Design and Train Neural Networks

AIPKU

人工智能引论实践课 计算机视觉小班

主讲人：胡越予

深度神经网络的设计和训练

- 输入和输出
- 设计基础网络网络结构
- 神经网络参数的初始化
- 激活函数
- 优化器与学习率

Some slides are edited from CS213n http://cs231n.stanford.edu

输入与输出

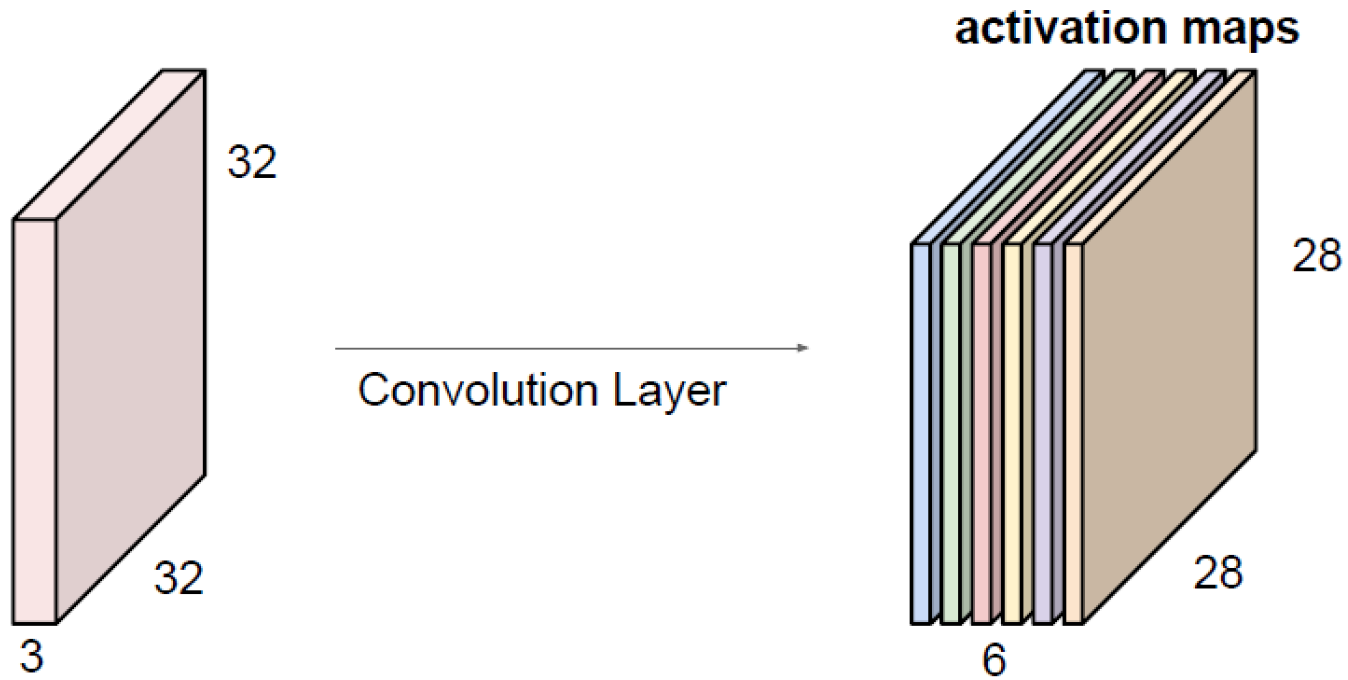
- 数据存储形式
  - 多个图像文件（视频处理成图像序列）
  - 读取：从磁盘中读取，解码，预处理
  - 开销较大，小文件读写较慢
- 固实数据存储方案
  - numpy
    - numpy可以对张量进行快速存取(save, load)
  - pickle
    - pickle能够序列化/反序列化任何python对象
  - HDF5 (h5py)
    - 存储和处理大容量科学数据设计的文件格式，跨语言使用

h5py 实例

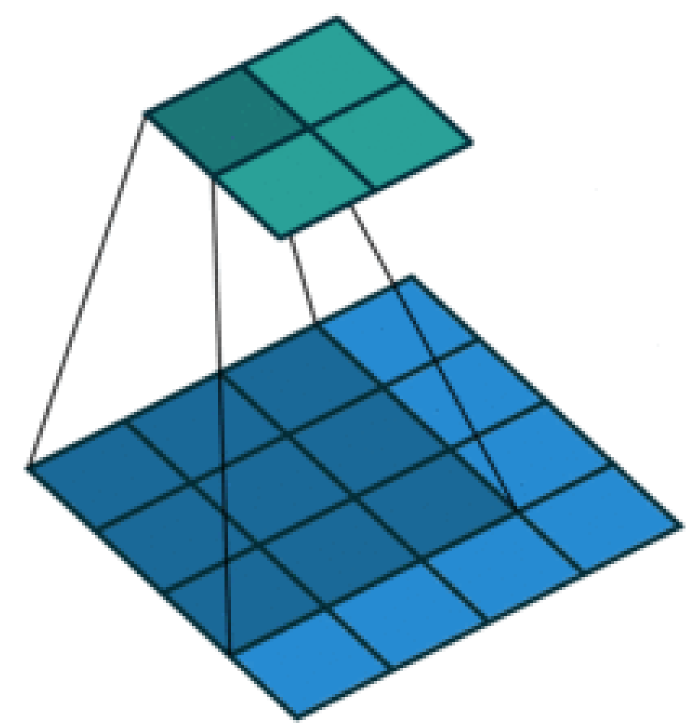
```
f = h5py.File('mytestfile.hdf5', 'w')
data = np.array([1,2,3,4])
f['data'] = data
f.close()

f = h5py.File('mytestfile.hdf5', 'r')
data = np.array(f['data'])
```

基础网络结构：卷积层

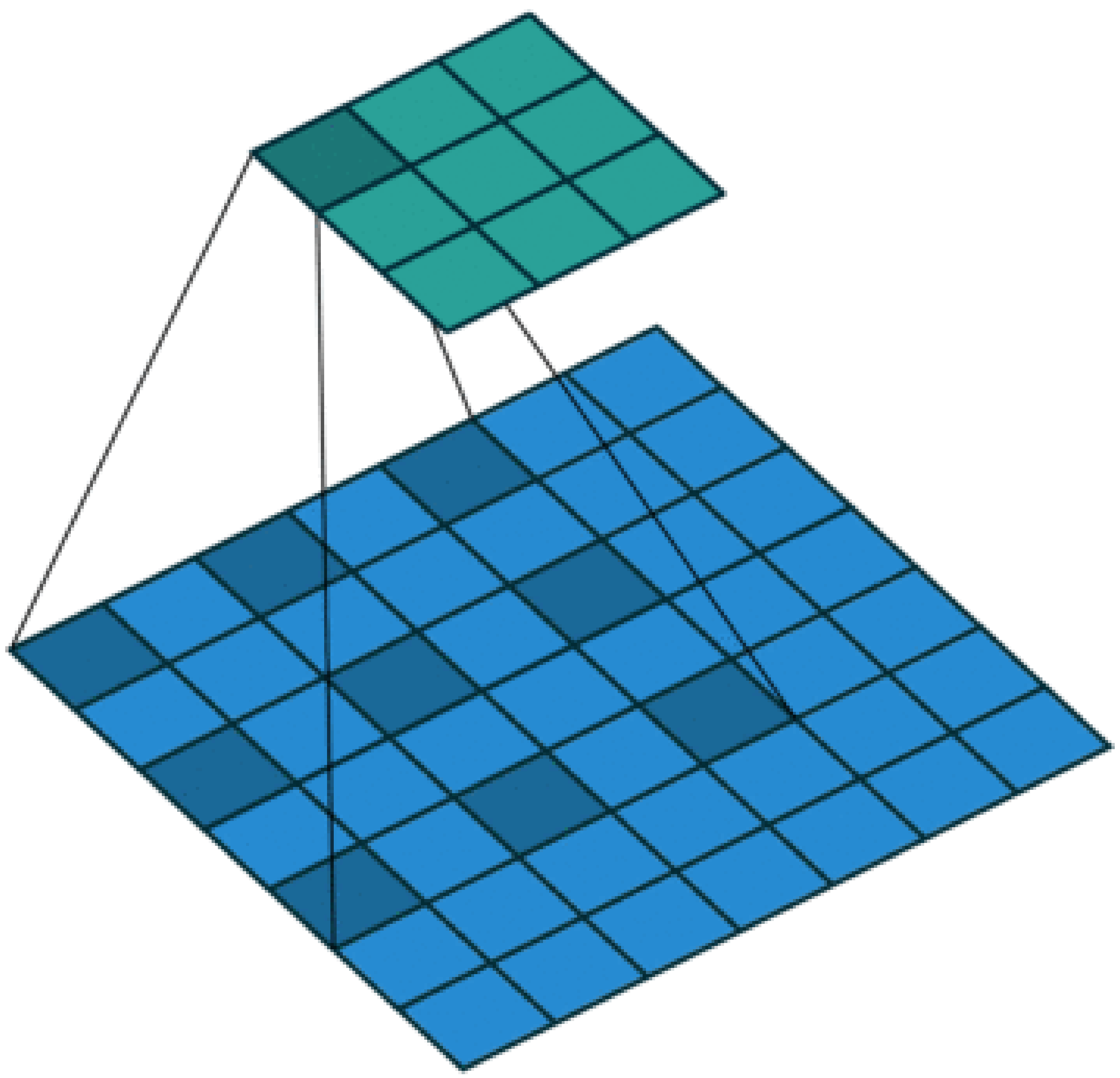
- 单层卷积：在通道上可以看作全连接层，在空间上看作卷积



- 感受野：单个神经元可以“看见”的区域大小
  - 影响感受野的因素：
    - 卷积核大小（大卷积核会导致参数过多）
    - 网络深度：越深的神经元感受野越大
    - 重采样，插空卷积...
  - 我们一般希望感受野能尽量大

