Deep Learning

Wenrui Dai（戴文睿）
http://min.sjtu.edu.cn

Department of Computer Science and Engineering

Shanghai Jiao Tong University

December 2nd, 2020

Today's Topics

- **Fundamentals of Deep Learning**
- **Convolutional Neural Networks (CNNs)**
- **Recurrent Neural Networks (RNNs)**
- **Generative Models**

Introduction

Forbes Billionaires Innovation Leadership Money Consumer Industry

Deep Learning's Promise To The Marketing Sector

Science Home News Journals Topics Careers

REPORT

All-optical machine learning using diffractive deep neural networks

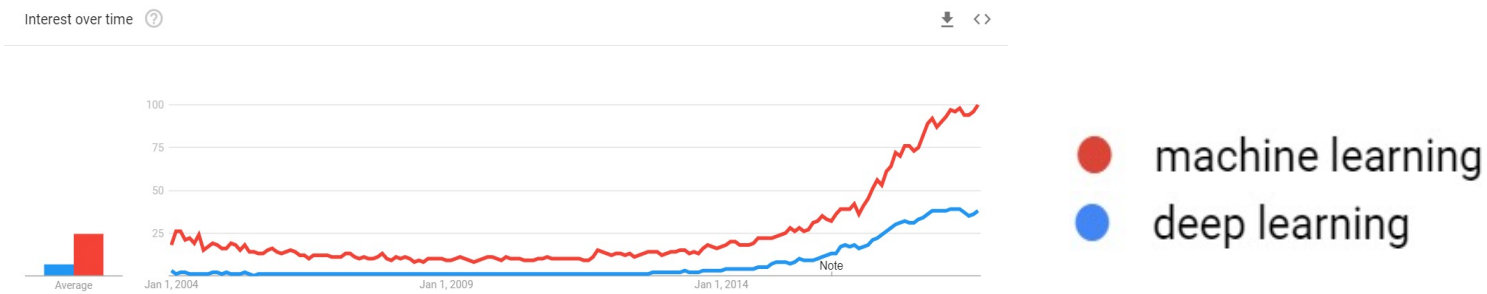
ONS WATCHLIST MarketWatch

PRESS RELEASE

MEDIA ALERT: Deep Learning Acceleration Startup Mipsology to Exhibit at Xilinx Developer Forum

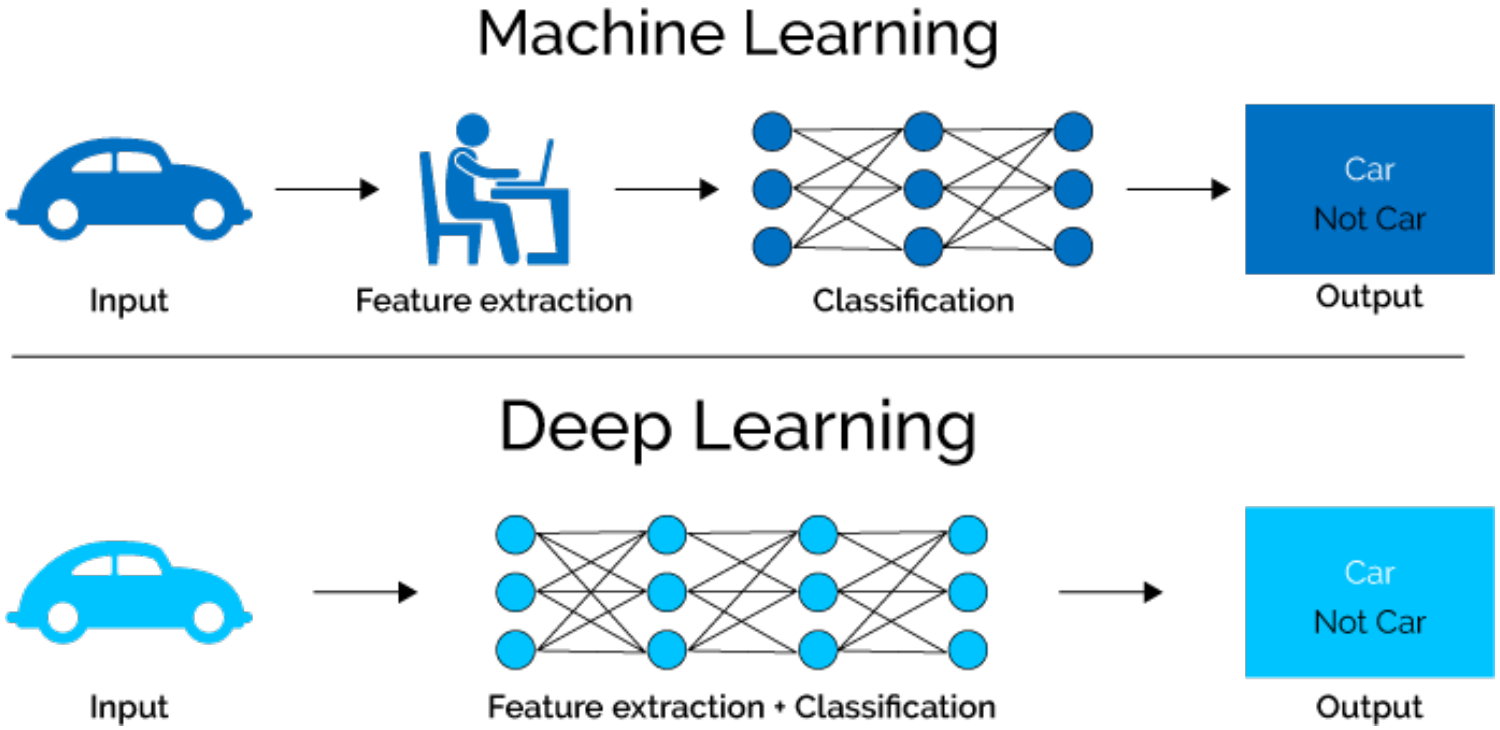
THE LANCET Respiratory Medicine

Deep learning for classifying fibrotic lung disease on high-resolution computed tomography: a case-cohort study



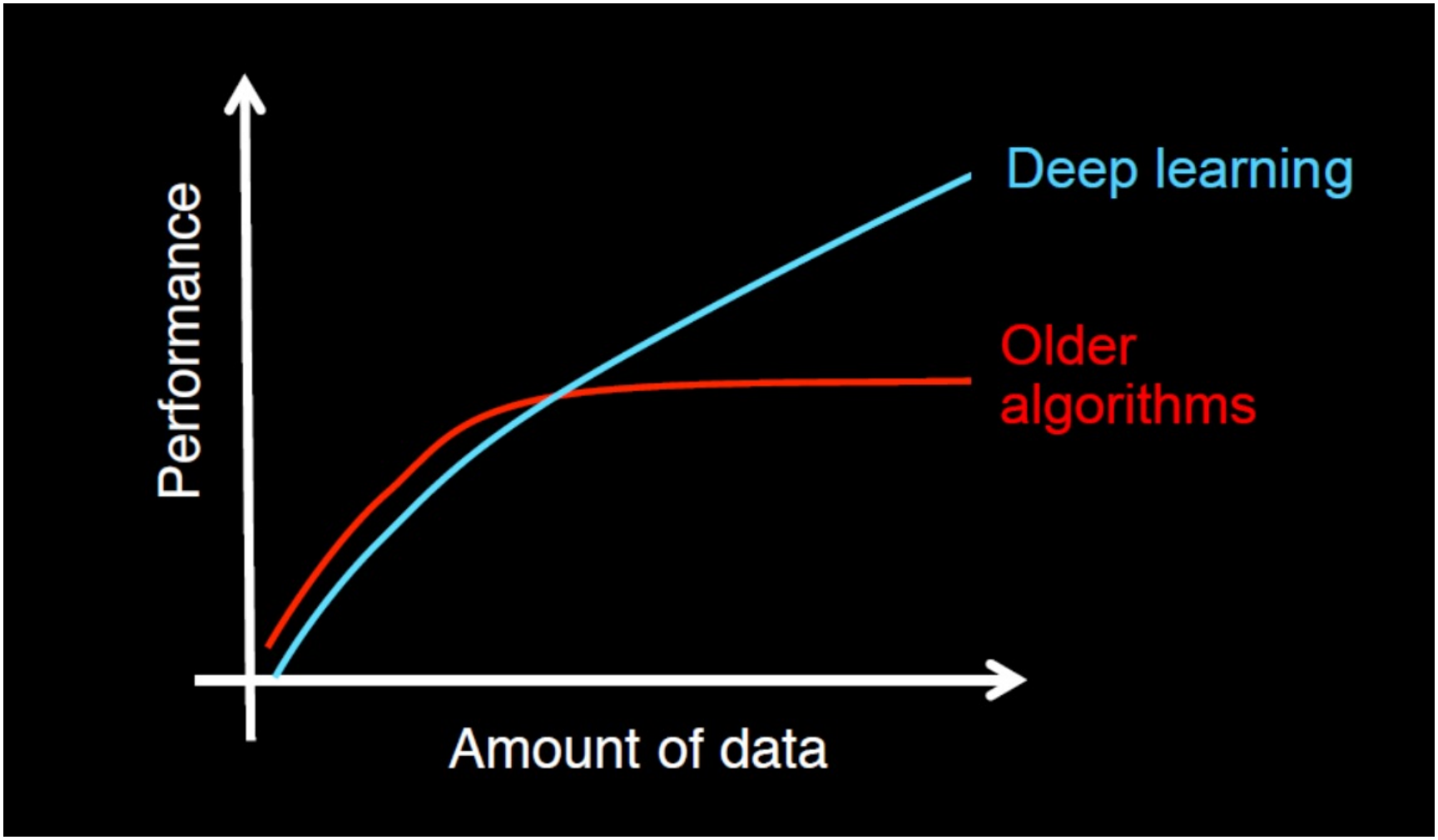
Introduction

- **Deep learning: end-to-end multi-layer neural network for feature generation and processing**



[https://www.xenonstack.com/blog/static/public/uploads/media/machine-learning-vs-deep-learning.png]

Introduction



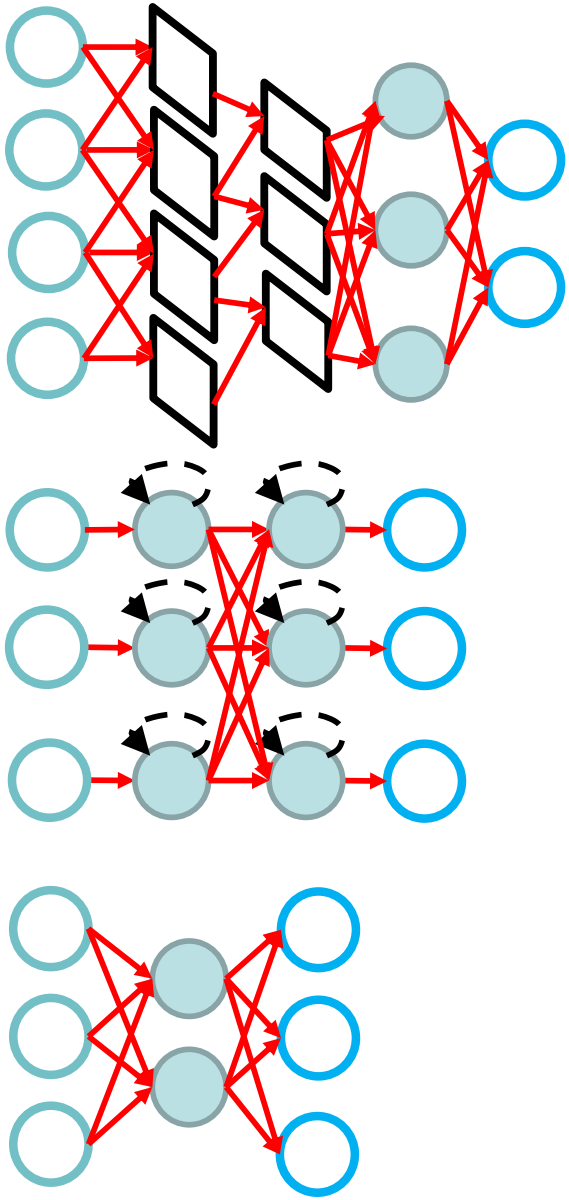
[Andrew Ng, http://cs229.stanford.edu/materials/CS229-DeepLearning.pdf]

Introduction

Need to design architectures!

Introduction

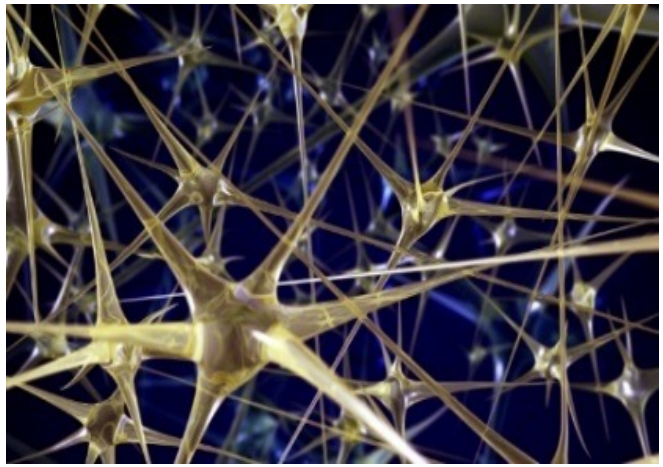
- Deep Learning Architecture
- Supervised
  - Convolutional neural network (CNN)
  - Recurrent neural network (RNN)
  - Multilayer perceptron
- Unsupervised
  - Autoencoder
  - Boltzman machine
- Generative Adversarial Networks



Fundamentals of Deep Learning

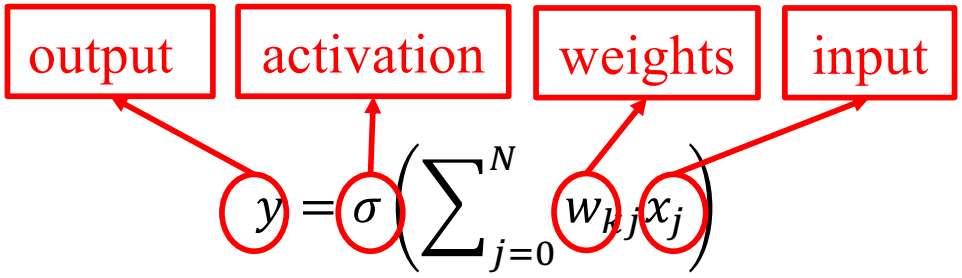
What's Deep Learning?

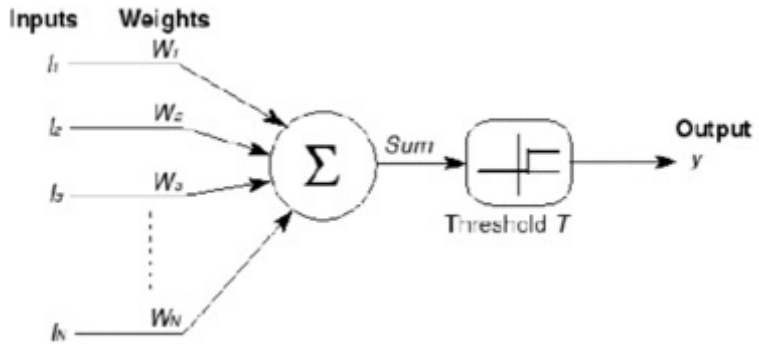
- **Deep learning** is part of a broader family of machine learning methods based on the layers used in **artificial neural networks**.
- **Artificial neural networks (ANNs) are computing systems that are inspired by, but not identical to, biological neural networks that constitute animal brains.**

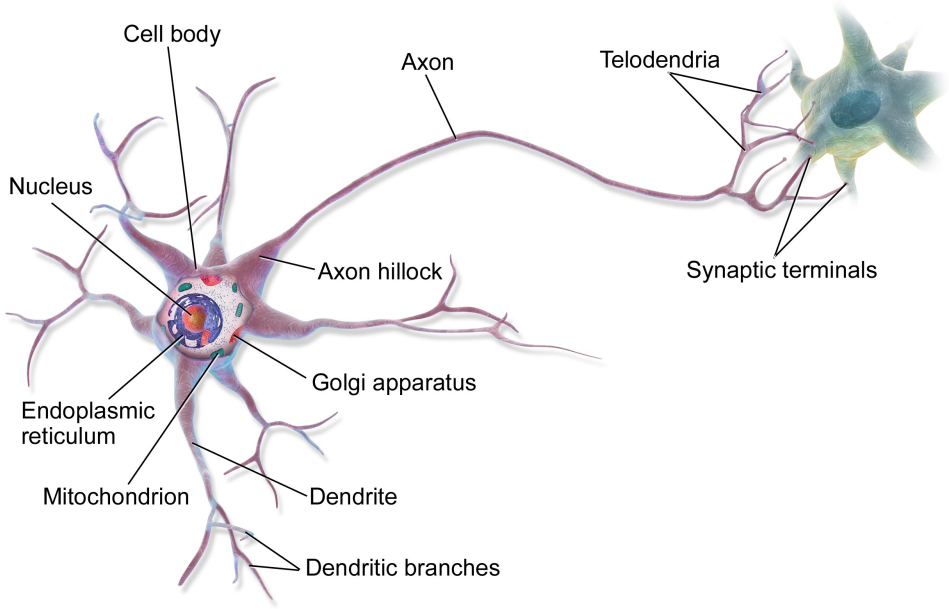


Neuron

- **The MCP (McCulloch-Pitts) neuron**
  - **Frist proposed in 1943 by the neurophysiologist Walter S. McCulloch and the logician Walter Pitts, the MCP neuron is a simple mathematical model of a *biological neuron*.**

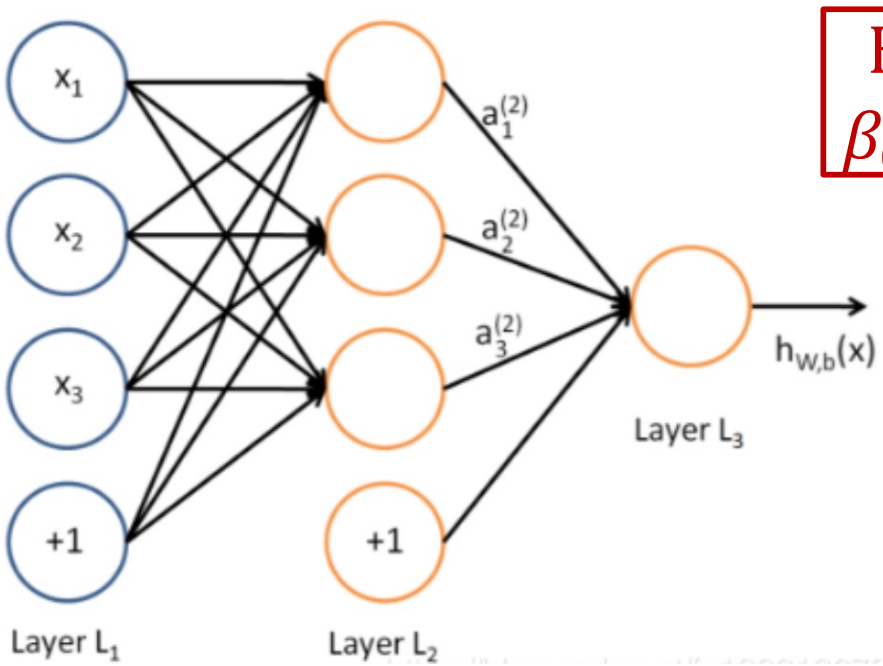






Perceptron

- The perceptron algorithm was invented in 1958 by Rosenblatt used to binary classification.
- Perceptron trained with gradient descent algorithm, and its convergence was guaranteed in 1962.



