CS286 AI for Science and Engineering

Lecture 11: Special Topics of Deep Learning

Jie Zheng (郑杰)
PhD, Associate Professor
School of Information Science and Technology (SIST), ShanghaiTech University

Fall, 2020

Outline

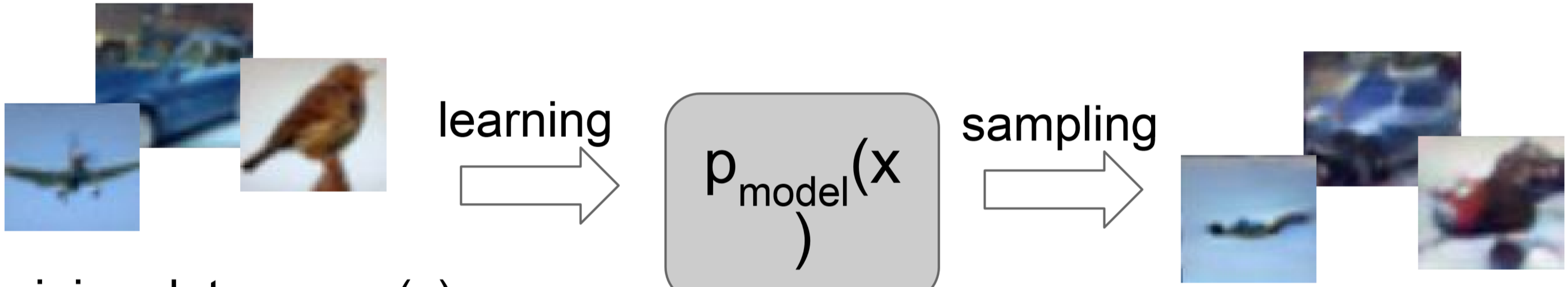
- Generative Adversarial Networks (GANs)
- Reinforcement Learning
- Graph Neural Networks (GNNs)

Generative Adversarial Networks (GANs)

立志 成才 报国 裕民

Generative modeling

- Given training data, generate new samples from the same distribution



Training data ~ $p_{data}(x)$

Objectives:

1. Learn $p_{model}(x)$ that approximates $p_{data}(x)$
2. **Sample new x from $p_{model}(x)$**

Taxonomy of generative models

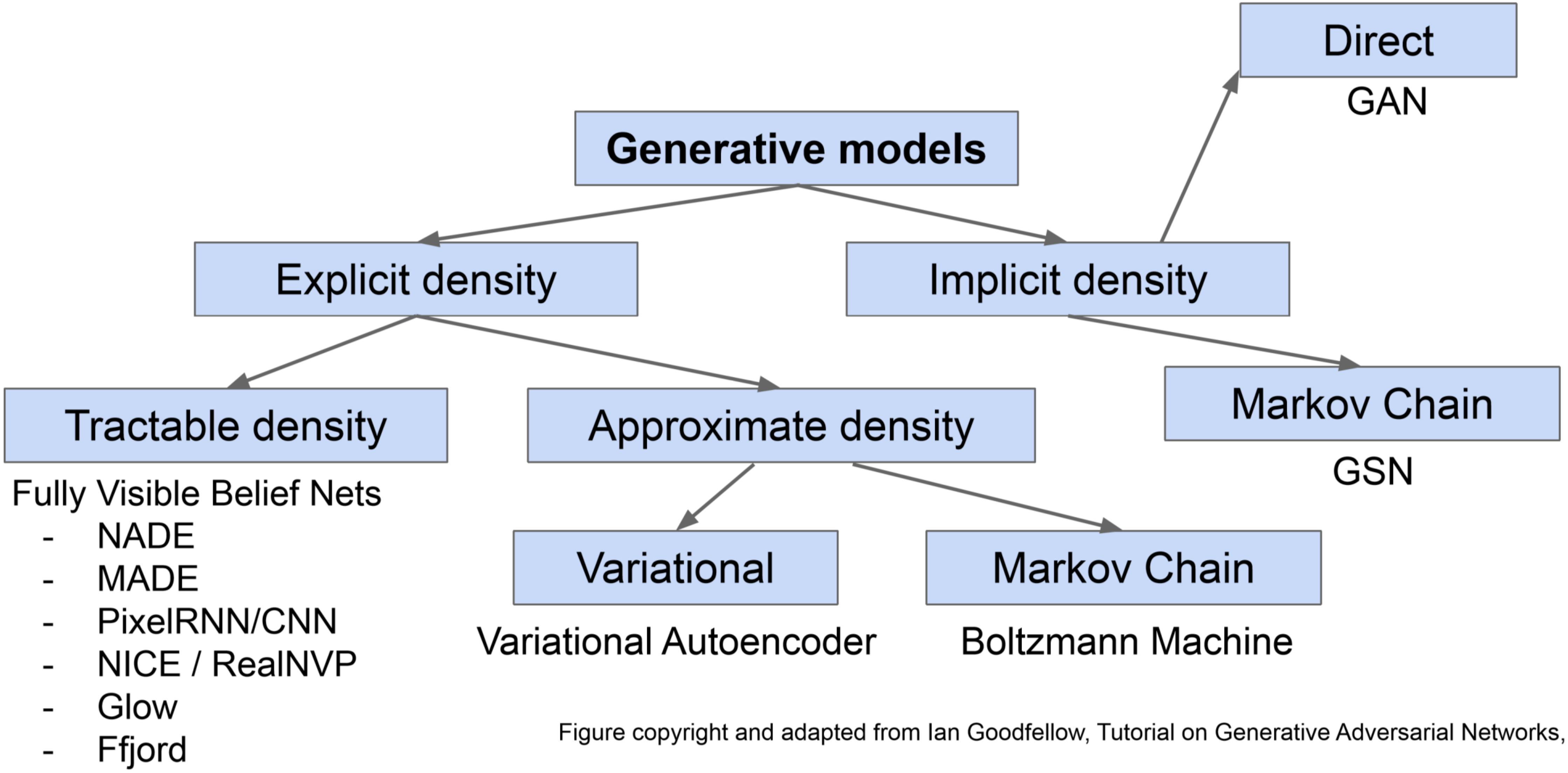
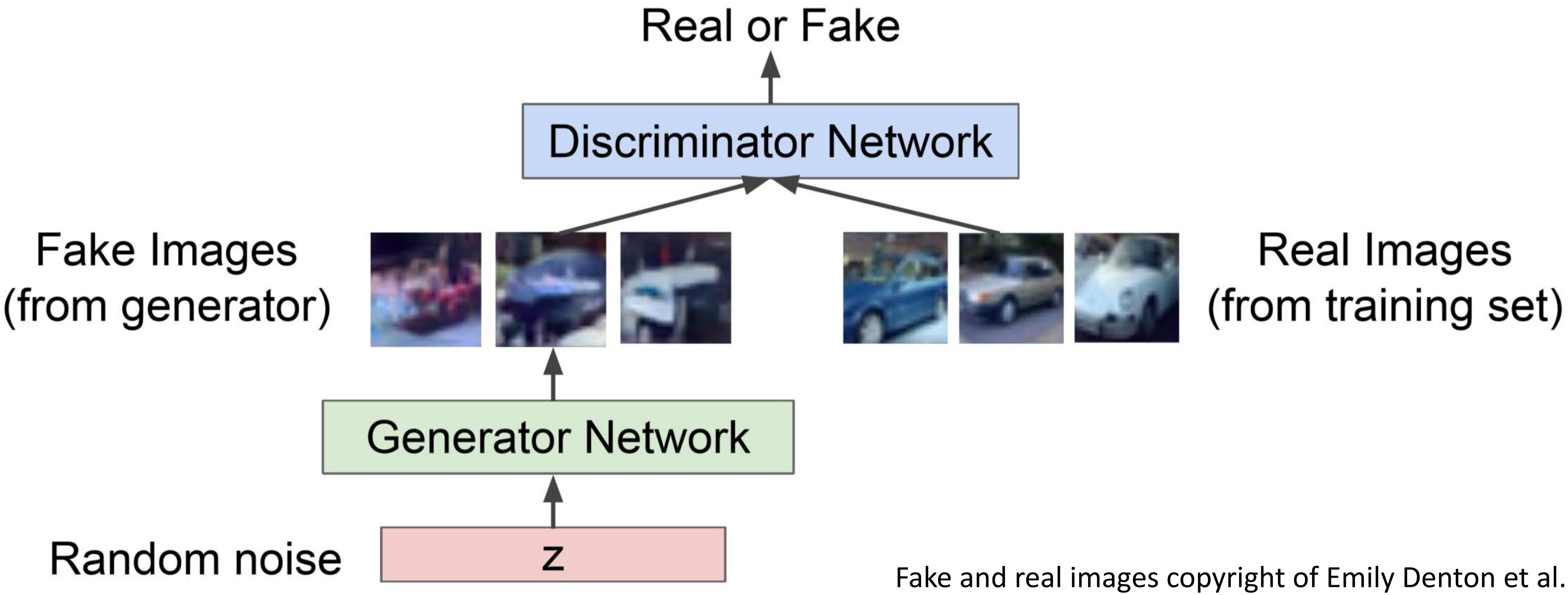


