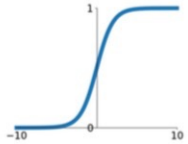
计算机视觉
Computer Vision

Lecture 7: 神经网络的训练2

L06：神经网络的训练1

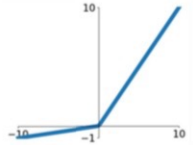
Sigmoid

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



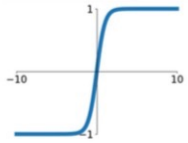
Leaky ReLU

$\max(0.1x, x)$



tanh

$\tanh(x)$

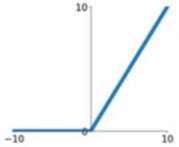


Maxout

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

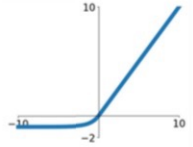
ReLU

$\max(0, x)$



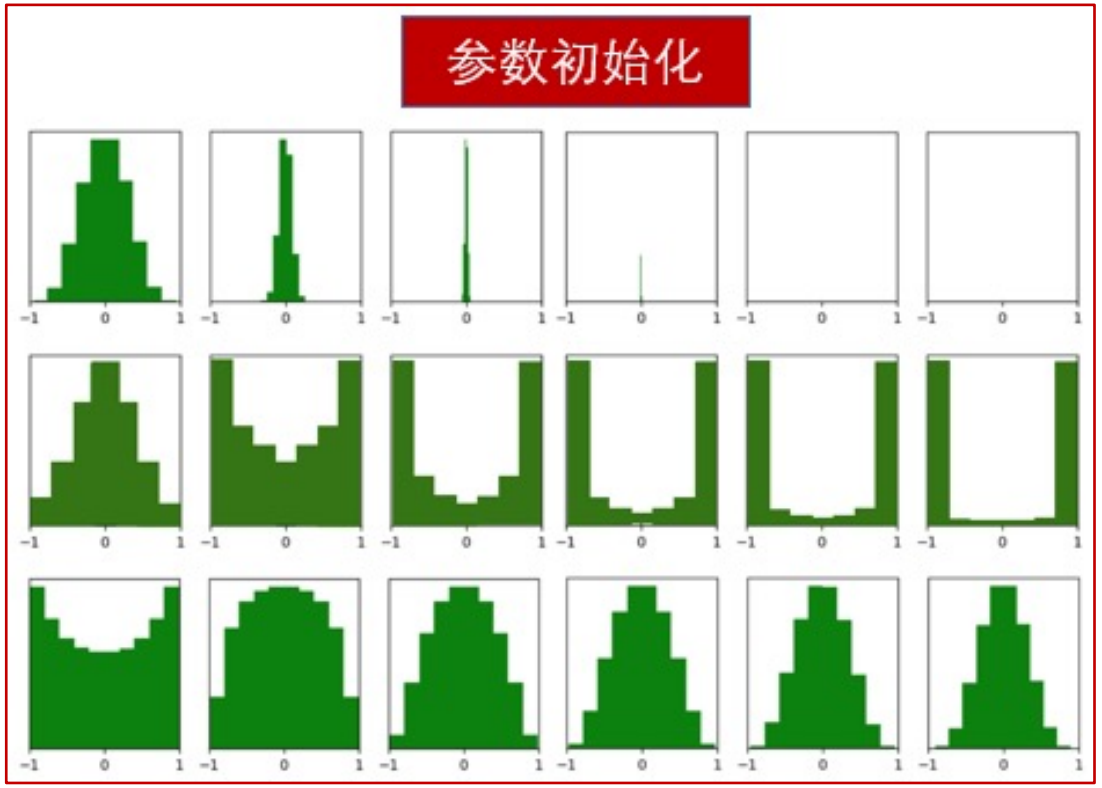
ELU

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



- 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）

数据预处理



L06：神经网络的训练1

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

训练时

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

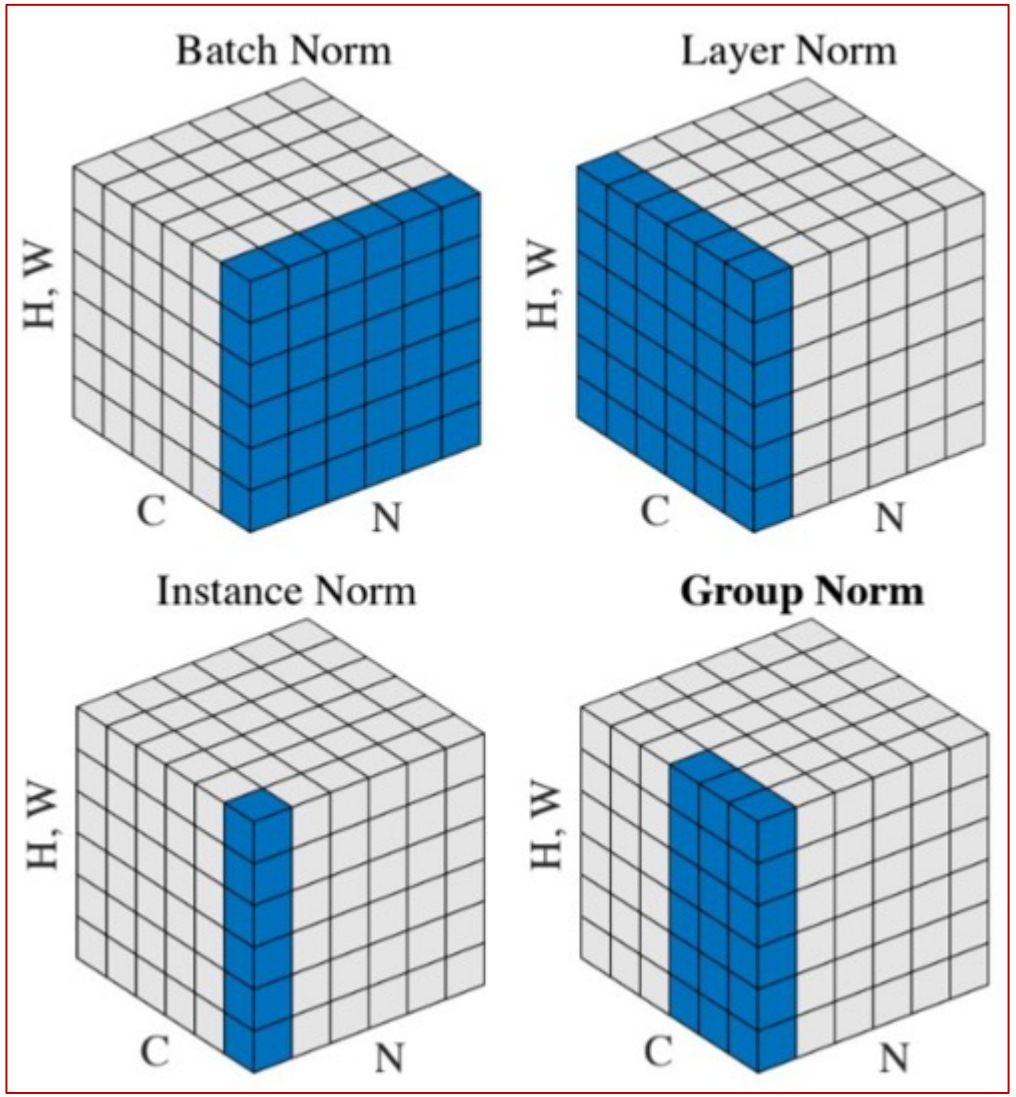
使用训练时 μ 的平均来估计 μ_j：$\mu_j = \mathbb{E}(\mu_j^{(b)})$ 1×D

推理时

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



神经网络的训练2

- 优化方法的演进
- 学习率的设置
- 正则化
- 超参数的选择

优化方法的演进

- Stochastic Gradient Descent（SGD）
- Momentum
- AdaGrad
- Adam（现在常用）

随机梯度下降

- 随机梯度下降（Stochastic Gradient Descent，SGD）
  - ✓每次使用一个数据（图像）计算loss
  - ✓计算梯度并更新参数

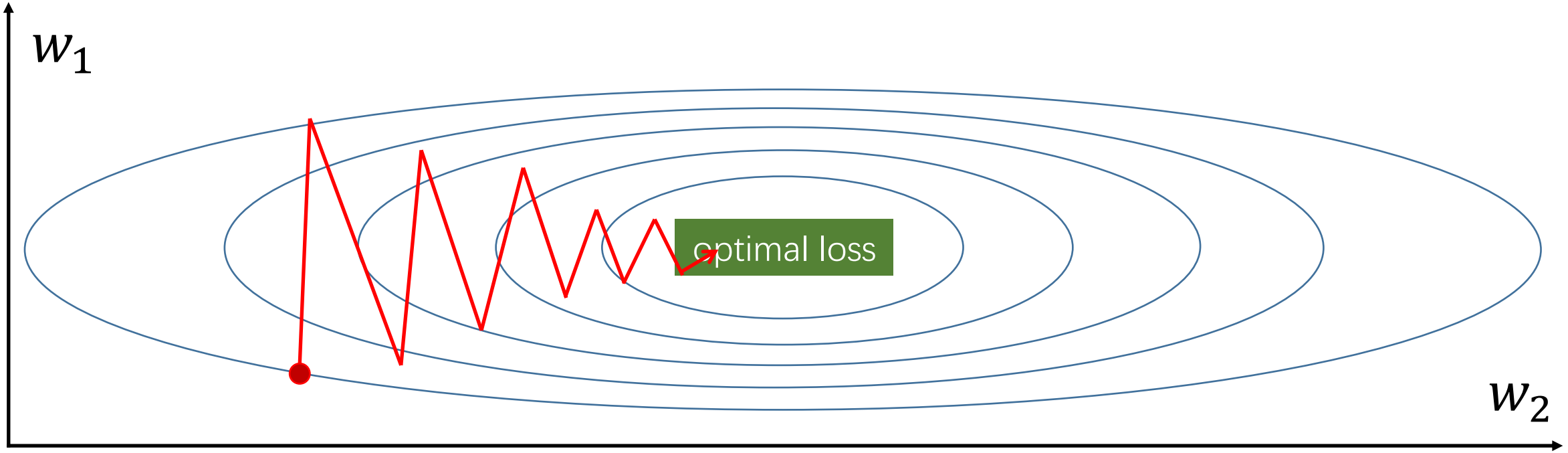
```
# Vanilla Gradient Descent

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

根据当前图像计算梯度

更新参数

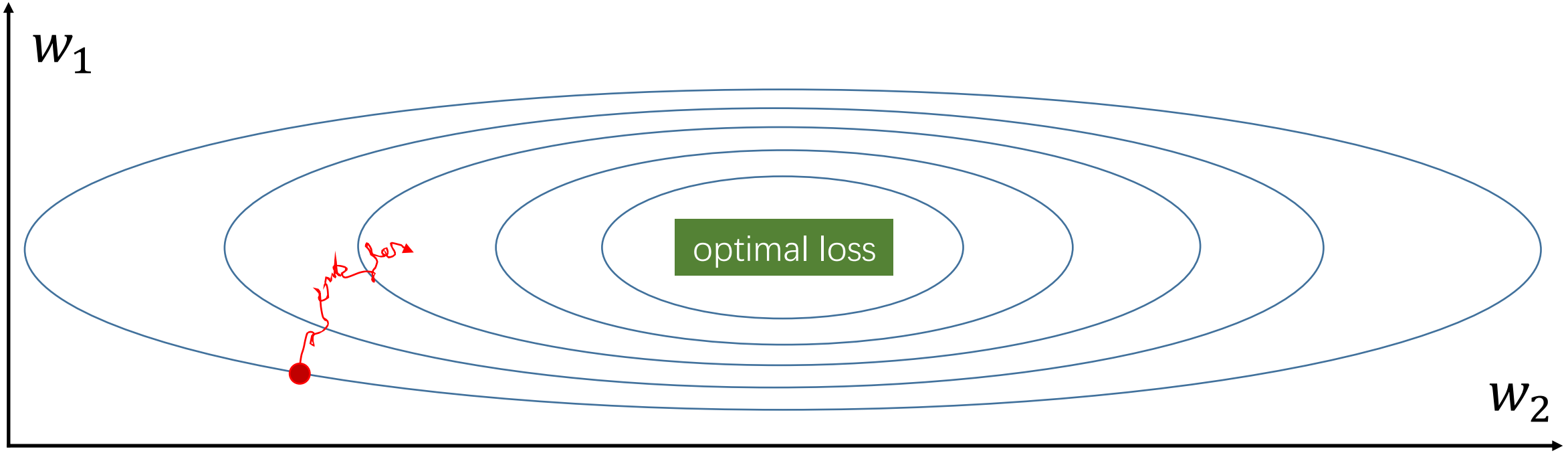
SGD的缺点：zig-zag问题

- 假设在二维空间，损失函数对参数w_1比较敏感，但是对w_2不敏感



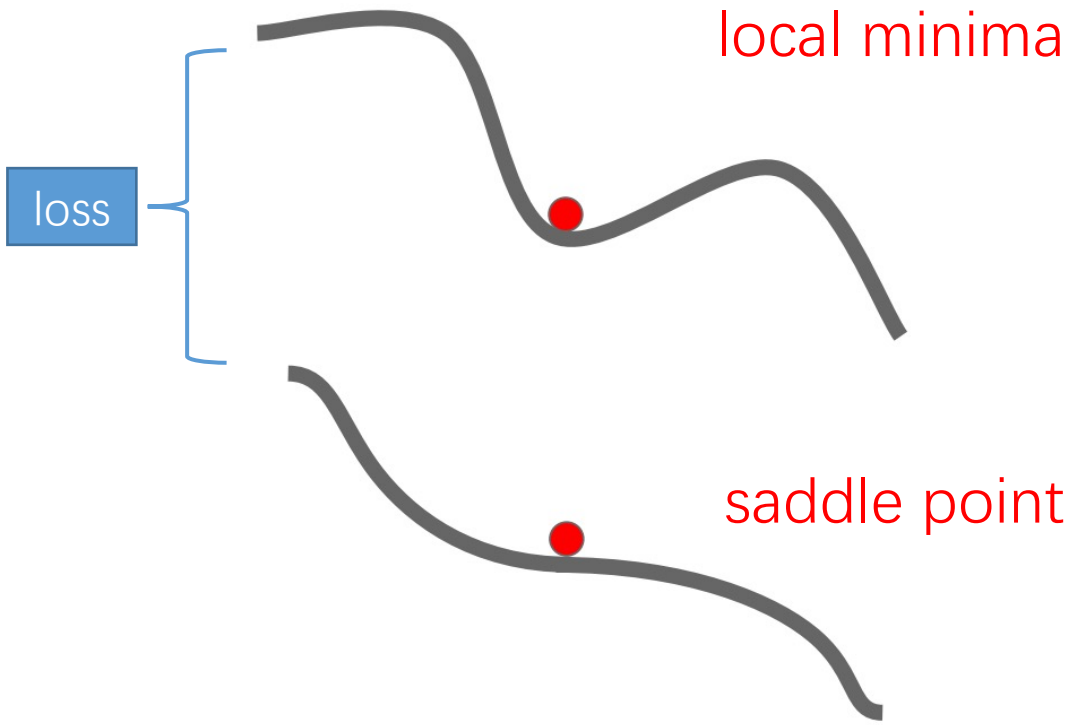
SGD的缺点：minibatch的噪声

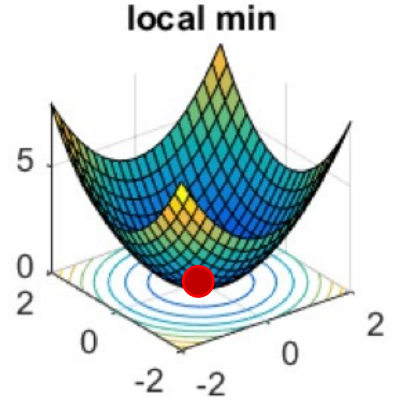
- 使用minibatch更新梯度，噪声较大

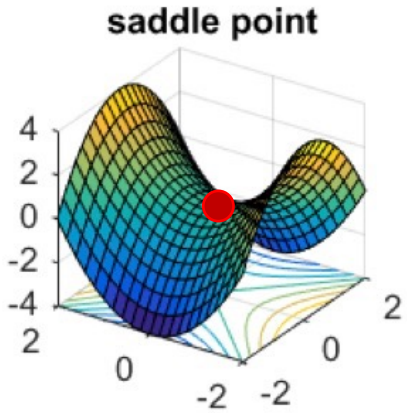


SGD的缺点：局部最优和鞍点

- 一旦陷入local minima或者saddle point，参数很难继续更新

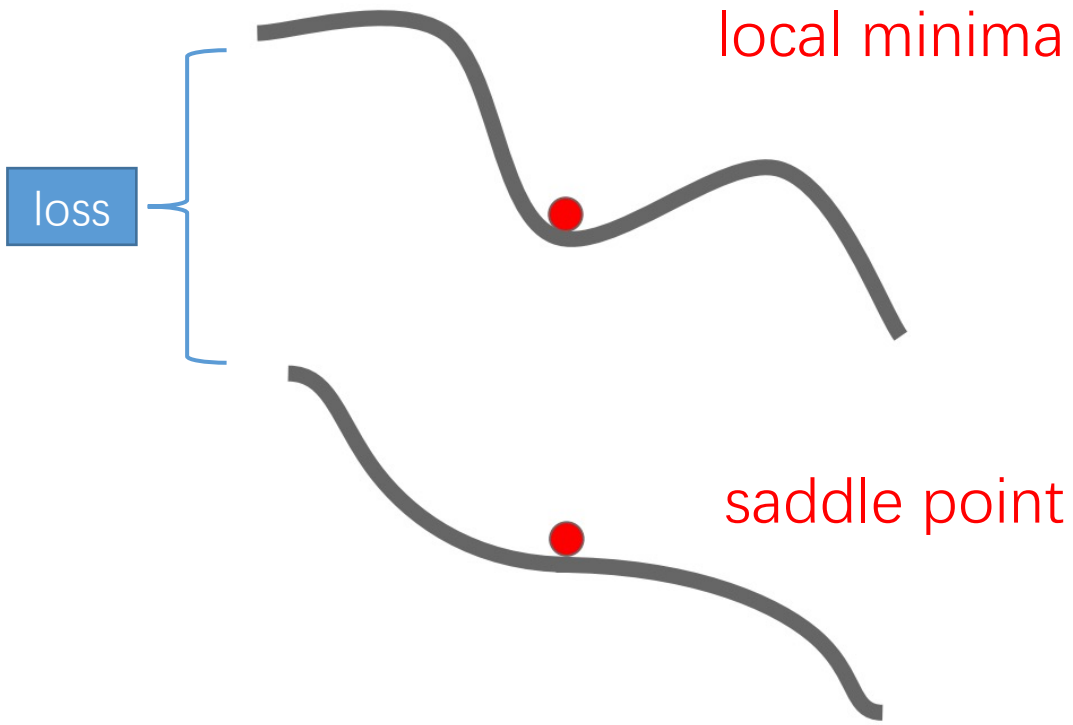






SGD的缺点：局部最优和鞍点

- 一旦陷入local minima或者saddle point，参数很难继续更新



✓ 这两种情况下梯度接近0，参数很难通过SGD来更新

✓ saddle points在高维空间几乎到处都是！

部分方向loss变大，其余方向loss变小

Momentum

SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