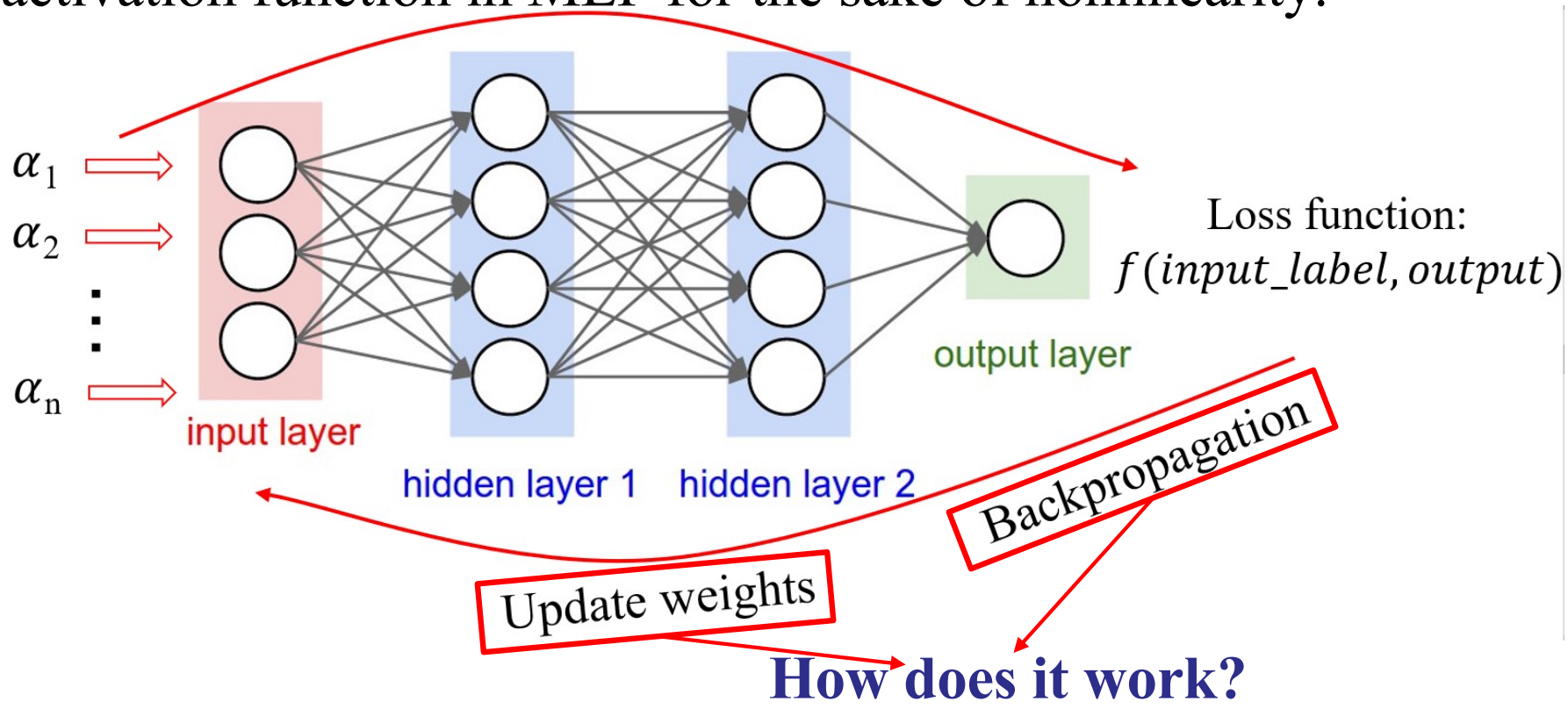
Hyperplane
$\beta_0 + \beta^T x = 0$

Limitation of Percetron

- **Marvin Minsky proved that perceptron is a linear model essentially in 1969, and perceptron cannot handle nonlinear classification.**
- **To solve this problem, it is necessary to increase the number of layers.**
- **Training algorithm for a multi-layer perceptron (MLP)**

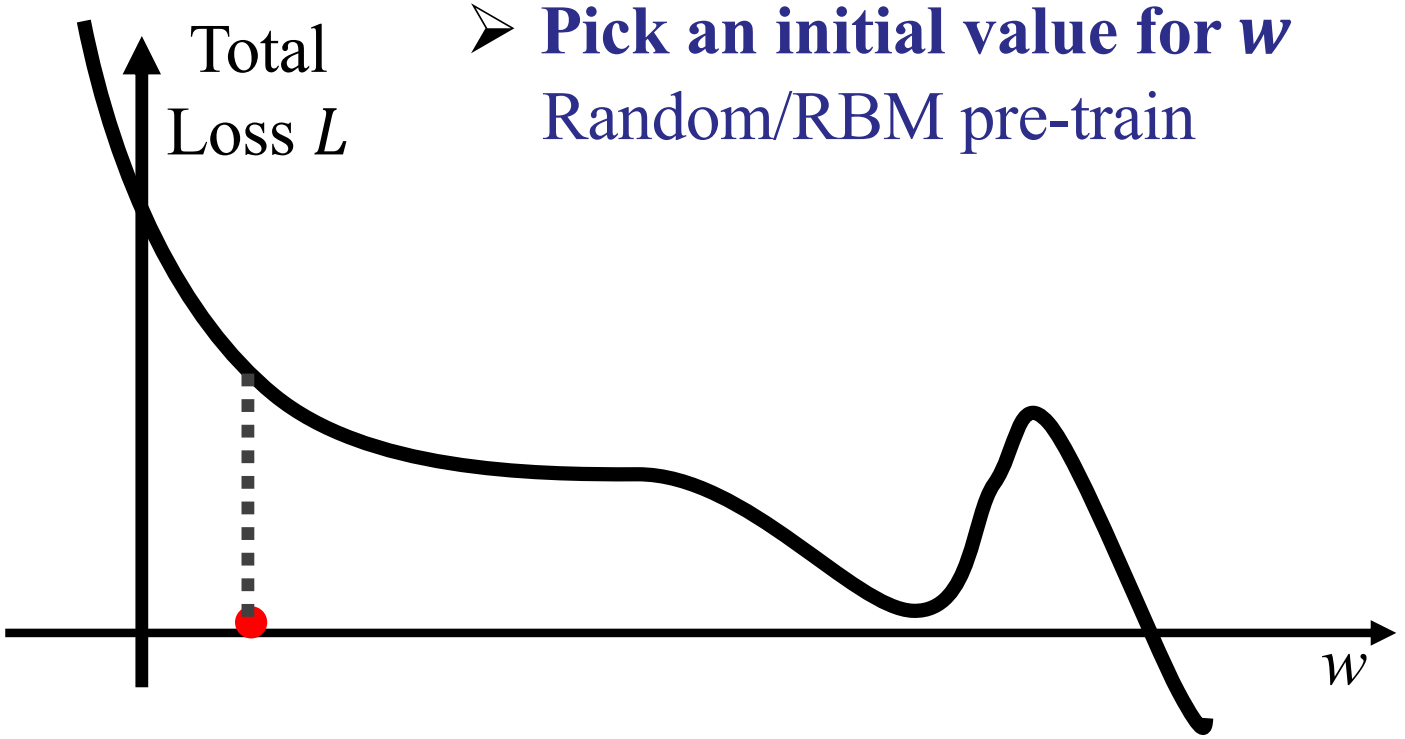
Backpropagation

- Geoffrey Hinton proposed the Backpropagation (BP) algorithm in 1986 for training the MLP.
- The sigmoid function $S(x) = 1/(1 + e^{-x})$ is used as the activation function in MLP for the sake of nonlinearity.



Gradient Descent

Network parameters $\theta = \{w_1, w_2, w_3, \cdots, b_1, b_2, b_3, \cdots\}$

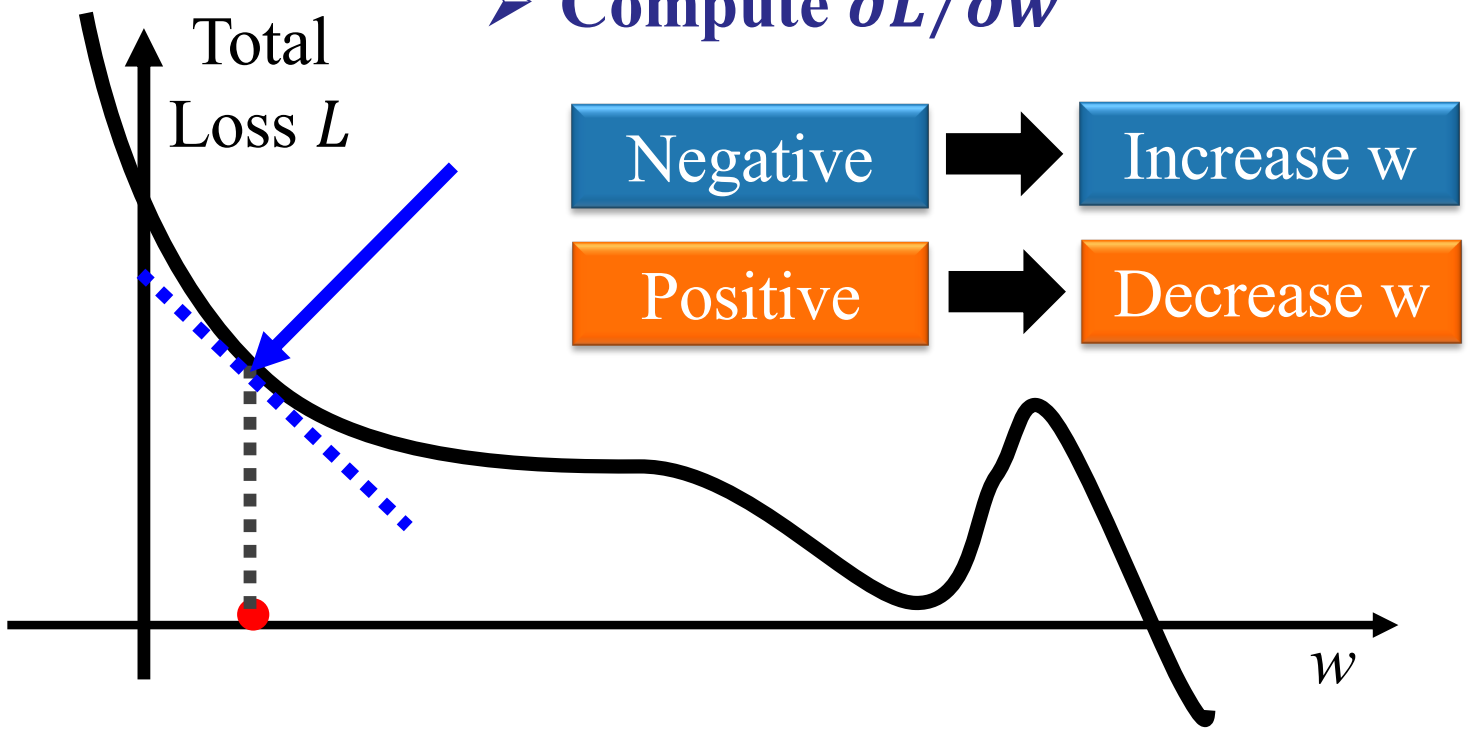


- **Pick an initial value for *w***
  Random/RBM pre-train

Gradient Descent

Network parameters $\theta = \{w_1, w_2, w_3, \cdots, b_1, b_2, b_3, \cdots\}$

- Pick an initial value for w
- **Compute $\partial L / \partial w$**



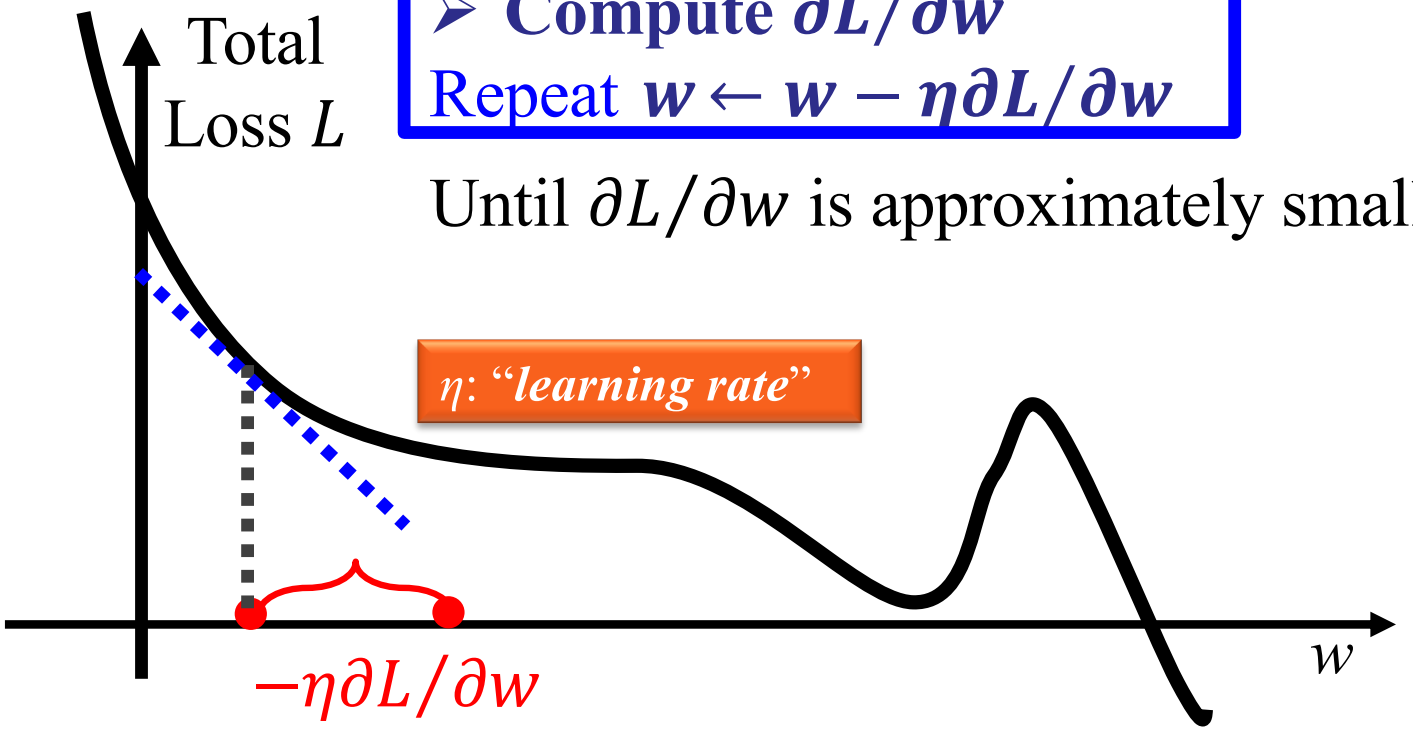
Gradient Descent

Network parameters $\theta = \{w_1, w_2, w_3, \cdots, b_1, b_2, b_3, \cdots\}$

- Pick an initial value for w
- **Compute $\partial L/\partial w$**

Repeat **$w \leftarrow w - \eta \partial L/\partial w$**

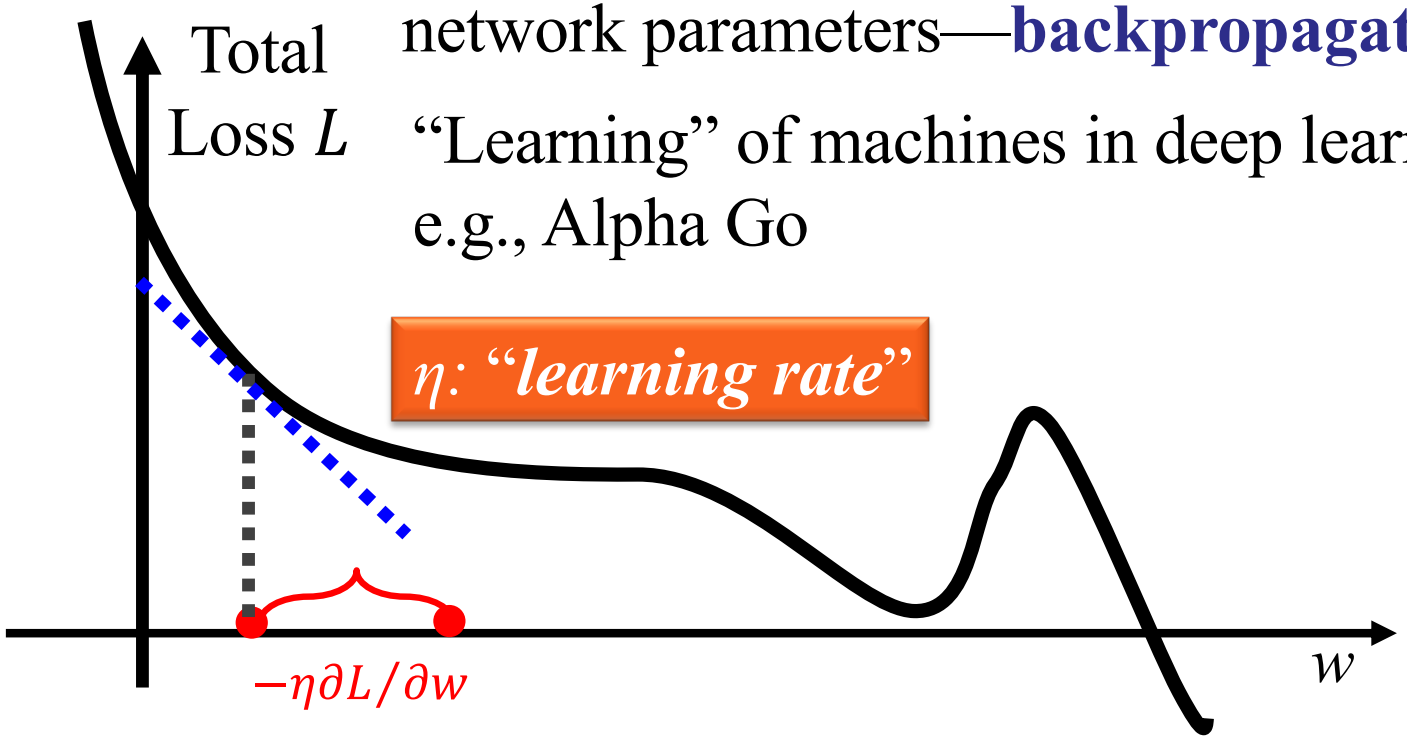
Until $\partial L/\partial w$ is approximately small



Gradient Descent

Network parameters $\theta = \{w_1, w_2, w_3, \cdots, b_1, b_2, b_3, \cdots\}$

An efficient way to compute the $\partial L/\partial w$ of the network parameters—**backpropagation**.