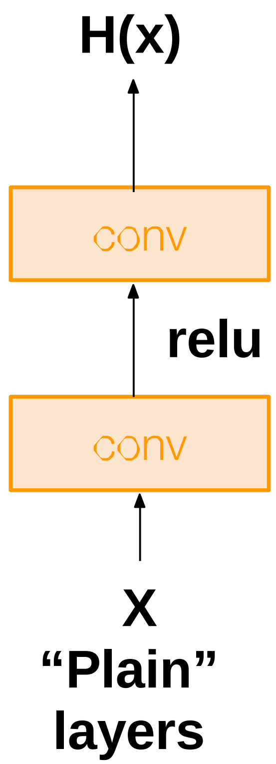


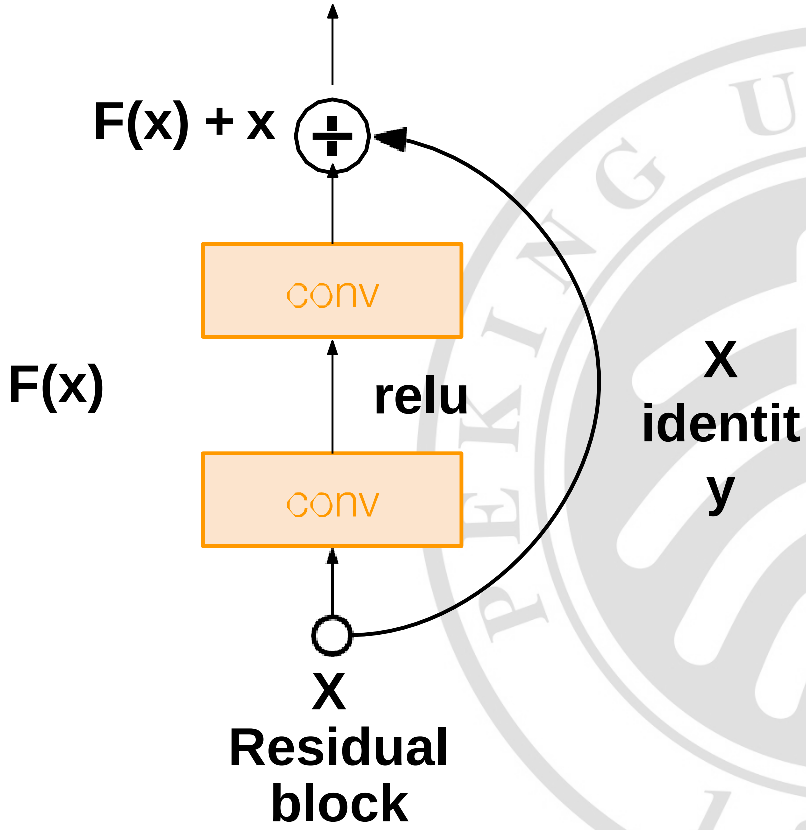
- 空洞卷积 Dilated Convolution



增大网络深度 ResNet *[He et al., 2015]*

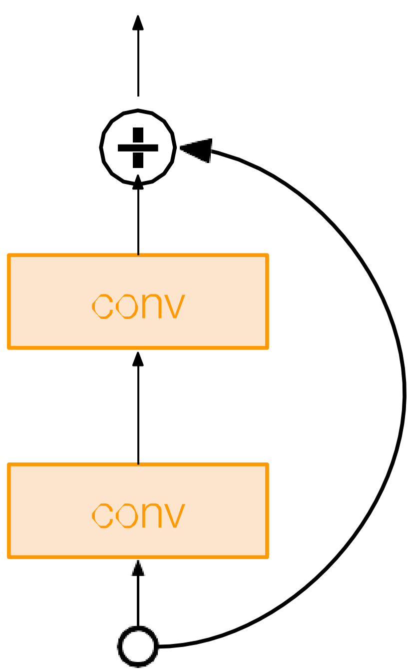
- 深层网络难以训练
  - 梯度反传时会经过长链，出现消失或者爆炸现象
  - 每一层都需要能够重建完整信息，训练难度较大
- 残差网络
  - 存在输入到输出的跳跃连接，梯度能够直接到达浅层
  - 每一层学习信息的残差，训练难度较小

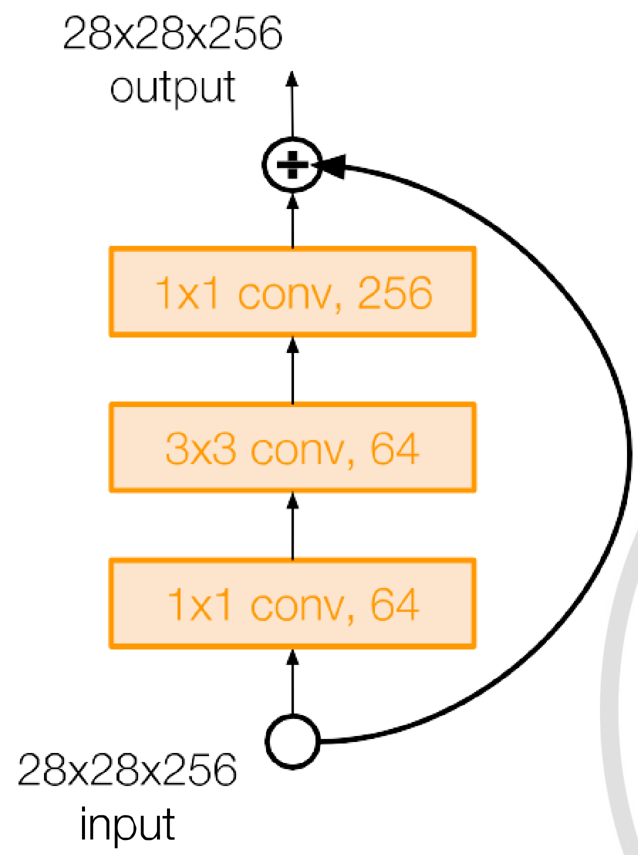




ResNet *[He et al., 2015]*

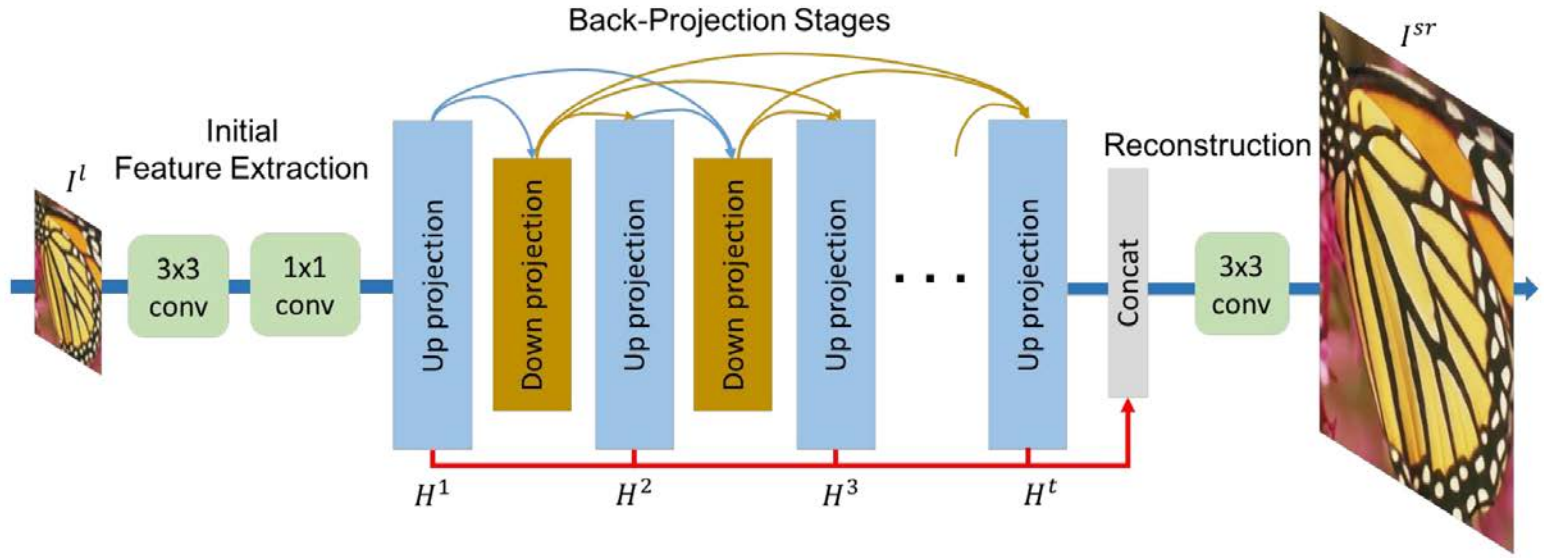
- 两种不同的残差块结构、
  - 使用 bottleneck 结构，降低维度，节省参数