```
while True:
  dx = compute_gradient(x)
  x -= learning_rate * dx
```

Momentum

SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

```
while True:
  dx = compute_gradient(x)
  x -= learning_rate * dx
```

SGD+Momentum

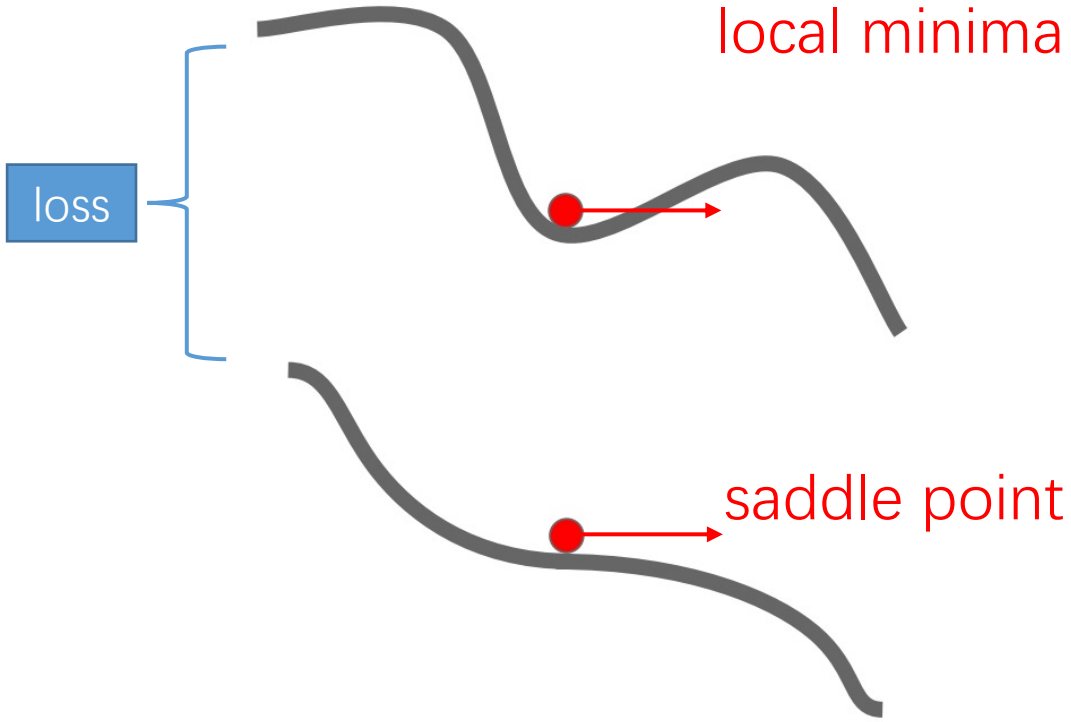
$$v_{t+1} = \rho v_t + \nabla f(x_t)$$
$$x_{t+1} = x_t - \alpha v_{t+1}$$

```
vx = 0
while True:
  dx = compute_gradient(x)
  vx = rho * vx + dx
  x -= learning_rate * vx
```

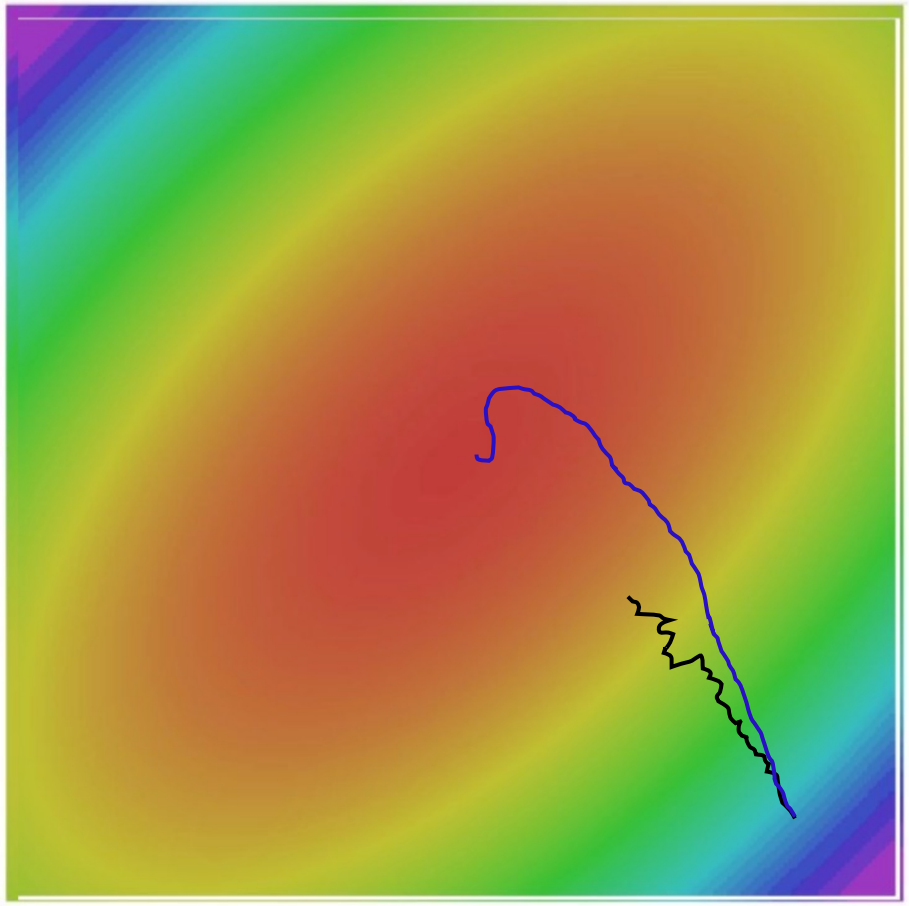
- ✓ 给梯度加上“速度”：$\nabla f(x)$(即dx)越大，速度增量越大
- ✓ “速度”等于梯度的moving average：缓解zig-zag和噪声问题
- ✓ 衰减因子ρ：扮演摩擦力的角色，一般设为0.9或0.99

Sutskever I, Martens J, Dahl G, Hinton G. On the importance of initialization and momentum in deep learning. ICML 2013.

Momentum



假设颜色越红，loss越小

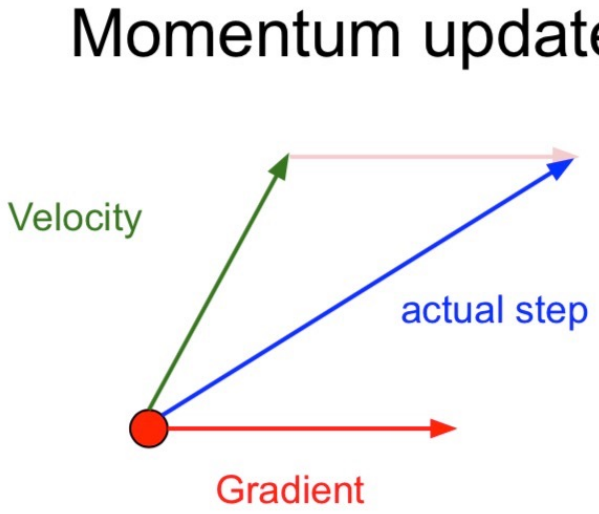


SGD SGD+Momentum

Sutskever I, Martens J, Dahl G, Hinton G. On the importance of initialization and momentum in deep learning. ICML 2013.

Nesterov Momentum

根据当前点的速度和梯度更新速度，然后更新参数



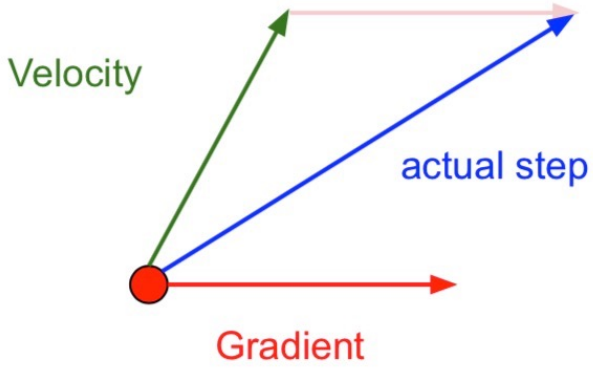
$$v_{t+1} = \rho v_t + \nabla f(x_t)$$

$$x_{t+1} = x_t - \alpha v_{t+1}$$

Nesterov Momentum

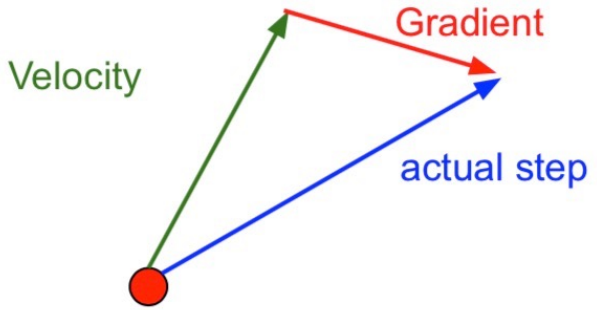
Momentum update:

根据当前点的速度和梯度更新速度，然后更新参数



$$v_{t+1} = \rho v_t + \nabla f(x_t)$$
$$x_{t+1} = x_t - \alpha v_{t+1}$$

Nesterov Momentum



根据当前点的速度和下一个点的梯度更新速度，然后更新参数

$$v_{t+1} = \rho v_t - \alpha \nabla f(x_t + \rho v_t)$$
$$x_{t+1} = x_t + v_{t+1}$$

Nesterov YE. A method for solving the convex programming problem with convergence rate O (1/k^ 2). 1983.

Sutskever I, Martens J, Dahl G, Hinton G. On the importance of initialization and momentum in deep learning. ICML 2013.

Nesterov Momentum

- 计算方法

$$v_{t+1} = \rho v_t - \alpha \nabla f(x_t + \rho v_t)$$

$$x_{t+1} = x_t + v_{t+1}$$

令$\tilde{x}_t = x_t + \rho v_t$

$$v_{t+1} = \rho v_t - \alpha \nabla f(\tilde{x}_t)$$

$$\begin{aligned}\tilde{x}_{t+1} &= \tilde{x}_t - \rho v_t + (1+\rho) v_{t+1} \\ &= \tilde{x}_t + v_{t+1} + \rho(v_{t+1} - v_t)\end{aligned}$$

Nesterov Momentum

- 计算方法

$$v_{t+1} = \rho v_t - \alpha \nabla f(x_t + \rho v_t)$$

$$x_{t+1} = x_t + v_{t+1}$$

令$\tilde{x}_t = x_t + \rho v_t$

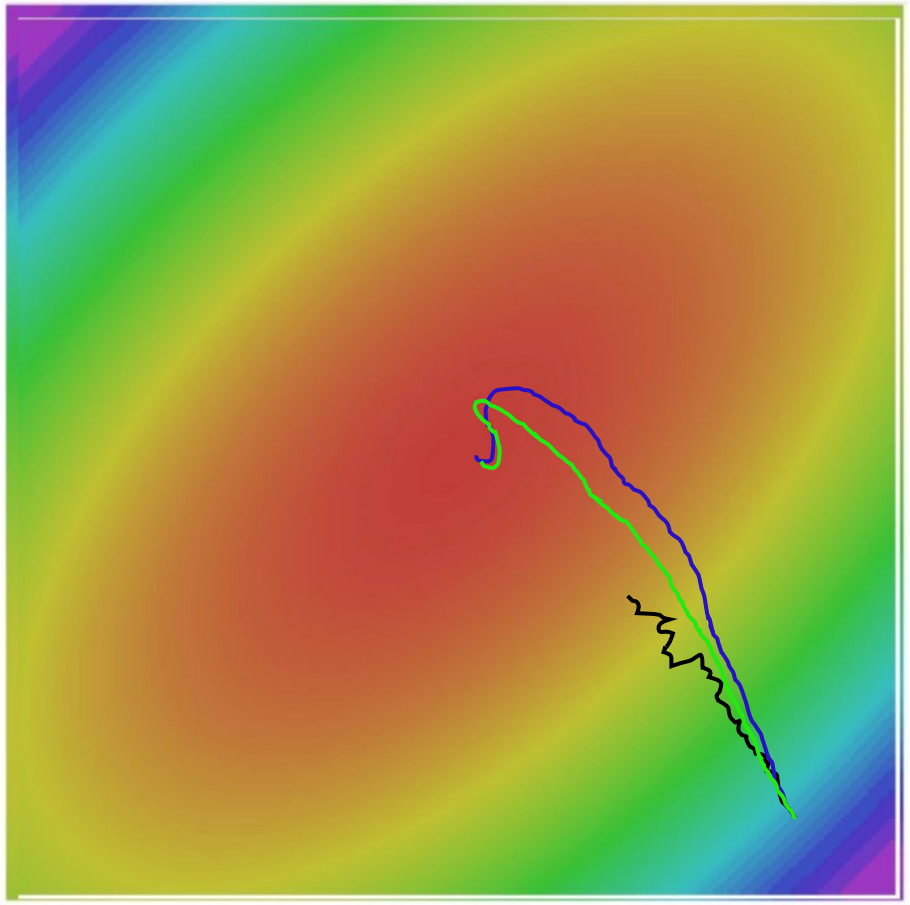
$$v_{t+1} = \rho v_t - \alpha \nabla f(\tilde{x}_t)$$

$$\tilde{x}_{t+1} = \tilde{x}_t - \rho v_t + (1+\rho) v_{t+1}$$

$$= \tilde{x}_t + v_{t+1} + \rho(v_{t+1} - v_t)$$