"Learning" of machines in deep learning, e.g., Alpha Go

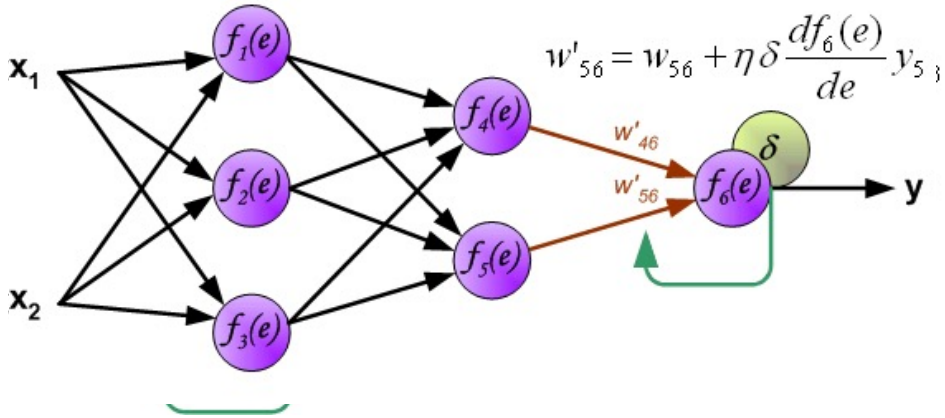
Backpropagation

- Three-layer neural network with two inputs and one output
  - **Forward propagation** through the hidden & output layers
  - Compare the output y with the target z (label).
  - The error δ is **propagated back** to all neurons from output to inputs one after the other.

$$w'_{15} = w_{15} + \eta \delta_5 \frac{df_5(e)}{de} y_1$$

$$w'_{46} = w_{46} + \eta \delta \frac{df_6(e)}{de} y_4$$

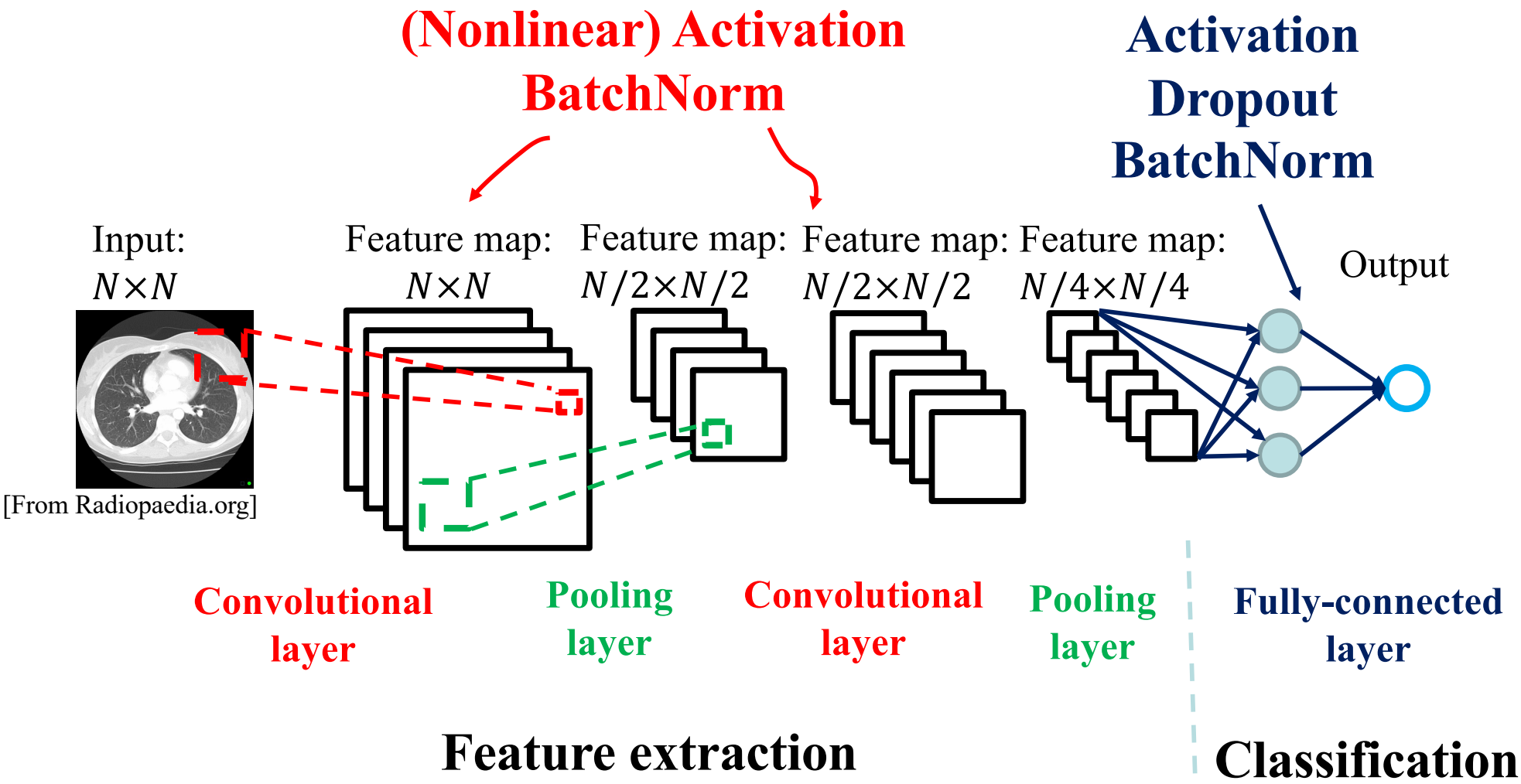
$$w'_{56} = w_{56} + \eta \delta \frac{df_6(e)}{de} y_5$$



Source : http://galaxy.agh.edu.pl

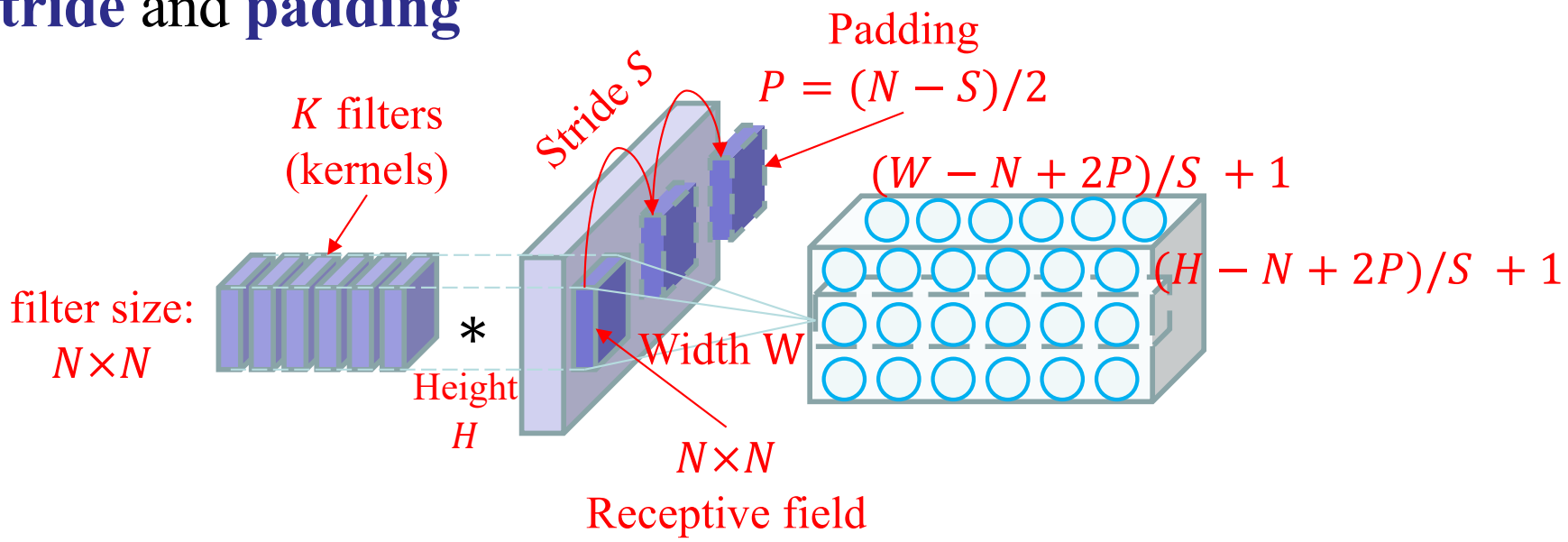
Convolutional Neural Networks (CNNs)

Convolutional Neural Networks



Convolutional Neural Networks

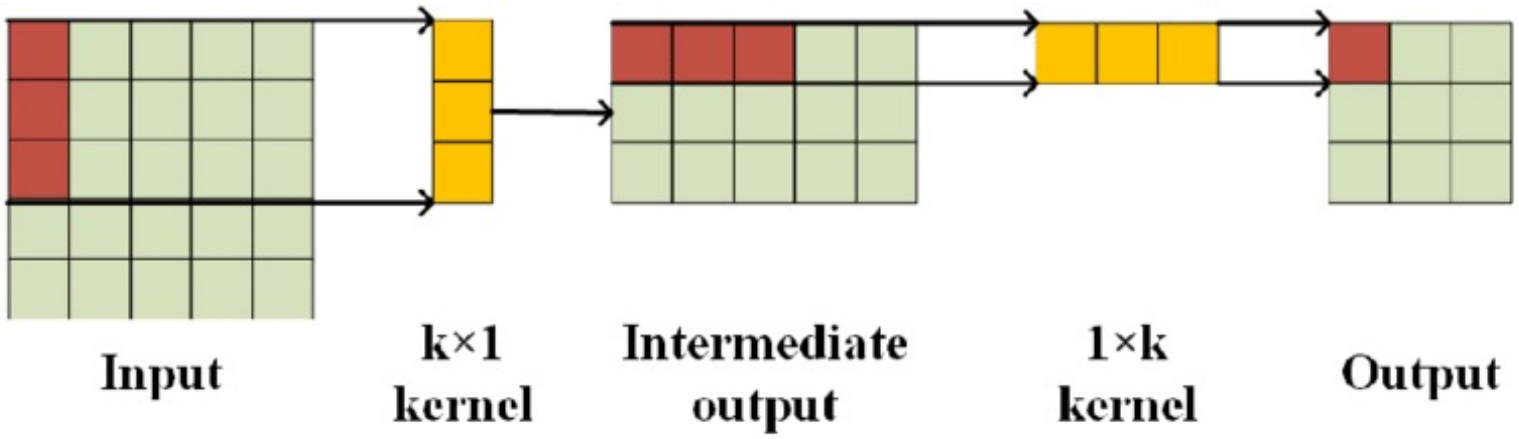
- **Convolutional layer**
  - A set of **filters (kernels)** learned to detect specific types of features at specific spatial location
  - **Receptive field** and **neuron**
  - **Stride** and **padding**



Convolutional Neural Networks

- **Separable Convolutions**
  - **Spatial separable convolutions**

Transform convolutions with $K \times K$ kernels into convolutions with $K \times 1$ and $1 \times K$ kernels



Convolutional Neural Networks

- **Separable Convolutions**
  - **Depthwise separable convolutions**

Realize convolutions with $N \times N$ kernels for CI input channels and CO output channels with depthwise convolutions with CI $N \times N \times 1$ kernels and pointwise convolution with CO $1 \times 1 \times CI$ kernels

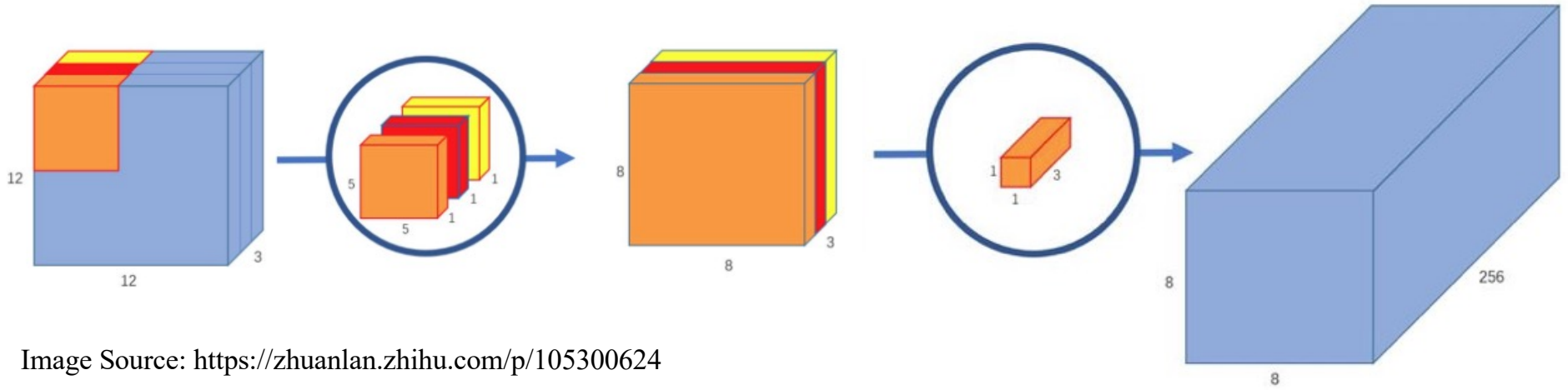
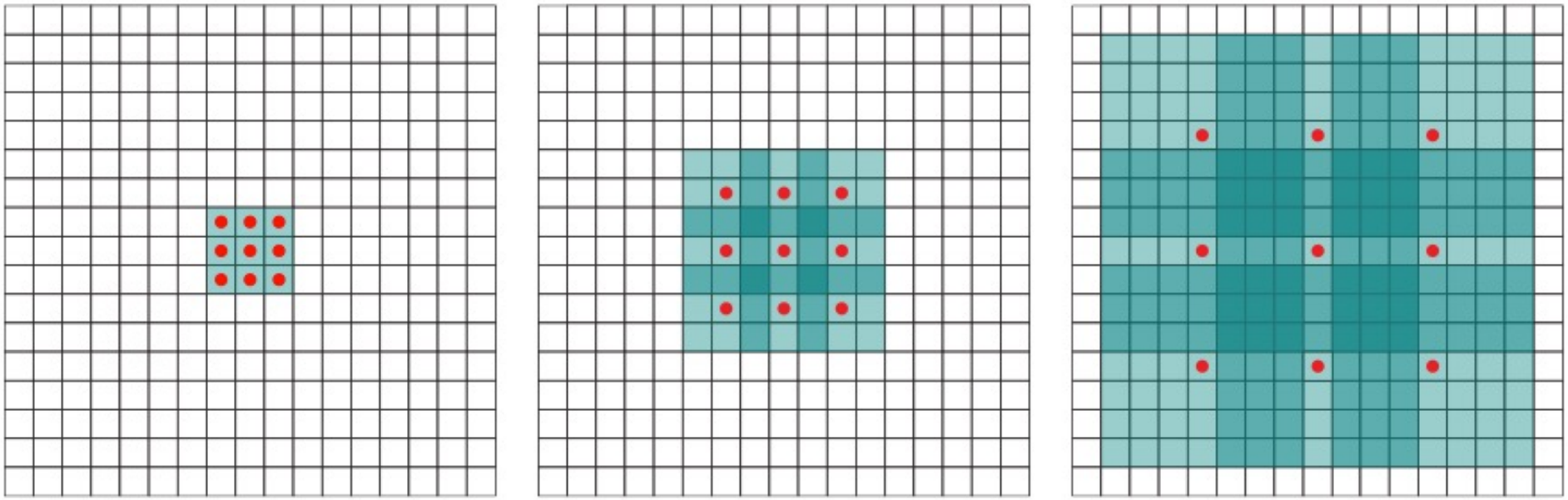


Image Source: https://zhuanlan.zhihu.com/p/105300624

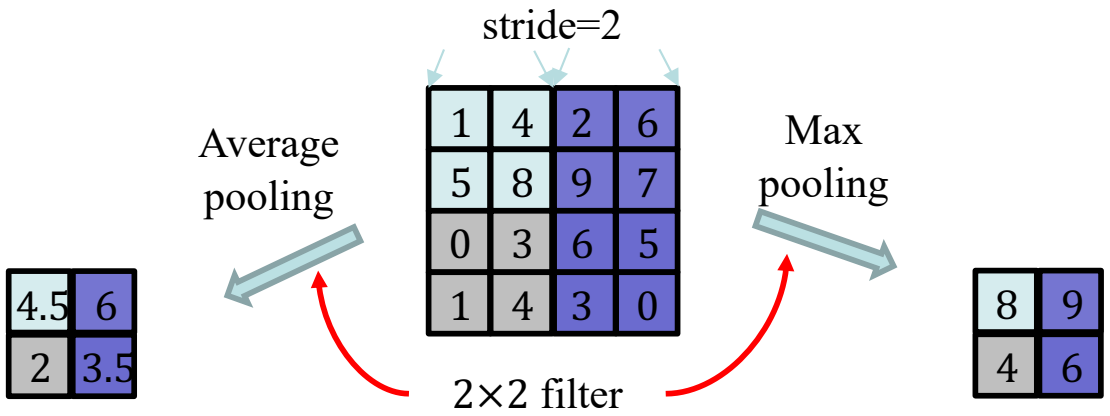
Convolutional Neural Networks

- **Dilated Convolutions**
  - **Exponentially expand receptive fields without losing resolution or coverage**
  - **Maintain the size of feature maps without excessive computations**



Convolutional Neural Networks

- **Pooling layer**
  - **Nonlinear downsampling** to reduce spatial size and number of parameters
  - Suppress overfitting
  - Common routines: **max pooling** & **average pooling**



Convolutional Neural Network

- **Activation layer**
  - Introduce nonlinearities to linear convolutional layers
  - Nonlinear functions: tanh, sigmoid, ReLU (rectified linear units)

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

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

$$\mathrm{relu}(x) = \max(x, 0)$$

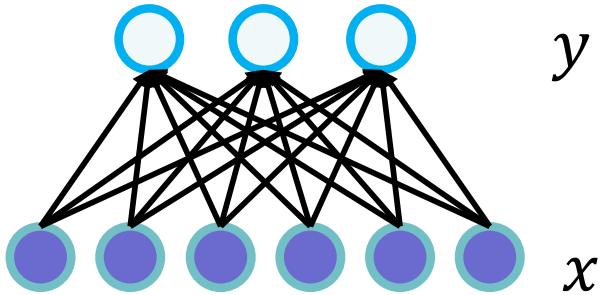
  - ReLU allows faster training without significant difference in accuracy

Convolutional Neural Network

- **Fully (densely) connected layer**
  - Each unit is connected to all units of the previous layers

