深度学习平台与应用

第七讲：训练神经网络

感受野

https://distill.pub/2019/computing-receptive-fields/#return-from-solving-receptive-field-size

- 卷积网络的感受野：
  - 对当前输入的感受野：只与滤波器的大小有关，与 padding、stride 等无关
  - 对网络输入的感受野：

$$r_0 = \sum_{l=1}^{L} \left((k_l - 1) \prod_{i=1}^{l-1} s_i \right) + 1$$

感受野

https://distill.pub/2019/computing-receptive-fields/#return-from-solving-receptive-field-size

$$r_0 = \sum_{l=1}^{L}\left((k_l - 1)\prod_{i=1}^{l-1} s_i\right) + 1$$

AlexNet 结构:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

- **Conv 层对网络输入的感受野?**
- **Pooling 层对网络输入的感受野?**
- **FC 层对网络输入的感受野?**

输出特征大小

- 输出特征大小：**向下取整**

Shape:

- Input: $(N, C_{in}, H_{in}, W_{in})$ or (C_{in}, H_{in}, W_{in})
- Output: $(N, C_{out}, H_{out}, W_{out})$ or $(C_{out}, H_{out}, W_{out})$, where

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times \text{padding}[0] - \text{dilation}[0] \times (\text{kernel\_size}[0] - 1) - 1}{\text{stride}[0]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times \text{padding}[1] - \text{dilation}[1] \times (\text{kernel\_size}[1] - 1) - 1}{\text{stride}[1]} + 1 \right\rfloor$$

Code Blame 277 lines (277 loc) · 3.54 KB

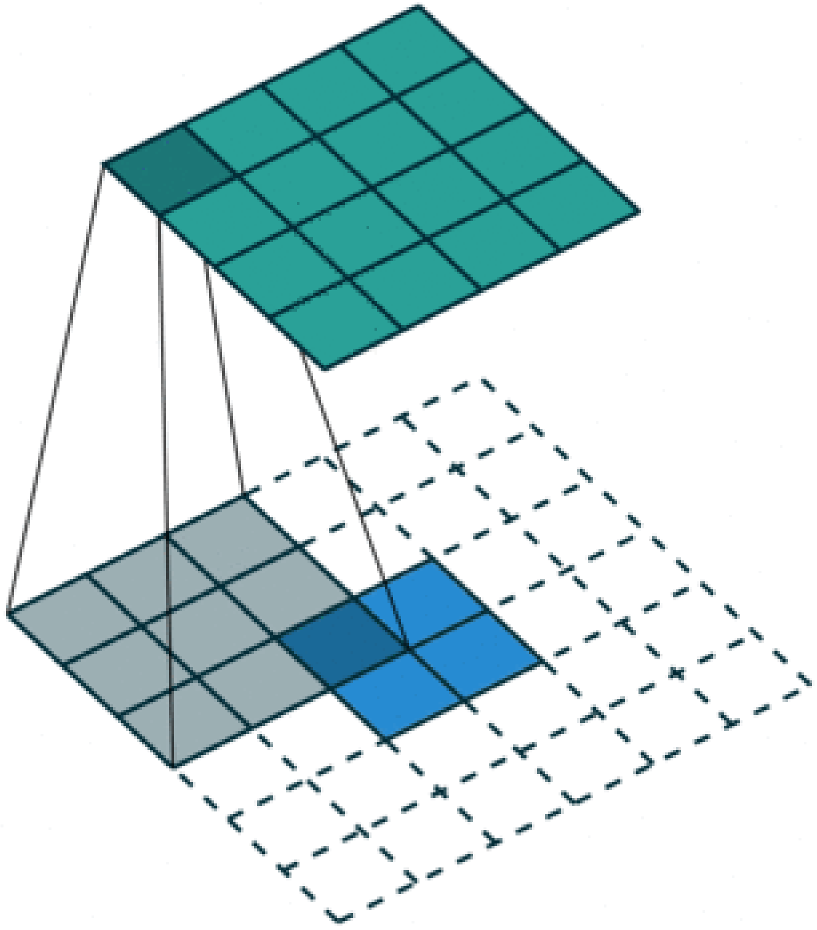
```
name: "AlexNet"
layer {
  name: "data"
  type: "Input"
  top: "data"
  input_param { shape: { dim: 10 dim: 3 dim: 227 dim: 227 } }
}
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 96
    kernel_size: 11
    stride: 4
  }
}
layer {
  name: "relu1"
  type: "ReLU"
  bottom: "conv1"
  top: "conv1"
}
layer {
  name: "norm1"
  type: "LRN"
  bottom: "conv1"
```

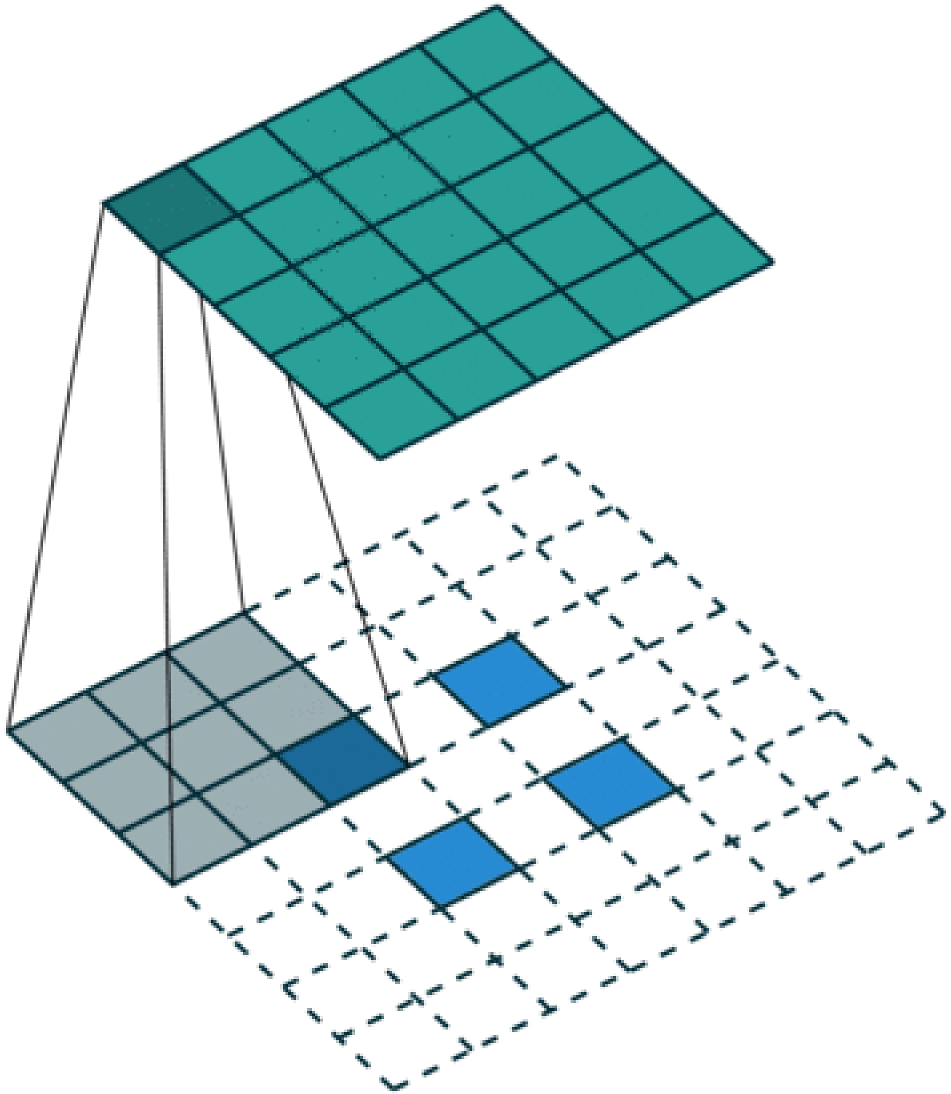
```python
class AlexNet(nn.Module):
    def __init__(self, num_classes: int = 1000, dropout: float = 0.5) -> None:
        super().__init__()
        _log_api_usage_once(self)
        self.features = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=11, stride=4, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(64, 192, kernel_size=5, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(192, 384, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(384, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
        )
        self.avgpool = nn.AdaptiveAvgPool2d((6, 6))
        self.classifier = nn.Sequential(
            nn.Dropout(p=dropout),
            nn.Linear(256 * 6 * 6, 4096),
            nn.ReLU(inplace=True),
            nn.Dropout(p=dropout),
            nn.Linear(4096, 4096),
            nn.ReLU(inplace=True),
            nn.Linear(4096, num_classes),
        )

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        x = self.features(x)
        x = self.avgpool(x)
        x = torch.flatten(x, 1)
        x = self.classifier(x)
```

转置卷积

https://paddlepedia.readthedocs.io/en/latest/tutorials/CNN/convolution_operator/Transpose_Convolution.html

转置卷积





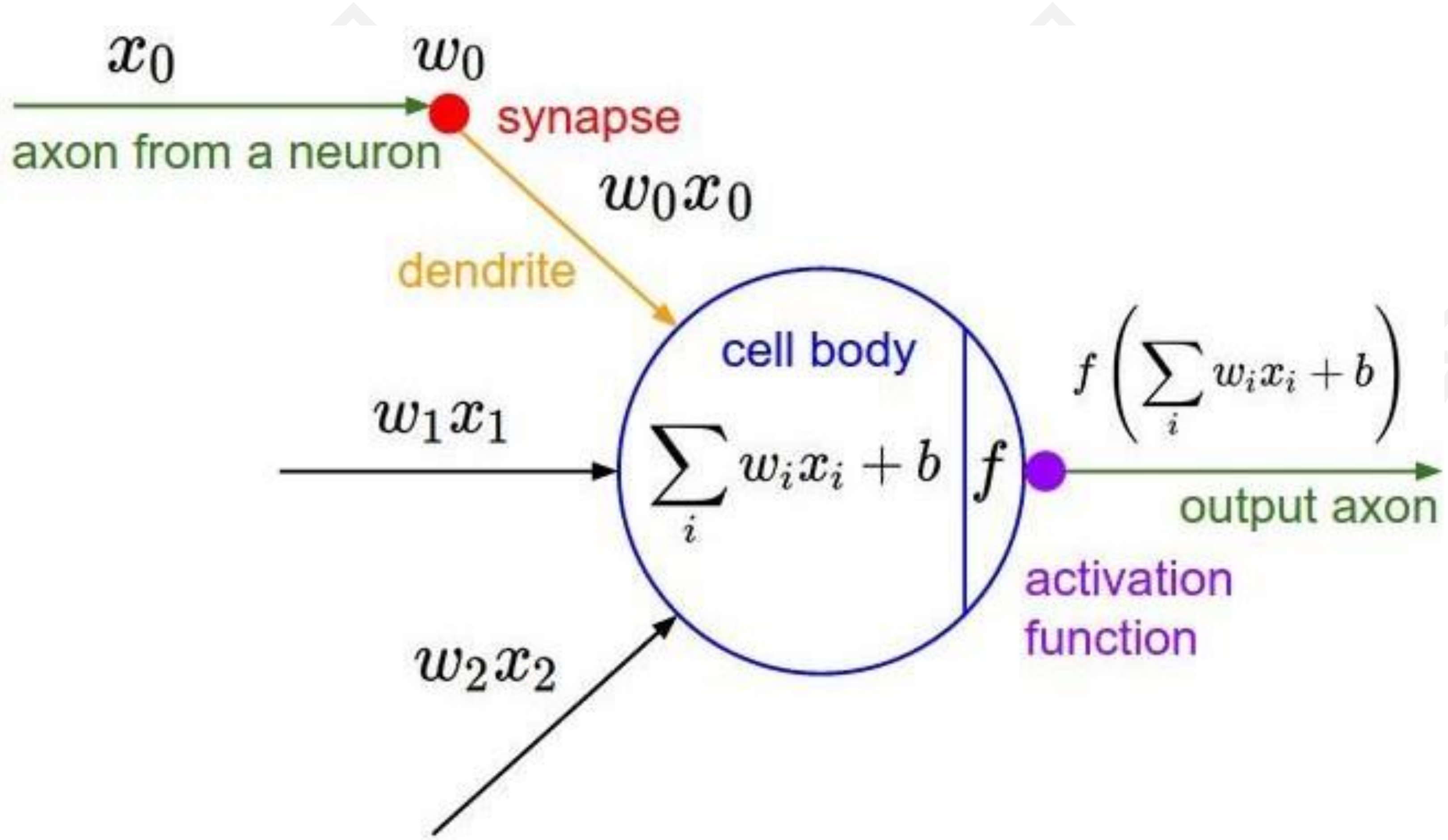
课程回顾

- 卷积神经网络架构（第五、六讲）
- 正则化与优化（SGD，mini batch）（第三讲）
- 反向传播（第四讲）
- 本节课：如何训练卷积神经网络

大　　　纲

- **激活函数**
- **数据预处理**
- **权重初始化**
- **正则化**
- **超参数选择**

激活函数

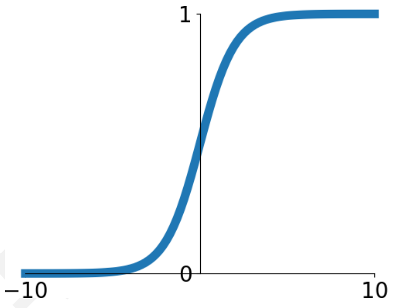


x_0
w_0
synapse
axon from a neuron
$w_0 x_0$
dendrite
cell body
$w_1 x_1$
$\sum_i w_i x_i + b$
f
$f\left(\sum_i w_i x_i + b\right)$
output axon
activation
function
$w_2 x_2$