Residue Learning 思想的推广

- 更稠密的连接，多路的连接



Deep Back-Projection Networks For Super-Resolution, CVPR 2018

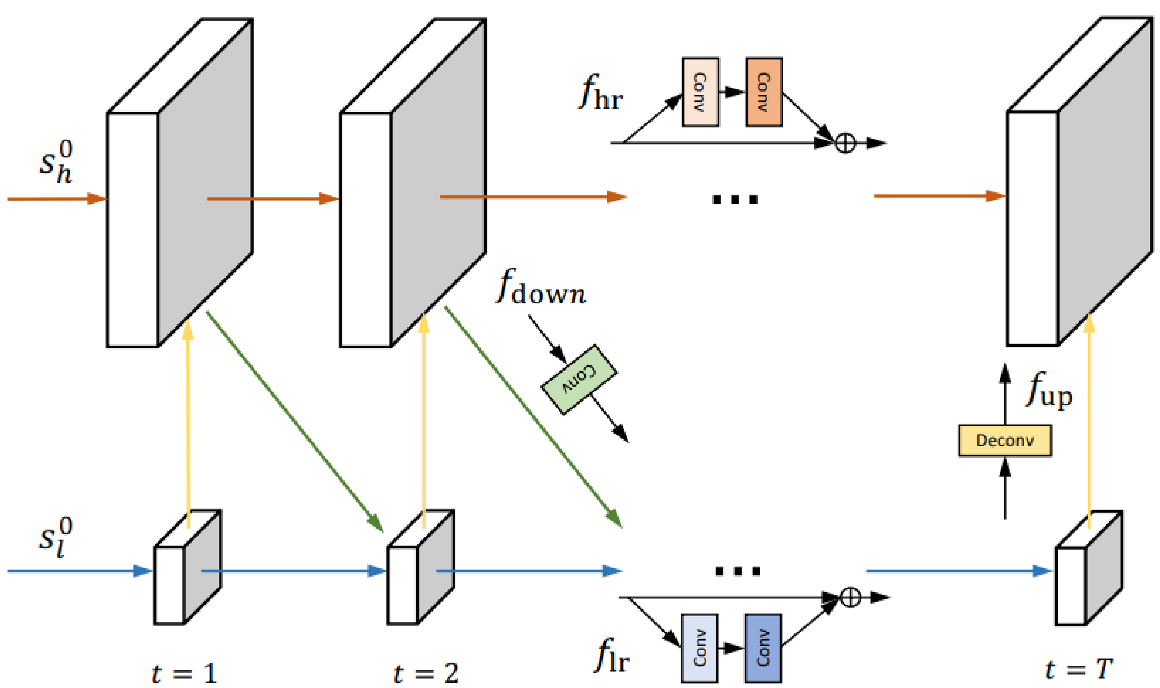
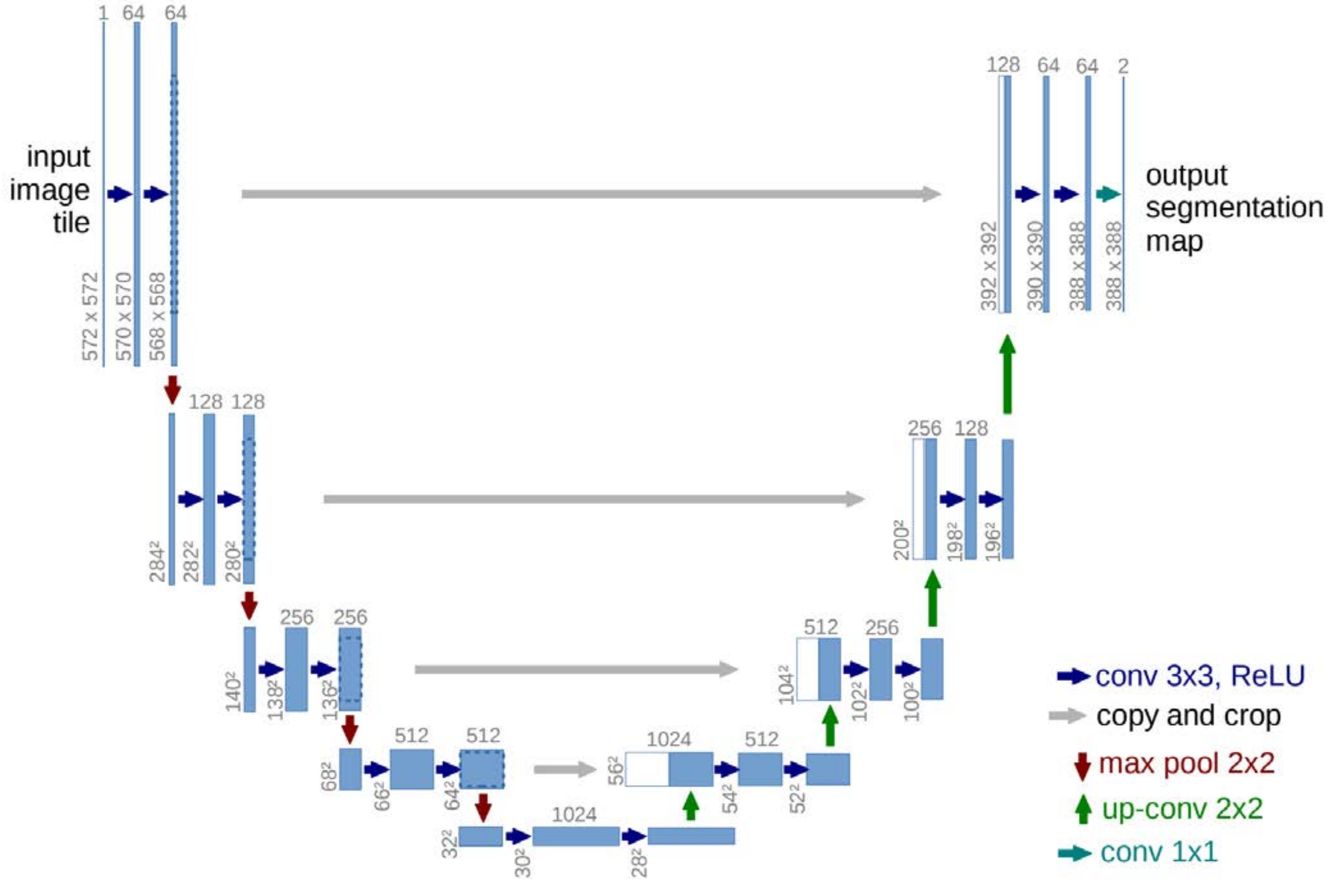


Image Super-Resolution via Dual-State Recurrent Networks, CVPR 2018

加入重采样：UNet

- 深层网络难以训练
  - 需要很大的存储
  - 消耗大量计算资源
  - 高分辨率带来计算量的快速增长
- 在网络中引入重采样：UNet



在网络中实现重采样

- 使用 stride convolution 实现下采样
- 使用 transposed convolution 实现上采样
  - 输入图像的每一个位置，通过卷积核投射到输出的图像中