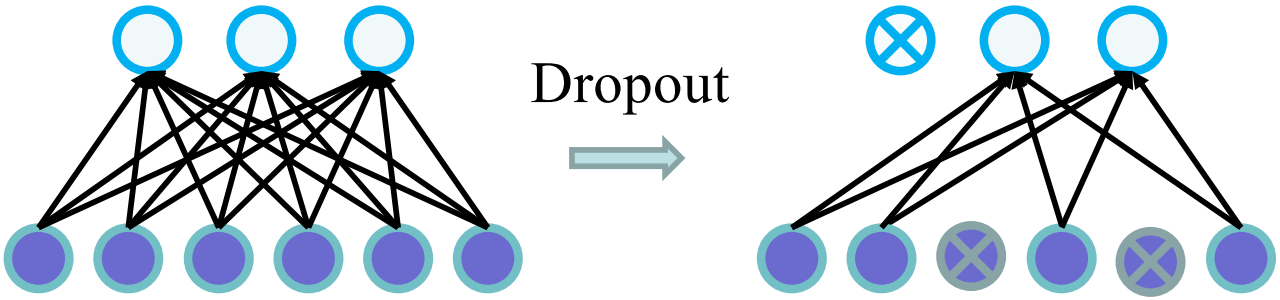
$$y = \sigma(wx + b)$$

  - Learn nonlinear combinations of features for classification
  - Activation function to introduce nonlinearity



Convolutional Neural Network

- **Dropout:** Regularization technique to reduce overfitting
  - An efficient way to approximately combine different neural network architectures
  - Randomly drop a portion of nodes and their connections with a probability (dropout rate) for gradient computation



Convolutional Neural Network

- **Batch Normalization**
  - Allow higher learning rates and robust to initialization (suppress explosive or vanishing gradients and early stop in local minimum)
  - Regularize the deep learning model
  - Normalize by scaling and shifting activations
  - For minibatch $B = \{x_1, \cdots, x_m\}$,

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i\,, \qquad \sigma_B^2 = \frac{1}{m}\sum_{i}^{m}(x_i - \mu_B)^2$$

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}}, \qquad y_i = \gamma\hat{x}_i + \beta$$

Convolutional Neural Network

- **Layer-by-layer training with backpropagation**
  - **For convolutional layer l, forward process:**

$$x_{i,j}^{l} = \sum_{u=0}^{n-1}\sum_{v=0}^{n-1} w_{u,v}^{l} h_{i+u,j+v}^{l-1} + b_{i,j}^{l}, \qquad h_{i,j}^{l} = f\left(x_{i,j}^{l}\right)$$

weights

activation function

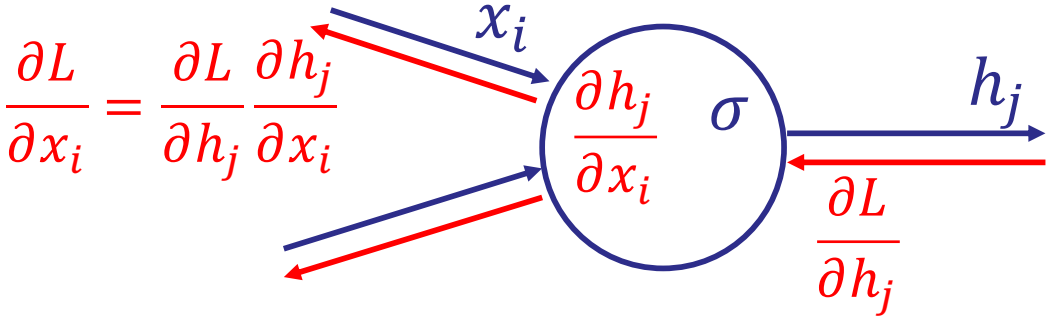
  - **Backward process:**

$$\frac{\partial L}{\partial w_{i,j}^{l}} = \sum_{u=0}^{H-n}\sum_{v=0}^{W-n} \delta_{u,v}^{l} h_{i+u,j+v}^{l-1}$$

$$\frac{\partial L}{\partial x_{i,j}^{l}} = \sum_{u=0}^{n-1}\sum_{v=0}^{n-1} \delta_{i-u,j-v}^{l+1} w_{u,v}^{l+1} \frac{\partial h_{i,j}^{l}}{\partial x_{i,j}^{l}}$$

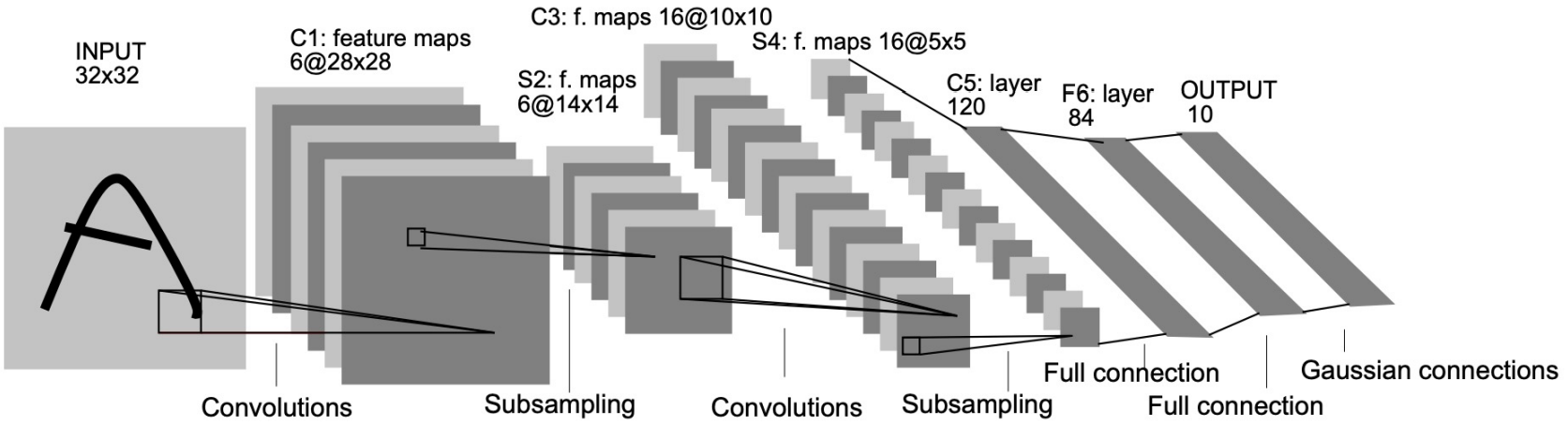
Loss function

$\partial L / \partial x_{i,j}^{l}$



Examples of CNN

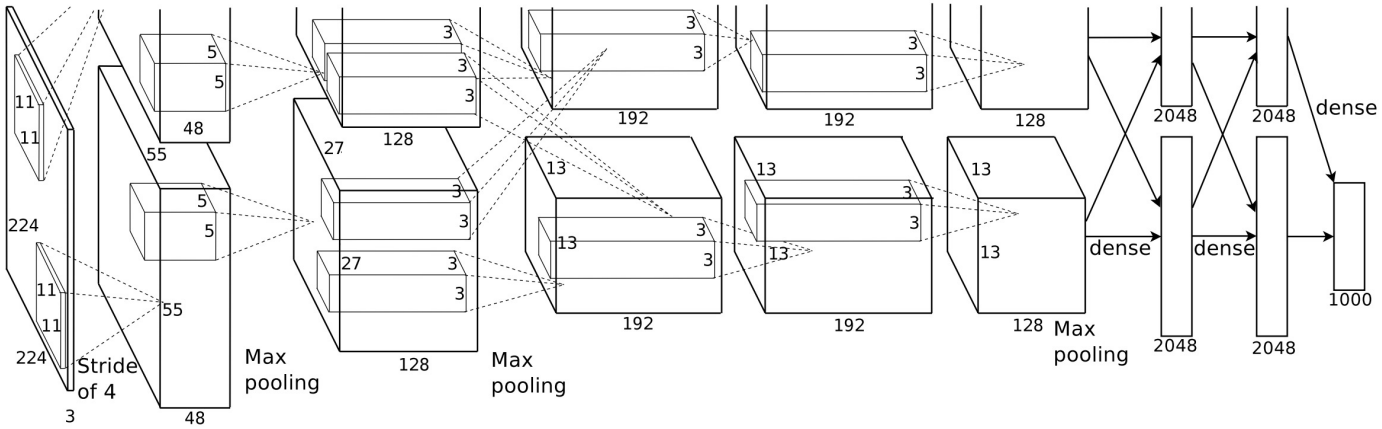
- **LeNet: The pioneer of CNN**
- **Insights:**
  - **Local Receptive Fields**
  - **Shared Weights**
  - **Sub-Sampling**



LeCun, Yann, et al. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE* 86.11 (1998): 2278-2324.

Examples of CNN

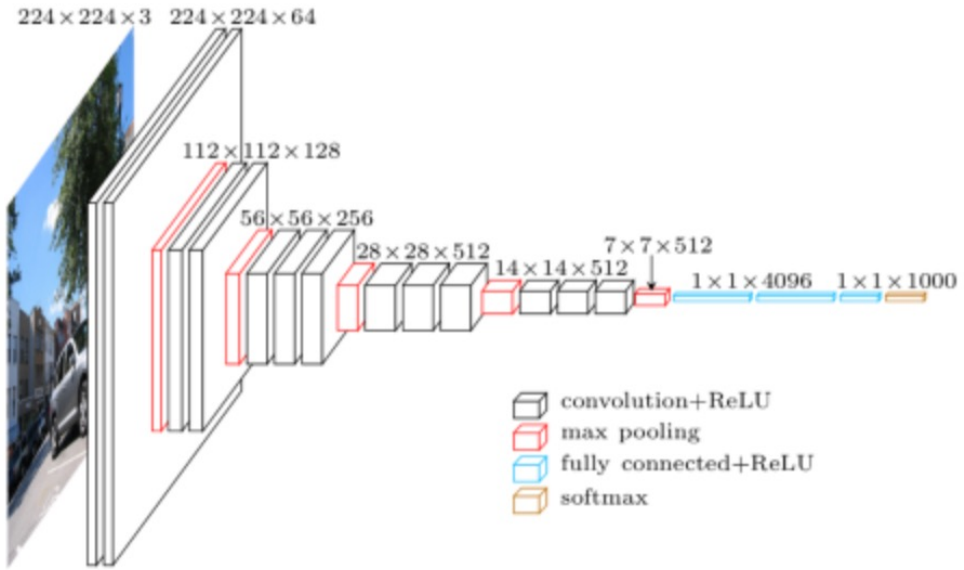
- **AlexNet: The Revealer of New Era**
- **Insights:**
  - **ReLU**
  - **Dropout**
  - **Overlapping Pooling**
  - **Groupwise Convolution**
  - **Local Response Normalization**



Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.

Examples of CNN

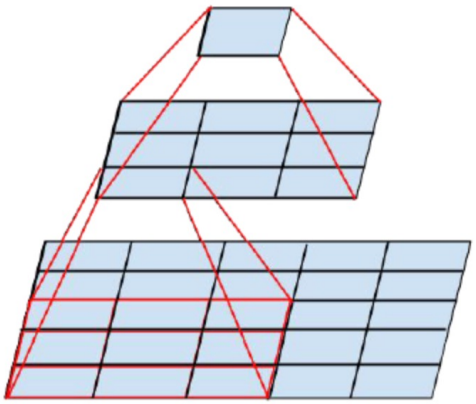
- **VGG: Very Deep CNN**
- **Insights:**
  - **Substitute large convolution kernels with Multi-layer small convolution kernels**
  - **Replace FC with convolution layer in evaluation**
  - **Expand channels**
  - **Deeper and Wider**

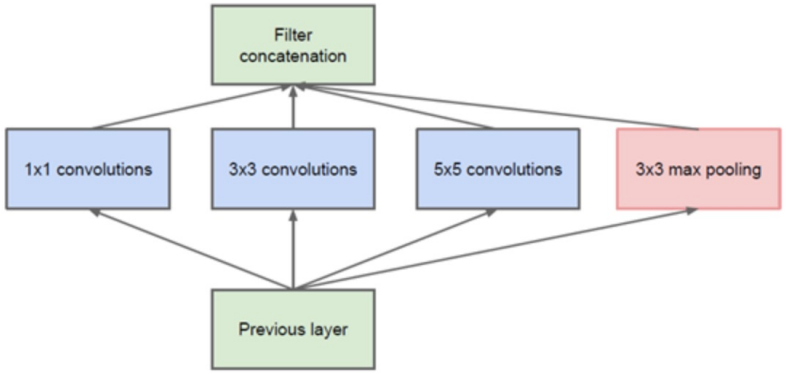


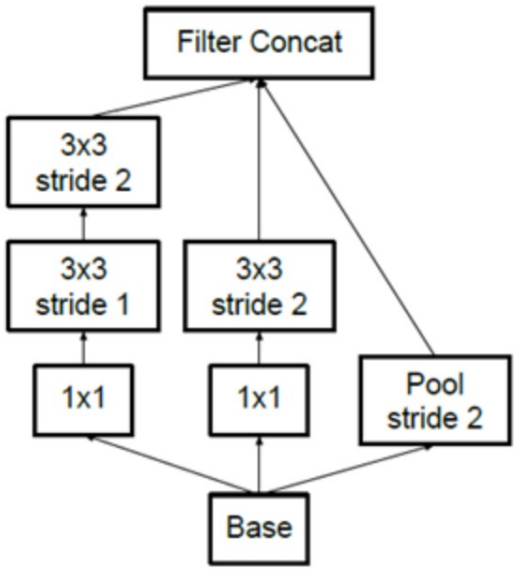
Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition[J]. arXiv preprint arXiv:1409.1556, 2014.

Examples of CNN

- **InceptionNet: The Champion of ILSVRC14**
- **Insights:**
  - **Multi-scale kernel size**
  - **Factorizing Convolutions**
  - **Downsampling in parallel**





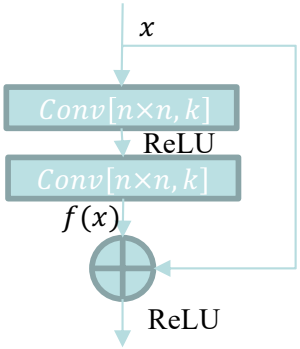


Szegedy C, Liu W, Jia Y, et al. Going deeper with convolutions[C]//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2015: 1-9.

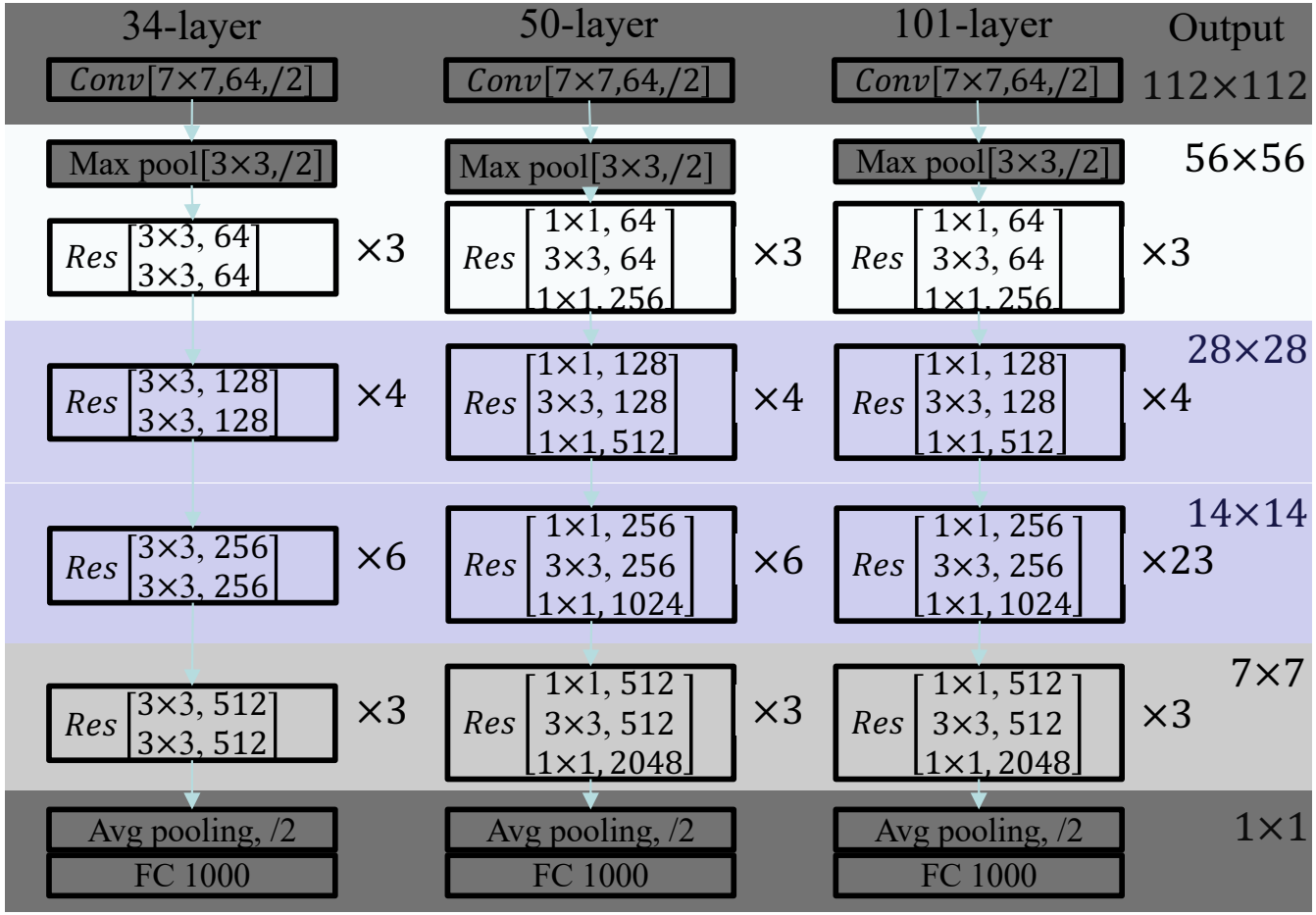
Examples of CNN

- **ResNet**
- **Insights:**
  - Train residual functions
  - Shortcut connection

$$y = f(x) + x$$
$$f = w_2\sigma(w_1 x)$$



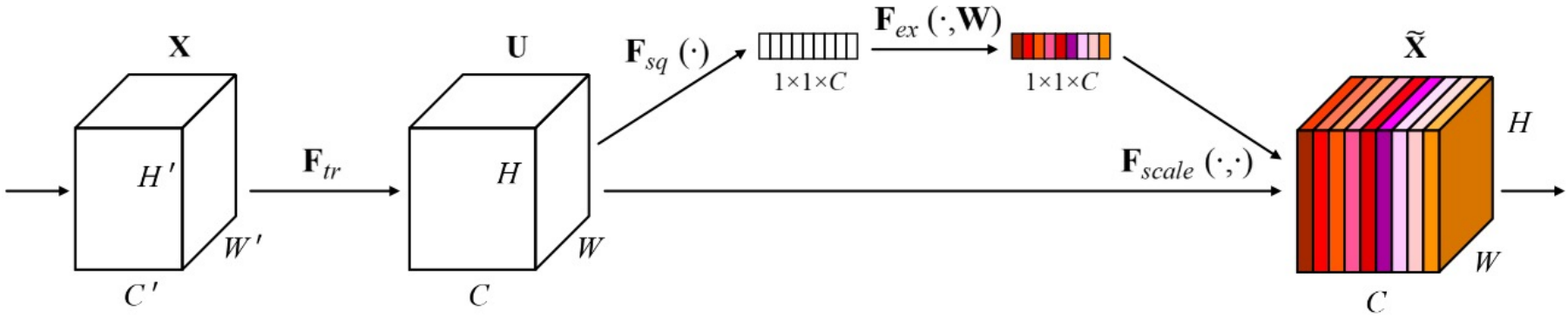
$$Res\begin{bmatrix} n\times n,\ k \\ n\times n,\ k \end{bmatrix}$$



K. He, X. Zhang, S. Ren, J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770-778.

Examples of CNN

- **SENet: The Champion of ILSVRC17**
- **Insights:**
  - Applying channel attention mechanism to CNN.
  - Squeeze & Excitation



Hu J, Shen L, Sun G. Squeeze-and-excitation networks[C]//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2018: 7132-7141.

