Random Guide

Temp subtitle

Author
Anthony Liang

February 3, 2021

Contents

Chapter 1

Introduction

This is a random collection of interesting concepts and ideas that I came across.

Chapter 2

Learning from demonstration

Chapter 3

Self-supervised Learning

Chapter 4

Meta-learning

Chapter 5

Embodied learning

Chapter 6

Statistics

6.1 Casual Inference

- Inferring the effects of any treatment/policy/intervention/etc.
  - effect of treatment on a disease
  - effect of social media on health
- Simpson's Paradox: mortality rate table
- Total population vs subgroup by conditions
- Correlation does not imply causation!
- Correlation: linear statistical dependence
- Association is the more correct term for statistical dependence
- It is possible to have large amounts of association with only *some* being casual. Some association and 0 causation is a case of "assocation is not casuation"
- e.g.Wearing shoes and waking up with a headache, common cause of drinking the night before
- This is a "confoundeer", this is a type of *confounding association*
- If association is causation, then causual inference could be solved using traditional statistics and ML
- Even with infinite amounts of data, we sometimes cannot compute casual quantities
- Identification of casual effects
- Intervention vs. observation. If we can intervene/experiment, identification becomes easy. Observational data is challenging because there is often confounding.

6.1.1 Potential Outcomes

- The *potential outcome* $Y(t)$ denotes what your outcome would be if you were to take treatment t

- Potential outcomes aren't always observed, they can be potentially observed
- The one that is actually observed depends on the treatment
- Individual treatment effect (ITE) for the ith individual

$$\tau_i \triangleq Y_i(1) - Y_i(0)$$

- Fundamental problem of causal inference: can't observe all potential outcomes for a given individual, we cannot observe both $Y(1)$ and $Y(0)$
- The outcomes that you can't observe are called *counterfactuals*
- Average treatment effect (ATE)

$$\tau \triangleq \mathbb{E}[Y_i(1) - Y_i(0)] = \mathbb{E}[Y(1) - Y(0)]$$

- Association difference is not the same as causal difference due to confounding

$$\mathbb{E}[Y|T = 1] - \mathbb{E}[Y|Y = 0] \neq \mathbb{E}[Y(1)] - \mathbb{E}[Y(0)]$$

6.1.2 Ignorability and Exchangability

- Ignorability - ignoring missing data, remove causal arrow from confounder to treatment
- Ignorability allows us to reduce ATE to associational difference
- Exchangability - treatment groups are exchangable such that if they were swapped, the new treatment group would observe the same outcome as the old treatment group
- Identifying a casual effect is to reduce causal expression to a statistical expression
- Conditional exchangability means if we condition on the covariate X, there is no longer any non-causal association between T and X.

Figure 6.1: Causal graphical models

(a) X is confounding the effect of T on Y (b) Conditioning on X leads to no confounding

- Adjustment formula: Given the assumptions of unconfoundedness, positivity, consistency, and no interference, we can identify the ATE:

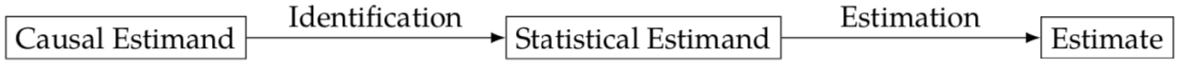
$$\mathbb{E}[Y(1) - Y(0)] = \mathbb{E}_X[\mathbb{E}[Y|T = 1, X] - \mathbb{E}[Y|T = 0, X]]$$

- Positivity-unconfoundedness tradeoff: conditioning on more covariates can lead to better chance of satisfying unconfoundedness, but it can lead to a higher chance of violating positivity
- No interference: $Y_i(t_1, \cdot, t_n = Y_i(t_i)$ otherwise my outcome is only a function of my own treatment
- Consistency: If treatment is T, then th observed outcome Y is the potential outcome under T. $T = t \rightarrow Y = Y(t)$
- Often we use a model (e.g. linear regression or some ML predictor) in place of conditional expectations $\mathbb{E}[Y|T = t, X = x]$, these models are known as *model-assisted estimators.*

6.1.3 Definitions

- Estimand: quantity that we want to estimate
- Estimate: is an approximation of some estimand
- Estimator: a function that maps a dataset to an estimate of an estimand
- Casual estimand: any estimand that contains a potential outcome
- Statistical estimand: any estimand that does not contain a potential outcome

Figure 6.2: Identification Flowchart



- Graph Terminology:
  - A **graph** is a collection of **nodes** and **edges** that connect the nodes
  - Undirected graphs: edges don't have any direction
  - Directed graphs: edges go from a *parent* node to a *child* node, parents of node X are $\text{pa}(X)$
  - Two nodes are *adjacent* if they're connected by an edge
  - A *path* is any sequence of adjacent nodes regardless of direction, vs. a *directed path*
  - X is an *ancestor* of Y, and Y is a *descendant* of X
  - Cycle
  - If there are no cycles in a graph, then it is a *directed acyclic graph* (DAG)
- Bayesian Networks
  - An intuitive way to model many variables together in a joint distribution is to only model local dependencies

- Local Markov Assumption: given its parents in the DAG, a node X is independent of its non-descendants
- Bayesian Network Factorization: given a probability P and a DAG G, P factorizes according to G if

$$P(x_1, \cdot, x_n) = \prod_i P(x_i | pa_i)$$

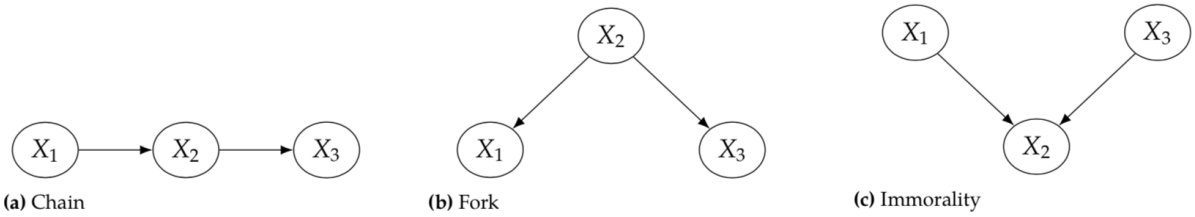
- Also known as the *chain rule for Bayesian networks*

