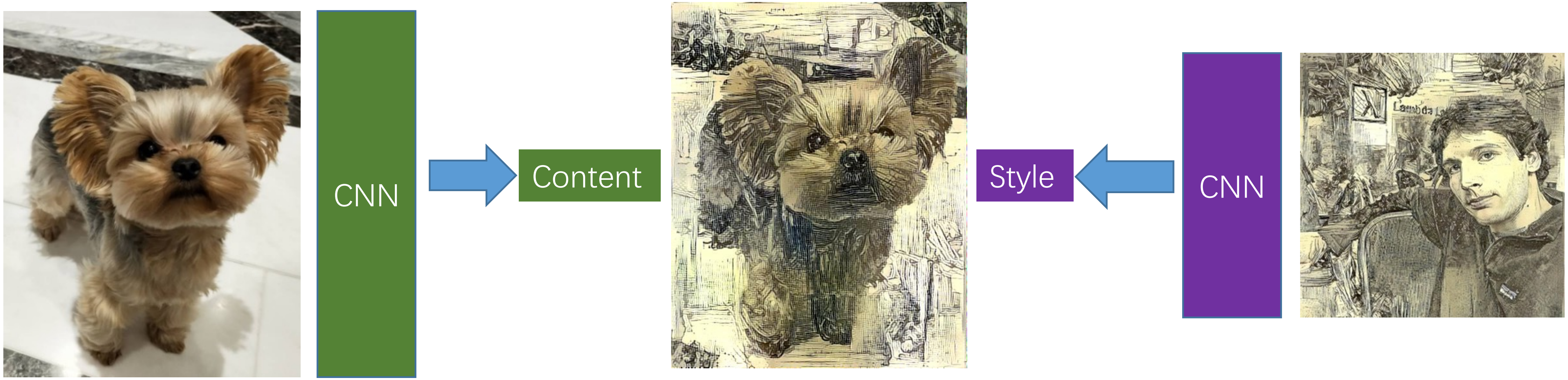
Alexander S. Ecker
Centre for Integrative Neuroscience, University of Tübingen, Germany
Bernstein Center for Computational Neuroscience, Tübingen, Germany
Max Planck Institute for Biological Cybernetics, Tübingen, Germany
Baylor College of Medicine, Houston, TX, USA

Matthias Bethge
Centre for Integrative Neuroscience, University of Tübingen, Germany
Bernstein Center for Computational Neuroscience, Tübingen, Germany
Max Planck Institute for Biological Cybernetics, Tübingen, Germany

除了分类，CNN还可以做什么？

- 风格迁移

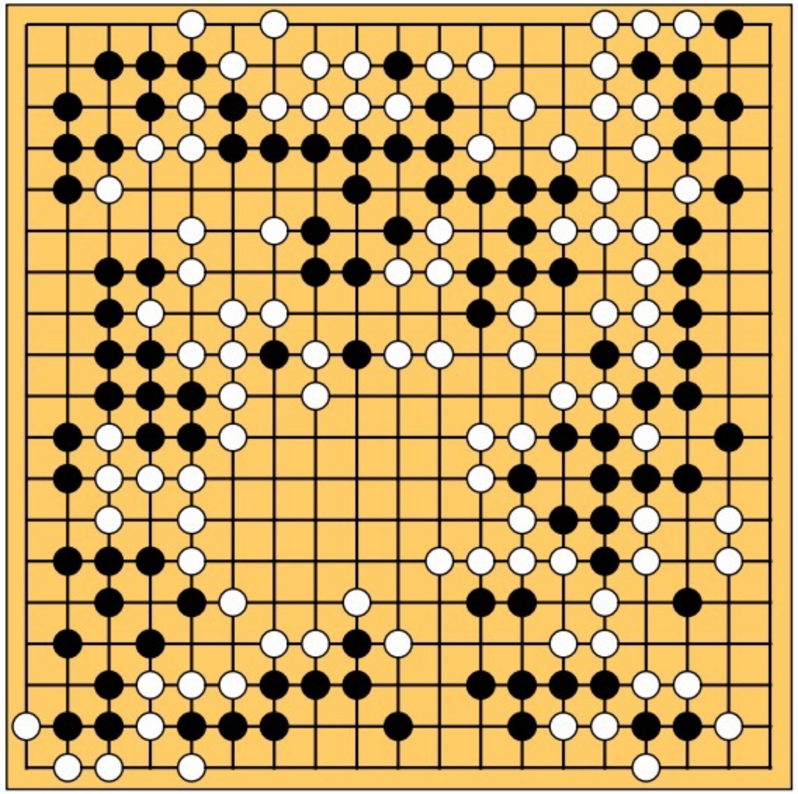
$$x = \underset{x}{\operatorname{argmin}}\, l_{(x,content)} + l_{(x,style)}$$



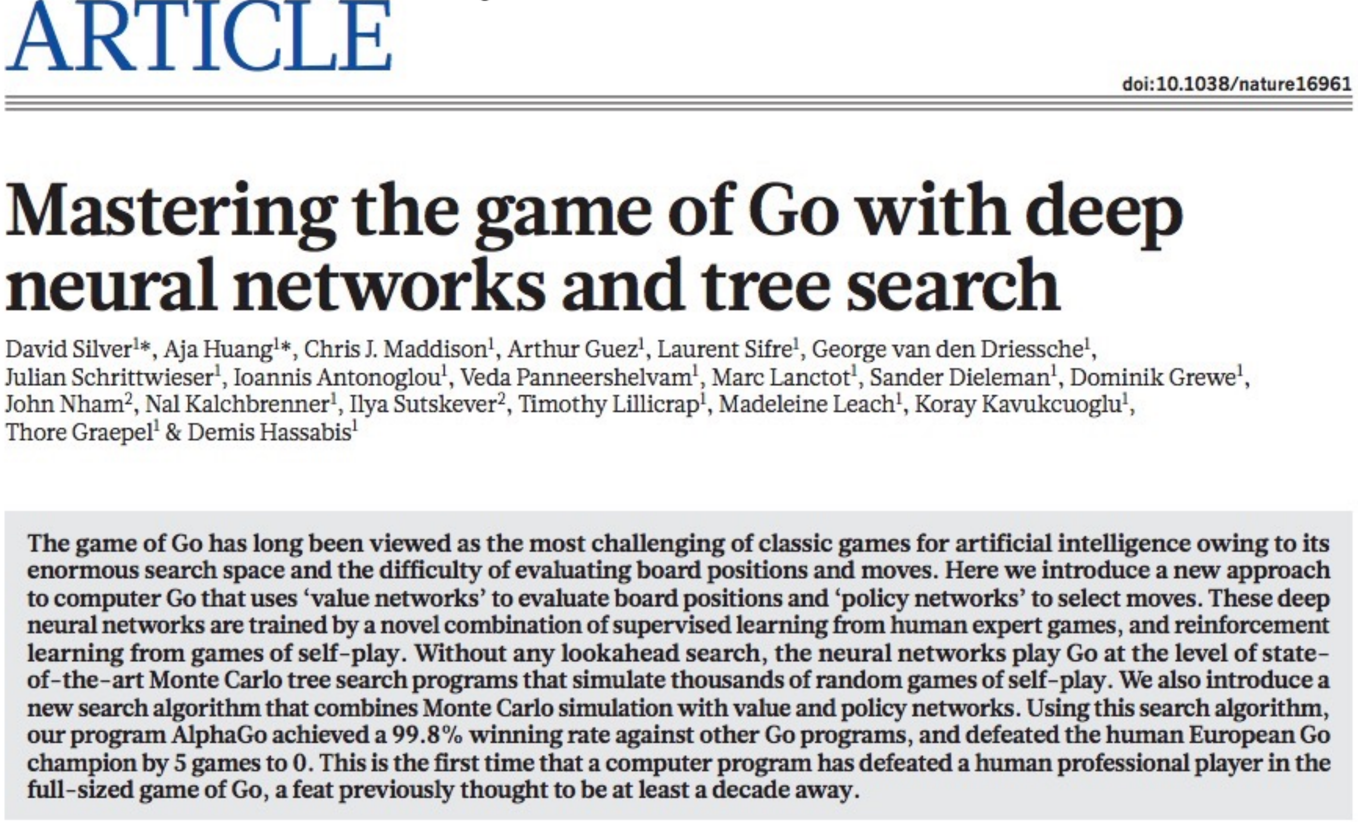
https://dreamscopeapp.com/

除了分类，CNN还可以做什么？

- 下围棋



AlphaGo



ARTICLE

doi:10.1038/nature16961

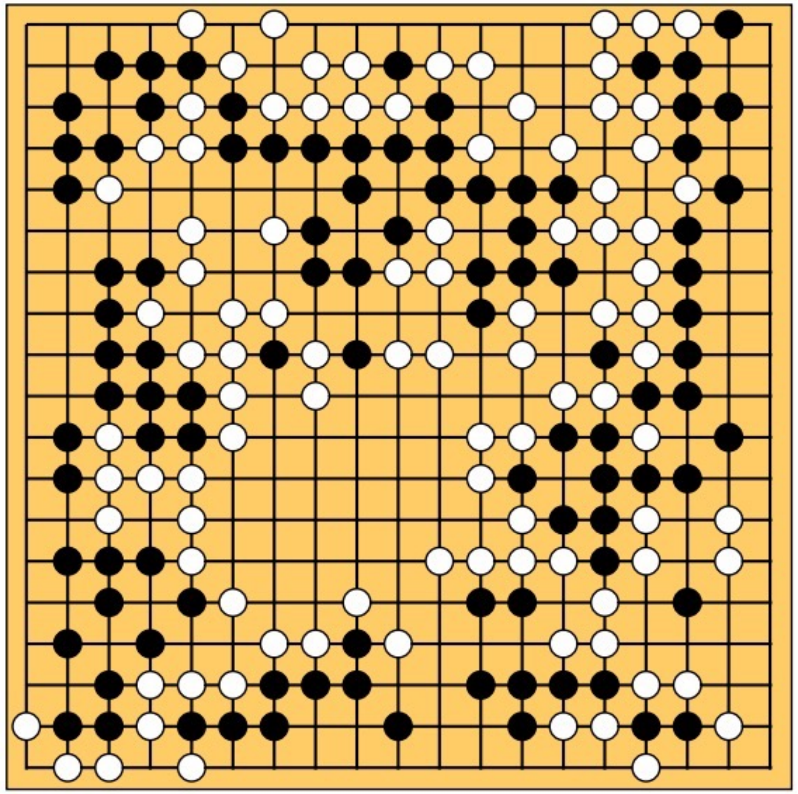
Mastering the game of Go with deep neural networks and tree search

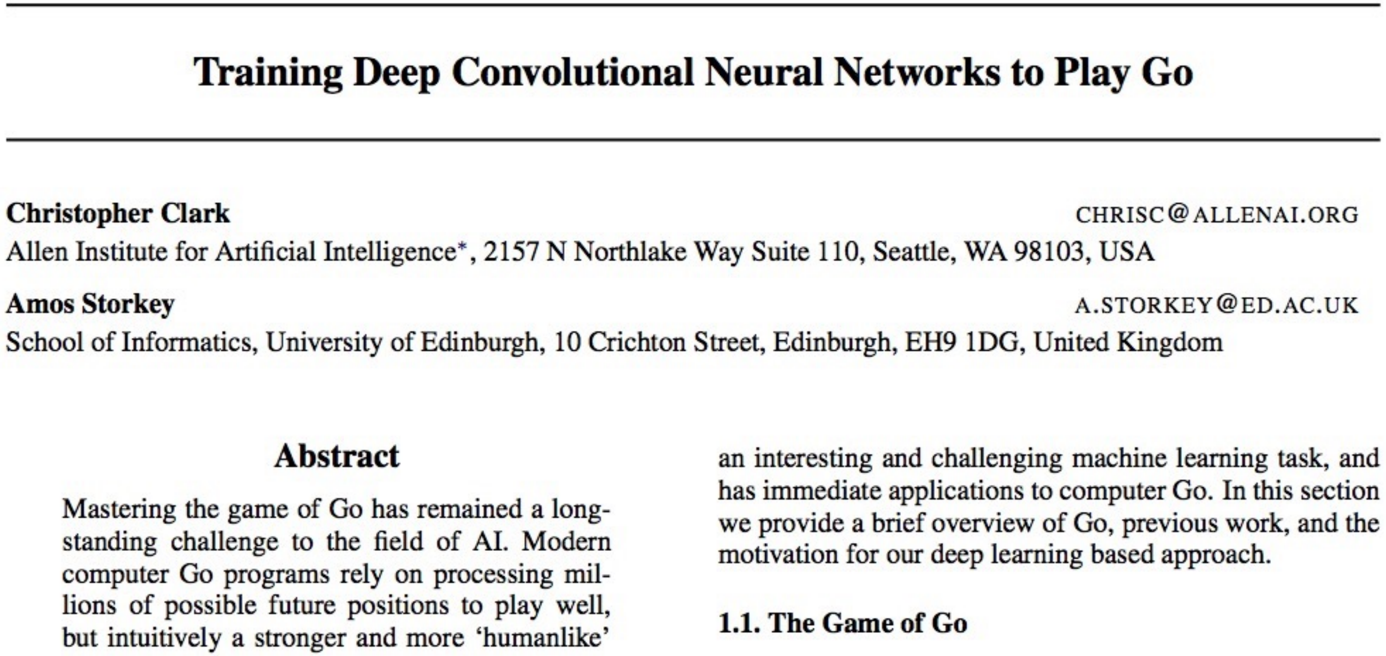
David Silver[1]*, Aja Huang[1]*, Chris J. Maddison[1], Arthur Guez[1], Laurent Sifre[1], George van den Driessche[1], Julian Schrittwieser[1], Ioannis Antonoglou[1], Veda Panneershelvam[1], Marc Lanctot[1], Sander Dieleman[1], Dominik Grewe[1], John Nham[2], Nal Kalchbrenner[1], Ilya Sutskever[2], Timothy Lillicrap[1], Madeleine Leach[1], Koray Kavukcuoglu[1], Thore Graepel[1] & Demis Hassabis[1]

The game of Go has long been viewed as the most challenging of classic games for artificial intelligence owing to its enormous search space and the difficulty of evaluating board positions and moves. Here we introduce a new approach to computer Go that uses 'value networks' to evaluate board positions and 'policy networks' to select moves. These deep neural networks are trained by a novel combination of supervised learning from human expert games, and reinforcement learning from games of self-play. Without any lookahead search, the neural networks play Go at the level of state-of-the-art Monte Carlo tree search programs that simulate thousands of random games of self-play. We also introduce a new search algorithm that combines Monte Carlo simulation with value and policy networks. Using this search algorithm, our program AlphaGo achieved a 99.8% winning rate against other Go programs, and defeated the human European Go champion by 5 games to 0. This is the first time that a computer program has defeated a human professional player in the full-sized game of Go, a feat previously thought to be at least a decade away.