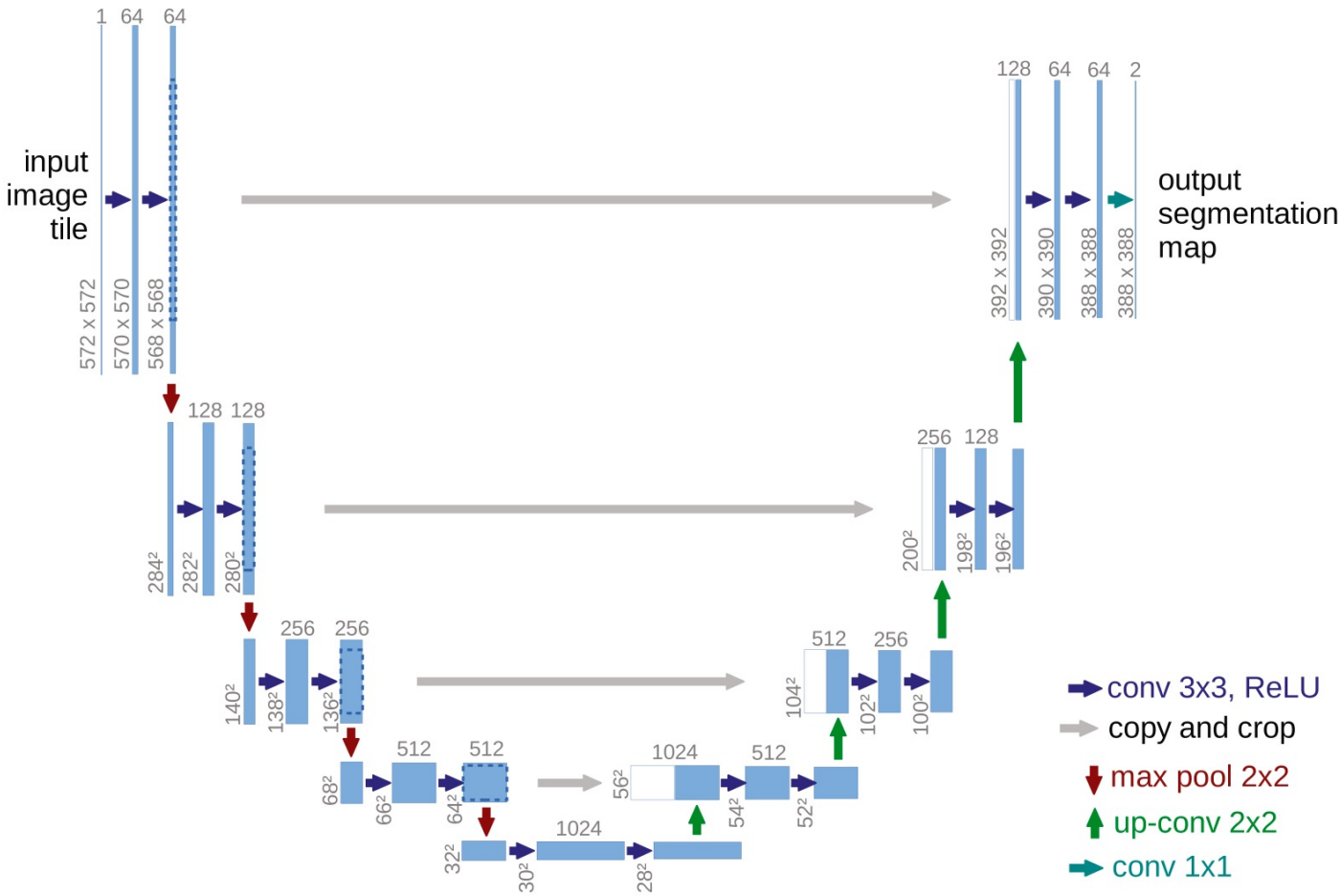
Examples of CNN

- **U-Net: The milestone in medical image segmentation**

CNN

FCN

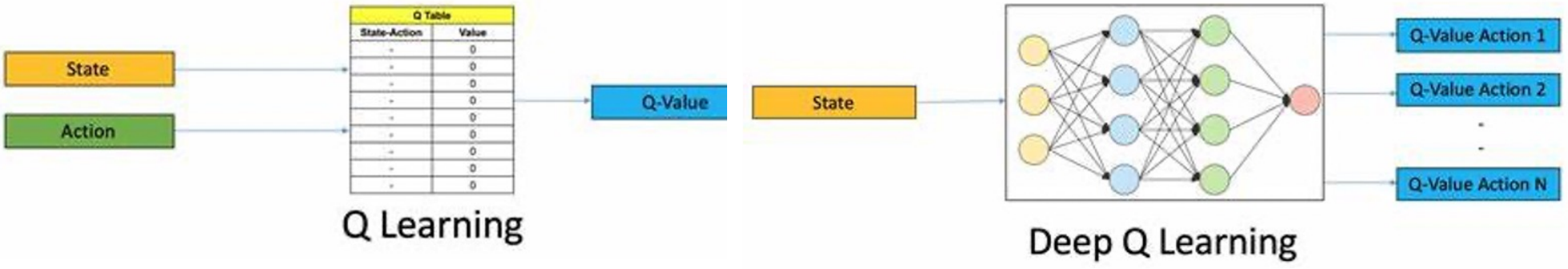
U-Net



Example: Deep Q-Network

- Neural network to approximate the Q-value function.
  - Input: the state
  - Output: the Q-value of all possible actions
- Maximum output of the Q-network as the next action
- Bellman equation for Q-value update

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$



[https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/]

Recurrent Neural Networks (RNNs)

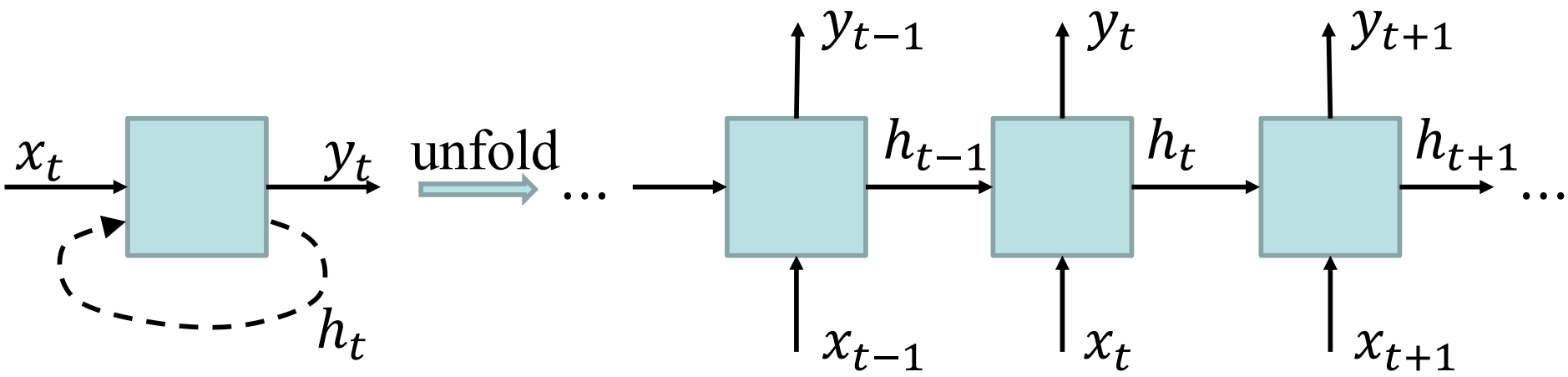
Recurrent Neural Networks

- Recurrent networks introduce cycles and a notion of time

$$h_t = \sigma(x_t w_{xh} + h_{t-1} w_{hh} + b_h)$$

$$y_t = \sigma(h_t w_{hy} + b_y)$$

- **Map a sequence of inputs** $x_1, \cdots, x_N$ to sequence of outputs $y_1, \cdots, y_n$
- Direct acyclic graph (DAG) for back propagation

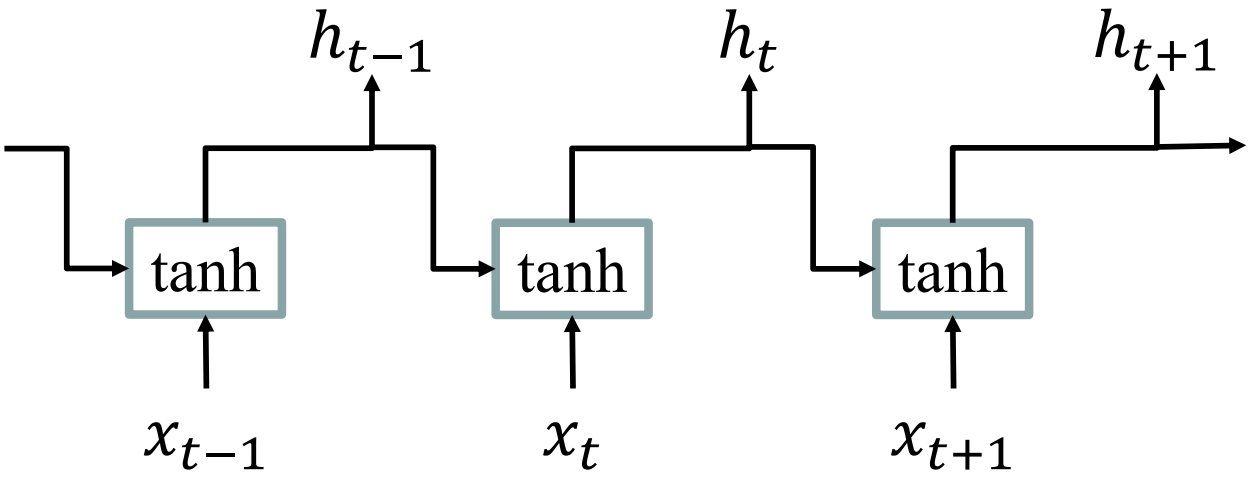


Recurrent Neural Networks

- **Standard RNNs**

$$h_t = \tanh(x_t w_{xh} + h_{t-1} w_{hh} + b_h)$$

$$\hat{y}_t = \text{softmax}(h_t w_{hy} + b_y)$$



Recurrent Neural Networks

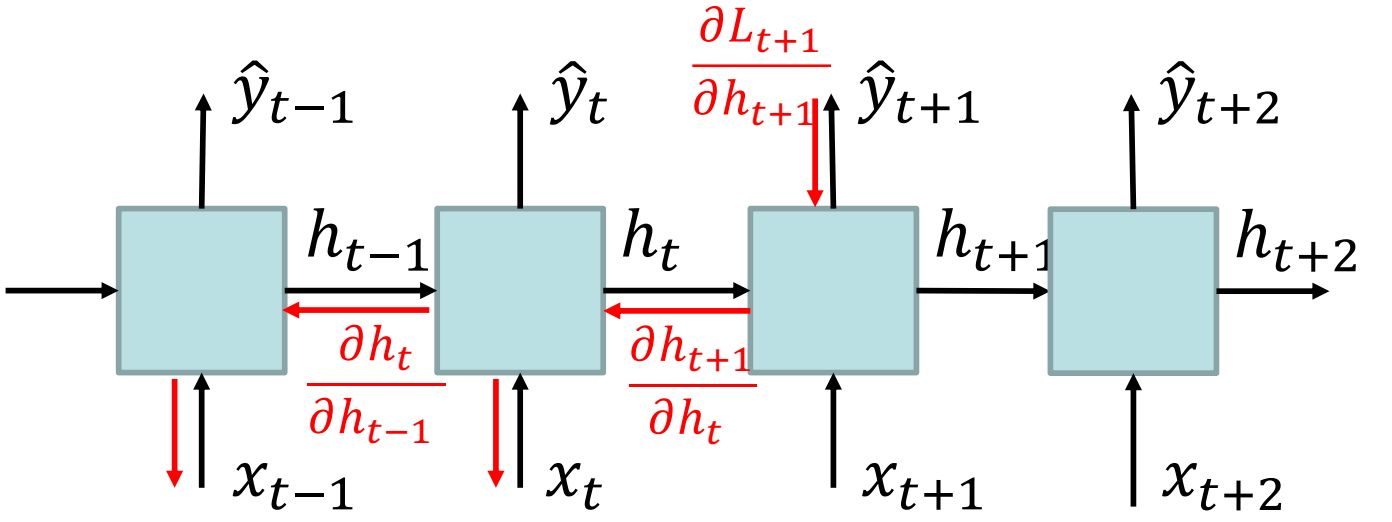
- $L(y, \hat{y}) = -\sum_t L_t(y_t, \hat{y}_t)$

Label

- $\frac{\partial L_t}{\partial w} = \frac{\partial L_t}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial h_t}\frac{\partial h_t}{\partial w} = \sum_{i=1}^{t}\frac{\partial L_t}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial h_t}\frac{\partial h_t}{\partial h_i}\frac{\partial h_i}{\partial w}$

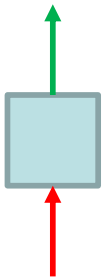
Depends on h_{t-1} and w

Loss function for t

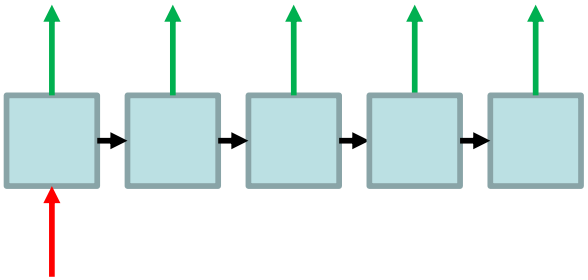


Recurrent Neural Networks

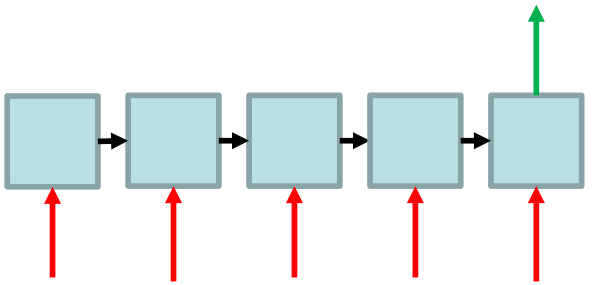
- **Flexible architecture**



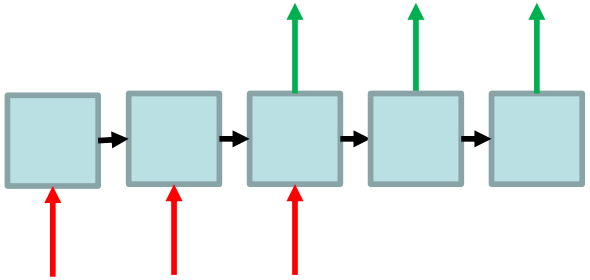
One to one



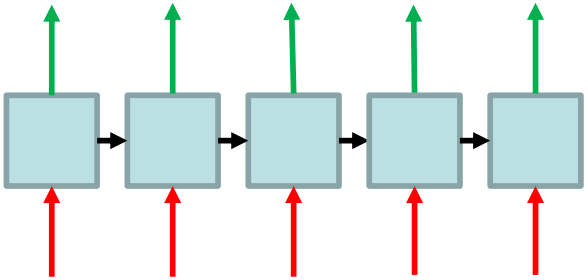
One to many



Many to one



Many to many



Many to many

Recurrent Neural Networks

- **Bidirectional RNN: Information from past and future states**
- **Forward pass:**

  Forward states: $h_{t-1} \rightarrow h_t \rightarrow h_{t+1}$

  Backward states: $h'_{t+1} \rightarrow h'_t \rightarrow h'_{t-1}$

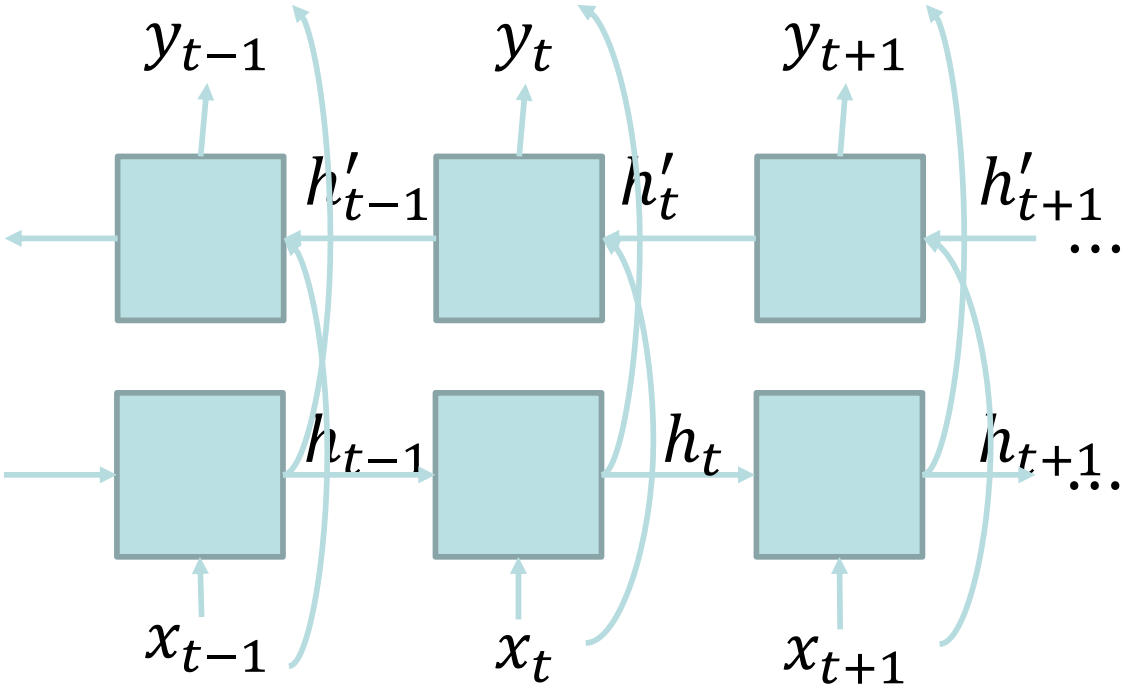
- **Backward pass:**

  Forward states:

