Lecture 21: Deep Generative Models V: Sequential Generative Models, Deep Reinforcement Learning I

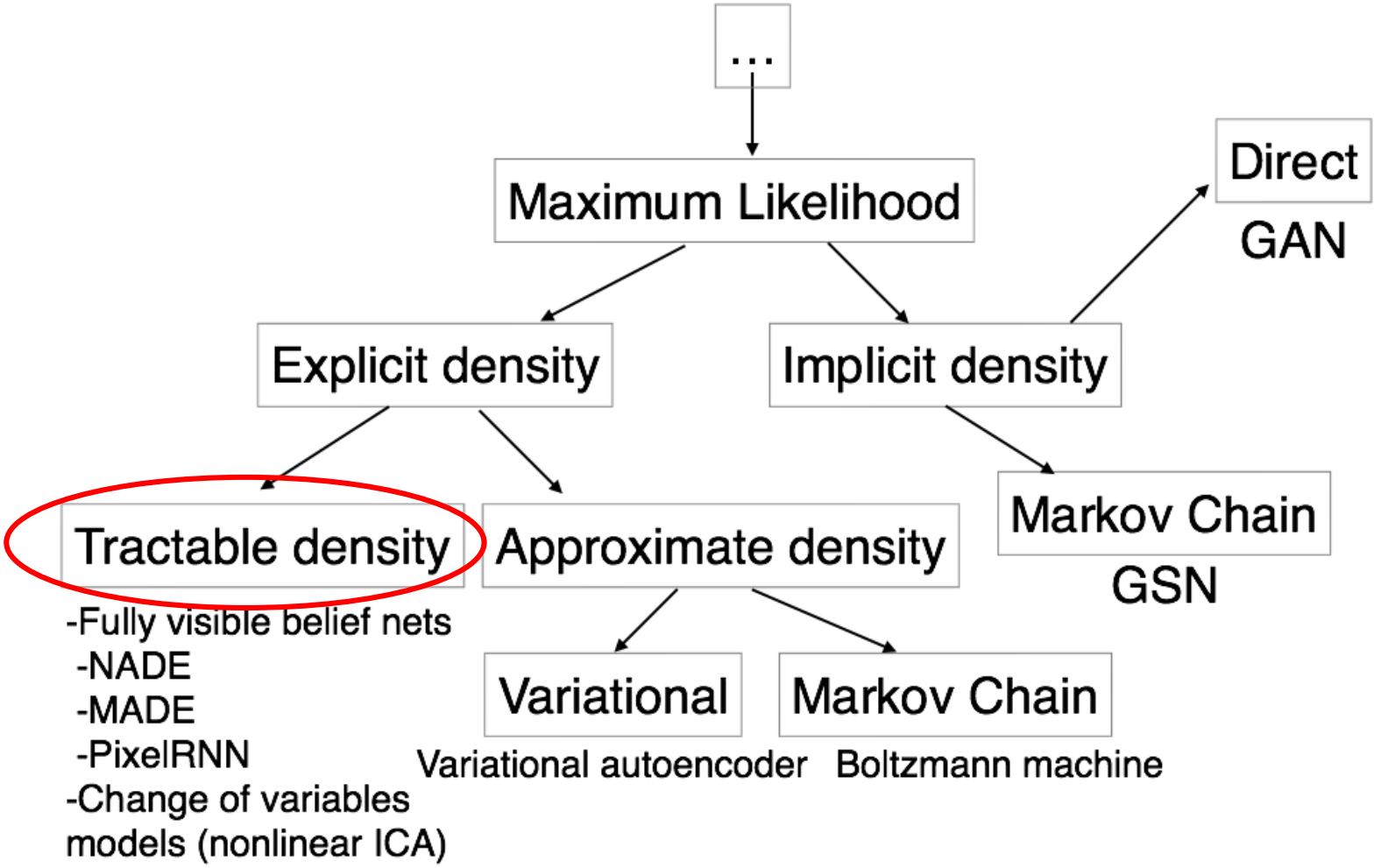
Xuming He
SIST, ShanghaiTech
Fall, 2019

Outline

- Autoregressive models
  - PixelRNN/CNN
- Deep Reinforcement Learning
  - Markov Decision Process

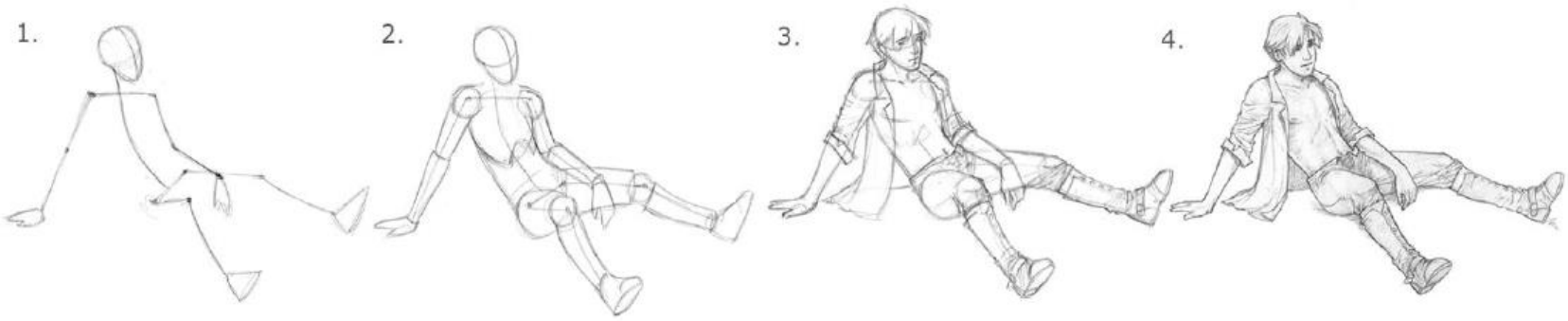
Acknowledgement: CMU, UofT, UCL, Stanford notes

Taxonomy of Generative Models



Intuition from Image Generation

- How we draw a picture?

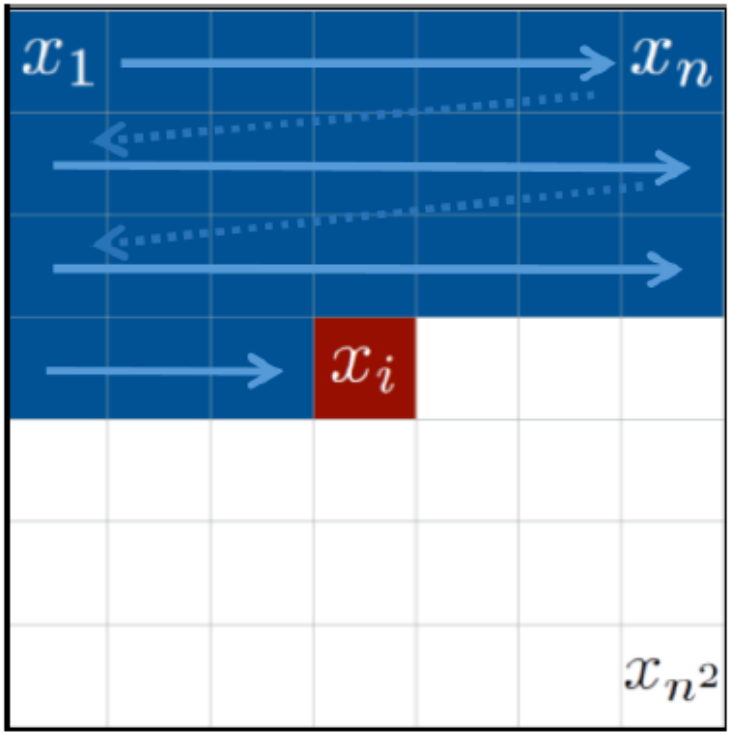


We pay **attention** on each subpart, we **iterate** in a **feedback** loop

Can we teach machines to do the same ?