除了分类，CNN还可以做什么？

- 下围棋





Training Deep Convolutional Neural Networks to Play Go

Christopher Clark CHRISC@ALLENAI.ORG
Allen Institute for Artificial Intelligence*, 2157 N Northlake Way Suite 110, Seattle, WA 98103, USA

Amos Storkey A.STORKEY@ED.AC.UK
School of Informatics, University of Edinburgh, 10 Crichton Street, Edinburgh, EH9 1DG, United Kingdom

Abstract

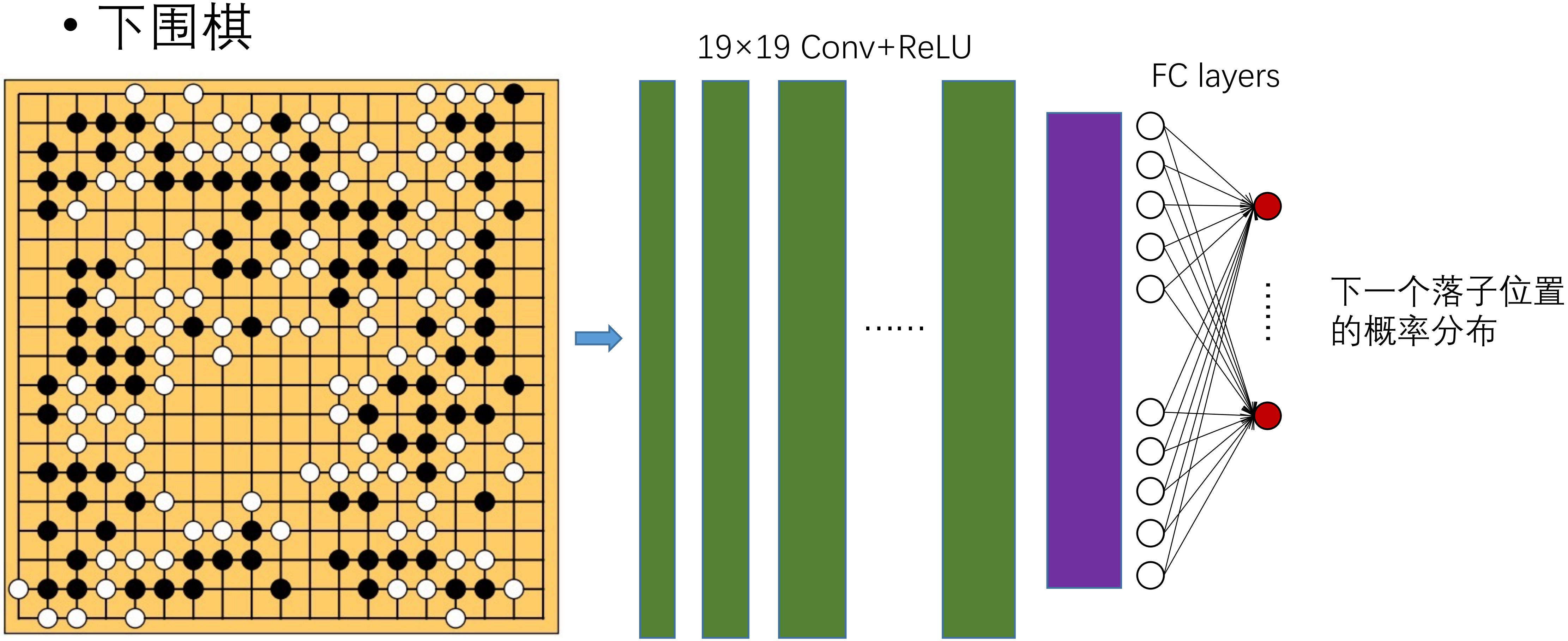
Mastering the game of Go has remained a long-standing challenge to the field of AI. Modern computer Go programs rely on processing millions of possible future positions to play well, but intuitively a stronger and more ‘humanlike’

an interesting and challenging machine learning task, and has immediate applications to computer Go. In this section we provide a brief overview of Go, previous work, and the motivation for our deep learning based approach.

1.1. The Game of Go

除了分类，CNN还可以做什么？

- 下围棋



19×19 Conv+ReLU

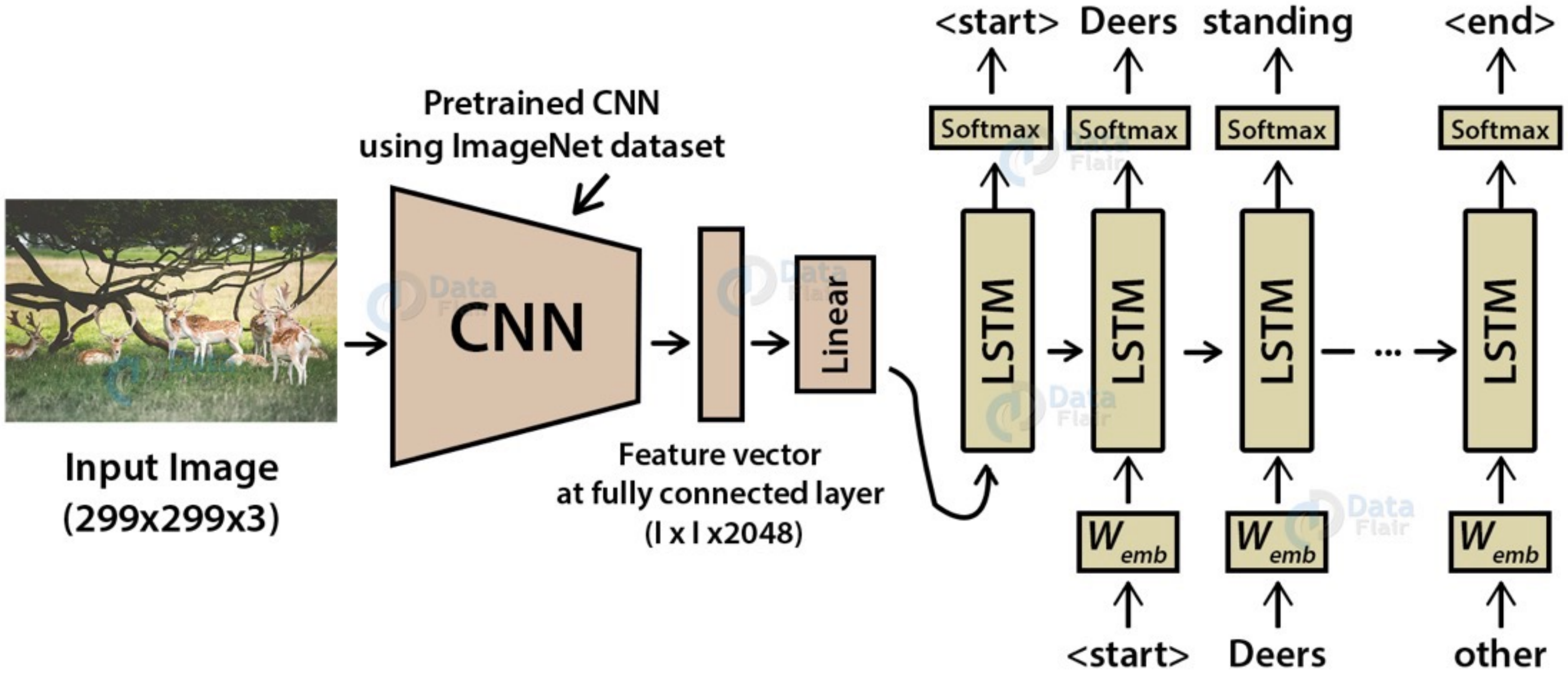
……

FC layers

下一个落子位置的概率分布

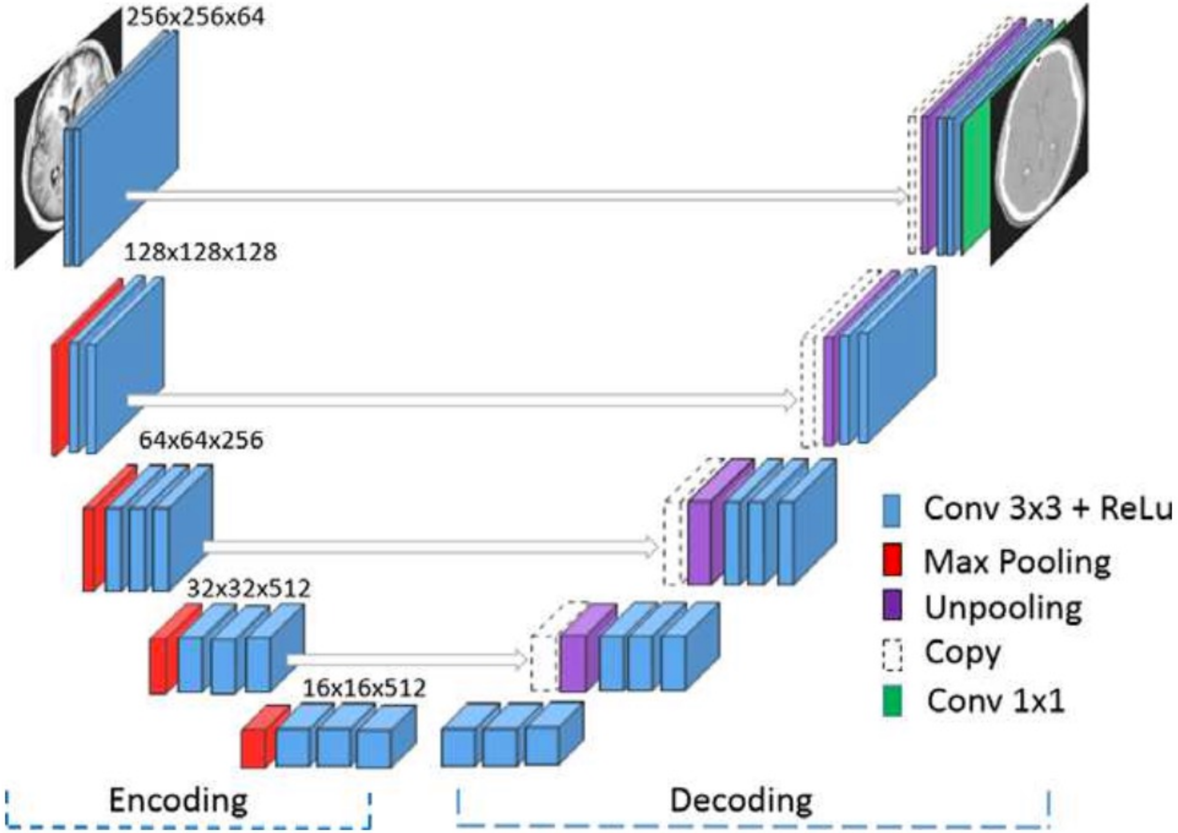
除了分类，CNN还可以做什么？

- 看图说话（图像描述）



除了分类，CNN还可以做什么？

- 图像生成



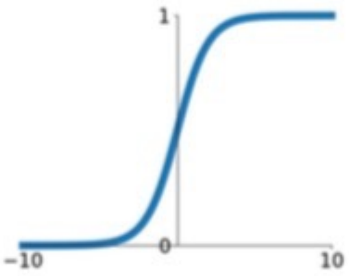
神经网络的训练

- 训练时一次性设置
  - ✓激活函数，数据预处理，权重参数初始化，正则化等
- 训练中动态变化
  - ✓训练过程的一般设置，超参数的选择，优化器，参数更新
- 模型评价
  - ✓模型集成，测试时补充

激活函数

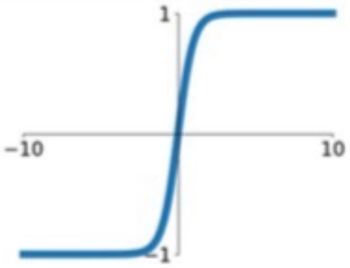
Sigmoid

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



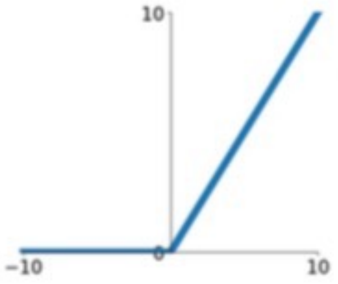
tanh

$\tanh(x)$



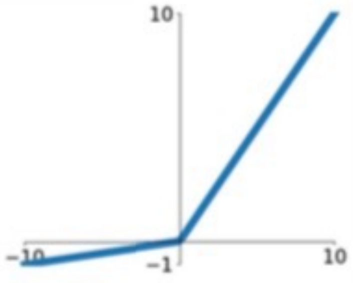
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

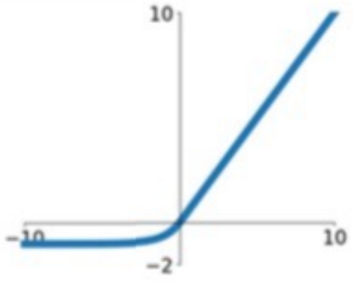


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

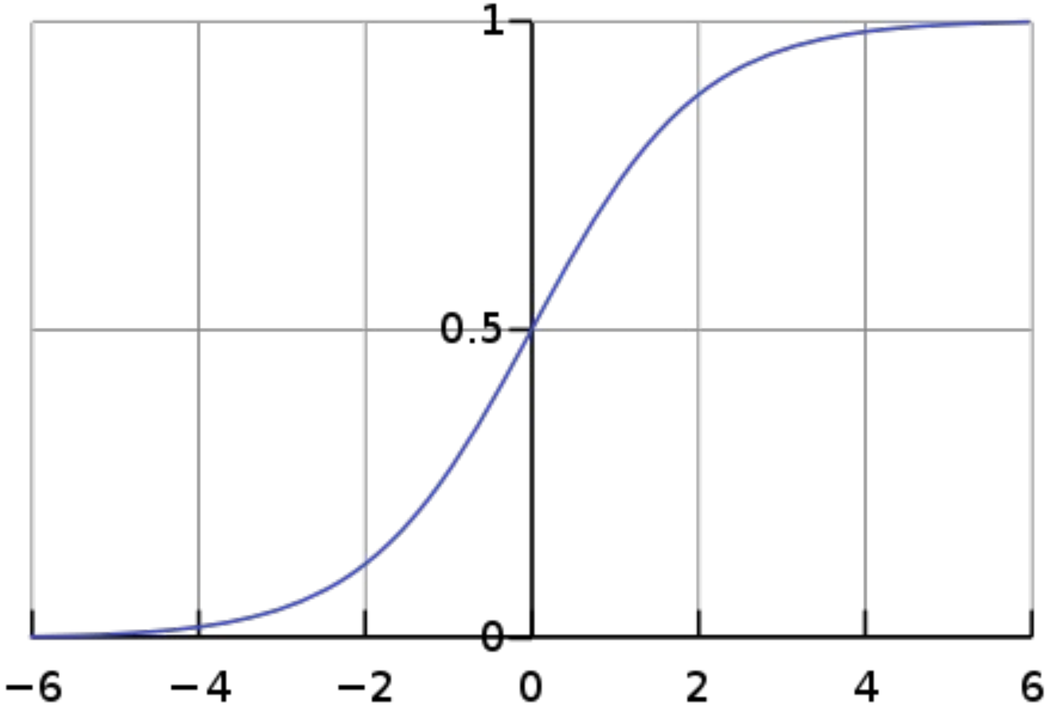
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Sigmoid