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

Generative Adversarial Networks (GANs)

- Use a neural network, called **generator**, to generate data
- Use another neural network, called **discriminator**, to determine if the data is real or fake



Fake and real images copyright of Emily Denton et al. 2015.

Cost functions of adversarial learning

- Let D denote the discriminator’s predicted probability of being real data
- **Discriminator**:
  - Try to distinguish between real and fake images
  - Cost function: cross-entropy loss for the task of classifying real vs. fake images:

$$\mathcal{J}_D = \mathbb{E}_{\mathbf{x}\sim\mathcal{D}}[-\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}}[-\log(1 - D(G(\mathbf{z})))]$$

- **Generator**:
  - Try to fool the discriminator by generating real-looking images
  - Cost function (one possible version) is the opposite of the discriminator’s:

$$\begin{aligned}\mathcal{J}_G &= -\mathcal{J}_D \\ &= \text{const} + \mathbb{E}_{\mathbf{z}}[\log(1 - D(G(\mathbf{z})))]\end{aligned}$$

Two-player game

- Minimax formulation:
  - The generator and discriminator are playing a zero-sum game against each other
  - Train jointly in a minimax game:

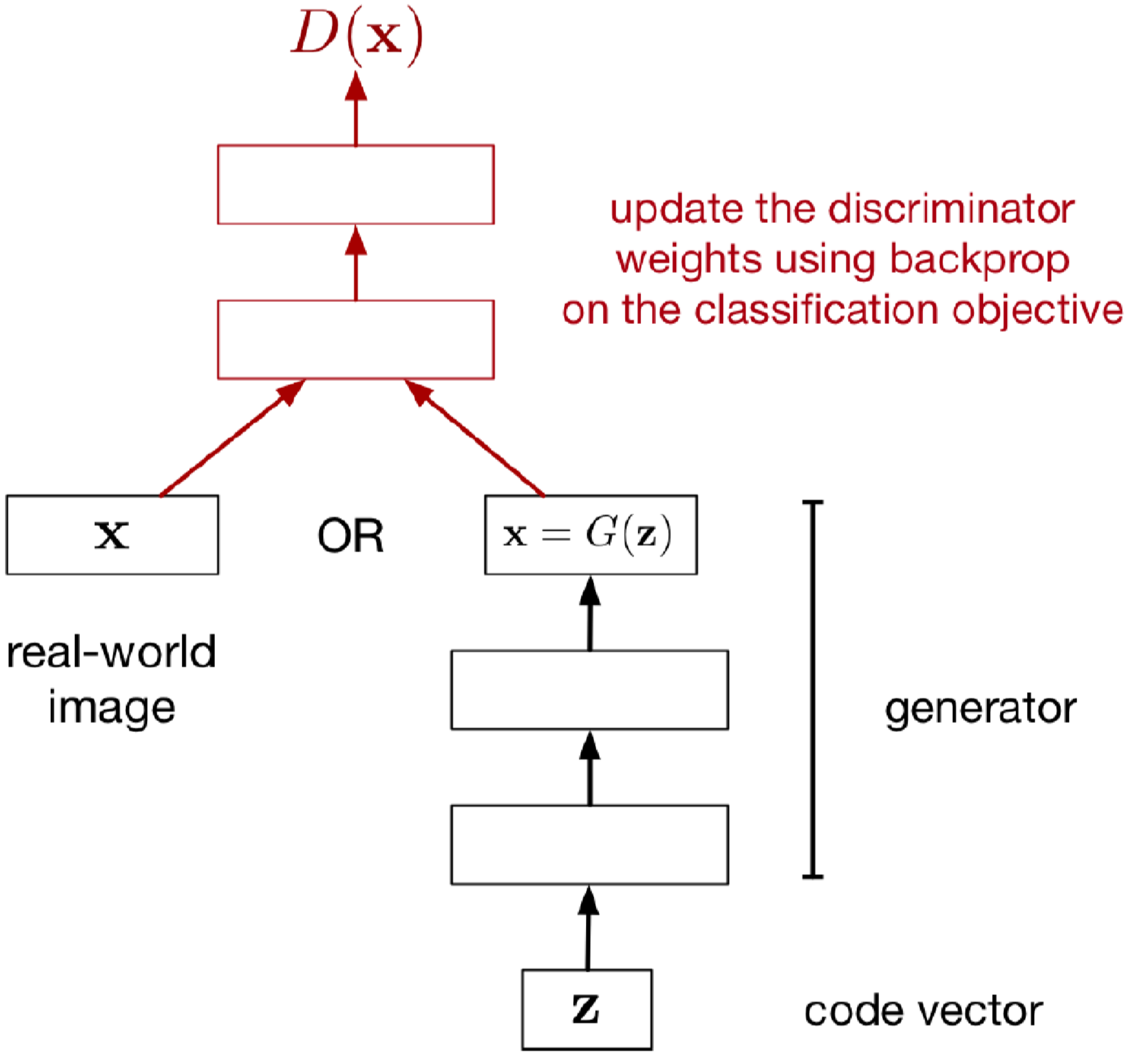
Discriminator outputs likelihood in (0,1) of real image