激活函数

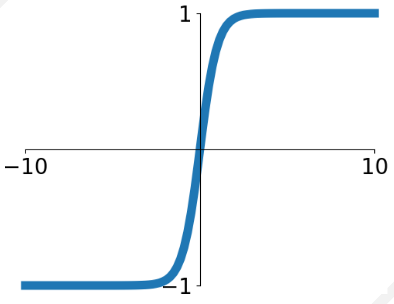
Sigmoid

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



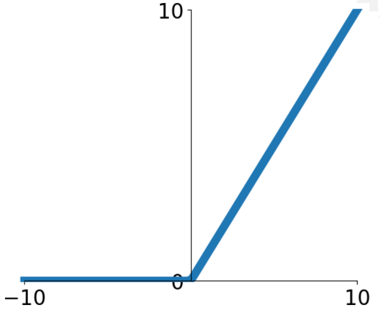
tanh

$$\tanh(x)$$



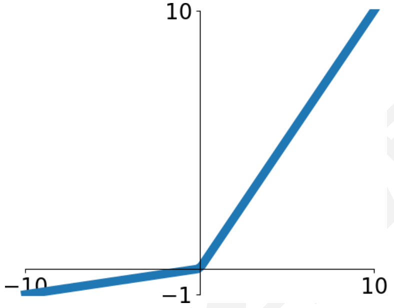
ReLU

$$\max(0, x)$$



Leaky ReLU

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

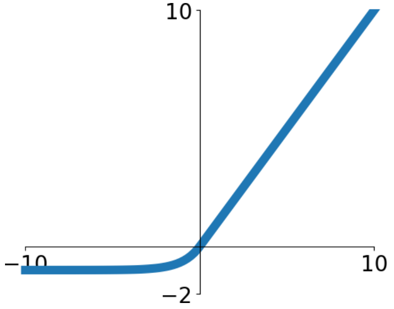


Maxout

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

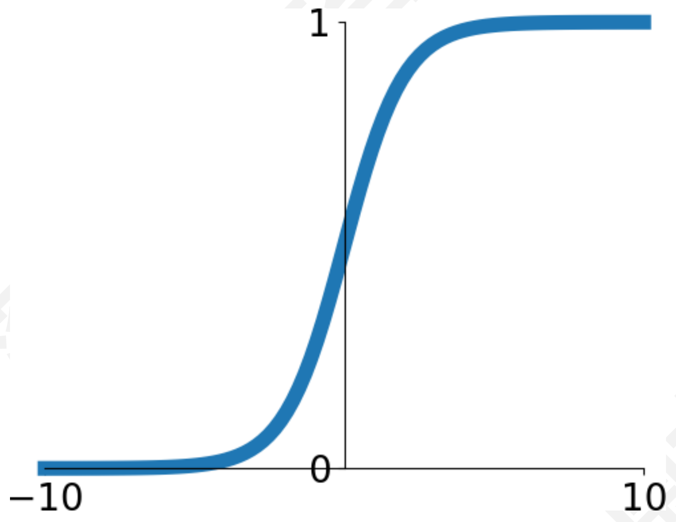
ELU

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



激活函数

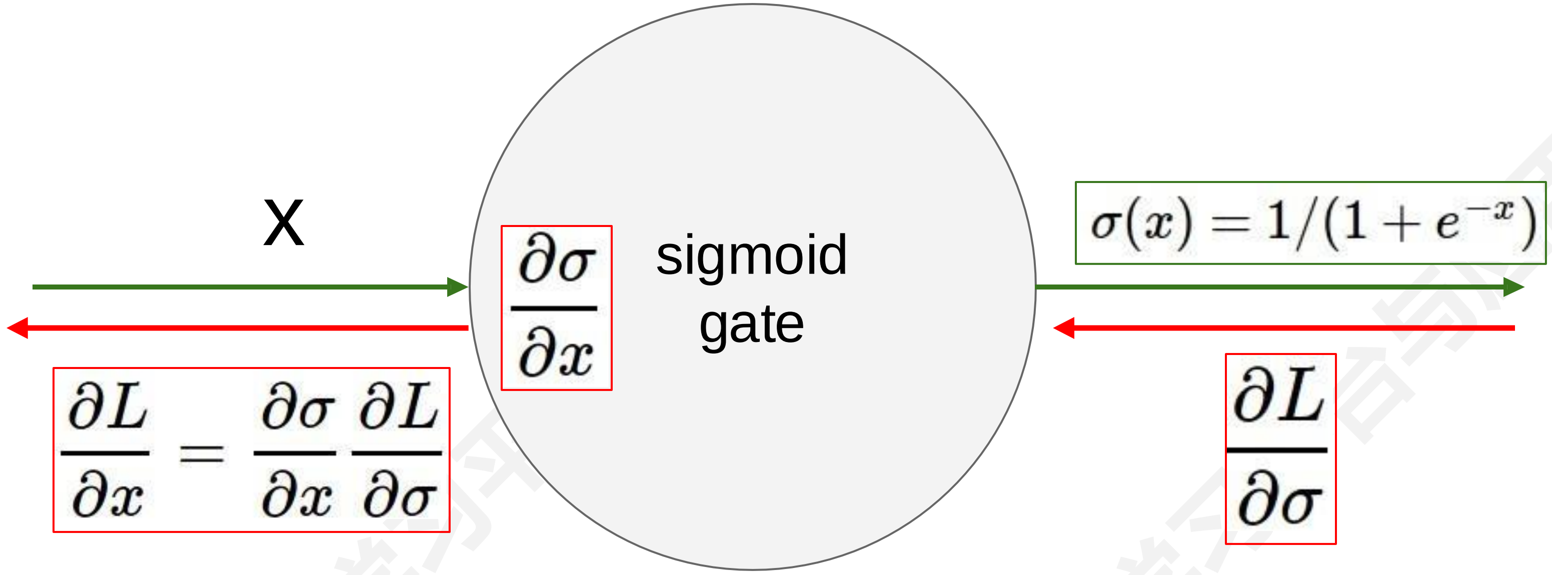
$$\sigma(x) = 1/(1+e^{-x})$$

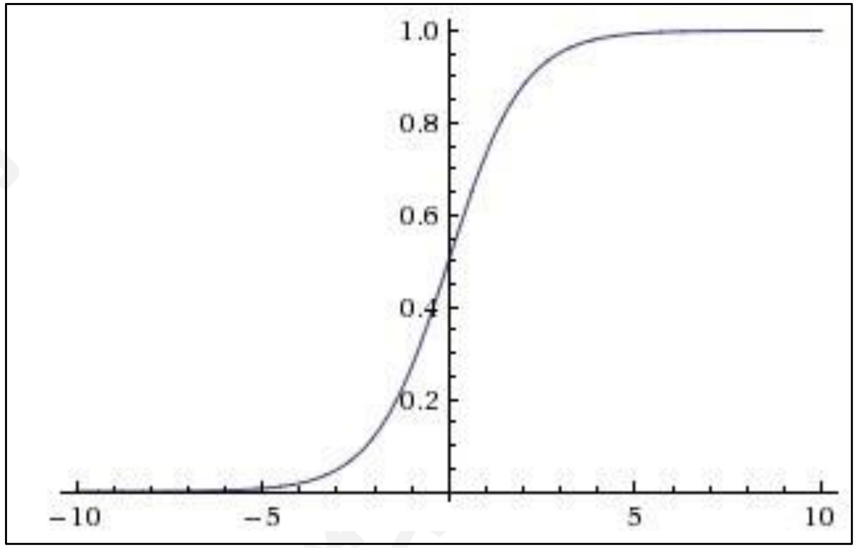


Sigmoid

- 将输出压缩到[0,1]
- 历史上很受欢迎，因为可以很好建模真实的生物神经元
- 问题：
  - 梯度消失问题（饱和时）
  - 输出不是以 0 为中心
  - 指数函数的计算成本高

激活函数

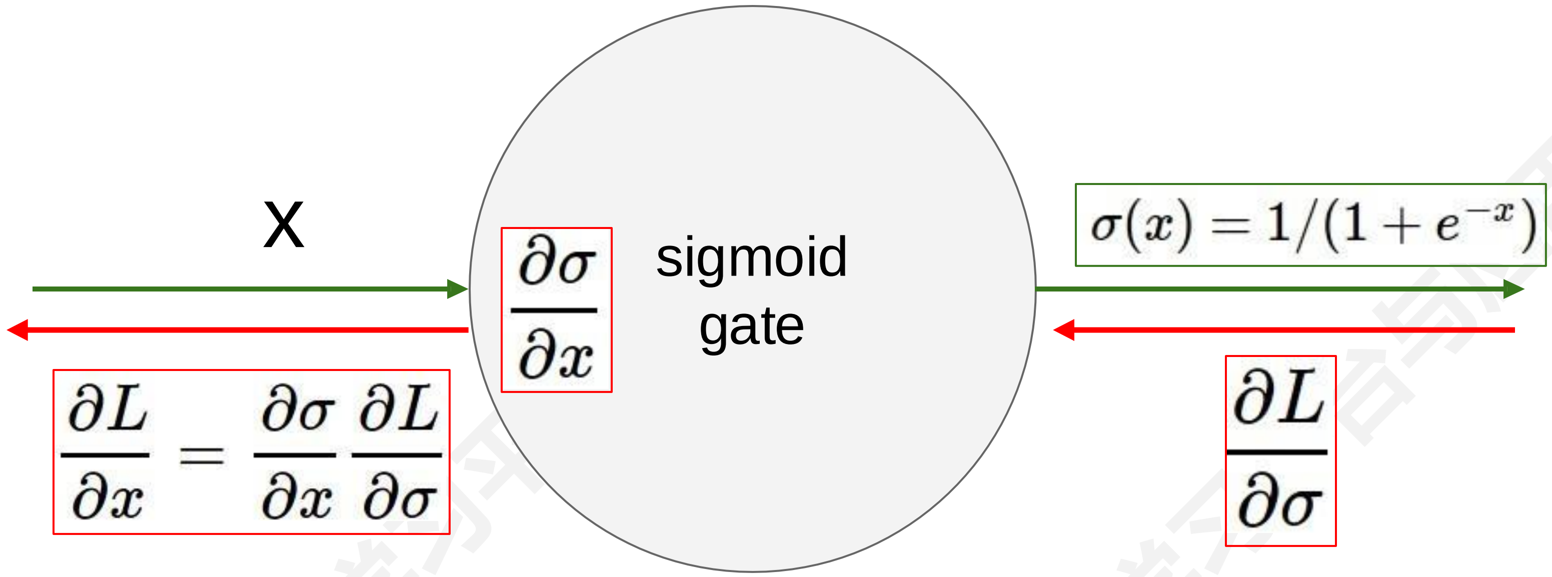


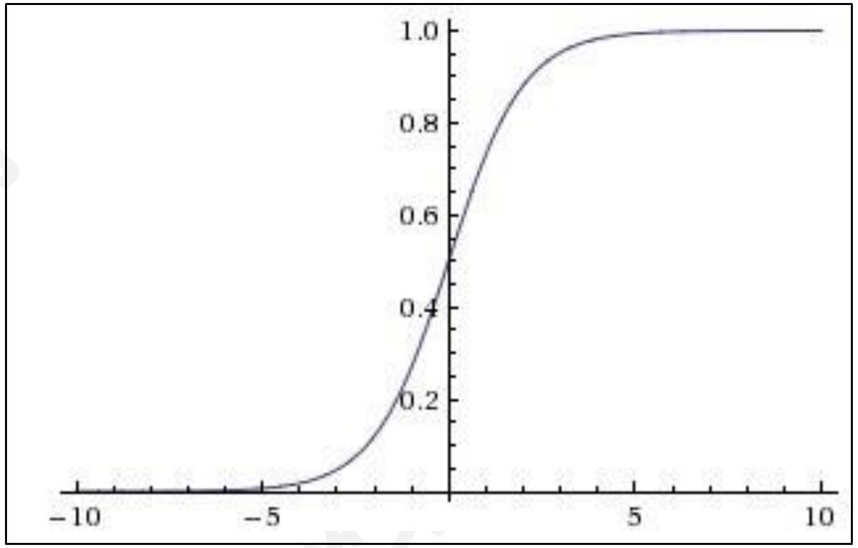


- **x = -10?** $\sigma(x) = \sim 0$
- **x = 0?**
- **x = 10?** $\sigma(x) = \sim 0$

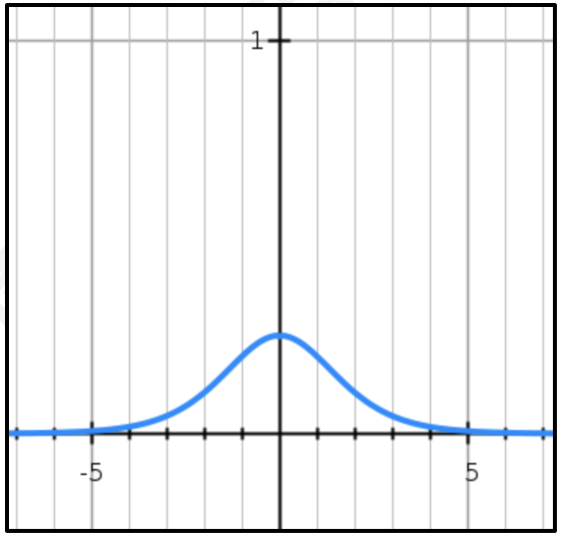
$$\frac{\partial \sigma(x)}{\partial x} = \sigma(x)\,(1 - \sigma(x))$$

激活函数



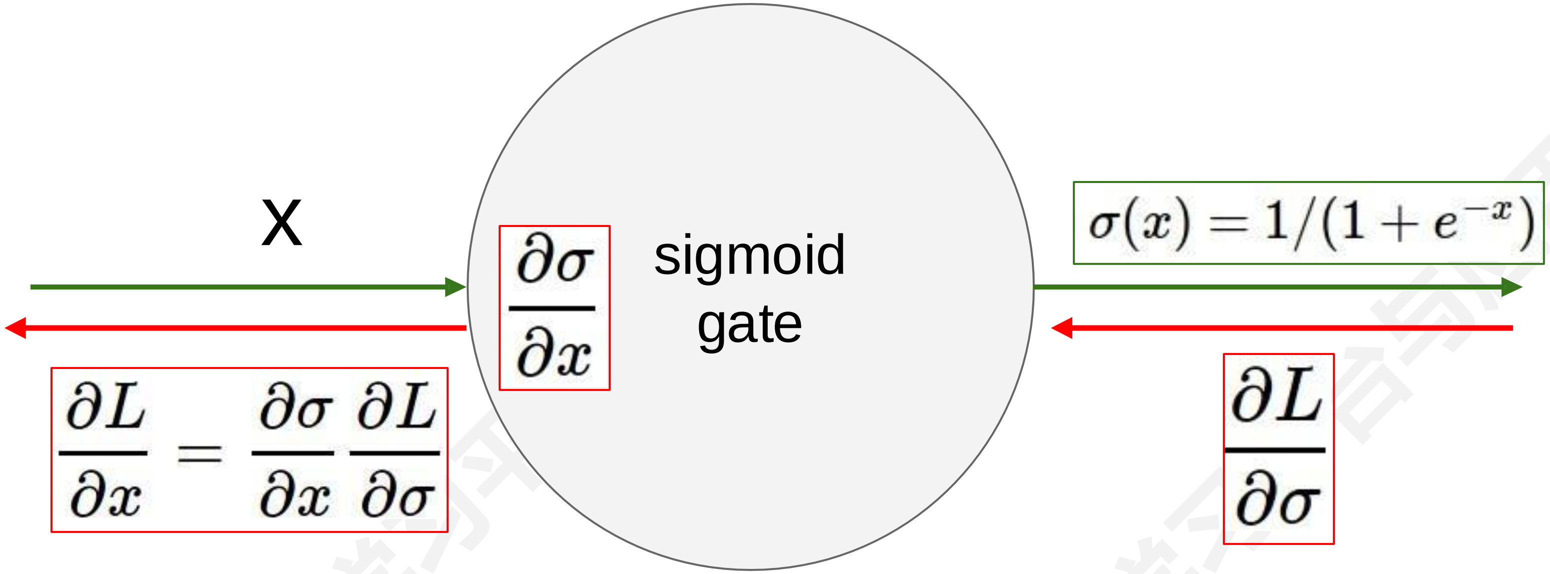


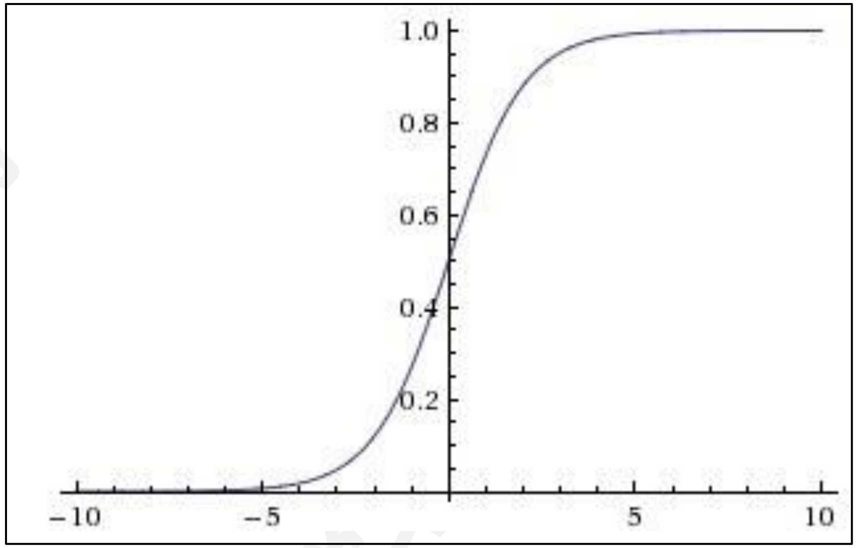
- x = -10?
- x = 0?
- x = 10?