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



优点

✓ 将数值压缩到(0, 1)之间

✓ 曲线平滑，便于求导

缺点

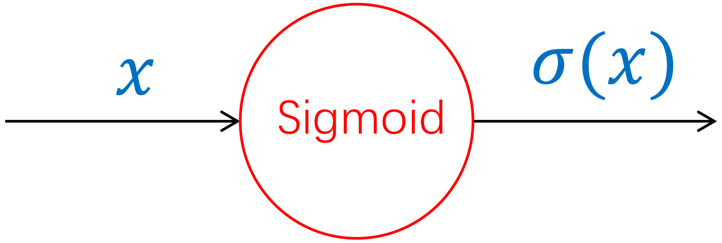
✓ 容易饱和输出

✓ 不是零均值（zero-centered）

✓ exp()函数计算复杂度高

Sigmoid：饱和输出

Local gradient: (1-$\sigma(x)$)×$\sigma(x)$

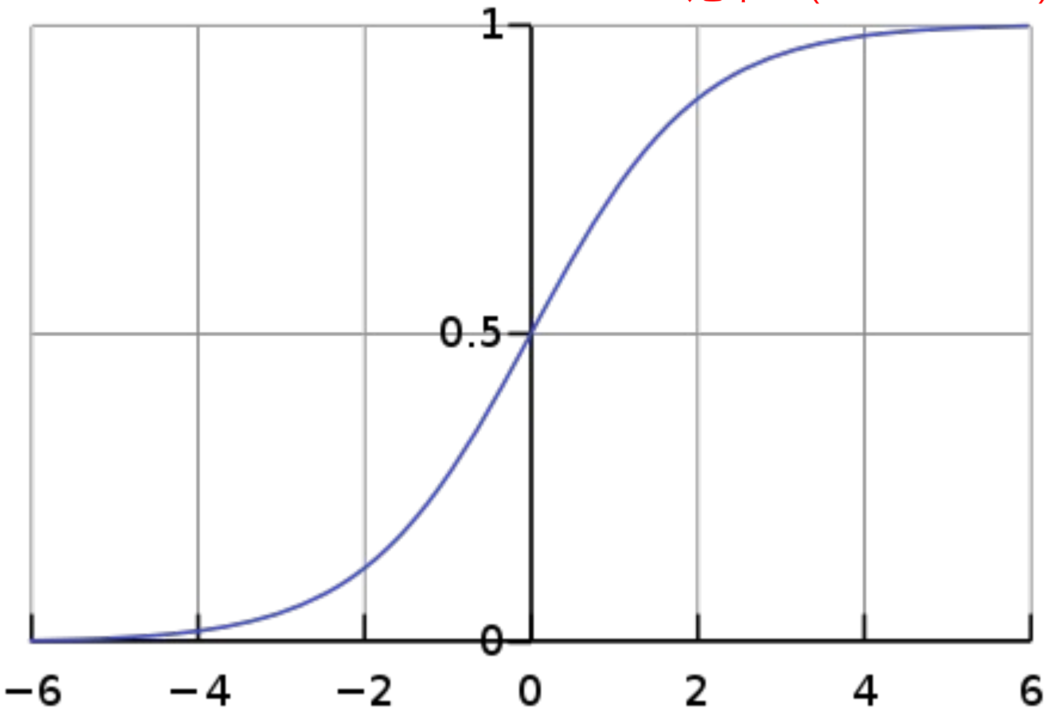


Sigmoid gate

当x稍大或稍小时

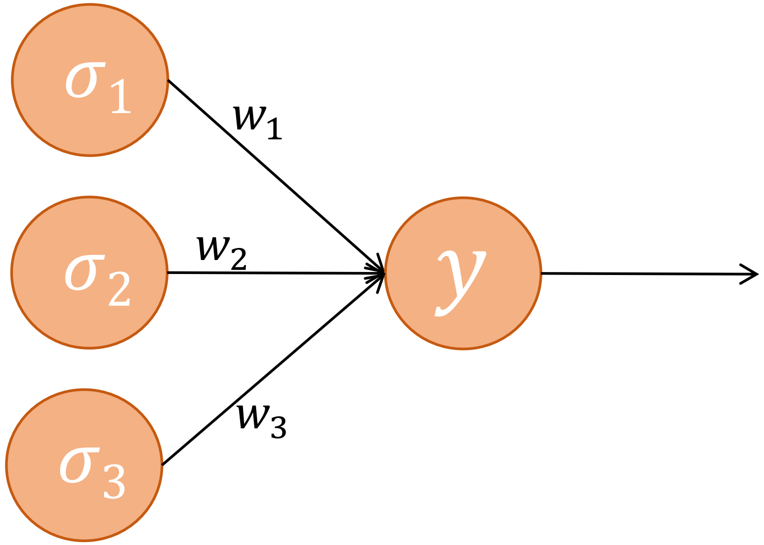
- ✓ $\sigma(x)$即接近1或者0，且基本维持不变
- ✓ sigmoid门的局部梯度接近于0，造成回传梯度消失，参数无法更新

饱和（Saturation）

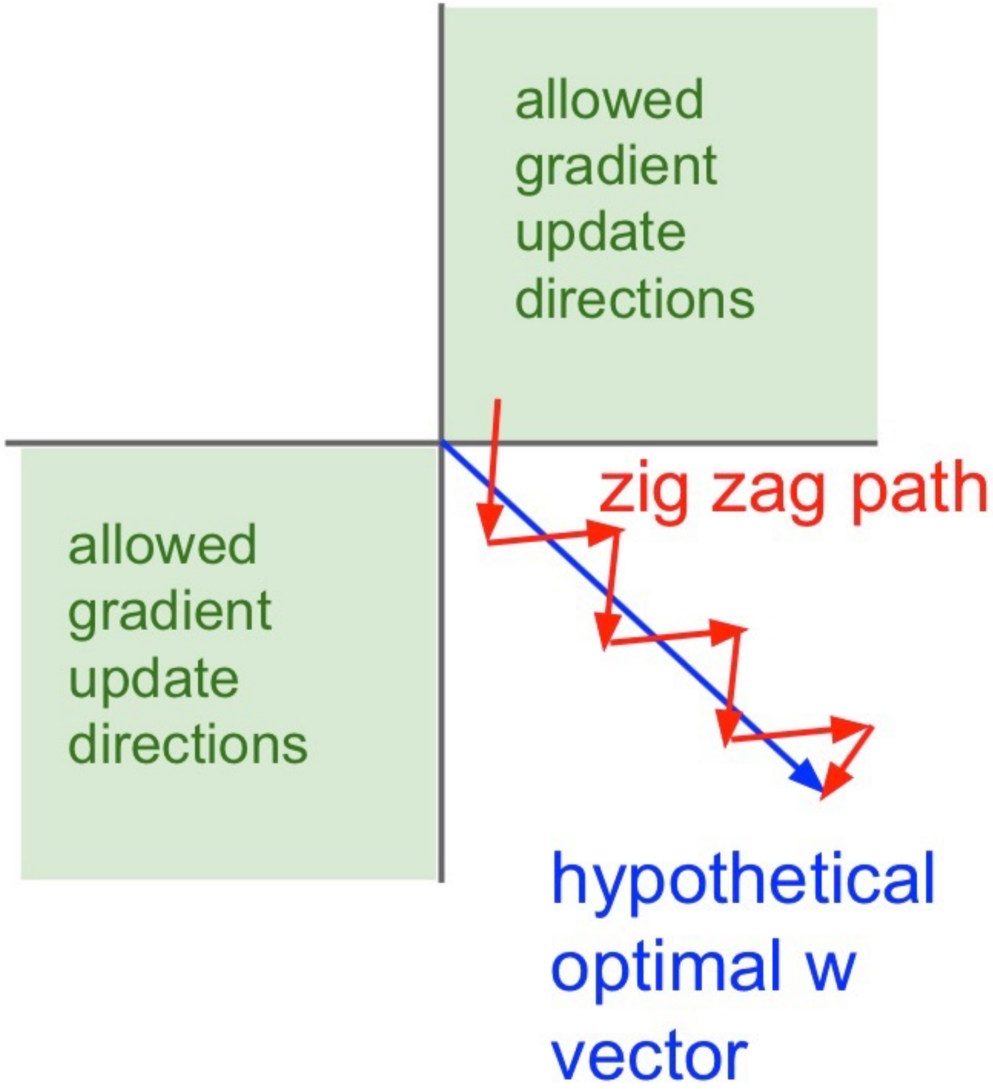


饱和（Saturation）

Sigmoid：非零均值



- ✓ σ_i都是正数
- ✓ $\frac{\partial h}{\partial w_i} = \frac{\partial h}{\partial y}\frac{\partial y}{\partial wi} = \frac{\partial h}{\partial y}\sigma_i$
- ✓ w_i的梯度都是正数或者负数，梯度“之字形”更新，收敛慢

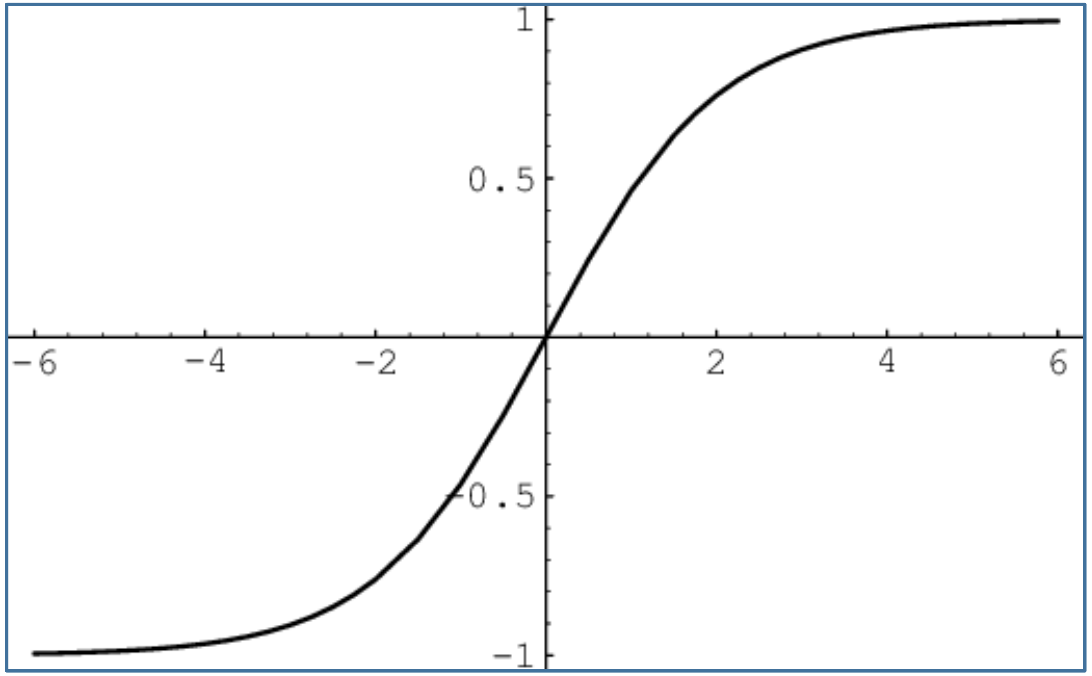


allowed gradient update directions
zig zag path
allowed gradient update directions
hypothetical optimal w vector

tanh

$$\tanh x = \frac{\sinh x}{\cosh x} = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1}$$

饱和（Saturation）



饱和（Saturation）

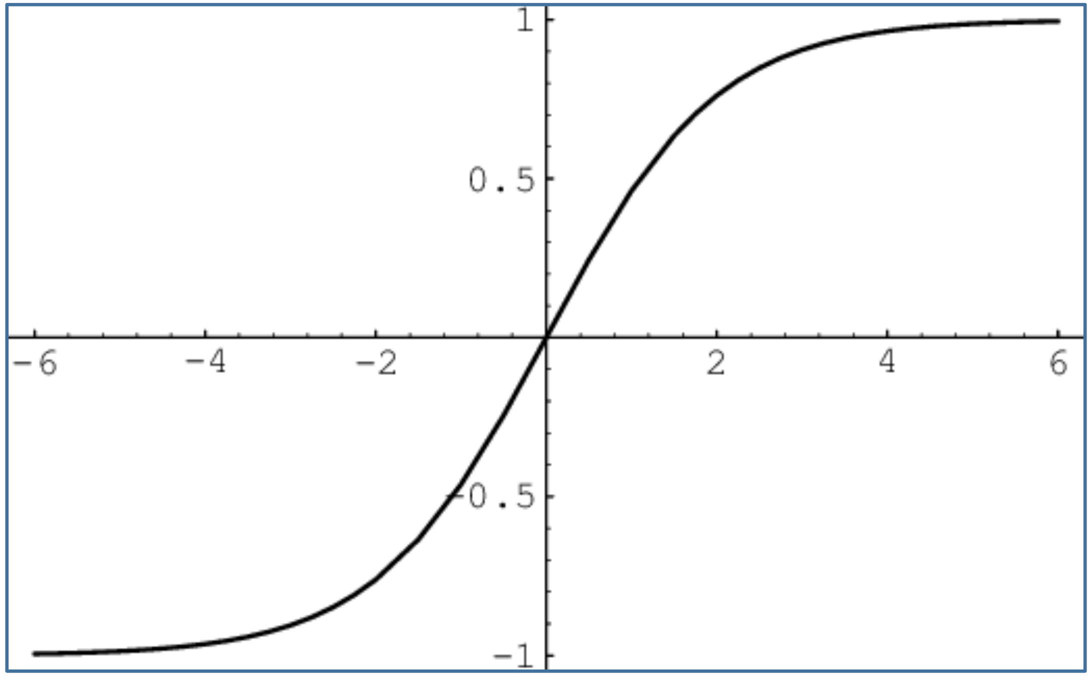
优点

- ✓ 将数值压缩到(-1, 1)之间（zero-centered）
- ✓ 曲线平滑，便于求导

tanh

$$\tanh x = \frac{\sinh x}{\cosh x} = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1}$$