Pixel-by-pixel Generation

- A simple way to iterate, employ feedback and capture pixel dependencies



Pixel-by-pixel Generation

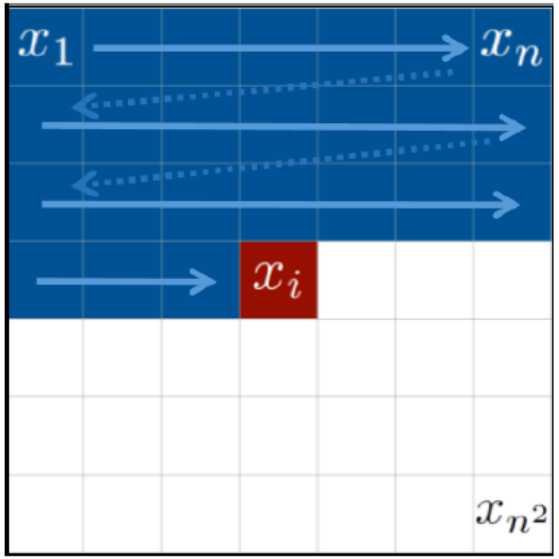
- Intuition

$$p(\mathbf{x}) = p(x_1, x_2, ..., x_{n^2})$$

Bayes Theorem:

$$p(\mathbf{x}) = \prod_{i=1}^{n^2} p(x_i | x_1, ..., x_{i-1})$$

A sequential model!



Fully Visible Belief Network

Explicit density model

Use chain rule to decompose likelihood of an image x into product of 1-d distributions:

$$p(x) = \prod_{i=1}^{n} p(x_i | x_1, \ldots, x_{i-1})$$

Likelihood of image x

Probability of i'th pixel value given all previous pixels

Then maximize likelihood of training data

Fully Visible Belief Network

Explicit density model

Use chain rule to decompose likelihood of an image x into product of 1-d distributions:

$$p(x) = \prod_{i=1}^{n} p(x_i | x_1, \ldots, x_{i-1})$$

Likelihood of image x

Probability of i'th pixel value given all previous pixels

Will need to define ordering of "previous pixels"

Complex distribution over pixel values => Express using a neural network!

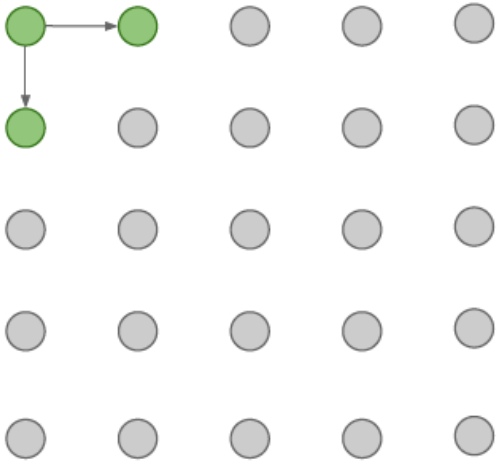
Then maximize likelihood of training data

PixelRNN

- Overview

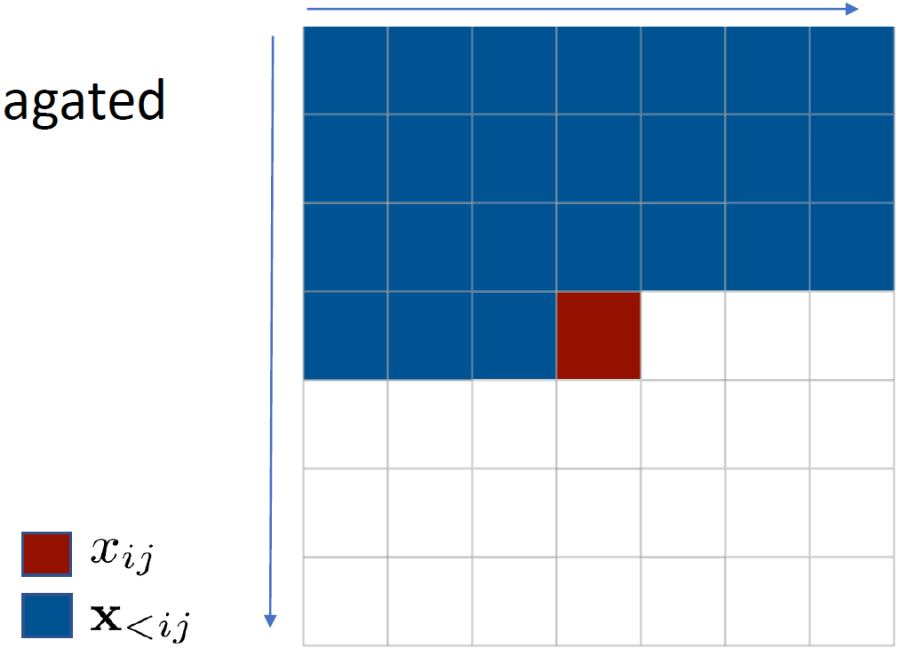
Generate image pixels starting from corner

Dependency on previous pixels modeled using an RNN (LSTM)