- Minimal Assumption: also adds that adjacant nodes in the DAG are dependent, also equivalent to saying that we can't remove any more edges from the graph

6.1.4 Causal Graphs

- A variable X is said to be a cause of variable Y if Y can change in response to changes to X
- In a DAG, every parent is a direct cause of all of its children

Figure 6.3: Graph building blocks



(a) Chain **(b)** Fork **(c)** Immorality

- Two unconnected nodes are conditionally independent. $P(x_1, x_2) = P(x_1)P(x_2)$
- X_1 and X_3 are associated in chains and forks because they're commonly associated with X_2
- When we condition on X_2 for forks and chains, it blocks the flow of association because of the local Markov Assumption

Figure 6.4: Causal graphical models

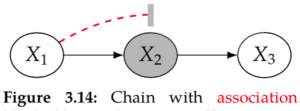


Figure 3.14: Chain with association blocked by conditioning on X_2.

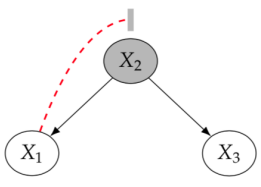


Figure 3.15: Fork with association blocked by conditioning on X_2.

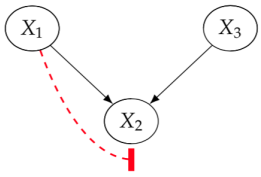


Figure 3.16: Immorality with association blocked by a collider.

- Colliders child of two parents that are not connected by an edge. In a collider, the parents are independent e.g. $X_1 \perp\!\!\!\perp X_3$, this is a *blocked path*

- When we condition on a collider (X_2), its parents X_1 and X_3 become dependent
- Conditioning on a collider can turn a *blocked path* to an *unblocked path*
- This phenomenon is known as *Berkson's paradox*
- Conditioning on descendants of a collider also induces association between parents of the collider
- In causal graphs, the edges have causal meaning

6.1.5 d-separation

- Two sets of nodes X and Y are d-separated by a set of nodes Z if all of the paths between X and Y are blocked by Z
- If all paths between X and Y are blocked, then they are *d-separated*
- D-separation implies conditional independence
- Global markov assumption: $X \perp\!\!\!\perp_G Y|Z \implies X \perp\!\!\!\perp_P Y|Z$
- Conditioning on $T = t$ means we restrict focus to subset of population to those who receive treatment t
- Intervention: take whole population and give everyone treatment t
- Denote intervention using $do(T = t)$ operator
- Interventional distribution: $P(Y|do(t))$ vs observational distribution: $P(Y)$
-

6.1.6 Structural Causal Models (SCMs)

- Structural equation: $B := f(A, U)$
- := gives us casual relation, A causes B
- $\mathcal{U}$ is some unobserved random variable and denotes all the relevant (noisy) background conditions that cause B

$$
\begin{aligned}
B &:= f_B(A, \mathcal{U}_B) \\
C &:= f_C(A, B, \mathcal{U}_C) \\
D &:= f_D(A, C, \mathcal{U}_D)
\end{aligned}
$$

- The variables that we write structural equations for are *endogenous* variables, these are the variables whose causal mechanisms we are modeling, $\{B, C, D\}$
- *Exogenous* variables are variables who don't have any parents in the causal graph, $\{A, \mathcal{U}_{\{B,C,D\}}\}$
- A structural casual model is a tuple of:
  - A set of endogenous variables V

  - A set of exogenous variables U
  - A set of functions f to generate each endogenous variable as a function of other variables
- If casual graph has no cycles and noise variables are independent then it is *Markovian*, if noise terms are dependent then it is *semi-Markovian*
- Intervention $do(T = t)$, replace structural equation for T with $T := t$, then we get the *interventional SCM* M_t
- This is by the modularity assumption

6.2 Bayesian Inference

6.3 Variational Inference

6.4 Expectation Maximization

- Parameters θ, evidence X
- Prior: probability of parameters, $p(\theta)$
- Likelihood: probability of evidence given parameters, $p(X|\theta)$
- Posterior: probability of the parametersgiven evidence, $p(\theta|X)$

$$p(\theta|x) = \frac{p(x|\theta)}{p(x)}p(\theta)$$
$$posterior = \frac{likelihood}{constant}prior$$

- Maximum a posteriori estimate (MAP): estimate of an unknown quantity, mode of a posterior distribution
- point estimate of unobservable quantity based on empirical data
- EM: iterative method to find maximum likelihood or maximum a posteriori (MAP) estimate of parameters in a statistical model
- alternate between Expectation step and Maximization step
- E-step: creates a function for expectation of log-likelihood evaluated using current estimates of parameter
- M-step: computes new parameters that maximize expected log-likelihood

6.5 Gaussian Processes

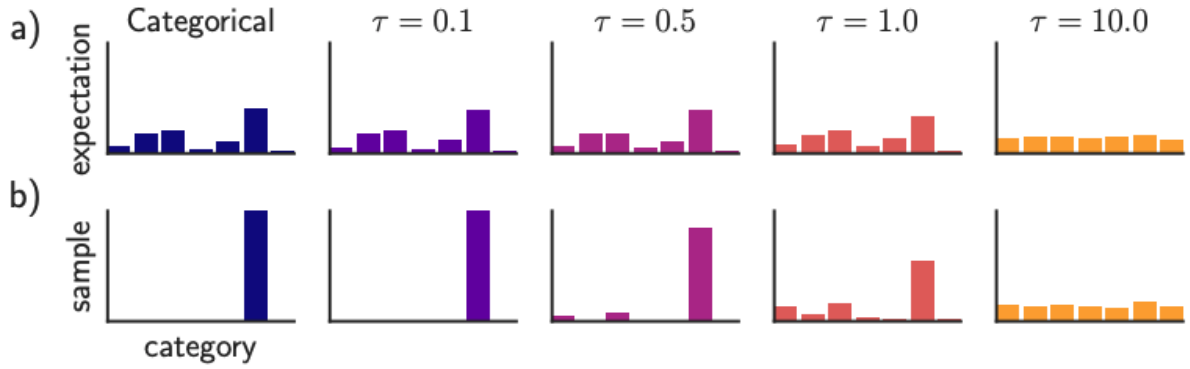
- Multivariate gaussian distributions are defined by a mean vector μ and covariance matrix Σ