饱和（Saturation）



饱和（Saturation）

优点

✓ 将数值压缩到(-1, 1)之间（zero-centered）

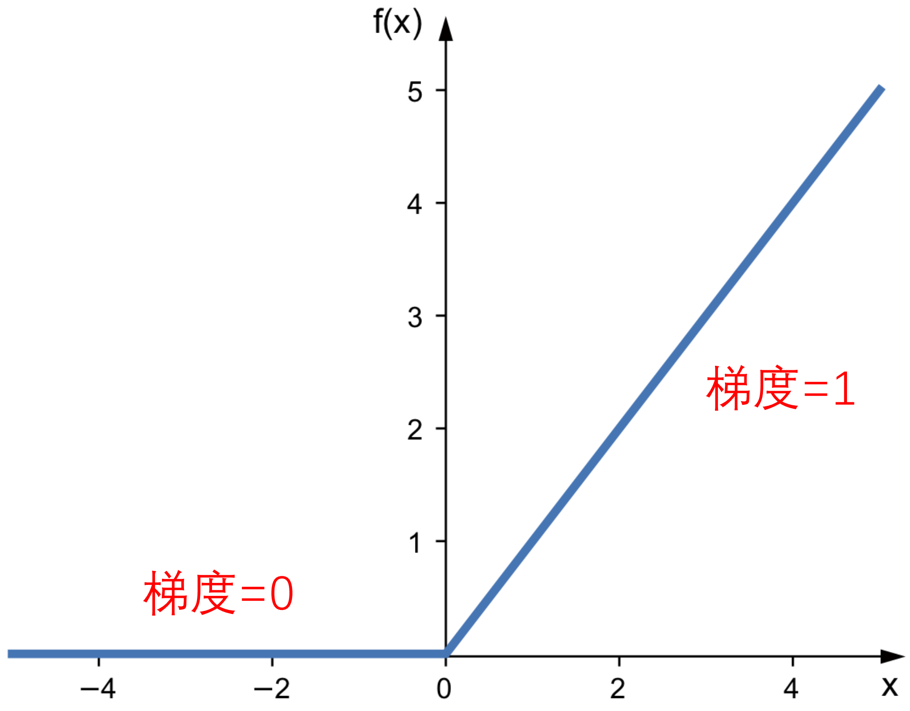
✓ 曲线平滑，便于求导

缺点

✓ 容易饱和输出

✓ exp()函数计算复杂度高

ReLU（Rectified Linear Unit）



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

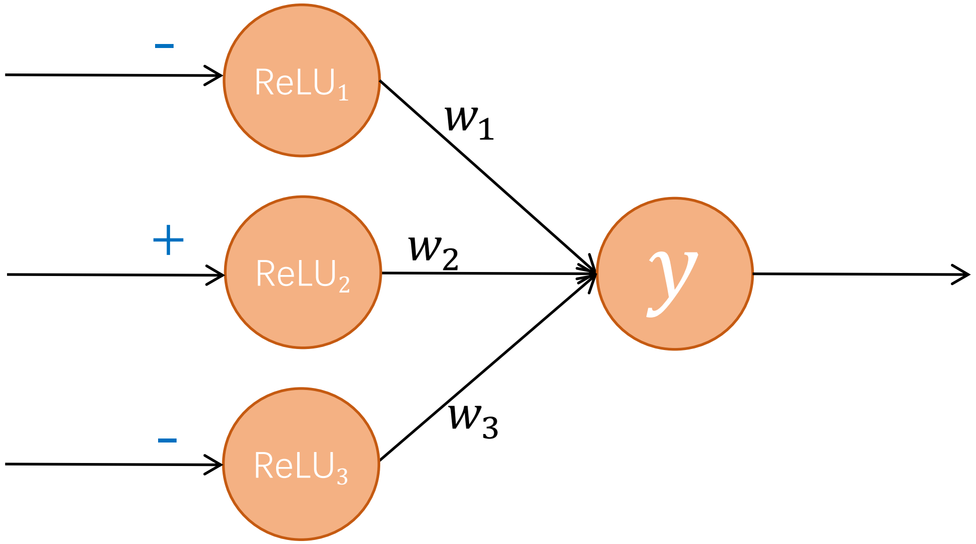
优点

- ✓ 在正区间不会饱和
- ✓ 计算复杂度极低
- ✓ 收敛速度比Sigmoid和tanh快

缺点

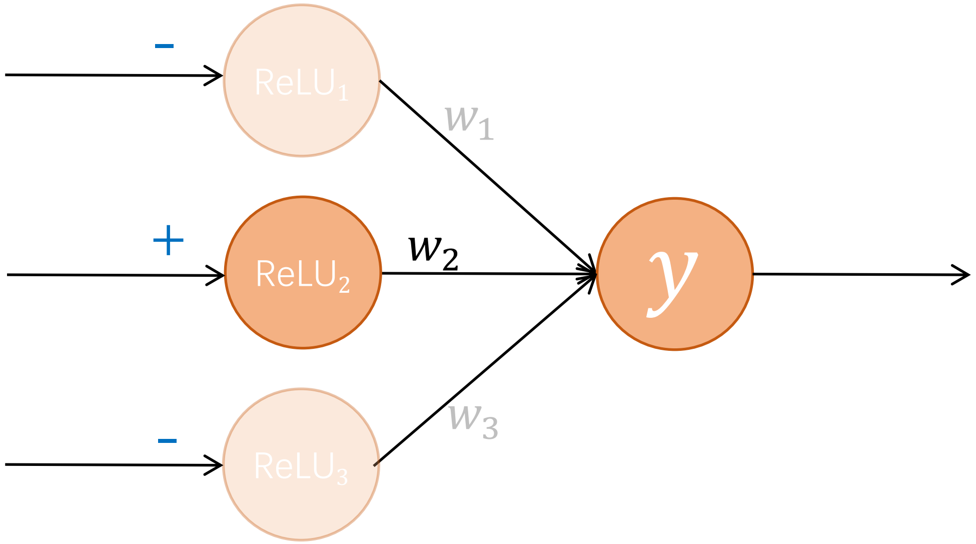
- ✓ 不是零均值
- ✓ 不压缩数据，数据幅度会随着网络加深不断增大
- ✓ 神经元坏死（Dead ReLU）

神经元坏死（Dead ReLU）



✓ 由于参数初始化或者学习率设置不当，导致某些神经元的输入永远是负数

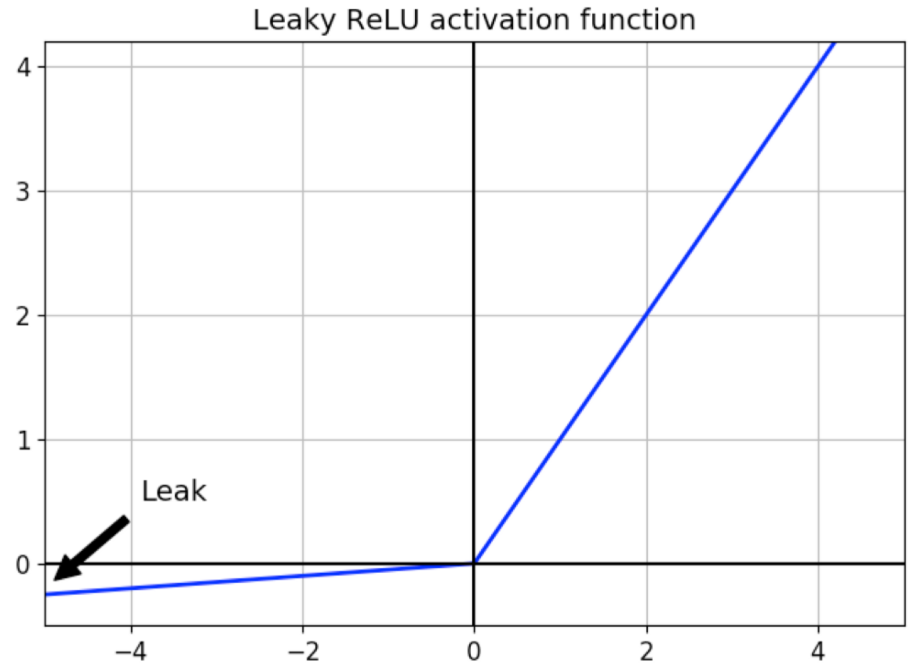
神经元坏死（Dead ReLU）



- ✓ 由于参数初始化或者学习率设置不当，导致某些神经元的输入永远是负数
- ✓ 导致相应的参数永远不会更新

采用合适的参数初始化和调整学习率可以缓解这种现象

Leaky ReLU



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

优点

- ✓ 不会造成饱和
- ✓ 计算复杂度低
- ✓ 收敛速度比Sigmoid和tanh快
- ✓ 近似零均值
- ✓ 解决ReLU的神经元坏死问题

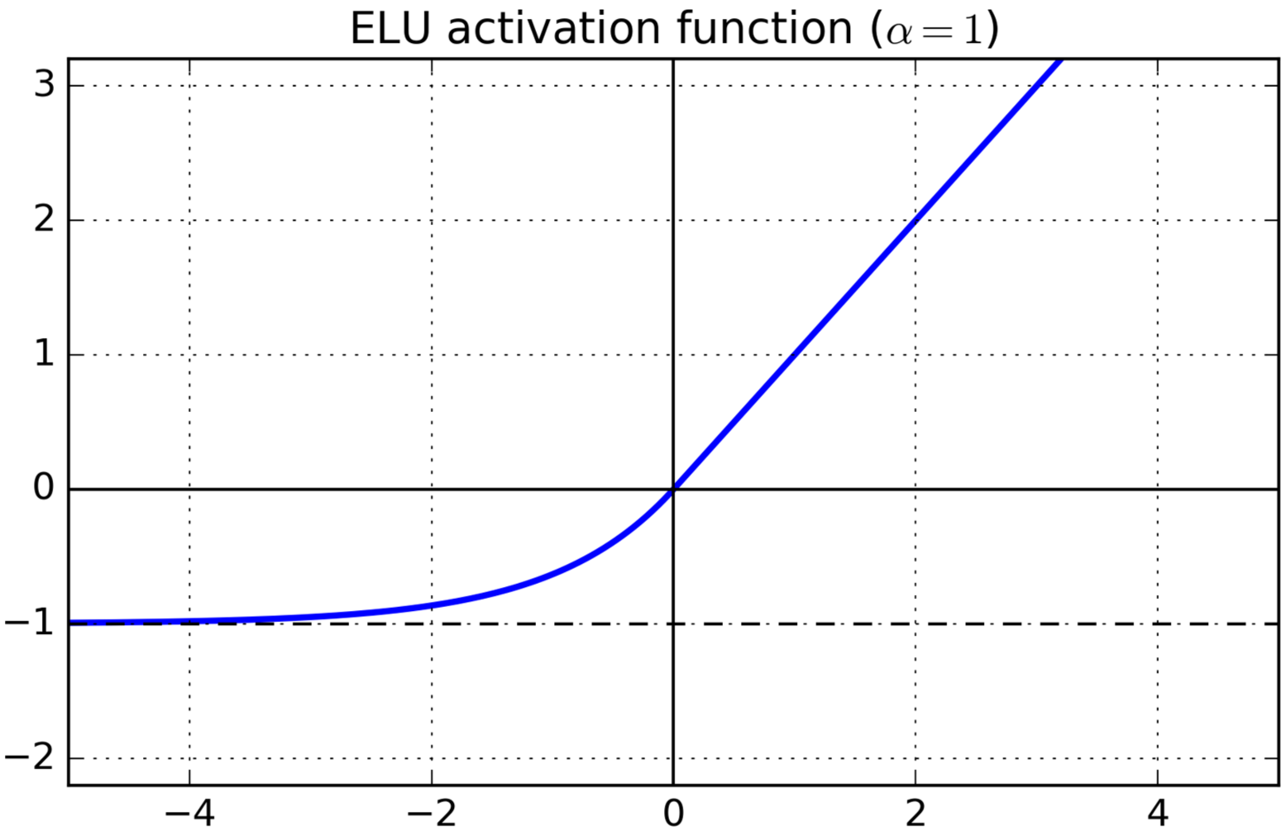
可泛化成Parametric ReLU (PReLU)

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

缺点

- ✓ 数值幅度不断增大
- ✓ 实际表现不一定比ReLU好

ELU（Exponential Linear Units）



$$f(x) = \begin{cases} x, & x > 0 \\ \alpha\,(\exp(x) - 1), & x \leq 0 \end{cases}$$

优点

✓ 不易造成饱和

✓ 收敛速度比Sigmoid和tanh快

✓ 近似零均值

✓ 解决ReLU的神经元坏死问题

缺点

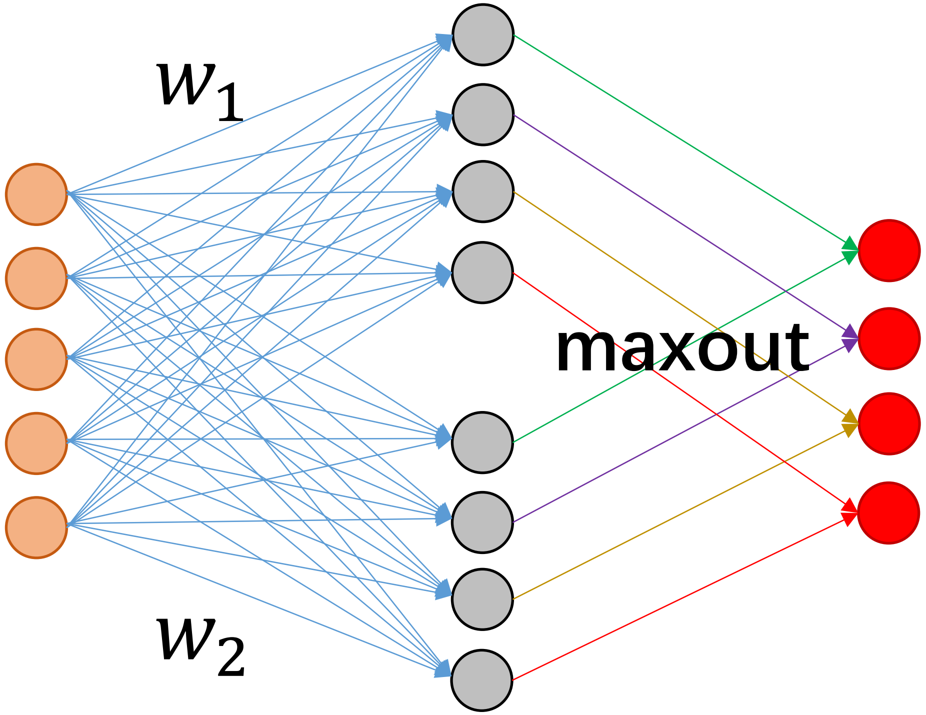
✓ exp()计算复杂度高

✓ 表现不一定比ReLU好

Maxout

$w \in \mathbb{R}^{m \times n \times 2}$

$$\mathrm{f}(x) = \max(w_1^T x + b_1, w_2^T\ x + b_2) \quad x \in \mathbb{R}^m,\ \mathrm{f}(x) \in \mathbb{R}^n,\ w_1,\ w_2 \in \mathbb{R}^{m \times n}$$



$$\mathrm{f}(x) = \max(w_1^T x + b_1, \dots, w_k^T\ x + b_k) \qquad w \in \mathbb{R}^{m \times n \times k}$$

实际搭建模型的时候。。。

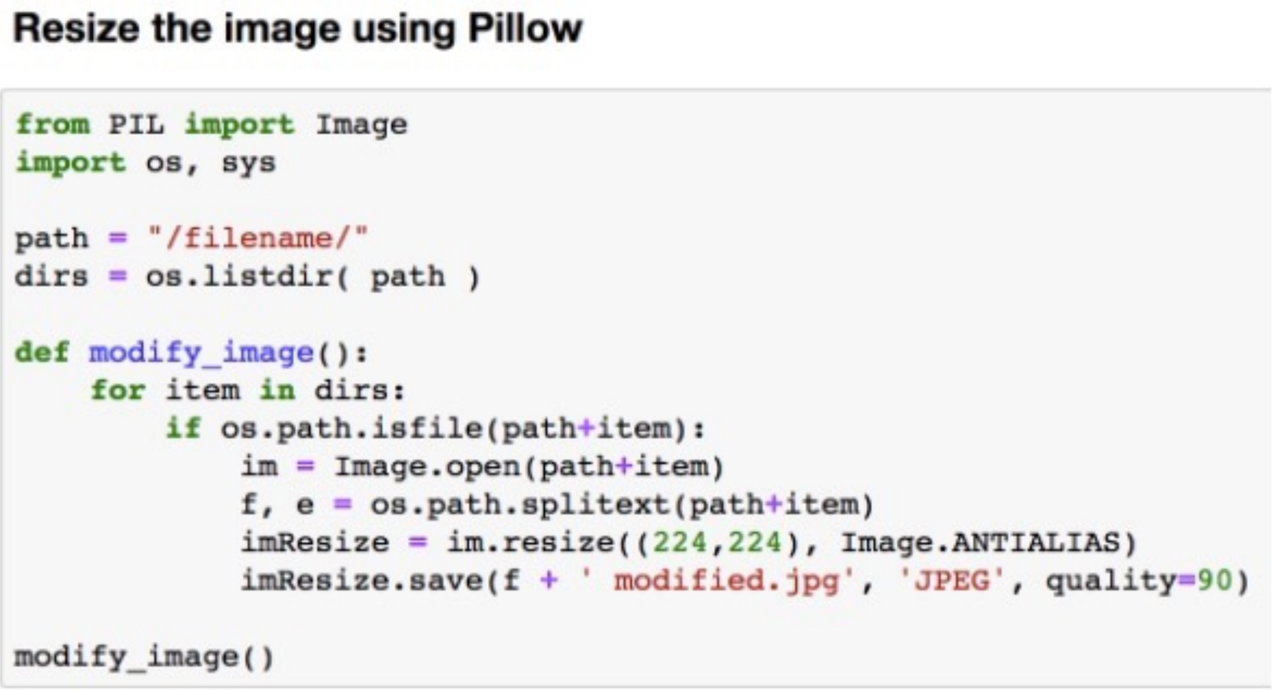
- 首选ReLU，但是要注意初始化和学习率设置
- 不要使用Sigmoid
- 可以使用tanh，不过效果一般来讲一般
- 尝试其余激活函数

数据预处理

- 调整图像大小（Resize the images）
- 图像序列化（Pickle the images）
- 零均值化（Zero centering）
- 标准化（Normalization）

调整图像大小

- 一般将图像裁剪为大小一致的正方形
- 可以通过downscale或者upscale调整大小
- e.g.，使用Pillow的crop()和resize()方法