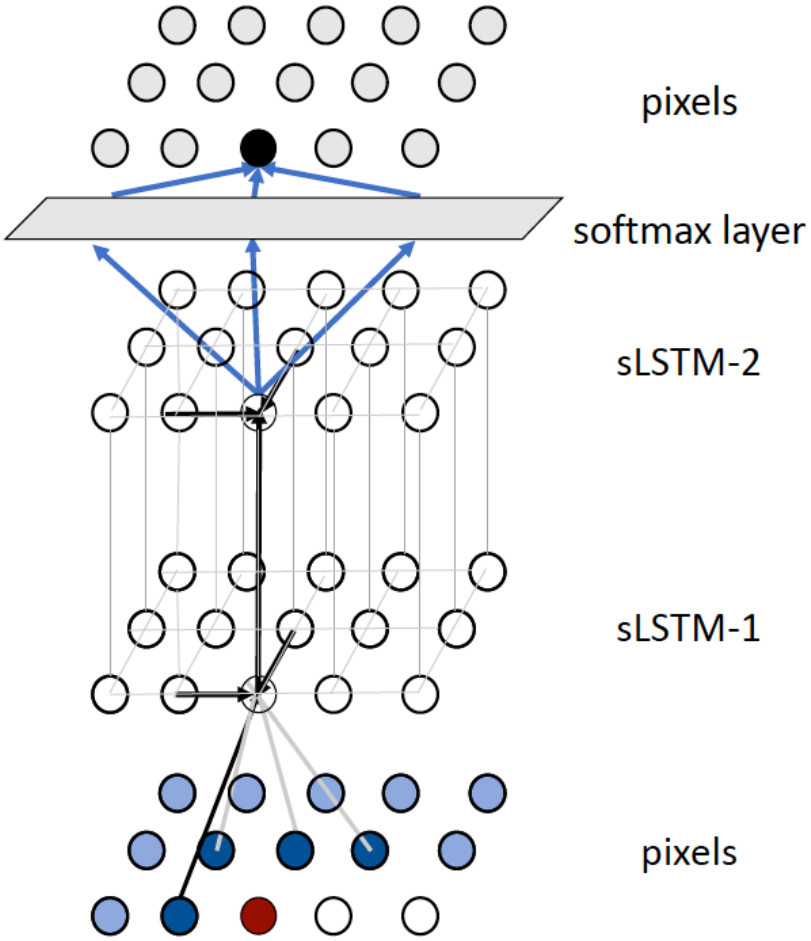
PixelRNN

- Question: Can we use plain-LSTM to generate images pixels by pixels?
- Ensure information is well propagated in two dimensions
- spatial LSTM (sLSTM)



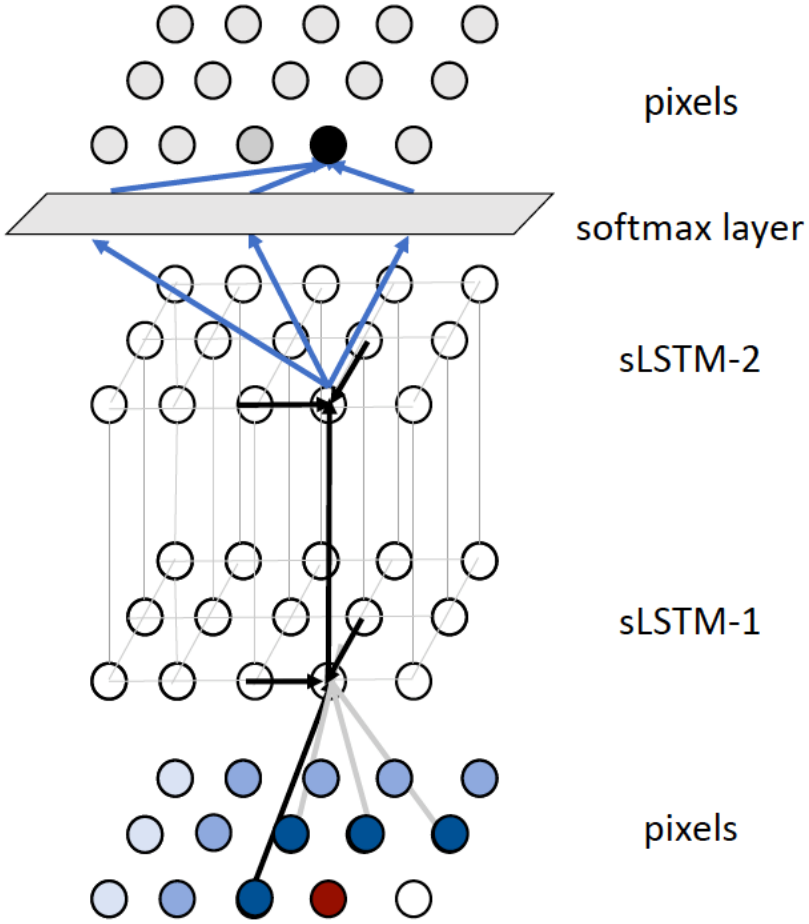
Spatial LSTM

- [Theis & Bethge, 2015]



Spatial LSTM

- [Theis & Bethge, 2015]



Modeling pixels

- Treat pixels as discrete variables
  - To estimate a pixel value, do classification in every channel (256 classes)
  - Implemented with a final softmax layer

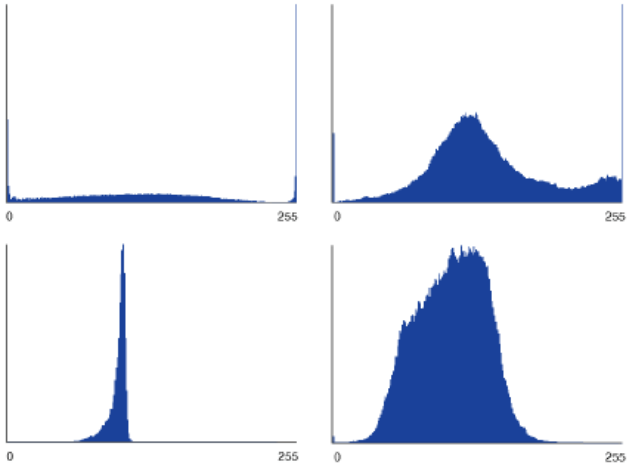
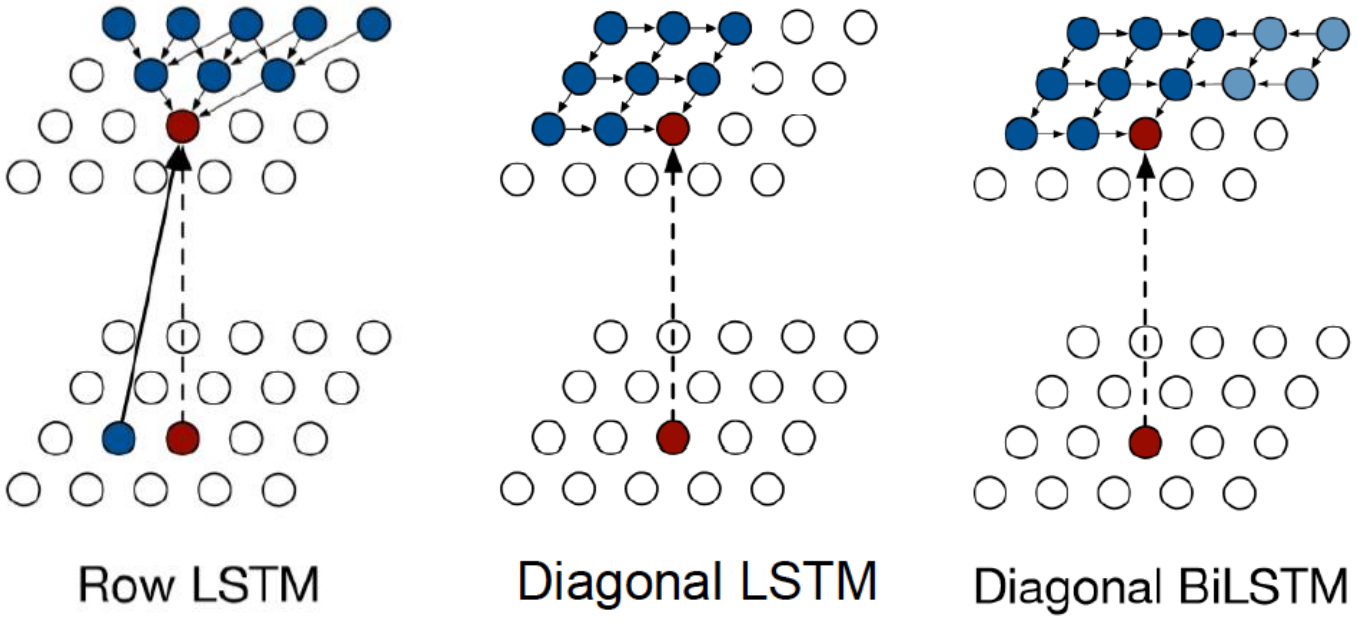


Figure: Example softmax outputs in the final layer, representing probability distribution over 256 classes.