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log \underbrace{D_{\theta_d}(x)}_{\substack{\text{Discriminator output} \\ \text{for real data x}}} + \mathbb{E}_{z \sim p(z)} \log(1 - \underbrace{D_{\theta_d}(G_{\theta_g}(z))}_{\substack{\text{Discriminator output for} \\ \text{generated fake data G(z)}}}) \right]$$

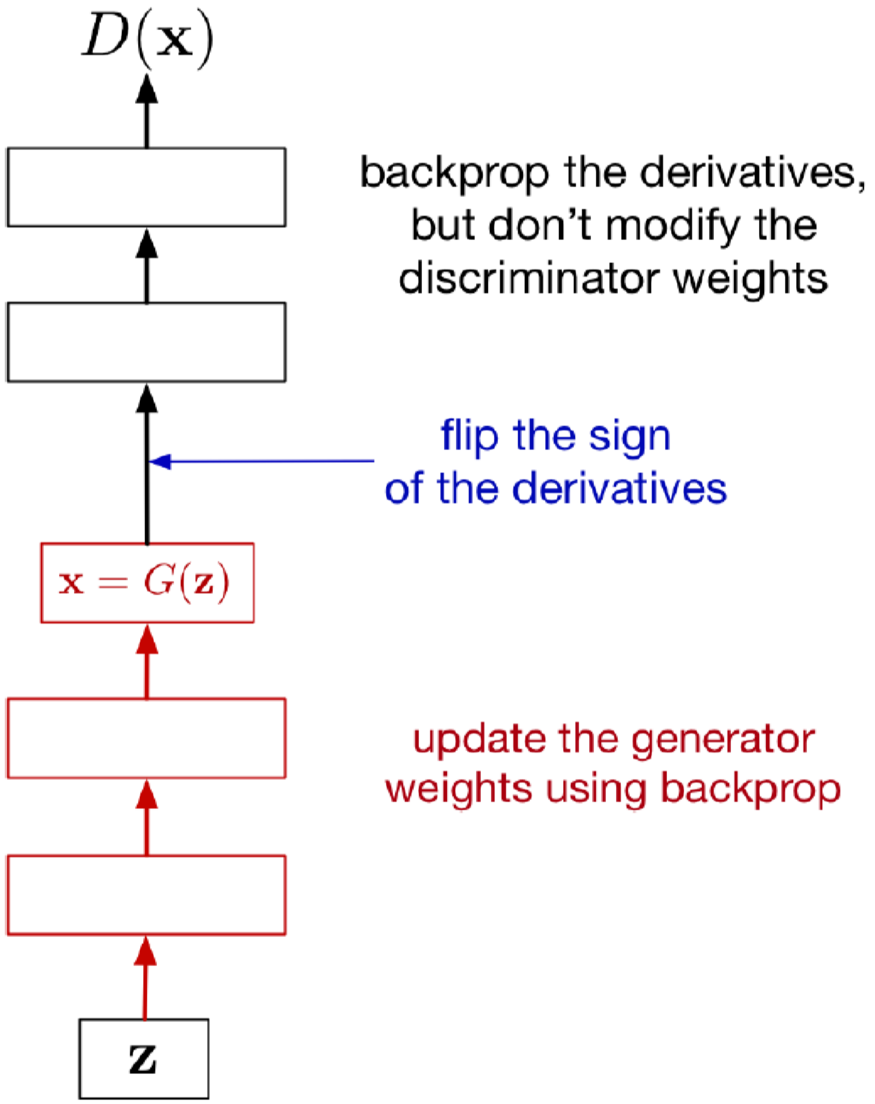
Learning procedure

- Updating the discriminator

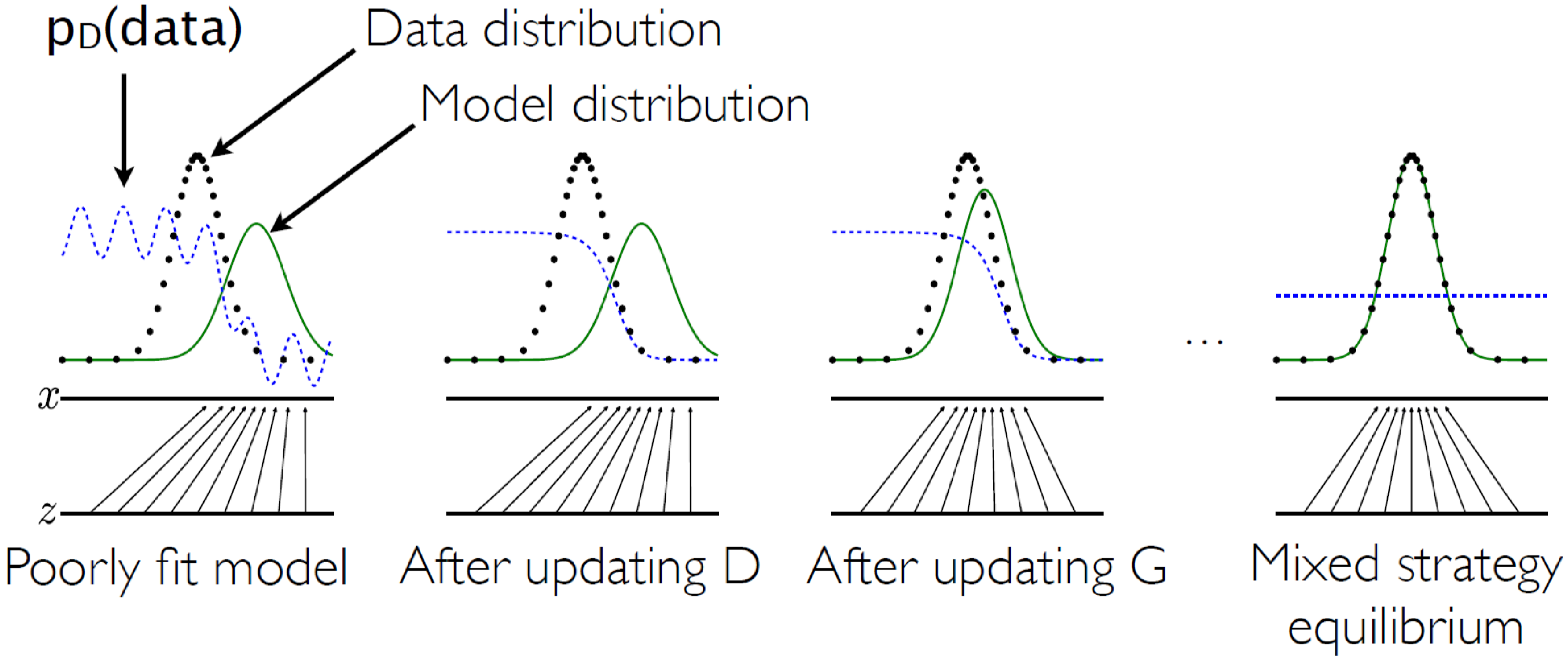


Learning procedure

- Updating the generator



Training GANs



Training GANs

- In general, training a GAN is tricky and unstable
- Since GANs were introduced in 2014, there have been hundreds of papers introducing various architectures and training methods
- GAN Zoo: https://github.com/hindupuravinash/the-gan-zoo
- Many tricks:
  - S. Chintala, How to train a GAN, ICCV 2017 tutorial
  - https://github.com/soumith/ganhacks