```
dx = compute_gradient(x)
old_v = v
v = rho * v - learning_rate * dx
x += -rho * old_v + (1 + rho) * v
```

Nesterov Momentum



SGD

SGD+Momentum

Nesterov Momentum

AdaGrad

1. $r_t = r_{t-1} + \nabla f(x_t) \times \nabla f(x_t)$
2. $\alpha_t = \frac{\alpha}{\sqrt{r_t} + \delta}$
3. $x_{t+1} = x_t - \alpha_t \nabla f(x_t)$

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared += dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

Duchi J, Hazan E, Singer Y. Adaptive subgradient methods for online learning and stochastic optimization. JMLR 2011

AdaGrad

1. $r_t = r_{t-1} + \nabla f(x_t) \times \nabla f(x_t)$
2. $\alpha_t = \frac{\alpha}{\sqrt{r_t} + \delta}$
3. $x_{t+1} = x_t - \alpha_t \nabla f(x_t)$

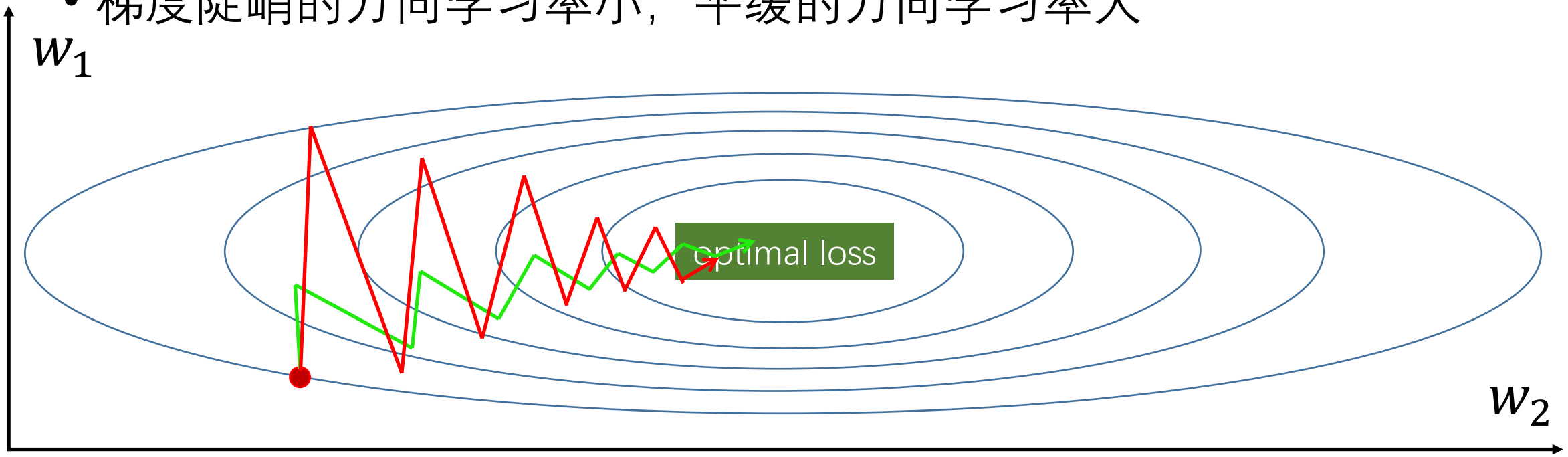
```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared += dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

- ✓ 对权重参数的每一维累积梯度的平方
- ✓ 用累积梯度平方根scale全局学习率，使得每个维度的学习率不同（adaptive learning rate）
- ✓ δ用以防止除零

Duchi J, Hazan E, Singer Y. Adaptive subgradient methods for online learning and stochastic optimization. JMLR 2011

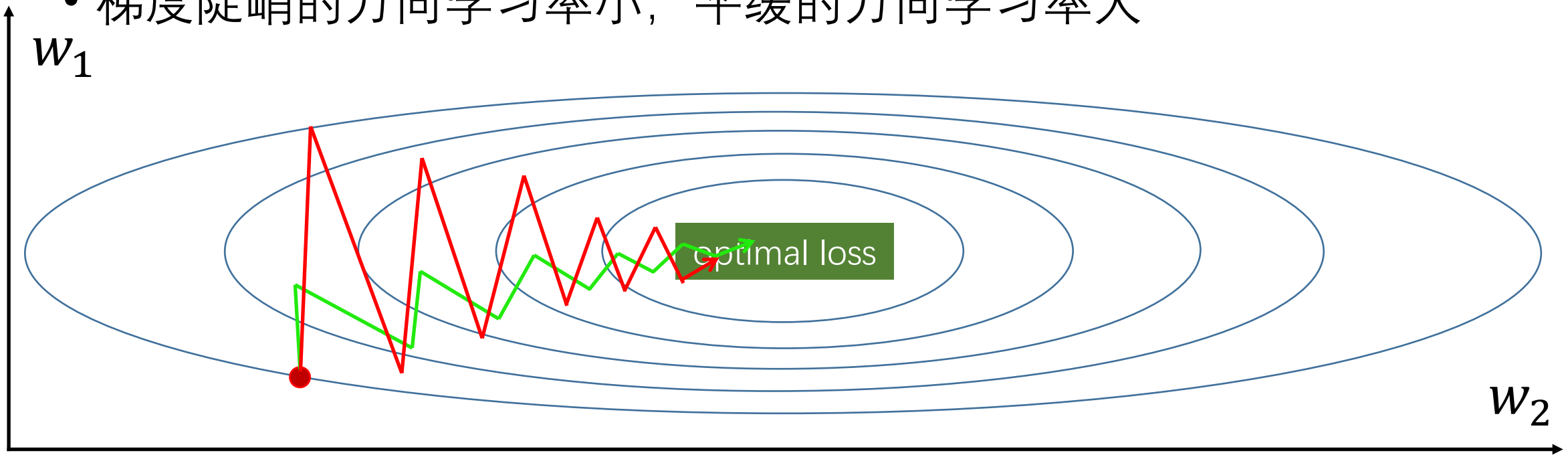
AdaGrad

- 梯度陡峭的方向学习率小，平缓的方向学习率大



AdaGrad

- 梯度陡峭的方向学习率小，平缓的方向学习率大



✓ 一段时间之后，学习率可能会趋近于0，有可能导致在saddle point附近参数无法更新

RMSProp（Leaky AdaGrad）

AdaGrad

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared += dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

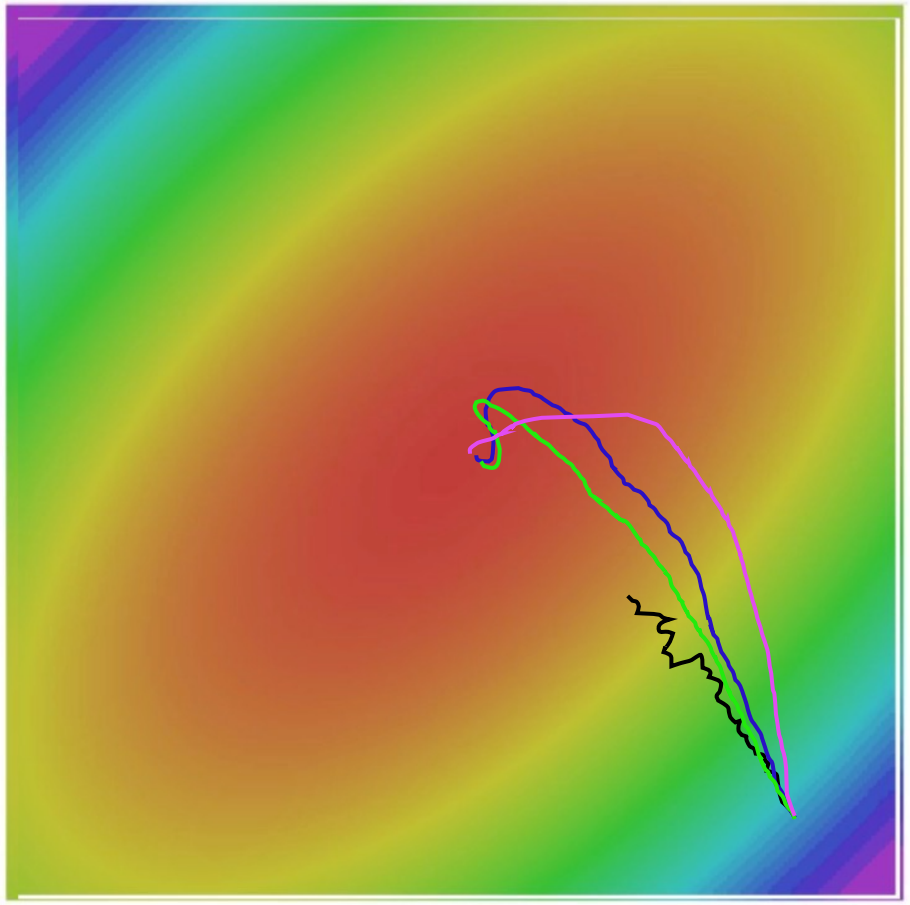
对梯度的平方做momentum

RMSProp

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared = decay_rate * grad_squared + (1 - decay_rate) * dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

Hinton G, Srivastava N, Swersky K. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. Lecture Notes 2012.

RMSProp（Leaky AdaGrad）



SGD

SGD+Momentum

Nesterov Momentum

RMSProp

Adam：Adaptive Moment estimation

- 综合momentum和AdaGrad的思路

```
first_moment = 0
second_moment = 0
for t in range(1, num_iterations):
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment  + (1 - beta1) * dx
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx

  x -= learning_rate * first_moment / (np.sqrt(second_moment) + 1e-7))
```

momentum

AdaGrad/RMSProp

Kingma DP, Ba J. Adam: A method for stochastic optimization. ICLR 2015.