PixelRNN

- Row LSTM and Diagonal LSTM



$$[\mathbf{o}_i, \mathbf{f}_i, \mathbf{i}_i, \mathbf{g}_i] = \sigma(\mathbf{K}^{ss} \circledast \mathbf{h}_{i-1} + \mathbf{K}^{is} \circledast \mathbf{x}_i)$$
$$\mathbf{c}_i = \mathbf{f}_i \odot \mathbf{c}_{i-1} + \mathbf{i}_i \odot \mathbf{g}_i$$
$$\mathbf{h}_i = \mathbf{o}_i \odot \tanh(\mathbf{c}_i)$$

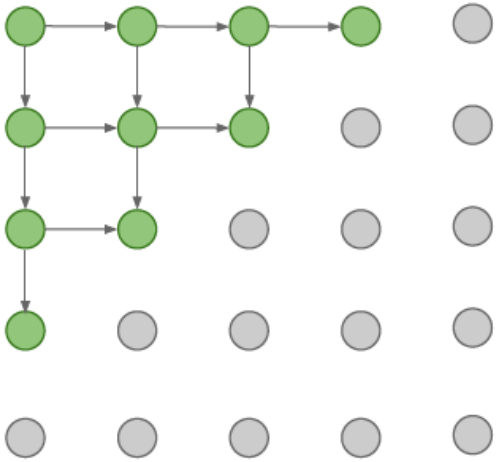
PixelRNN

- Drawbacks

Generate image pixels starting from corner

Dependency on previous pixels modeled using an RNN (LSTM)

Drawback: sequential generation is slow!

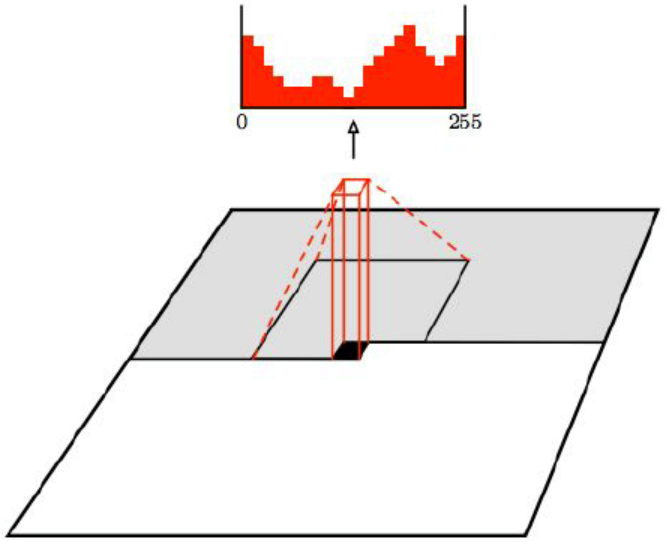


PixelCNN

- [van der Oord et al. 2016]

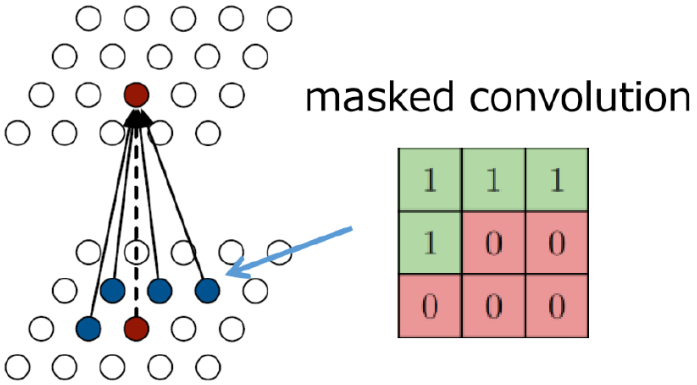
Still generate image pixels starting from corner

Dependency on previous pixels now modeled using a CNN over context region



PixelCNN

- [van der Oord et al. 2016]
  - 2D convolution on previous layer
  - Apply masks so a pixel does not see future pixels (in sequential order)



PixelCNN

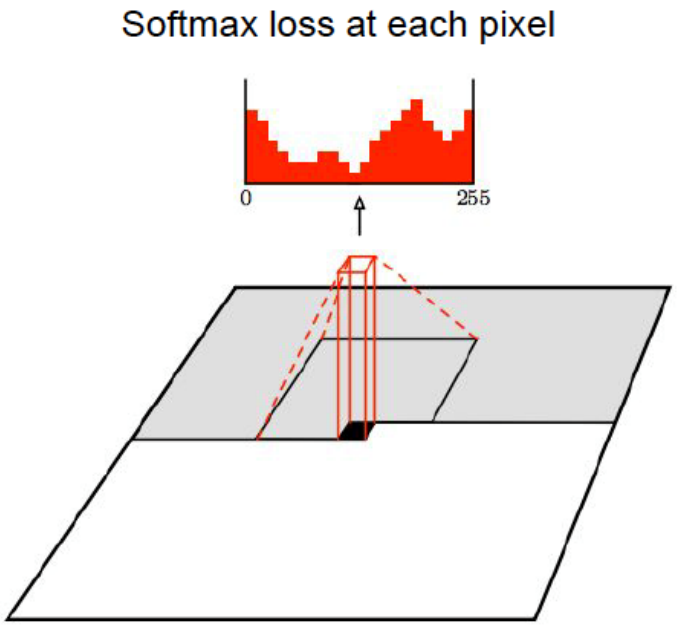
- [van der Oord et al. 2016]

Still generate image pixels starting from corner

Dependency on previous pixels now modeled using a CNN over context region

Training: maximize likelihood of training images

$$p(x) = \prod_{i=1}^{n} p(x_i | x_1, ..., x_{i-1})$$



PixelCNN

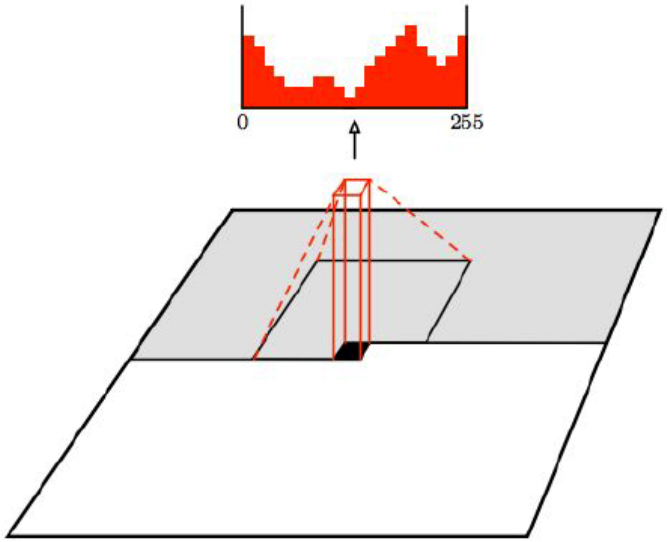
- [van der Oord et al. 2016]