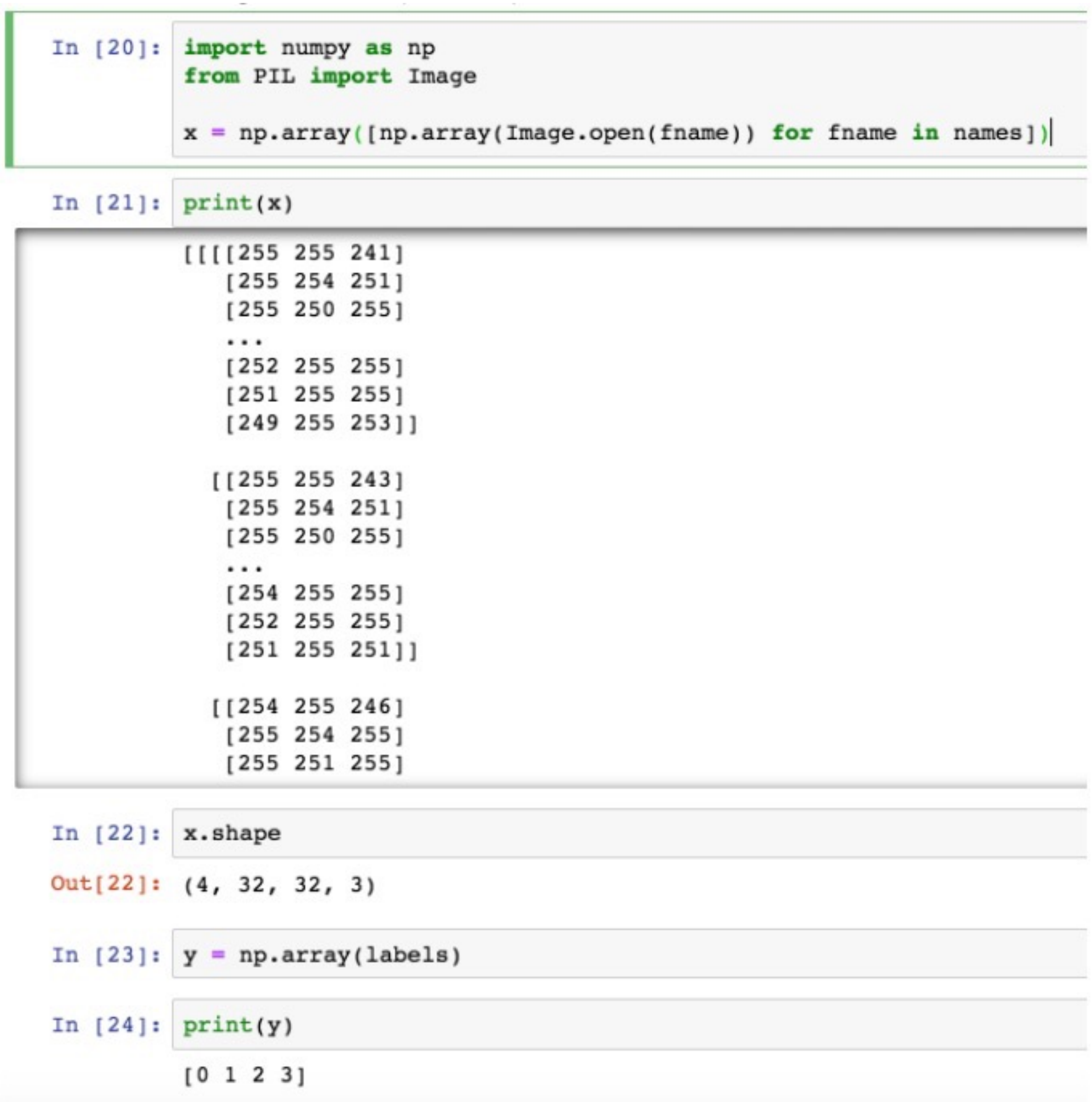


Resize the image using Pillow

```
from PIL import Image
import os, sys

path = "/filename/"
dirs = os.listdir( path )

def modify_image():
    for item in dirs:
        if os.path.isfile(path+item):
            im = Image.open(path+item)
            f, e = os.path.splitext(path+item)
            imResize = im.resize((224,224), Image.ANTIALIAS)
            imResize.save(f + ' modified.jpg', 'JPEG', quality=90)

modify_image()
```

图像序列化

- 使用pickle模块将图片转化为像素值数组，并附上相应标签



```
In [20]: import numpy as np
         from PIL import Image

         x = np.array([np.array(Image.open(fname)) for fname in names])
```

```
In [21]: print(x)
```

```
[[[[255 255 241]
   [255 254 251]
   [255 250 255]
   ...
   [252 255 255]
   [251 255 255]
   [249 255 253]]

  [[255 255 243]
   [255 254 251]
   [255 250 255]
   ...
   [254 255 255]
   [252 255 255]
   [251 255 251]]

  [[254 255 246]
   [255 254 255]
   [255 251 255]
```

```
In [22]: x.shape
Out[22]: (4, 32, 32, 3)
```

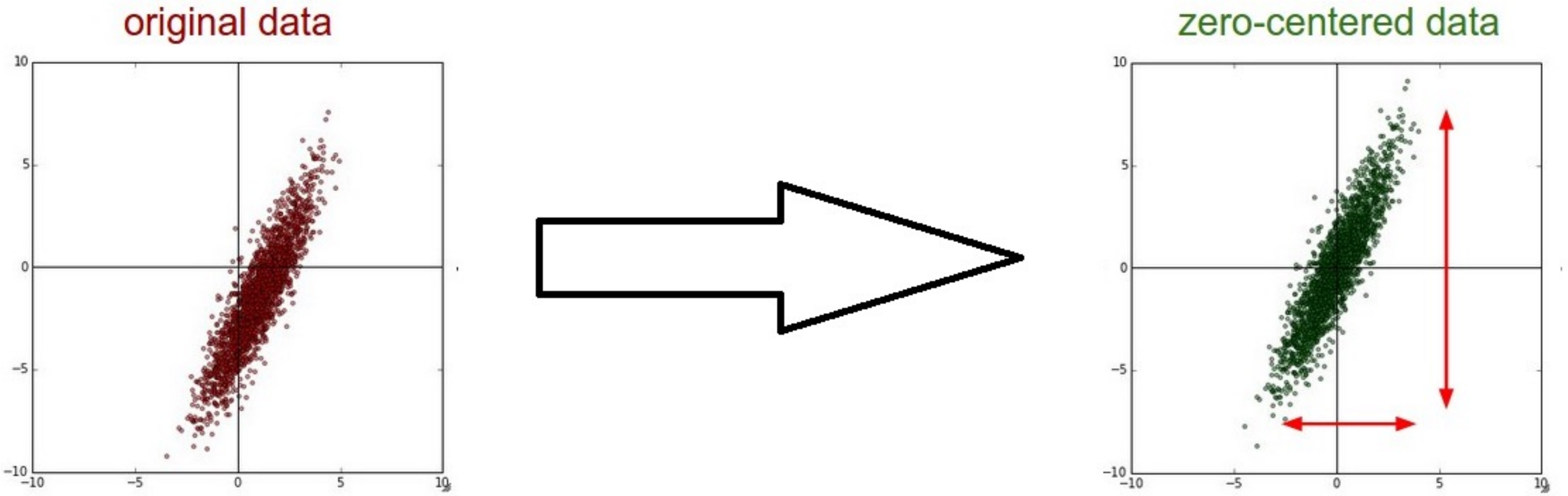
```
In [23]: y = np.array(labels)
```

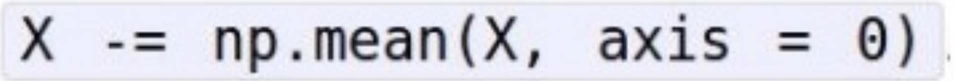
```
In [24]: print(y)
         [0 1 2 3]
```

零均值化

- 将原始像素值从[0, 255]调整为[-128, 127]

回想如果输入数据不是零均值，会有什么影响？





```
X -= np.mean(X, axis = 0)
```

零均值化

- 将原始像素值从[0, 255]调整为[-128, 127]

- 计算所有图像的平均，得到mean image
  - ✓mean image和原始图像的大小一致

- 将每个图像减去mean image
  - ✓e.g.，AlexNet

零均值化

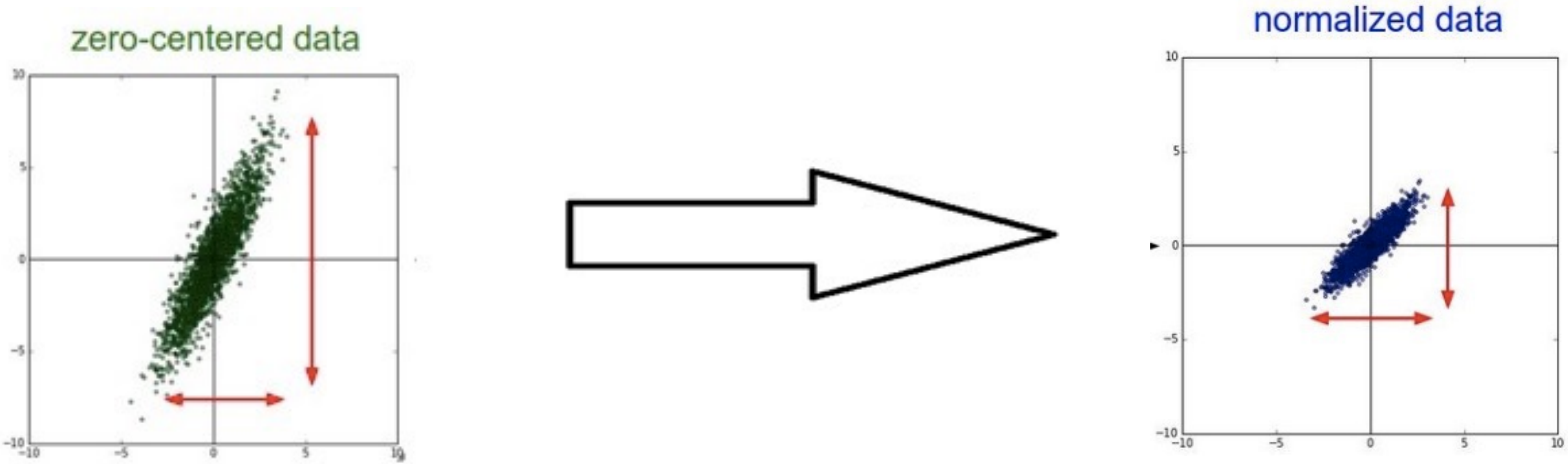
- 将原始像素值从[0, 255]调整为[-128, 127]

- 每个channel减去各自的平均
  - ✓e.g., VGGNet

- 每个channel减去各自的平均，再除以std
  - ✓e.g., ResNet

标准化

- 将数值压缩到一个较小的区间
  - ✓减小损失函数对权重参数变化的敏感度
  - ✓方便优化参数



```
X -= np.mean(X, axis = 0)
```

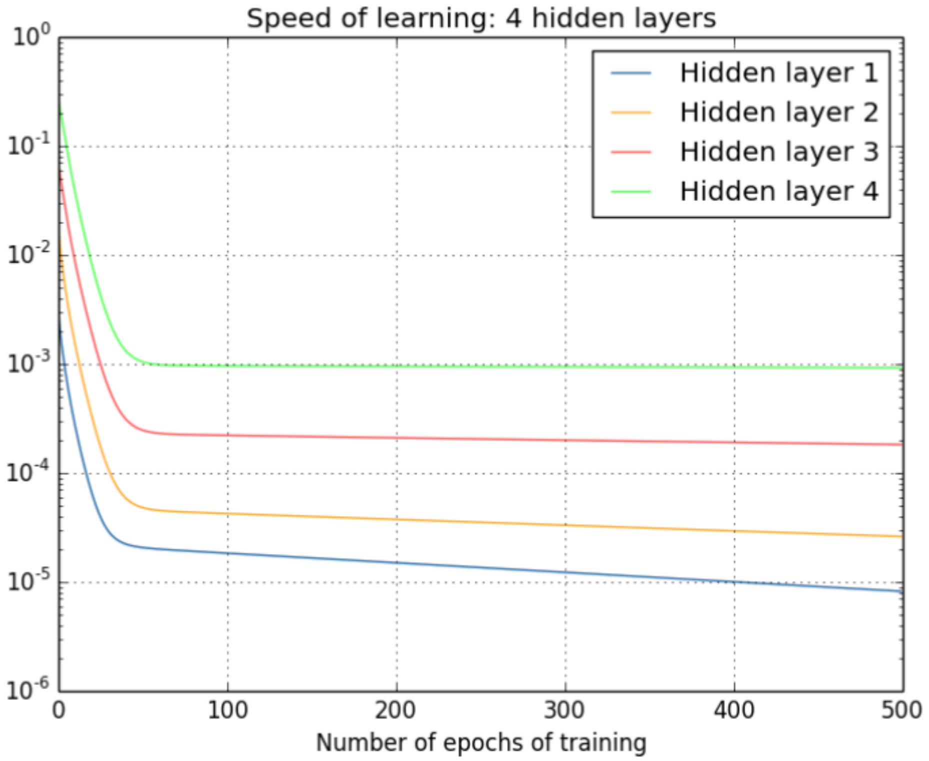
```
X /= np.std(X, axis = 0)
```

实际搭建模型的时候。。。

- 调整图像大小（Resize the images）
- 图像序列化（Pickle the images）
- 零均值化（Zero centering）
- 标准化（Normalization）： 一般先不做

权重参数的初始化

- 参数初始化对深度网络训练的影响



Chain rule

$$\frac{\partial L}{\partial \boldsymbol{W}^1} = \frac{\partial L}{\partial \boldsymbol{h}}\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{a}^l}\frac{\partial \boldsymbol{a}^l}{\partial \boldsymbol{a}^{l-1}}\frac{\partial \boldsymbol{a}^{l-1}}{\partial \boldsymbol{a}^{l-2}}\cdots\cdots\frac{\partial \boldsymbol{a}^2}{\partial \boldsymbol{W}^1}$$