Adam：Adaptive Moment estimation

- 综合momentum和AdaGrad的思路

```
first_moment = 0
second_moment = 0
for t in range(1, num_iterations):
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment  + (1 - beta1) * dx
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
  first_unbias = first_moment / (1 - beta1 ** t)
  second_unbias = second_moment / (1 - beta2 ** t)
  x -= learning_rate * first_unbias / (np.sqrt(second_unbias) + 1e-7))
```

常用初始设置：
- ✓ beta1=0.9
- ✓ beta2=0.999
- ✓ learning_rate= 1e-3 or 5e-4

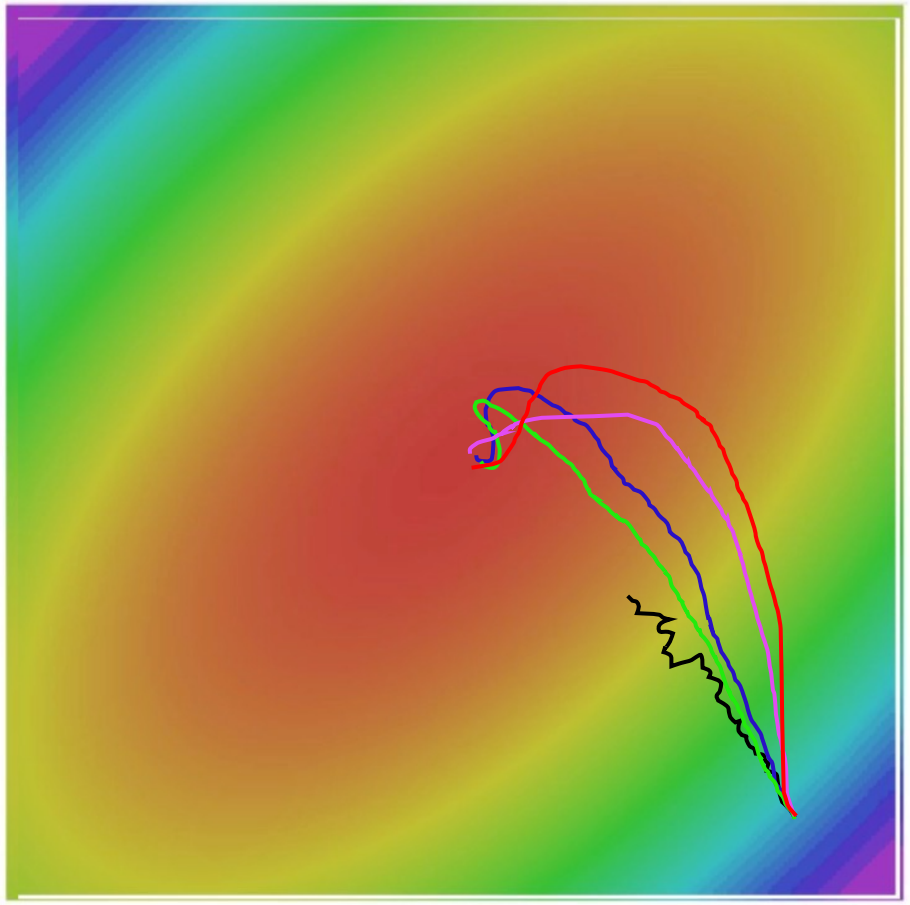
momentum

AdaGrad/RMSProp

bias correction: 防止训练初期学习率过大

Kingma DP, Ba J. Adam: A method for stochastic optimization. ICLR 2015.

Adam



SGD

SGD+Momentum

Nesterov Momentum

RMSProp

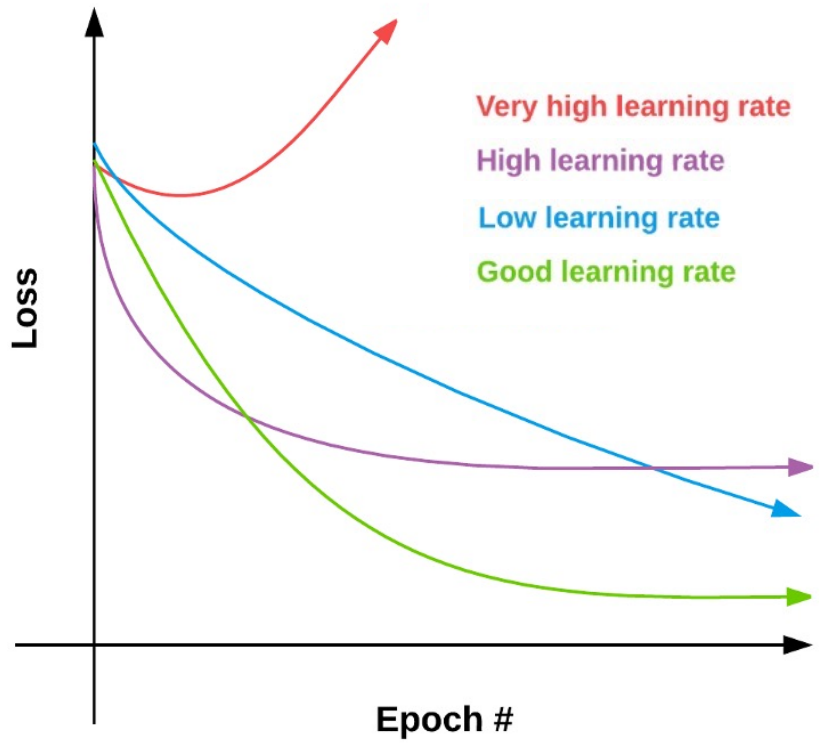
Adam

优化方法的演进

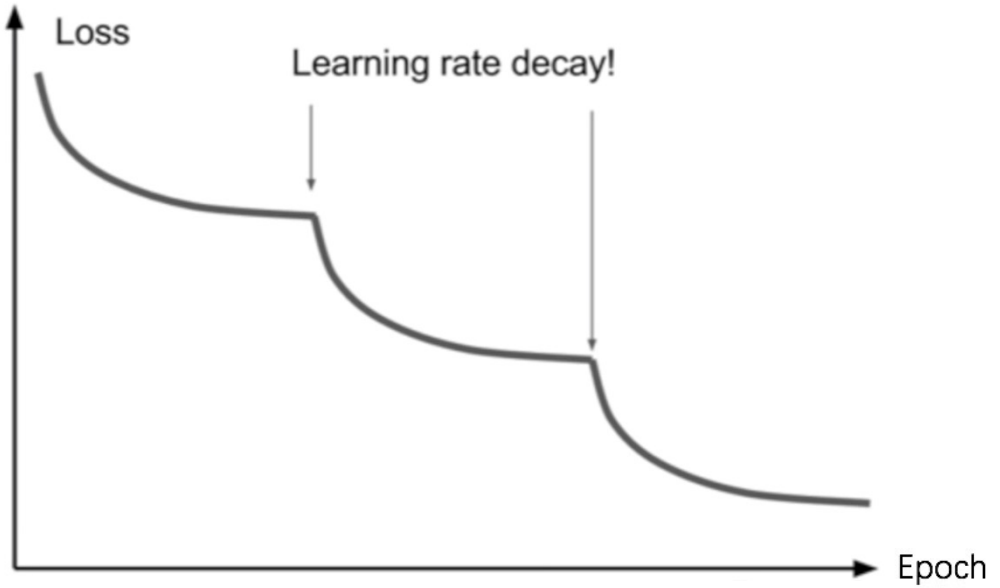
- AdaGrad
  - ✓ Duchi J, Hazan E, Singer Y. Adaptive subgradient methods for online learning and stochastic optimization. JMLR 2011.
- RMSProp
  - ✓ Hinton G, Srivastava N, Swersky K. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. Lecture Notes 2012.
- SGD+Momentum, Nesterov Momentum
  - ✓ Sutskever I, Martens J, Dahl G, Hinton G. On the importance of initialization and momentum in deep learning. ICML 2013.
- Adam
  - ✓ Kingma DP, Ba J. Adam: A method for stochastic optimization. ICLR 2015.

学习率（learning rate）

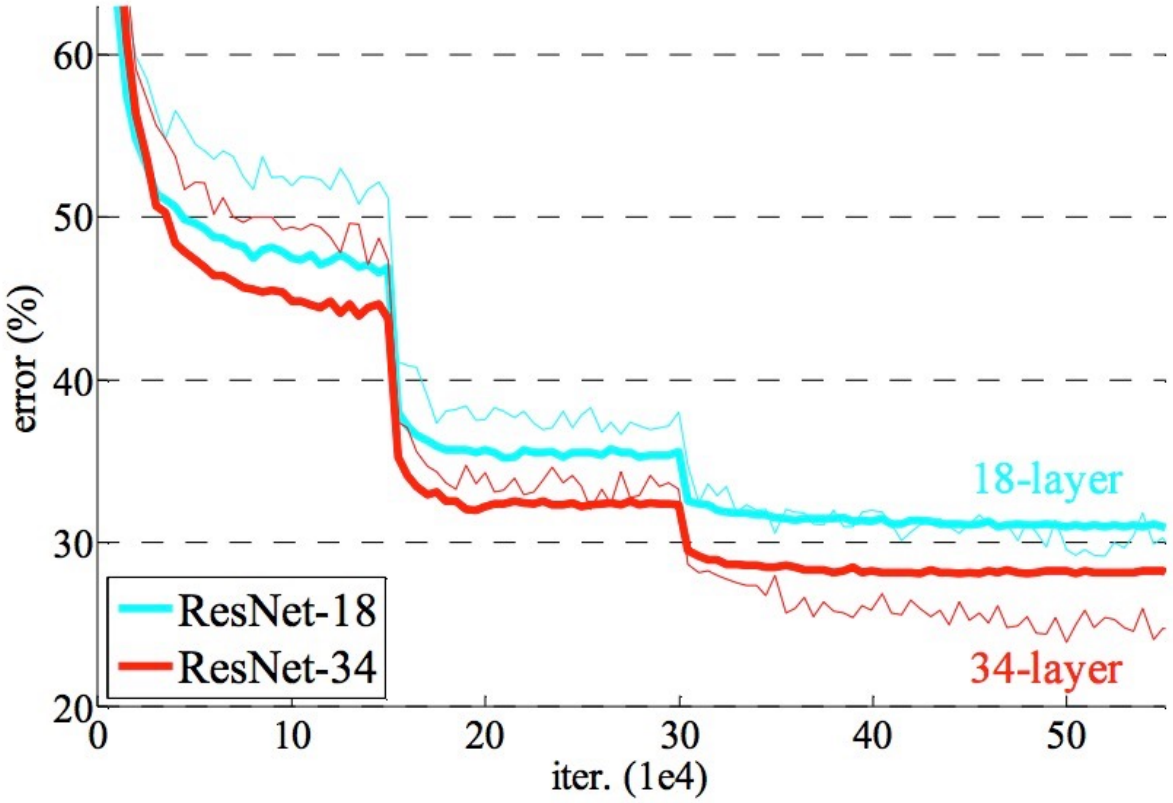
- 前述优化方法包含超参数：学习率



- learning rate schedule：在训练过程中调整学习率大小
- 常用设置：使用逐渐衰减（decay）的学习率
  - ✓ step decay：每训练N个iteration/epoch除以常数
  - ✓ exponential decay：$\alpha = \alpha_0 e^{-kt}$
  - ✓ 1/t decay：$\alpha = \frac{\alpha_0}{1+kt}$



ResNets

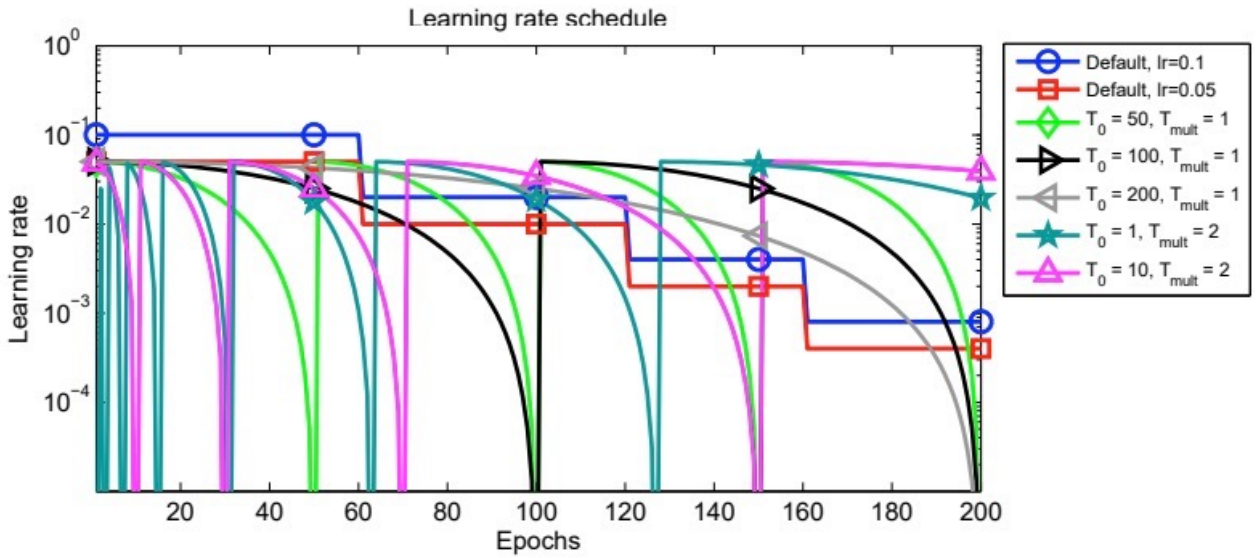


- ✓ 一开始设置learning_rate=0.1
- ✓ 在第32k和第48k个iteration，分别除以10
- ✓ 在第64k个iteration，训练终止

step decay

He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. CVPR 2016.

SGDR：学习率周期性热重启，然后衰减



$$\eta_t = \eta_{min}^i + \frac{1}{2}(\eta_{max}^i - \eta_{min}^i)(1 + \cos(\frac{T_{cur}}{T_i}\pi)),$$

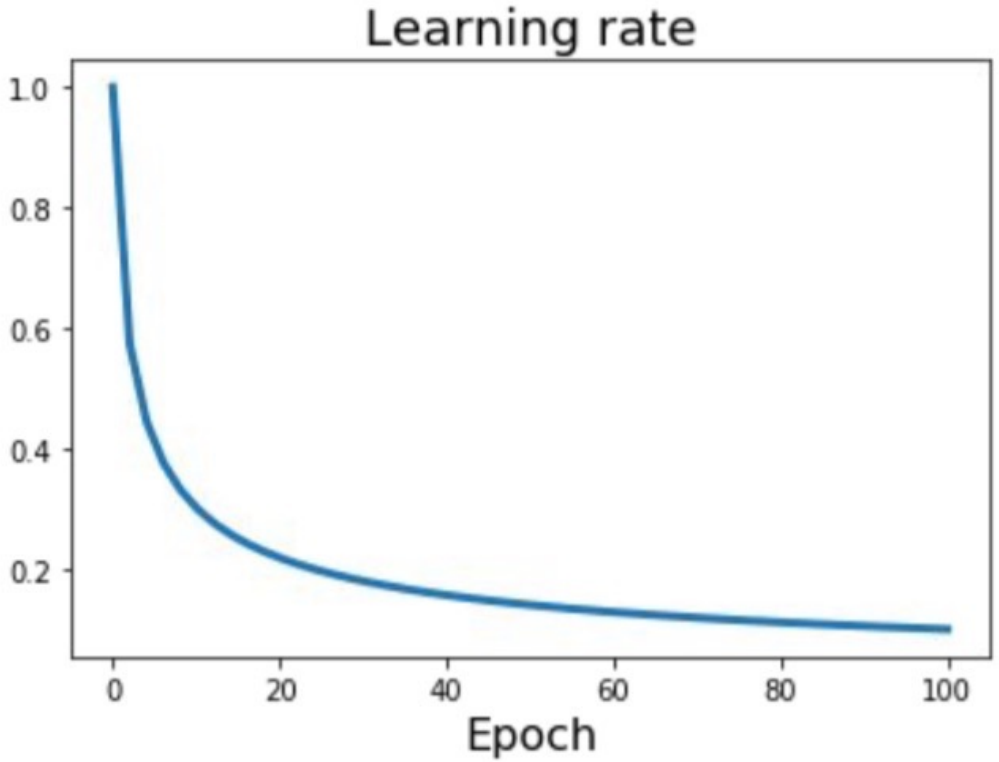
$$\alpha_t = \frac{1}{2}\alpha_0 \left(1 + \cos(t\pi/T)\right)$$

cosine decay

- ✓ α_0: 初始学习率
- ✓ α_t: 第t个epoch或者iteration的学习率
- ✓ T: 总的epoch或者iteration个数

Loshchilov I, Hutter F. Sgdr: Stochastic gradient descent with warm restarts. ICLR 2017.