Still generate image pixels starting from corner

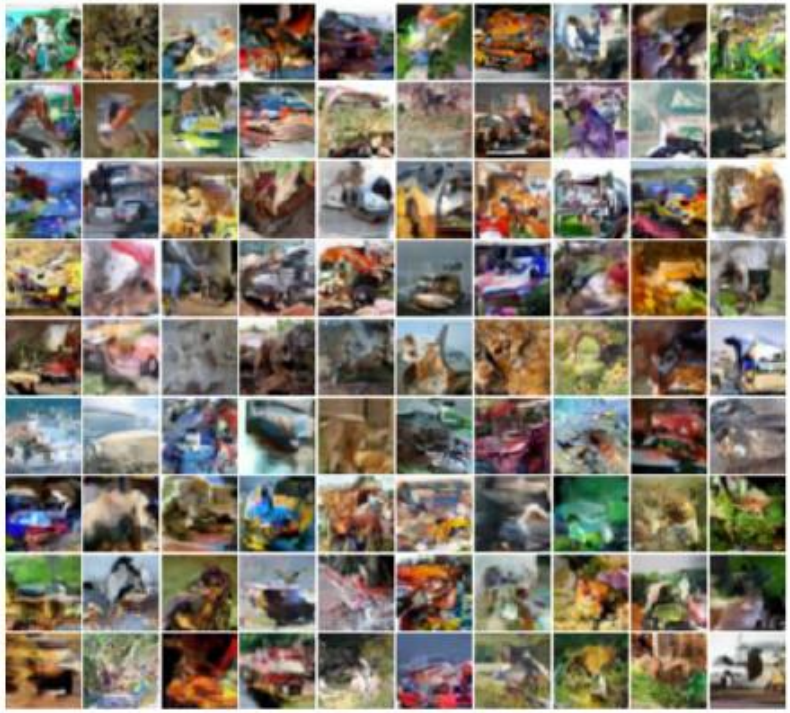
Dependency on previous pixels now modeled using a CNN over context region

Training is faster than PixelRNN
(can parallelize convolutions since context region values known from training images)

Generation must still proceed sequentially
=> still slow



Generated Samples



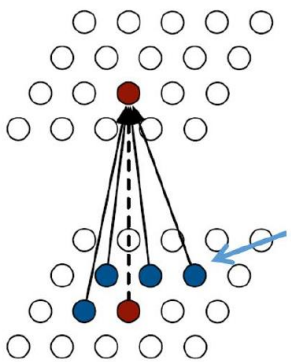
32x32 CIFAR-10

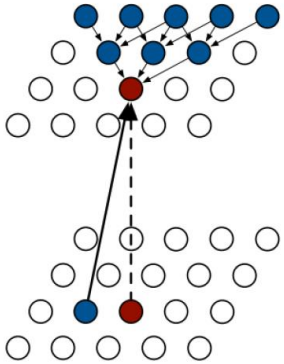
32x32 ImageNet

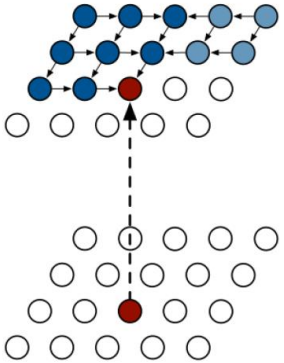
Summary of Autoregressive models

- Comparison

PixelCNN	PixelRNN – Row LSTM	PixelRNN – Diagonal BiLSTM
Full dependency field	Triangular receptive field	Full dependency field
Fastest	Slow	Slowest
Worst log-likelihood	-	Best log-likelihood







- Other recent improvements
  - Gated PixelCNN (Oord et al.)
    - Improve PixelCNN by removing blind spots and replacing ReLU units
  - PixelCNN++ (Salimans et al.)
    - Improve PixelCNN by optimization techniques
  - Video Pixel Networks (Kalchbrenner et al.)
    - Extend the work to 4 dimension

Summary

- Deep Generative Models

Autoregressive models (PixelRNNs, PixelCNNs)	GAN	VAE
• Simple and stable training process (e.g. softmax loss) • Best log likelihoods so far	• Sharpest images	• Easy to relate image with low-dimensional latent variables
• Inefficient during sampling • Don't easily provide simple low-dimensional codes for images	• Difficult to optimize due to unstable training dynamics	• Tends to have blurry outputs
		

Outline

- Autoregressive models
  - PixelRNN/CNN
- Deep Reinforcement Learning
  - Markov Decision Process

Acknowledgement: CMU, UofT, UCL, Stanford notes

Reinforcement Learning

- Supervised learning

Data: (x, y)
x is data, y is label

Goal: Learn a *function* to map x -> y

Examples: Classification, regression, object detection, semantic segmentation, image captioning, etc.

Cat

Classification

- Unsupervised learning

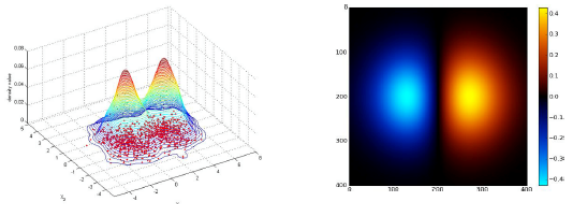
Data: x
Just data, no labels!

Goal: Learn some underlying hidden *structure* of the data

Examples: Clustering, dimensionality reduction, feature learning, density estimation, etc.

Figure copyright Ian Goodfellow, 2016. Reproduced with permission.

1-d density estimation



2-d density estimation

Reinforcement Learning

- A new class of problems: Reward-based
  - E.g. Autonomous driving

  - What is my next move?
    - Reaching the destination with minimum cost

Reinforcement Learning

- Common theme: control problems where
  - Your actions beget rewards
    - Win the 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

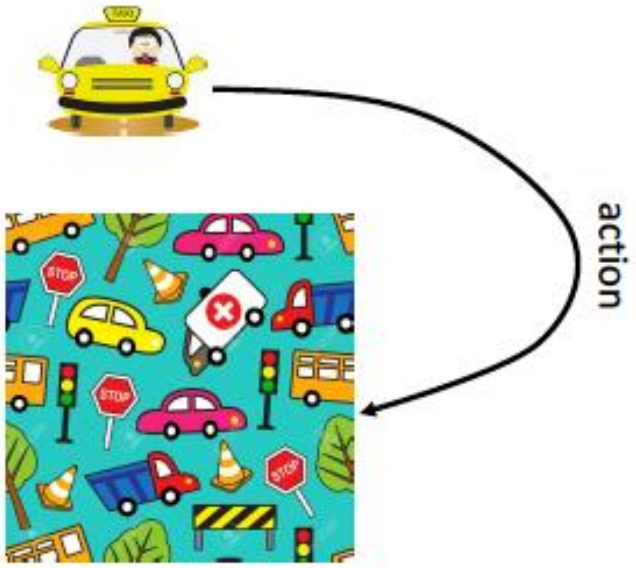
RL Problem Setting

- A cartoon of the world

- Agent operates in an environment
  - Agent may be you..
  - Environment is the game, the market, the road..

