机器学习
Machine Learning

北京航空航天大学计算机学院智能识别与图像处理实验室
IRIP Lab, School of Computer Science and Engineering, Beihang University
黄 迪　刘庆杰

2018年秋季学期
Fall 2018

Adagrad

- 对稀疏参数进行大幅更新和对频繁参数进行小幅更新
- 适合处理稀疏数据

$$\eta^{\tau} = \frac{1}{\sqrt{\sum_{t=1}^{\tau} \Delta E^2(\mathrm{w}^{\mathrm{t}}) + \epsilon}} \cdot \eta^0$$

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{\tau} - \eta^{\tau} \Delta E(\mathbf{w}^{(\tau)})$$

RMSprop

- **Adagrad引起学习率衰减**

$$\eta^{\tau} = \frac{1}{\sqrt{\boxed{\sum_{t=1}^{\tau} \Delta E^2(\mathrm{w}^{\tau})} + \epsilon}} \eta^{1}$$

学习率衰减

- **减弱梯度累积**

$$\boxed{\mathbf{w}^{(\tau+1)} = \mathbf{w}^{\tau} - \frac{\eta^{0}}{\sqrt{g^{(\tau)}} + \epsilon} \Delta E(\mathbf{w}^{(\tau)})}$$

Adadelta

- 使用前一次的梯度开方 $\sqrt{\mathbf{w}^{(\tau-1)}} + \epsilon$ 代替 η^0

$$\eta^\tau = \frac{\sqrt{\mathbf{w}^{(\tau-1)}}+\epsilon}{\sqrt{g^{(\tau)}}+\epsilon}$$

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^\tau - \frac{\sqrt{\mathbf{w}^{(\tau-1)}} + \epsilon}{\sqrt{g^{(\tau)}} + \epsilon}\Delta E(\mathbf{w}^{(\tau)})$$

动量SGD

- SGD在遇到局部极值和鞍点时容易震荡
- 引入动量momentum，抑制梯度的震荡

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{\tau} - \eta \boxed{\frac{1}{m}\sum_{i=1}^{m}\Delta E(\mathrm{x}_i;\mathbf{w}^{(\tau)})}$$

$$\downarrow$$

$$\nabla_{\mathbf{w}} E^{(\tau)}$$

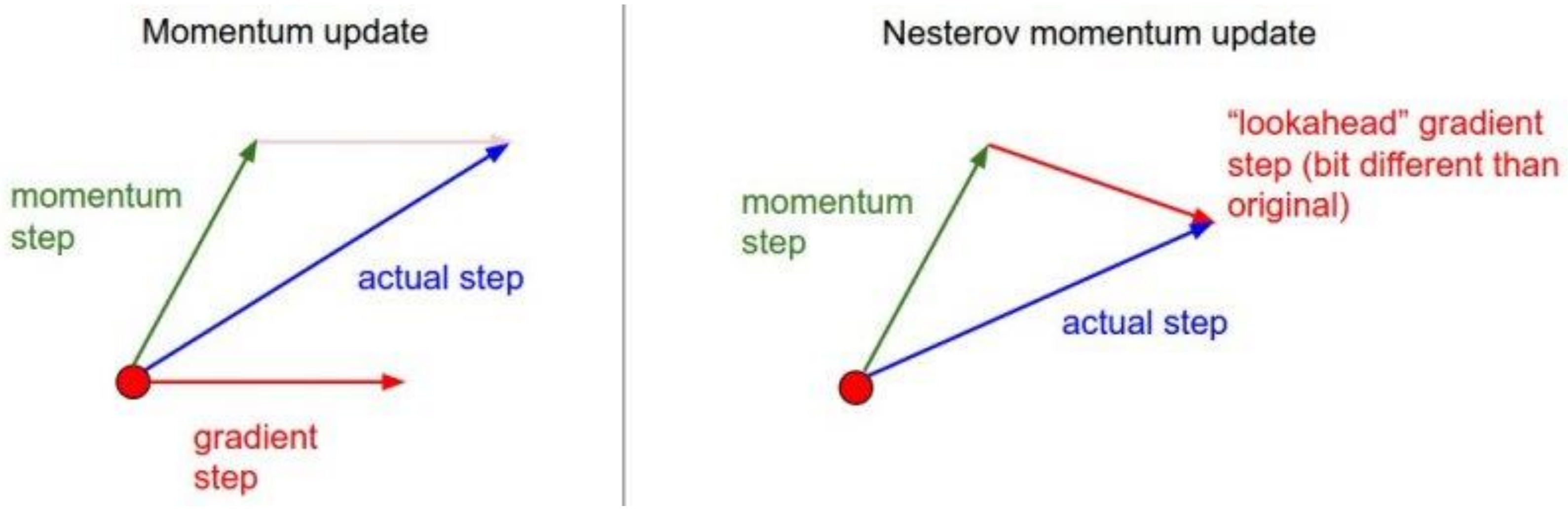
$g^{(\tau)}$ **表示** $\mathcal{T}$ **时刻的优化方向，且** $g^{(0)} = \nabla_{\mathbf{w}} E^{(0)}$

$\mathcal{T}$ **时刻的优化方向为：** $g^{(\tau)} = \alpha g^{(\tau-1)} + \eta \nabla_{\mathbf{w}} E^{(\tau)}$

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{\tau} - g^{\tau}$$

Nesterov梯度（NAG）

- 具有一定的预测性



Adam

- 最常用的方法
- Adaptive + momentum

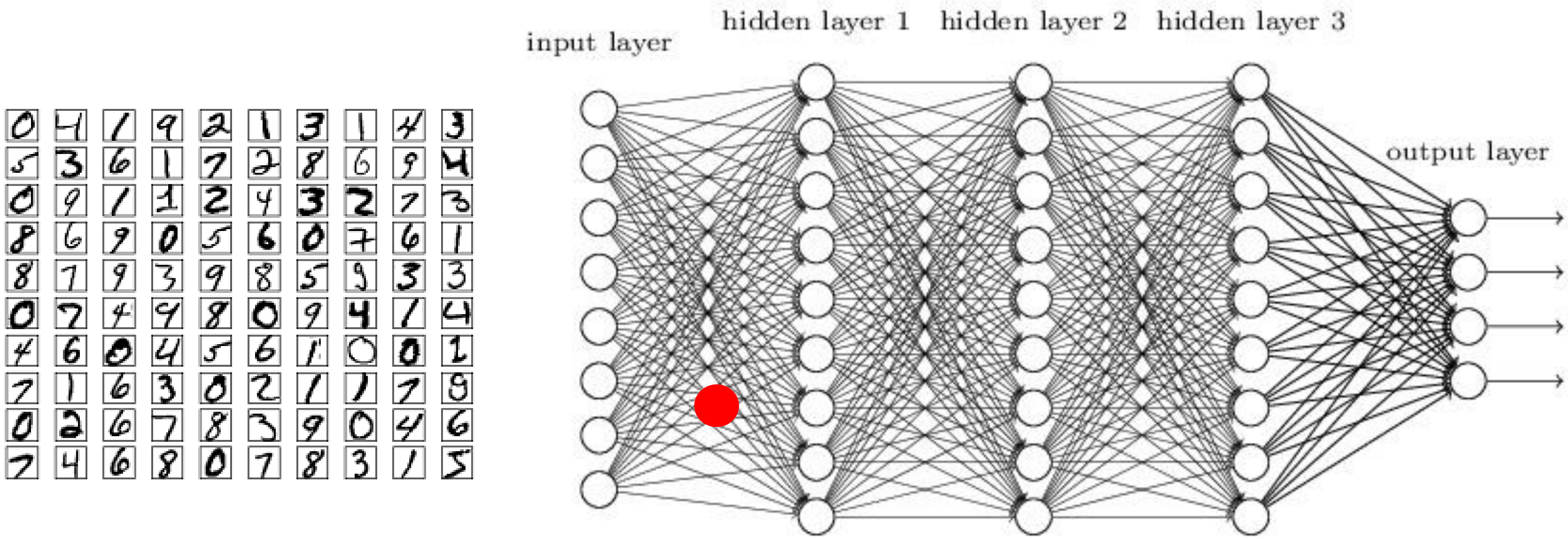
$$m^{(\tau)} = \alpha \cdot m^{(\tau-1)} + (1-\alpha) \cdot \nabla_{\mathbf{w}} E^{(\tau)}$$

$$n^{(\tau)} = \beta \cdot n^{(\tau-1)} + (1-\beta) \cdot [\nabla_{\mathbf{w}} E^{(\tau)}]^2$$

$$\hat{m}^{(\tau)} = \frac{m^\tau}{1-u^\tau} \qquad \hat{n}^{(\tau)} = \frac{m^\tau}{1-v^\tau}$$

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^\tau - \frac{1}{\sqrt{n^\tau} + \epsilon} \hat{m}^\tau$$

梯度消失问题



隐藏层神经元： $$neuro_i^l = \sigma(\mathrm{w}_i^l \mathrm{x}_i^l + \mathrm{b}_i^l)$$

梯度为： $$\delta_i^l = \frac{\partial C}{\partial b_i^l}$$ ➡ $||\delta^l||$ 表示l层的学习速度