$$\frac{\partial \sigma(x)}{\partial x} = \sigma(x)\,(1-\sigma(x))$$

激活函数



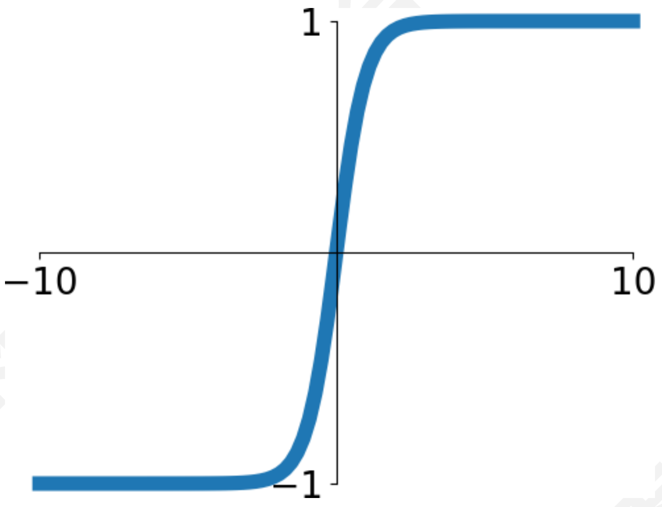


- 如果梯度为 0，则模型权重不会更新

$$\frac{\partial \sigma(x)}{\partial x} = \sigma(x)\,(1-\sigma(x))$$

激活函数

$$\tanh(x) \doteq \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

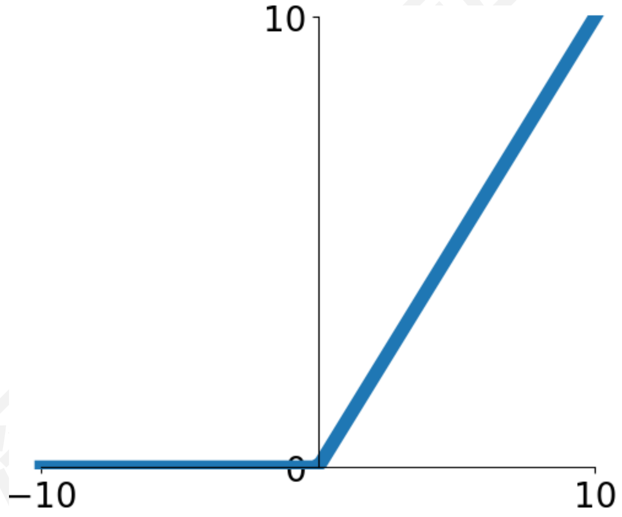


tanh(x)

- 将输出压缩到[-1,1]
- 以 0 为中心
- 问题：
  - 梯度消失问题（饱和时）

激活函数

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

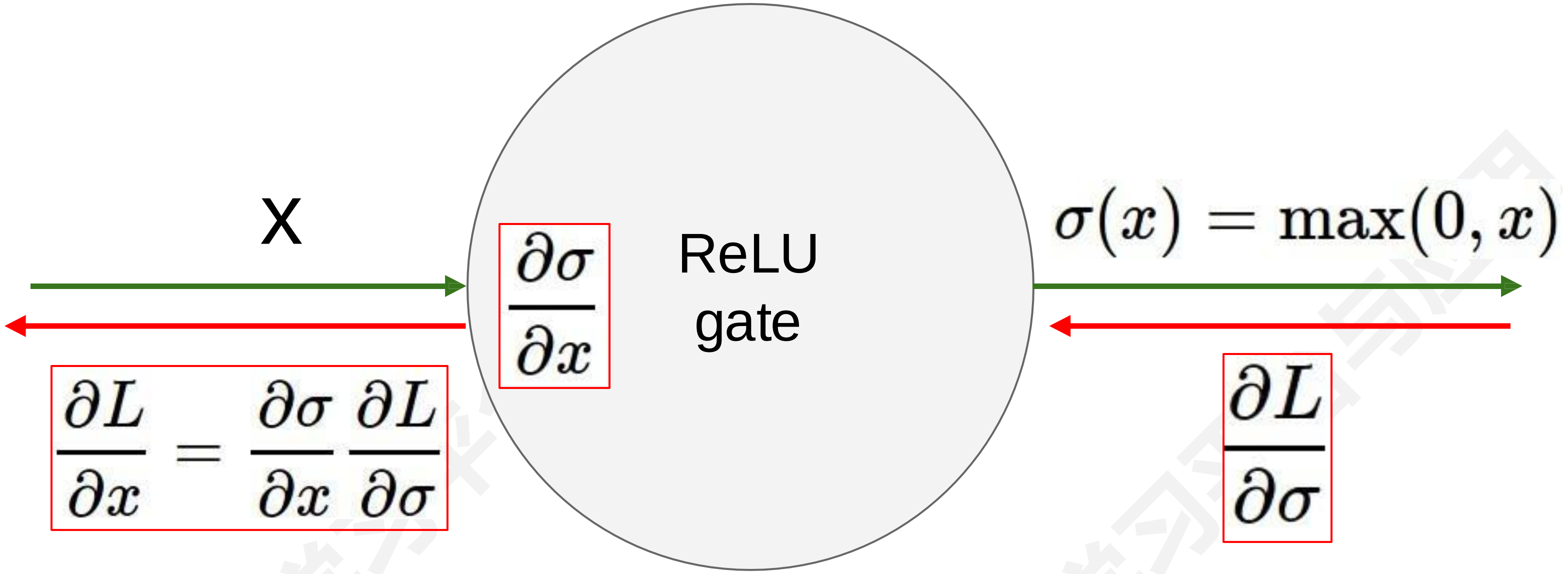


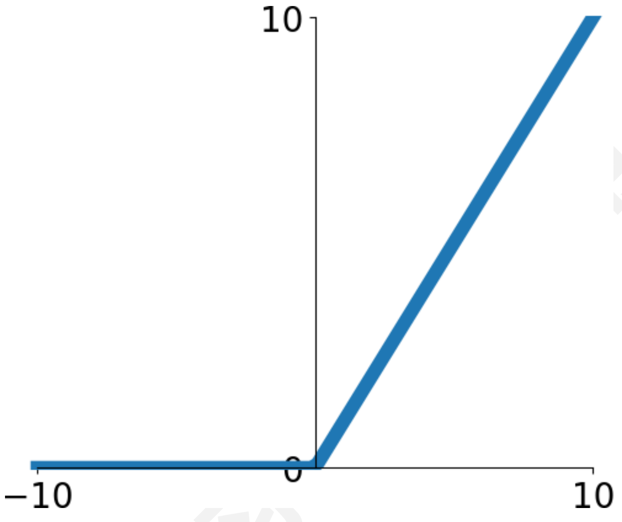
ReLU

(Rectified Linear Unit)

- 在正数区域，梯度不会消失
- 计算效率非常高
- 收敛速度更快
- 问题：
  - 输出不是以 0 为中心
  - 在负数区域，梯度为 0
    （神经元“死掉”）

激活函数

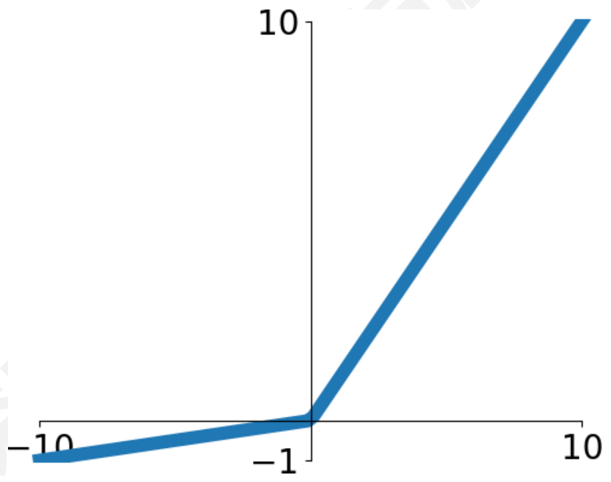




- **x = -10?**
- **x = 0?**
- **x = 10?**

激活函数

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



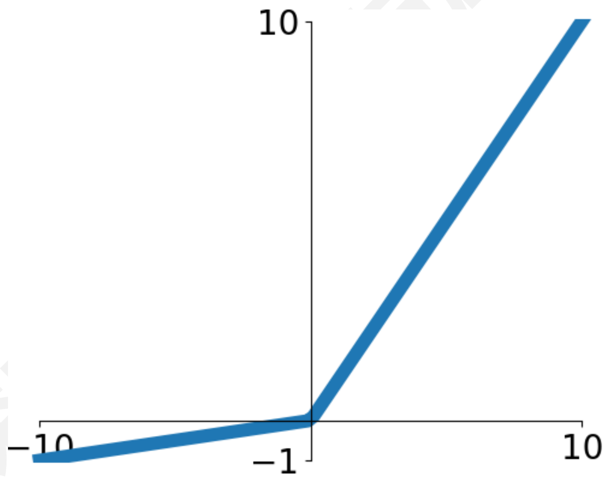
Leaky ReLU

- 在正数区域，梯度不会消失
- 计算效率非常高
- 收敛速度更快
- 任何时候，梯度都不会为 0

  （神经元不会“死掉”）

激活函数

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



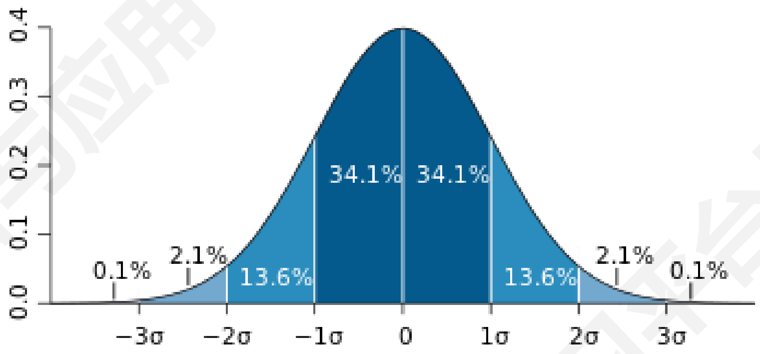
Leaky ReLU

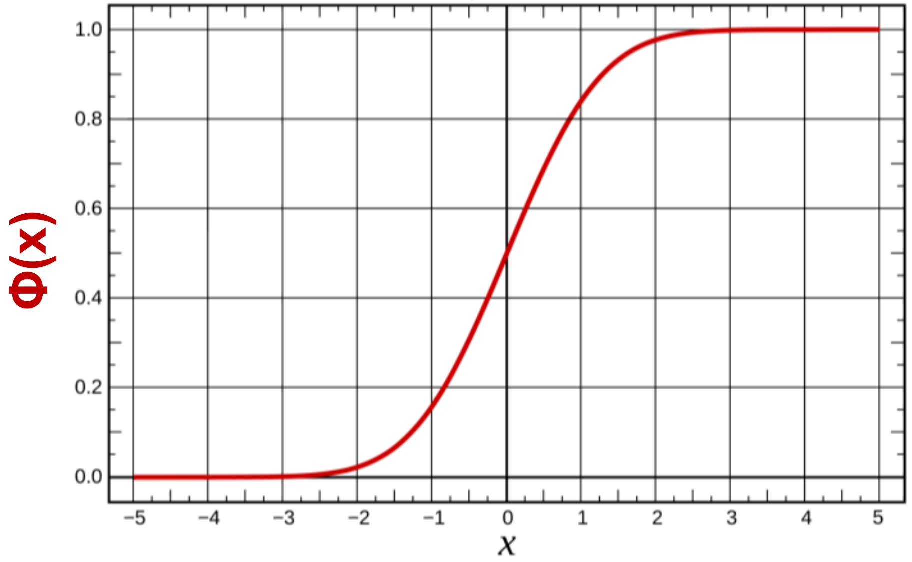
$$f(x) = \max(\alpha x, x)$$

backprop into α (parameter)

Parametric Rectifier (PReLU)

激活函数

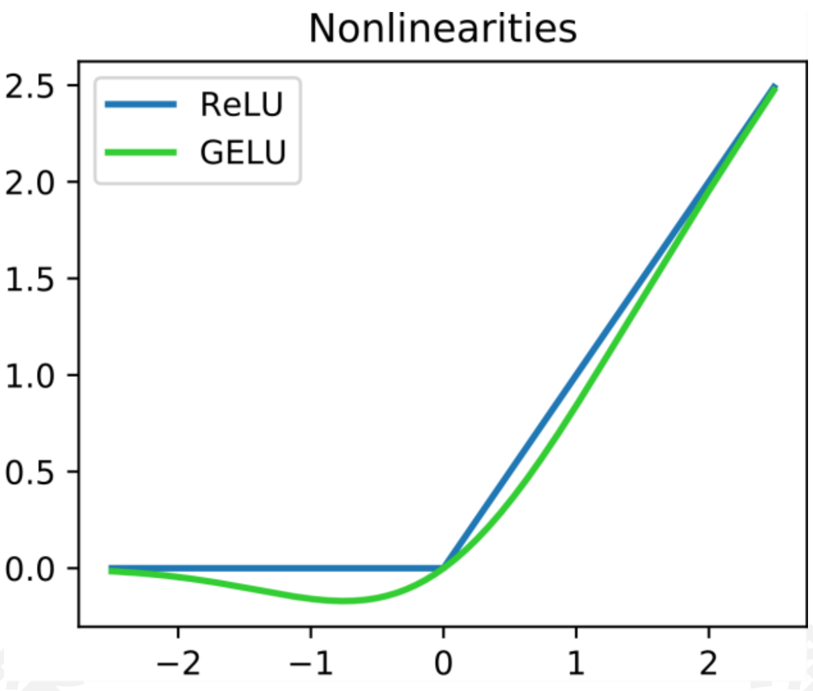


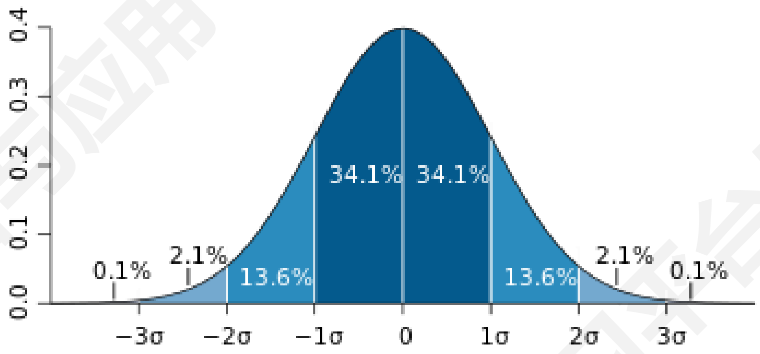


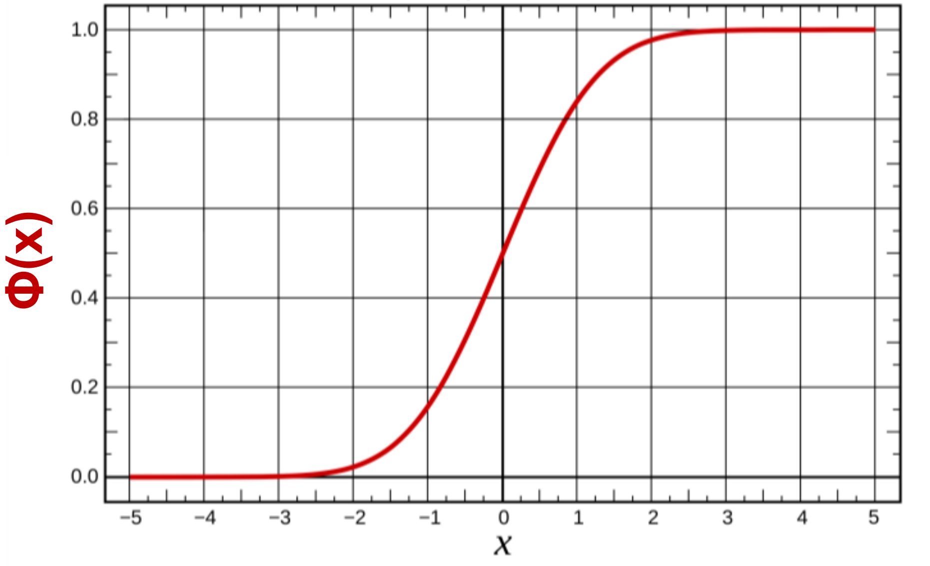
$f(x) = x*\Phi(x)$

GELU
(Gaussian Error Linear Unit)