参数初始化过小（≈ 0）

✓ 回传梯度快速接近0，梯度消失

靠近输入层的梯度无法更新

参数初始化过大（> 1）

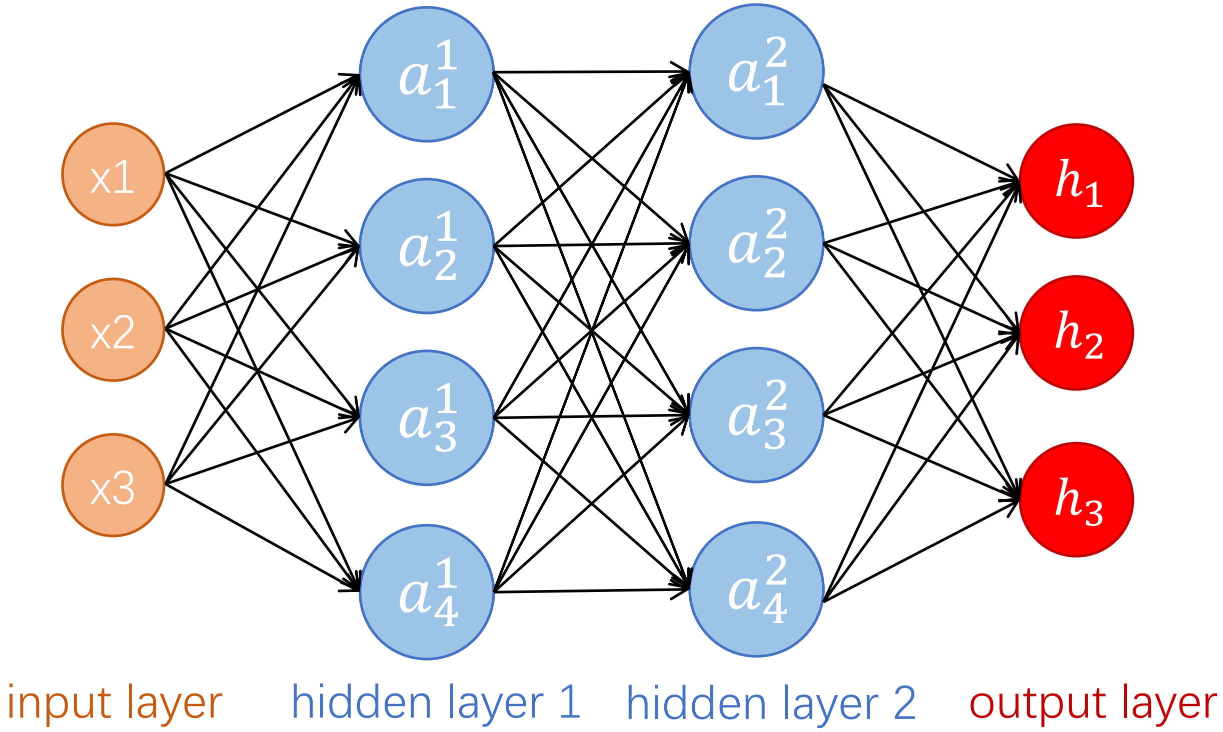
✓ 回传梯度快速增大，梯度爆炸

靠近输入层的梯度更新太快

参数初始化方法的演进

- 全部初始化为0
- 完全随机初始化
- Xavier初始化
- Kaiming/He/MSRA初始化

全部初始化为0



✓每一层的神经元输出完全一样

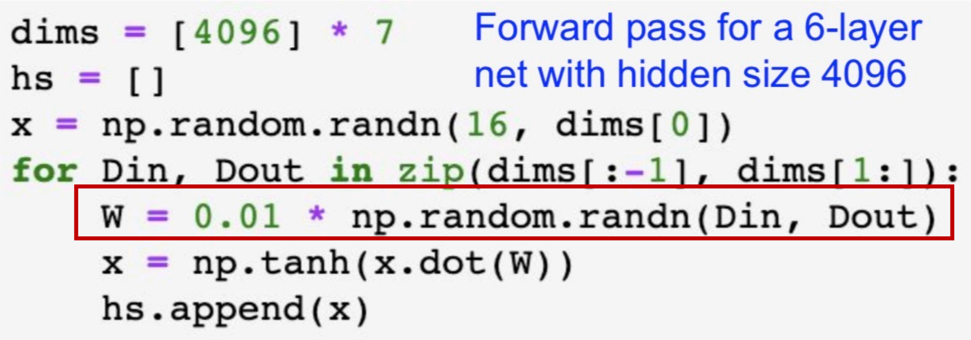
✓每一层的参数梯度完全一样

✓每一层的参数永远相同

✓无法学习数据特征

完全随机初始化

- 零均值，方差较小的正态分布随机数



```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = 0.01 * np.random.randn(Din, Dout)
    x = np.tanh(x.dot(W))
    hs.append(x)
```

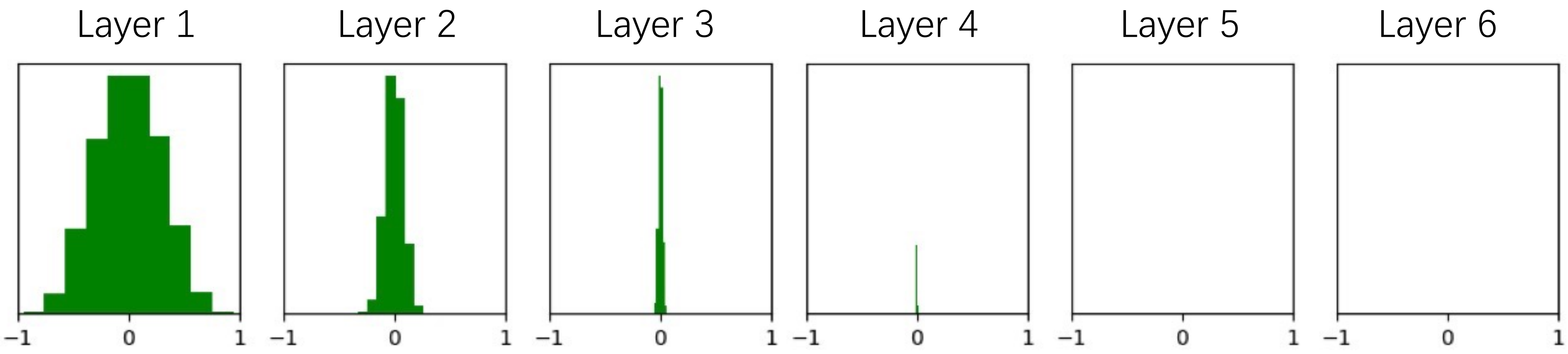
Forward pass for a 6-layer net with hidden size 4096

✓ 网络越深，所有激活越靠近0

✓ 越靠近输出层的梯度$\frac{\partial L}{\partial w}$越接近于0

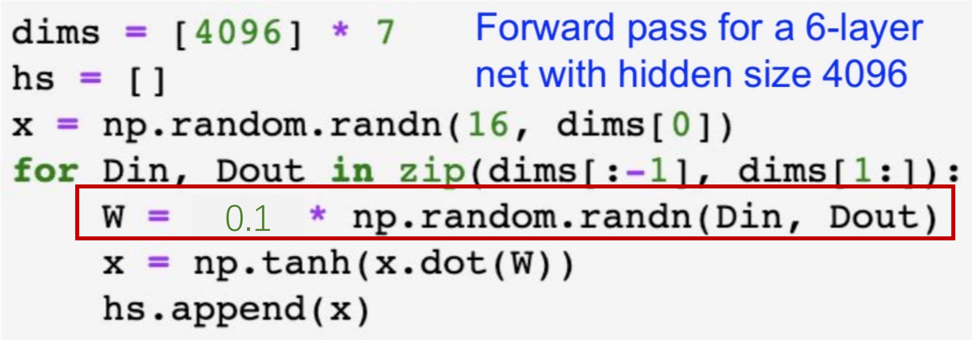
✓ 靠近输出层的w无法更新

每一层的激活



完全随机初始化

- 零均值，方差较大的正态分布随机数



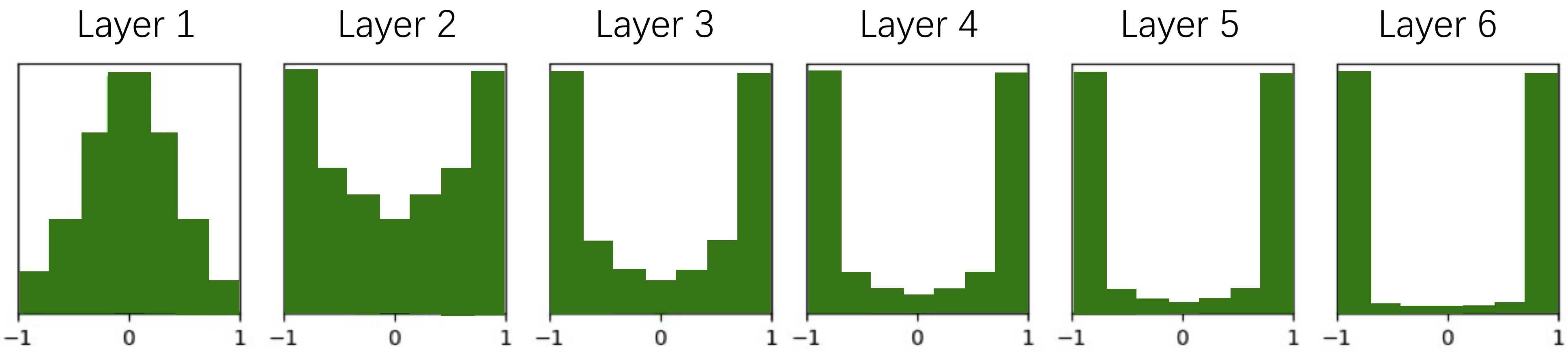
```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W =   0.1  * np.random.randn(Din, Dout)
    x = np.tanh(x.dot(W))
    hs.append(x)
```

Forward pass for a 6-layer net with hidden size 4096

✓ 网络越深，所有激活越饱和

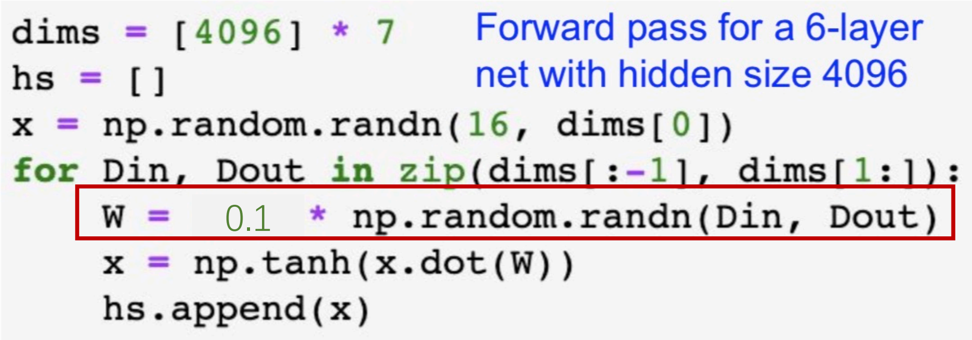
✓ 激活门的局部梯度接近于0，回传梯度消失

每一层的激活



完全随机初始化

- 零均值，方差较大的正态分布随机数



```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W =   0.1  * np.random.randn(Din, Dout)
    x = np.tanh(x.dot(W))
    hs.append(x)
```

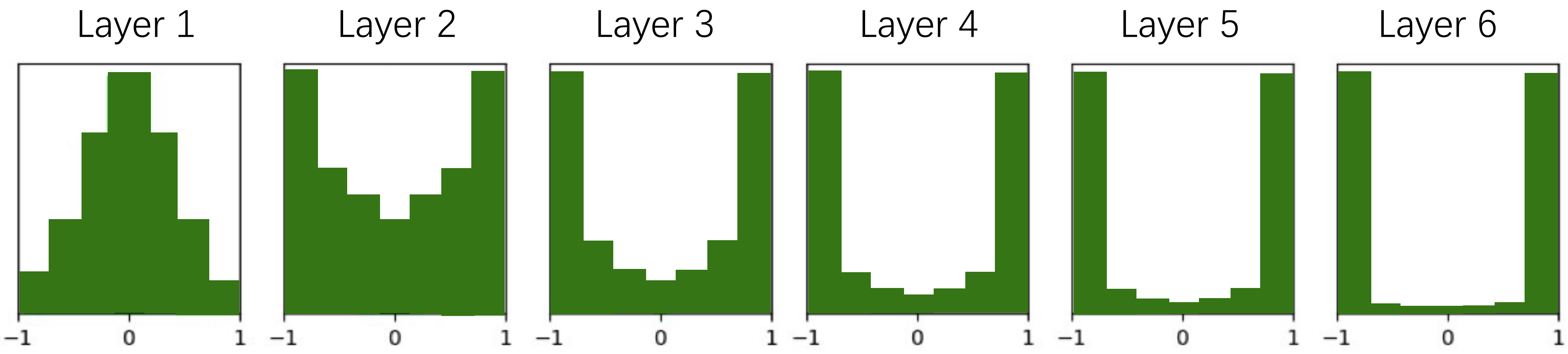
Forward pass for a 6-layer net with hidden size 4096

✓ 网络越深，所有激活越饱和

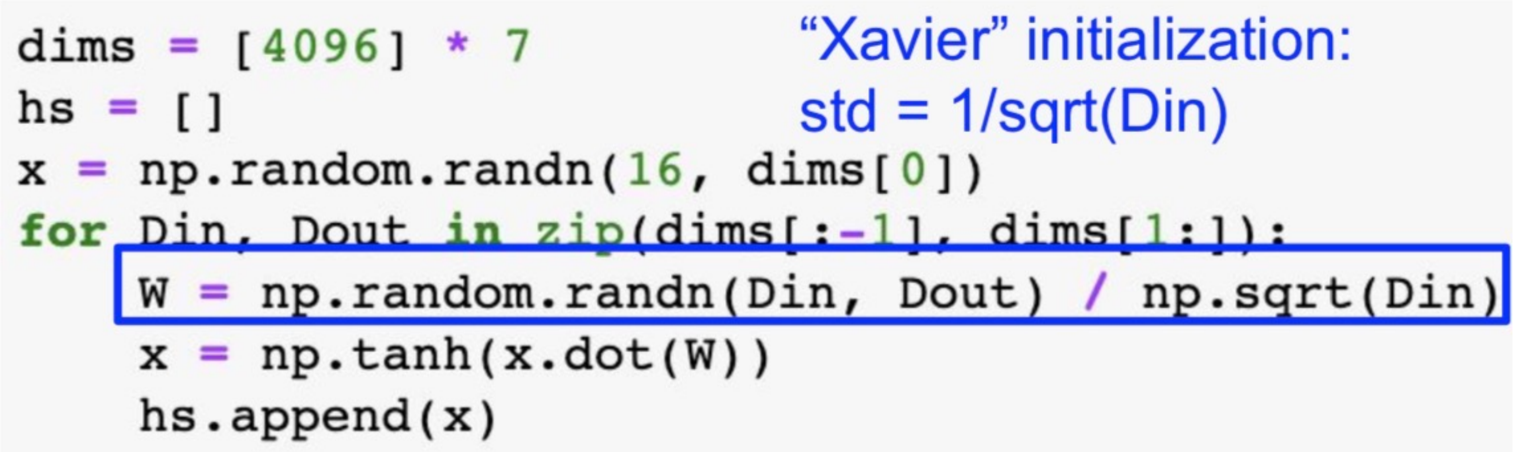
✓ 激活门的局部梯度接近于0，回传梯度消失

尽可能保持y和x的分布保持一致

每一层的激活



Xavier初始化 (Xavier Initialization)



```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = np.random.randn(Din, Dout) / np.sqrt(Din)
    x = np.tanh(x.dot(W))
    hs.append(x)
```

“Xavier” initialization:
std = 1/sqrt(Din)

$$\mathrm{Var}(y) = (n\mathrm{Var}(w))\mathrm{Var}(x)$$

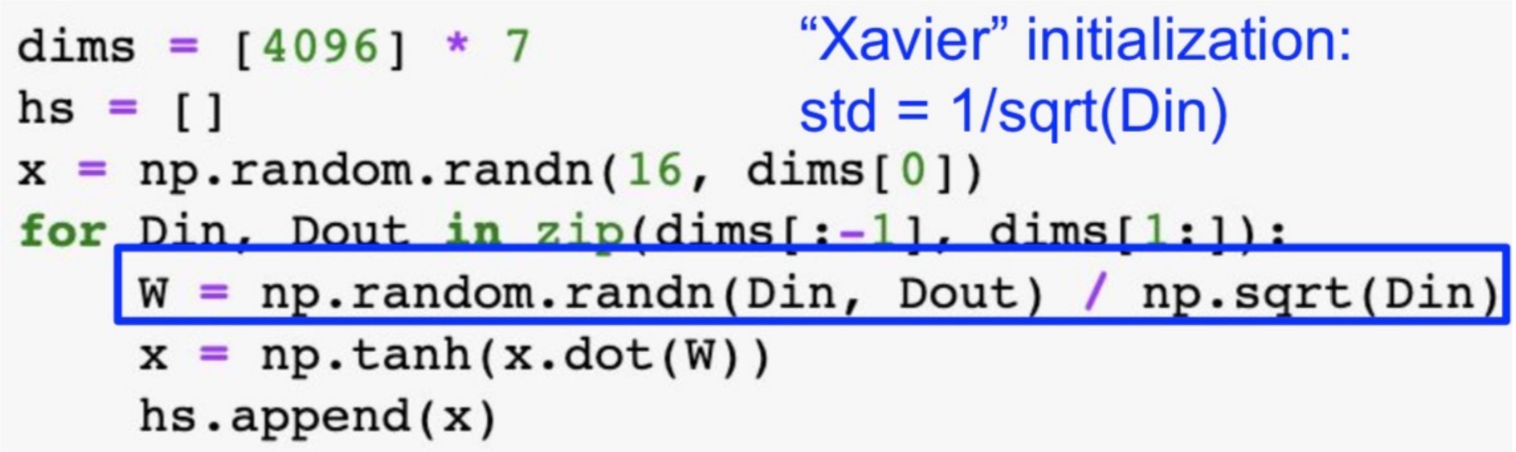
n为上一层的大小

https://cs231n.github.io/neural-networks-2/#init

$$W_{ij} \sim U\left[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\right]$$

Glorot X, Bengio Y. Understanding the difficulty of training deep feedforward neural networks. 2010.