  $h_{t+1} \rightarrow h_t \rightarrow h_{t-1}$

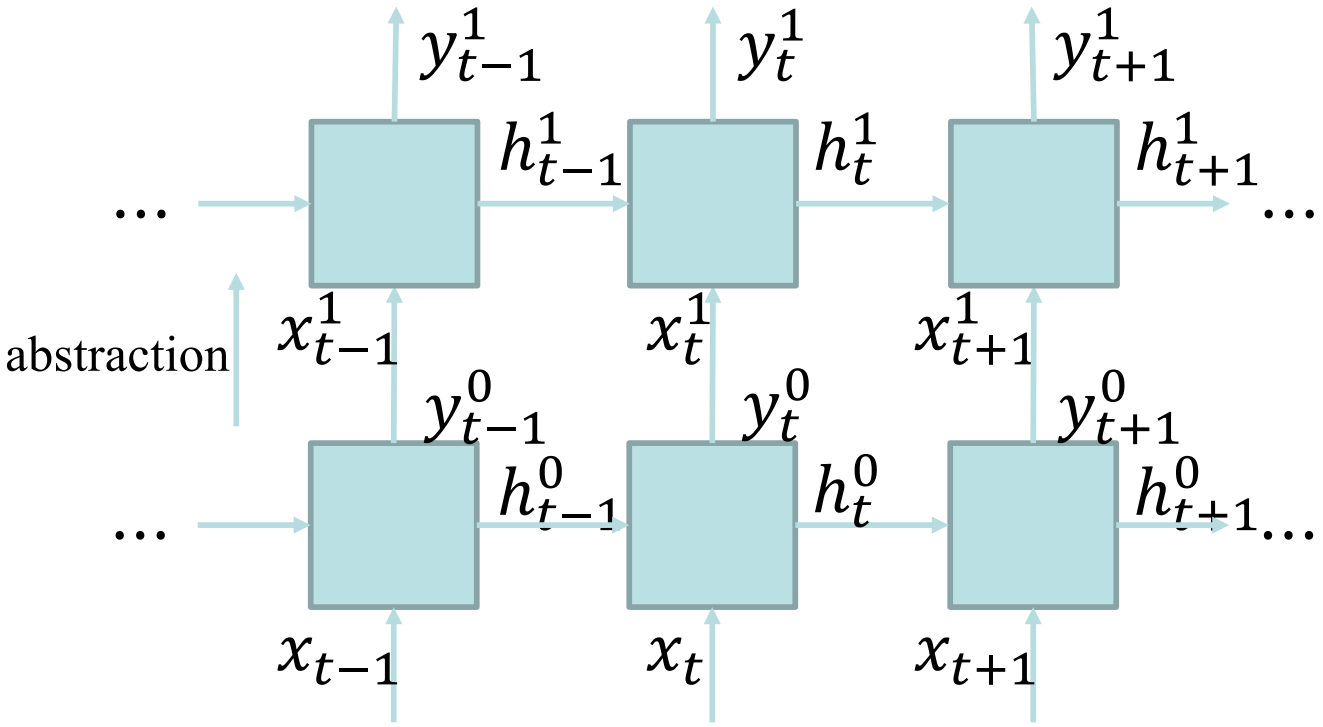
  Backward states:

  $h'_{t-1} \rightarrow h'_t \rightarrow h'_{t+1}$



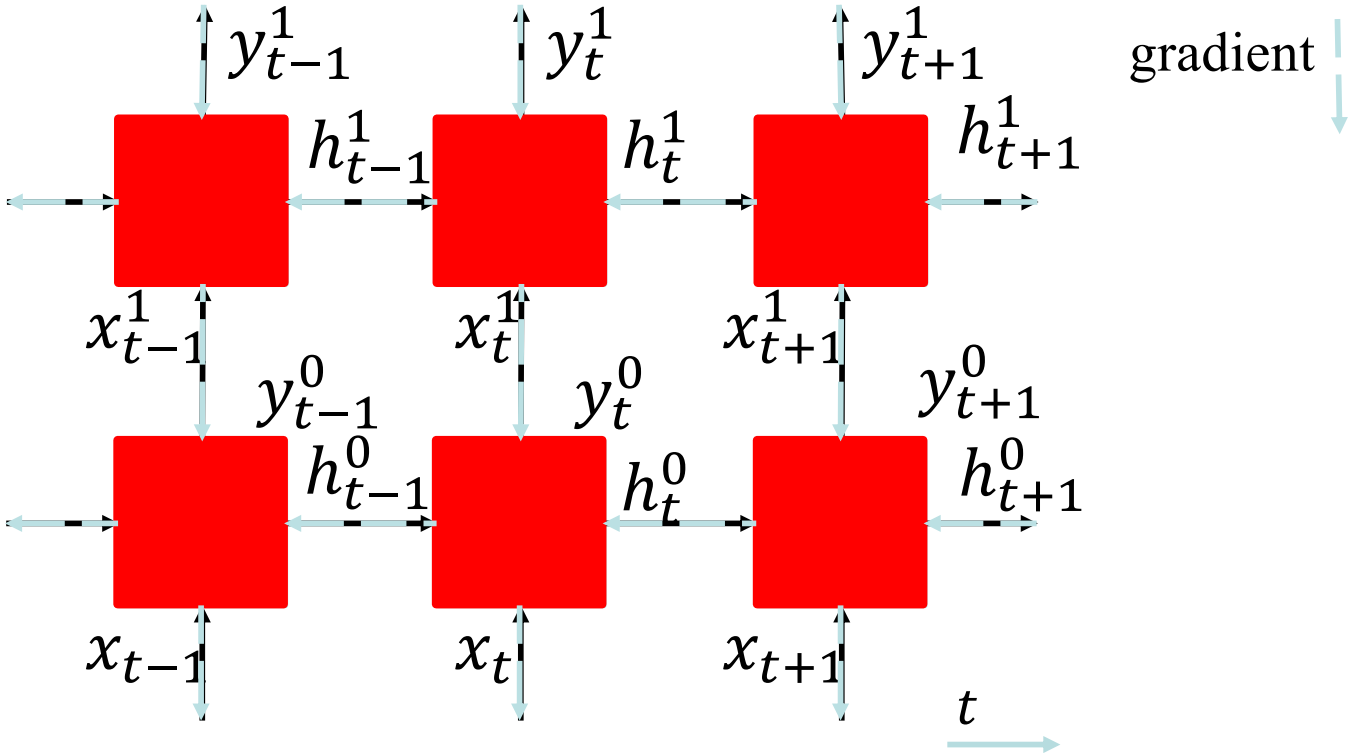
Recurrent Neural Networks

- **Deep RNN:** Stack hidden layers
- Higher layers for higher-layer features



Recurrent Neural Networks

- **Backpropagation for optimization**
  - Forward: Along t layer by layer
  - Backward: Update with gradient descent



Long Short-Term Memory (LSTM)

- **Input:** h_{t-1}, c_{t-1} and x_t
- **Forget gate:** $f_t = \sigma\left(x_t w_{xf} + h_{t-1} w_{hf} + b_f\right)$
- **Input gate:**

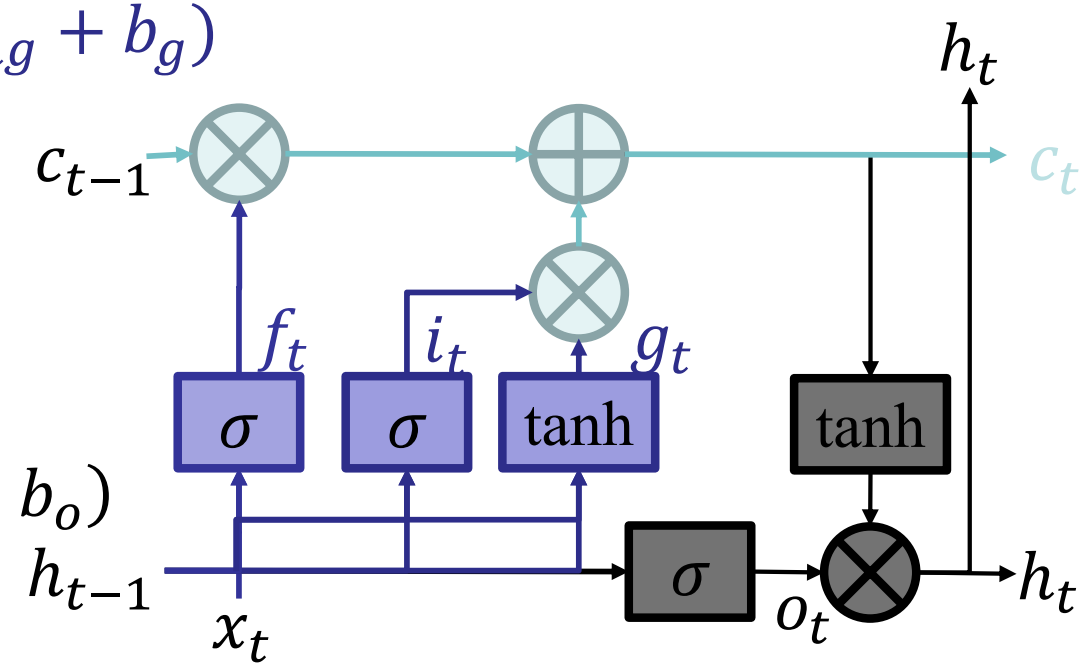
  $i_t = \sigma(x_t w_{xi} + h_{t-1} w_{hi} + b_i)$

  $g_t = \tanh(x_t w_{xg} + h_{t-1} w_{hg} + b_g)$
- **Current state:**

  $c_t = f_t \odot c_{t-1} + i_t \odot g_t$
- **Output:**

  $o_t = \sigma(x_t w_{xo} + h_{t-1} w_{ho} + b_o)$

  $h_t = o_t \odot \tanh(c_t)$



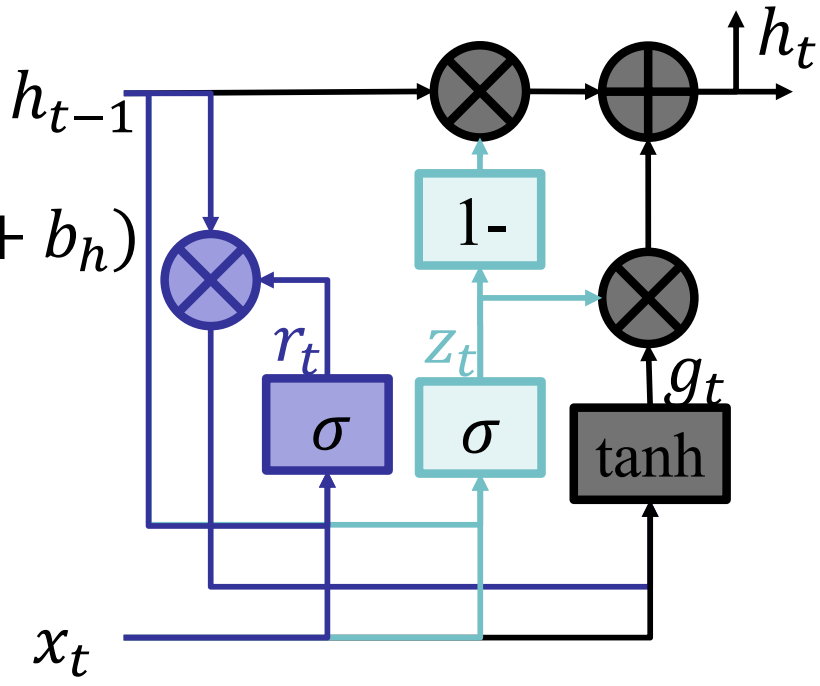
Gated Recurrent Unit (GRU)

- Input: h_{t-1} and x_t
- **Reset gate:** $r_t = \sigma(x_t w_{xr} + h_{t-1} w_{hr} + b_r)$
- Update gate:

  $z_t = \sigma(x_t w_{xz} + h_{t-1} w_{hz} + b_z)$
- Output:

  $g_t = \tanh(x_t w_{xh} + (r_t \odot h_{t-1}) w_{hh} + b_h)$

  $h_t = z_t \odot h_{t-1} + (1 - z_t) \odot g_t$



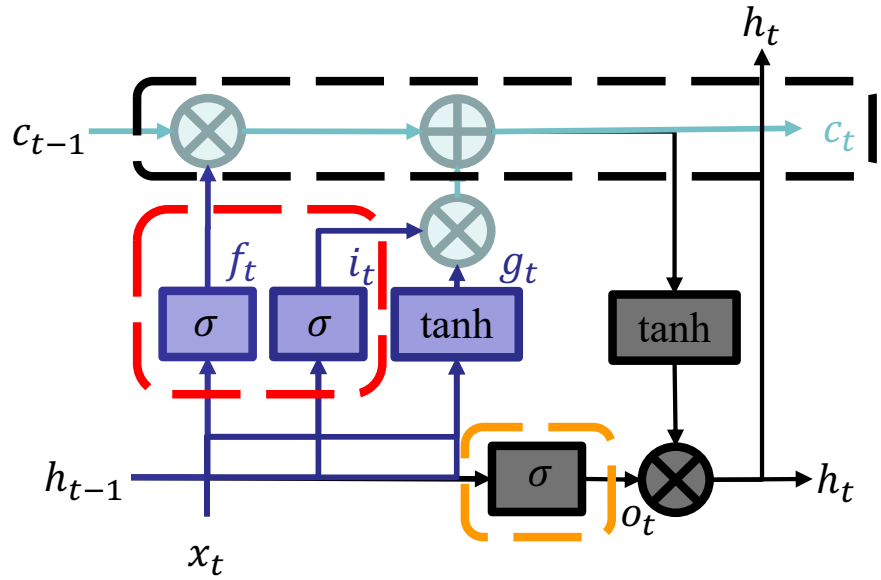
LSTM vs. GRU

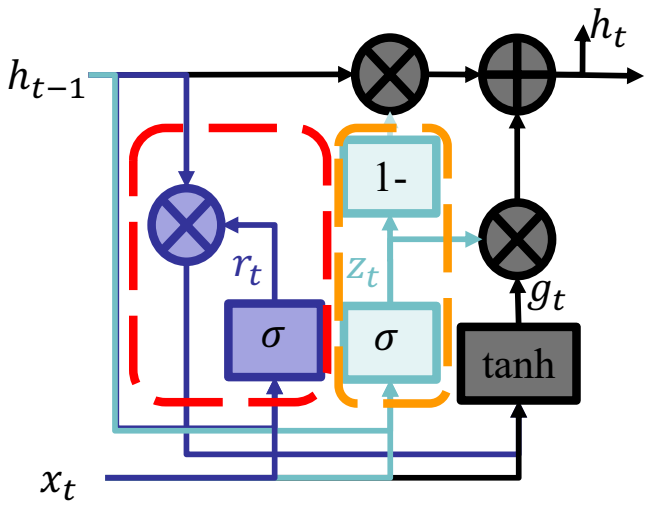
- GRU is more compressed in comparison to LSTM:

  GRU merges cell state and hidden state;

  GRU combines forget gate and input into update gate

- GRU directly uses h_t to retain information from previous cell or jump start with new information

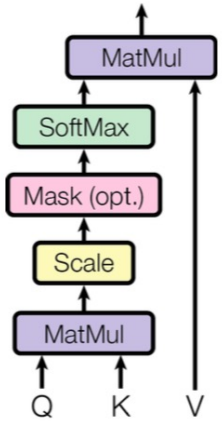




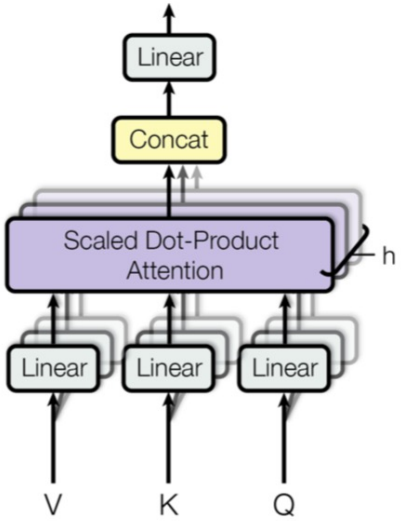
Transformer

- Feed forward network + Self attention

Scaled Dot-Product Attention



Multi-Head Attention



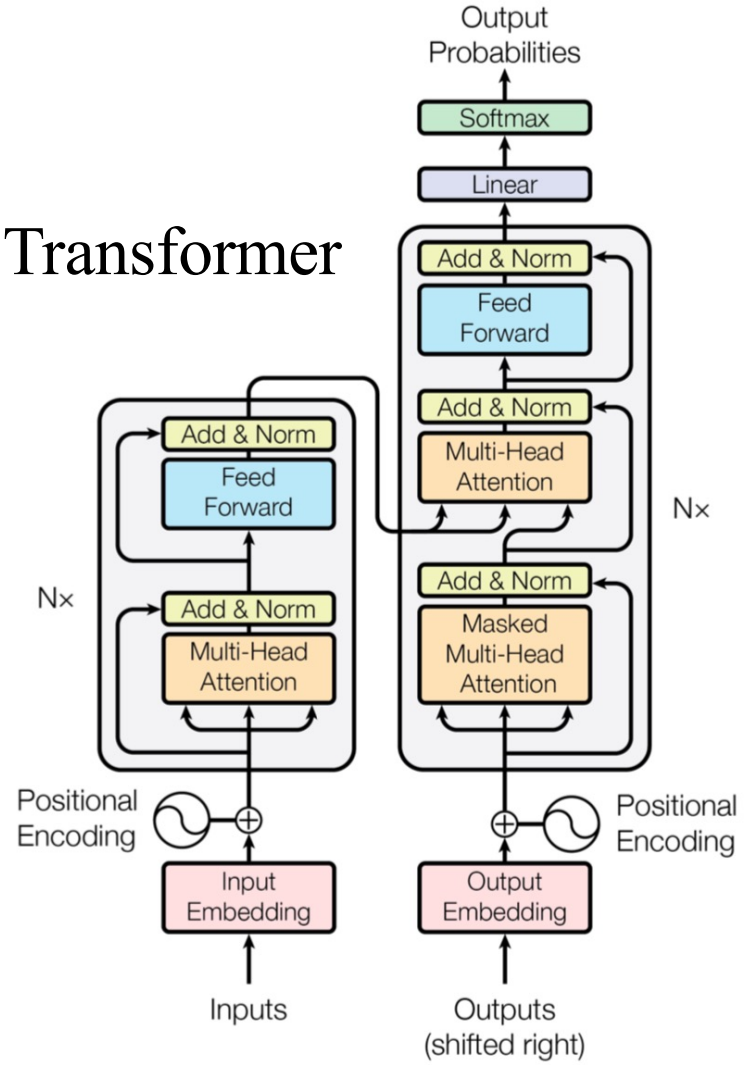
Attention mechanism

$$Q^{\mathcal{T}} = X^{\mathcal{T}} W_q^{\mathcal{T}}$$
$$K^{\mathcal{T}} = X^{\mathcal{T}} W_k^{\mathcal{T}}$$
$$V^{\mathcal{T}} = X^{\mathcal{T}} W_v^{\mathcal{T}}$$