激活函数





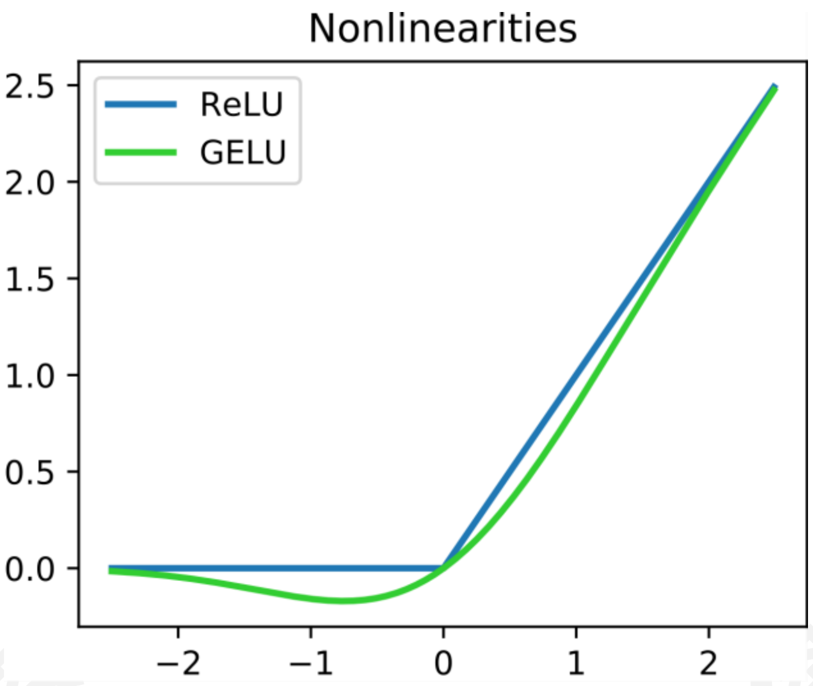


$f(x) = x*\Phi(x)$

GELU

(Gaussian Error Linear Unit)

激活函数



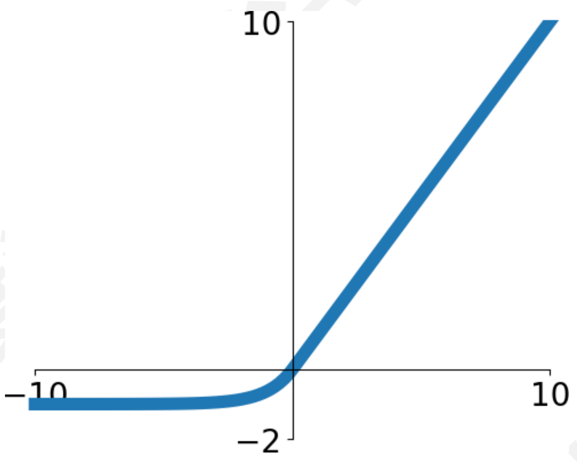
$$f(x) = x*\Phi(x)$$

GELU
(Gaussian Error Linear Unit)

- **0 附近的梯度可计算**
- **平滑函数有助于训练**
- **问题：**
  - **更高的计算成本**
  - **较大的负值处梯度为 0**

激活函数

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

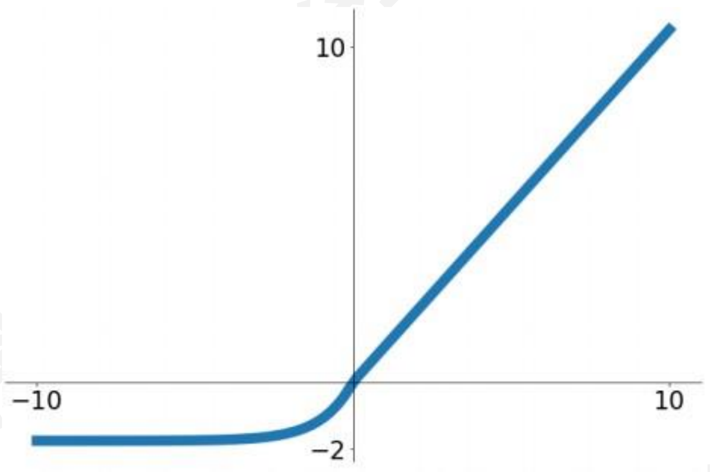


Exponential Linear Units (ELU)

- 具有 ReLU 的所有优点
- 相对于Leaky ReLU，负数区域的稳定性更高
- 问题：
  - 计算成本较高

激活函数

$$f(x) = \begin{cases} \lambda x & \text{if } x > 0 \\ \lambda\alpha(e^x - 1) & \text{otherwise} \end{cases}$$



- ELU 的扩展版本，更适合深度网络
- 具有“自我规范”属性
- 可以在没有 BN 的情况下训练深度网络

Scaled Exponential Linear Units (SELU)

$$\alpha = 1.6732632423543772848170429916717$$
$$\lambda = 1.0507009873554804934193349852946$$

Comments: 9 pages (+ 93 pages appendix)
Subjects: **Machine Learning (cs.LG)**; Machine Learning (stat.ML)
Cite as: arXiv:1706.02515 **[cs.LG]**

激活函数

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

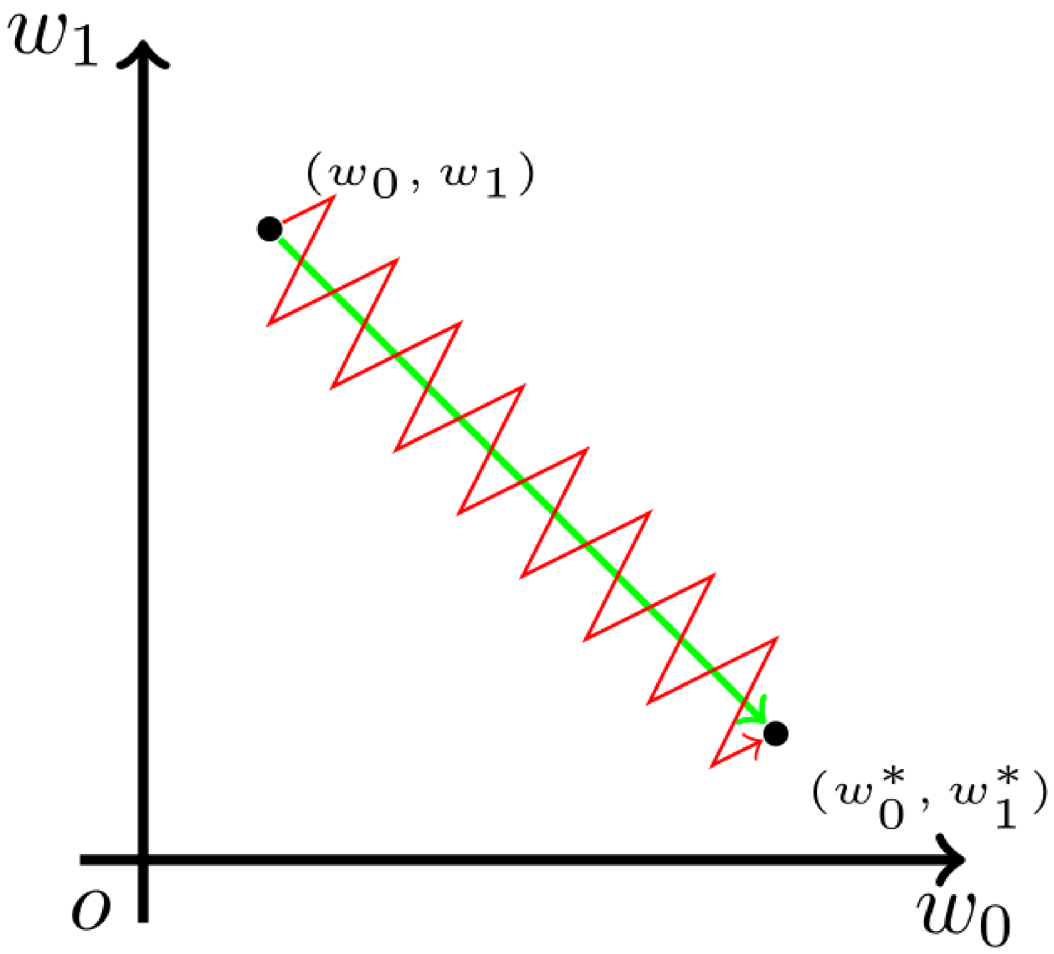
MaxOut

- 非线性
- 具有 Leaky ReLU 的优点
- 不会饱和，梯度不会为 0
- 问题：
  - 参数数量翻倍

激活函数

- 输出以 0 为中心的好处?
- **模型训练收敛更快**
- 假设输出同时为正或负 →

$$
\begin{aligned}
\frac{\partial L}{\partial w_i} &= \frac{\partial L}{\partial f}\frac{\partial f}{\partial z}\frac{\partial z}{\partial w_i} \\
&= \frac{\partial L}{\partial f}\frac{\partial f}{\partial z}x_i
\end{aligned}
$$



激活函数

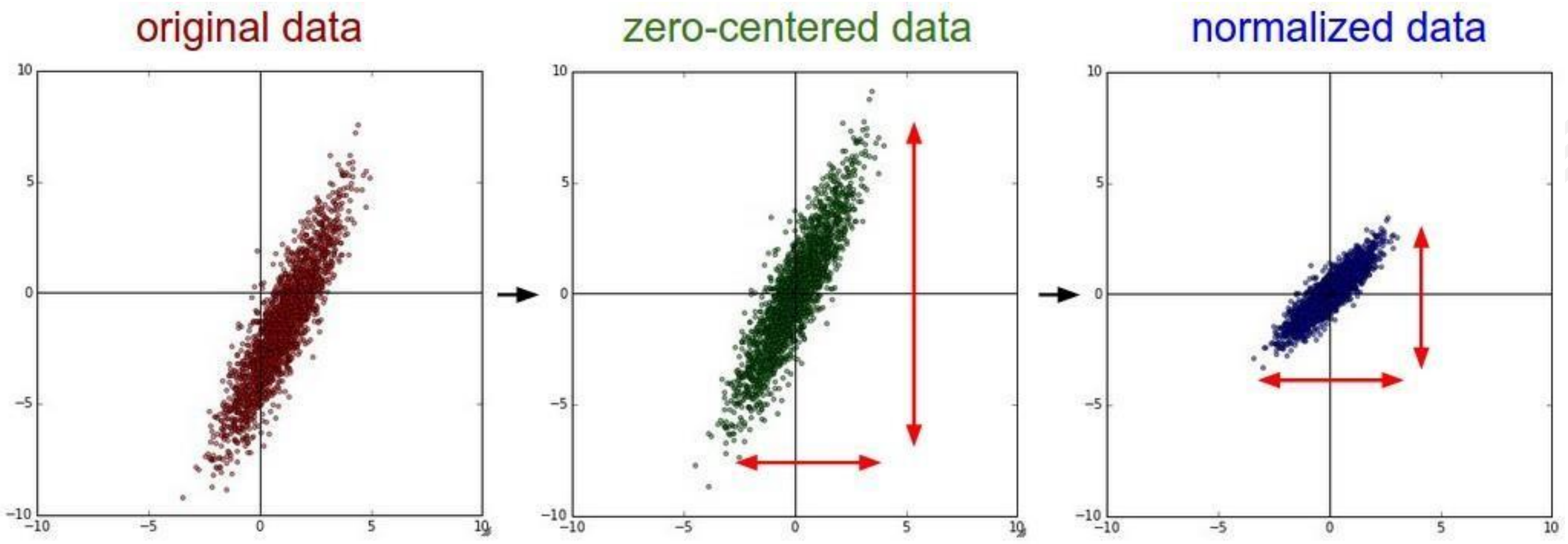
NANJING UNIVERSITY

- 使用建议：
  - 使用 ReLU
  - 可以尝试 Leaky ReLU / PReLU / GELU
  - 尽量避免使用 sigmoid / tanh

大　　纲

- 激活函数
- 数据预处理
- 权重初始化
- 正则化
- 超参数选择

数据预处理




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

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

数据预处理

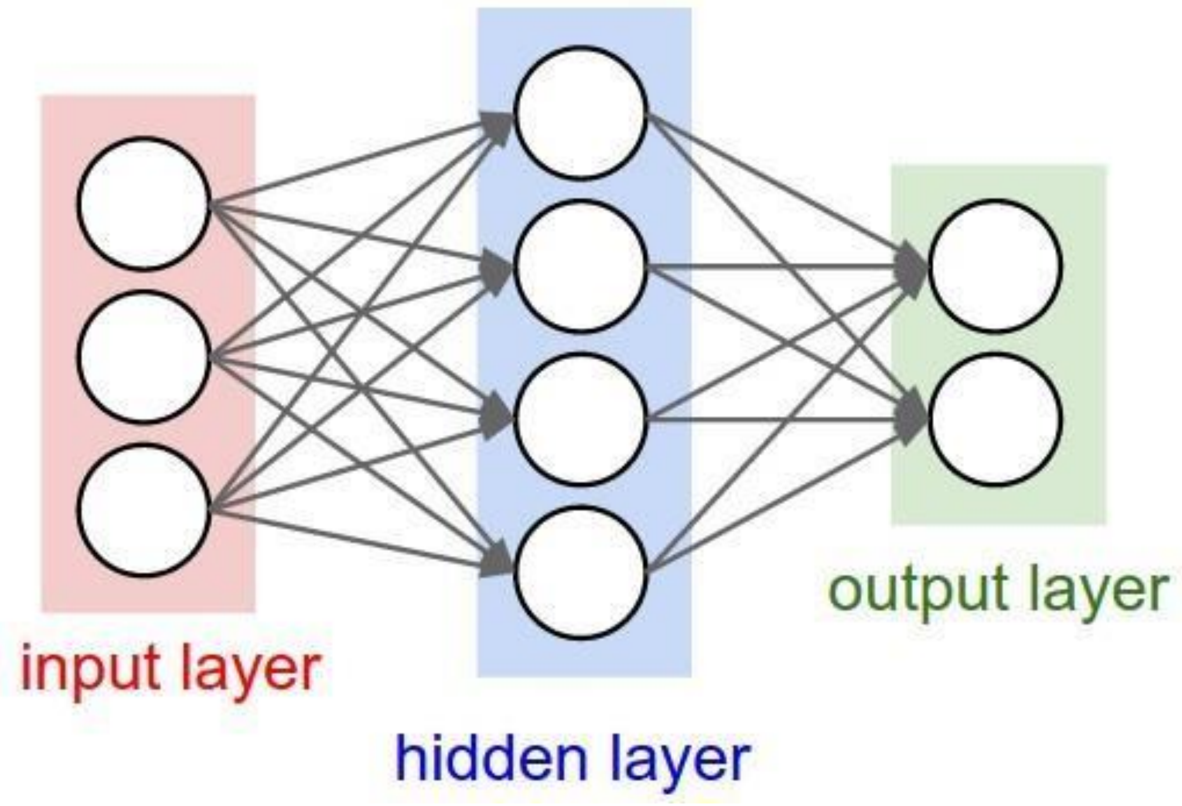
- 假设输入图像大小为[32, 32, 3]
- 输入图像**减去数据集中的平均图像**（[32, 32, 3]）（AlexNet)
- 输入图像**减去每个通道的均值**（[1, 1, 3]）（VGGNet)
- 输入图像**减去每个通道的均值**（[1, 1, 3]），**除以每个通道的标准差**（[1, 1, 3]）（现在最常用的方法)

大　　纲

- 激活函数
- 数据预处理
- 权重初始化
- 正则化
- 超参数选择

权重初始化

- 如果权重初始化为相同的常数，会发生什么情况?