RL Problem Setting

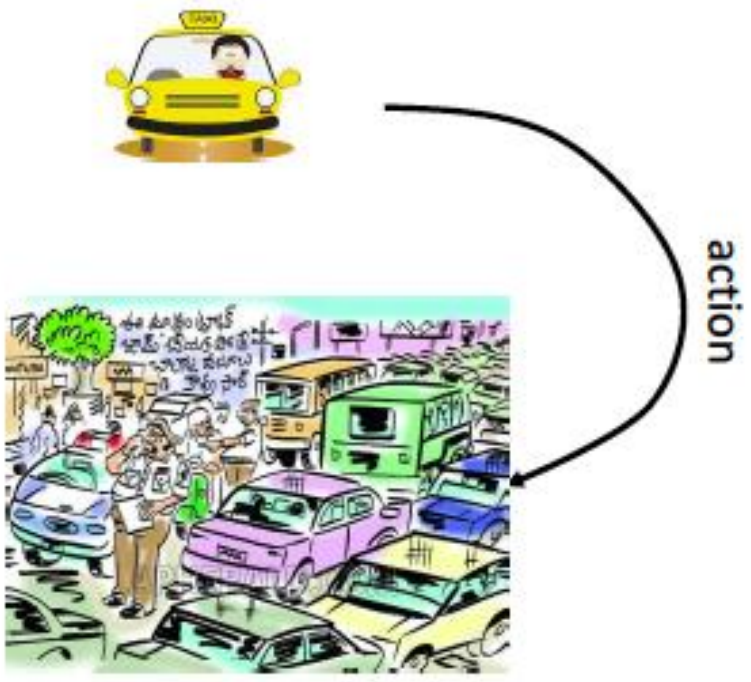
- A cartoon of the world



- Agent takes actions which affect the environment

RL Problem Setting

- A cartoon of the world



- Agent takes actions which affect the environment
- Which changes in a somewhat unpredictable way

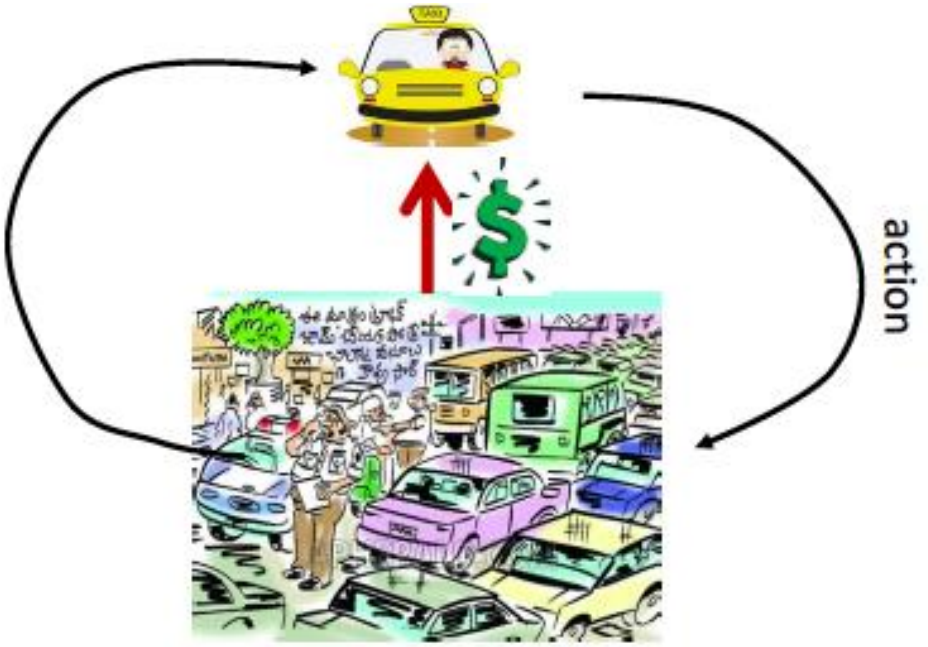
RL Problem Setting

- A cartoon of the world

- Agent takes actions which affect the environment
- Which changes in a somewhat unpredictable way
- Which affects the agent's situation

RL Problem Setting

- A cartoon of the world



- The agent also receives rewards..
- Which may be apparent immediately
- Or not apparent for a very long time

RL Problem Setting

- Problem to solve

- How must the agent behave to maximize its rewards?

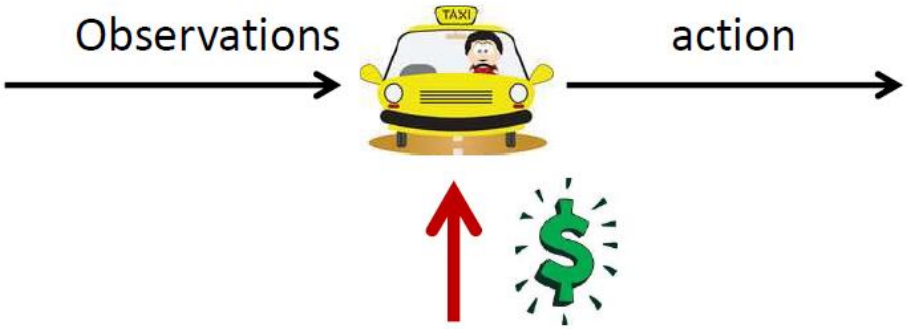
RL Problem Setting

- What the environment “experiences”

- Responds to some action by the agent
  - Changes in response
- Returns some reward (or punishment) to agent

RL Problem Setting

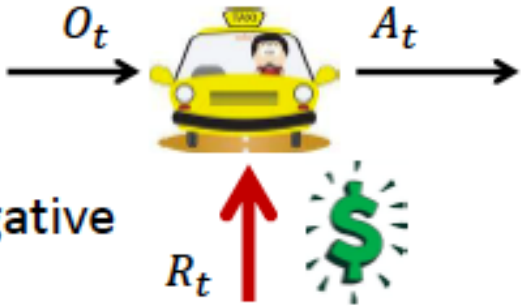
- What the agent sees



- The agent may observe something about its environment
  - Sensor readings, images of a game board, stock indices..
- The agent takes actions
- The agent receives rewards
- This is the agent's world; it must make sense of it
  - Again: Agent's objective is to take the actions that maximize rewards

RL Problem Formulation

- These can be cast of problems of *reinforcement* learning
- There is no supervisor, only a *reward* signal
  - Did you get home sooner
  - Did you win the game
  - Did you make money?
- i.e. nobody telling the agent "you did well"