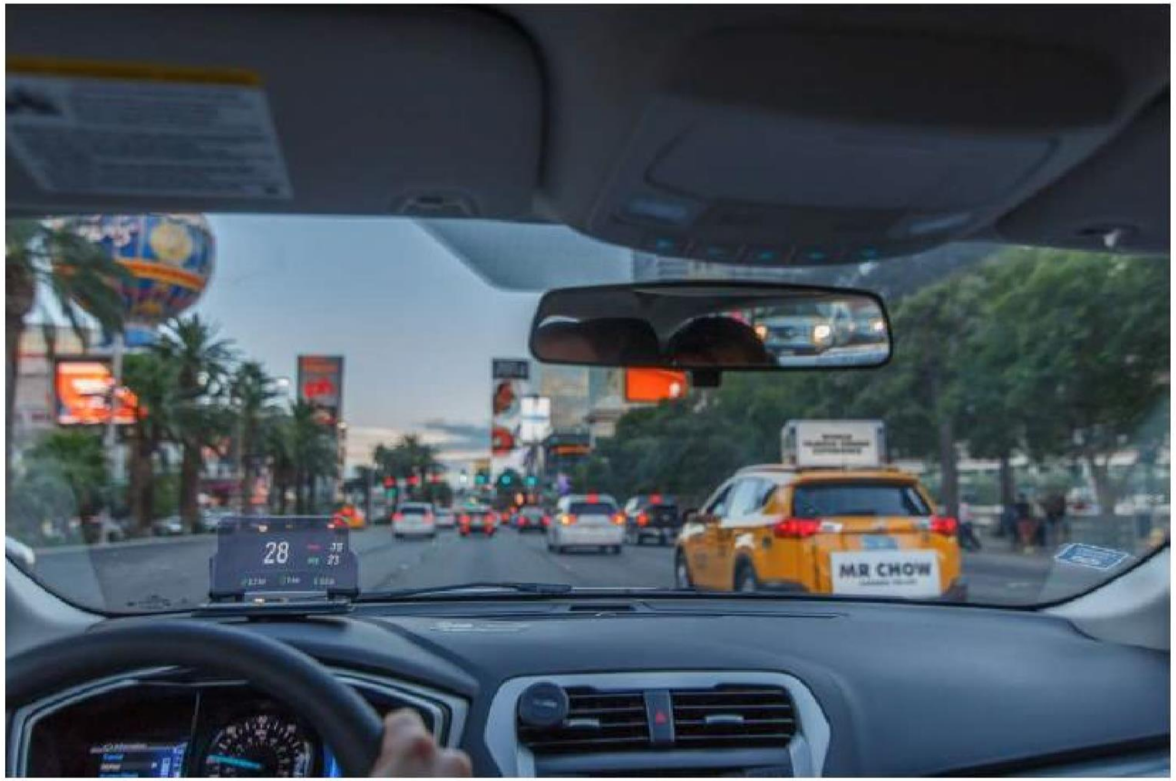
Interpretable vector arithmetic

Radford et al, ICLR 2016

Reinforcement Learning

Reinforcement learning (RL)

- A new class of problems: Reward-based
  - E.g. autonomous driving
  - What is my next move?
  - Reaching the destination with minimum cost

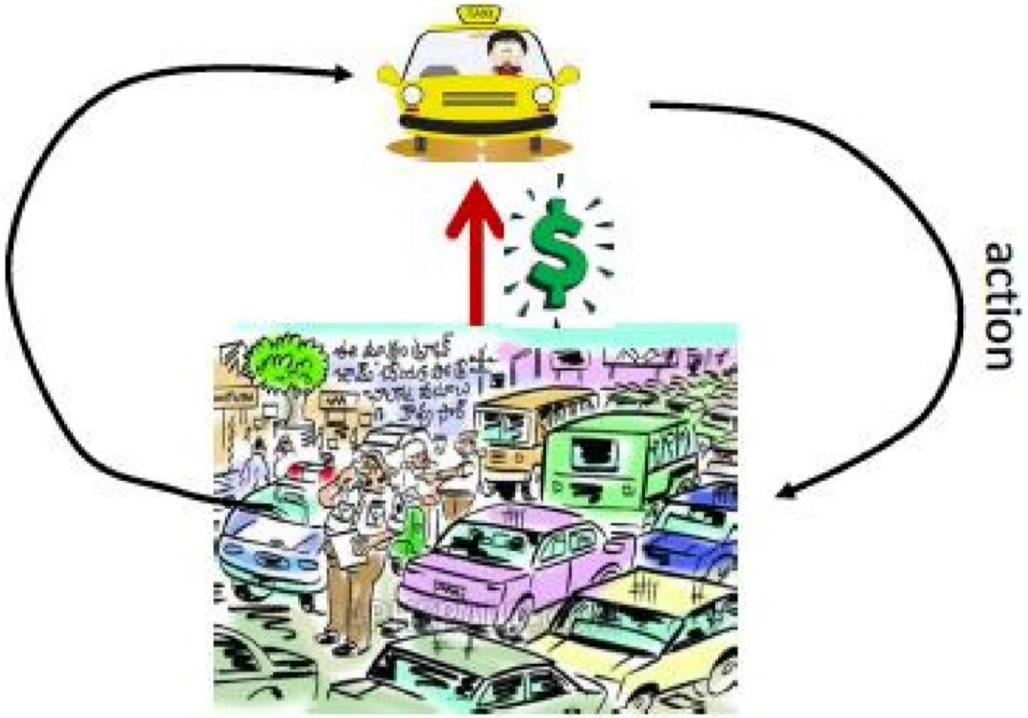


Reinforcement learning (RL)

- Common theme: control problems where
  - Your actions beget rewards
    - Win a Go game
    - Make money by investing
    - Get home sooner
  - But not deterministically
    - A world out there that is not predictable
- From experience of **belated/delayed** rewards, you must learn to act rationally