权重初始化

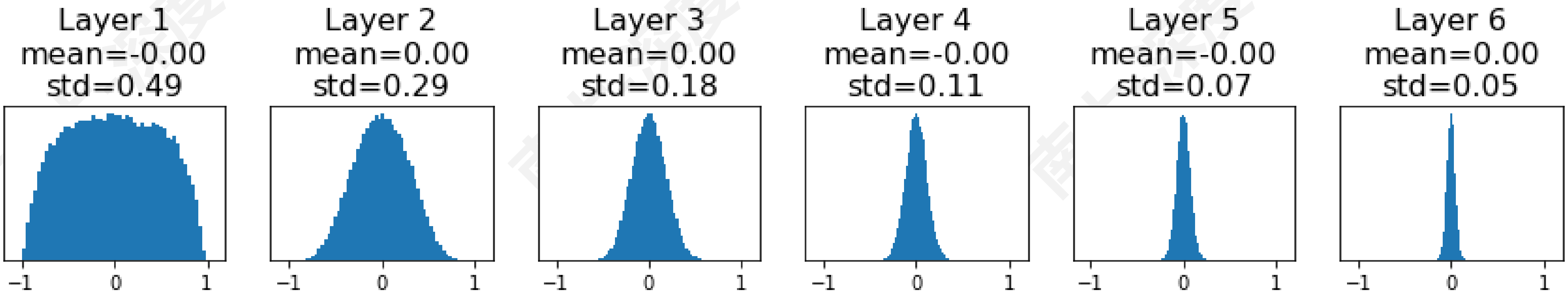
- 想法一：随机初始化为一个小数值（高斯噪声）
- 问题：在浅层网络中效果不错，但在深层网络中效果较差

```
W = 0.01 * np.random.randn(Din, Dout)
```

权重初始化

- 深层网络的的输出趋向于 0
- 梯度很小，网络难以训练

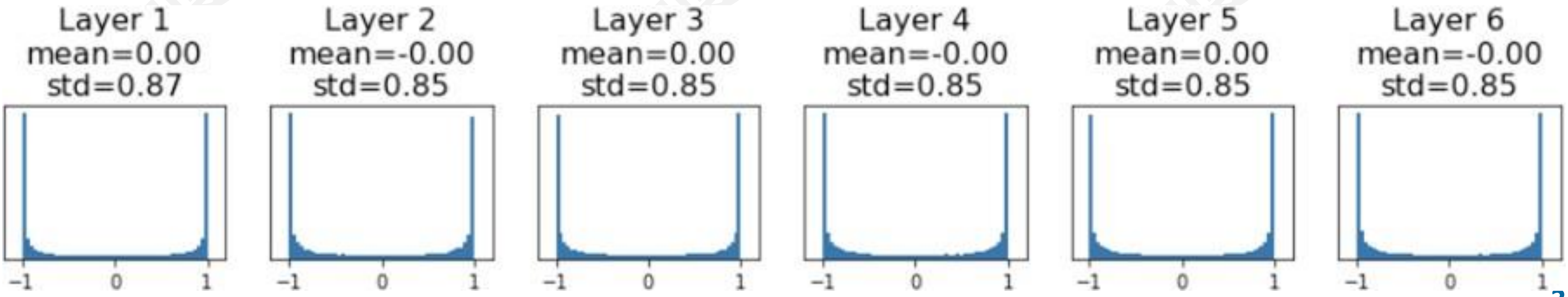
```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = 0.01 * np.random.randn(Din, Dout)
    x = np.tanh(x.dot(W))
    hs.append(x)
```



权重初始化

- 所有的输出都饱和
- 梯度很小，网络难以训练

```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = 0.05 * np.random.randn(Din, Dout)
    x = np.tanh(x.dot(W))
    hs.append(x)
```

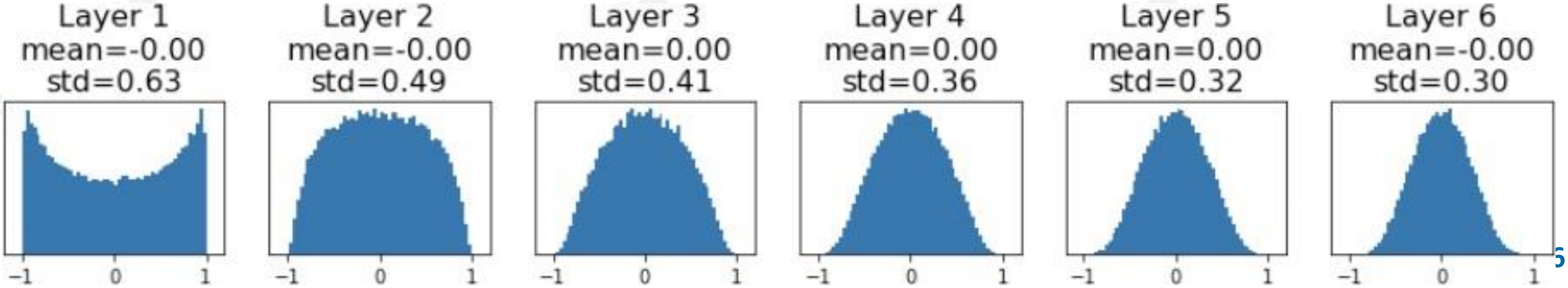


权重初始化

- 所有层都进行合适的初始化
- 对卷积来说， Din是 filter_size² * input_channels

```
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)

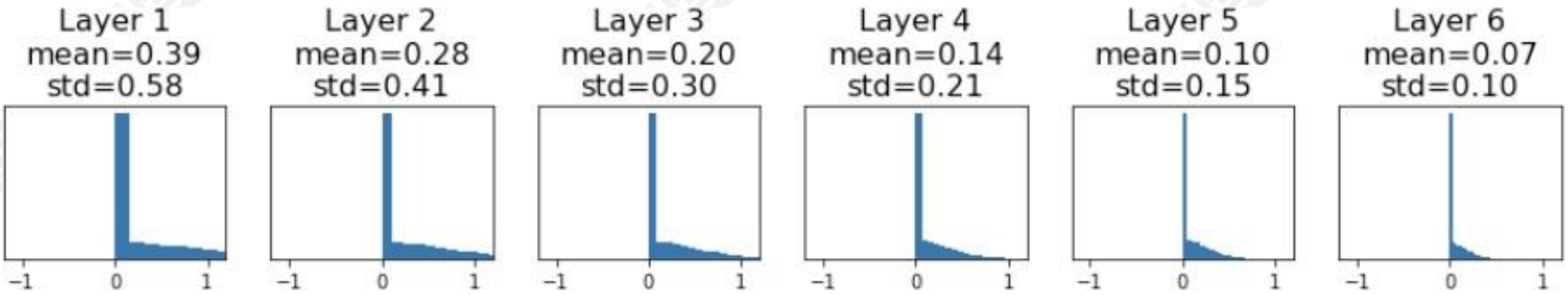


权重初始化

- Xavier 假设激活函数的均值为 0
- 输出又趋向于 0

```
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)
```

将 tanh 变为 ReLU

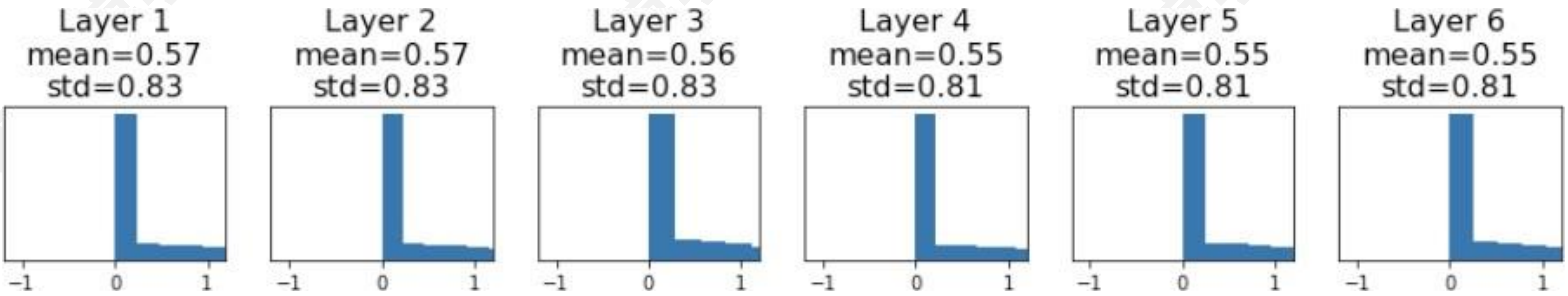


权重初始化

- **Kaiming Initialization**
- **适用于 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(2/Din)
    x = np.maximum(0, x.dot(W))
    hs.append(x)
```

ReLU correction: std = sqrt(2 / Din)



He et al, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification", ICCV 2015

权重初始化

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

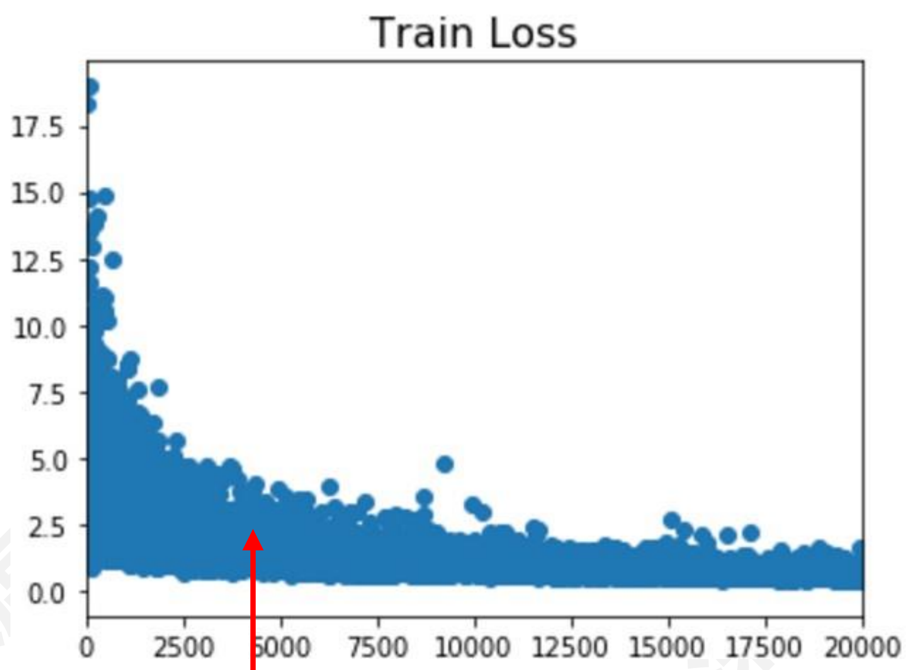
The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks, Frankle and Carbin, 2019