- The diagonal of Σ consists of the variances σ_i^2 of the i-th random variable and the off diagonal elements describe correlation between i, j-th random variables
- Gaussian distributions are closed under conditioning and marginalization
- $X|Y \sim \mathcal{N}(\mu_X + \Sigma_{XY}\Sigma_{YY}^{-1}(Y - \mu_Y), \Sigma_{XX} - \Sigma_{XY}\Sigma_{YY}^{-1}\Sigma_{YX}$
- Nice visualizations of these operations
- Simple intuitive explanations
- Kernel, often called the *covariance function*. Kernel computes pairwise similarity between points $k(t, t')$
- Different kernels: rbf kernel, periodic, linear, etc
- GP is a non-parametric approach to regression meaning it finds a distribution over all the functions that are consistent with the observed data
- Starts with a prior distribution and updates this as points are observed
- Non-parametric: infinite parameters
- GP can derive posterior from prior and observations
- We want $p(f_*|x_*, x, f)$ where x is observed data, x_* is test data
- Can sample points from the posterior

6.6 Gumbel Softmax Trick

- Argmax is a non-differentiable function
- Wherever you are as long as you are not on the x1=x2 line, if you move an infinitesimal tiny bit in any direction the output of argmax doesn't change, thus the gradient of argmax is (0,0) almost everywhere
- Gumbel-Softmax is a continuous ditribution that can approximate samples from a categorical distribution
- $z = \text{one\_hot}(argmax_i[g_i + log(\pi_i)])$
- Gumbel-Softmax interpolates between a one-hot encoded categorical distributions and continuous categorical densities
- Low temperature = categorical random variable, high temperature = uniform distribution
- It is smooth for $\tau > 0$ and has well-defined gradient
- Tradeoff during training where smaller temperatures lead to samples closer to one-hot but variance of gradients is large and vice versa for high temperature
- Straight-trhough Gumbel-Softmax allows samples to be sparse, even when temperature is high

Figure 6.5: Gumbel-Softmax distribution



Chapter 7

Robotics

Chapter 8

Vision

8.1 Glossary

- Image classification: assign a class label to an image
- Object localization: draw a bounding box around one or more objects in an image
- Object detection: draw bounding box and assign a label
- Object segmentation
- Object recognition

8.2 Datasets

- CalTech-101
- VOC-2012
- ILSVRC-2013
- ImageNet

8.3 Convolutional Networks

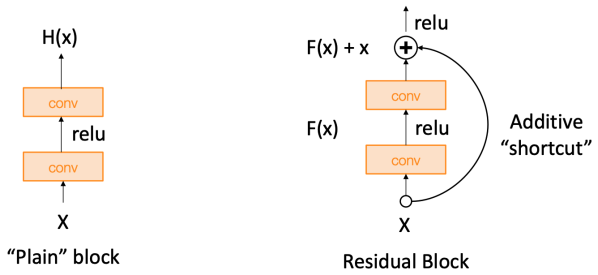
- Convolution layers
- Pooling layers
- Fully-connected layers
- Filters
- Stride
- BatchNorm
- Activation functions

8.4 CNN Architectures

- ImageNet Classification Challenge
- AlexNet
  - 227 x 227 inputs
  - 5 convolutional layers
  - Max pooling
  - 3 fully-connected layers
  - ReLU nonlinearities
  - 61 million parameters total
  - Most memory usage is in the early convolution layers
  - Nearly all the parameters are from the fully-connected layers
  - Most floating-point ops occur in convolution layer
- VGG
  - Enabled training deeper networks
  - Established standard network design patterns
  - Design rules:
    - All conv are 3x3 stride 1 pad 1
    - All max pool are 2x2 stride 2
    - After pool, double the # of channels
  - Notes: 2 3x3 conv has the same receptive field as a single 5x5 conv, but fewer parameters and less computation!
  - VGG-16 total: 138 million parameters
- GoogleLeNet
  - Many innovations for efficiency: reduce parameters, memory usage, and computation
  - Stem network: aggressively downsample input from the start
  - Inception module: local unit with parallel branches, use 1x1 bottleneck to reduce channel before conv
  - No FC layers at the end
  - Uses global average pooling to collapse spatial dimensions and one linear layer
  - Hack to overcome deepness of network, use "auxillary classifier" at intermediate points in network
  - This work was before BatchNorm!

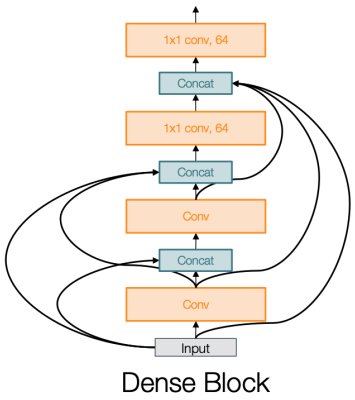
- ResNet
  - 152 layers!
  - Change network so learning identity function with extra layers is easy
  - Able to train very deep networks

Figure 8.1: Residual Block



- Densely Connected Neural Networks
  - Each layer is connected to every other layer in feedforward

Figure 8.2: Dense Block



  -

- Tiny networks
  - MobileNet
  - ShuffleNet
- Neural Architecture Search
  - Automate searching for efficient network architectures
  - A network outputs network architectures
  - The controller generates child networks and trains them
  - After training a batch of child networks, make gradient step on controller network
  - VERY EXPENSIVE! Trained 800 GPUs for 28 days

8.5 Recurrent Networks

- Recurrent Neural Networks
  - useful for processing sequences with inputs
  - one-to-one, one-to-many (image captioning), many-to-one (video classification), many-to-many (machine translation)
  - Key idea is that RNN have an "internal state" that is updated as a sequence is processed

$$h_t = f_W(h_{t-1}, x_t)$$

x_t = input vector
h_t = new state
f_W = function with parameter W

  - Vanilla RNN

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$$
$$y_t = W_{hy}h_t + b_y$$