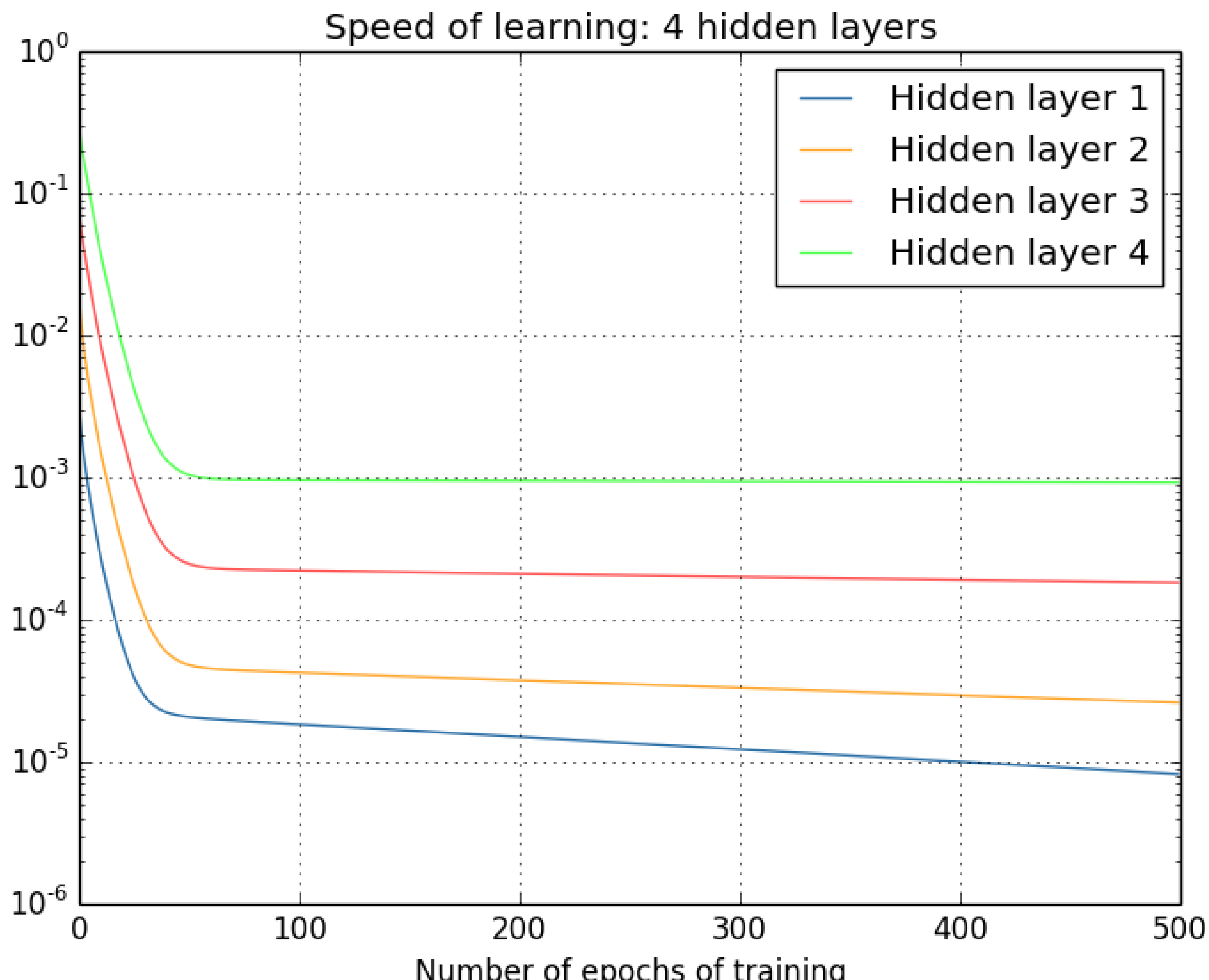
梯度消失问题

- 在MNIST数据集上对[728 30 30 30 30 10]的网络进行训练

初始化：

$$W \sim \mathcal{N}(0,1)$$

$$\eta = 0.1$$



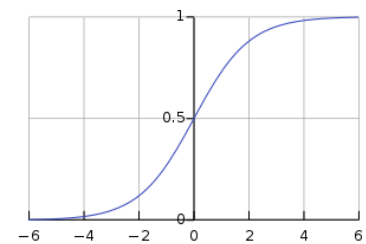
梯度消失问题

- 梯度消失的原因

考虑一个只有一个神经元的多层神经网络

$$\bigcirc \xrightarrow{w_1} b_1 \xrightarrow{w_2} b_2 \xrightarrow{w_3} b_3 \xrightarrow{w_3} b_4 \longrightarrow C$$

$$y_i = \sigma(z_i) = \sigma(w_i x_i + b_i) \quad \sigma - \text{sigmoid}$$



可以推出：

$$\frac{\partial C}{\partial b_1} = \frac{\partial C}{\partial y_4}\frac{\partial y_4}{\partial z_4}\frac{\partial z_4}{\partial x_4}\frac{\partial x_4}{\partial z_3}\frac{\partial z_3}{\partial x_3}\frac{\partial x_3}{\partial z_2}\frac{\partial z_2}{\partial x_2}\frac{\partial x_2}{\partial z_1}\frac{\partial z_1}{\partial b_1}$$

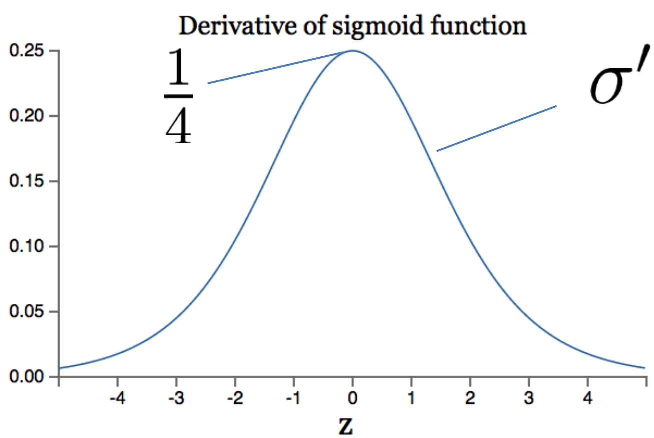
$$= \frac{\partial C}{\partial y_4}\sigma'(z_4)\, w_4 \sigma'(z_3)\, w_3 \sigma'(z_2)\, w_2 \sigma'(z_1)$$

梯度消失问题

- 梯度消失的原因

$$\frac{\partial C}{\partial b_1}=\frac{\partial C}{\partial y_4}\frac{\partial y_4}{\partial z_4}\frac{\partial z_4}{\partial x_4}\frac{\partial x_4}{\partial z_3}\frac{\partial z_3}{\partial x_3}\frac{\partial x_3}{\partial z_2}\frac{\partial z_2}{\partial x_2}\frac{\partial x_2}{\partial z_1}\frac{\partial z_1}{\partial b_1}$$

$$=\frac{\partial C}{\partial y_4}\sigma'\left(z_4\right)w_4\sigma'\left(z_3\right)w_3\sigma'\left(z_2\right)w_2\sigma'\left(z_1\right)$$



$$\left.\begin{array}{c}\sigma'(z)\leq\frac{1}{4}\\ |w|<1\end{array}\right\}\Rightarrow|\sigma'(z)w|\leq\frac{1}{4}$$

梯度消失问题

- 梯度消失的原因

$$\left.\begin{array}{l}\sigma'(z) \le \frac{1}{4} \\ |w| < 1\end{array}\right\} \Rightarrow |\sigma'(z)w| \le \frac{1}{4}$$

$$\frac{\partial C}{\partial b_1} = \sigma'(z_1)\overbrace{w_2\sigma'(z_2)}^{<\frac{1}{4}}\overbrace{w_3\sigma'(z_3)}^{<\frac{1}{4}}\underbrace{w_4\sigma'(z_4)\frac{\partial C}{\partial a_4}}_{\text{common terms}}$$

$$\frac{\partial C}{\partial b_3} = \sigma'(z_3)\overbrace{w_4\sigma'(z_4)\frac{\partial C}{\partial a_4}}^{\text{common terms}}$$

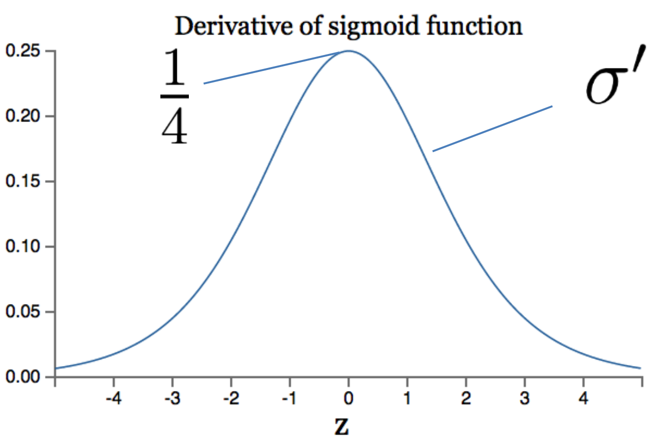
$$\frac{\partial C}{\partial b_1} < \frac{1}{16}\frac{\partial C}{\partial b_3} \ll \frac{\partial C}{\partial b_n},\quad n > 3$$

梯度消失问题

- 梯度消失的原因

$$\frac{\partial C}{\partial b_1}=\frac{\partial C}{\partial y_4}\frac{\partial y_4}{\partial z_4}\frac{\partial z_4}{\partial x_4}\frac{\partial x_4}{\partial z_3}\frac{\partial z_3}{\partial x_3}\frac{\partial x_3}{\partial z_2}\frac{\partial z_2}{\partial x_2}\frac{\partial x_2}{\partial z_1}\frac{\partial z_1}{\partial b_1}$$

$$=\frac{\partial C}{\partial y_4}\sigma'(z_4)w_4\sigma'(z_3)w_3\sigma'(z_2)w_2\sigma'(z_1)$$



$$\left.\begin{aligned}\sigma'(z)&\le\frac{1}{4}\\|w|&<1\end{aligned}\right\}\Rightarrow|\sigma'(z)w|\le\frac{1}{4}$$

$$\Rightarrow\frac{\partial C}{\partial b_1}\to 0$$ 梯度消失

$$|\sigma'(z)w|>1\to$$ 梯度爆炸

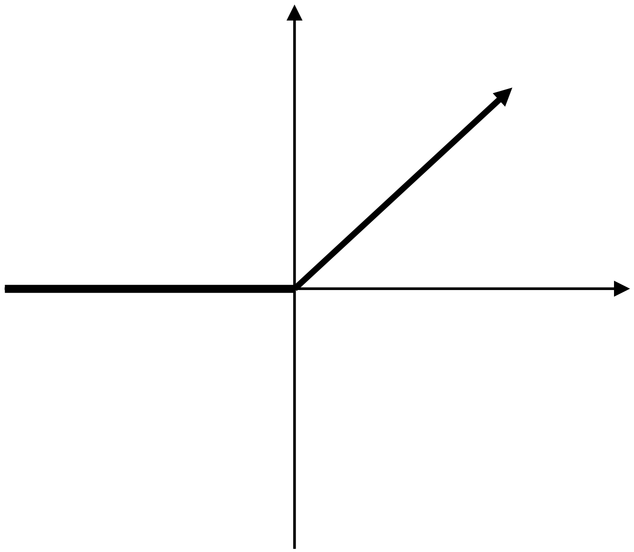
第七讲：神经网络（续）

Chapter 7: Neural Networks

如何有效训练深度神经网络

- 线性整流激活函数（Rectified Linear Units, ReLU）

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

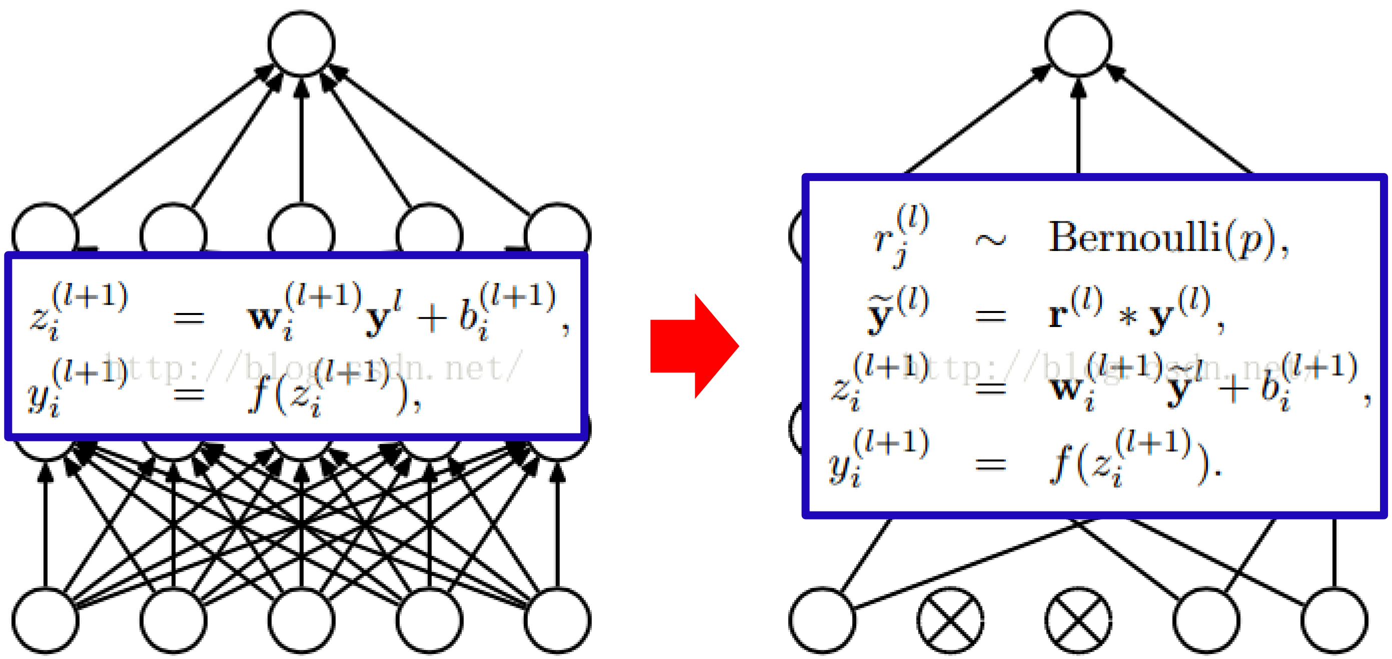


优势：

1. 避免了梯度爆炸和梯度消失问题
2. 简化计算过程
3. 训练稀疏网络

如何有效训练深度神经网络

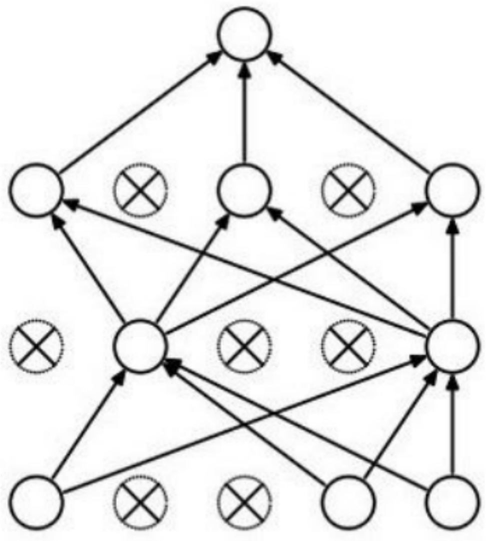
- **正则化—Dropout**
  - Dropout是避免深度神经网络过拟合非常简单而有效的方法