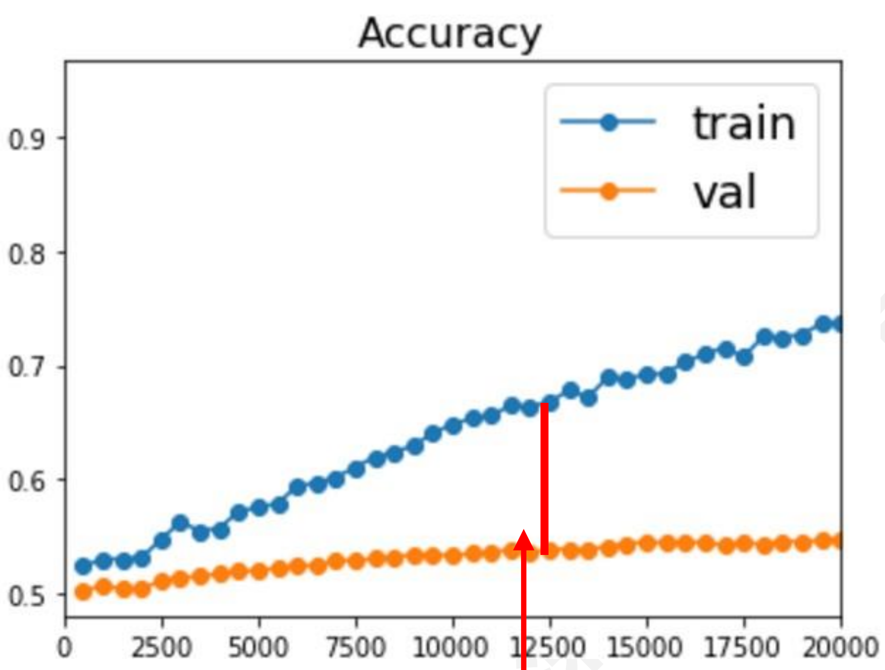
大　　纲

NANJING UNIVERSITY

- 激活函数
- 数据预处理
- 权重初始化
- 正则化
- 超参数选择

训练与测试

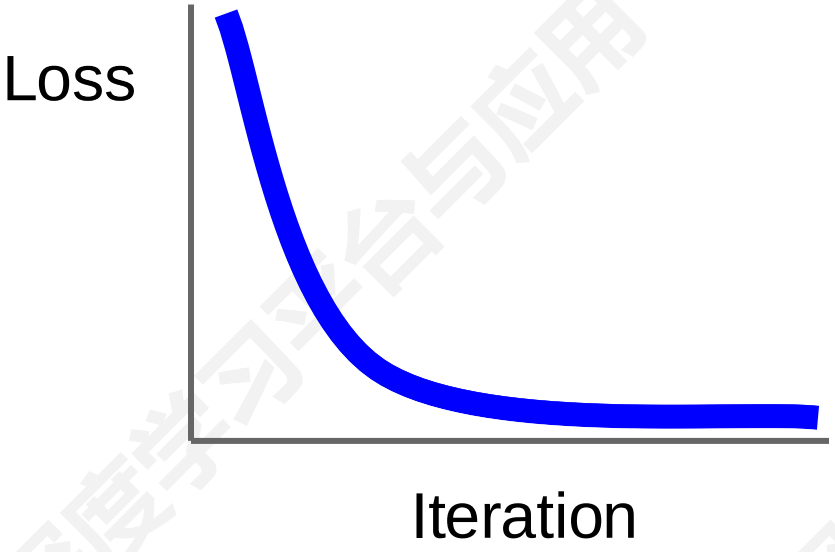


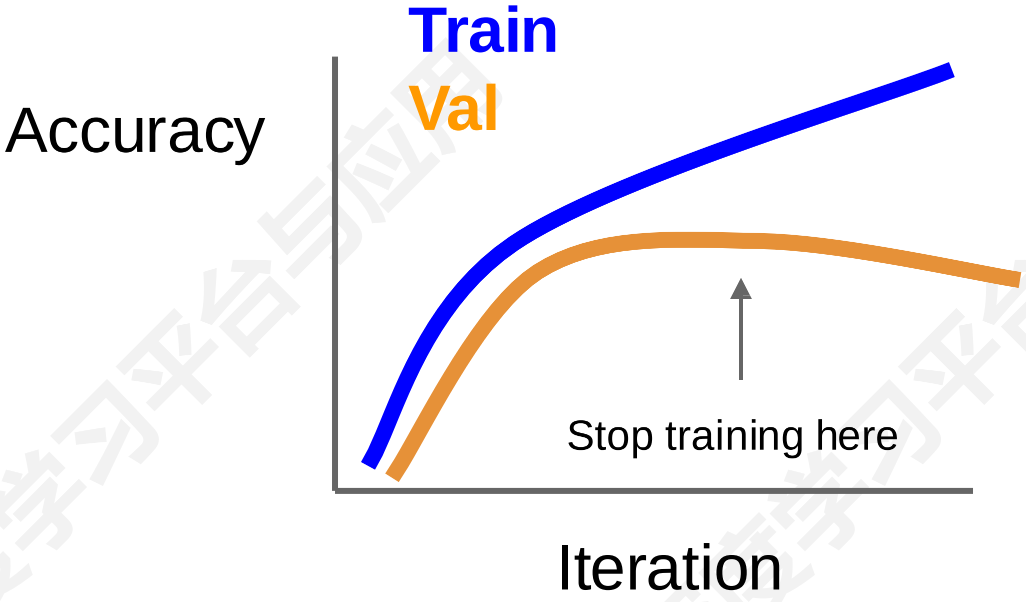


更好的优化算法有助于降低训练损失

但我们真正关心的是：降低测试泛化损失，减少 gap

训练：提前停止





当 val 上的准确性降低时停止训练模型

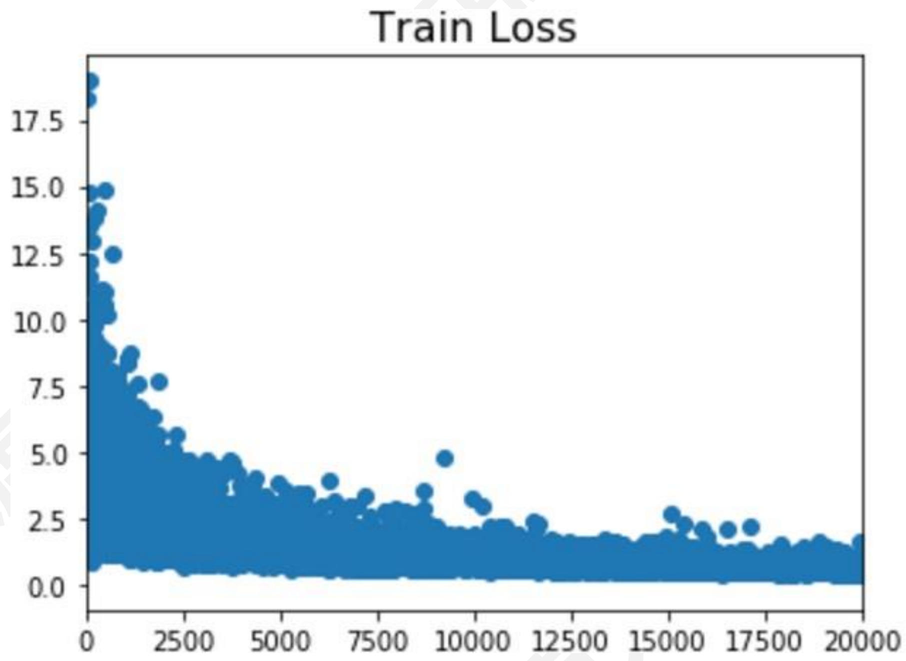
或者：一直训练，但始终存储在 val 上效果最好的的模型快照

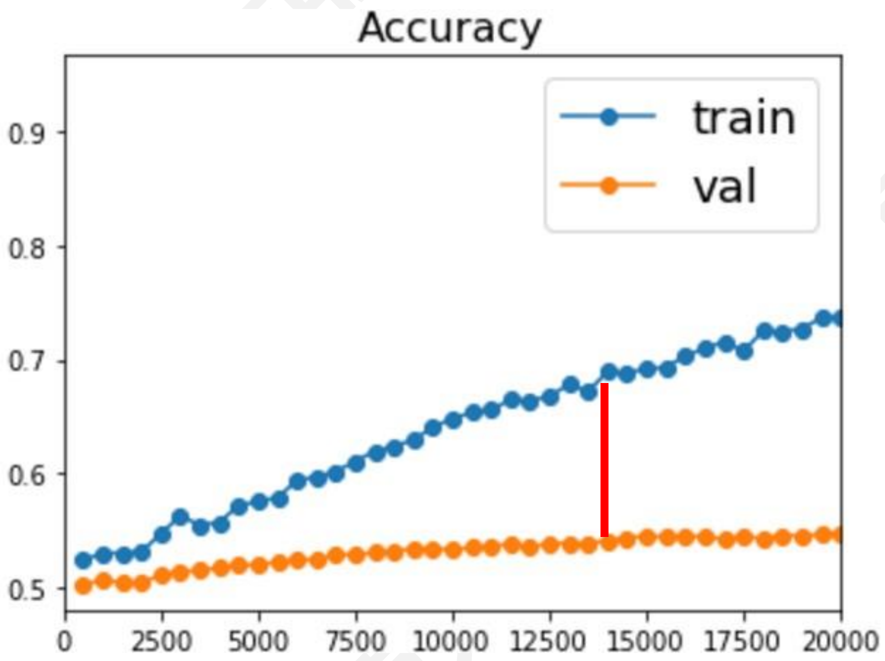
模型集成

南京大学 NANJING UNIVERSITY

- 训练多个独立模型
- 测试时使用所有模型的预测结果（例如去均值）
- 可以带来可观的性能提升

如何提升单个模型的性能





正则化

正则化：在损失函数中加入正则化项

$$L = \frac{1}{N}\sum_{i=1}^{N}\sum_{j\neq y_i}\max(0, f(x_i;W)_j - f(x_i;W)_{y_i} + 1) + \boxed{\lambda R(W)}$$

常用的正则化项:

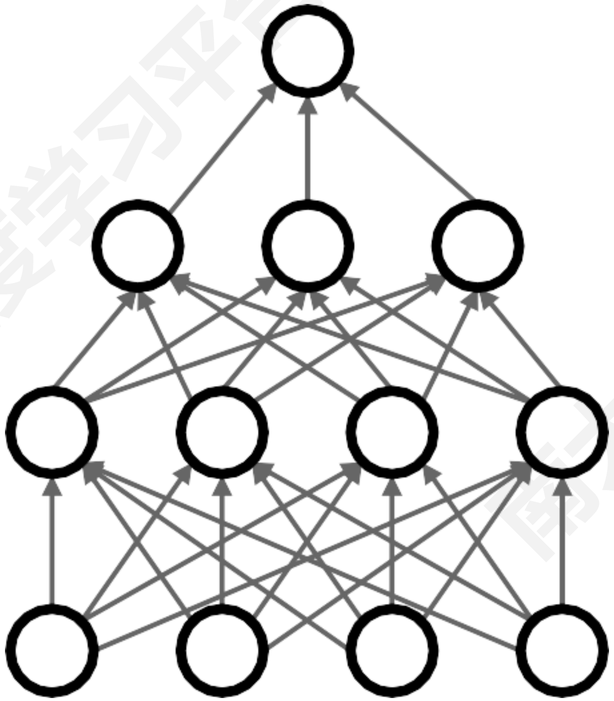
L2 regularization $R(W) = \sum_k \sum_l W_{k,l}^2$ (Weight decay)

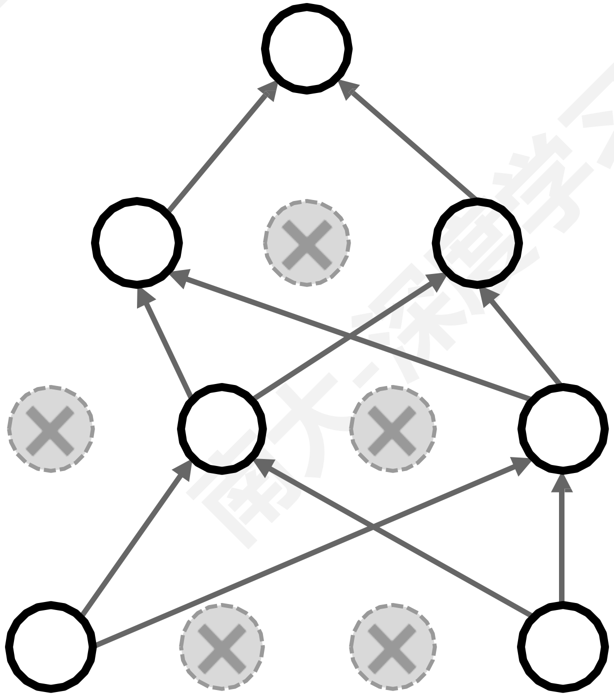
L1 regularization $R(W) = \sum_k \sum_l |W_{k,l}|$

Elastic net (L1 + L2) $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

正则化：dropout

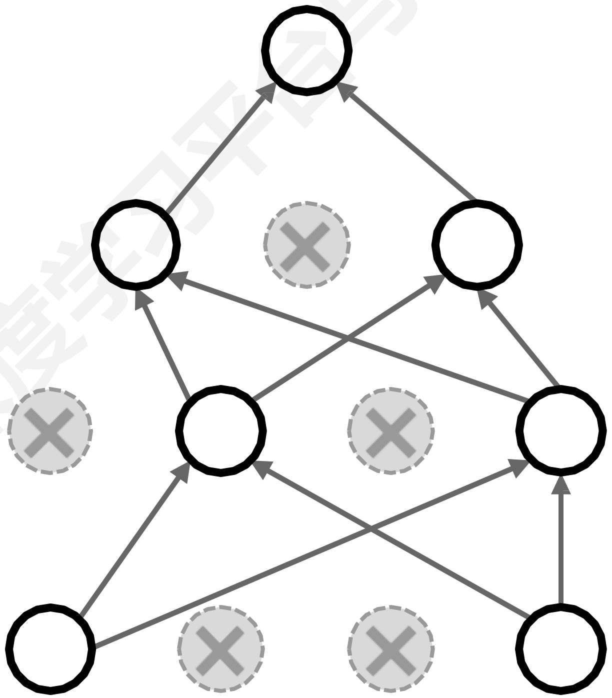
- 在模型训练的每次向前传播计算中，将一些神经元随机设置为零
- 随机概率是一个超参数，常用 0.5





正则化：dropout

```python
p = 0.5 # probability of keeping a unit active. higher = less dropout

def train_step(X):
  """ X contains the data """

  # forward pass for example 3-layer neural network
  H1 = np.maximum(0, np.dot(W1, X) + b1)
  U1 = np.random.rand(*H1.shape) < p # first dropout mask
  H1 *= U1 # drop!
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  U2 = np.random.rand(*H2.shape) < p # second dropout mask
  H2 *= U2 # drop!
  out = np.dot(W3, H2) + b3

  # backward pass: compute gradients... (not shown)
  # perform parameter update... (not shown)
```



正则化：dropout

- 让网络学习冗余表示，防止特征的协同适应

正则化：dropout

- 如果测试时也使用 dropout，模型的输出会有随机性

Output (label) Input (image) Random mask

$$\boxed{y} = f_W(\boxed{x}, \boxed{z})$$

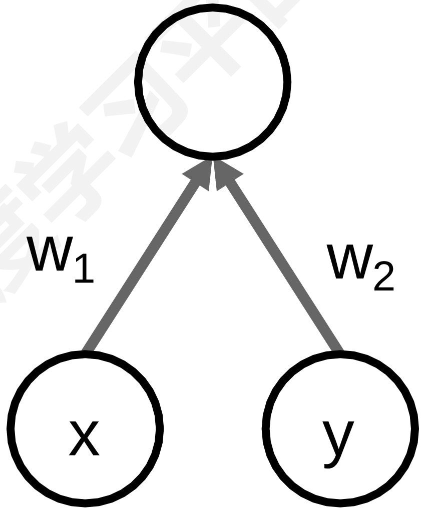
- 测试时，我们想要求随机性的期望作为确定的输出

$$y = f(x) = E_z\big[f(x, z)\big] = \int p(z)f(x, z)dz$$

- 但是这个积分计算过于困难

正则化：dropout

- 对测试时的 dropout 积分进行近似
- 测试： $$E\left[a\right] = w_1x + w_2y$$
- 训练： $$\begin{aligned} E[a] =& \frac{1}{4}(w_1x + w_2y) + \frac{1}{4}(w_1x + 0y) \\ &+ \frac{1}{4}(0x + 0y) + \frac{1}{4}(0x + w_2y) \\ =& \frac{1}{2}(w_1x + w_2y) \end{aligned}$$



- 测试时，将输出乘以 dropout 概率

$$y = f(x) = E_z\left[f(x,z)\right] = \int p(z)f(x,z)dz$$

正则化：dropout

- 在测试时，所有神经元始终处于激活状态
- 我们必须缩放激活，以便让每个神经元：
  - 测试时的输出=训练时的预期输出

```
def predict(X):
  # ensembled forward pass
  H1 = np.maximum(0, np.dot(W1, X) + b1) * p # NOTE: scale the activations
  H2 = np.maximum(0, np.dot(W2, H1) + b2) * p # NOTE: scale the activations
  out = np.dot(W3, H2) + b3
```

正则化：dropout

南京大學 NANJING UNIVERSITY

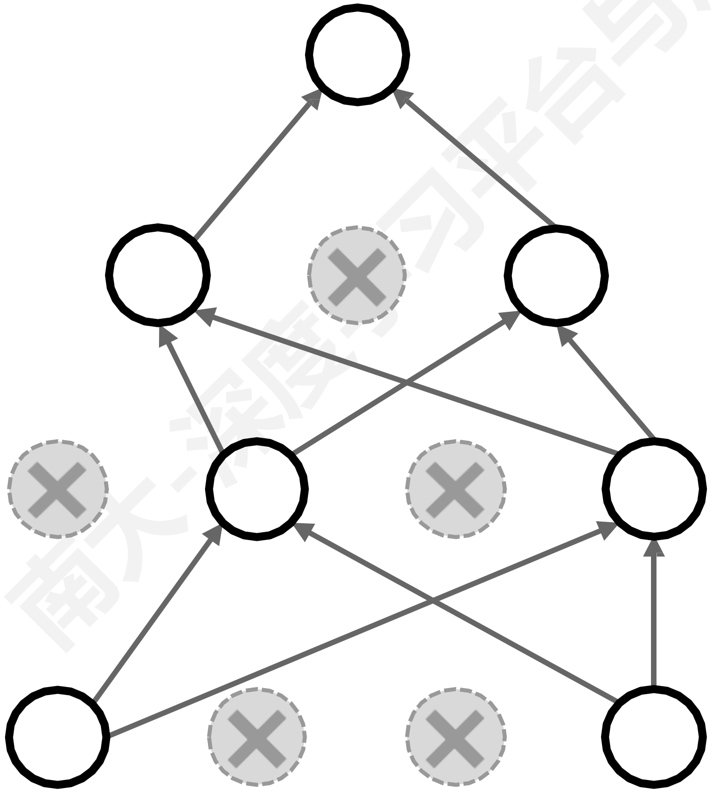
```python
p = 0.5 # probability of keeping a unit active. higher = less dropout

def train_step(X):
  # forward pass for example 3-layer neural network
  H1 = np.maximum(0, np.dot(W1, X) + b1)
  U1 = (np.random.rand(*H1.shape) < p) / p # first dropout mask. Notice /p!
  H1 *= U1 # drop!
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  U2 = (np.random.rand(*H2.shape) < p) / p # second dropout mask. Notice /p!
  H2 *= U2 # drop!
  out = np.dot(W3, H2) + b3

  # backward pass: compute gradients... (not shown)
  # perform parameter update... (not shown)

def predict(X):
  # ensembled forward pass
  H1 = np.maximum(0, np.dot(W1, X) + b1) # no scaling necessary
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  out = np.dot(W3, H2) + b3
```