Transformer



$$lrate = d_{\text{model}}^{-0.5} \cdot \min(step\_num^{-0.5}, step\_num \cdot warmup\_steps^{-1.5})$$

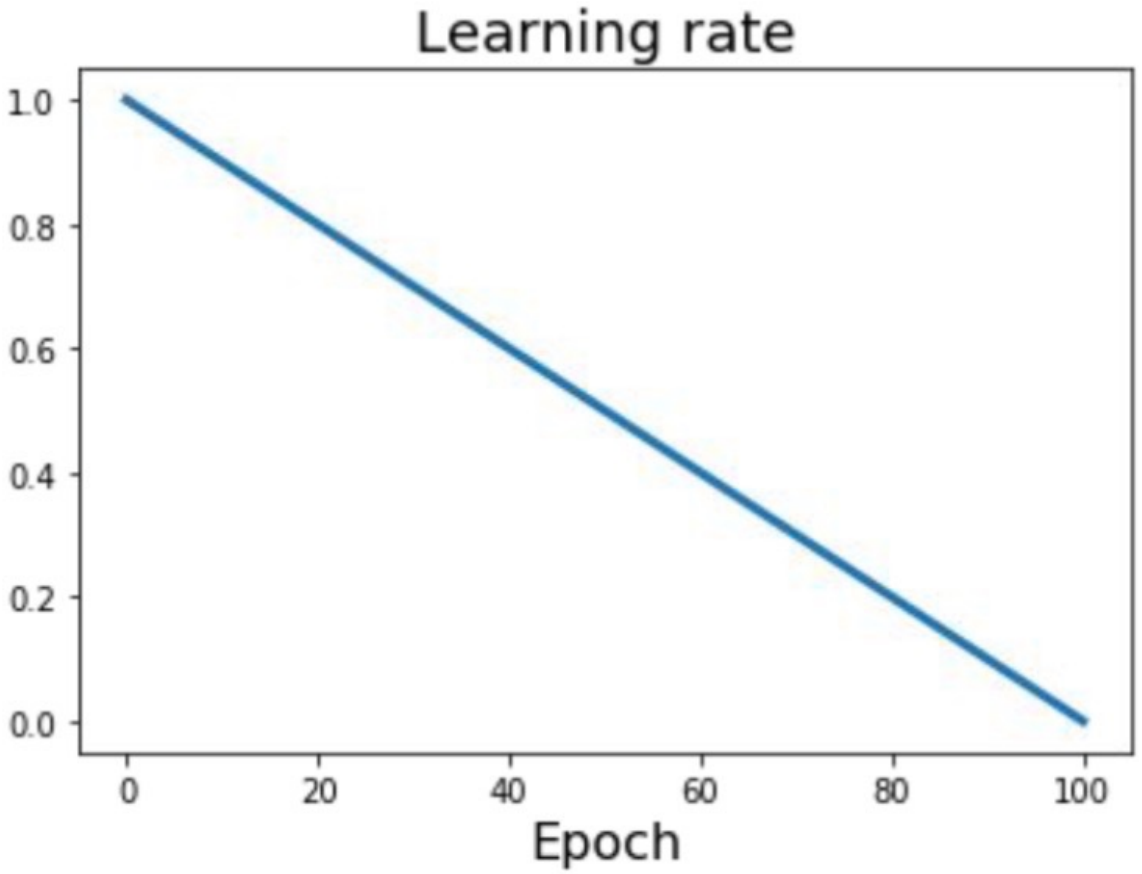
$$\alpha_t = \alpha_0 / \sqrt{t}$$

inverse square root decay

- ✓ α_0: 初始学习率
- ✓ α_t: 第t个epoch或者iteration的学习率
- ✓ T: 总的epoch或者iteration个数

Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need. NIPS 2017.

BERT



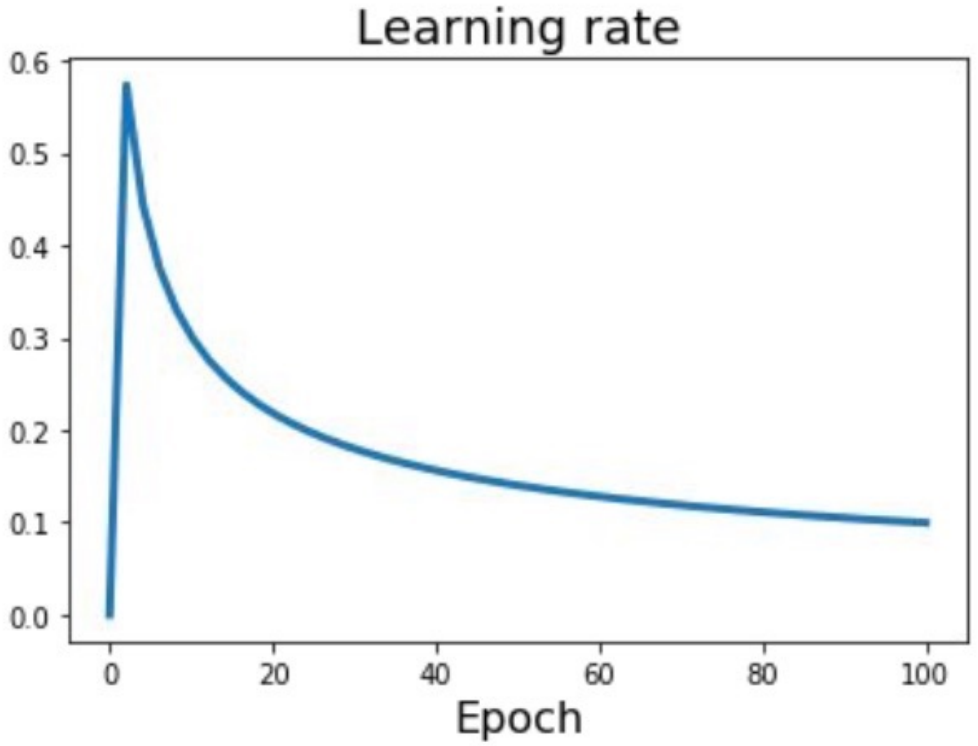
$$\alpha_t = \alpha_0(1 - t/T)$$

linear decay

- ✓ α_0: 初始学习率
- ✓ α_t: 第t个epoch或者iteration的学习率
- ✓ T: 总的epoch或者iteration个数

Devlin J, Chang MW, Lee K, Toutanova K. Bert: Pre-training of deep bidirectional transformers for language understanding. 2018.

初始学习率：Linear Warmup



Transformer =4000

$$lrate = d_{\text{model}}^{-0.5} \cdot \min(step\_num^{-0.5}, step\_num \cdot warmup\_steps^{-1.5})$$

$$\alpha_t = \alpha_0 \times t/n$$

- ✓ n：warmup period/iterations
- ✓ $t \in [1, n]$

- ➢ 初始学习率过高可能会导致损失不断增大
- ➢ Linear warmup：在前几千个iteration，从0开始线性增大学习率

Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need. NIPS 2017.

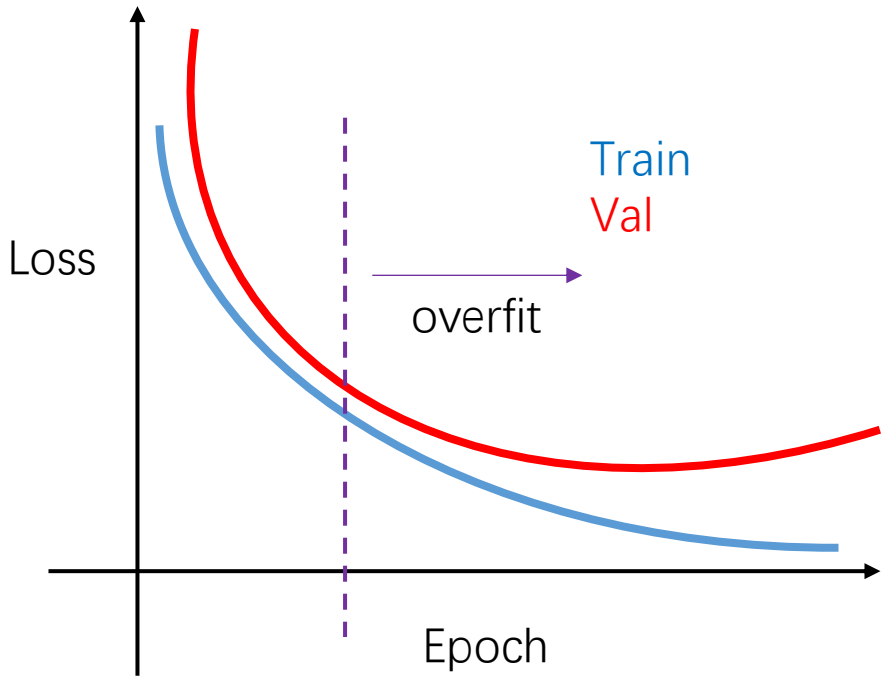
Goyal P, Dollár P, Girshick R, Noordhuis P, Wesolowski L, Kyrola A, Tulloch A, Jia Y, He K. Accurate, large minibatch sgd: Training imagenet in 1 hour. 2017.

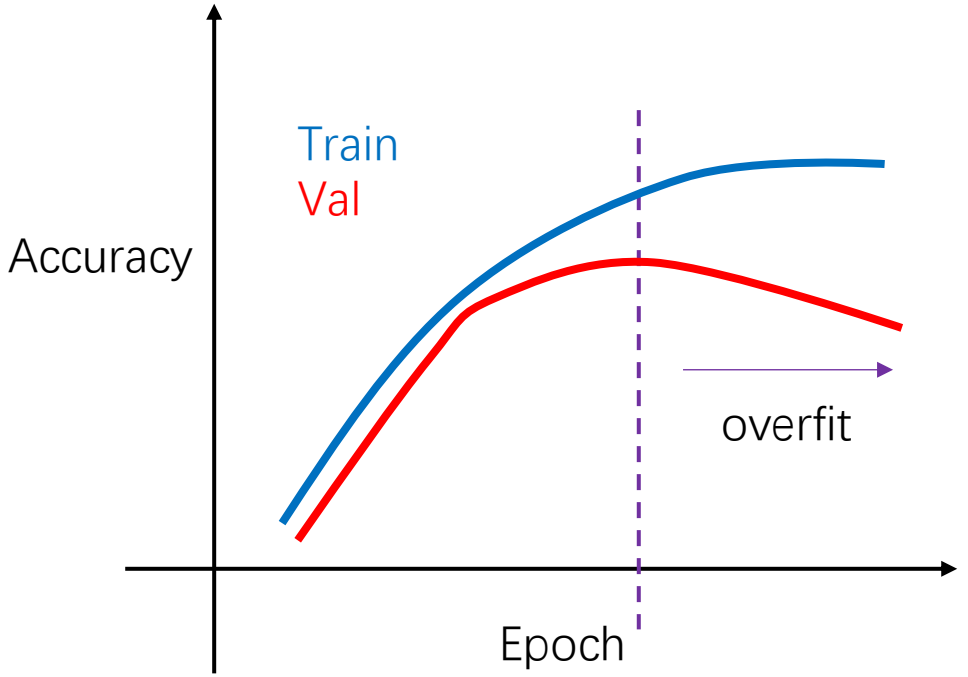
实际模型中。。。

- 首选Adam作为优化器

- 然后尝试SGD+Momentum
  - ✓最终性能可能会更好，但是需要调整学习率和schedule

正则化：防止overfitting





正则化

$$L(\boldsymbol{W}) = \frac{1}{N}\sum_{i=1}^{N} l(f(\boldsymbol{W}, \boldsymbol{x}_i), y_i) + \lambda R(\boldsymbol{W})$$

正则项
✓ 防止模型过度拟合训练集

超参数，正则化强度

数据损失
✓ 使得模型尽可能拟合训练集

L1: $\sum_k \sum_l |w_{k,l}|$
L2: $\sum_k \sum_l w_{k,l}^2$
L1+L2(elastic net): $\sum_k \sum_l |w_{k,l}| + \beta w_{k,l}^2$

正则化

正则项
✓ 防止模型过度拟合训练集

超参数，正则化强度

$$L(\boldsymbol{W}) = \frac{1}{N}\sum_{i=1}^{N} l(f(\boldsymbol{W}, \boldsymbol{x}_i), y_i) + \lambda R(\boldsymbol{W})$$

数据损失
✓ 使得模型尽可能拟合训练集

weight decay

L1: $\sum_k \sum_l |w_{k,l}|$

L2: $\sum_k \sum_l w_{k,l}^2$

L1+L2(elastic net): $\sum_k \sum_l |w_{k,l}| + \beta w_{k,l}^2$

$$L(\boldsymbol{W}) = f(\boldsymbol{W}) + \frac{\lambda}{2}\sum_k \sum_l w_{k,l}^2$$

$$\nabla_{kl} L(\boldsymbol{W}) = \nabla_{kl} f(\boldsymbol{W}) + \lambda w_{kl}$$

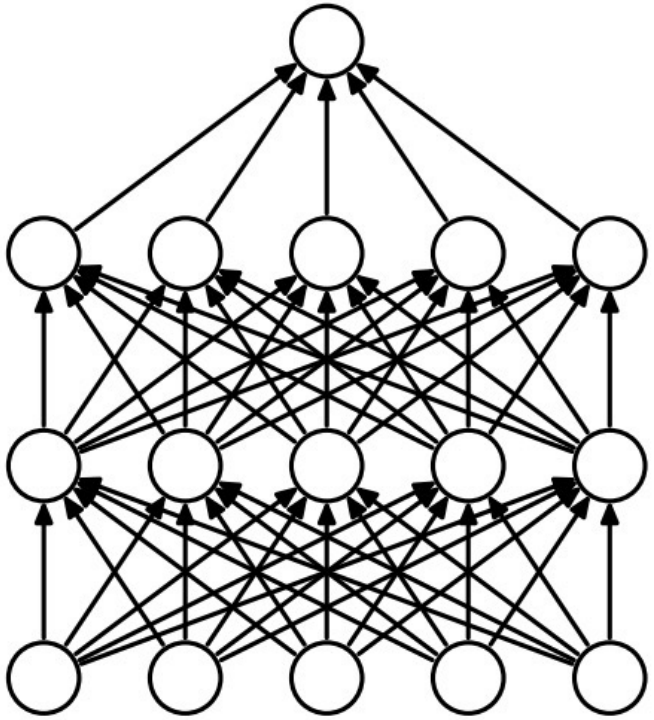
$$w'_{kl} = w_{kl} - \alpha \nabla_{kl} L(\boldsymbol{W}) = (1 - \alpha\lambda) w_{kl} - \alpha \nabla_{kl} f(\boldsymbol{W})$$