Input

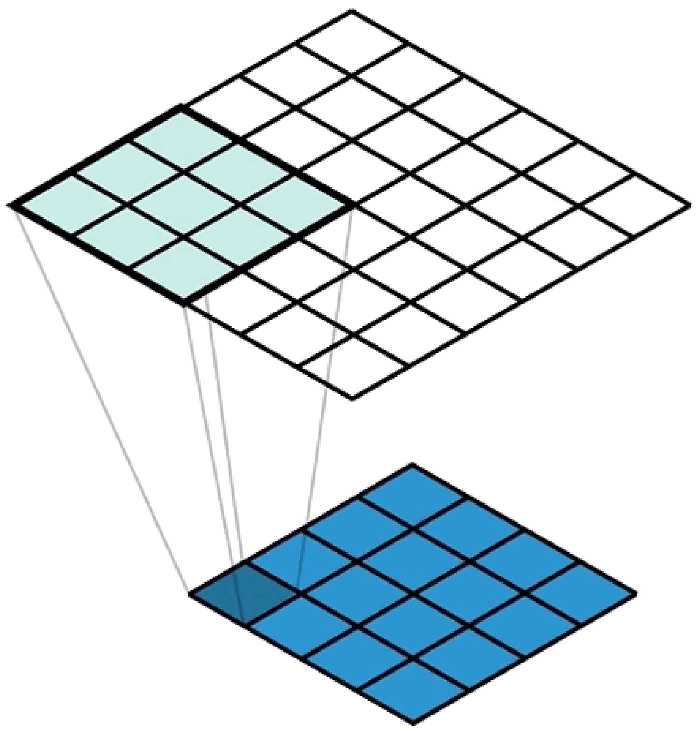
1	1	1	1
1	1	1	1
1	1	1	1
1	1	1	1

Kernel

1	1	1
1	1	1
1	1	1

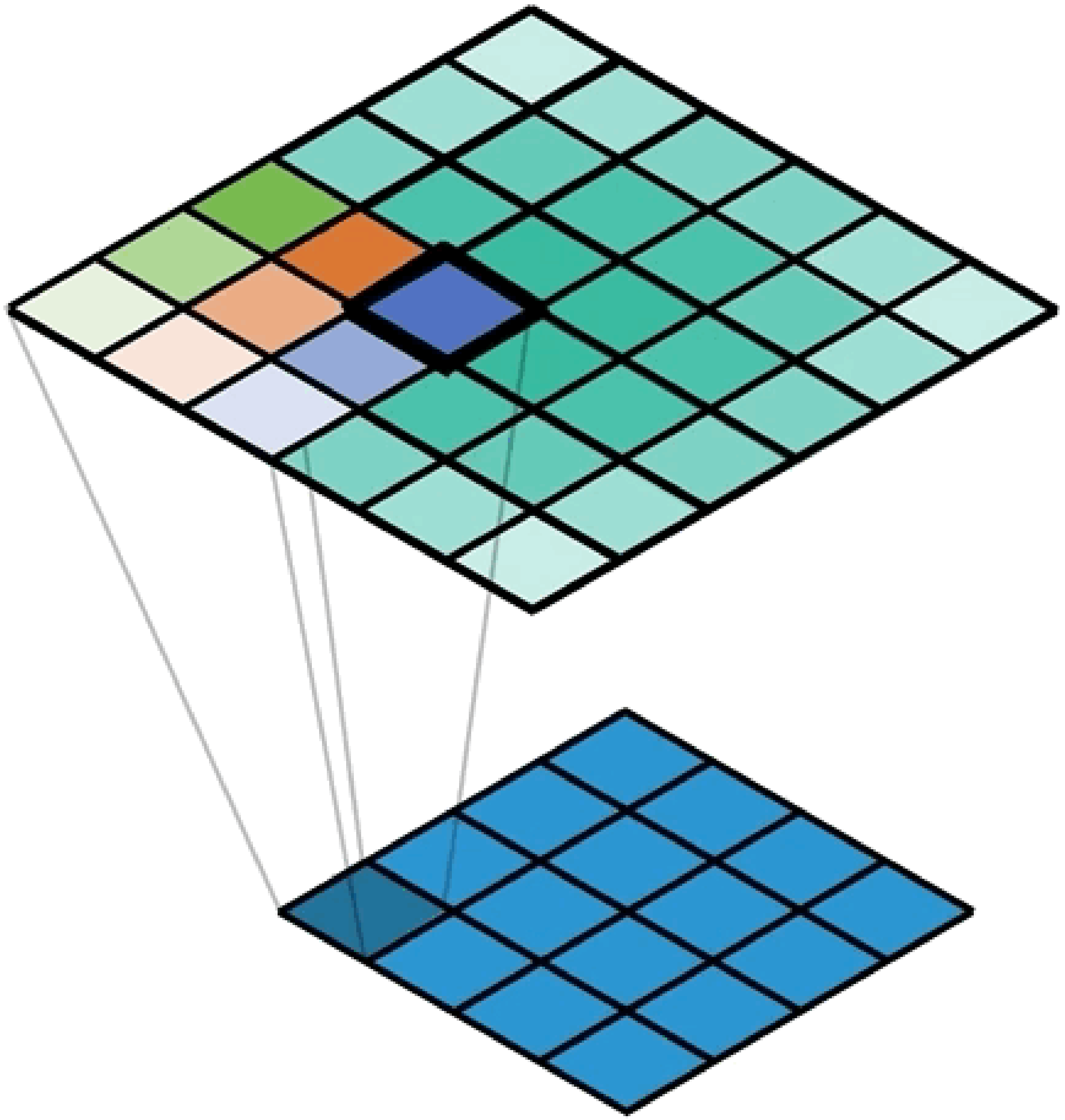
Output

1	2	3	3	2	1
2	4	6	6	4	2
3	6	9	9	6	3
3	6	9	9	6	3
2	4	6	6	4	2
1	2	3	3	2	1



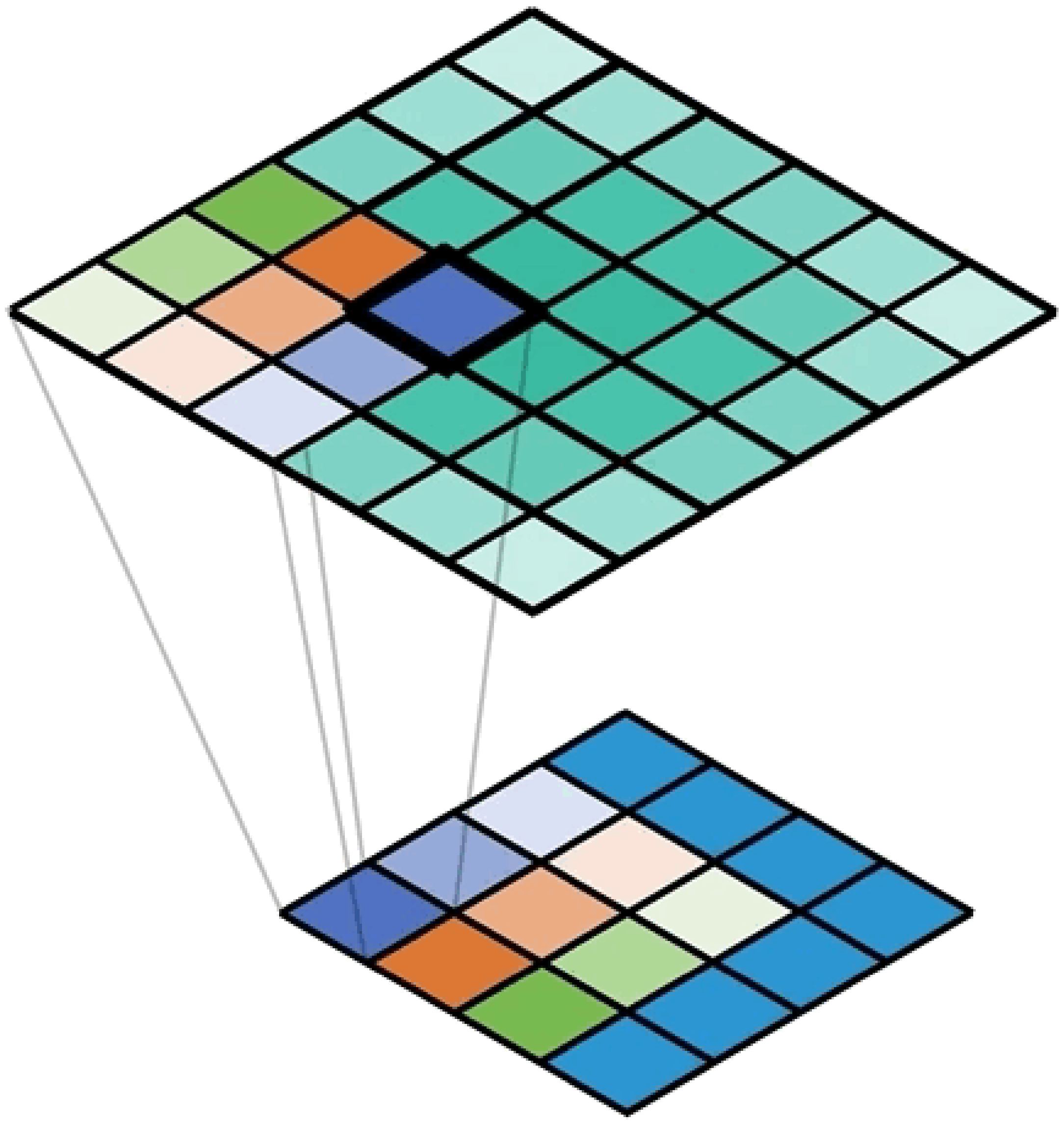
Transposed Convolution

- 实际上反卷积也是只是卷积
  - 考虑一个输出的位置



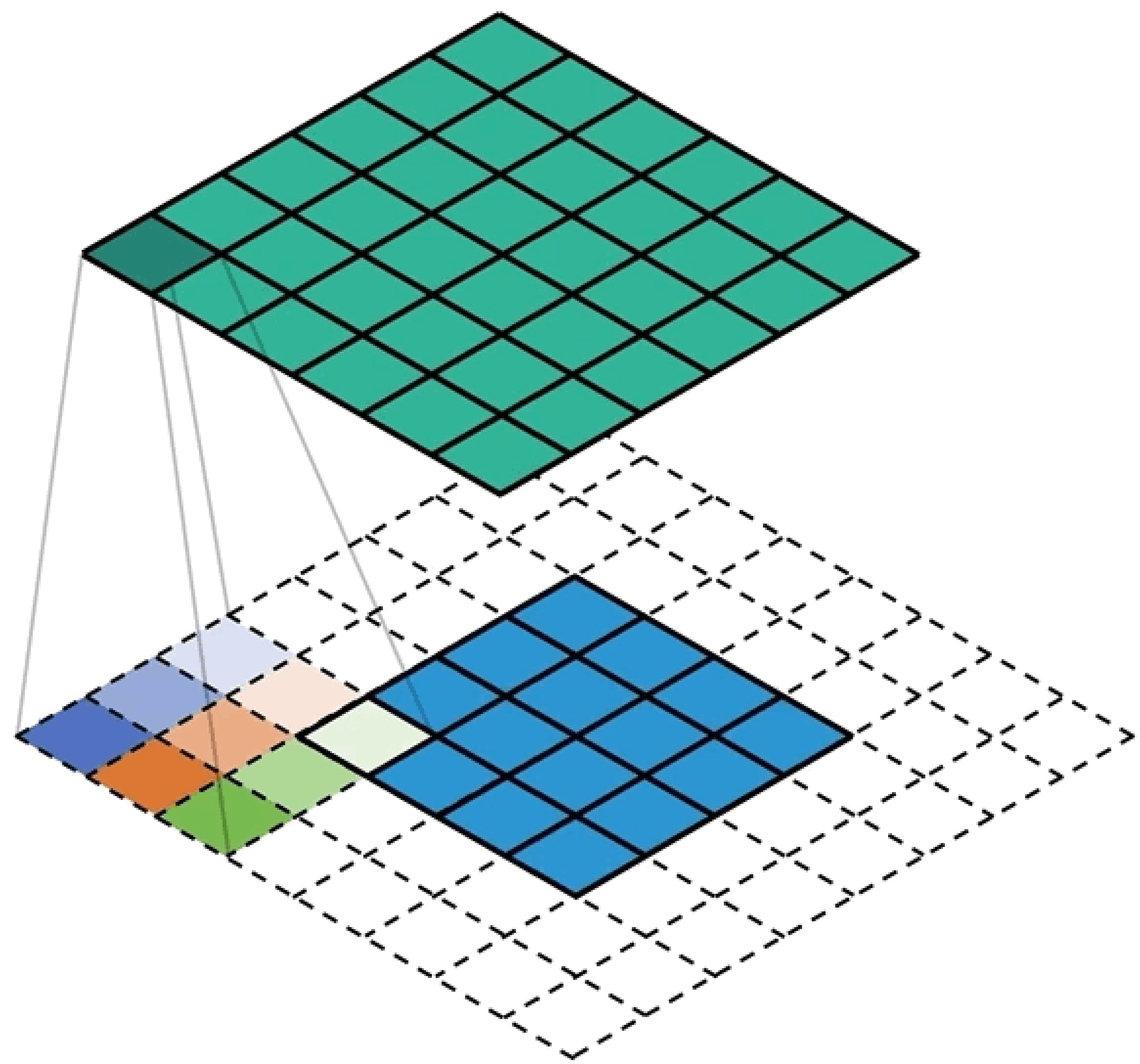
Transposed Convolution

- 实际上反卷积也是只是卷积
  - 考虑一个输出的位置



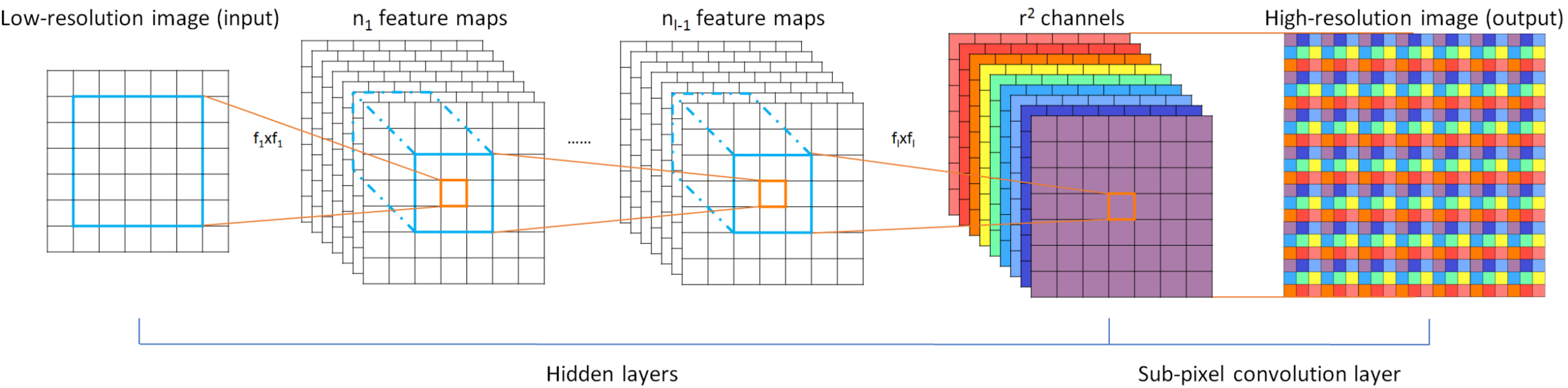
Transposed Convolution

- 实际上反卷积也是只是卷积
  - 考虑一个输出的位置
  - 与下图等价



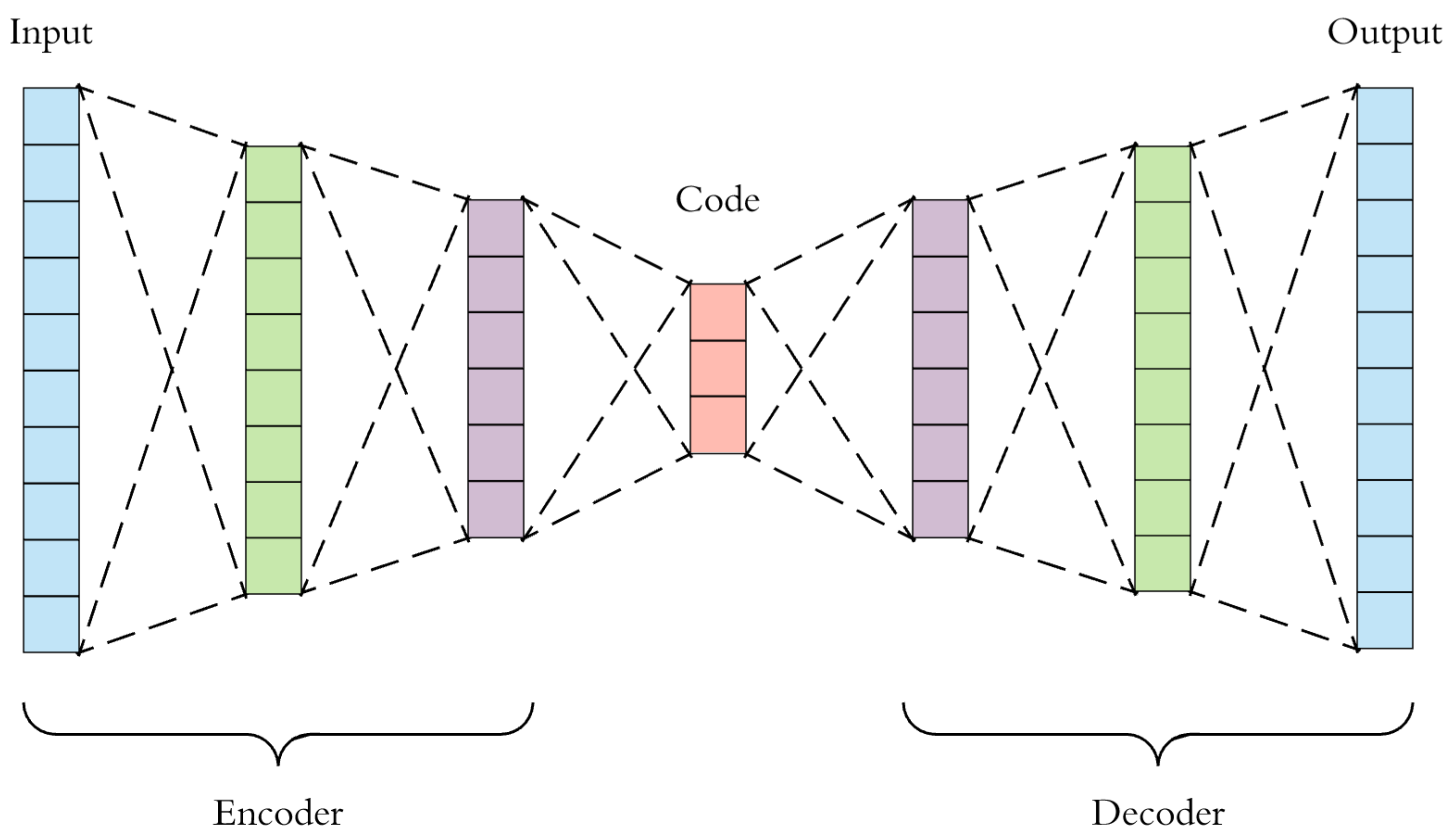
其它重采样方法