$$S^{\mathcal{T}} = \text{softmax}(\frac{Q^{\mathcal{T}}(K^{\mathcal{T}})^{\mathcal{T}}}{\sqrt{d_k^{\mathcal{T}}}})$$

$$M^{\mathcal{T}} = S^{\mathcal{T}} V^{\mathcal{T}}$$

Transformer

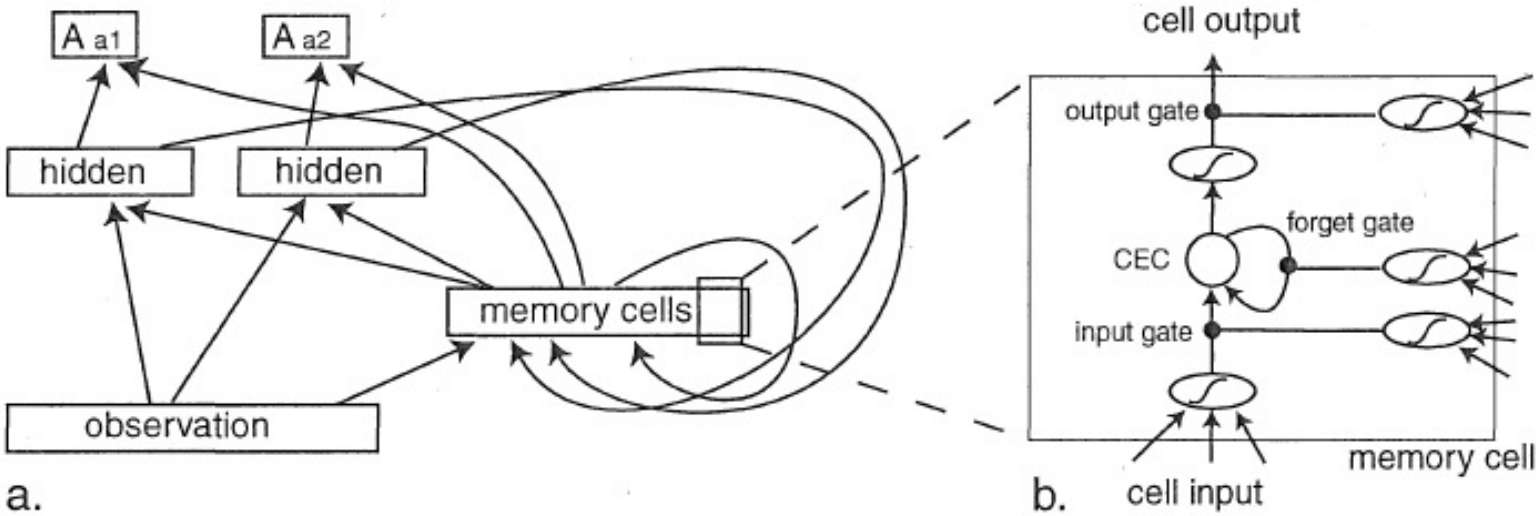


[Vaswani et al. 2017]

Example: Non-Markovian RL

- **RL-LSTM:** Model-free RL with LSTM for non-Markovian tasks with long-term dependencies
- Directly approximate the value function of advantage learning and LSTM to backpropagate prediction error

$$E^{TD}(t) = V(s(t)) + \frac{r(t) + \gamma V(s(t+1)) - V(s(t))}{\kappa} - A(s(t), a(t))$$



Bakker, Bram. "Reinforcement learning with long short-term memory." In *Advances in neural information processing systems*, pp. 1475-1482. 2002.

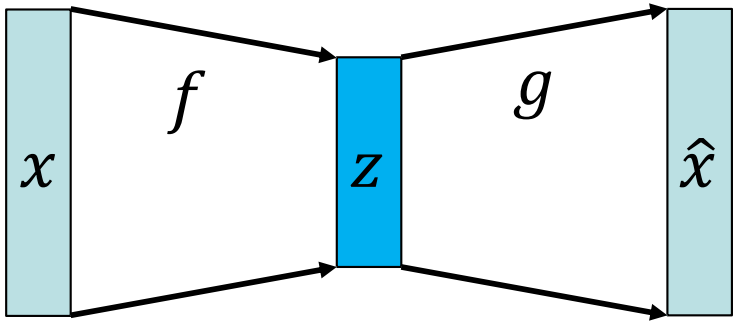
Generative Models

Generative vs. Discriminative

- Discriminative models
  - Predict a label or category given the features of an instance of data
  - Learn the **boundary** between classes
- Generative models:
  - Predict features given a certain label
  - Model the **distribution** of individual classes

Autoencoders

- **Neural networks trained to copy its input to its output**
  - **Encoder:** map the input x to a hidden representation z
  - **Decoder:** map the hidden representation z to the output $\hat{x}$
  - **Bottleneck layer** with lower dimension
  - Loss function for training: $L(x, \hat{x}) = L\big(x, g(f(x))\big)$
- **Regularization**
  - Sparsity: Sparse Autoencoder
  - Robustness: Denoising Autoencoder
- **Dimensionality reduction & PCA**



Encoder $z = f(x)$ Decoder $\hat{x} = g(z)$

Sparse Autoencoders

- **Sparsity penalty on Bottleneck layer**

$$J_s(W,b) = J(W,b) + \beta \sum_{j=1}^{S} KL(\rho||\hat{\rho}_j)$$

- **KL divergence for penalty on hidden representations**

$$KL(\rho||\hat{\rho}_j) = \rho \log \frac{\rho}{\hat{\rho}_j} + (1-\rho) \log \frac{1-\rho}{1-\hat{\rho}_j}$$

Expected activation

Average activation for *j*-th hidden unit

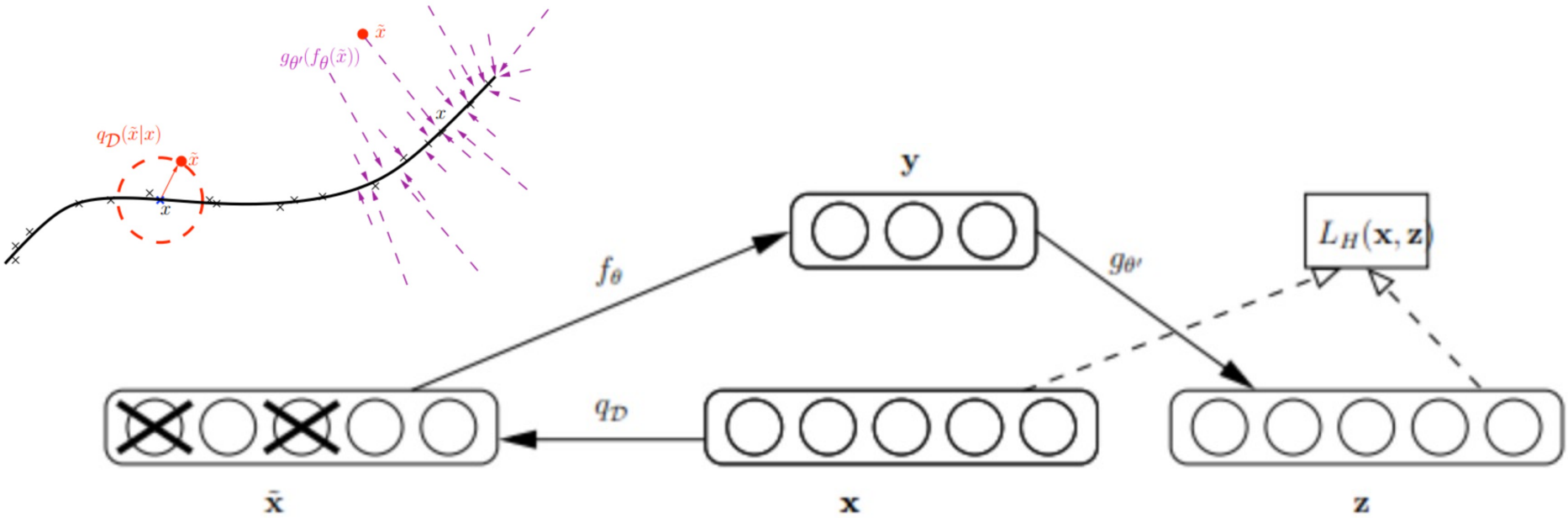
- **For *j*-th hidden representation**

$$\hat{\rho}_j = \frac{1}{m} \sum_{i=1}^{m} \left[a_j^{(2)}(x_i) \right]$$

Activation for specific *input x*

Denoising Autoencoders

- Noise perturbation on input x to eliminate partial components
- Minimize loss $L_H(x, z)$ between x and z
- Relation to Dropout (on hidden units)



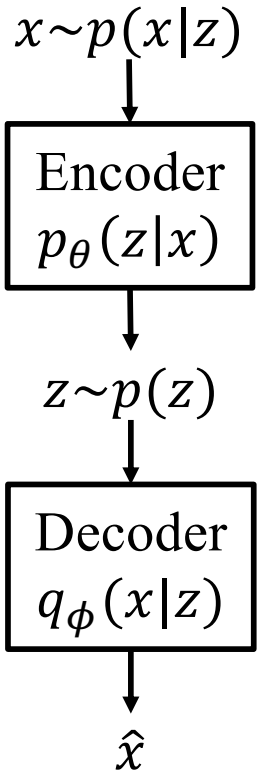
Variational Autoencoders (VAEs)

- **Autoencoder with regularization on distribution for encoder/decoder to organize the latent space for generation**
  - Encode the input x as distribution $p_\theta(z|x)$ over the latent space
  - Decode the sample from the latent space using the distribution
  - Backpropagate the reconstruction error for training

$$L(\theta,\phi) = \sum_i -\mathbb{E}_{z\sim p_\theta(z|x_i)}[\log q_\phi(x_i|z)] + \mathbb{D}_{\mathrm{KL}}(p_\theta(z|x_i)||p(z))$$

Reconstruction loss

Regularization

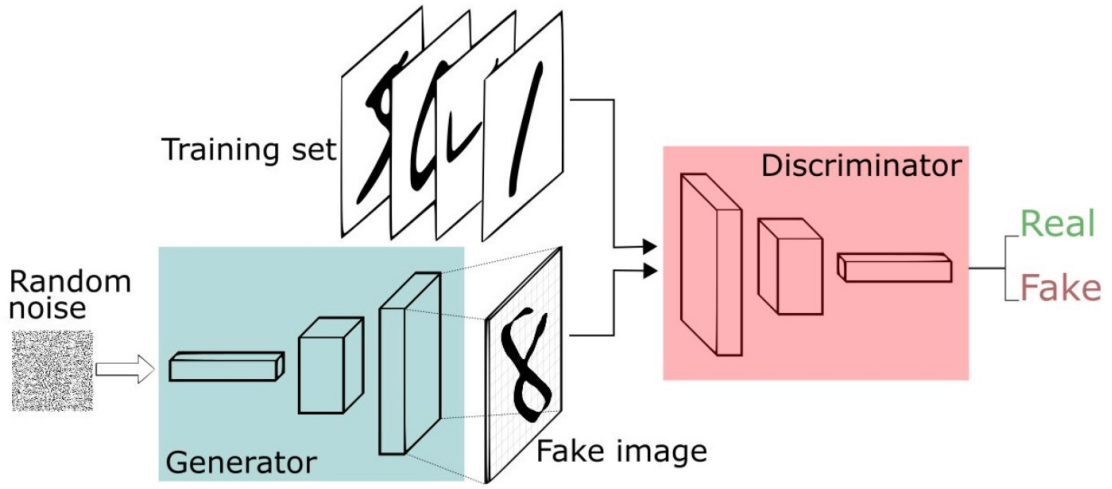


- **Equivalent to ELBO in variational inference**
- **Reparametrization trick**

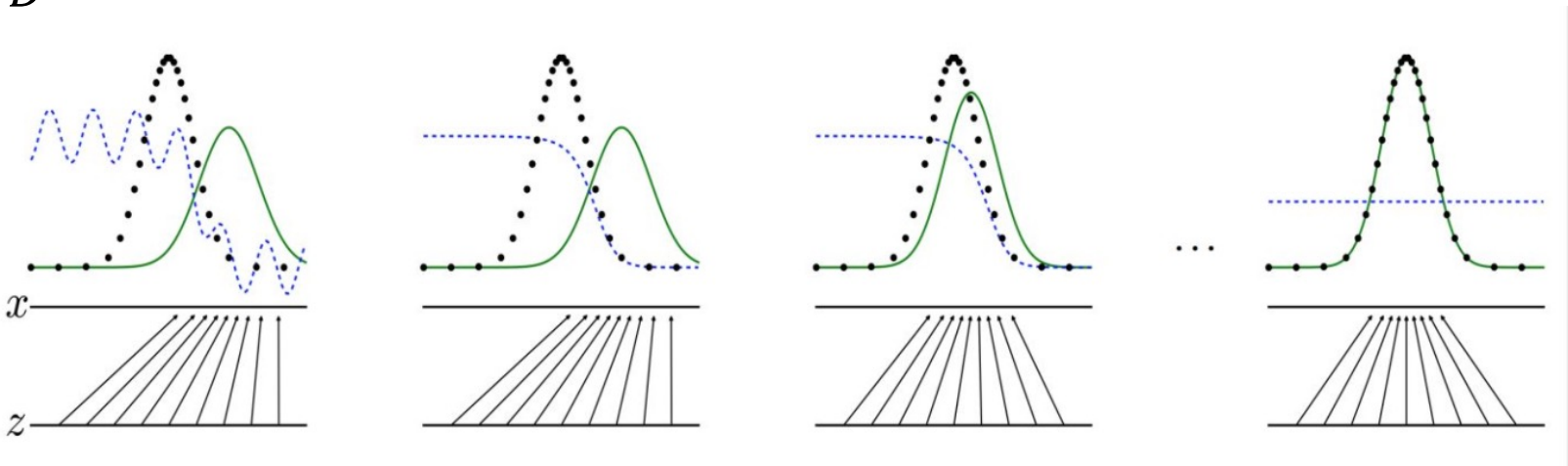
$$\varepsilon\sim\mathcal{N}(0,1) \xrightarrow{z=\mu+\varepsilon\cdot\sigma} z\sim\mathcal{N}(\mu,\sigma^2) \longrightarrow \begin{matrix}\mu = f(x)\\ \log\sigma = g(x)\end{matrix}$$

Generative Adversarial Networks

- **Generator**
- **Discriminator**
- **Update Alternately**



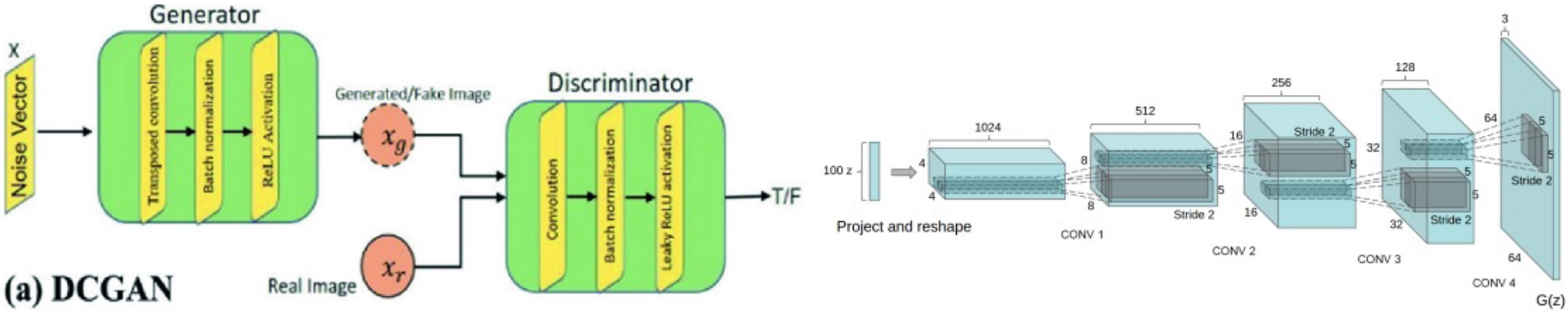
$$\min_G \max_D L(G,D) = \mathbb{E}_{x\sim p_X(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z))]$$



Goodfellow I. NIPS 2016 tutorial: Generative adversarial networks[J]. arXiv preprint arXiv:1701.00160, 2016.

Deep Convolutional GAN

- DCGAN: Combining CNN and GAN
  - Replace pooling layers with **fractional-stride convolutions**
  - ***Batch Normalization*** and ***LeakyReLU*** activation
  - Fully convolution network

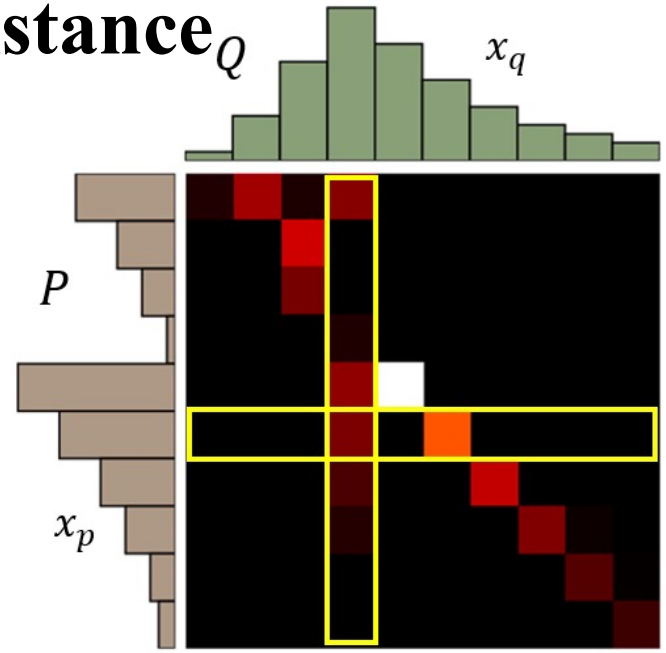


Radford, A., Metz, L., & Chintala, S. (2016). Unsupervised representation learning with deep convolutional generative adversarial networks. In 4th International; Conference on Learning Representations ICLR 2016—Conference Track Proceedings (pp. 1–16).