RL problem setting

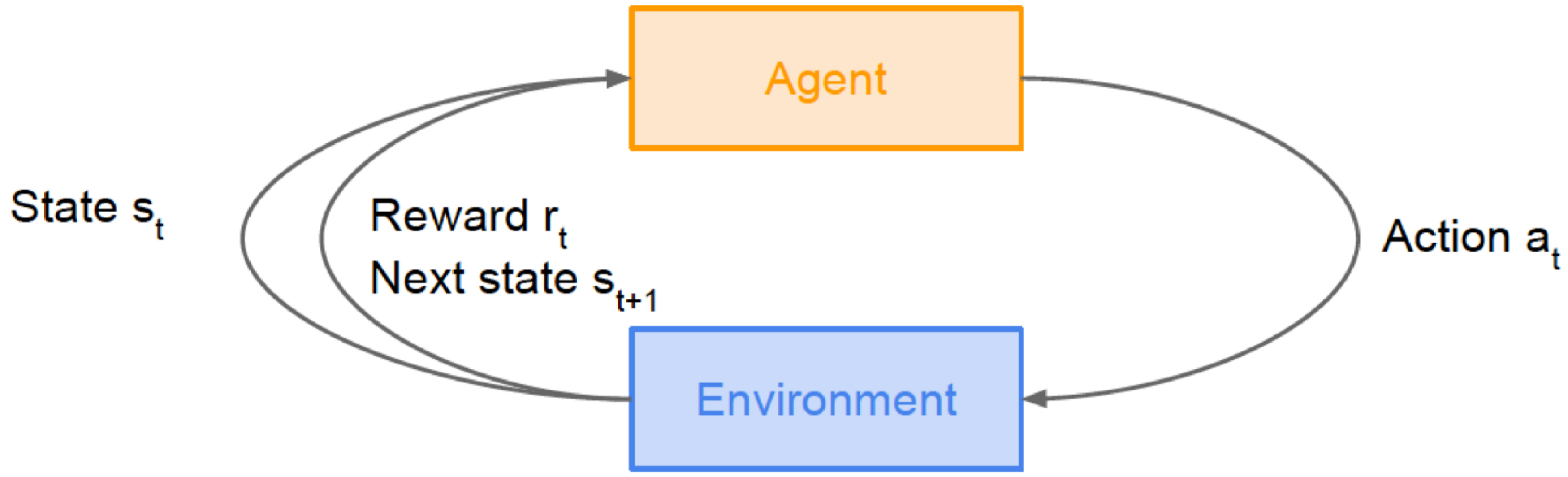
- An agent operates in an environment
- The agent takes actions that
  - affect the environment
  - change in a somewhat unpredictable way
  - affect the agent's situation
- The agent also receives rewards
  - which may be apparent immediately
  - or not apparent for a very long time



Problem to solve:

How must the agent behave to maximize its rewards?

Markov Decision Process (MDP)



- Markov assumption:
  - All relevant information is encapsulated in the current state
  - i.e. the policy, reward, and transitions are all independent of past states given the current state
- Assume a fully observable environment, i.e. the state can be observed directly

Formal definition of MDP

上海科技大学 ShanghaiTech University

A *Markov Decision Process* is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

- $\mathcal{S}$ is a finite set of states
- $\mathcal{A}$ is a finite set of actions
- $\mathcal{P}$ is a state transition probability matrix,
  $\mathcal{P}_{ss'}^{a} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s, A_t = a\right]$
- $\mathcal{R}$ is a reward function, $\mathcal{R}_s^a = \mathbb{E}\left[R_{t+1} \mid S_t = s, A_t = a\right]$
- γ is a discount factor $\gamma \in [0, 1]$.

立志成才报国裕民

Formal definition of MDP

上海科技大学 ShanghaiTech University

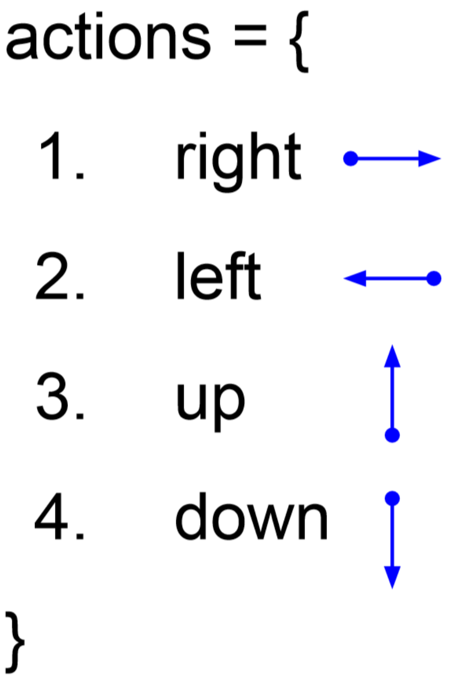
- At time step $t = 0$, environment samples initial state $s_0 \sim p(s_0)$
- Then, for $t = 0$ until done:
  - Agent selects action a_t
  - Environment samples reward $r_t \sim R(\cdot\,|s_t, a_t)$
  - Environment samples next state $s_{t+1} \sim P(\cdot\,|s_t, a_t)$
  - Agent receives reward r_t and next state s_{t+1}

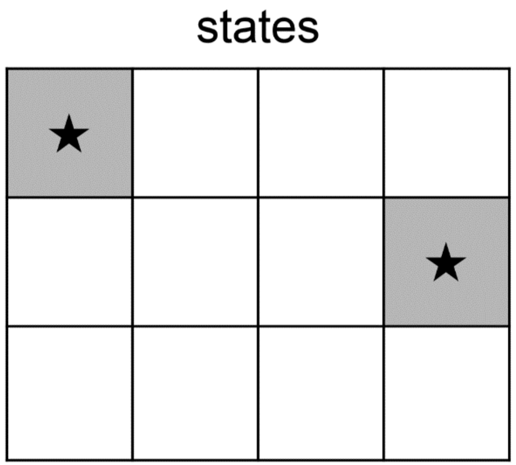
- A policy π is a function from S to A that specifies what action to take in each state
- **Objective**: find policy π^* that maximizes cumulative discounted reward: $\sum_{t \geq 0} \gamma^t r_t$

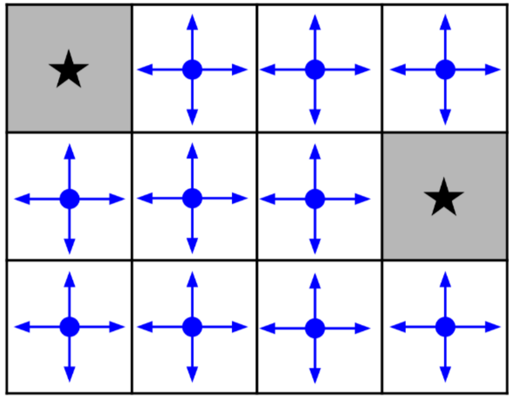
立志成才报国裕民

A simple MDP: Grid World

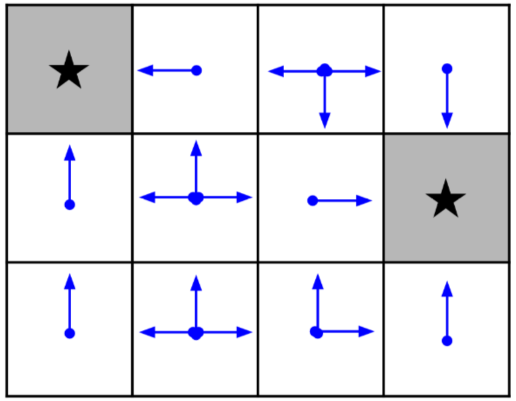
- Set a negative "reward" for each transition
- **Objective**: reach one of terminal states (greyed out) in the least number of actions







Random Policy



Optimal Policy

Problems in MDP

- Planning: Given a complete MDP as input, compute a policy with optimal expected return
  - Goal: maximize the expected return, $R = \mathrm{E}_{p(\tau)}[r(\tau)]$
  - The expectation is over both the environment’ s dynamics and the policy, but we only have control over the policy

- Learning: Given samples of trajectories of an unknown MDP,
  - Prediction: estimate the expected return given a policy
  - Control: find the optimal policy that maximizes the expected return

Value function and Q-value function

上海科技大学 ShanghaiTech University

- Following a policy produces sample trajectories (or paths) s_0, a_0, r_0, s_1, a_1, r_1, …
- How good is a state?