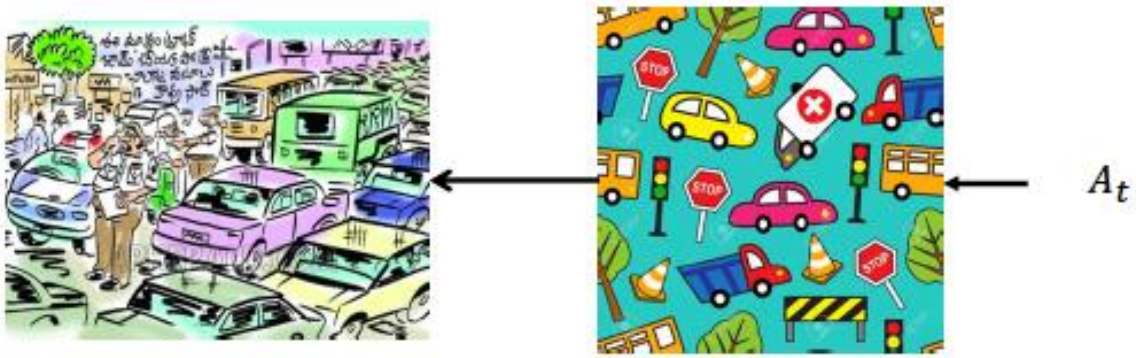
- *Reward is a scalar* – a single number, may be negative
  - Game was won/lost (binary)
  - Time taken to arrive
  - Amount of money made
- Reward may be delayed
  - Wait till the end of the game!
- Agents actions affect its current and future rewards
  - Must optimize actions for maximum reward

RL Problem Formulation

- The agent must model the environment process
  - Formulate its own model for the environment based on what it observes
  - The environment process is unknown to the agent (must be learned from experiences)
- The agent must formulate winning strategy based on the model of its environment
  - Seek for optimal actions (usually in sequence) that maximize the expected total reward
  - Or has to learn the strategy directly from experiences

RL Problem Formulation

- How to model the environment?



- The environment's state!
  - This is what will finally decide the rewards
- May be a complex combination of many things
- Generally assumed to be dynamic – keeps changing
- The agent's actions can affect the way in which it responds
  - But agent may not be able to observe all of it

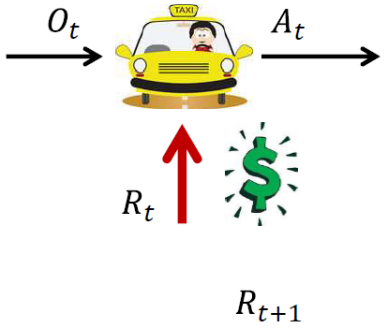
RL Problem Formulation

- Examples of state

- Fully captures the "status" of the system
  - E.g., in an automobile: [position, velocity, acceleration]
  - In traffic: the position, velocity, acceleration of *every* vehicle on the road
  - In Chess: the state of the board + whose turn it is next

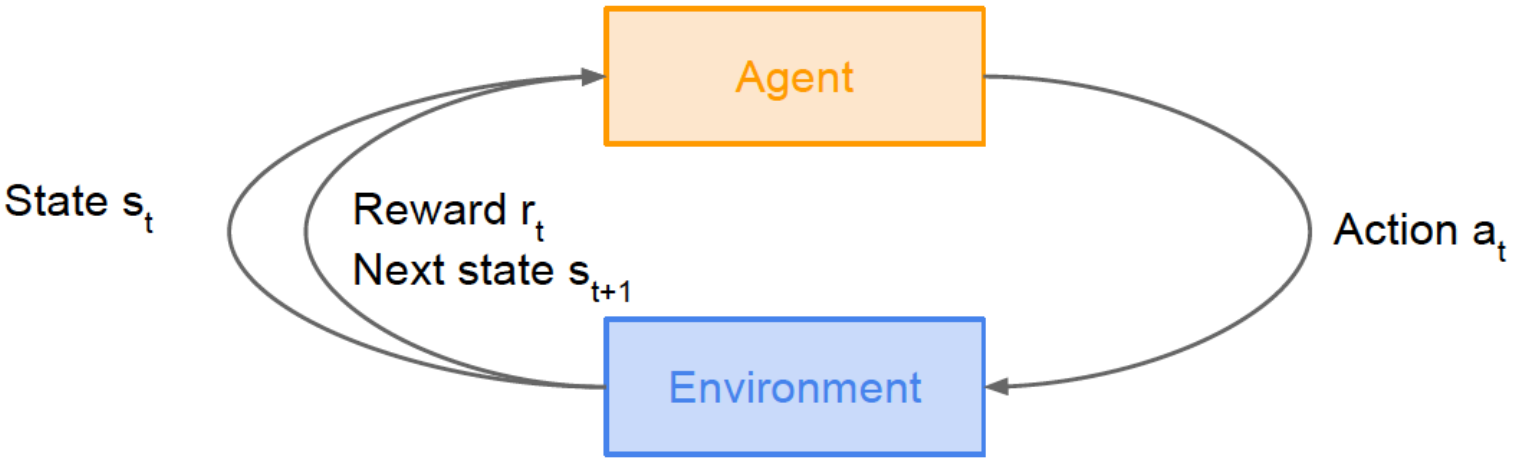
RL Problem Formulation

- How to formulate winning strategy?
  - At each time t the agent:
    - Makes an observation O_t of the environment
    - Receives a reward R_t
    - Performs an action A_t



  - $H_t = O_0, R_0, A_0, O_1, R_1, A_1, \ldots, O_t, R_t$
  - The total history at any time is the sequence of observations, rewards and actions
  - We need to model this sequence such that at any time t, the best $A_t|H_t$ can be chosen
    - The Strategy that maximizes total reward $R_0 + R_1 + \cdots + R_T$

Markov Decision Processes



- 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. state can be observed directly

MDP

- Formal definition

Definition

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, $\mathcal{P}^{a}_{ss'} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s, A_t = a\right]$
- $\mathcal{R}$ is a reward function, $\mathcal{R}^{a}_{s} = \mathbb{E}\left[R_{t+1} \mid S_t = s, A_t = a\right]$
- γ is a discount factor $\gamma \in [0, 1]$.

MDPs

- State Transition Matrix

For a Markov state s and successor state s', the *state transition probability* is defined by

$$\mathcal{P}_{ss'} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s\right]$$

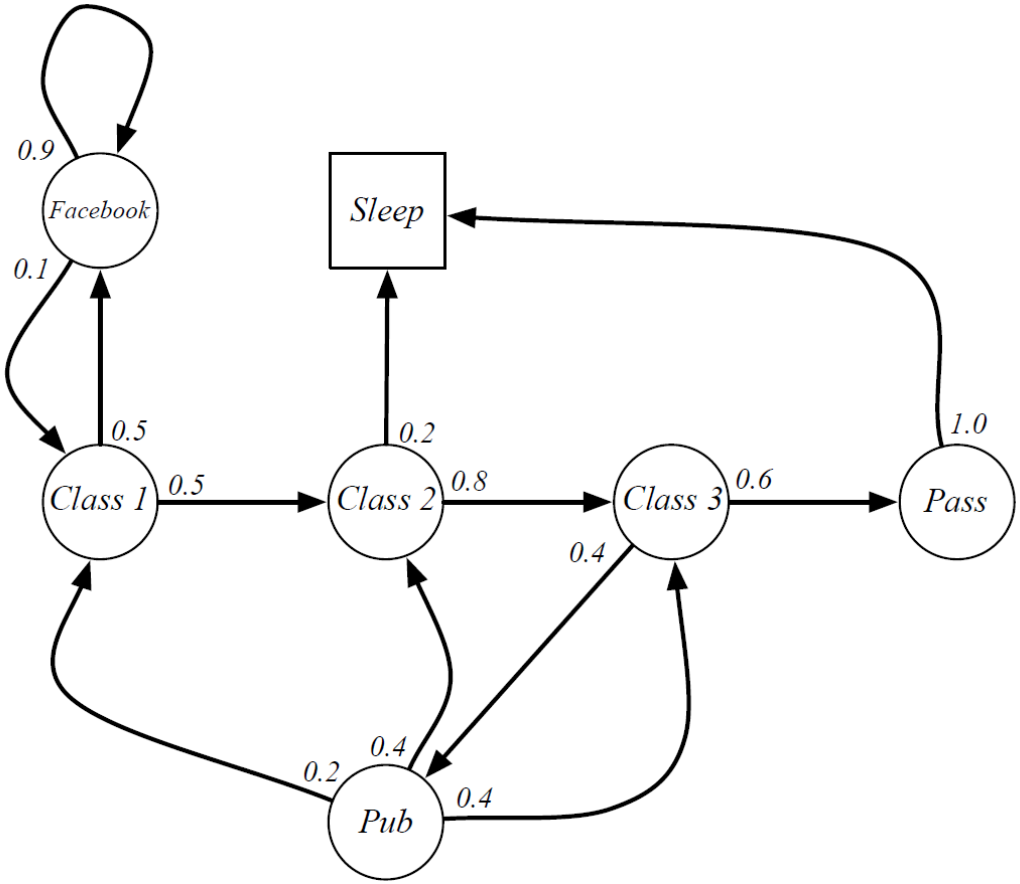
State transition matrix $\mathcal{P}$ defines transition probabilities from all states s to all successor states s',