Xavier初始化 (Xavier Initialization)

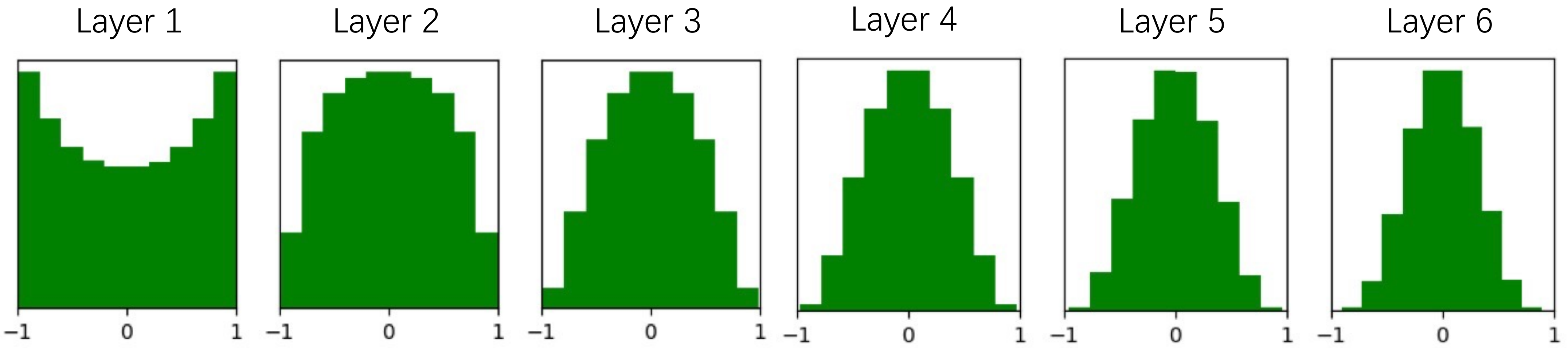


```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = np.random.randn(Din, Dout) / np.sqrt(Din)
    x = np.tanh(x.dot(W))
    hs.append(x)
```

“Xavier” initialization:
std = 1/sqrt(Din)

在卷积层中

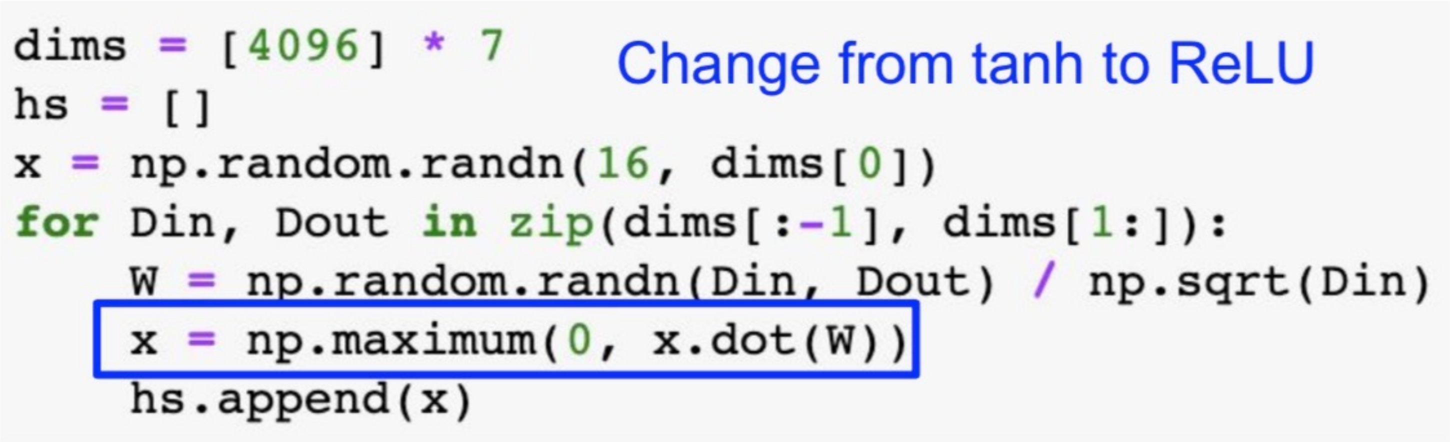
- ✓ Din=F^2×K
- ✓ F为filer大小， 输入K为信道数



Glorot X, Bengio Y. Understanding the difficulty of training deep feedforward neural networks. 2010.

Xavier初始化 (Xavier Initialization)

- 激活函数替换为ReLU

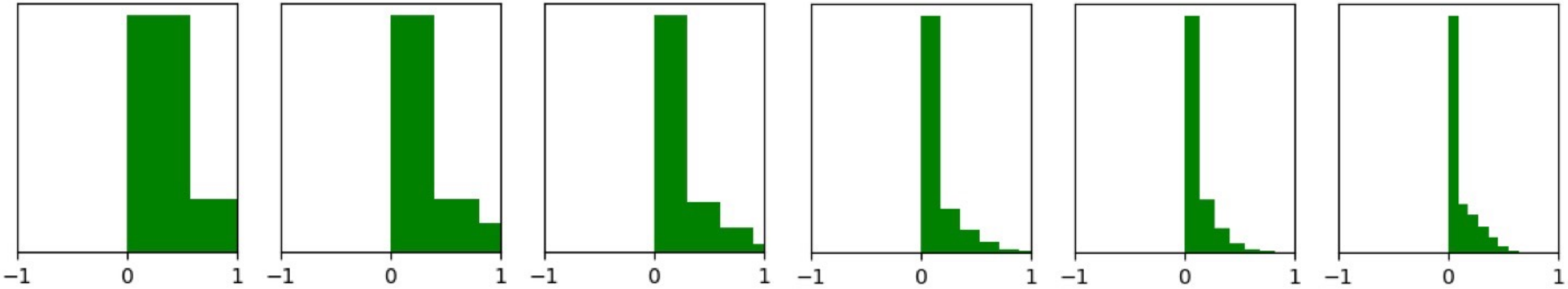


```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = np.random.randn(Din, Dout) / np.sqrt(Din)
    x = np.maximum(0, x.dot(W))
    hs.append(x)
```

Change from tanh to ReLU

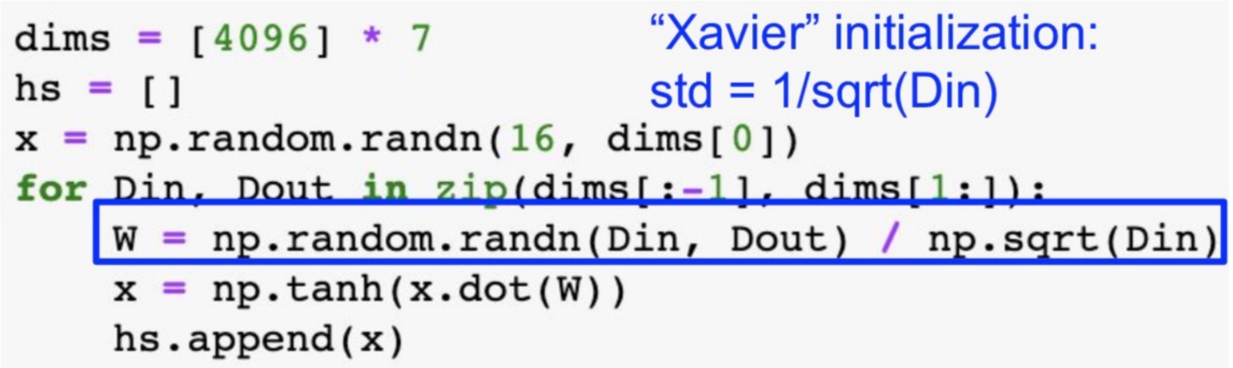
✓ 网络越深，所有激活越靠近0

✓ 靠近输出层参数无法更新



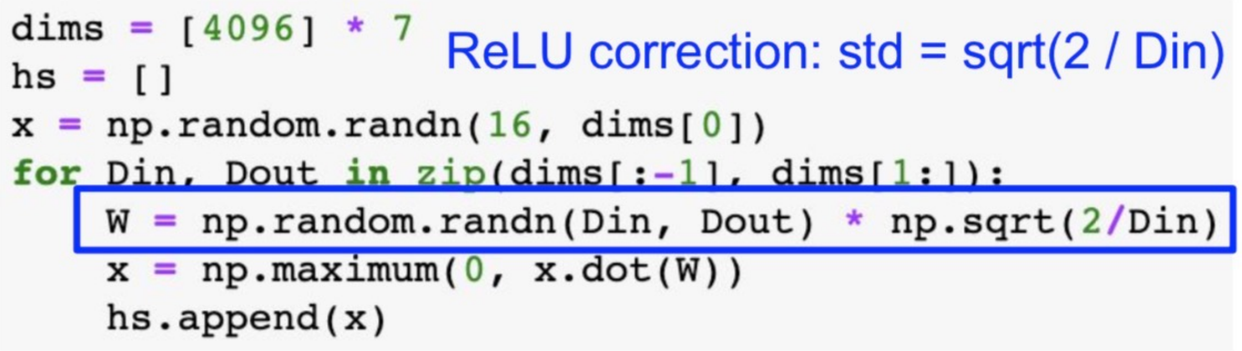
He初始化（He/Kaiming/MSRA Initialization）

- 激活函数替换为ReLU



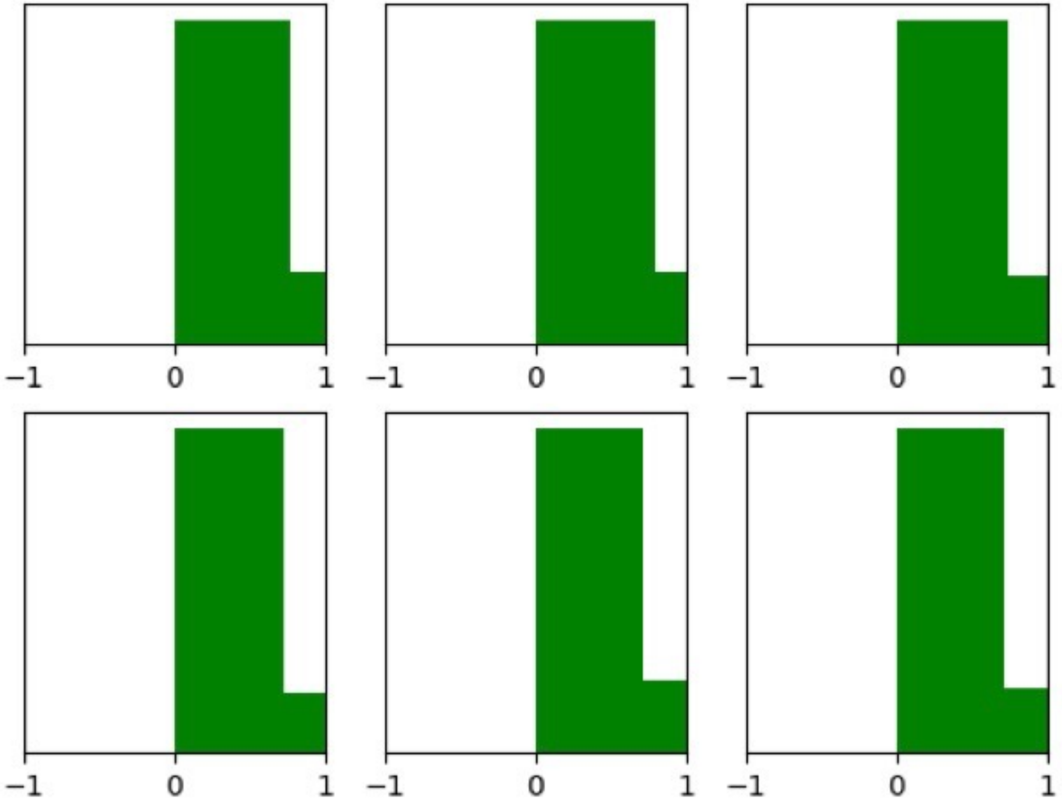
```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = np.random.randn(Din, Dout) / np.sqrt(Din)
    x = np.tanh(x.dot(W))
    hs.append(x)
```

“Xavier” initialization:
std = 1/sqrt(Din)



```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = np.random.randn(Din, Dout) * np.sqrt(2/Din)
    x = np.maximum(0, x.dot(W))
    hs.append(x)
```

ReLU correction: std = sqrt(2 / Din)



He K, Zhang X, Ren S, Sun J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. 2015.

实际搭建模型的时候。。。

- 优先使用ReLU+He初始化

Understanding the difficulty of training deep feedforward neural networks
by Glorot and Bengio, 2010

Exact solutions to the nonlinear dynamics of learning in deep linear neural networks by Saxe et al, 2013

Random walk initialization for training very deep feedforward networks by Sussillo and Abbott, 2014

Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification by He et al., 2015

Data-dependent Initializations of Convolutional Neural Networks by Krähenbühl et al., 2015

All you need is a good init, Mishkin and Matas, 2015

Fixup Initialization: Residual Learning Without Normalization, Zhang et al, 2019

The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks, Frankle and Carbin, 2019

Batch Normalization

- Internal covariate shift
  - ✓内部输出的分布由于参数变化而变化
  - ✓导致激活容易饱和或者趋近0
  - ✓神经网络训练不易收敛

Batch Normalization

- Internal covariate shift
  - ✓内部输出的分布由于参数变化而变化
  - ✓导致激活容易饱和或者趋近0
  - ✓神经网络训练不易收敛

- 对神经元输出特征的每一维单独做normalization

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

Ioffe S, Szegedy C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. 2015

Batch Normalization

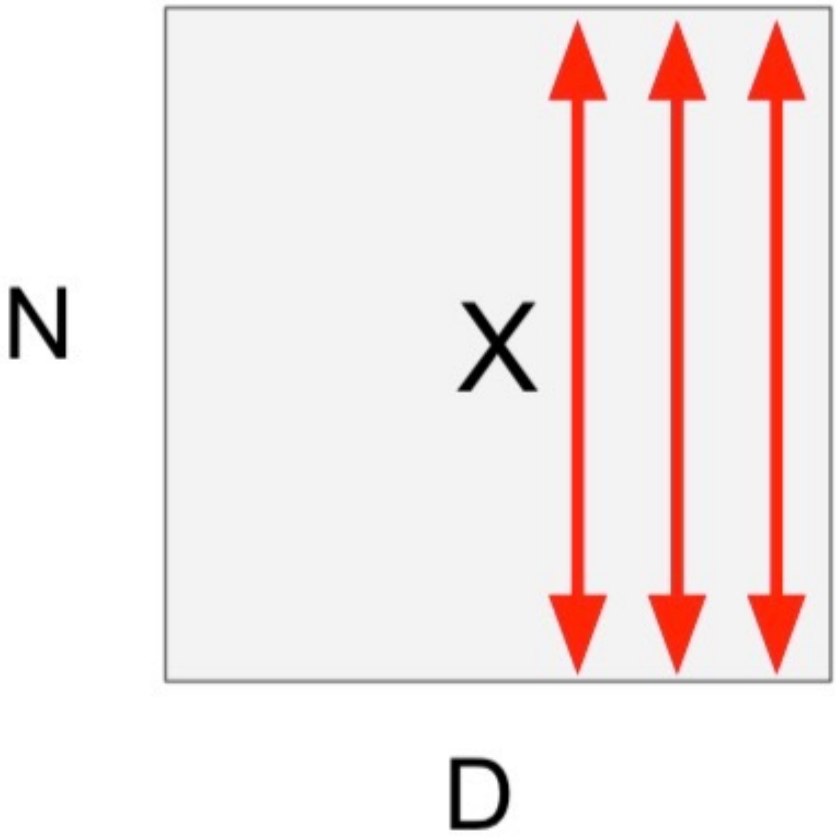
输入X： N×D

求batch中每一维特征的平均： $\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$ 1×D

求batch中每一维特征的方差： $\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{i,j} - \mu_j)^2$ 1×D