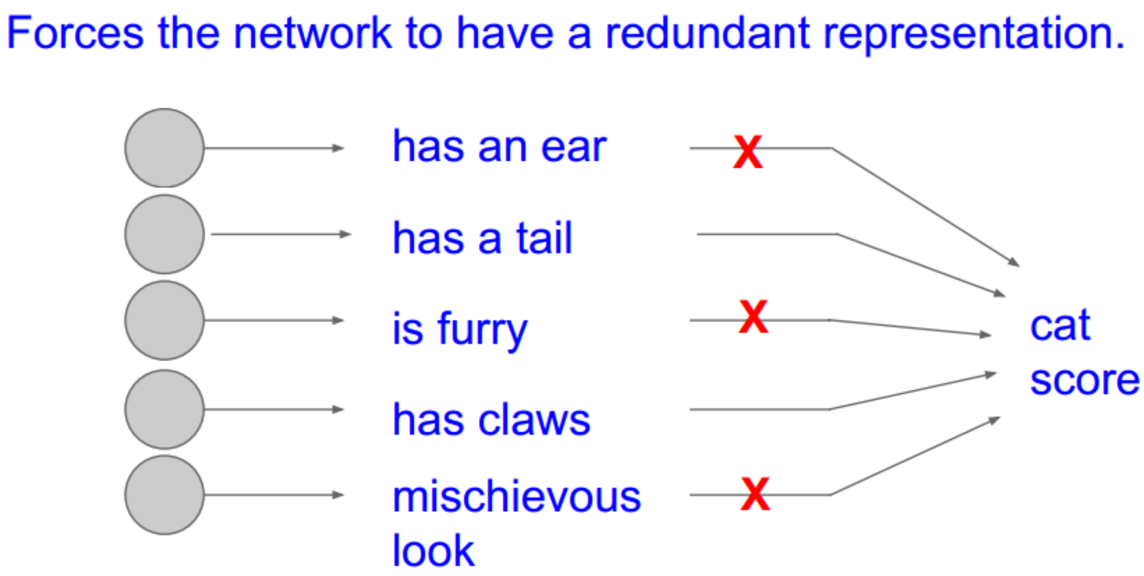


随机“关闭”一些神经元

如何有效训练深度神经网络

- **Dropout为什么能防止过拟合-解释1**
  - 降低模型参数
  - 强制使网络有冗余表示

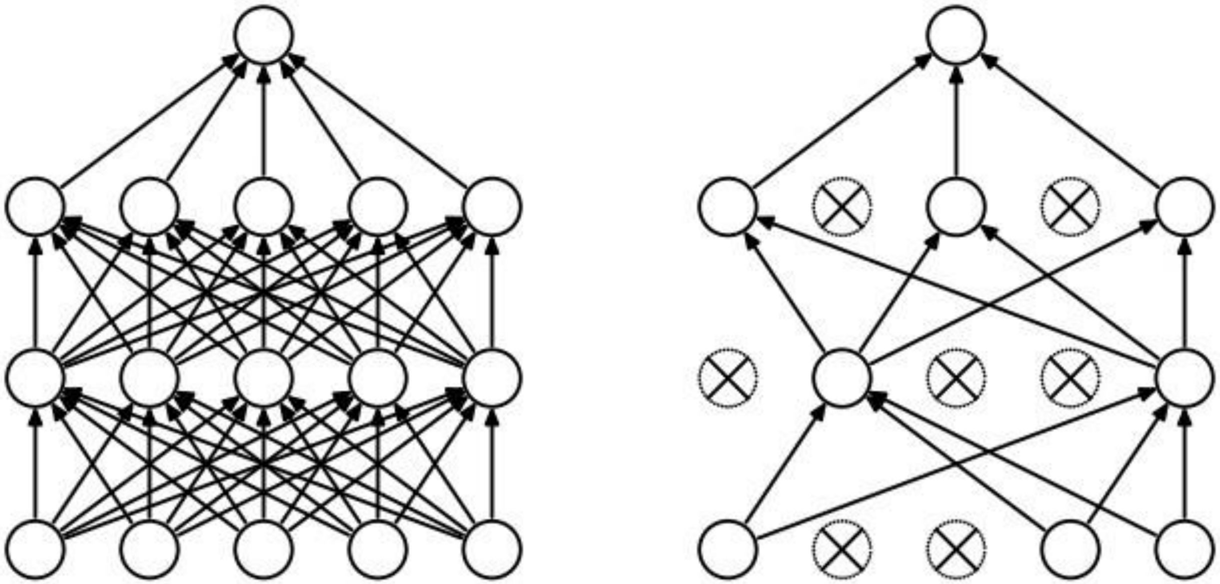




CS231n

如何有效训练深度神经网络

- **Dropout为什么能防止过拟合-解释2**
  - 每次dropout都得到一个新模型
  - 最终结果是多个模型的融合-

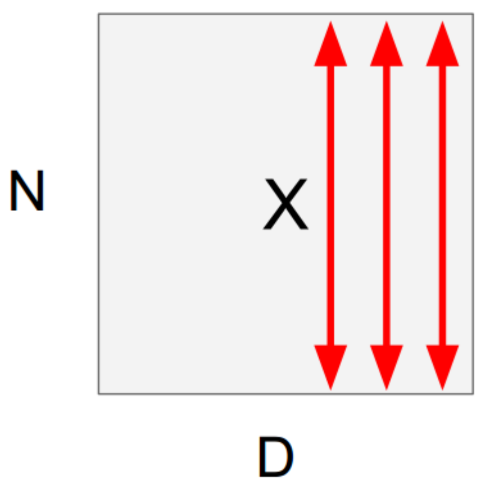


(a) Standard Neural Net　　(b) After applying dropout.

如何有效训练深度神经网络

- **Batch normalization**
  - 对激活后的输出归一化

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



1. compute the empirical mean and variance independently for each dimension.

2. Normalize

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

如何有效训练深度神经网络

- **Batch normalization**
  - 对激活后的输出归一化

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

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

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

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

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

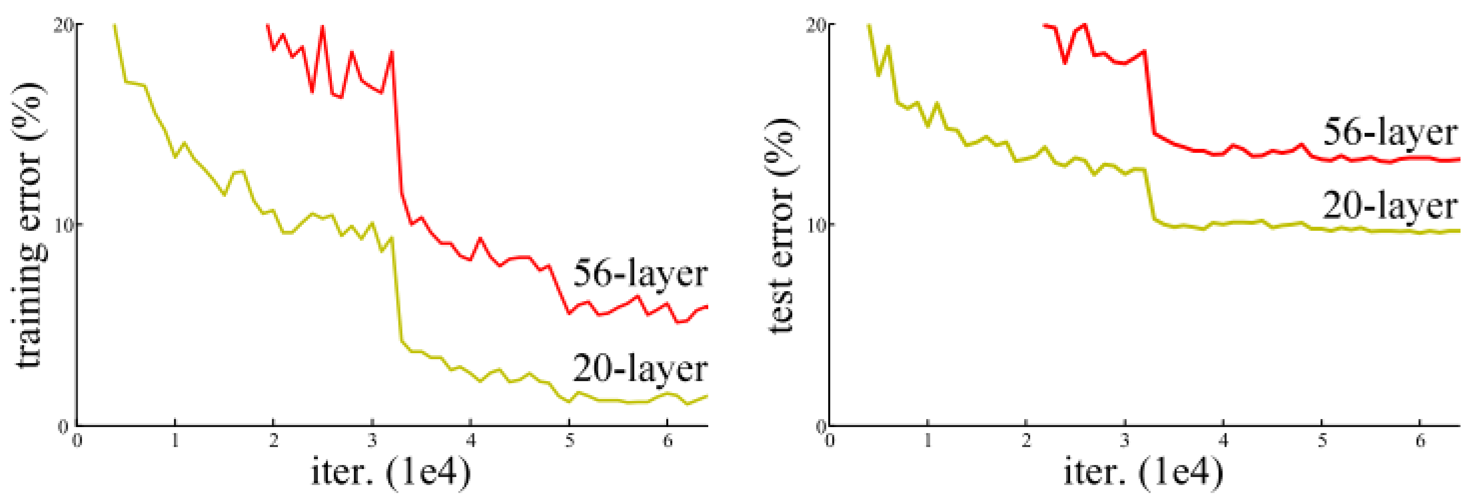
$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