更多常用正则化

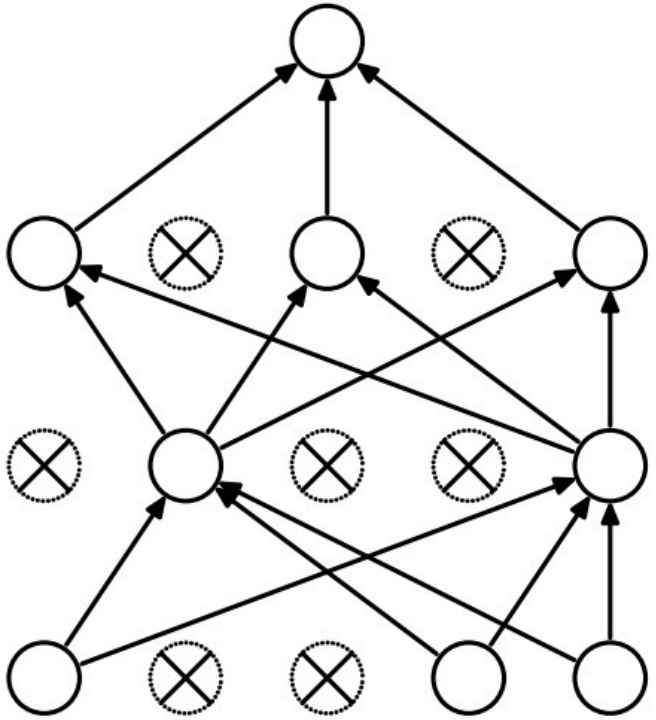
- Batch Normalization
- Dropout
- Data Augmentation
- Fractional pooling, DropConnect, Cutout, Mixup
- Early stopping, Model Ensembles

Dropout：减少特征的捕获

- 随机drop一些output
- drop的概率一般设为0.5（超参数）



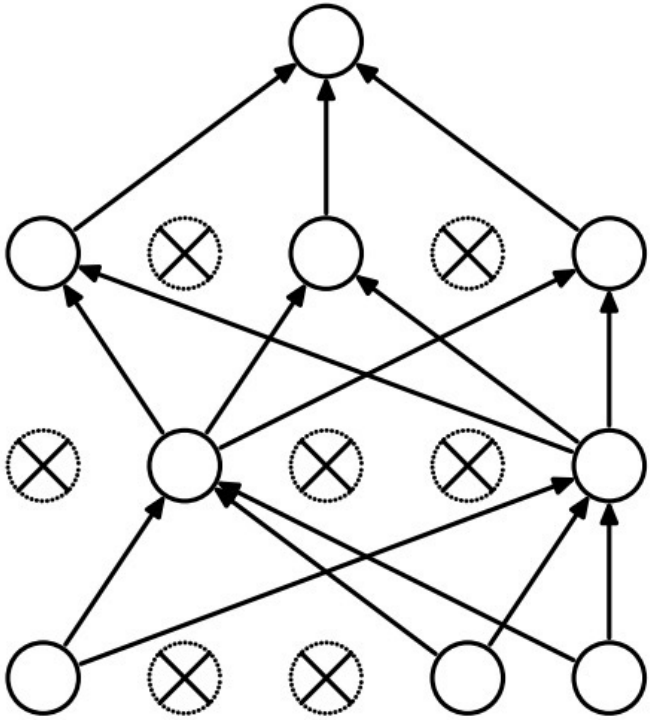
(a) Standard Neural Net



(b) After applying dropout.

Dropout

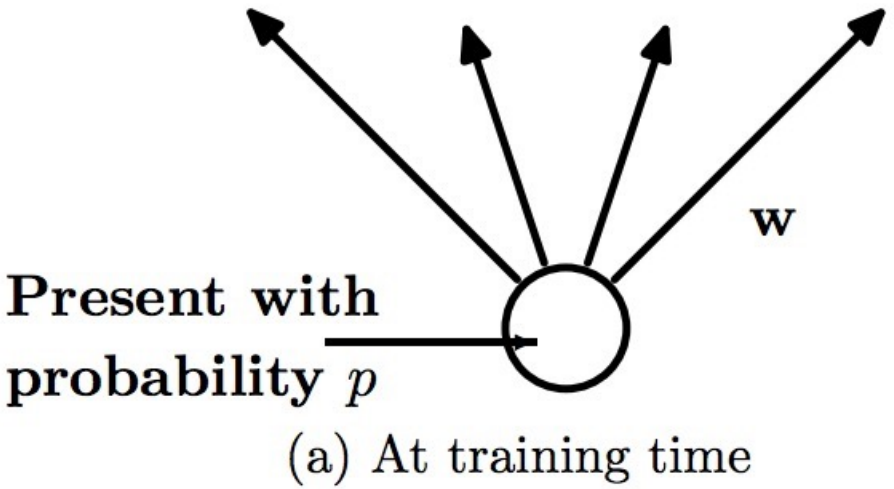
```
H1 = np.maximum(0, np.dot(W1, X) + b1)
U1 = np.random.rand(*H1.shape) < p # first dropout mask
H1 *= U1 # drop!
H2 = np.maximum(0, np.dot(W2, H1) + b2)
U2 = np.random.rand(*H2.shape) < p # second dropout mask
H2 *= U2 # drop!
out = np.dot(W3, H2) + b3
```



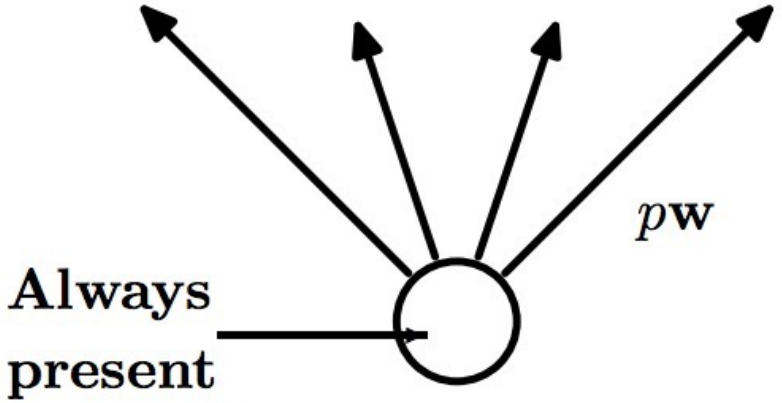
(b) After applying dropout.

Dropout：推理

- 将神经元被drop的概率平均到输出权重上
- 近似于计算训练时该神经元的期望输出



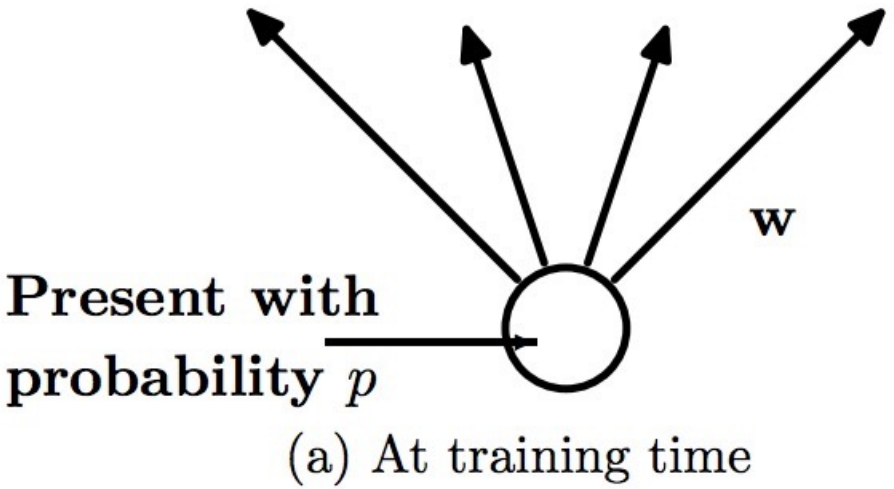
(a) At training time



(b) At test time

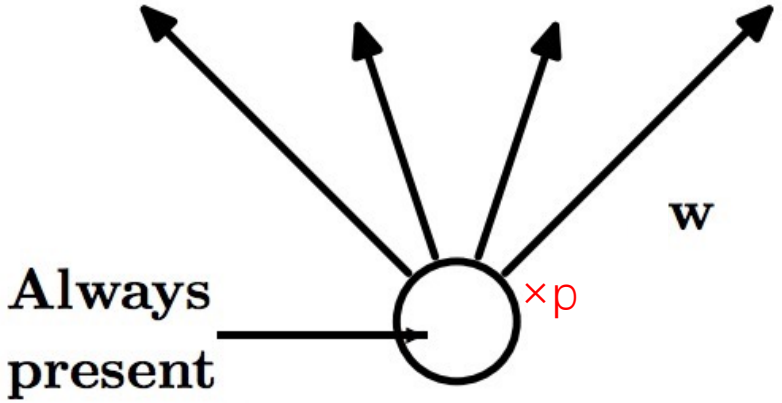
Dropout：推理

- 具体实现时，直接将output（activations）乘以dropout概率p



(a) At training time

```
H1 = np.maximum(0, np.dot(W1, X) + b1) * p
H2 = np.maximum(0, np.dot(W2, H1) + b2) * p
out = np.dot(W3, H2) + b3
```



(b) At test time

训练时：增加随机噪声
推理时：将噪声的影响平均化

Dropout：Inverted dropout

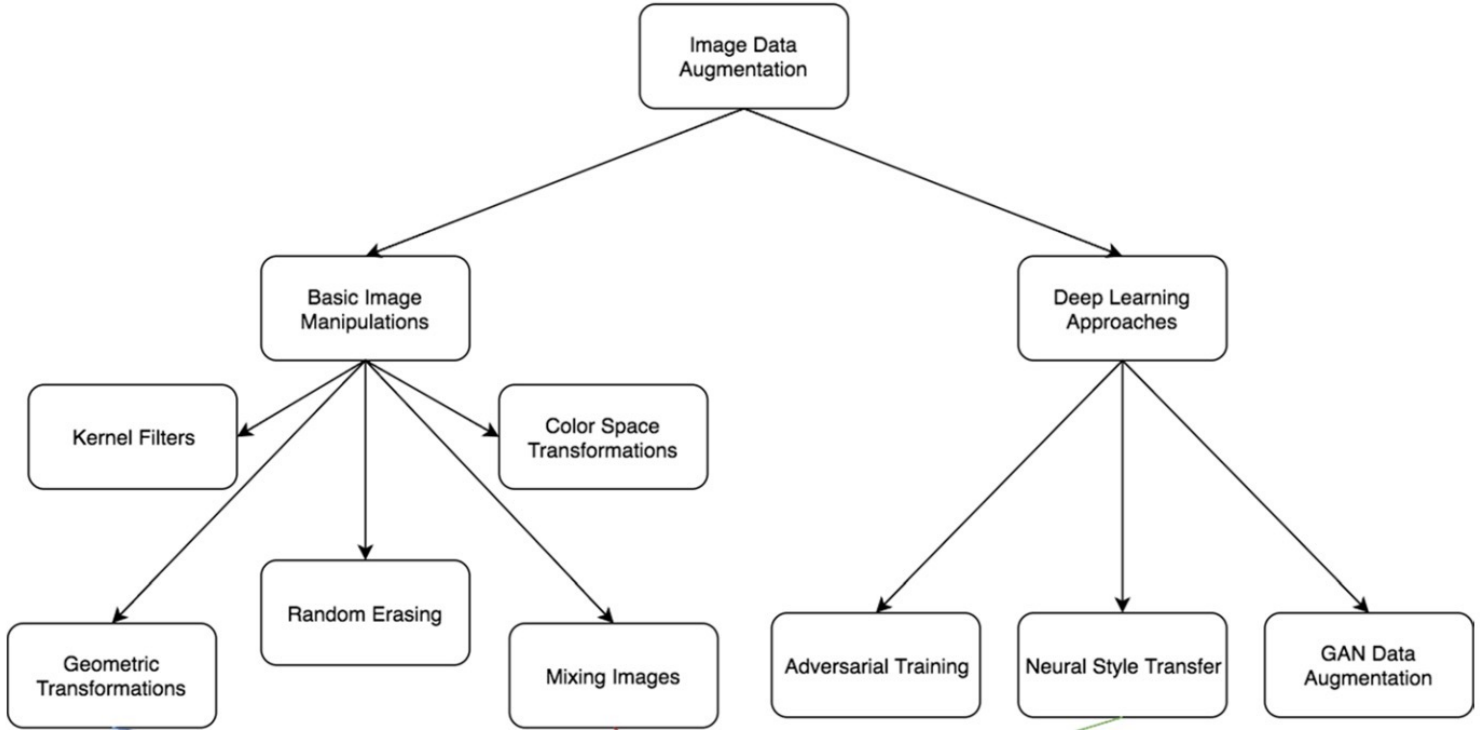
```python
p = 0.5 # probability of keeping a unit active. higher = less dropout

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
```

训练时

推理时

Data Augmentation：增加训练数据

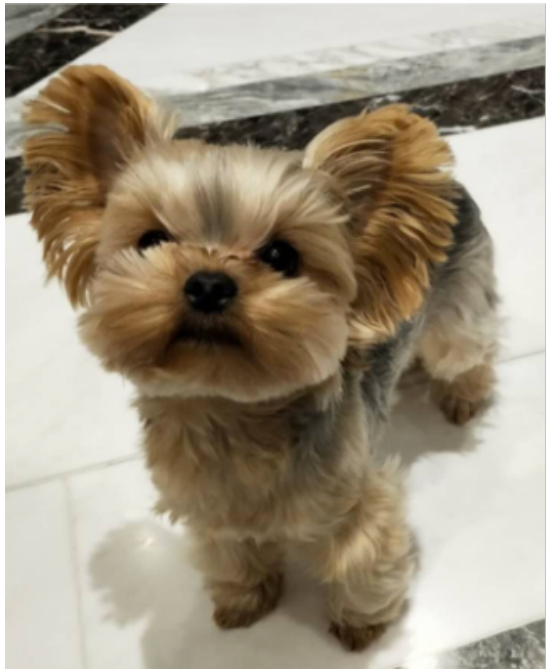
- A taxonomy of image data augmentations



Shorten C, Khoshgoftaar TM. A survey on image data augmentation for deep learning. Journal of Big Data. 2019.