- 直接对特征进行插值
- Subpixel Convolution



UNet & Auto Encoder

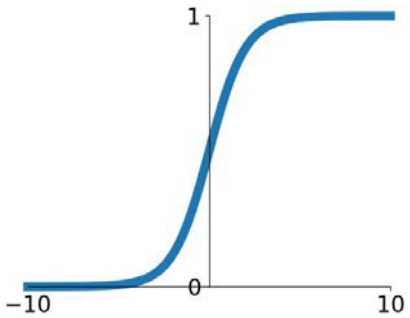
- Auto Encoder 一般用于非监督学习
- 区别在于是否存在 Skip-Connection
- 用于图像-图像映射，细粒度分割，压缩，生成......



激活函数

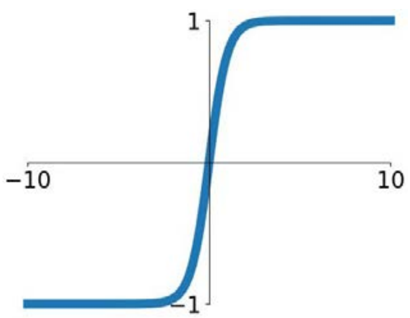
Sigmoid

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



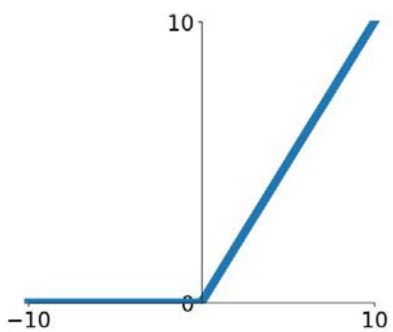
tanh

$$\tanh(x)$$



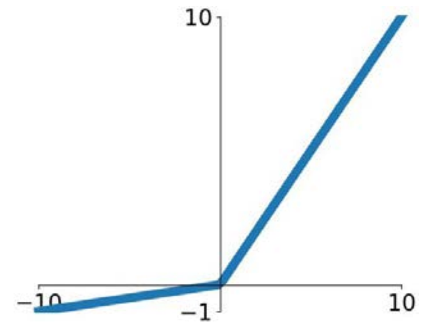
ReLU

$$\max(0, x)$$



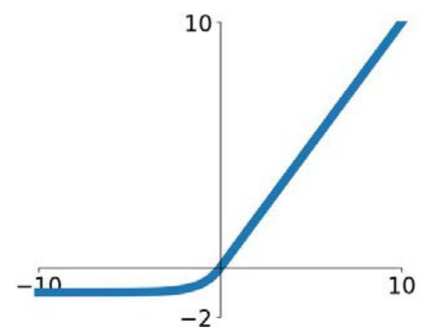
Leaky ReLU

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



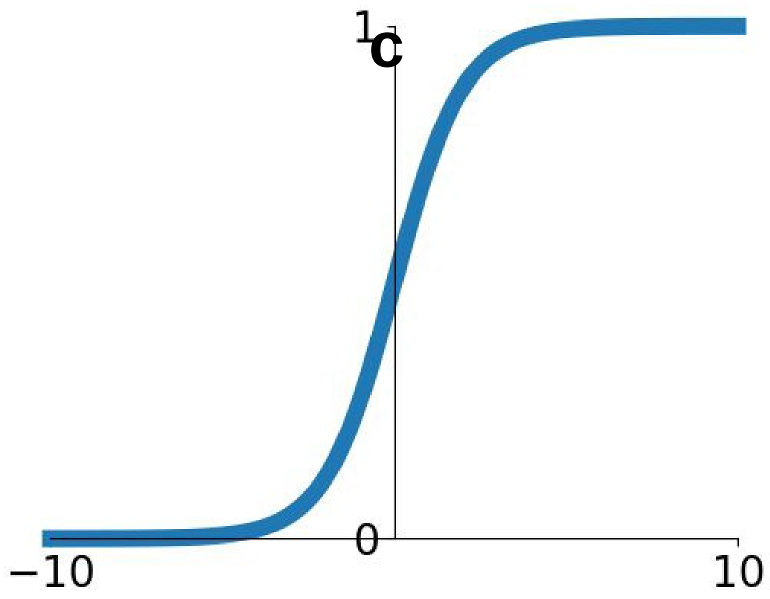
ELU