- 可以选择较大的学习率
- 可以减少或不使用dropout
- 缓解梯度消失/爆炸问题

CS231n

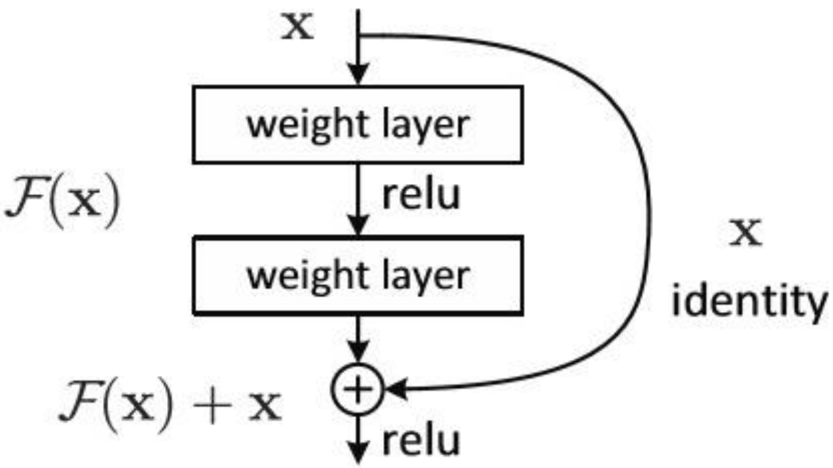
残差神经网络（Residual CNN）

- 越深的网络性能越好 the deeper the better
- 更深的网络带来优化的困难-难收敛
- 训练收敛，又易引起网络退化



残差神经网络（Residual CNN）

- 残差模块Residual block



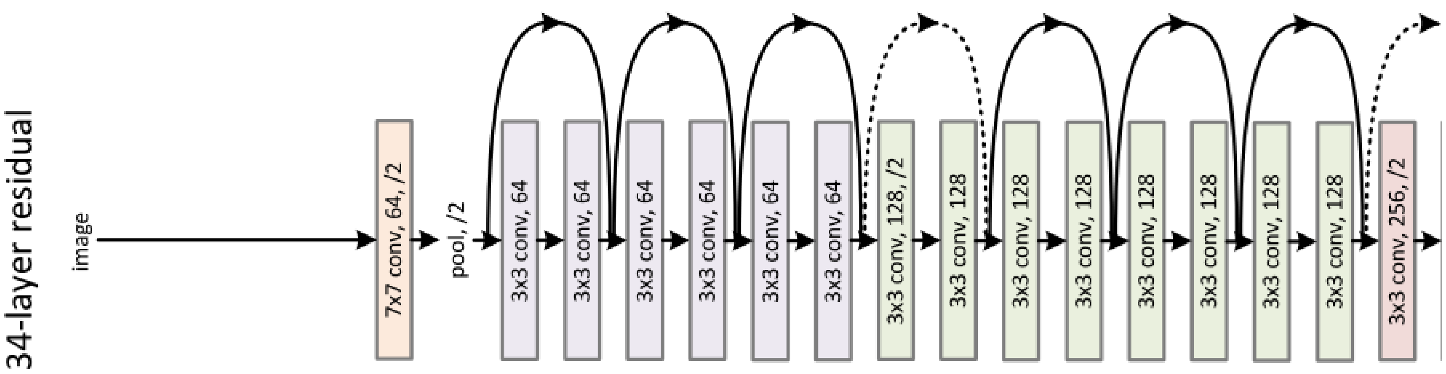
- 在输入和输出之间加入“捷径”连接shortcut connection

$$y = \mathcal{H}(x)$$

$$\mathcal{F}(x) := \mathcal{H}(x) - x$$

残差神经网络（Residual CNN）

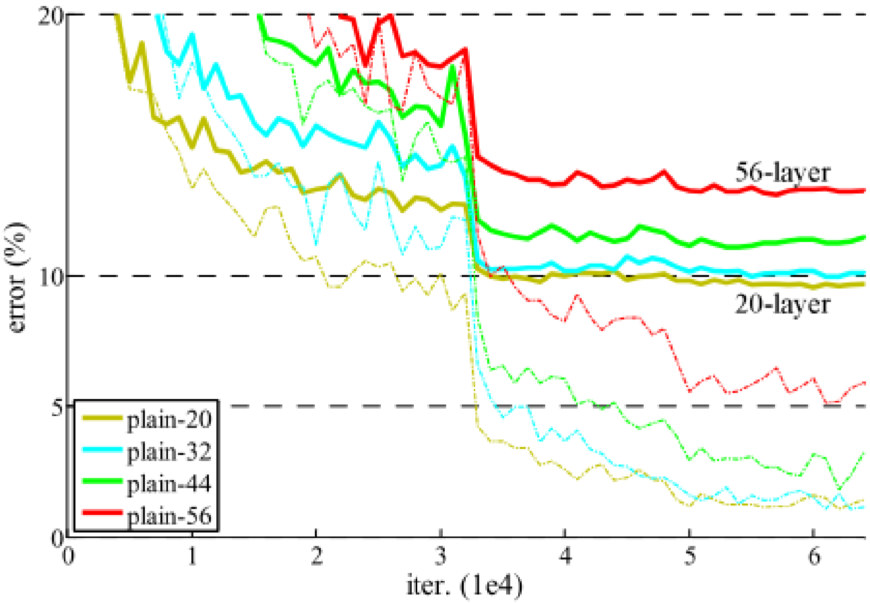
- 残差神经网络**Residual neural network**

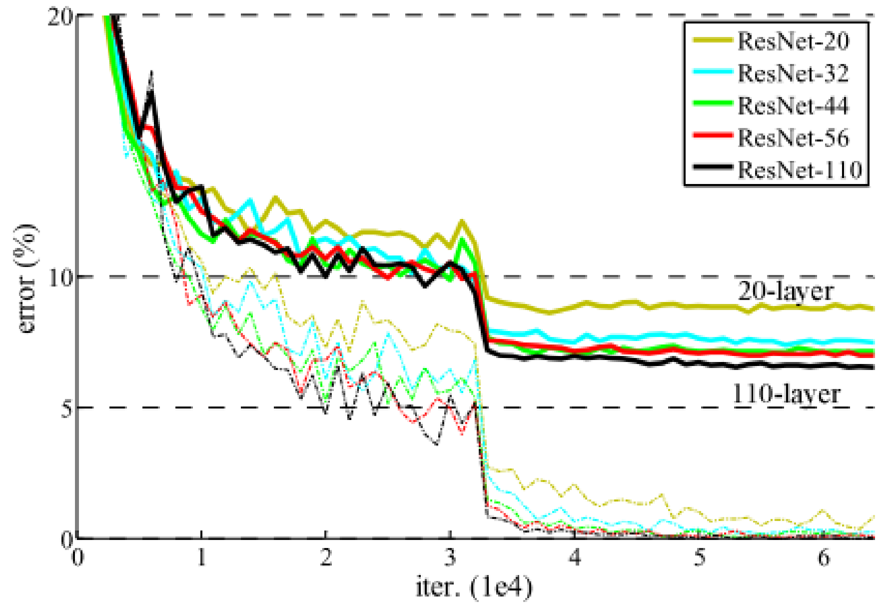


- 构造更深的神经网络，易收敛，不退化
  - ➢ 深达千层
  - ➢ Block内部取消pooling
  - ➢ 用更小的卷积核

残差神经网络（Residual CNN）

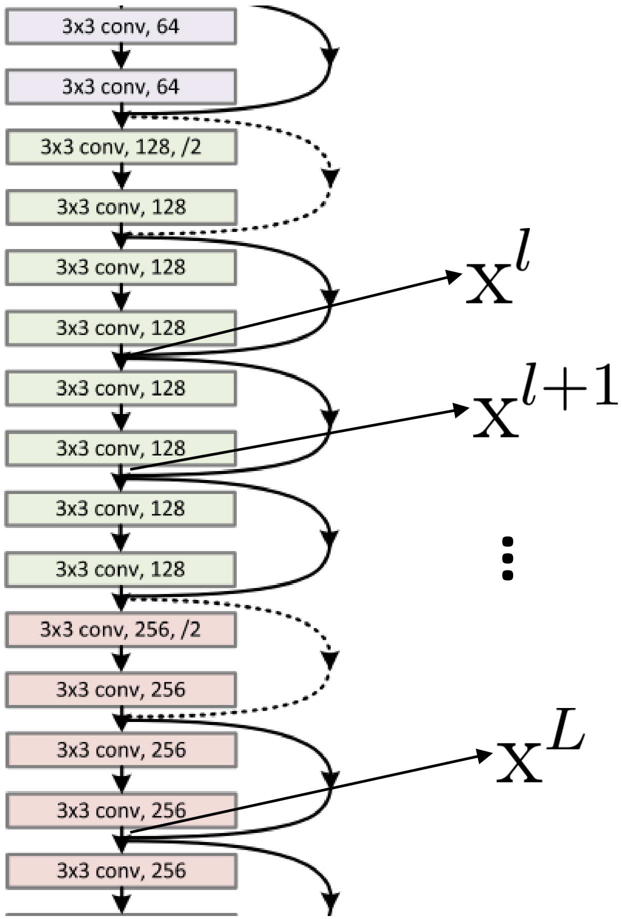
- 残差神经网络**Residual neural network**

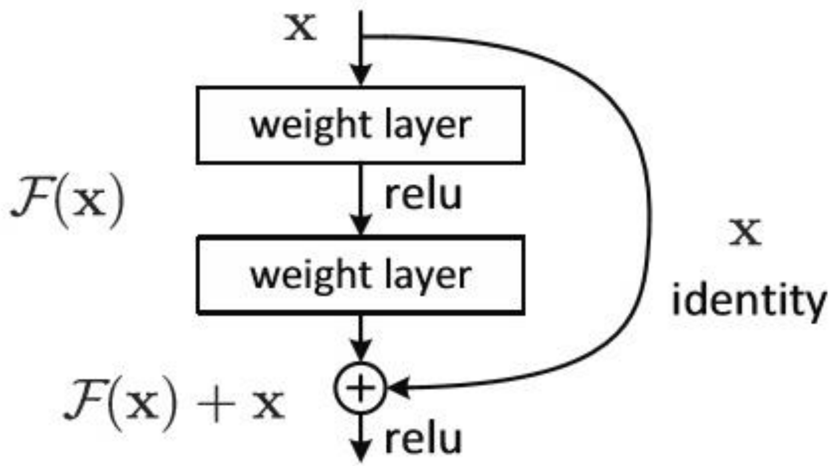




- 深达千层
- Block内部取消pooling
- 用更小的卷积核

残差神经网络（Residual CNN）

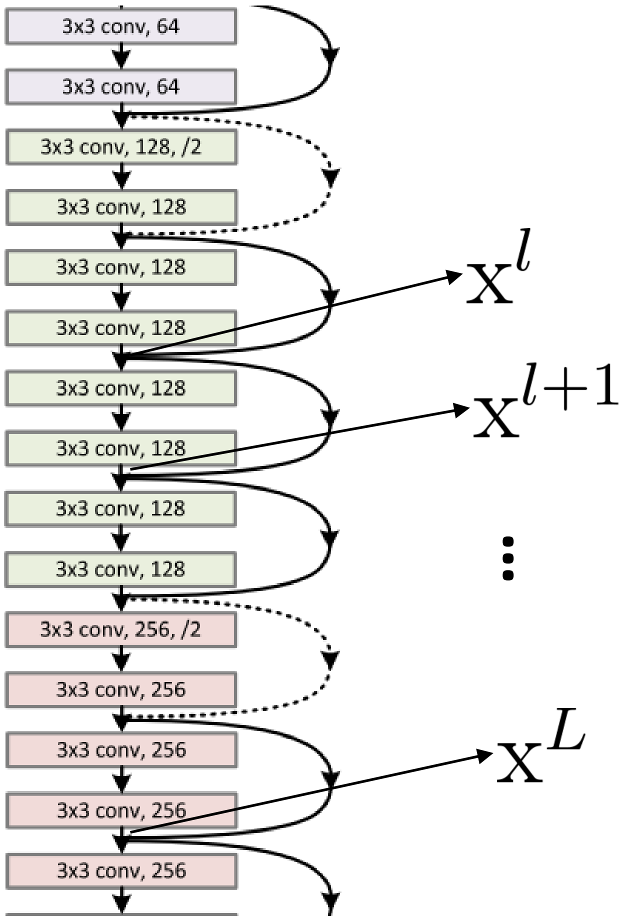




$$\mathrm{x}^{l+1} = \mathrm{x}^l + \mathcal{F}(\mathrm{x}^l)$$

$$\begin{aligned}\mathrm{x}^{l+2} &= \mathrm{x}^{l+1} + \mathcal{F}(\mathrm{x}^{l+1}) \\ &= \mathrm{x}^l + \mathcal{F}(\mathrm{x}^l) + \mathcal{F}(\mathrm{x}^{l+1}) \\ &= \mathrm{x}^l + \sum_{i=l}^{l+1} \mathcal{F}(\mathrm{x}^i)\end{aligned}$$

残差神经网络（Residual CNN）

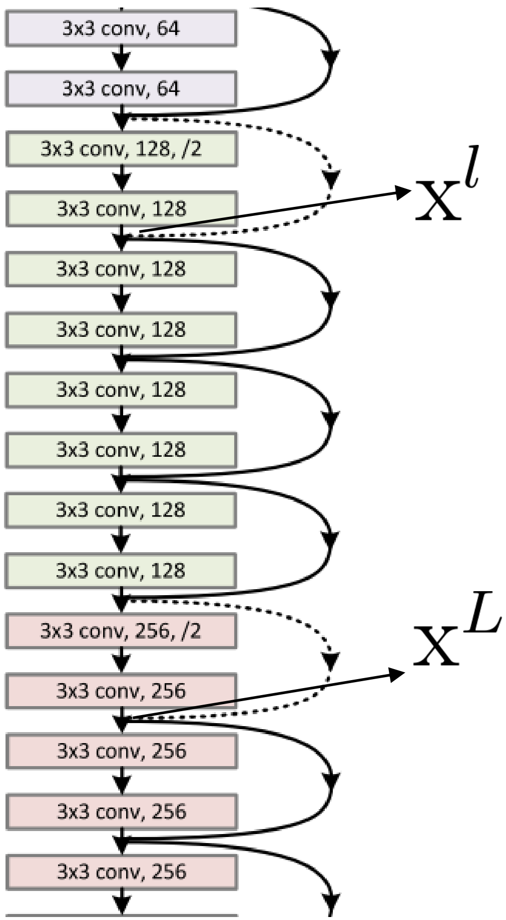


$$\mathrm{x}^{l+1} = \mathrm{x}^{l} + \mathcal{F}(\mathrm{x}^{l})$$

$$\mathrm{x}^{l+2} = \mathrm{x}^{l} + \sum_{i=l}^{l+1} \mathcal{F}(\mathrm{x}^{i})$$

$$\boxed{\mathrm{x}^{L} = \mathrm{x}^{l} + \sum_{i=l}^{L-1} \mathcal{F}(\mathrm{x}^{i})}$$