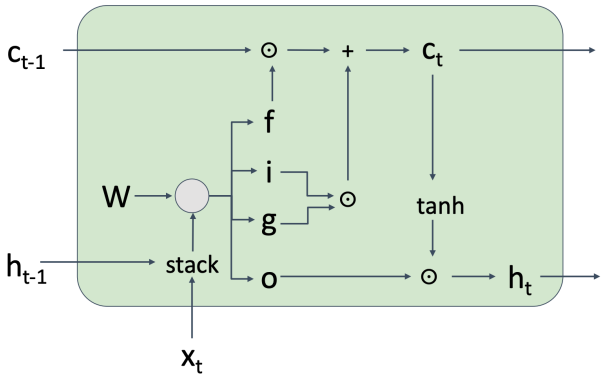
  - Reuse weight matrix at every timestep
  - Backpropagation through time
  - RNN forwards the entire sequence to compute loss and then backward through the entire sequence to compute gradients
  - Language modeling - generating Shakespeare poems, generating code, etc
  - Image captioning - first run CNN on image and feed output from FC-4096 into the RNN along with captions
  - Backprop results in exploding / vanishing gradients
  - For exploding gradients, we can use gradient clipping to scale the gradient if norm is too big
  - For vanishing gradients, we need to change the RNN architecture
- Long-short Term Memory

$$i_t = \sigma(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$$
$$f_t = \sigma(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$$
$$o_t = \sigma(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$$
$$g_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$
$$h_t = o_t \odot \tanh(c_t)$$

i = input gate: whether to write to cell
f = forget gate: whether to erase cell
o = output gate: how much to reveal cell
g = gate gate (?): how much to write in a cell

- Backpropagation from c_t to c_{t-1} is only elementwise multiplication by f, does not depend on matrix multiply by W

Figure 8.3: LSTM Cell



- Gated Recurrent Units
  -

8.6 RCNN and variants

- [1]
- Region Proposal
  - Generate and extract category independent region proposals, candidate bounding boxes
  - Use CV technique called "selective search"
  - AlexNet
- Feature Extractor
  - Extract feature from each candidate region
- Classifier
  - Classify feature as one of the know classes

8.6.1 Other popular architectures

8.7 Detection

8.7.1 Single-stage detectors

8.7.2 Two-stage detectors

8.8 Segmentation

8.8.1 Semantic segmentation

8.8.2 Instance segmentation

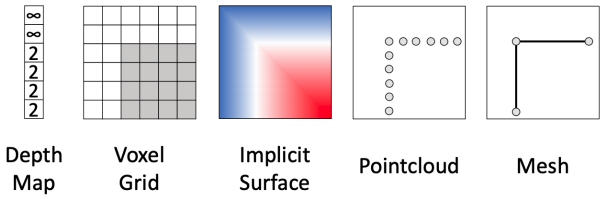
8.8.3 Keypoint estimation

8.9 Domain Adaptation

8.10 3D

8.10.1 Representations

Figure 8.4: 3D shape representations



- Depth Map
  - Distance from camera to object into the world at each pixel
  - Often combined with RGB image = RGB-D image (2.5D)
  - Can't capture occuluded areas
  - e.g. Microsoft Kinect and new IPhone-12 sensors
  - Task: predicting depth map from RGB image
  - Scale / depth ambiguity
- Surface Normals
  - Normal vector to object in world at pixel
  - Useful for robotics applications such as grasping objects
  - Can get ground-truth surface normal using synthetic data or multiview 3D reconstruction
  - Predict surface normal representation, loss: penalizes the angle between vectors

- Voxel Grid
  - A shape is a V x V x V grid of binary occupancies, any type of data can be stored in a cell of voxel representation
  - Like segmentation mask in 3D rather than 2D
  - Need high spatial resolution to capture fine-grain objects
  - Scaling to high resolutions is nontrivial, computationally expensive
  - Process these with 3D convolution
  - Generating voxel shapes using 3D convolution, flatten 2D features and convert into 3D feature
  - Predict per voxel and take loss per voxel
  - Memory usage scales cubically, not tractable
  - Oct-Trees for scaling voxels with heterogenous resolutions
- Implicit Surfaces
  - Learn function to classify arbitrary 3D points as inside / outside the shape or predict some sort of data
  - The surface of the 3D object is the level set
  - $o : \mathbb{R}^3 \rightarrow \{0, 1\}$
- Pointclouds
  - Represent shape as set of points in 3D space
  - Can represent fine structures without huge # of points
  - Requires new architectures / losses
  - Don't explicitly represent the surface! Need some postprocessing to get a mesh representation
  - PointNet: run MLP on each point in the cloud to generate per-point feature vector $\rightarrow$ max pooling then a fully-connected vector
- Triangle Mesh
  - Commonly used in computer graphics
  - Represent 3D shape as a set of triangles
  - Adaptive: can reprsent flat surfaces very efficiently, allocate more faces with fine detail
  - Can interpolate data on vertices over whole surfaces like RGB color, normal vector
  - Pixel2Mesh: single RGB image $\rightarrow$ triangle mesh

Chapter 9

Generative Models

9.1 Definitions

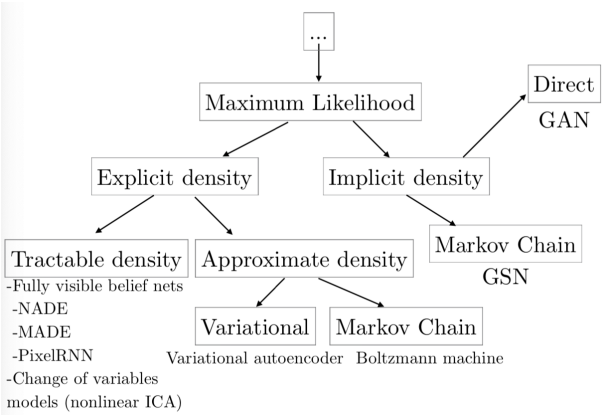
- Generative models: any model that takes in a training set, samples drawn from distribution p_{data} and learns to represent an estimate of that distribution somehow, the result is a distribution p_{model}