Data Augmentation

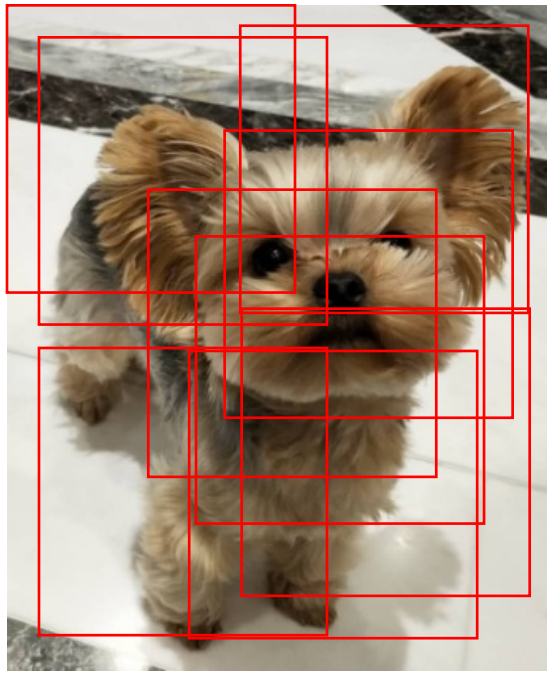
- （水平）翻转



Data Augmentation

- 裁剪和缩放（crops and scales）
  - ✓将短边缩放到固定大小L（e.g., 256）
  - ✓从图片中随机裁剪$l \times l$的图片（e.g., 224×224）

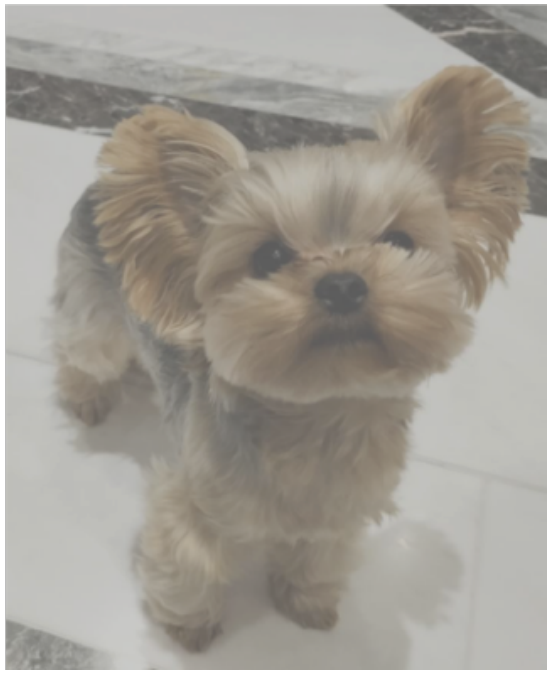
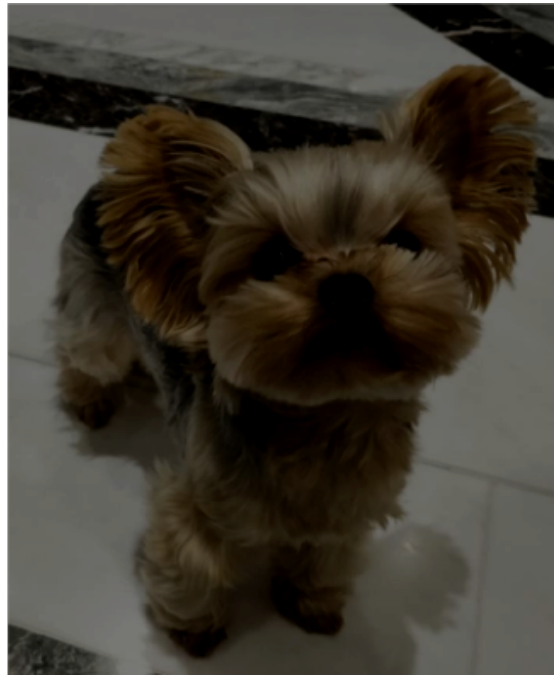
- 推理时
  - ✓平均多个crops的预测结果，e.g. 中心+四角，然后翻转



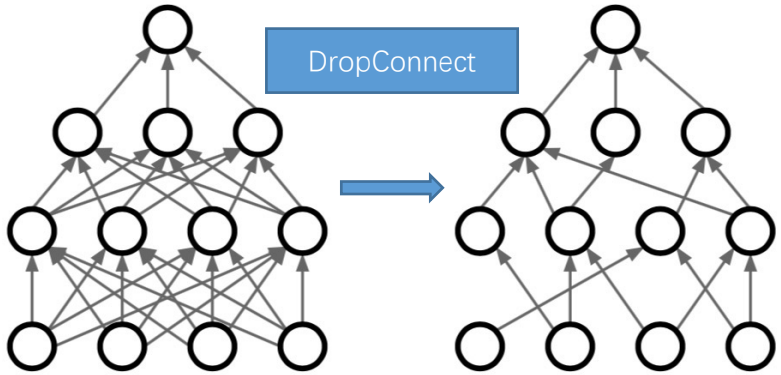
Krizhevsky A, Sutskever I, Hinton GE. Imagenet classification with deep convolutional neural networks. NIPS 2012.
K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. ICLR, 2015.
He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. CVPR 2016.

Data Augmentation

- 色彩抖动（Color Jitter）
  - ✓e.g.，随机调整对比度、亮度

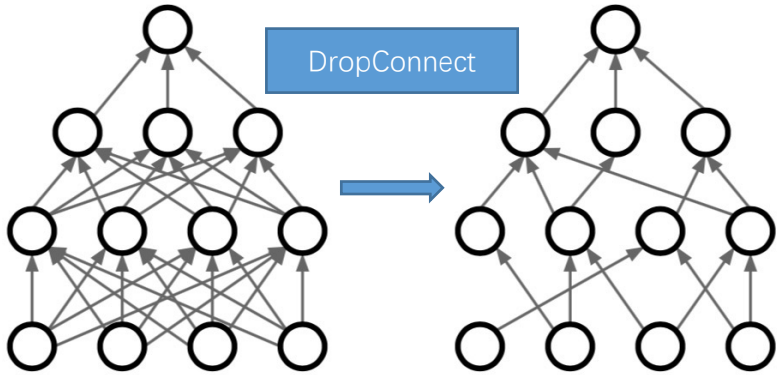




Data Augmentation

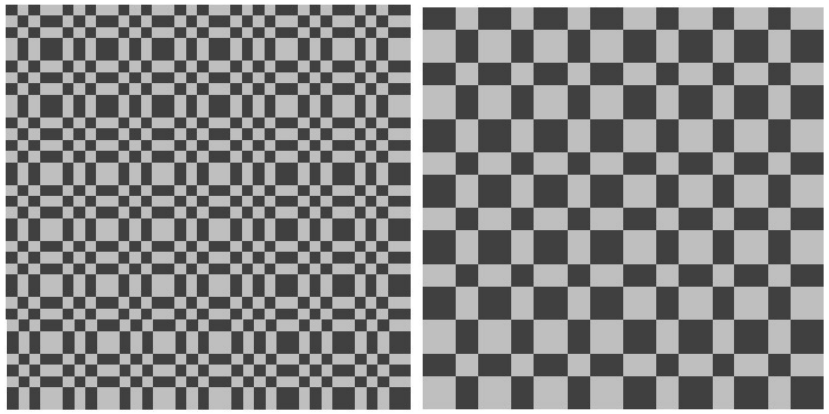


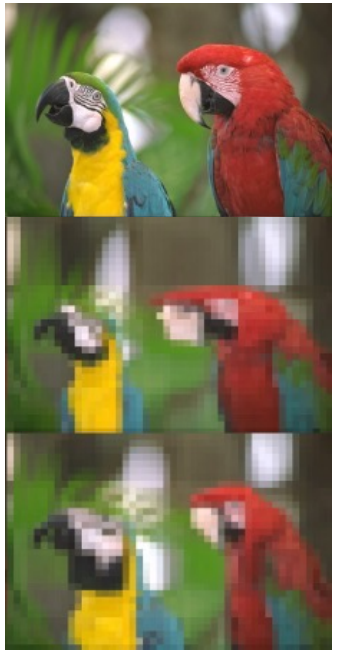
Wan L, Zeiler M, Zhang S, Le Cun Y, Fergus R. Regularization of neural networks using dropconnect. ICML 2013.

Data Augmentation



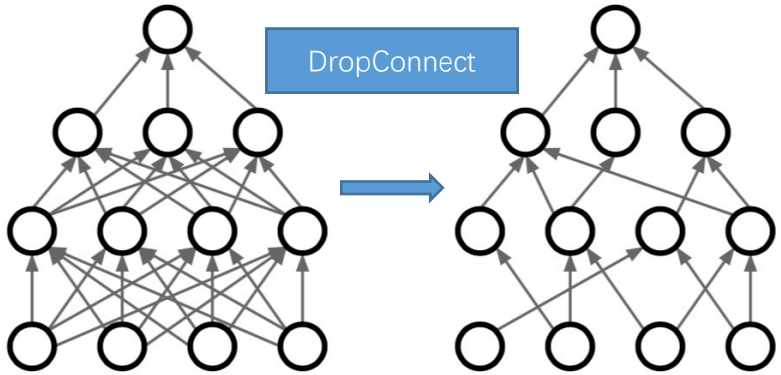
随机池化大小

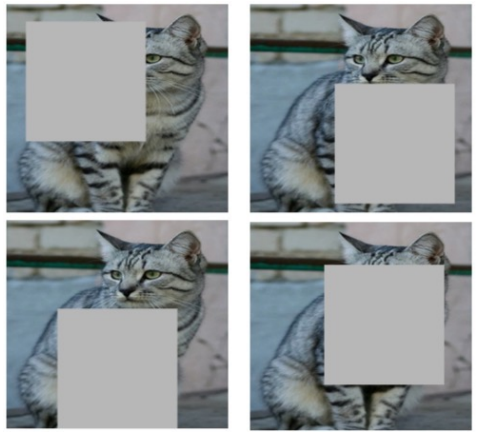




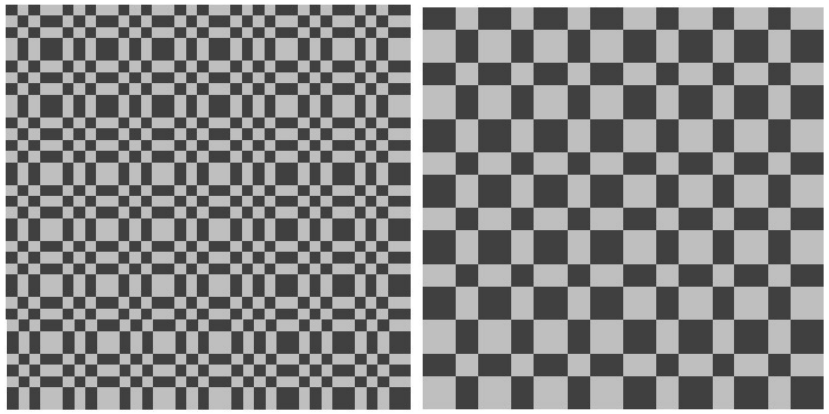
Wan L, Zeiler M, Zhang S, Le Cun Y, Fergus R. Regularization of neural networks using dropconnect. ICML 2013.
Graham B. Fractional max-pooling. 2014.

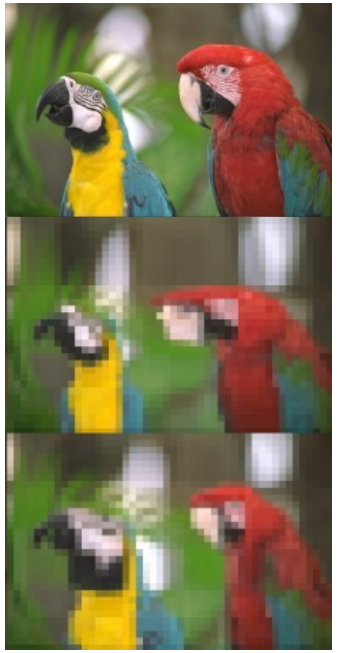
Data Augmentation





随机池化大小

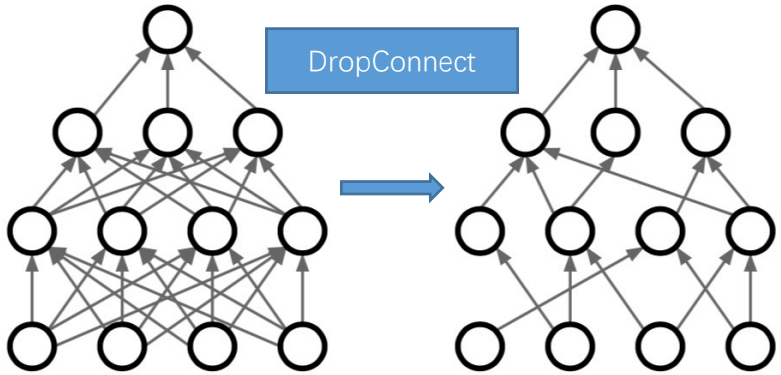




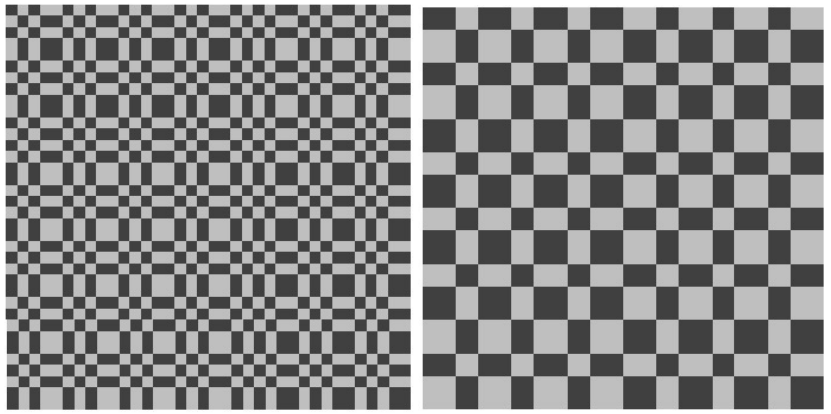
Wan L, Zeiler M, Zhang S, Le Cun Y, Fergus R. Regularization of neural networks using dropconnect. ICML 2013.
Graham B. Fractional max-pooling. 2014.
DeVries T, Taylor GW. Improved regularization of convolutional neural networks with cutout. 2017.