残差神经网络（Residual CNN）

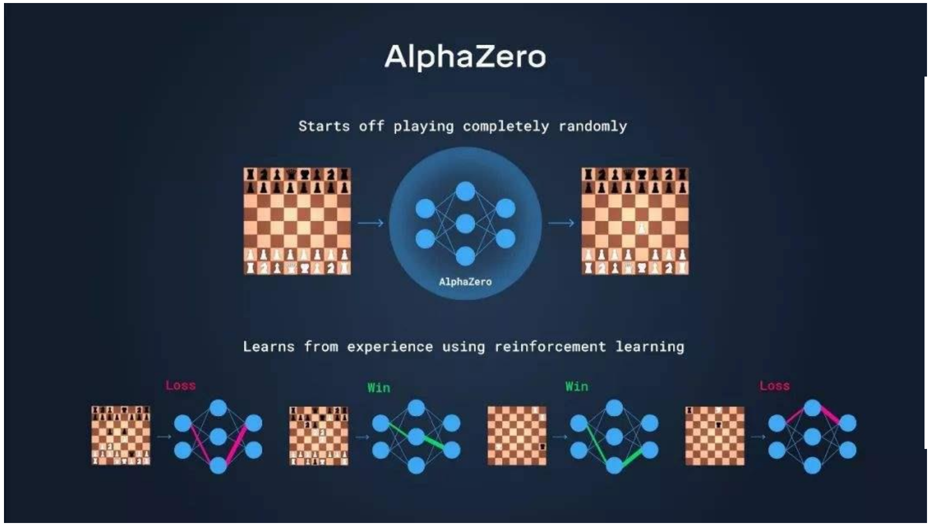


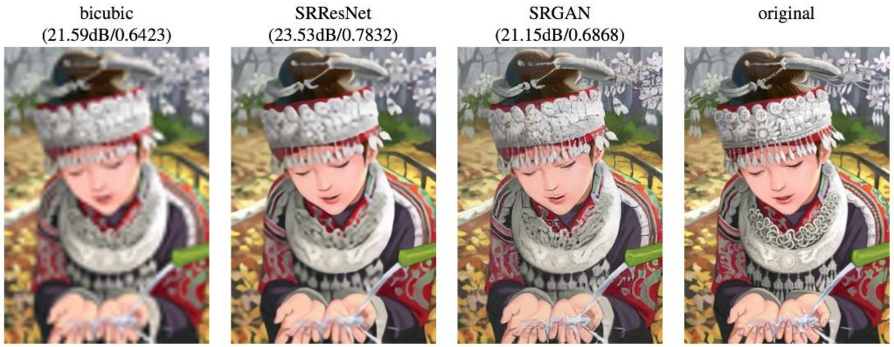
损失函数： E **求** $\dfrac{\partial E}{\partial \mathrm{x}^l}$

$$\frac{\partial E}{\partial \mathrm{x}^l} = \frac{\partial E}{\partial \mathrm{x}^L}\frac{\partial \mathrm{x}^L}{\partial \mathrm{x}^l}$$

$$= \frac{\partial E}{\partial \mathrm{x}^L}\left(1 + \frac{\partial \sum_{i=l}^{L-1} \mathcal{F}(\mathrm{x}^i)}{\partial \mathrm{x}^l}\right)$$

残差网络的成功应用


AlphaZero
Starts off playing completely randomly
AlphaZero
Learns from experience using reinforcement learning
Loss
Win
Win
Loss


bicubic
(21.59dB/0.6423)
SRResNet
(23.53dB/0.7832)
SRGAN
(21.15dB/0.6868)
original

循环神经网络（Recurrent neural Network，RNN）

- 序列问题

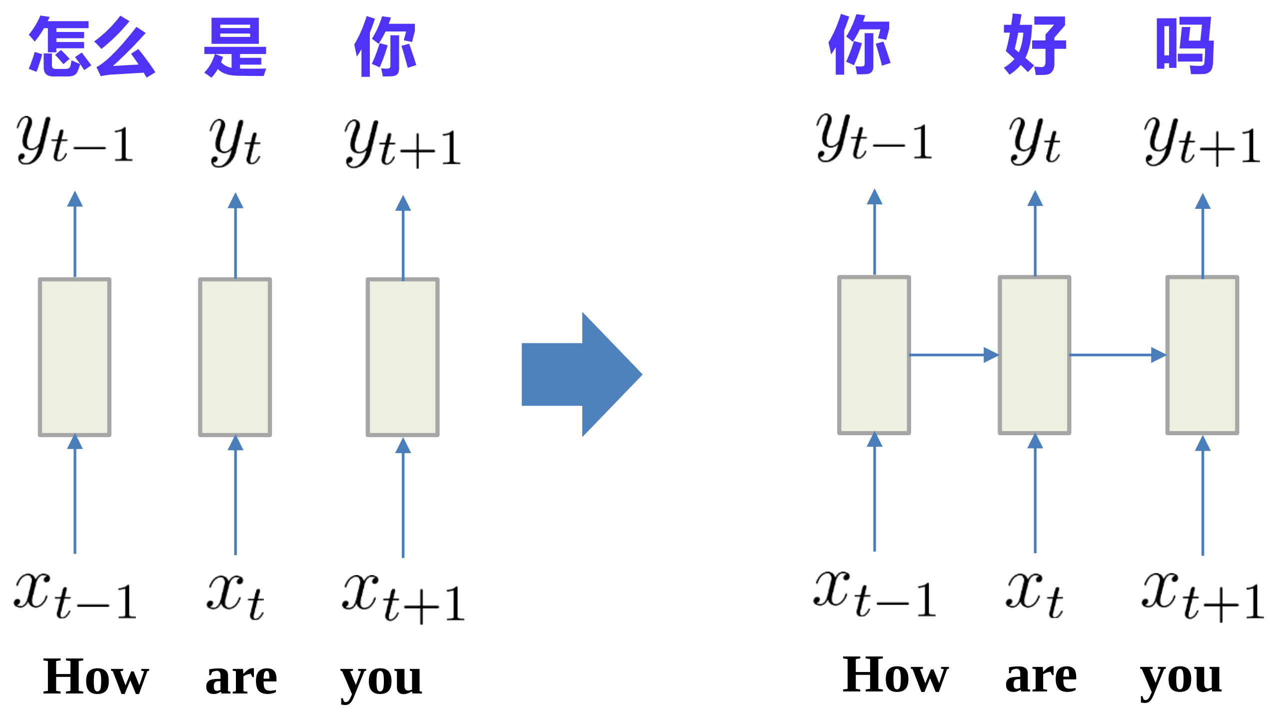
循环神经网络（Recurrent neural Network，RNN）

- 序列问题

输入：

x_1　x_2　x_3　x_4　x_5　……　x_t

循环神经网络（Recurrent neural Network，RNN）



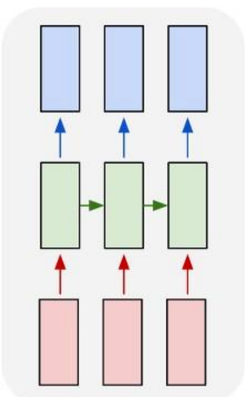
输入：

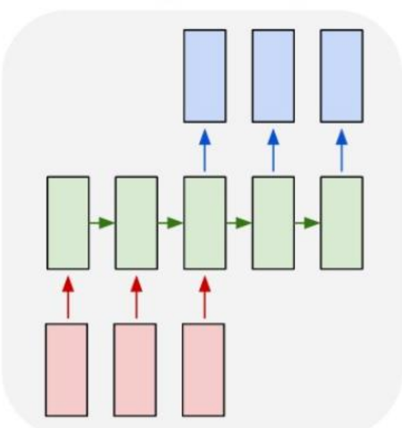
x_1 x_2 x_3

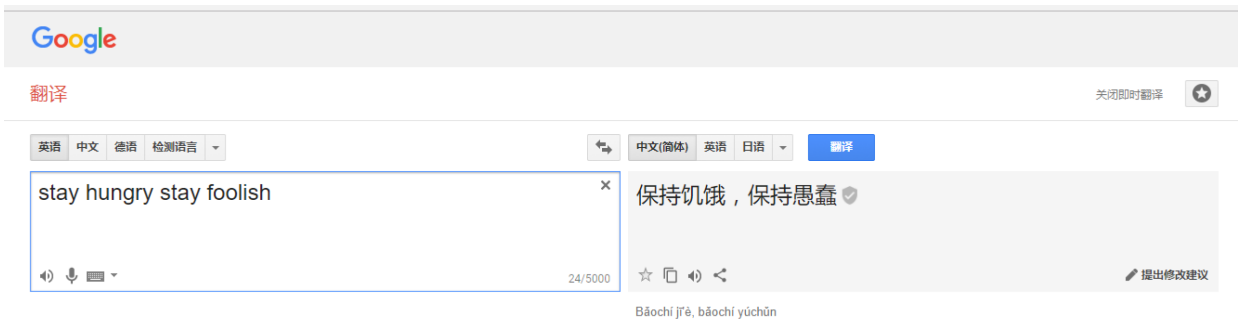
How are you $\xrightarrow{\text{翻译}}$ 中文意思？

循环神经网络（Recurrent neural Network，RNN）

- 多对多





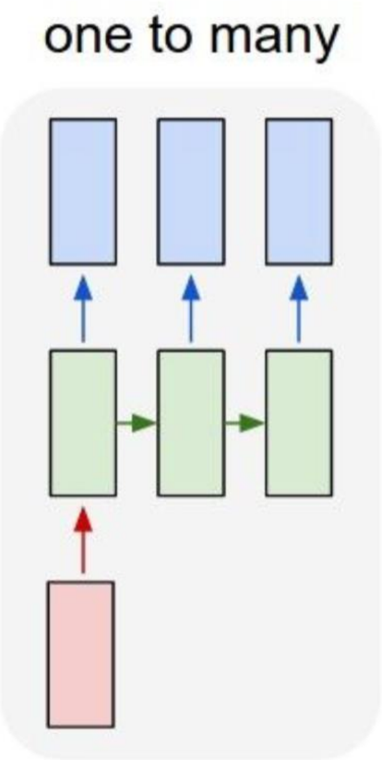


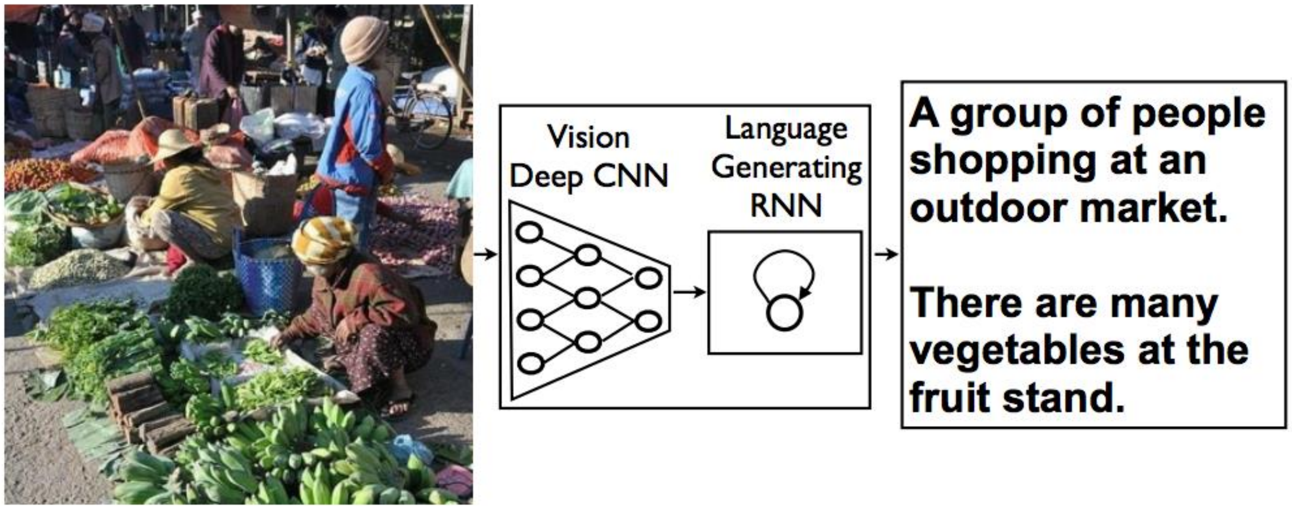
1. A white car is drifting.
2. Cars racing on a road surrounded by lots of people.
3. Cars are racing down a narrow road.
4. A race car races along a track.
5. A car is drifting in a fast speed.