$$\mathcal{P} = \begin{matrix} \\ \textit{from} \end{matrix} \begin{matrix} \textit{to} \\ \end{matrix} \begin{bmatrix} \mathcal{P}_{11} & \dots & \mathcal{P}_{1n} \\ \vdots & & \\ \mathcal{P}_{n1} & \dots & \mathcal{P}_{nn} \end{bmatrix}$$

where each row of the matrix sums to 1.

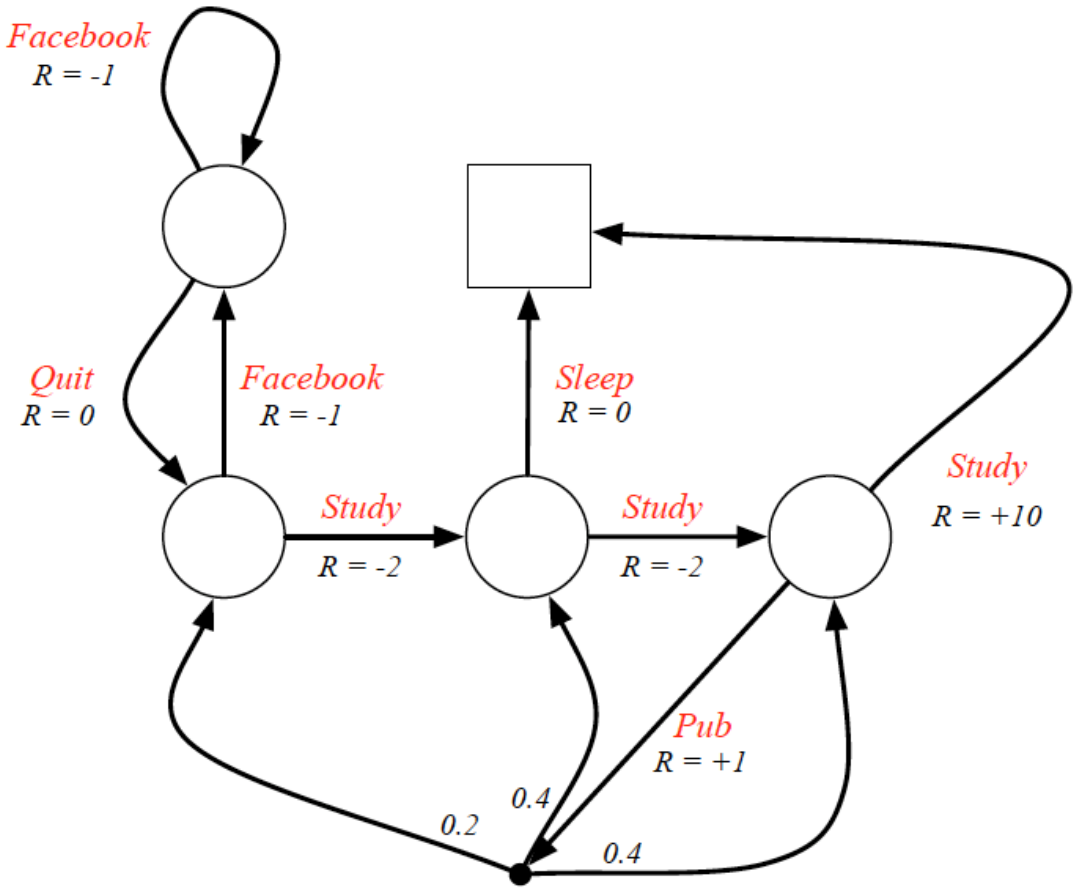
MDP

- Example


0.9
Facebook
0.1
Sleep
0.5
0.2
1.0
Class 1
0.5
Class 2
0.8
Class 3
0.6
Pass
0.4
0.4
0.2
Pub
0.4

MDP

- Example


Facebook
R = -1
Quit
R = 0
Facebook
R = -1
Sleep
R = 0
Study
R = -2
Study
R = -2
Study
R = +10
Pub
R = +1
0.4
0.2
0.4

Policy of MDP

- Agent's action strategy

Definition

A *policy* π is a distribution over actions given states,

$$\pi(a|s) = \mathbb{P}[A_t = a \mid S_t = s]$$

- A policy fully defines the behaviour of an agent
- MDP policies depend on the current state (not the history)
- i.e. Policies are *stationary* (time-independent), $A_t \sim \pi(\cdot|S_t), \forall t > 0$

Trajectory of MDP

- Observed instance of an MDP
  - initial state distribution $p(\mathbf{s}_0)$
  - policy $\pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t)$
  - transition distribution $p(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)$
  - reward function $r(\mathbf{s}_t, \mathbf{a}_t)$
- Finite horizon T (infinite case later)
  - Rollout, or trajectory $\tau = (\mathbf{s}_0, \mathbf{a}_0, \mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T)$
  - Probability of a rollout

$$p(\tau) = p(\mathbf{s}_0)\, \pi_\theta(\mathbf{a}_0 \mid \mathbf{s}_0)\, p(\mathbf{s}_1 \mid \mathbf{s}_0, \mathbf{a}_0) \cdots p(\mathbf{s}_T \mid \mathbf{s}_{T-1}, \mathbf{a}_{T-1})\, \pi_\theta(\mathbf{a}_T \mid \mathbf{s}_T)$$

  - Return for a rollout: $r(\tau) = \sum_{t=0}^{T} r(\mathbf{s}_t, \mathbf{a}_t)$

Problems in MDP

- Planning: given a **complete** MDP as input, compute policy with optimal expected return
  - Goal: maximize the expected return, $R = \mathbb{E}_{p(\tau)}[r(\tau)]$
  - 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

Summary

- Deep Reinforcement Learning
  - Markov Decision Process
- Roadmap
  - Q learning and DQN
  - Direct approach: Policy gradient method
  - Last lecture: Recent topics in deep learning