Wasserstein GAN

- **WGAN: GAN using Wasserstein distance**
  - Earth mover & moving plan

    Average distance of a plan γ



$$\mathrm{B}(\gamma) = \sum_{x_p, x_q} \gamma(x_p, x_q) \|x_p - x_q\|$$

Wasserstein distance $W(P, Q) = \min_{\gamma \in \Pi} B(\gamma)$

  - Optimization

$$V(G, D) = \max_{D \in \mathcal{C}} \mathbb{E}_{x \sim p_{\text{data}}}[D(x)] + \mathbb{E}_{x \sim p_G}[D(x)]$$

$$\mathcal{C} = \{D \mid \|D(x_1) - D(x_2)\| \leq \|x_1 - x_2\|\}$$

1-Lipschitz

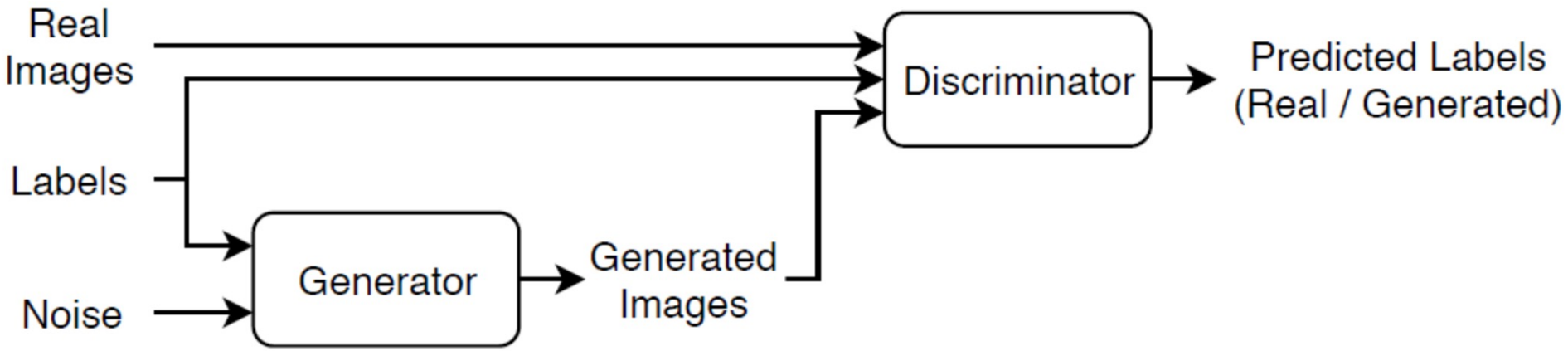
  - WGAN-GP & SNGAN

Conditonal GAN

- **cGAN: GAN using labels during training**
  - Generator: generate data with the same structure as the training data observations corresponding to the same label.
  - Discriminator: classify the observations as "real" or "generated", given training and generated data, this network

$$\min_G \max_D L(G,D) = \mathbb{E}_{x\sim p_X(x)}[\log D(x|y)] + \mathbb{E}_{z\sim p_z(z)}\left[\log(1 - D\left(G(z|y)\right)\right]$$

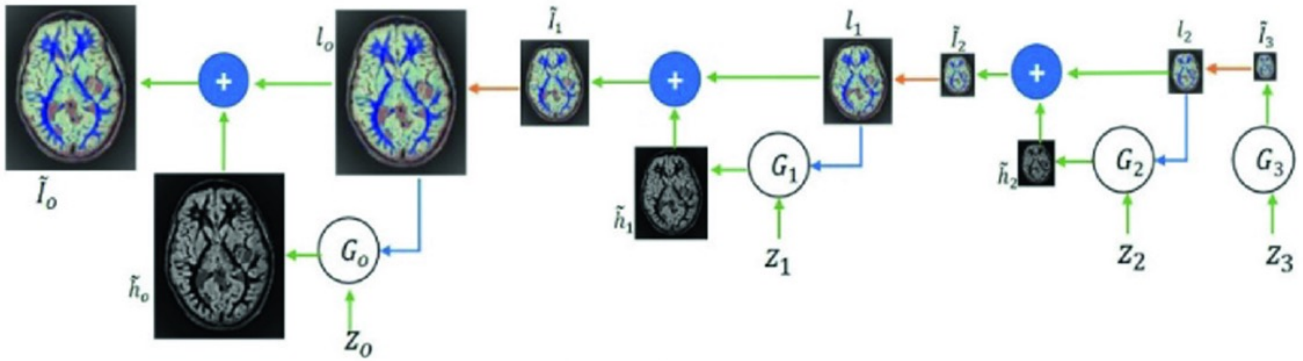
Label



LAPGAN

- **LAPGAN: Successive sampling for full resolution**
  - A set of generative models $\{G_0, G_1, \ldots, G_K\}$ at each level k of the Laplacian Pyramid L(I) to capture the coefficients h_k for input image I
  - h_k is calculated by difference of adjacent levels in Gaussian Pyramid $G_k(I)$ and up-sampled value $U(G_k(I))$

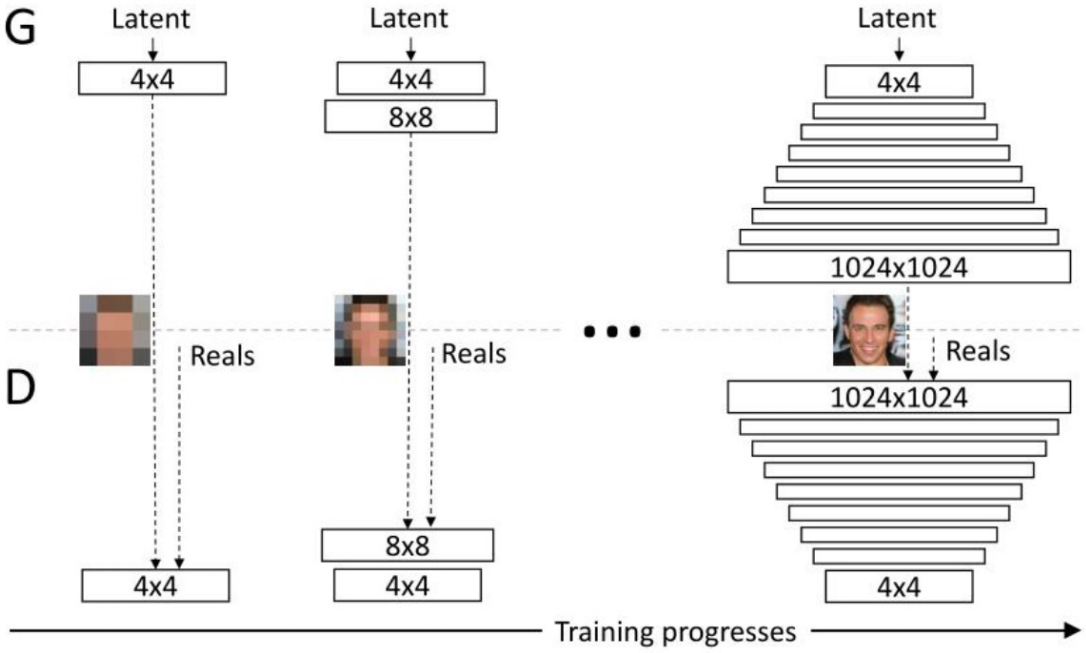
$$h_k = L_k(I) = G_k(I) - U\big(G_{k+1}(I)\big) = I_k - U(I_{k+1})$$

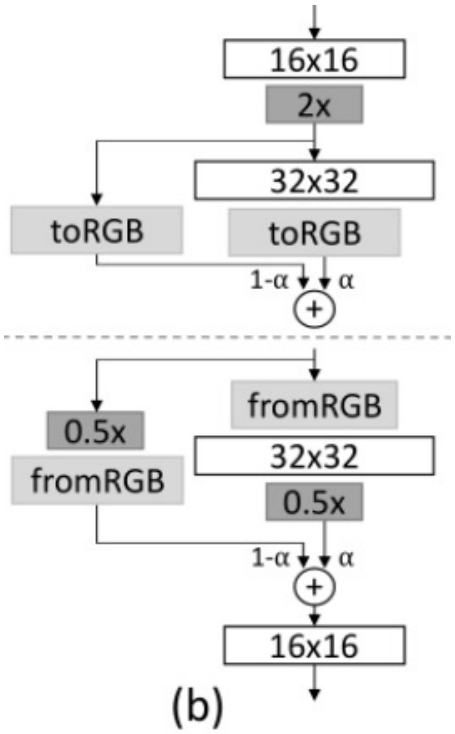


(b) LAPGAN

ProGAN

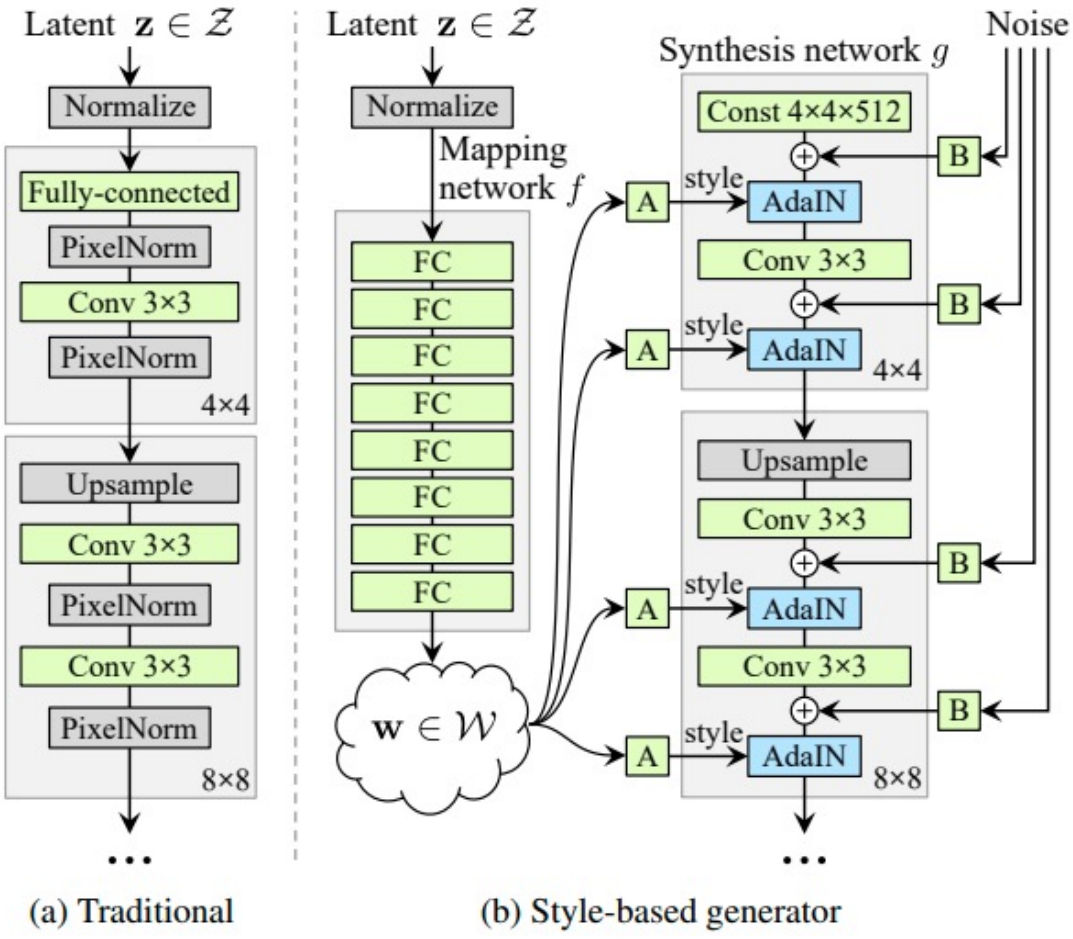
- **ProGAN: Progressive Growing GAN**
  - **Start from low resolution and refine details**
  - **Usage of Skip connection**





StyleGAN

- **StyleGAN: Regulating outputs**
  - **Style-based generator: Sequentially produce the simulated image**
  - **Transform the input of each level individually without altering other levels**

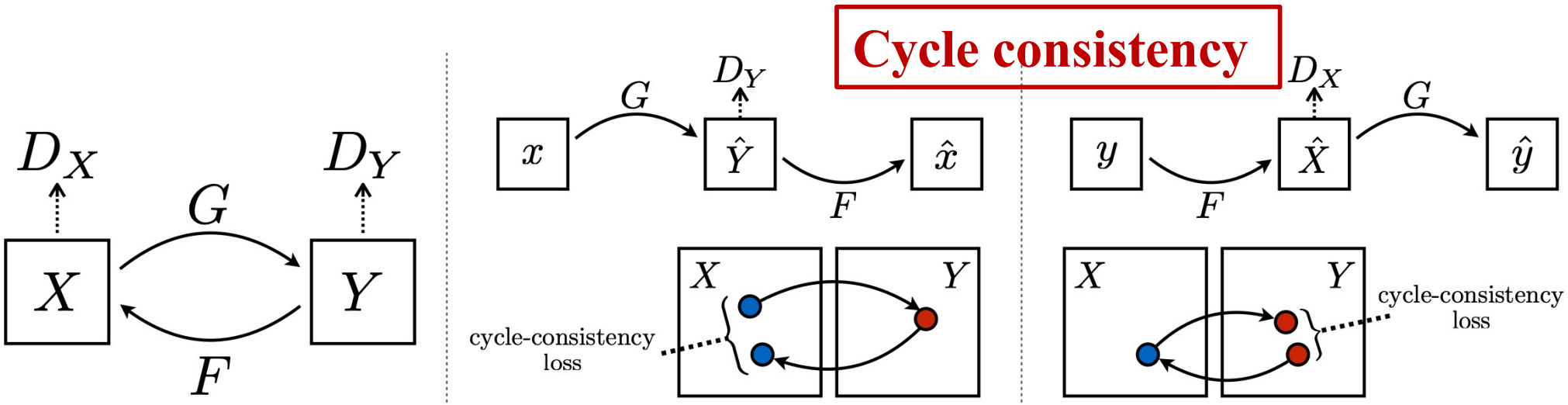


CycleGAN

- Absence of paired training data. Learn mapping function between two domains X and Y
- Two adversarial discriminators D_X and D_Y

$$\min_{G,F} \max_{D_X,D_Y} \mathcal{L} = L_{GAN}(G, D_Y, X, Y) + L_{GAN}(F, D_X, Y, X) + L_{cyc}(G, F)$$

$$L_{cyc}(G, F) = \mathbb{E}_{x\sim p(x)}\left[\left\|F\left(G(x)\right) - x\right\|_1\right] + \mathbb{E}_{y\sim p(y)}\left[\left\|G\left(F(y)\right) - y\right\|_1\right.$$



Jun-Yan Zhu*, Taesung Park*, Phillip Isola, and Alexei A. Efros. "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks", in IEEE International Conference on Computer Vision (ICCV), 2017.

Example: Generative Adversarial Imitation Learning

- Directly learn a policy from expert trajectories without inverse RL
- Fit distributions of states and actions with generative adversarial training

Policy

Discriminator

Causal Entropy

$$\mathbb{E}_{\pi}[\log(D(s,a)] + E_{\pi_E}[\log(1 - D(s,a))] - \lambda H(\pi)$$

Algorithm 1 Generative adversarial imitation learning

1: **Input:** Expert trajectories $\tau_E \sim \pi_E$, initial policy and discriminator parameters θ_0, w_0
2: **for** $i = 0, 1, 2, \ldots$ **do**
3: Sample trajectories $\tau_i \sim \pi_{\theta_i}$
4: Update the discriminator parameters from w_i to w_{i+1} with the gradient

$$\hat{\mathbb{E}}_{\tau_i}[\nabla_w \log(D_w(s,a))] + \hat{\mathbb{E}}_{\tau_E}[\nabla_w \log(1 - D_w(s,a))] \tag{17}$$

5: Take a policy step from θ_i to θ_{i+1}, using the TRPO rule with cost function $\log(D_{w_{i+1}}(s,a))$. Specifically, take a KL-constrained natural gradient step with

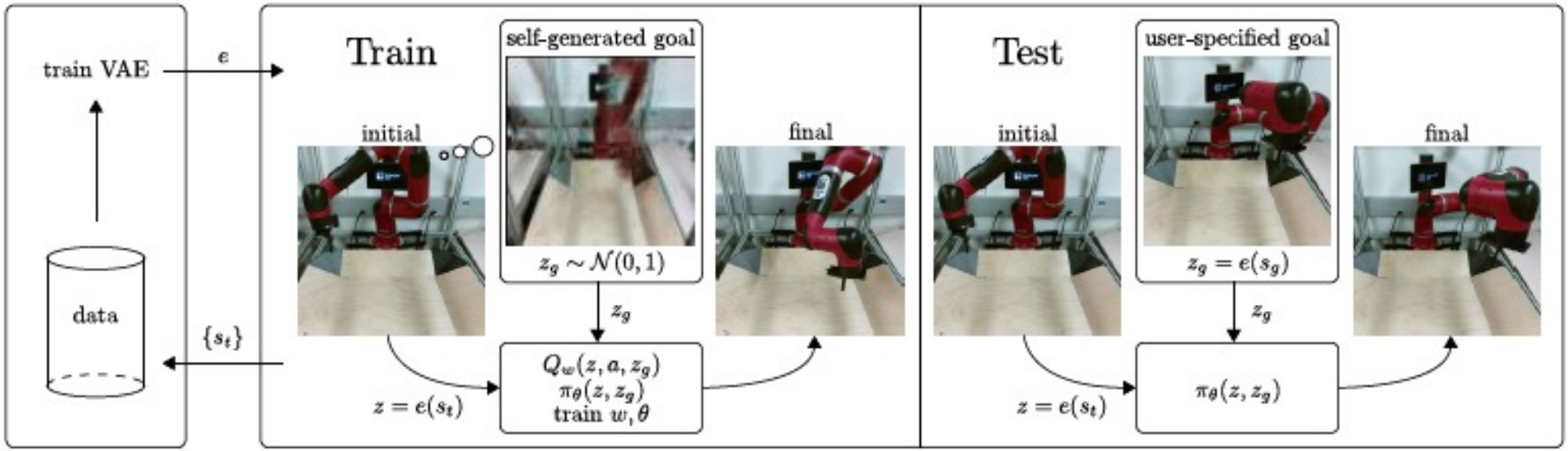
$$\hat{\mathbb{E}}_{\tau_i}[\nabla_\theta \log \pi_\theta(a|s) Q(s,a)] - \lambda \nabla_\theta H(\pi_\theta),$$
$$\text{where } Q(\bar{s},\bar{a}) = \hat{\mathbb{E}}_{\tau_i}[\log(D_{w_{i+1}}(s,a)) \,|\, s_0 = \bar{s}, a_0 = \bar{a}] \tag{18}$$

6: **end for**

Ho, Jonathan, and Stefano Ermon. "Generative adversarial imitation learning." In *Advances in neural information processing systems*, 2016.

Example: Visual RL

- Combine unsupervised representation learning and RL
- Train a VAE using data generated by exploration policy
  - Sample goals to train the policy
  - Embed the observation into a latent space
  - Compute distances in the latent space



Nair, Ashvin V., Vitchyr Pong, Murtaza Dalal, Shikhar Bahl, Steven Lin, and Sergey Levine. "Visual reinforcement learning with imagined goals." In *Advances in Neural Information Processing Systems*, pp. 9191-9200. 2018.

Institute of Media,
Information, and Network

Q & A

Many Thanks