1. A player is putting the basketball into the post from distance.
2. The player makes a three-pointer.
3. People are playing basketball.
4. A 3 point shot by someone in a basketball race.
5. A basketball team is playing in front of speculators.

循环神经网络（Recurrent neural Network，RNN）

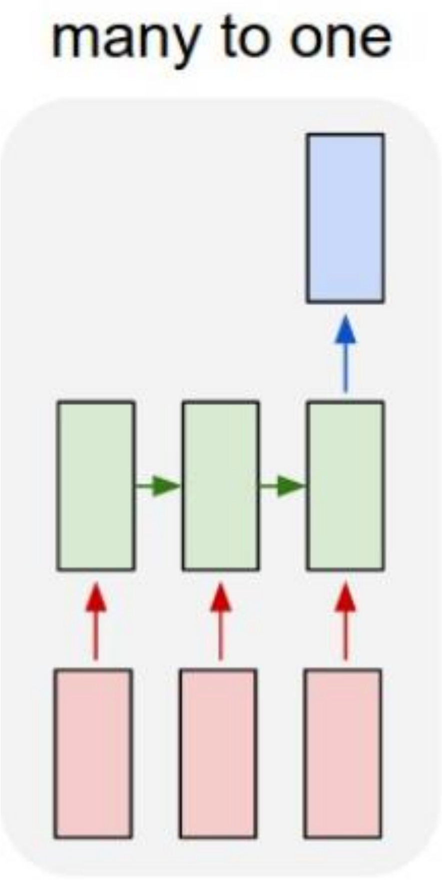
● 一对多





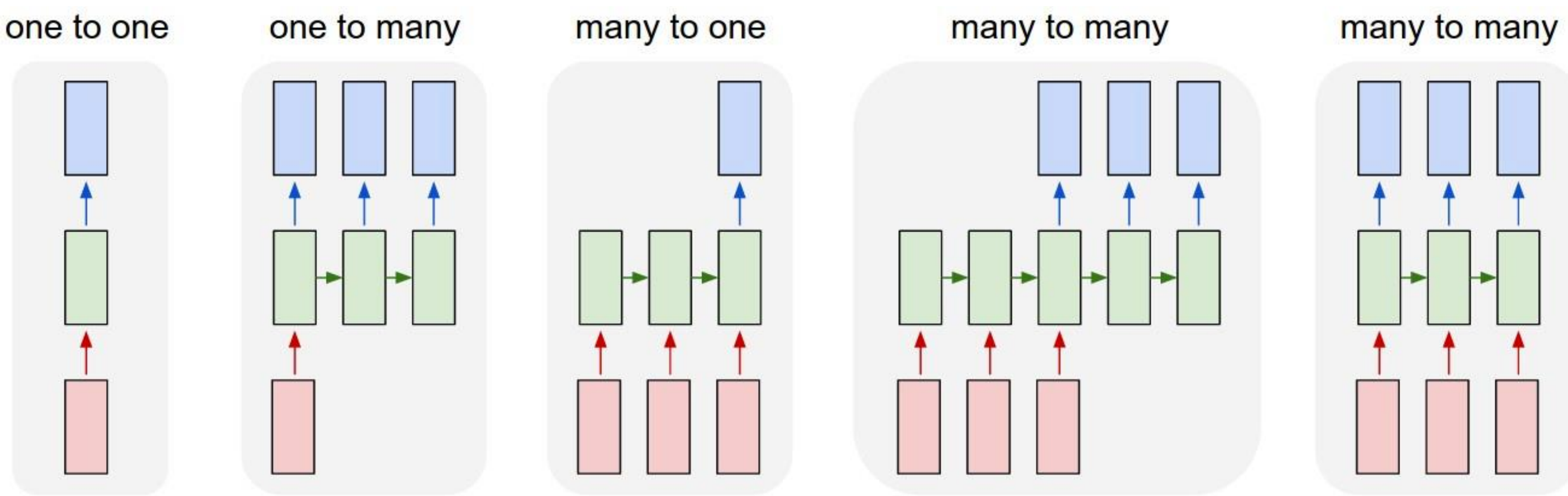
循环神经网络（Recurrent neural Network，RNN）

- 多对一



循环神经网络（Recurrent neural Network，RNN）

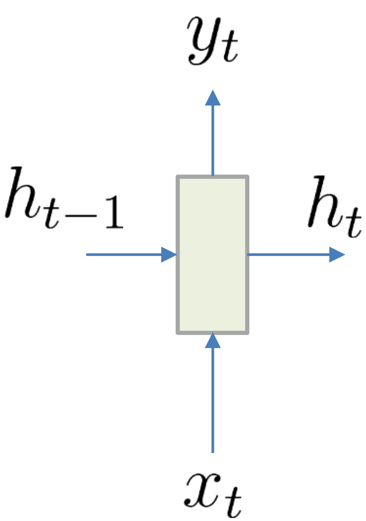
- **RNN处理序列问题**



- 既能处理序列输入，也能得到序列输出

循环神经网络（Recurrent neural Network，RNN）

- **RNN由输入，隐状态、及输出三部分组成**



当前输入： x_t

上一刻状态（历史信息）： h_{t-1}

更新当前状态： h_t

$$h_t = \sigma\left(W^{(hh)} h_{t-1} + W^{(hx)} x_t\right)$$

输出： y_t

$$y_t = \mathrm{softmax}\left(W^{(S)} h_t\right)$$

循环神经网络（Recurrent neural Network，RNN）

- **RNN训练-Back Prop Through Time(BTPP)**
  - 计算*W*的偏导，需要对每个*t*求偏导

$$\frac{\partial E}{\partial W} = \sum_{t=1}^{T} \frac{\partial E_t}{\partial W}$$

  - 链式法则

$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \boxed{\frac{\partial h_t}{\partial h_k}} \frac{\partial h_k}{\partial W}$$

循环神经网络（Recurrent neural Network，RNN）

- 已知 $$h_t = W f(h_{t-1}) + W^{(hx)} x_{[t]}$$
- 可得 $$\frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}}$$

$$\frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} = \prod_{j=k+1}^{t} W^T \mathrm{diag}[f'(h_{j-1})]$$

$$\mathrm{diag}(z) = \begin{pmatrix} z_1 & & & & \\ & z_2 & & 0 & \\ & & \ddots & & \\ & 0 & & z_{n-1} & \\ & & & & z_n \end{pmatrix}$$

循环神经网络（Recurrent neural Network，RNN）

- RNN中gradient vanishing/exploding
  - 已知 $$\frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} = \prod_{j=k+1}^{t} W^T \mathrm{diag}[f'(h_{j-1})]$$
  - 根据 $||xy|| \leq ||x|| \cdot ||y||$

$$\left\|\frac{\partial h_j}{\partial h_{j-1}}\right\| \leq \|W^T\|\|\mathrm{diag}[f'(h_{j-1})]\| \leq \beta_W \beta_h$$

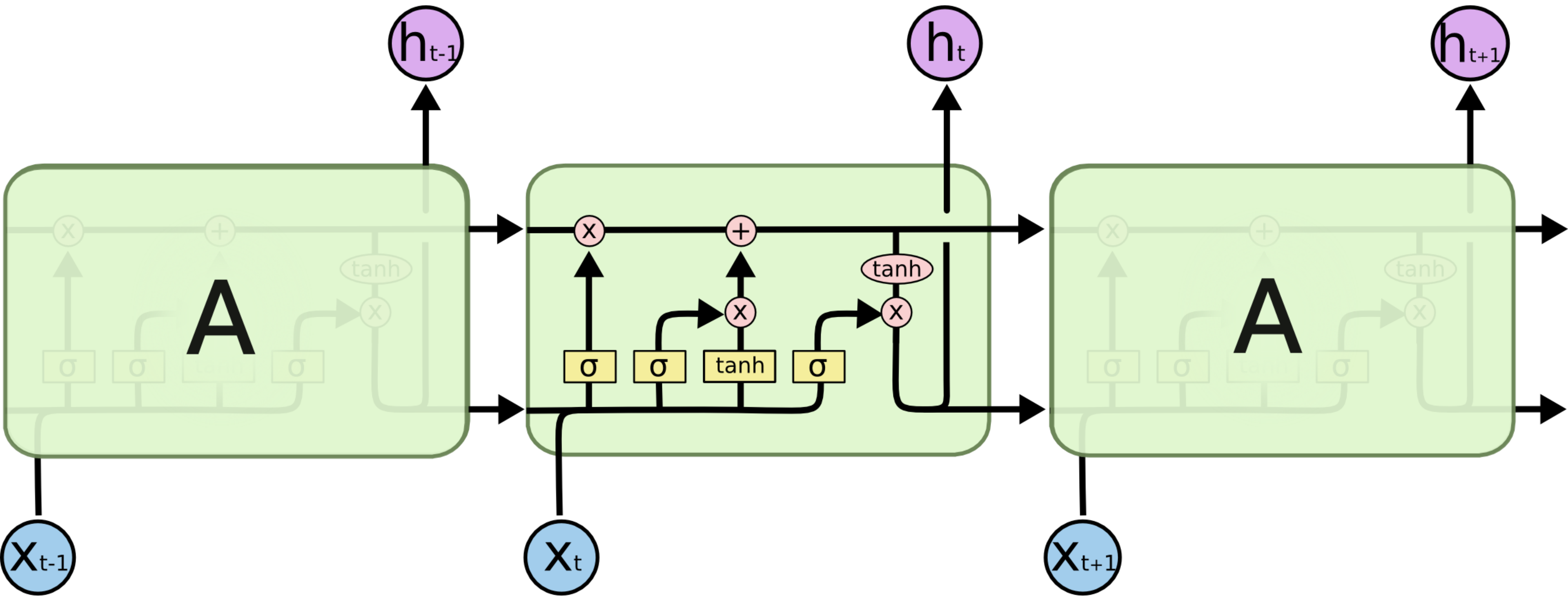
代表上限

$$\left\|\frac{\partial h_t}{\partial h_k}\right\| = \left\|\prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}}\right\| \leq (\beta_W \beta_h)^{t-k}$$

当t与k间隔较远时，梯度会很快的变的很大或很小

Long Short Term Memory (LSTM)