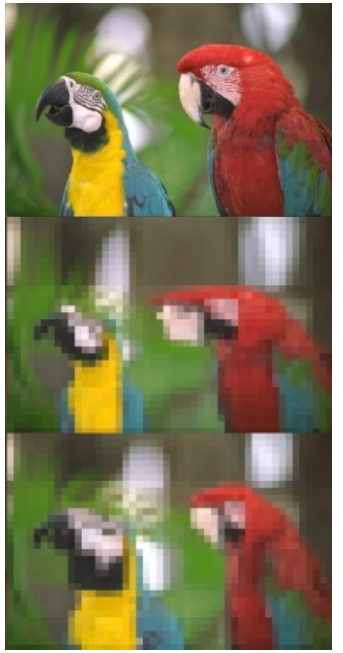
Data Augmentation

DropConnect



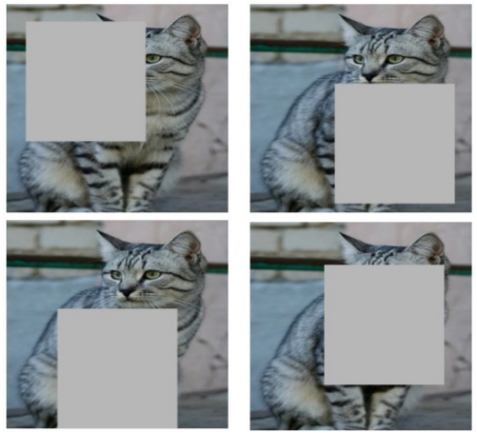
随机池化大小

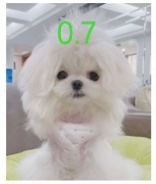


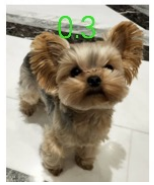


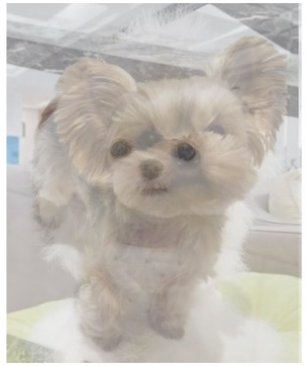
Cutout

小数据集常用









label

奥莉：0.3

毛栗子：0.7

Mixup

```
# y1, y2 should be one-hot vectors
for (x1, y1), (x2, y2) in zip(loader1, loader2):
    lam = numpy.random.beta(alpha, alpha)
    x = Variable(lam * x1 + (1. - lam) * x2)
    y = Variable(lam * y1 + (1. - lam) * y2)
    optimizer.zero_grad()
    loss(net(x), y).backward()
    optimizer.step()
```


Wan L, Zeiler M, Zhang S, Le Cun Y, Fergus R. Regularization of neural networks using dropconnect. ICML 2013.

Graham B. Fractional max-pooling. 2014.

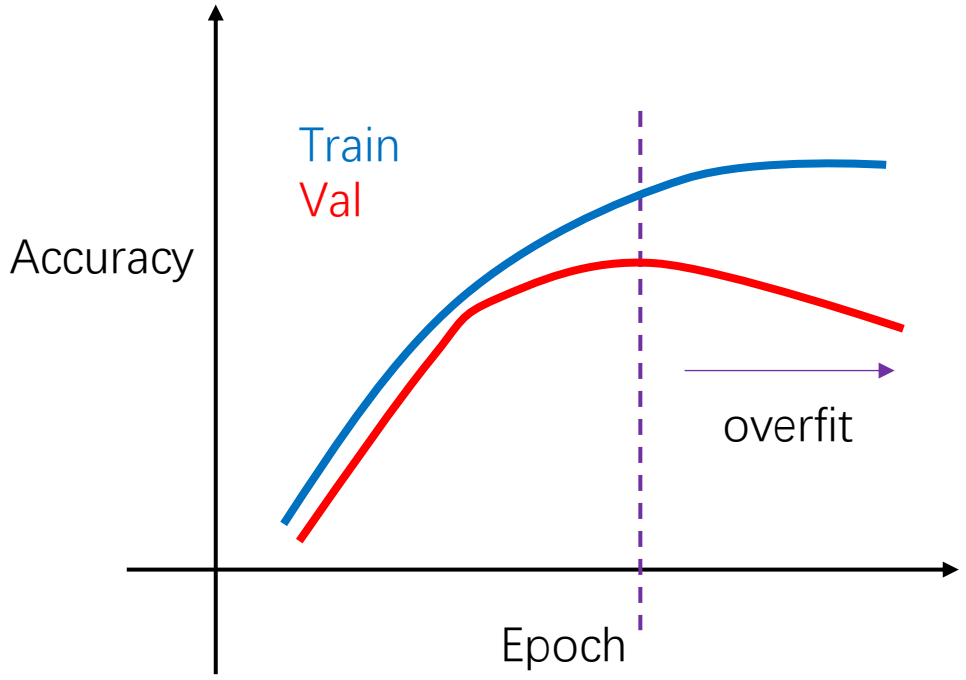
DeVries T, Taylor GW. Improved regularization of convolutional neural networks with cutout. 2017.

Zhang H, Cisse M, Dauphin YN, Lopez-Paz D. mixup: Beyond empirical risk minimization. ICLR 2018.

实际模型中。。。

- 在大型全连接网络中使用Dropout
- 使用batch normalization和各种常用的data augmentation
- 在小数据集上尝试随机池化、cutout、mixup等
- 使用early stopping和ensembles

Early Stopping（早停法）



- ✓ 当验证集上精度不再提高甚至下降时，停止模型训练
- ✓ 调整超参重新开始训练，或者使用早停前最优模型进行推理

Prechelt L. Early stopping-but when? Neural Networks 1998.

Caruana R, Lawrence S, Giles CL. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. NIPS 2001.

Model Ensembles（模型集成）

- 训练多个独立的模型，平均预测结果
- 使用同一个模型的多个快照去预测，平均预测结果
- 对模型参数做momentum（moving average），作为模型的最终参数

超参数的选择：学习率和weight decay

1. 验证loss计算的正确性
   - ✓不加正则项
   - ✓加上正则项，loss应该适当增大

第i个图片的损失为：

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

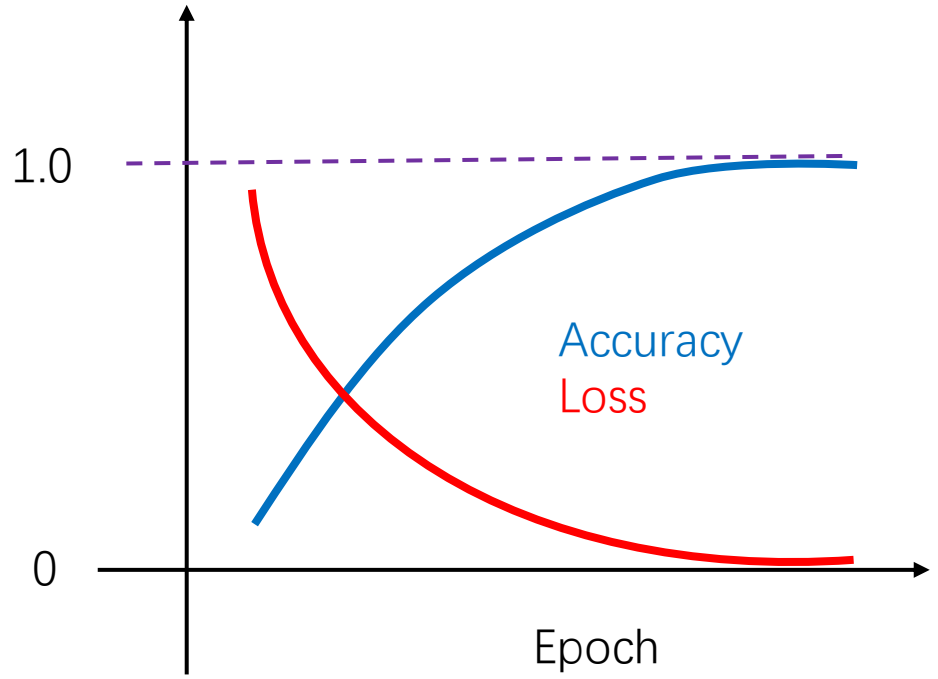
Q：假如初始化$\boldsymbol{W}$接近0，导致所有输出分数都≈0，那么L约等于多少？

A：$\log C$，C为类别数，这里为log3≈0.477

超参数的选择

1. 验证loss计算的正确性
2. 确保能够overfit一个小样本集
   - ✓e.g., 从cifar10选出～50张图片训练模型
   - ✓不用正则项（weight decay（reg）=0），初步调整学习率和初始化方法

loss不降：学习率太小
loss暴增：学习率太大



超参数的选择

1. 验证loss计算的正确性
2. 确保能够overfit一个小样本集
3. 使用全部数据集寻找合适的学习率
   - ✓加上很小的weight decay，搜索使loss快速下降（100个iteration以内）的学习率
   - ✓尝试学习率=0.1，0.01，0.001，0.0001

超参数的选择

1. 验证loss计算的正确性
2. 确保能够overfit一个小样本集
3. 使用全部数据集寻找合适的学习率
4. 粗粒度搜索学习率和weight decay，观察验证集结果
   - ✓在第3步确定的学习率范围内随机搜索，并随机搜索weight decay
   - ✓每个学习率和weight decay组合训练～5个epoch

```
>>> count=100
>>> for i in range(count):
...     lr = 10**np.random.uniform(-5,-3)
...     reg = 10**np.random.uniform(-4, 4)
...     //training for 5 epoch
```

超参数的选择

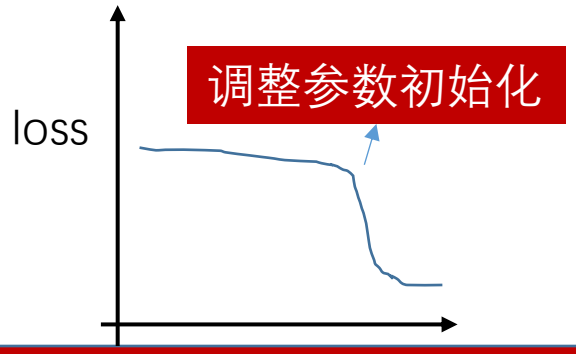
1. 验证loss计算的正确性
2. 确保能够overfit一个小样本集
3. 使用全部数据集寻找合适的学习率
4. 粗粒度搜索学习率和weight decay，观察验证集结果
5. 细粒度搜索学习率和weight decay，观察验证集结果
   - ✓缩小第4步的搜索范围，训练更长时间（~20 epoch）

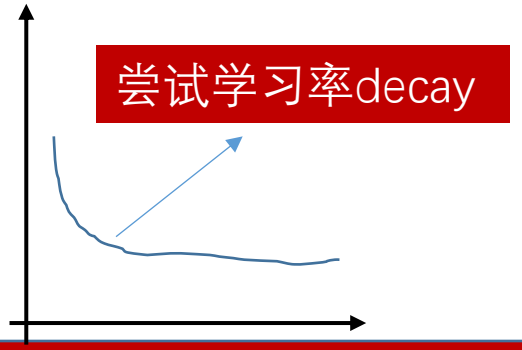
```
>>> count=100
>>> for i in range(count):
...     lr = 10**np.random.uniform(-5,-3)
...     reg = 10**np.random.uniform(-4, 4)
...     //training for 5 epoch
```

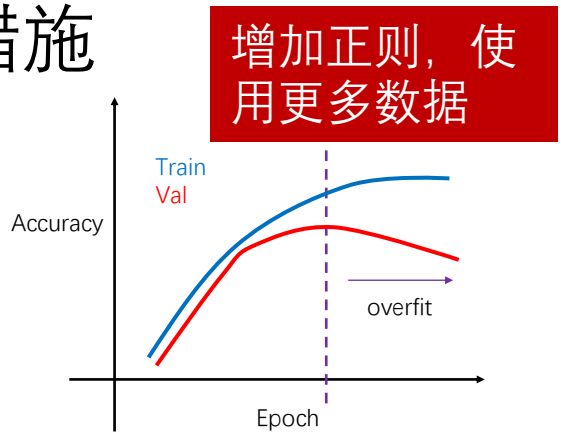
```
count=100
for i in range(count):
    lr = 10**np.random.uniform(-4,-3)
    reg = 10**np.random.uniform(-3, 0)
    //training for 20 epoch
```

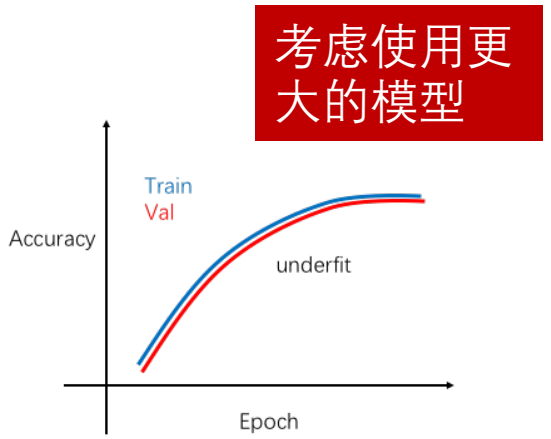
超参数的选择

1. 验证loss计算的正确性
2. 确保能够overfit一个小样本集
3. 使用全部数据集寻找合适的学习率
4. 粗粒度搜索学习率和weight decay，观察验证集结果
5. 细粒度搜索学习率和weight decay，观察验证集结果
6. 根据learning curve采取相应措施



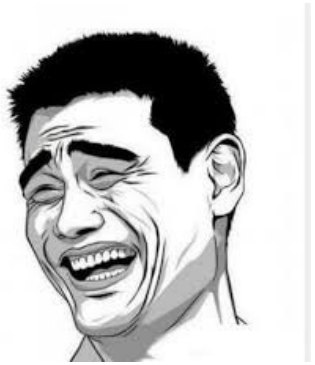






超参数的选择

1. 验证loss计算的正确性
2. 确保能够overfit一个小样本集
3. 使用全部数据集寻找合适的学习率
4. 粗粒度搜索学习率和weight decay，观察验证集结果
5. 细粒度搜索学习率和weight decay，观察验证集结果
6. 根据learning curve采取相应措施
7. 还是不理想？回到第5步重做吧



小结

- 优化方法的演进
  - ✓SGD, Momentum, AdaGrad/RMSProp, Adam
- 学习率的设置
  - ✓各种decay
- 正则化
  - ✓BN, dropout, data augmentation, early stopping, ensembles
- 超参数的选择
  - ✓小数据集→大数据集，粗粒度搜索→细粒度搜索
  - ✓plot学习曲线（loss/accuracy），采取相应措施

L08

- DL硬件
  - ✓CPU，GPU，TPU
- DL软件
  - ✓PyTorch，TensorFlow