Inverted dropout

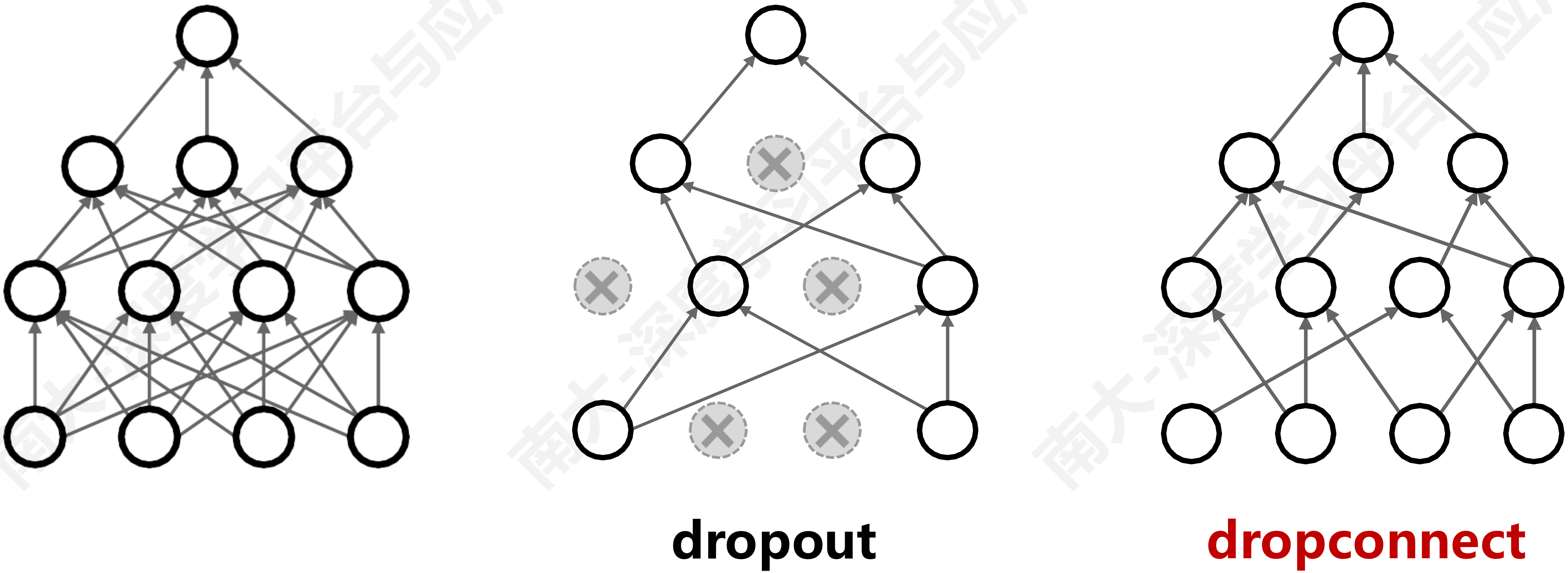
测试时不变

正则化：dropout

- dropout 可以看作模型集成
- dropout 训练了大量参数共享的模型
- 每次的随机掩码都对应了一个模型

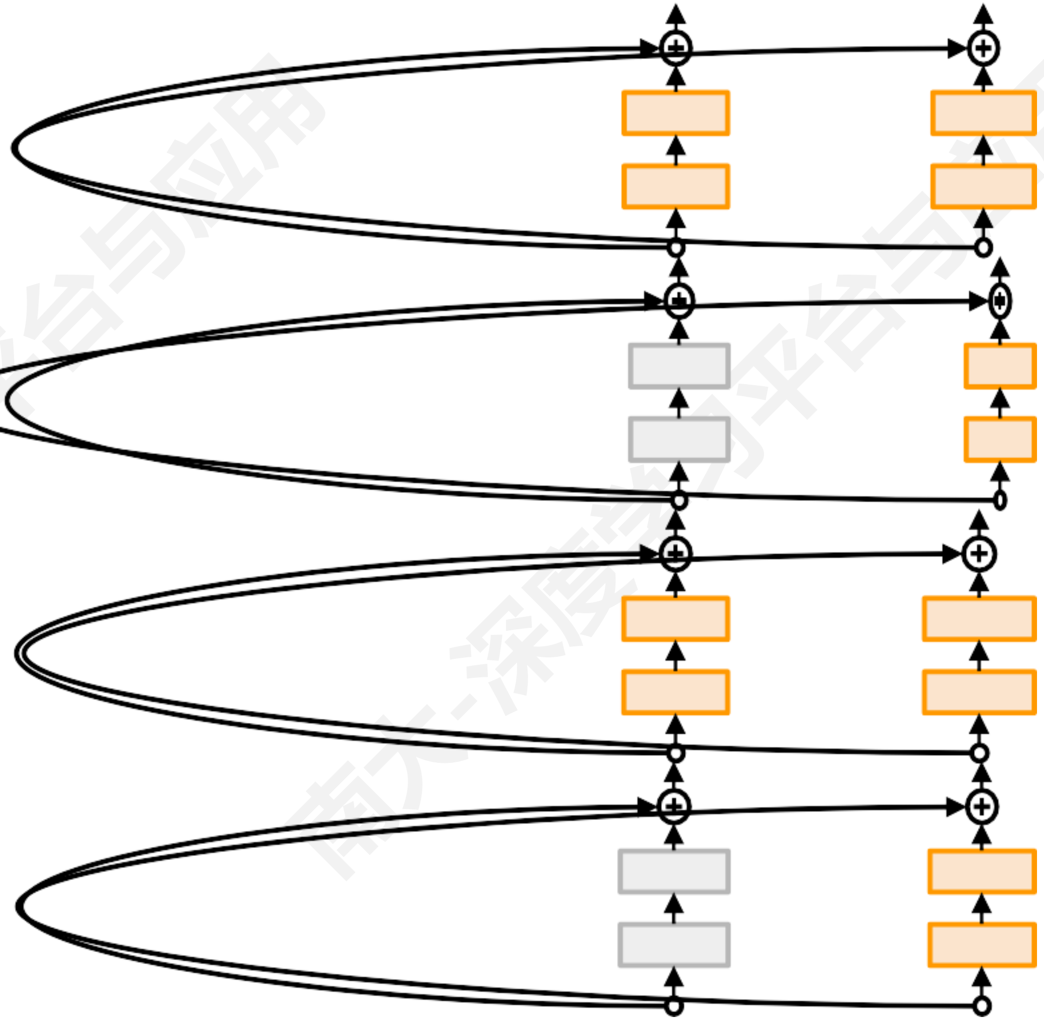


正则化：dropconnect



正则化：Stochastic Depth

- 训练时：随机跳过某些层
- 测试时：使用所有层



正则化

- 训练时，增加随机性

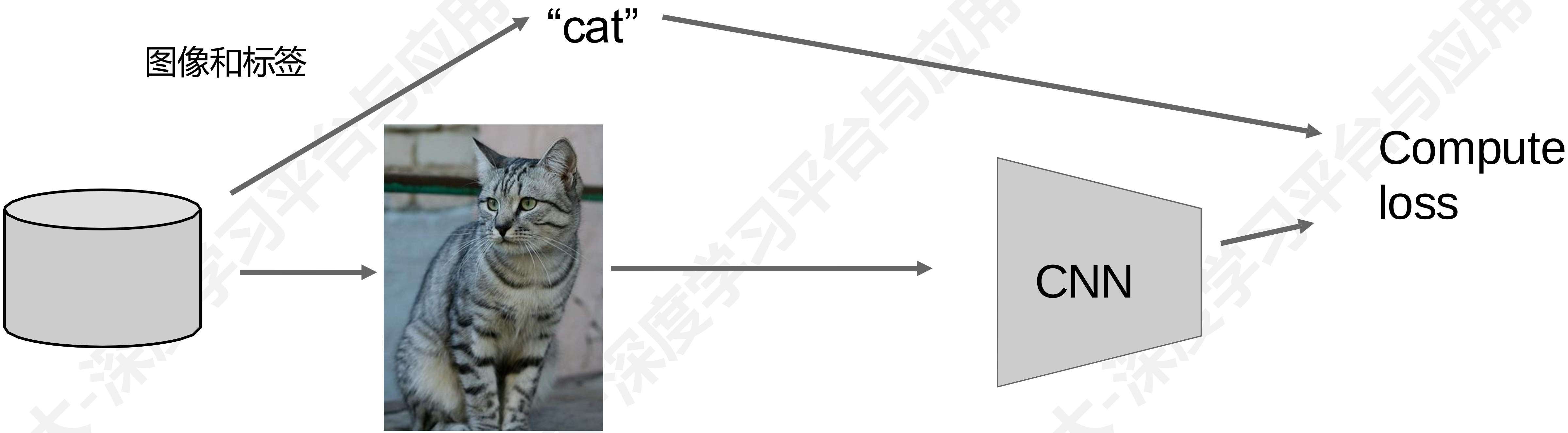
$$y = f_W(x, z)$$

- 测试时，得到随机性的期望输出

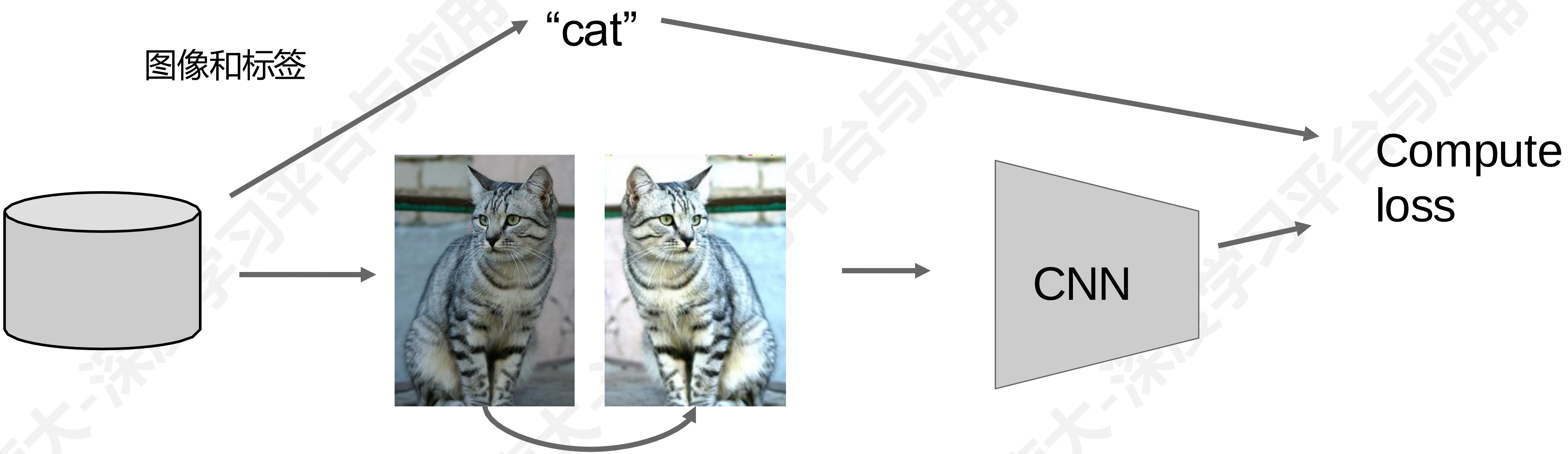
$$y = f(x) = E_z\left[f(x, z)\right] = \int p(z) f(x, z) dz$$

- Batch Normalization
- 训练时，对随机 mini-batch 的统计量进行归一化
- 测试时，使用固定的统计量进行归一化

正则化：数据增强



正则化：数据增强



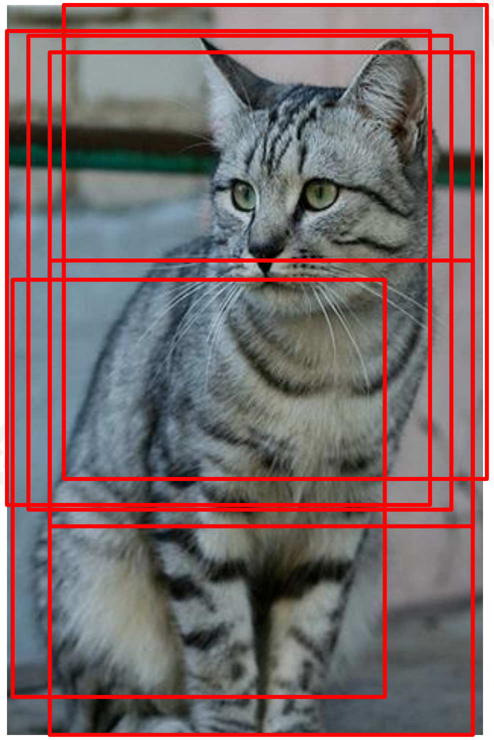
图像变换

正则化：数据增强

■ 水平翻转

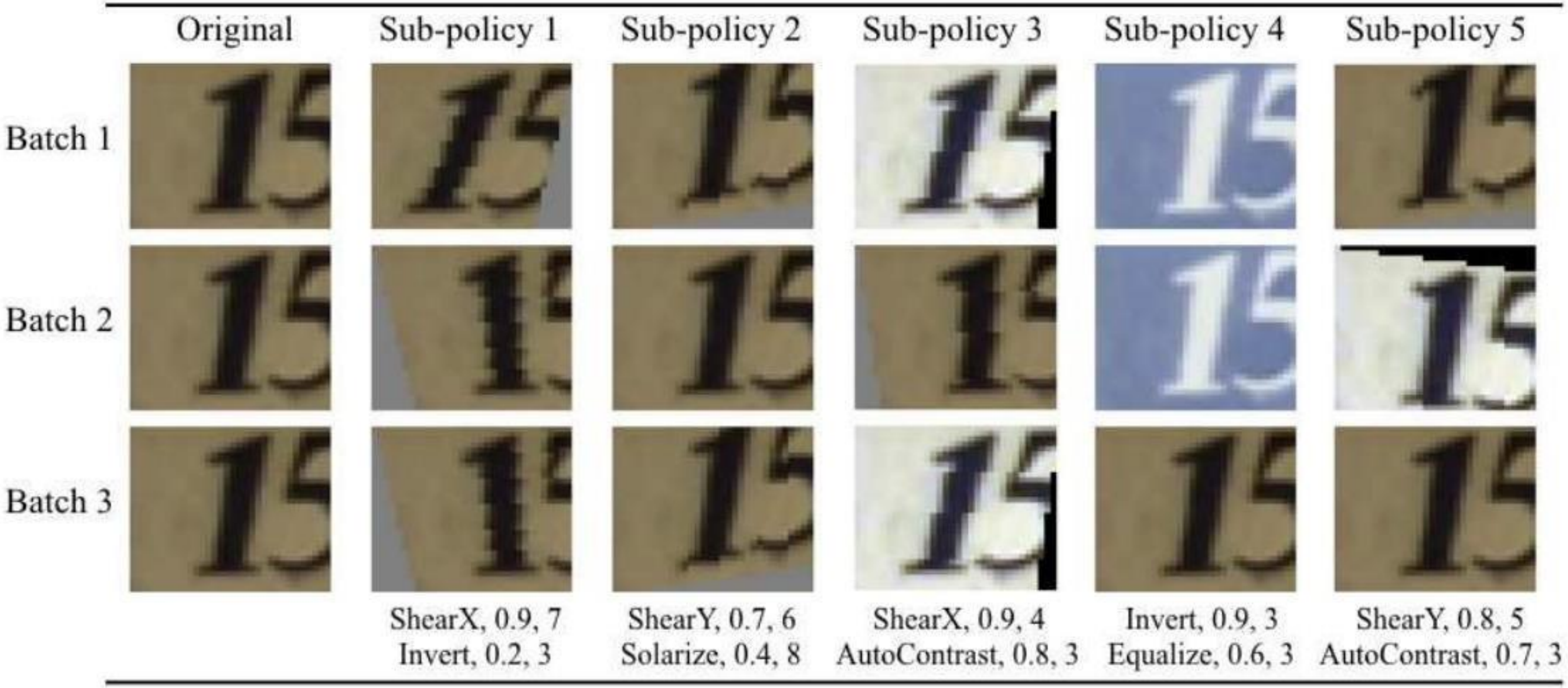
正则化：数据增强

- 随机裁剪和缩放



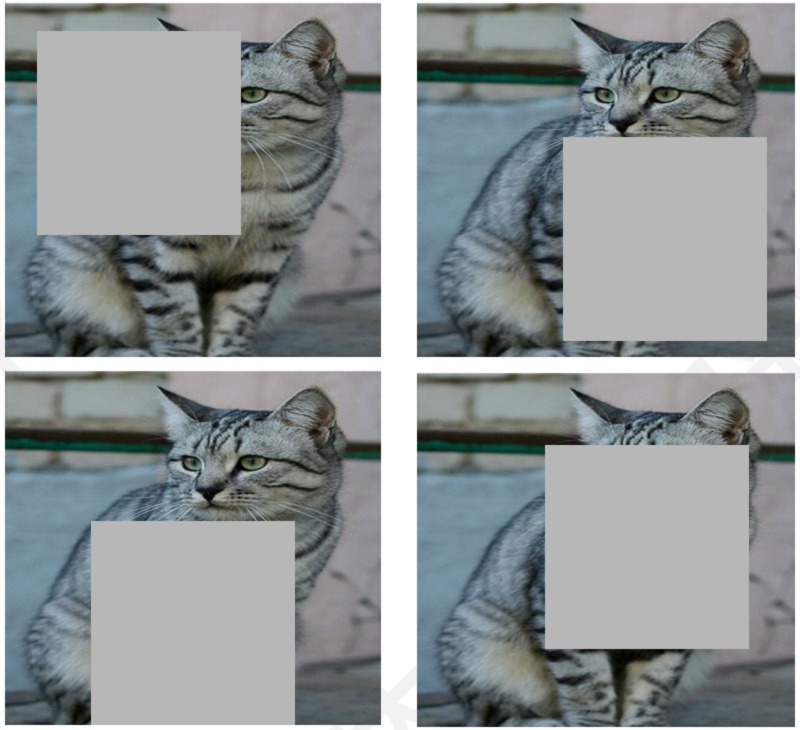
- 颜色扰动

正则化：自动数据增强


Original
Sub-policy 1
Sub-policy 2
Sub-policy 3
Sub-policy 4
Sub-policy 5
Batch 1
Batch 2
Batch 3
ShearX, 0.9, 7
Invert, 0.2, 3
ShearY, 0.7, 6
Solarize, 0.4, 8
ShearX, 0.9, 4
AutoContrast, 0.8, 3
Invert, 0.9, 3
Equalize, 0.6, 3
ShearY, 0.8, 5
AutoContrast, 0.7, 3