The **value function** at state s is the expected cumulative reward from following the policy from state s:

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, \pi\right]$$

- How good is a state-action pair?

The **Q-value function** at state s and action a is the expected cumulative reward from taking action a in state s and then following the policy:

$$Q^{\pi}(s,a) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi\right]$$

立志成才报国裕民

Bellman equation

上海科技大学 ShanghaiTech University

- The optimal Q-value function Q^* is the maximum expected cumulative reward achievable from a given (state, action) pair:

$$Q^*(s,a) = \max_{\pi} \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi\right]$$

- Q^* satisfies the following **Bellman equation**:

$$Q^*(s,a) = \mathbb{E}_{s'\sim\mathcal{E}}\left[r + \gamma \max_{a'} Q^*(s',a')|s,a\right]$$

- **Intuition**: If the optimal state-action values for the next time-step $Q^*(s',a')$ are known, then the optimal strategy is to take the action that maximizes the expected value of $r + \gamma Q^*(s',a')$
- The optimal policy $\pi^* = \text{argmax}_a Q^*(s,a)$

立志成才报国裕民

Value Iteration algorithm

- **Value Iteration algorithm** uses the Bellman equation for an iterative update

$$Q_{i+1}(s,a) = \mathbb{E}\left[r + \gamma \max_{a'} Q_i(s',a') | s,a\right]$$

  - Q_i will converge to Q^* as $i \to \infty$

- **Problem**: Not scalable
  - Must compute Q(s, a) for every (state, action) pair
  - Often computationally infeasible to compute for the entire state space
- **Solution**: Use a function approximator to estimate Q(s, a), e.g. a neural network

- But Q-values are more useful, as they give us optimal policies
  - An iterative algorithm similar to the value iteration algorithm was found by Bellman, which is called **Q-Value Iteration** algorithm

Q-learning

- **Q-learning**: Use a function approximator to estimate the action-value function

$$Q(s, a; \theta) \approx Q^*(s, a)$$

function parameters (weights)

- Learn $Q(s, a)$ values as you go
  - Receive a sample (s, a, s', r)
  - Consider your old estimate: $Q(s, a)$
  - Consider your new sample estimate
  - Incorporate the new estimate into a running average

- If the function approximator is a deep neural network, then it is called **deep Q-learning**

Q-learning properties

- Q-learning converges to optimal policy, even if you are acting suboptimally
  - It is called **off-policy learning**, as the policy being trained is not necessarily the one being executed

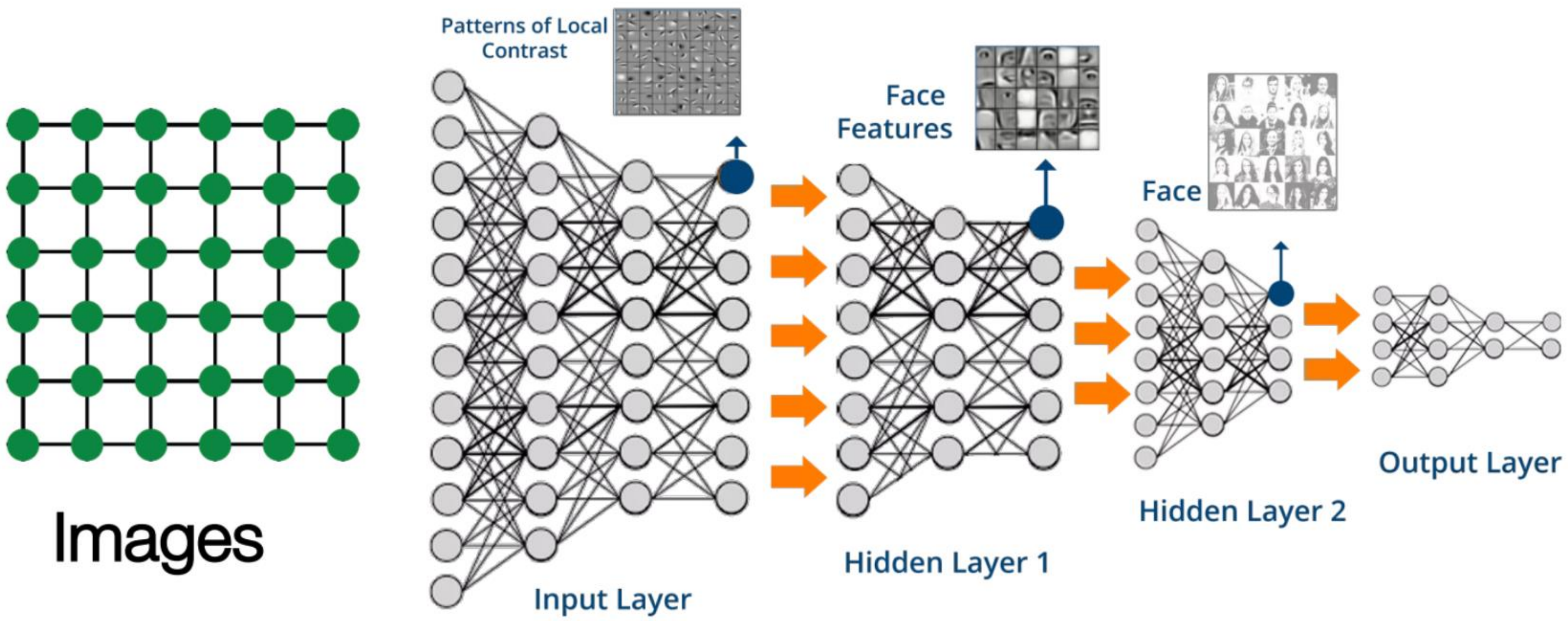
- **Caveats**:
  - You have to explore enough
  - You have to eventually make the learning rate small enough, but not decrease it too quickly
  - In the limit, it doesn' t matter how you select actions

Graph Neural Networks (GNNs)