- **LSTM由重复的单元连接而成**

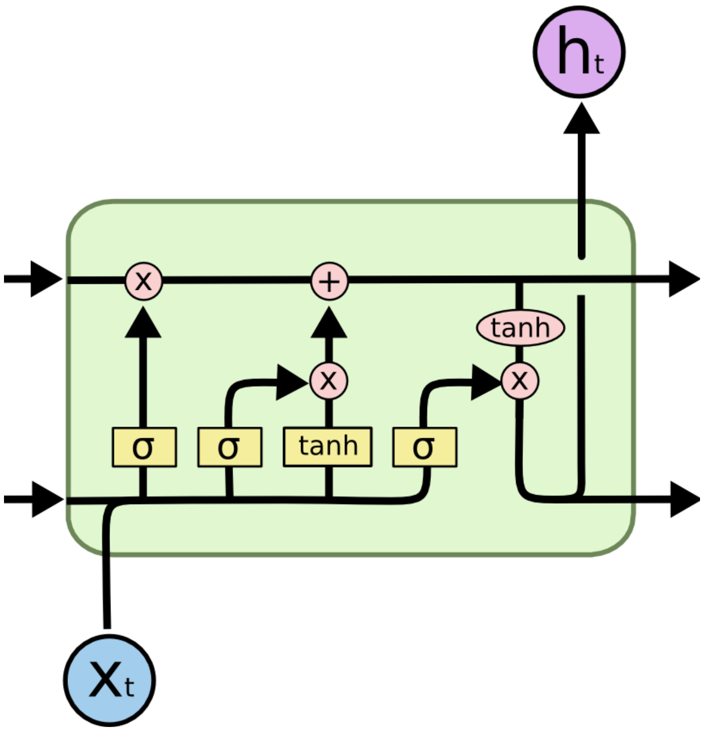


- **当成最成功，应用最广泛的神经网络结构之一**

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

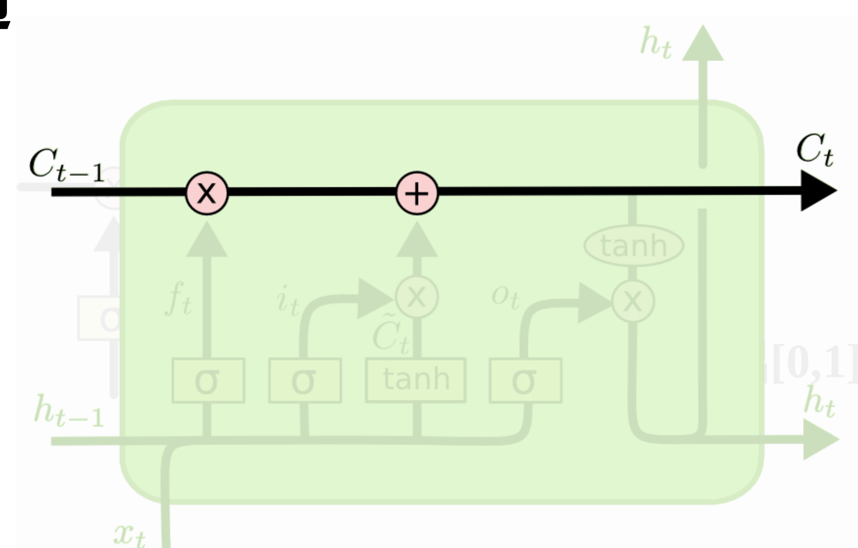
Long Short Term Memory (LSTM)

- **LSTM单元（cell）组成**



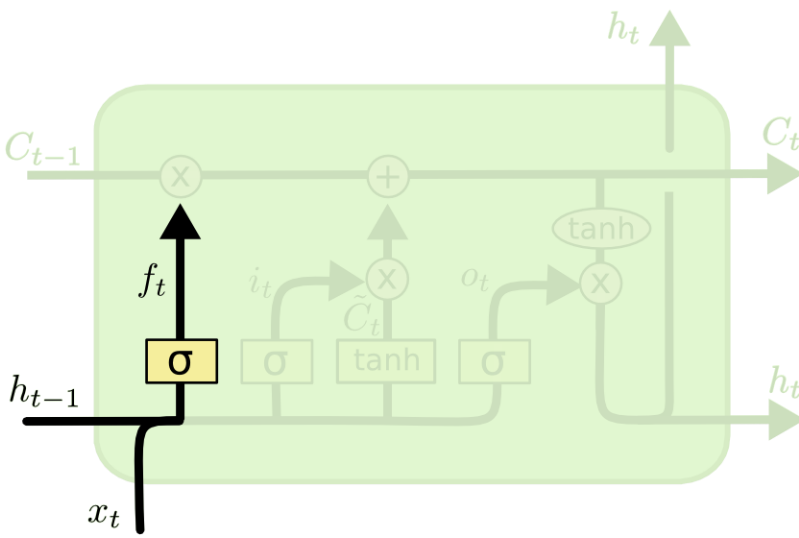
神经单元　按位操作　前向传播　合并串联　复制

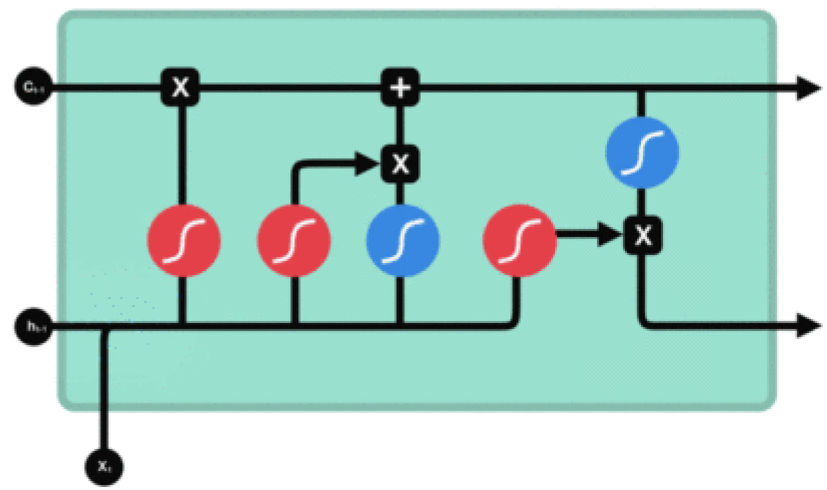
➢ **LSTM通过门控单元控制信息的流通**



Long Short Term Memory (LSTM)

- “遗忘门”





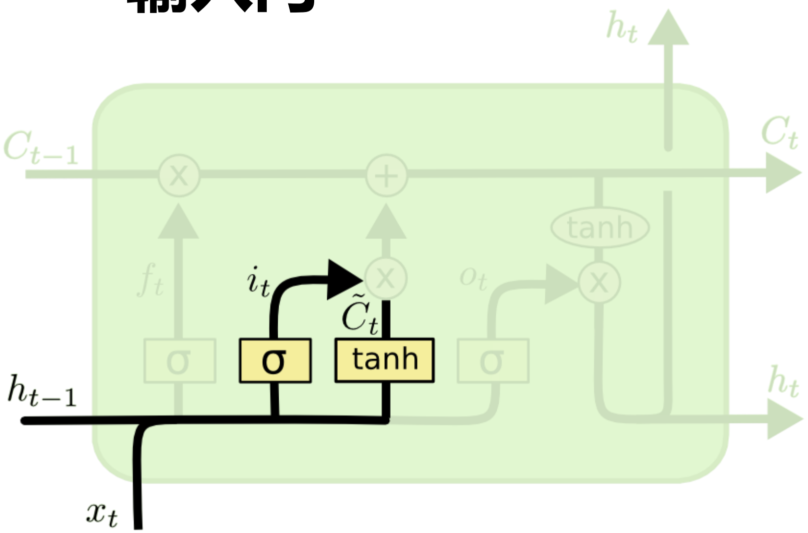
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

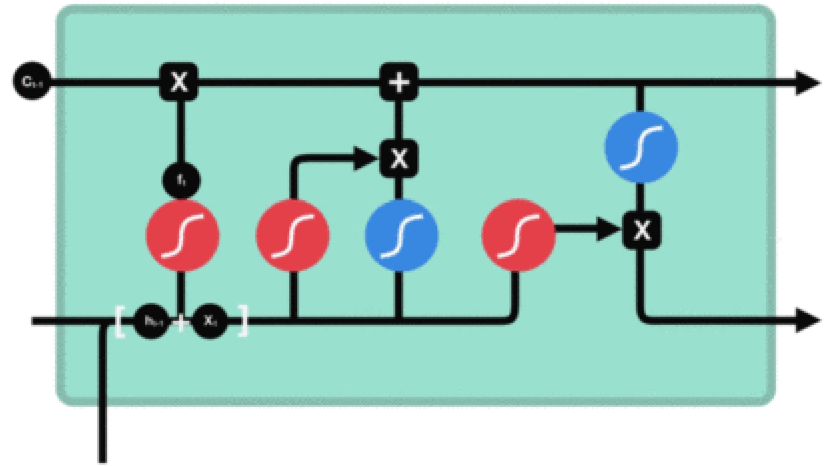
➢ **选择哪些信息应该被遗忘**

先前输出（即隐藏状态）的信息和来自当前输入的信息经sigmoid函数激活，输出介于0-1之间。越接近0意味着越容易被忘记，越接近1则越容易被保留。

Long Short Term Memory (LSTM)

- **“输入门”**





$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

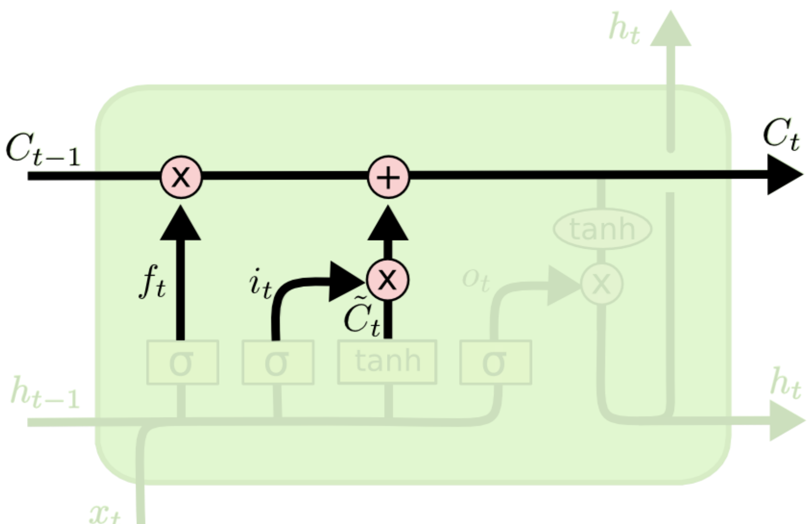
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

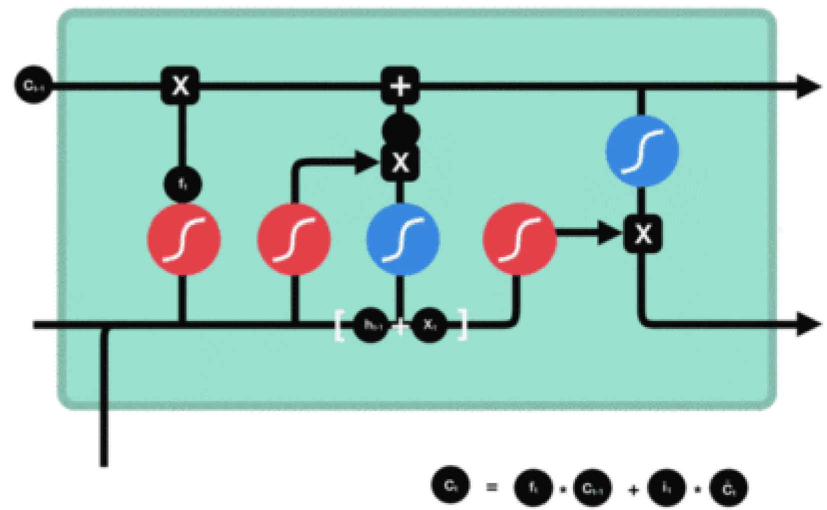
➢ **Cell新状态**

1. 先前输出和当前输入经sigmoid函数，计算出哪些值更重要；
2. 同时，把先前输出和当前输入给tanh函数，生成候选状态；
3. 最后，把tanh的输出与sigmoid的输出相乘，生成更新状态