9.2 Generative Adversarial Networks

9.2.1 Why GANs?

- High-dimensional probability distributions are important
- Can be incorporated into reinforcement learning:
  - simulate possible futures by learning a conditional distribution over future states
  - enable learning in an imaginary environment
  - guide exploration by keeping tracking of how often different states have been visited
  - inverse reinforcement learning
- Provide predictions on inputs that are missing data (e.g. semi-supervised learning)
- Enable machine leanring to work with multi-modal outputs
- Many tasks require realistic generation of samples from some distributions
  - Single image super-resolution
  - Creating art
  - Image-to-image translation
-
- Benefits

Figure 9.1: Taxonomy of deep generative models



  - Can generate samples in parallel
  - Design of generator has few restrictions
  - No Markov Chains are needed
  - No variational bound is needed
  - Produces better samples than other methods

9.2.2 How do GANs work?

- Game between two players: the generator and the discriminator
- Generator (G)
  - creates samples that are intended to come from the same distribution as the training data
  - typically implemented as a deep neural network
  - inputs z are sampled from a prior over latent variables and generator $G(z)$ creates a new fake sample
  - different formulations of loss function
    1. minimax, zero sum game
    2. $J^{(G)} = -J^{(D)}$
    3. amenable to theoretical analysis
    4. heuristic
    5. maximum-likelihood game
- Discriminator (D)
  - examines samples to determine whether they are real or fake
  - output probability that input sample x is real
  - goal is for $D(x)$ to be near 1

  - also wanted to make $D(G(z))$ approach 0
  - cost function:

$$J^{(D)}(\theta^{(D)}, \theta^{(G)}) = -\frac{1}{2}\mathbb{E}_{x\sim p_{data}}[log(D(x))] - \frac{1}{2}\mathbb{E}_z[log(1 - D(G(z)))]$$

  - this is the cross entropy loss, pretty standard across all discriminator loss functions
  - classifier is trained on two minibatches of data: one from the dataset (label = 1) and another from the generator (label = 0)

- The solution to a game is the Nash equilibrium
- tmp

Chapter 10

Reinforcement Learning

10.1 Definitions

- Return (G_t) is the total discounted reward

$$G_t = R_{t+1} + \gamma R_{t+2} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

- Value function: expected return starting from state s

$$v(s) = \mathbb{E}[G_t | S_t = s]$$

- State value function: expected return starting from state s and following policy π

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t | S_t = s] \\ &= \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s,a) \end{aligned}$$

- Action value function: expected return starting from state s, taking action a and following policy π

$$\begin{aligned} q_\pi(s,a) &= \mathbb{E}_\pi[G_t | S_t = s, A_t = a] \\ &= \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s') \end{aligned}$$

- Policy: distribution over actions given states

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

- Optimal state-value function : maximum value function over all polices

$$v_*(s) = \max_\pi v_\pi(s)$$

- Optimal action-value function is maximum action-value function over all policies

$$q_*(s,a) = \max_\pi q_\pi(s,a)$$

- Off-policy: Q-values are updated using the Q-value of the next state s' and the greedy action a'. It estimates the return for state-action pairs assuming a greedy policy were following.
- On-policy: Q-values are updated using the Q-value of the next state s' and the *current policy's* action a'. It estimates the return for state-action pairs assuming the current policy continues to be followed.
- Online learning: alternate between optimizing a policy and using that policy to collect more data

10.2 Markov Decision Processes

- describes an environment for reinforcement learning
- environment is fully observable
- the future is independent of the past given the present (e.g. $P[S_{t+1}|S_t] = P[S_{t+1}|S_1, ..., S_t]$)
- a Markov Process is a tuple $\langle \mathcal{S}, \mathcal{P} \rangle$
  - $\mathcal{S}$ is a set of states
  - $\mathcal{P}$ is a state transition probability matrix
- a Markov reward process is a tuple $\langle \mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma \rangle$
  - $\mathcal{S}$ is a finite set of states
  - $\mathcal{P}$ is a state transition probability matrix
  - $\mathcal{R}$ is a reward function
  - γ is a discount factor, $\gamma \in [0, 1]$
- a Markov decision process (MDP) is a Markov reward process with decisions, $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$
  - $\mathcal{S}$ is a finite set of states
  - $\mathcal{A}$ is a finite set of actions
  - $\mathcal{P}$ is a state transition probability matrix, $\mathcal{P}^a_{ss'} = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$
  - $\mathcal{R}$ is a reward function, $\mathcal{R}^a_s = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$
  - γ is a discount factor, $\gamma \in [0, 1]$
- For any MDP, there exists an optimal policy π_* better than or equal to all other policies, $\pi_* \geq \pi, \forall \pi$
- If we know $q_*(s, a)$, we immediately have the optimal policy.
- Bellman Optimality Equation is non-linear
- Many iterative solution methods:
  - Value Iteration
  - Policy Iteration

- Q-learning
- Sarsa

10.3 Value-iteration

- it is a method of computing an **optimal MDP policy** and its value

10.4 Q-learning

- Off-policy Temporal Difference Control
- One step Q-learning:
- $$q(s_t, a_t) = q(s_t, a_t) + \alpha[r_{t+1} + \gamma \text{max}_a q(s_{t+1}, a) - q(s_t, a_t)]$$

Algorithm 1: Q-learning

Initialize $q(s, a) \forall s \in \mathcal{S}, \forall s \in \mathcal{A}, q(\text{terminal-state}) = 0$;
for *each episode* **do**
  Initialize s;
  for *each step in episode* **do**
    Choose a from s using policy derived from q (e.g. ϵ-greedy);
    Take action a and observe r, s';
    $q(s, a) = q(s, a) + \alpha[r + \gamma \text{max}_a q(s', a) - q(s, a)]$;
    $s = s'$;
  end
end