Outline of a survey on GNNs

- Background of GNN
- Categorization and frameworks
- Overview of GNN models
  - Recurrent Graph Neural Networks
  - Convolutional Graph Neural Networks
  - Graph Autoencoders
  - Spatial–Temporal Graph Neural Networks
- Applications
- Challenges and future directions

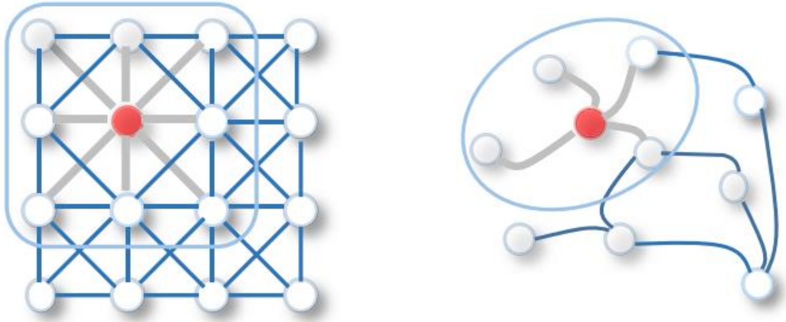
Modern deep learning



- Data are typically represented in Euclidean space
- Designed for simple sequences or grids

Motivation for GNNs

- Lots of data are represented in the form of graphs
  - E-commerce: interactions between users and products
  - Chemistry: molecules are modeled as graphs, and their bioactivities need to be identified for drug discovery
  - Citation network: articles are linked to each other via citation



- Properties of data:
  - Graphs can be irregular
  - The assumption that instances are independent of each other no longer holds

Graph Neural Networks (GNNs)

- **Graph Neural Networks (GNNs)** are deep learning-based methods that operate on graph domain
  - "They capture the dependence of graphs via message passing between the nodes of graphs" (Jie Zhou et al. , arXiv, 2019)

- Categorization:
  - Recurrent GNNs (RecGNNs)
  - Convolutional GNNs (ConvGNNs)
  - Graph Autoencoders (GAEs)
  - Spatial-Temporal GNNs (STGNNs)

Frameworks

- Input:
  - Graph structure
  - Node content information

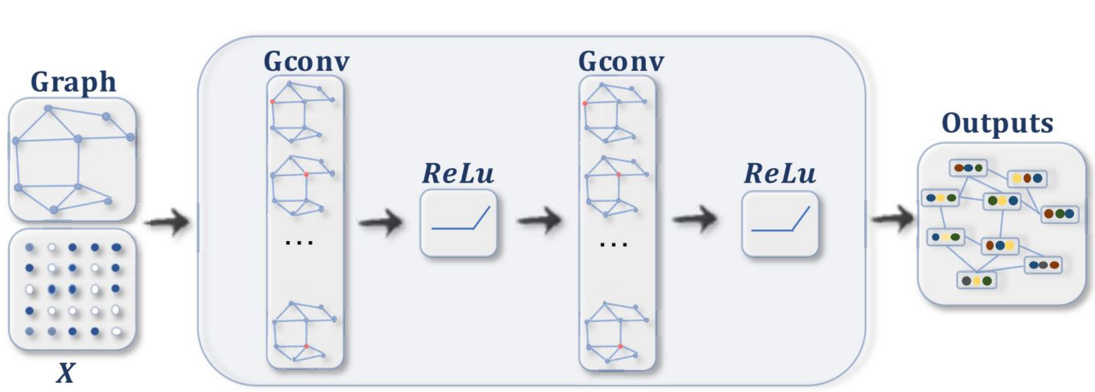
- Output (depending on different graph analytics tasks):
  - Node level: node regression and node classification
  - Edge level: edge classification and link prediction
  - Graph level: graph classification

立志成才报国裕民

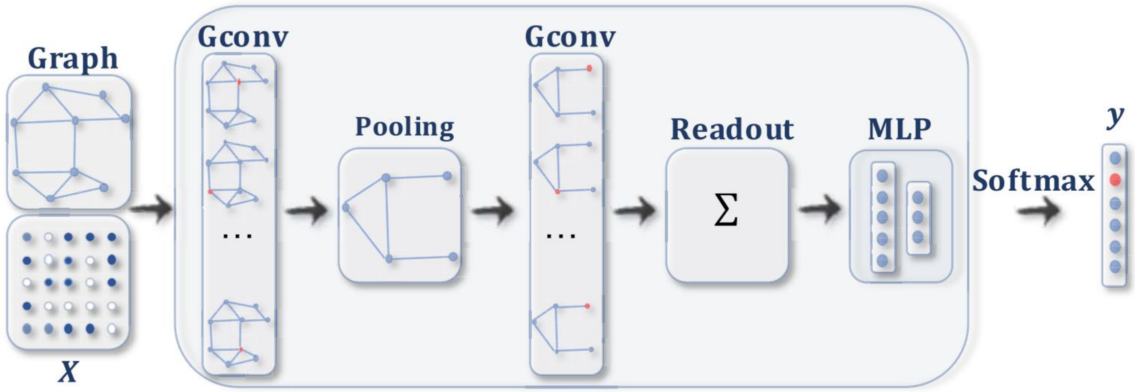
Recurrent GNNs (RecGNNs)

- **Aim**: To learn node representations with recurrent neural network architectures
- **Assumption**: A node in a graph constantly exchange information (or message) with its neighbors, until a stable equilibrium is reached
- **Impact**:
  - Most pioneering works on GNNs
  - RecGNNs inspired later research on ConvGNNs, which inherited the idea of **message passing**

Convolutional GNNs (ConvGNNs)