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



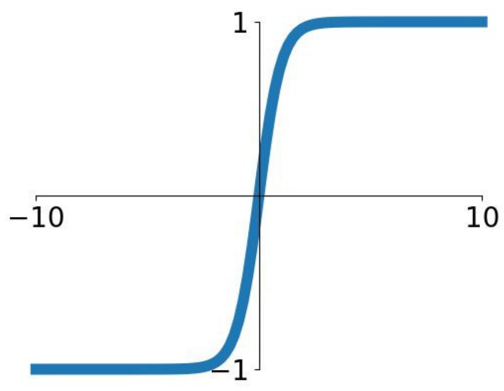
Sigmoid

- 能够将数据归一化到[0,1]
- 曾一度被用于神经网络
  - 与生物神经元的激活类似
- 存在问题
  - 梯度饱和
    - 输入数据绝对值较大的时候几乎没有梯度
  - 计算困难
    - 实现指数运算较为困难，运算较慢



tanh

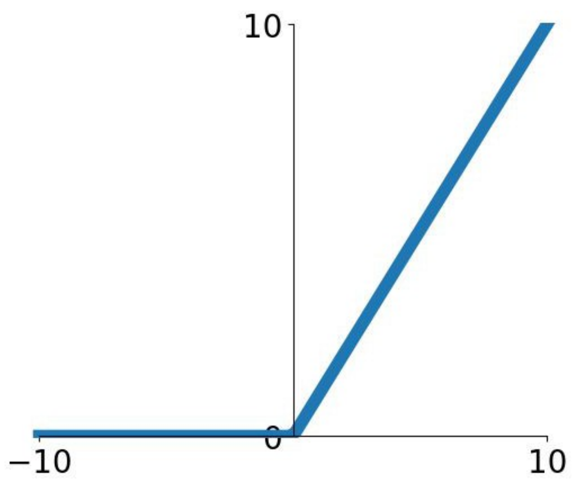
- 能够将数据归一化到[-1,1]
- 存在问题
  - 梯度饱和
    - 输入数据绝对值较大的时候几乎没有梯度
  - 计算困难
    - 实现指数运算较为困难，运算较慢



tanh(x)

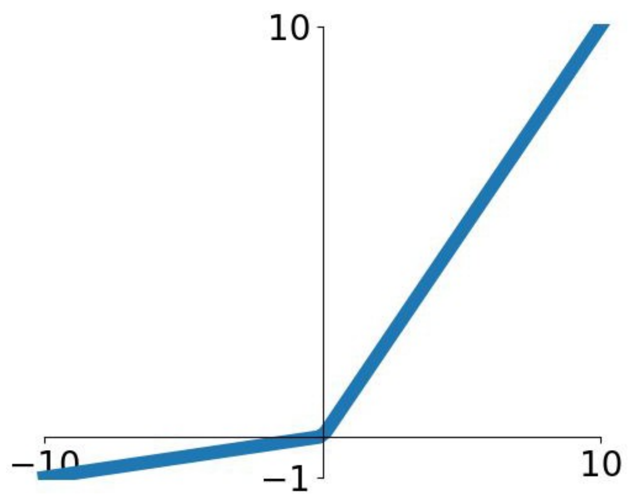
ReLU

- Rectified Linear Unit
- 不会出现梯度饱和问题
- 硬件实现非常简单
- 网络收敛速度更快
- 存在问题：
  - 当输入小于0时没有梯度



Leaky ReLU & PReLU

- Leaky ReLU
  - 拥有 ReLU 的所有优点
  - 不会导致无梯度问题
  - 左半边斜率如何选取?