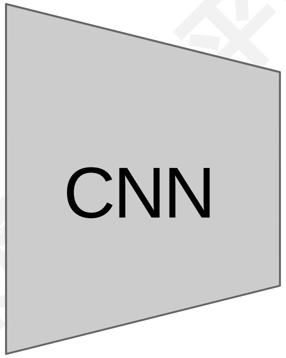
正则化：Cutout

- 在小数据集上的效果很好
- 大数据集上不常用



正则化：Mixup

- 训练：随机将多张图片混合为一张图片
- 测试：使用原始图像



标签：
cat: 0.4
dog: 0.6

随机混合两张训练图片的像素，例如40% cat, 60% dog

大　　　纲

- 激活函数
- 数据预处理
- 权重初始化
- 正则化
- **超参数选择**

超参数选择

- 第一步：检查初始损失（关闭权重衰减）
  - C 个类别，softmax loss 的初始损失应为 -log(1/C) = log(C)

$$-\log\left(\frac{e^{y_i}}{\sum_j e^{j}}\right) = -\log\frac{1}{C} = \log C$$

超参数选择

- 第一步：检查初始损失（关闭权重衰减）
- 第二步：过拟合少量样本

超参数选择

南京大学 NANJING UNIVERSITY

- 第一步：检查初始损失（关闭权重衰减）
- 第二步：过拟合少量样本
- 第三步：选择合适的 LR
  - 让损失在 100 iterations 内显著下降
  - LR: 1e-1, 1e-2, 1e-3, 1e-4, ...

超参数选择

- 第一步：检查初始损失（关闭权重衰减）
- 第二步：过拟合少量样本
- 第三步：选择合适的 LR
- 第四步：粗调 1-5 epochs

超参数选择

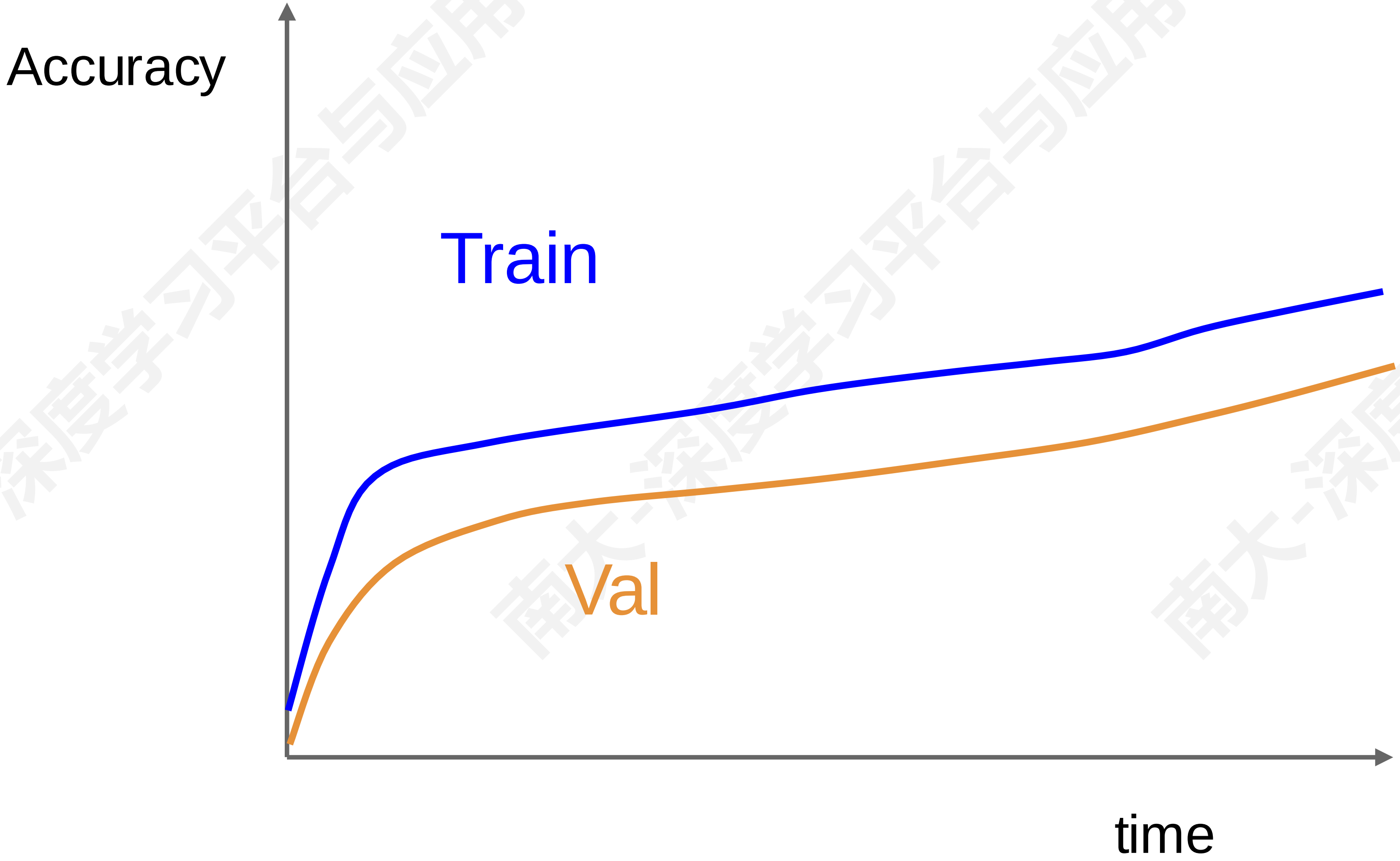
- 第一步：检查初始损失（关闭权重衰减）
- 第二步：过拟合少量样本
- 第三步：选择合适的 LR
- 第四步：粗调 1-5 epochs
- 第五步：精调 10-20 epochs（从第四步选择最好的模型）

超参数选择

- 第一步：检查初始损失（关闭权重衰减）
- 第二步：过拟合少量样本
- 第三步：选择合适的 LR
- 第四步：粗调 1-5 epochs
- 第五步：精调 10-20 epochs（从第四步选择最好的模型）
- 第六步：检查损失和准确率曲线

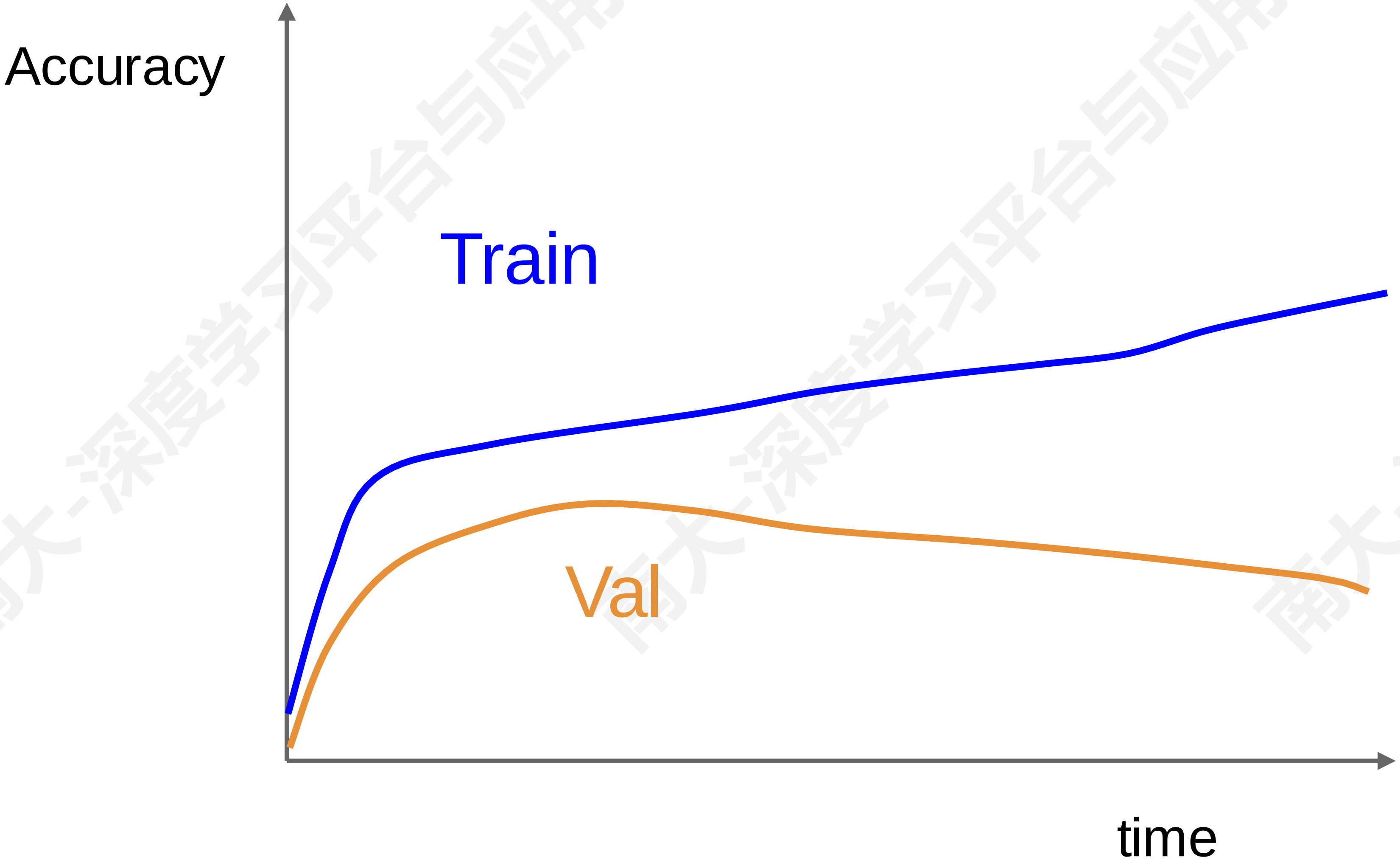
超参数选择

■ 如果准确率持续上升，需要训练更多轮次



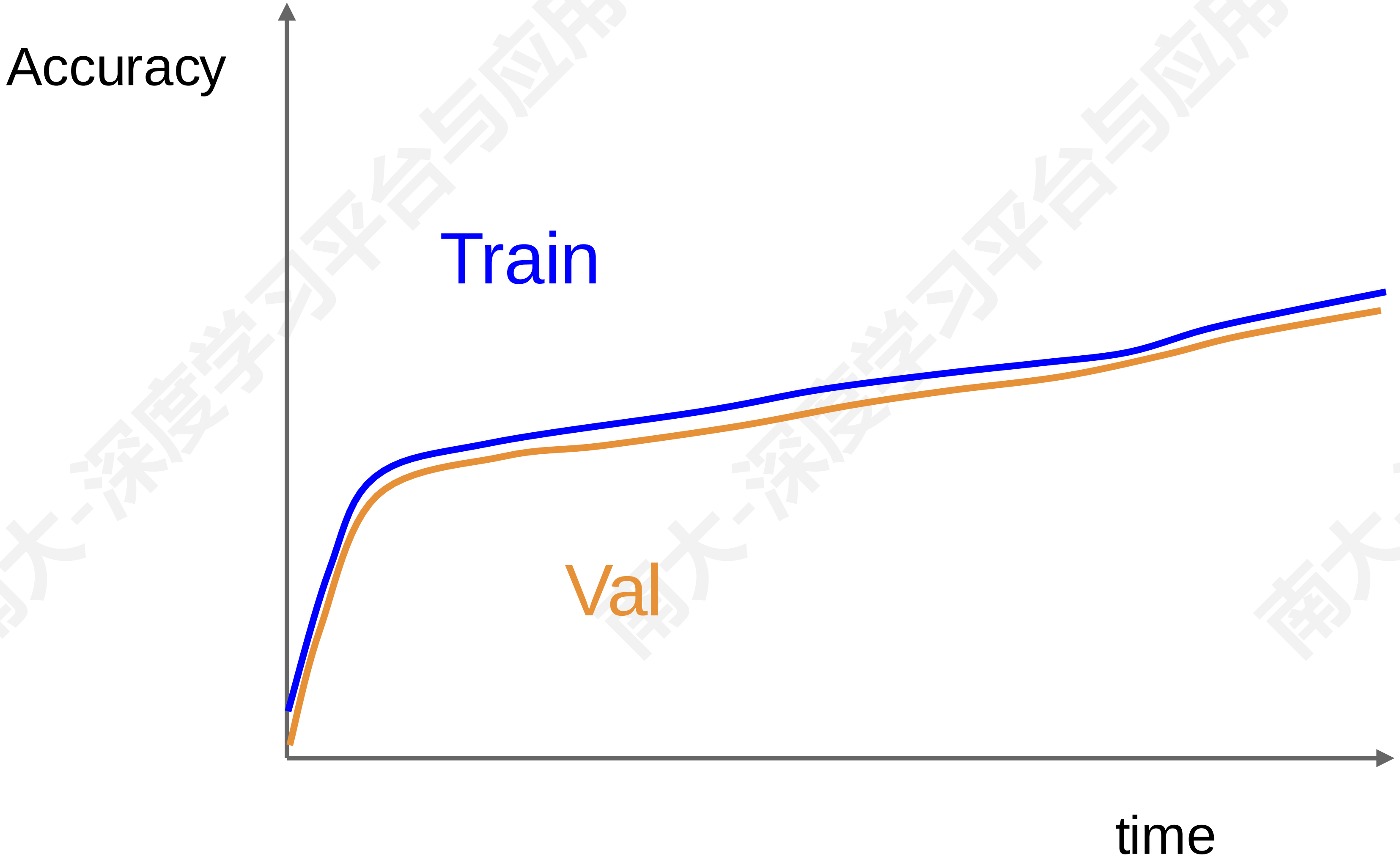
超参数选择

- 巨大的 train-val gap 意味着过拟合，需要增加正则化/数据



超参数选择

- Train-val 的 gap 很小意味着欠拟合，需要训练更久/更大模型



超参数选择

南京大学 NANJING UNIVERSITY

- 第一步：检查初始损失（关闭权重衰减）
- 第二步：过拟合少量样本
- 第三步：选择合适的 LR
- 第四步：粗调 1-5 epochs
- 第五步：精调 10-20 epochs（从第四步选择最好的模型）
- 第六步：检查损失和准确率曲线
- 第七步：如果第六步出现问题，返回第五步

超参数选择

■ 参数搜索方法