ConvGNN for node classification

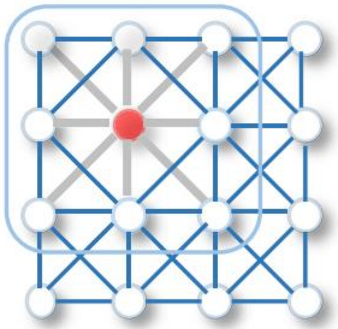


ConvGNN for graph classification

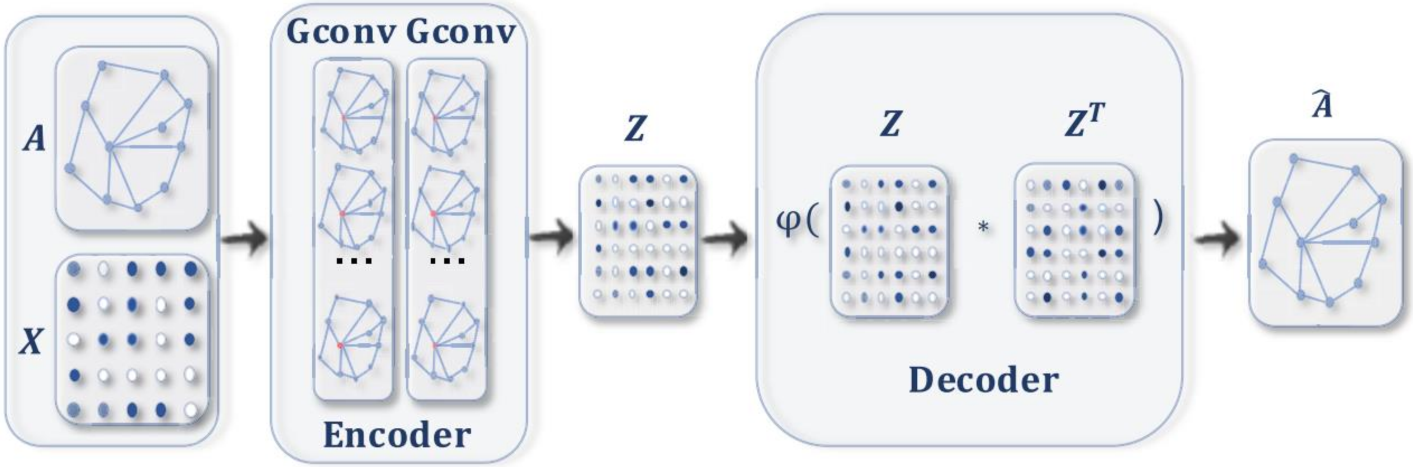
- **Main idea**: Generalize the operation of convolution from grid data to graph data
  - Generate a node v' s representation by aggregating its own features X_v and its neighbors' features X_u
  - Different from RecGNNs, it stacks multiple graph convolutional layers to extract high-level node representations

Two types of ConvGNNs

- **Spectral-based**: Define graph convolutions by introducing filters from the perspective of graph signal processing
  - Consider graph convolution as removing noises from graph signals
- **Spatial-based**: Define graph convolutions by information propagation (an idea inherited from RecGNNs)
  - Images can be considered as a special form of a graph with each pixel representing a node
  - Each pixel is directly connected to its nearby pixels
  - A filter is applied to a 3 x 3 patch by taking the weighted average of pixel values of the central node and its neighbors across each channel



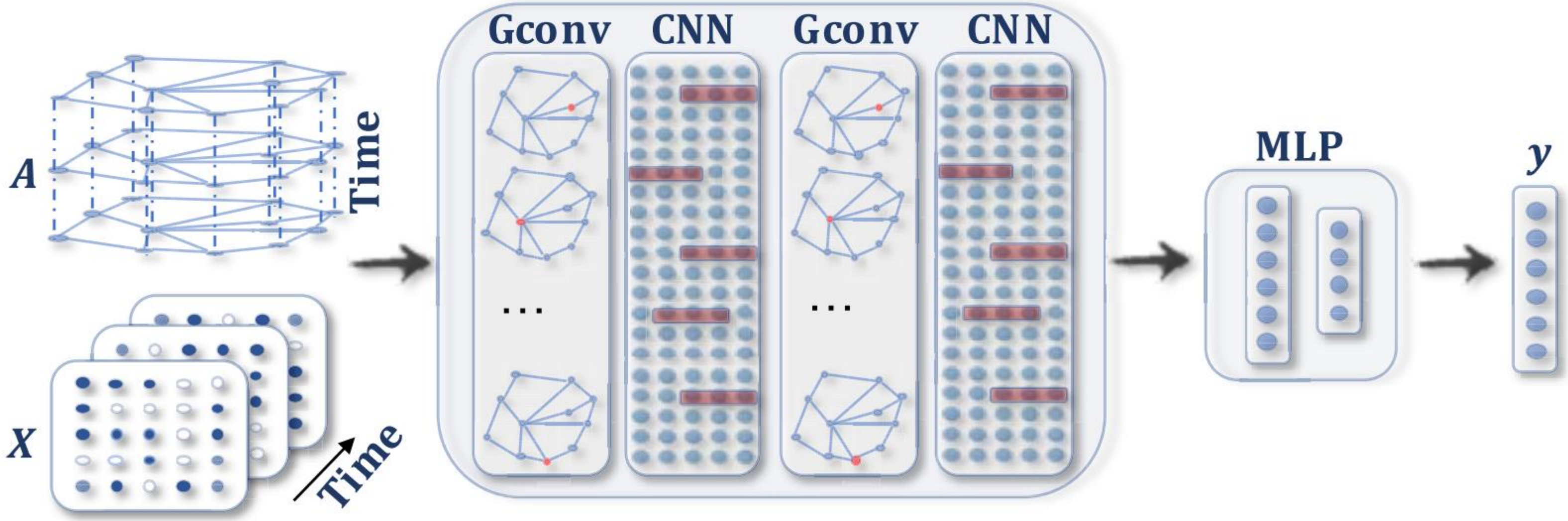
Graph Autoencoders (GAEs)



A GAE for network embedding

- **Main idea**: Encode nodes/graphs into a latent vector space, and reconstruct graph data from the encoded information
- **Applications**:
  - Network embedding: GAEs learn latent node representations through reconstructing graph structural information, e.g. adjacency matrix
  - Graph generation: GAEs learn graph generative distributions

Spatial-Temporal GNNs (STGNNs)



STGNN for spatial–temporal graph forecasting

- **Aim**: To model the dynamic node inputs while assuming interdependency between connected nodes
  - Capture spatial and temporal dependencies of a graph simultaneously
- **Applications**:
  - Learn hidden patterns from spatial-temporal graphs in applications, e.g. traffic speed forecasting, human action recognition

Some applications of GNNs

- **Computer Vision**: scene graph generation, point clouds classification, and human action recognition
- **Natural Language Processing (NLP)**: text classification (utilizing the interrelations of documents or words to infer document labels)
- **Recommender Systems**: take products and users as node, and formulate recommendation as a "link prediction" problem
- **Chemistry**: To study the graph structures of molecules / compounds
- Others:
    - Drug discovery
    - Brain science
    - Knowledge graph

Challenges and future directions of GNNs

上海科技大学 ShanghaiTech University

- Model depth: performance of ConvGNN drops with the number of layers
- How to balance scalability and graph integrity (or completeness)?
- How to handle heterogeneous graphs, which have different types of nodes and edges, or different forms of nodes and edge inputs, e.g. images and text?
- Dynamicity: Graphs are dynamic in that nodes or edges may appear or disappear, and inputs may change over time
  - STGNNs can partially address the dynamicity of graphs
  - Future: To find new graph convolutions that can adapt to the dynamicity of graphs

立志成才报国裕民

Summary

- In this lecture we learned basics of:
  - Generative Adversarial Networks (GANs)
  - Reinforcement learning (RL)
  - Graph Neural Networks (GNNs)
- Some are new (GANs), and some are old (RL), but they are all very popular recently
  - However, many challenges remain to be addressed
  - These techniques are under intensive development

- References:

  - Chapters 17 and 18 of Aurélien Géron's book "Hands-On Machine Learning with SciKit-Learn, Keras & TensorFlow" (2019)
  - Z. Wu et al. "A Comprehensive Survey on Graph Neural Networks", IEEE Transactions on Neural Networks and Learning Systems, 2020
  - Jie Zhou et al. "Graph Neural Networks: A Review of Methods and Applications", arXiv, 2019