- PReLU
  - 使用一个可以学习的斜率

神经网络参数初始化

- 为什么初始化会影响性能?
  - 如果问题足够简单，可以直接求解
  - 深度神经网络可以求解复杂的函数优化问题
    - 但往往无法找到全局最优解
    - 初始化会影响找到的局部最优
- 初始化方法:
  - He Initialization

    `stddev = sqrt(2 / fan_in)`
  - Xavier Initialization

    `stddev = sqrt(2 / (fan_in + fan_out))`
  - 预训练: ImageNet Pretraining

关于初始化的研究

- ***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

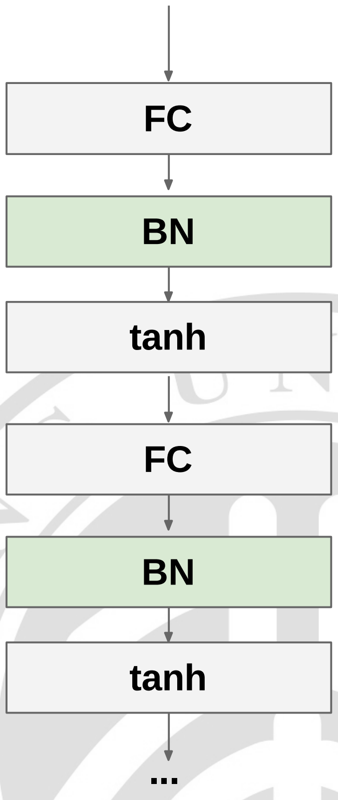
- ***All you need is a good init***, Mishkin and Matas, 2015

Batch Normalization

- 激活函数往往以 0 为激活分界
  - 如果在网络中所有的输出都是正数，激活函数等于没用
  - 如果在网络中所有的输出均为负数，同理
  - 我们希望一个分布均匀规整的数据
  - 想要就要!
- 在 batch 上将数据归一化，均值为0，方差为1

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

- 在激活函数前使用



Batch Normalization

- 是否真的需要处处归一化?
  - 有些情况下需要输出多样化的结果
  - 给网络一个反悔的机会:

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

$$y^{(k)} = \gamma^{(k)}\widehat{x}^{(k)} + \beta^{(k)}$$

Note, the network can learn:

$$\gamma^{(k)} = \sqrt{\mathrm{Var}[x^{(k)}]}$$

$$\beta^{(k)} = \mathrm{E}[x^{(k)}]$$

to recover the identity mapping.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

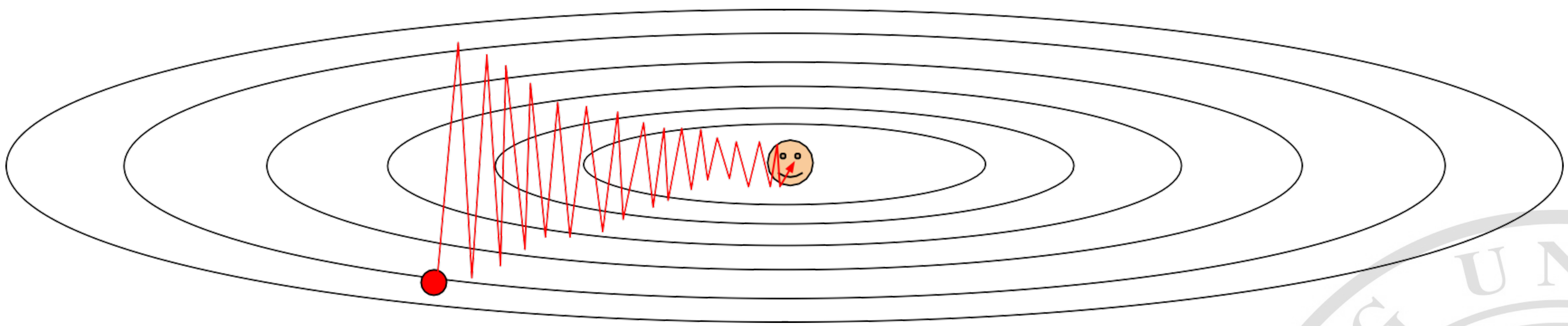
$$\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_\mathcal{B})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} \qquad \text{// normalize}$$

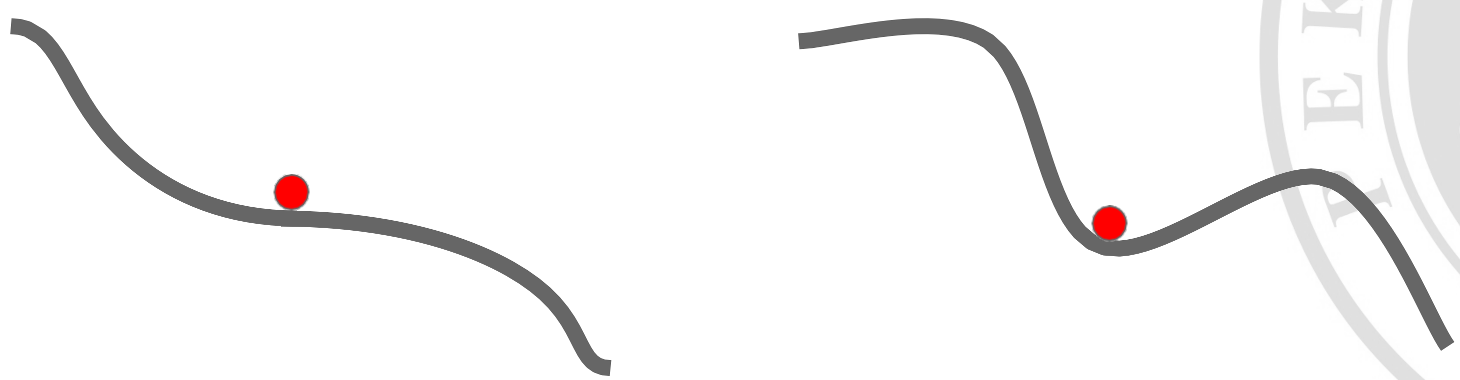
$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

SGD 遇到的困难

- 随机梯度下降 Stochastic Gradient Descent
- 在多维的情况下会遇到问题
  - 由于 Learning Rate 的存在，会在梯度大的维度震荡
  - 收敛较慢



- 依赖梯度
  - 在鞍点无法继续优化



动量方法 Momentum

SGD

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

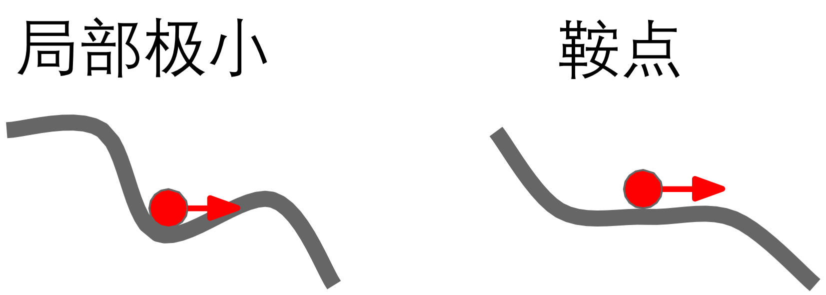
```
while True:
  dx = compute_gradient(x)
  x -= learning_rate * dx
```

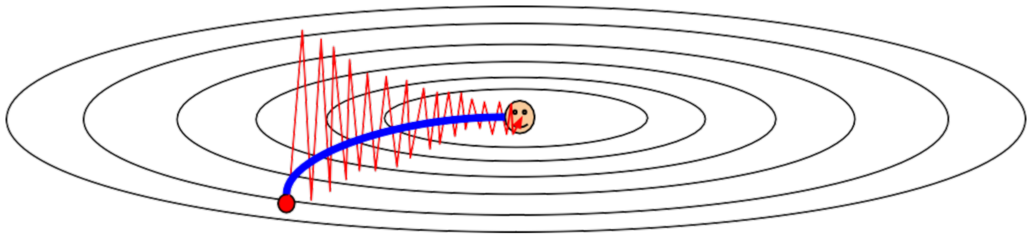
SGD+Momentum

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

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

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

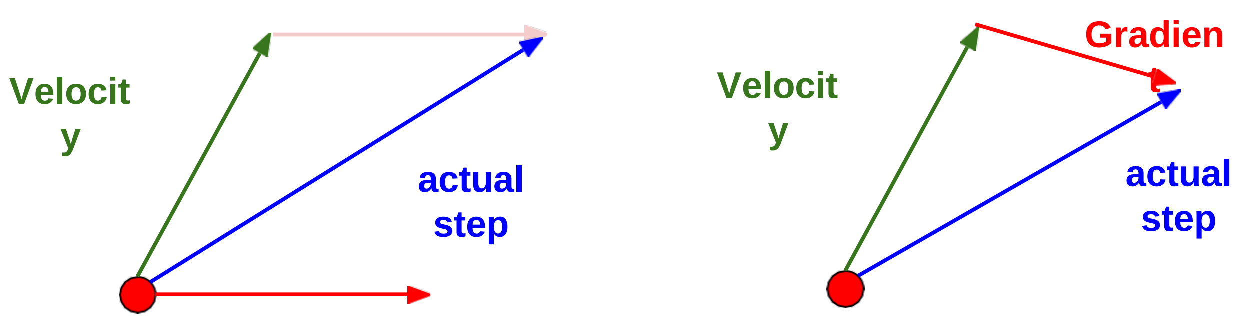




Sutskever et al, “On the importance of initialization and momentum in deep learning”, ICML 2013