对每个x做normalization： $\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$ N×D

常量



强行标准化到$\mathcal{N}(0,1)$可能导致某些特征发生改变

Batch Normalization

求batch中每一维特征的平均： $\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$ 1×D

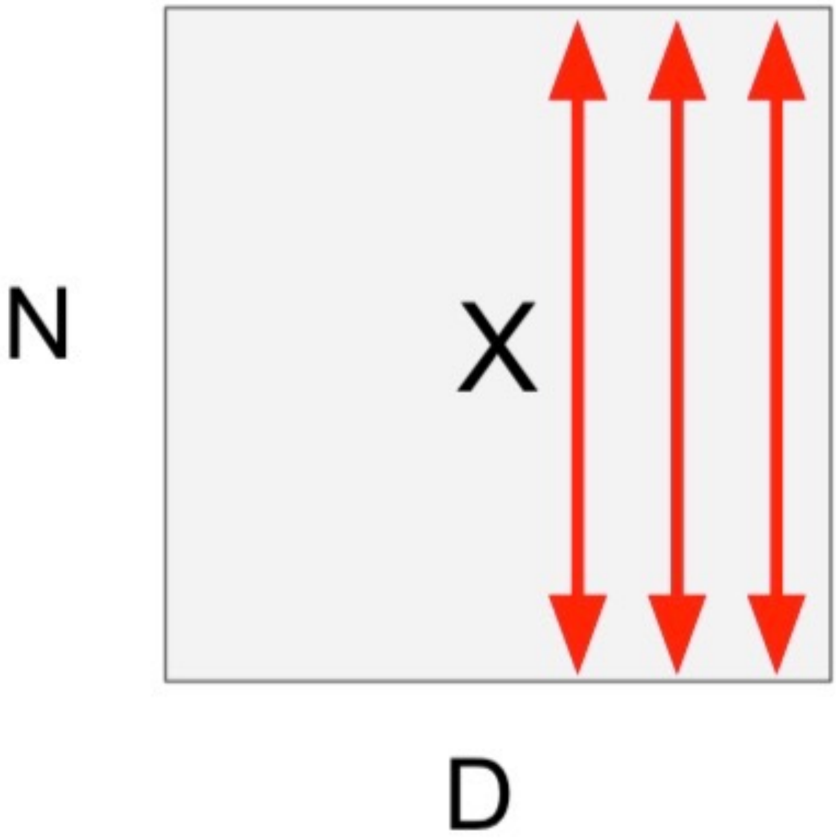
求batch中每一维特征的方差： $\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{i,j} - \mu_j)^2$ 1×D

对每个x做normalization： $\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$ N×D

引入参数γ和β，尝试还原X： $y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$ N×D

强行标准化到$\mathcal{N}(0,1)$可能导致某些特征发生改变

输入X： N×D



Batch Normalization： inference/testing

对每个x做normalization：

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

N×D

引入参数γ和β，尝试还原X：

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

N×D

推理时可能只有一个或者几个样本，无法有效计算μ和σ^2

希望使用固定的μ和σ^2

使用训练过程中保存的μ和σ^2来估计真实值

Batch Normalization： inference/testing

使用训练时μ的平均来估计μ_j：

$$\mu_j = \mathbb{E}(\mu_j^{(b)})$$

1×D

使用训练时σ^2的平均来估计σ_j^2：

$$\sigma_j^2 = \frac{N}{N-1}\mathbb{E}(\sigma_j^{2(b)})$$

1×D

对每个x做normalization：

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

N×D

引入参数γ和β，尝试还原X：

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

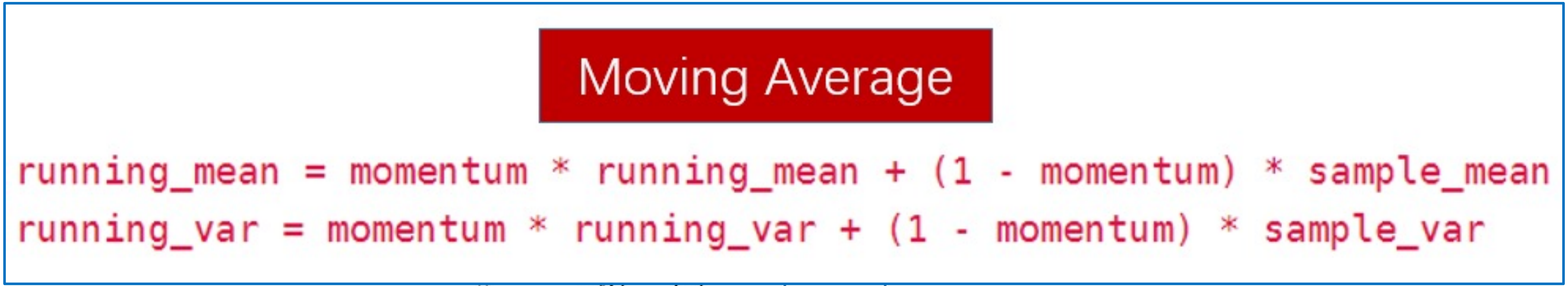
N×D

推理时可能只有一个或者几个样本，无法有效计算μ和σ^2

希望使用固定的μ和σ^2

使用训练过程中保存的μ和σ^2来估计真实值

Batch Normalization： inference/testing



Moving Average

```
running_mean = momentum * running_mean + (1 - momentum) * sample_mean
running_var = momentum * running_var + (1 - momentum) * sample_var
```

对每个x做normalization：

$$x_{i,j} = \frac{}{\sqrt{\sigma_j^2 + \varepsilon}}$$

N×D

引入参数γ和β，尝试还原X：

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

N×D

使用训练过程中保存的μ和σ^2来估计真实值

Batch Normalization： inference/testing

使用训练时 μ 的平均来估计 μ_j ： $$\mu_j = \mathbb{E}(\mu_j^{(b)})$$ 1×D

使用训练时σ^2的平均来估计σ_j^2 ： $$\sigma_j^2 = \frac{N}{N-1}\mathbb{E}(\sigma_j^{2(b)})$$ 1×D

对每个x做normalization： $$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$ N×D

引入参数γ和β，尝试还原X： $$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$ N×D

融合成线性结构

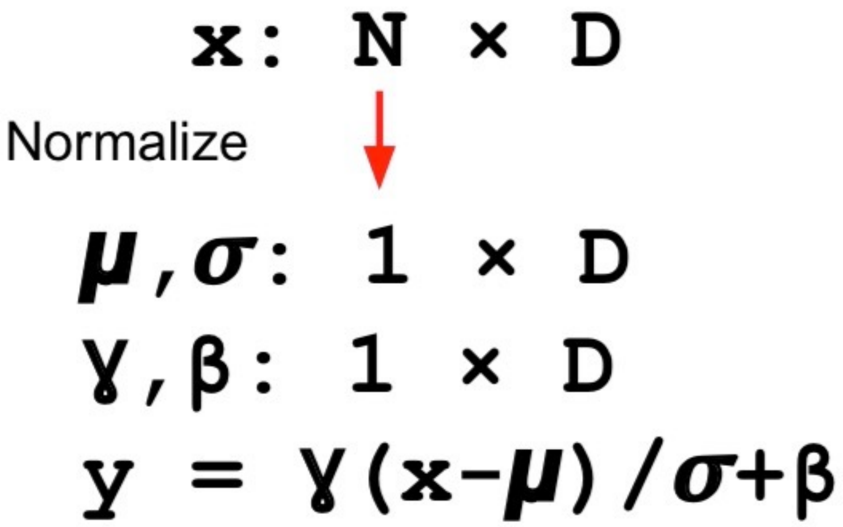
推理时可能只有一个或者几个样本，无法有效计算μ和σ^2

希望使用固定的μ和σ^2

使用训练过程中保存的μ和σ^2来估计真实值

矩阵运算视角

全连接层的
batch normalization



$\mathbf{x}: N \times D$

Normalize

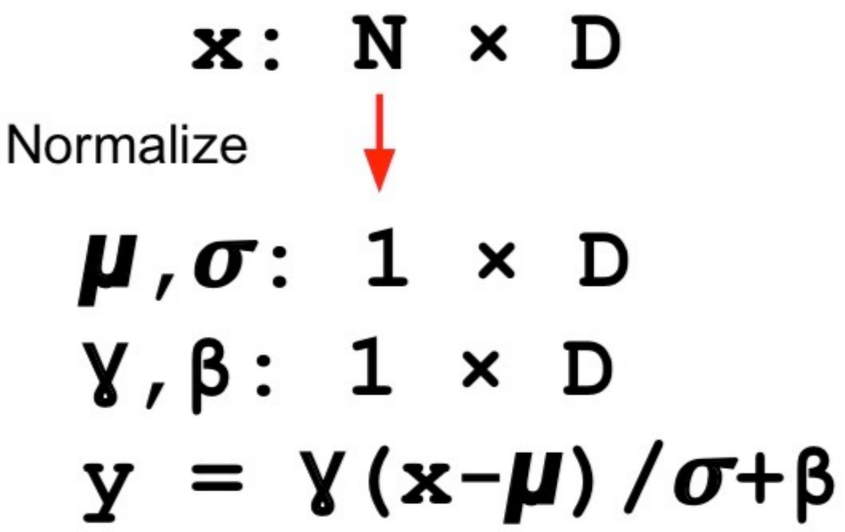
$\boldsymbol{\mu}, \boldsymbol{\sigma}: 1 \times D$

$\gamma, \beta: 1 \times D$

$y = \gamma(\mathbf{x}-\boldsymbol{\mu})/\boldsymbol{\sigma}+\beta$

矩阵运算视角

全连接层的
batch normalization



$x: N \times D$

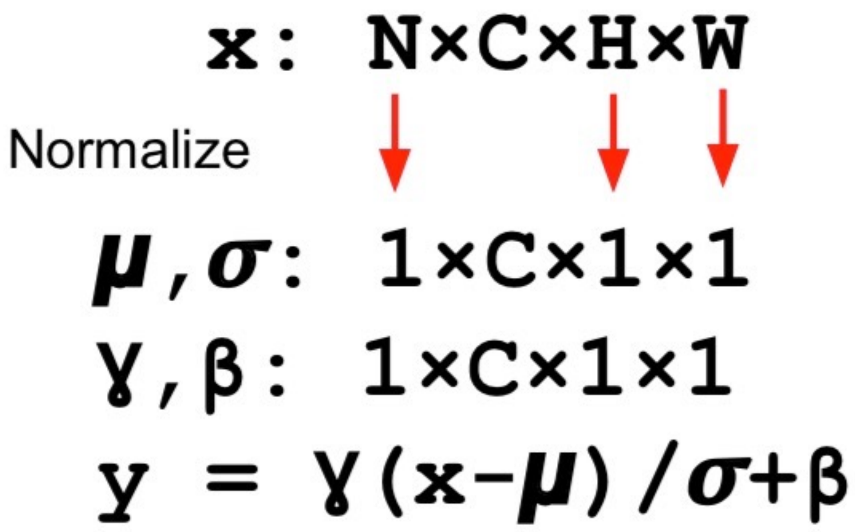
Normalize

$\mu, \sigma: 1 \times D$

$\gamma, \beta: 1 \times D$

$y = \gamma(x-\mu)/\sigma+\beta$

卷积层的
batch normalization



$x: N\times C\times H\times W$

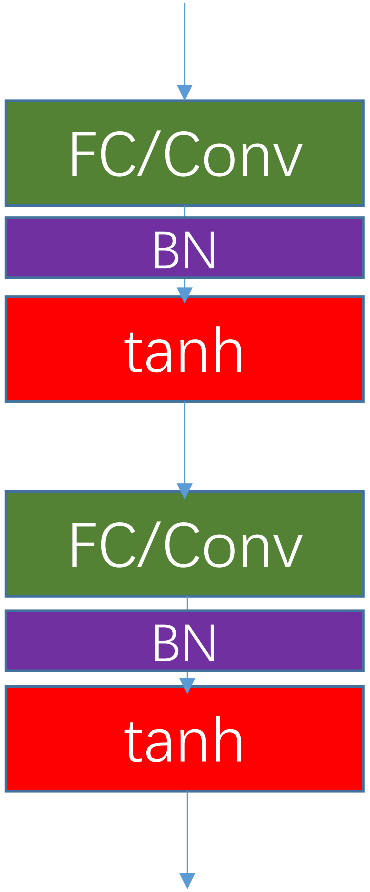
Normalize

$\mu, \sigma: 1\times C\times 1\times 1$

$\gamma, \beta: 1\times C\times 1\times 1$

$y = \gamma(x-\mu)/\sigma+\beta$

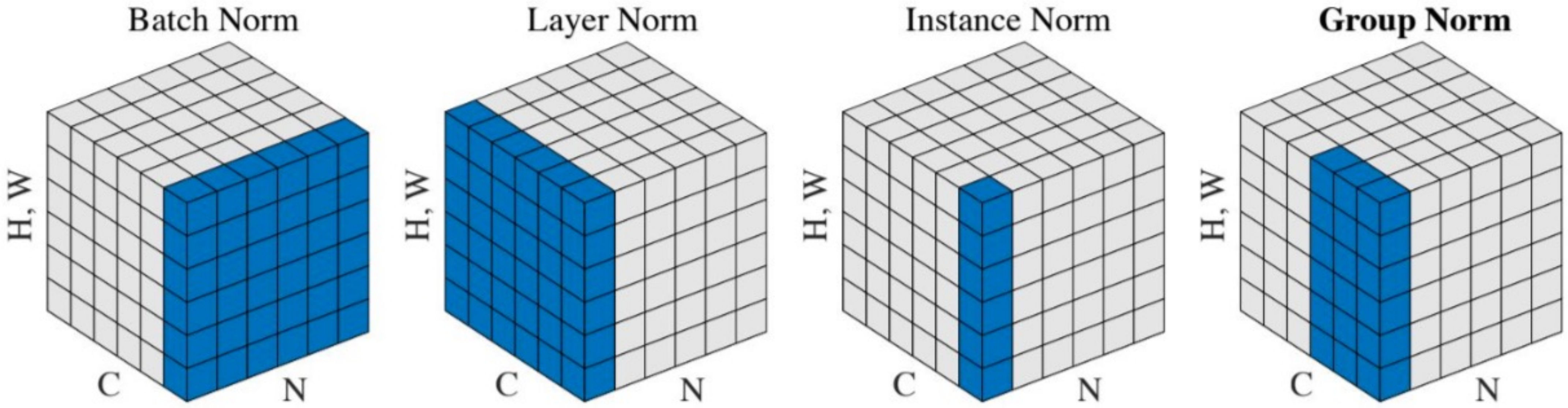
在神经网络中是BN



- ✓ 优化了梯度流，使深度网络训练起来更加容易
- ✓ 可以使用较大learning rate，加速收敛
- ✓ 受权重参数初始化影响较小
- ✓ 在训练过程中起到正则化的作用
- ✓ 推理时和FC/Conv层融合，几乎不增加开销

注意训练和推理BN的具体实现不同！

Normalization的变化



Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. ICML 2015.
Ba, J.L., Kiros, J.R. and Hinton, G.E. Layer normalization. 2016.
Ulyanov, D., Vedaldi, A. and Lempitsky, V. Instance normalization: The missing ingredient for fast stylization. 2016.
Wu, Y. and He, K. Group normalization. ECCV 2018.

小结

- CNN的应用
  ✓风格迁移、下棋、图像生成、看图说话

- 激活函数的选择
- 数据预处理
- 权重参数初始化
- Batch Normalization及其演化

L07

- 神经网络的训练细节2