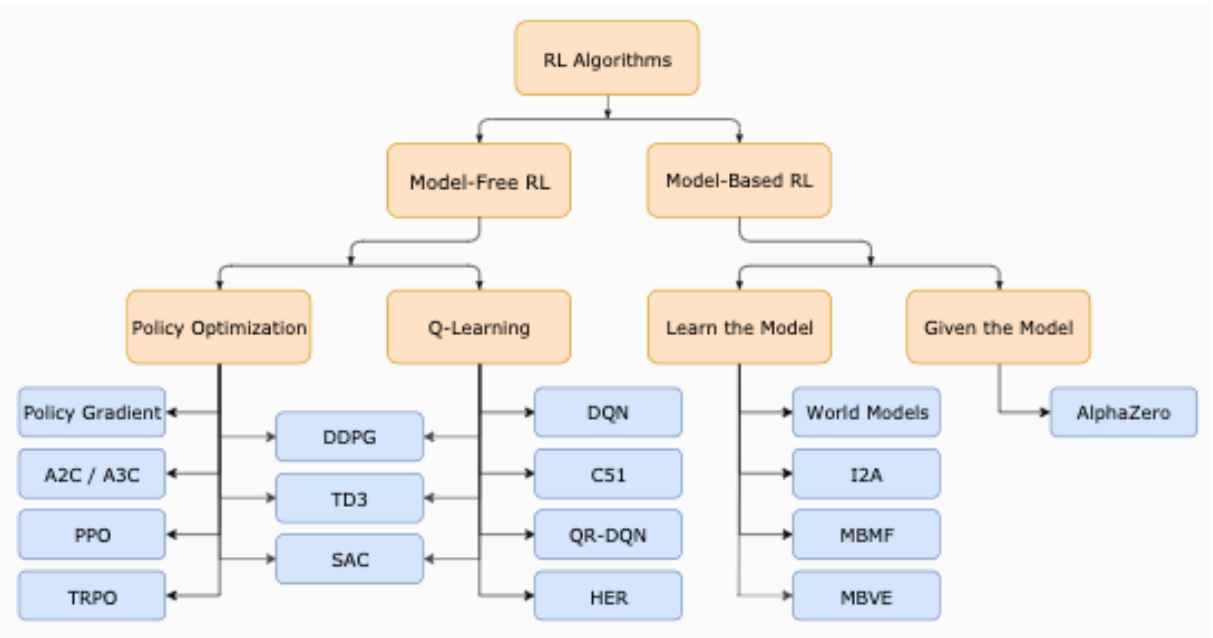
10.5 Hindsight Experience Replay

10.6 RL Algorithms

10.6.1 Taxonomy

-
- Figure from OpenAI SpinningUp
- Does the agent have access to a model of the environment?
- Having a model alows the agent to plan by thinking ahead
- Ground-truth model is not always available to the agent, must learn model from experience
- In model-free there are two main approaches for training agents: policy optimization and q-learning
- Policy optimization

Figure 10.1: Taxonomy of RL Algorithms



- Represents the policy as $\pi_\theta(a|s)$ and optimize the parameters θ by gradient ascent on $J(\pi_\theta$
- The optimization is almost always performed on-policy
- Also learn an approximator $V_\phi(s)$
- Directly optimizes for the thing that you want**
- Q-learning learns an approximator $Q_\theta(s,a)$ for the optimal action-value function
- Q-learning is almost always performed off-policy, update uses data collected at any point during training
- $a(s) = \text{argmax}_a Q_\theta(s,a)$
- Indirectly optimization, tens to be less stable, more sample efficient because they can reuse data more effectively

10.6.2 Policy Gradients

- Maximize the expected return $J(\pi_\theta) = E_{\tau\sim\pi_\theta}[R(\tau)]$
- Optimize policy: $\theta_{k+1} = \theta_k + \alpha\nabla_\theta J(\pi_\theta)|_{\theta_k}$
- Gradient of policy performance $\nabla_\theta J(\pi_\theta)$ is the policy gradient
- $\nabla_\theta J(\pi_\theta) = E_{\tau\sim\pi_\theta}[\sum_{t=0}^{T} \nabla_\theta \text{log}\pi_\theta(a_t|s_t)R(\tau)]$
- This is the grad log probability and since it is an expectation, we can estimate it with a sample mean by collecting a set of trajectories
- The "loss" for policy gradient algorithm is not the same loss in the typical sensen from supervised learning

- Loss doesn't measure performance, it doesn't mean anything. It is possible to have low loss and have poor policy performance, this is due to policy overfitting to a batch of data
- EGLP or Expected Grad-Log-Prob Lemma states that $E_{x\sim P_\theta}[\nabla_\theta log P_\theta(x)] = 0$
- Reward-to-go policy gradient, we only care about the reward that came after taking an action
- Past rewards add noise to sample estimates of the policy gradient
- EGLP lemme implies that we can add/subtract any function b that only depends on the state from the policy gradient without changing its expectation (this is called a baseline)
- Results in faster and more stable policy learning
- Common baselines:
  - $\Phi_t = R(\tau)$
  - $\Phi_t = \Sigma_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1})$
  - $\Phi_t = \Sigma_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1}) - b(s_t)$
  - $\Phi_t = Q^{\pi_\theta}(s_t, a_t)$ (on-policy action-value function)
  - $\Phi_t = A^{\pi}(s_t, a_t)$ (advantage function, describes how much better or worse an action is compared to other actions)
- Baseline is often approximated using a neural network that is updated concurrently wiht the policy using a mean-squared error

10.6.3 Deep Deterministic Policy Gradient (DDPG)

-

10.7 Batch/Offline Reinforcement Learning

- Most of RL algorithms assume that an agent interacts with an online environment or simulator and learns from its own collected experience. This is expensive and often requires a high-fidelity simulator which can be hard to build
- Offline RL addresses the problem of learning a policy from a fix set of trajectories, without any further interaction with the environment
- In principle, off-policy algorithm can learn from data collected by any policy
- Recent work shows that standard off-policy deep RL algorithms diverge in offline setting
- Removes design of replay buffer and exploration
- Challenging due to distribution mismatch between current policy and offline data collection policy
- Datasets:

  - D4RL
  - Developed tasks that reflect both real-world dataset challenges and real-world applications
  - Autonomous driving, robotics, and other domains
- Batch RL algorithms
  - Quantile Regression DQN (QR-DQN)
  - Random Ensemble Mixture REM
  - Batch Constrained Deep Q-learning (BCQ)
- Batch Constrained deep Q-learning (BCQ) [2]
  - Run normal Q-learning but in the maximization step, instead of considering max over all possible actions, only consider actions a' such that (s', a') actually appear in the batch of data
  - Train a *generative model* - variational autoencoder - to generate actions that are likely to be from the batch
  - Also a *perturbation model* to perturb the actions

10.8 Curiosity Driven Exploration

10.9 Inverse Reinforcement Learning

10.10 General tips when using RL

- Notes taken from the stable-baselines documentation
- Data in RL for training an agent is collected through interactions with the environment
- This dependences can lead to vicious circle, if the agent collects poor data, it will not improve and keep exploring bad trajectories
- Algorithms are dependent on hyperparameters
- Normalize input to agent and use common preprocessing techniques (e.g. frame-stack, etc)
- Model-free RL algorithms are usually *sample inefficient*
- Expert knowledge is often needed to design adequate reward function
- Unstable training, often huge drops in performance especially in DDPG. Methods to remedy this include TD3, TRPO, PPO, etc
- Most algorithms use exploration noise, need a separate test environment to evaluate agent at test time
- Some policies are stochastic by default (e.g. A2C or PPO)

- DQN only supports discrete actions, SAC is restricted to continuous actions
- Also depends on whether you want to parallelize your training
- Keep action space in range $[-1, 1]$ and symmetrical, used in Gaussian policies

Chapter 11

Natural Language Processing

11.1 Natural Language Tasks and Application

- Machine Translation
- Summarization (long text → short text)
- Dialogue (previous utterance → next utterance)
- Parsing (input text → output parse)
- Code generation (natural language → Python code)
- Entailment
- Natural Language Generation

11.2 Neural Machine Translation

- task of translating a sentence x from one language (source) to a sentence y in another language (target)
- (2014) sequence-to-sequence and involves using two RNNs

11.3 RNN

11.4 Encoder-decoder models

- Conditional Language Model
- conditional because predictions are conditioned on the source sentence x
- Encoder RNN produces an encoding of the source sentence
- Decoder RNN is a Language Model generates a target sentence, conditioned on encoding
- decoder predicts the next word of the target sentence y

- "end-to-end", learn entire system with respect to a single loss, optimize the system as a whole, more likely to succeed
- Cross entropy loss on probability distribution output from decoder model
- Padding short sentences to maximum length of the batch and apply attention mask
- Greedy decoding has no way to undo decisions or perform backtracking
- Beam search decoding

11.5 Evaluation Metrics

- BLEU (Bilingual Evaluation Understudy)
  - compares machine-written translation to one or several human-written translation and compute a similarity score based on n-gram precision
  - plus a penalty for too-short translations
  - useful but imperfect!

11.6 Challenges

- Out-of-vocabulary words
- Domain mismatches between train and test data
- Maintaining context over longer text (e.g. translate a whole news article or book)
- Low-resource language pairs (e.g. don't have large corpus, language that is not commonly used)
- Common sense is hard
- Biases in training data (e.g. gender, race, etc)
- Models are uninterpretable and do strange things

11.7 Attention

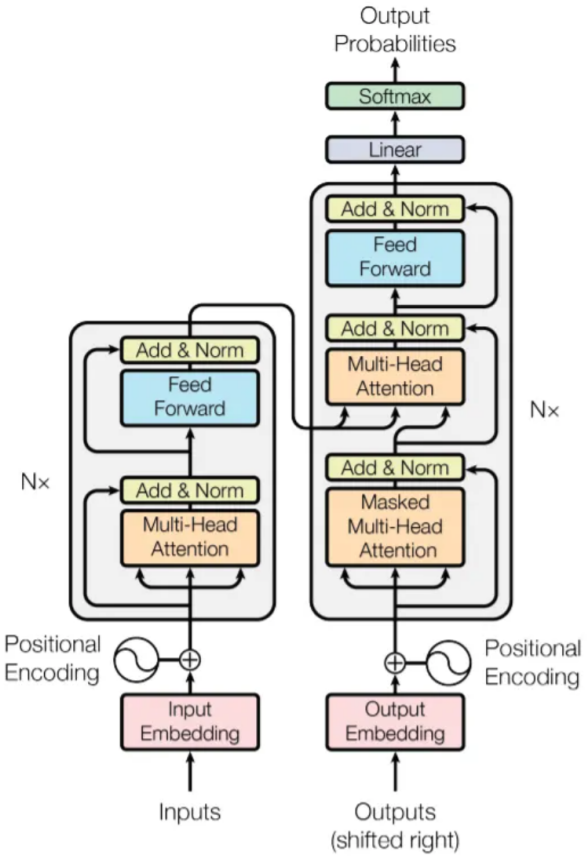
- Seq-to-seq encodes source sentence into a single embedding vector. This needs to capture all information in the sentence and causes an information bottleneck!
- Idea is to encode each word in a sentence into a vector
- When decoding, take the linear combination of these vectors weighted by "attention weights"
- "Query" vector and "key" vector
- Calculate weight for each query-key pair and normalize using softmax to get a probability distribution
- Use attention distribution to take weighted sum of encoder hidden states where the weights are the attention scores

- Encoder hidden states: $h_1, \ldots, h_N \in \mathbb{R}^h$
- Decoder hidden states: $s_t \in \mathbb{R}^h$
- Attention scores: $e^t = [s_t^T h_1, \ldots, s_t^T h_N] \in \mathbb{R}^N$
- $\alpha^t = \text{softmax}(e^t) \in \mathbb{R}^N$
- $a_t = \sum_{i=1}^N \alpha_i^t h_i \in \mathbb{R}^h$
- Attention allows decoder to focus on certain part of the source
- solves the information bottleneck problem and the vanishing gradient problem and more interpretability
- we get (soft) alignments for free, never explicitly trained alignment system like Statistical Machine Translation
- Given a set of vector values and vector query, attention computes a weighted sum of the values, dependent on the query
- Query attends to the values, selective summary, fixed-sized representation of an arbitrary set of representations
- Variants of computing attention scores
  - Dot-product attention $e_i = s^T h_i \in \mathbb{R}$
  - Multiplicative attention: $e_i = s^T W h_i \in \mathbb{R}$
  - Additive attention: $e_i = v^T \tanh(W_1 h_i + W_2 s \in \mathbb{R})$