Long Short Term Memory (LSTM)

- **状态更新**





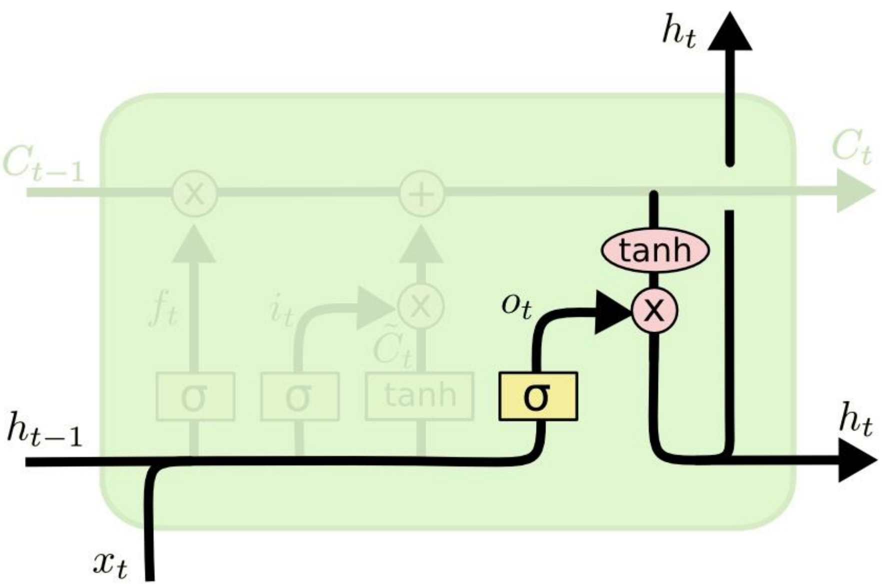
$$C_t = f_t \otimes C_{t-1} + i_t \otimes \tilde{C}_t$$

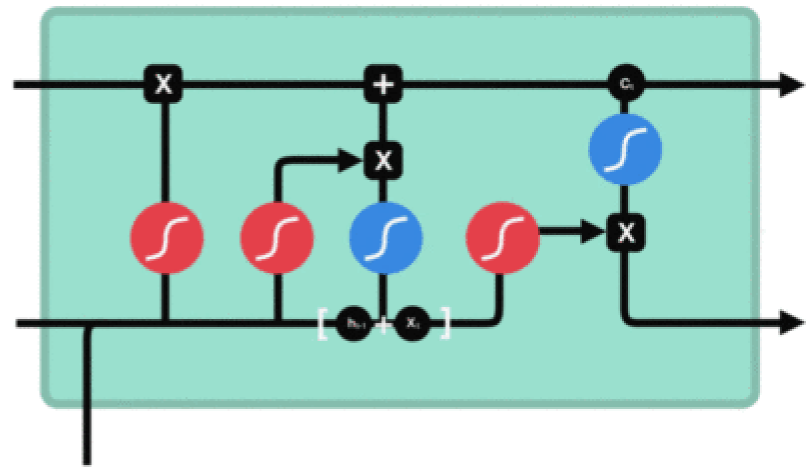
➢ **更新cell状态**

1. 先前cell状态和遗忘门输出的向量点乘，由于越不重要的值越接近0，原隐藏状态中不重要的信息被丢弃。
2. 新的输出，与当前cell的候选状态相加，输出更新后的cell状态。

Long Short Term Memory (LSTM)

- **输出**





$$o_t = \sigma(W \cdot [h_{t-1}, x_t] + b_o)$$
$$h_t = o_t \otimes \tanh(C_t)$$

➢ **Cell输出**

1. 输出建立在cell状态基础上
2. 先前输出与当前输入经过sigmod，决定哪一部分cell状态需要被输出-输出门
3. 状态经过tanh后，与输出门相乘，只输出想要输出的。

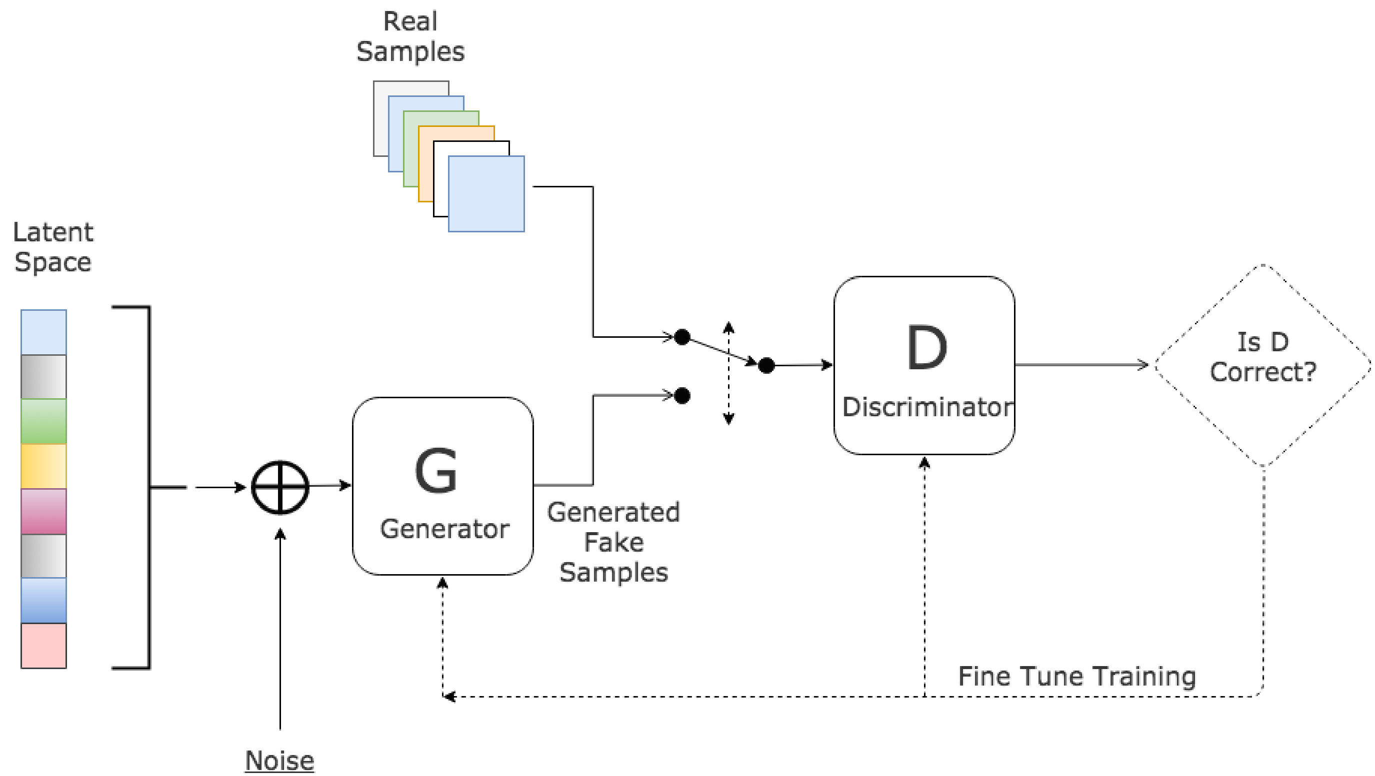
Generative Adversarial Networks(GAN)

- **Generative Adversarial Network（GAN），生成对抗网络由Ian Goodfellow于2014年在一篇发表在NIPS的文章中提出**

- **GAN是一个生成模型**
- **二人零和博弈中受启发**
- **由一个生成器网络generator 和一个判别器网络discriminator组成**
- ***G*试图生成接近真实的数据，以“骗过”*D*，*D* 则以更好的区分生成的和真实的数据为学习目标**

Generative Adversarial Networks(GAN)

- **Generative Adversarial Network（GAN）结构**



Generative Adversarial Networks(GAN)

- **Generative Adversarial Network（GAN）结构**

- **假设有真实的数据集，分布为** $P_{data}(x)$ $\sim x$ **是一张人脸**
- 假设一个生成器G生成的分布是$P_G(x;\theta)$
- 假设我们在真实分布中取一些数据 $\{x^1, x^2, \cdots, x^m\}$
- 这些数据是从生成器中得到的似然是 $L = \prod_{i=1}^{m} P_G(x^i;\theta)$
- 最大化该似然，即拟合生成器对真实数据的分布

Generative Adversarial Networks(GAN)

- **Generative Adversarial Network（GAN）结构**

➢ 最大化该似然，即拟合生成器对真实数据的分布

$$
\begin{aligned}
\theta^* &= \arg\max_\theta \prod_{i=1}^{m} P_G(x^i;\theta) \\
&= \arg\max_\theta \ log \prod_{i=1}^{m} P_G(x^i;\theta) \\
&= \arg\max_\theta \sum_{i=1}^{m} logP_G(x^i;\theta) \\
&\approx \arg\max_\theta \ E_{x\sim P_{data}}[logP_G(x;\theta)] \\
&= \arg\max_\theta \int_x P_{data}(x) logP_G(x;\theta)dx - \int_x P_{data}(x) logP_{data}(x)dx \\
&= \arg\max_\theta \int_x P_{data}(x)(logP_G(x;\theta) - logP_{data}(x))dx \\
&= \arg\min_\theta \int_x P_{data}(x) log\frac{P_{data}(x)}{P_G(x;\theta)}dx \\
&= \arg\min_\theta \ KL(P_{data}(x)||P_G(x;\theta))
\end{aligned}
$$

转化为最小化两个分布间的KL距离

Generative Adversarial Networks(GAN)

- **Generative Adversarial Network（GAN）的优化函数**

$$V(G,D) = E_{x\sim P_{data}}[\log D(x)] + E_{x\sim P_G}[\log(1-D(x))]$$

$$\min_G \max_D V(G,D)$$

- **交替优化 *G* 和 *D***

$$G^* = \arg\ \min_G\ \max_D\ V(G,D)$$

Generative Adversarial Networks(GAN)

- 固定 G

$$V(G, D) = \int_{\boldsymbol{x}} p_{\text{data}}(\boldsymbol{x}) \log(D(\boldsymbol{x}))dx + \int_{z} p_{\boldsymbol{z}}(\boldsymbol{z}) \log(1 - D(g(\boldsymbol{z})))dz$$
$$= \int_{\boldsymbol{x}} p_{\text{data}}(\boldsymbol{x}) \log(D(\boldsymbol{x})) + p_g(\boldsymbol{x}) \log(1 - D(\boldsymbol{x}))dx$$

- 最优的 D 为

$$D_G^*(\boldsymbol{x}) = \frac{p_{data}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})}$$

Generative Adversarial Networks(GAN)

- **固定** ***D***

$$
\begin{aligned}
C(G) &= \max_D V(G, D) \\
&= \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}}[\log D_G^*(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}}[\log(1 - D_G^*(G(\boldsymbol{z})))] \\
&= \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}}[\log D_G^*(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{x} \sim p_g}[\log(1 - D_G^*(\boldsymbol{x}))] \\
&= \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}}\left[\log \frac{p_{\text{data}}(\boldsymbol{x})}{P_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right] + \mathbb{E}_{\boldsymbol{x} \sim p_g}\left[\log \frac{p_g(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right]
\end{aligned}
$$

- **最优的** ***G*** **为**

$$p_g = p_{data}.$$