Nesterov Momentum

- 先按速度走，再求梯度



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

AdaGrad and RMSprop

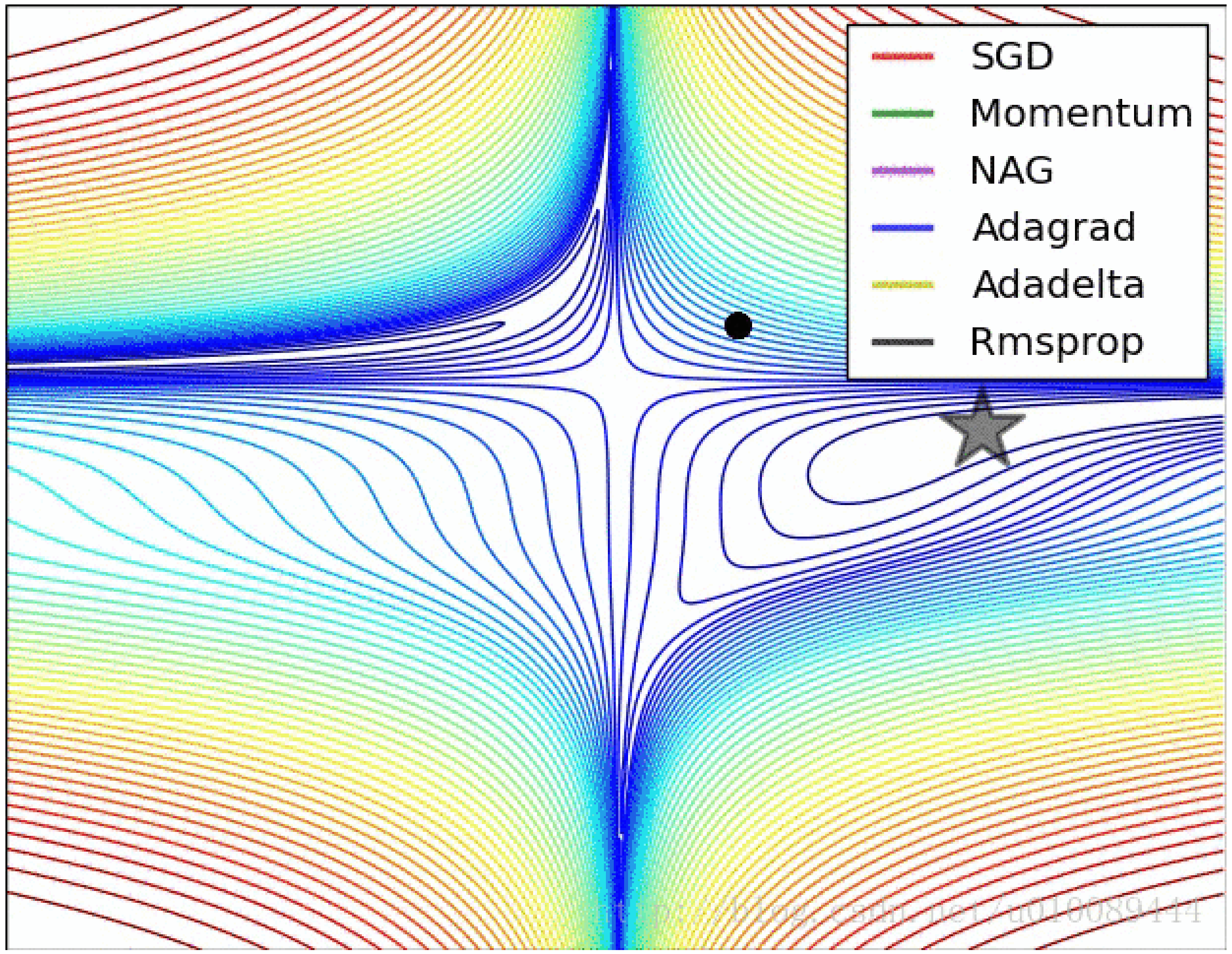
AdaGrad

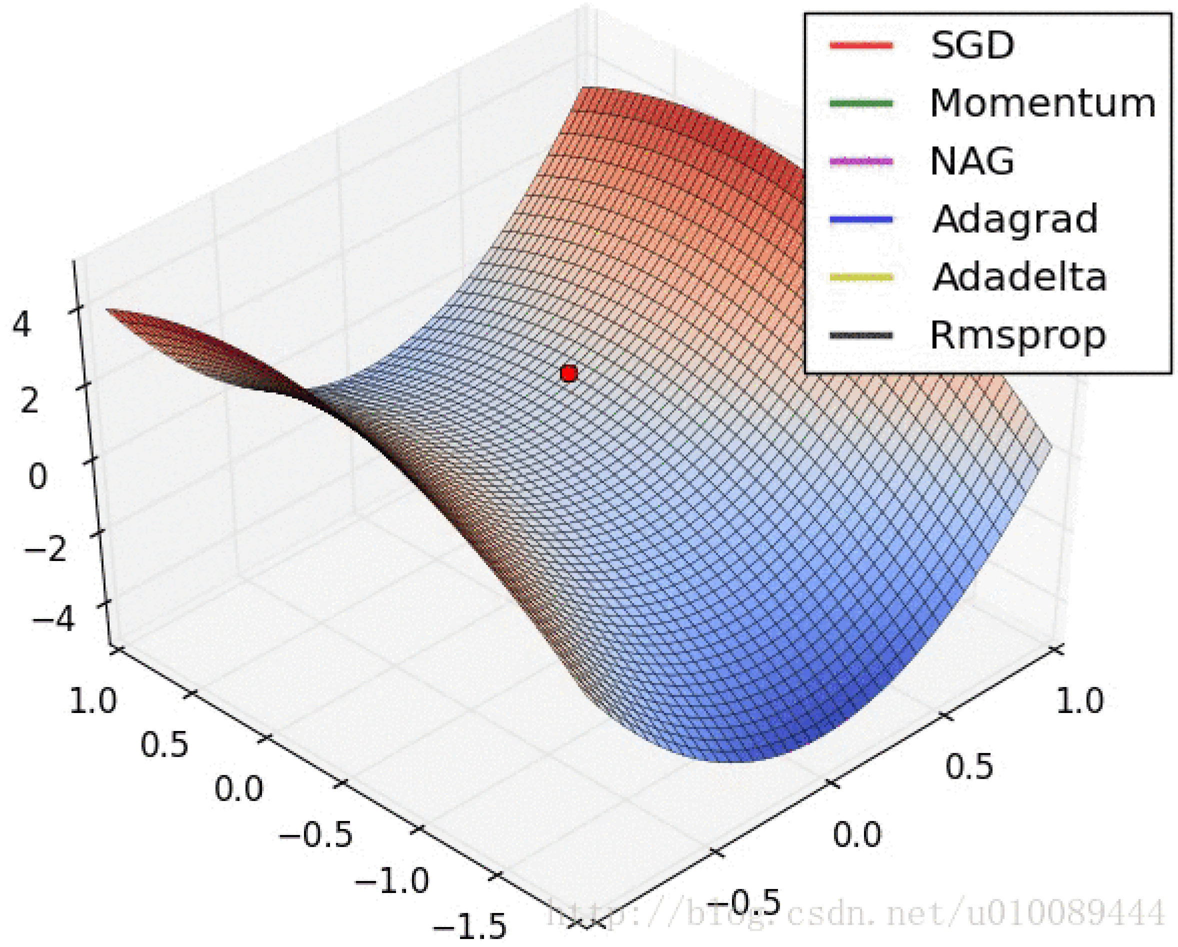
```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared += dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

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)
```

AdaGrad and RMSprop





Adam

```
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))
```

Momentum

Bias correction

AdaGrad / RMSProp

Kingma, D. P., & Ba, J. L. (2015). Adam: a Method for Stochastic Optimization. International Conference on Learning Representations, 1–13.

学习率 Learning Rate

- 学习率会影响收敛性能
  - 训练逐渐收敛时可能需要减小学习率
- 学习率与batch大小有关系
  - 小的batch会导致梯度随机性更大，需要更小的学习率