11.8 Transformers and Self-attention

- In self-attention, the concept of attention is used to encode sequences instead of RNNs
- Both encoder and decoder don't have RNNs and instead use attention mechanisms
- Each word in the sentence attends to every other word and thus relationships between words in the sequence are captured
- Encoder layer (can stack N times)
- Word embedding vectors
- Positional encodings: make sure that even if we don't have RNN that processes input sequentially, we can still distinguish positionss
- Multi-headed attention, each head will learn something different, more representation power
- Decoder layer - it is autoregressive
- Masking to prevent model from access to future words
-

Figure 11.1: Transformer model



Chapter 12

Clustering

12.1 K-means

12.2 Hierarchical clustering

12.3 Nonparametric clustering

12.3.1 Dirichlet processes

- DD is a distribution of distributions, each sample from DD is a categorial distribution over K categories
- DD is parameterized by G_0 and α a scale factor
- When α is large, samples from $DD(\alpha \cdot G_0)$ will be close to G_0
- Dirichlet process is used to cluster data *without specifying the number of clusters in advance*
- nonparametric because its dimensionality is infinite
- exhibits a rich-gets-richer property
- observations are probabilitistically assigned to clusters based on the # of observations in that cluster

$$P(cluster = k) = \frac{n_k}{\alpha + n - 1}$$

$$P(cluster = new) = \frac{\alpha}{\alpha + n - 1}$$

- DP tutorial
- Another DP tutorial

12.3.2 Chinese restaurant process

- Restaurant starts off empty
- First person selects a group

- Second person sits at a new table with probability $\frac{\alpha}{\alpha+1}$ and sits with the first person with probability $\frac{1}{\alpha+1}$

12.3.3 Polya Urn Model

- Same model as CRP
- urn contains αG_0 balls of color x for each x
- at each timestep, draw a ball from the urn and drop it back into the urn plus another ball of the same color
- CRP specifies only a distribution over partitions, but does not assign parameters to each group whereas the Polya Urn Model does both

12.3.4 Stick-breaking construction

- Figure out the proportion of points that fall into a particular group
- Start with a stick of length 1
- Generate a random variable $\beta_1 \sim Beta(1, \alpha)$ and break off the stick at β_1
- Take the stick to the right and repeat
- Stick-breaking is CRP or Polya Urn from a different perspective

12.3.5 Gibbs sampling

-

12.3.6 Metropolis Hastings

- CRP, Stick-breaking, and Polya Urn are all *sequential* models for generating groups, there are also *parallel* models for generating groups
- Bayesian clustering algorithms often rely on the Dirichlet Distribution (DD) to encode prior information about cluster assignments
- Streaming data

Chapter 13

10-708 PGM

13.1 Lecture #1: Introduction to GMs

- Graphical model is a method of modeling a probability distribution for reasoning under uncertainty
- Three important questions: 1) representation, 2) inference, and 3) learning
- Benefits of GMs:
  - 1) reduces the number of parameters required to describe the joint distribution of a set of random variables
  - a full joint distribution would require 2^{n-1} where n is the number of variables parameters
  - 2) data integration, each term becomes self-contained and we can estimate terms with only the relevant data points
  - problem of joint estimation n be modularized
  - 3) generic method of representing knowledge and making inferences
  - inject domain knowledge through priors
- Bayes Net is represented by a DAG
- Each node in a Bayes net has a Markov blanket = its parents, children, and children's parents
- Every node is conditionally independent of nodes outside its Markov blanket
- Markov Random Field uses an undirected graph and every node is conditionally independent of all nodes other than its immediate neighbors
- Notes 2019
- Notes 2020

13.2 Lecture #2: Bayesian Networks

- **Evaluation:** how likely is the sequence of observations?
- **Decoding:** what portion of the sequence was generated with the fair die, and what portion with the loaded die?
- **Learning:** how "loaded" is the loaded die?
- We can model the casino problem using a Hidden Markov Model (HMM)
- The past is conditionally independent of the future given the present (Markov property)
-
- Notes

13.3 Lecture #3: Parameter Estimation

- GM learning can be split into: structural learning and parameter learning
- Learning parameters for BN that has known, fixed structure and completely observable
- We have a dataset of N iid examples $D = \{x_1, \ldots, x_N\}$
- *Exponential family* is a parametric set of probability distributions that characterize many distributions like Bernoulii, Multinomial, Gaussian, Poisson, and Gamma

$$\begin{aligned} p(x|\eta) &= h(x)\exp[\eta^T T(x) - A(\eta)] \\ &= \frac{1}{Z(\eta)} h(x)\exp[\eta^T T(x)] \end{aligned}$$

$A(\eta)$ = log normalizer
$T(x)$ = sufficient statistic
η = canonical parameter

- Easy to compute $q-th$ central moments of exponential family distributions by taking $q-th$ derivative of the log normalizer $A(\eta)$
- 1-to-1 relationship between canonical and moment parameters, $\eta = \psi(\mu)$

13.3.1 Generalized Linear Models (GLIMs)

- Input enters the model via a linear combination $\xi = \theta^T x$
- μ is a function of $\xi : f(\xi)$
- Output y is characterized by an exponential family distribution p
- $E_p(y) = \mu = f(\theta^T x)$ is a GLIM
- Examples of GLIM include linear regression and logistic regression

- Other examples: MRFs, Restricted Boltzmann Machines, CRFs
- MLE for learning GLIMs (?)
- MLE of a normal distribution, we take the log-likelihood of $l(\mu)$ and take the derivative wrt to μ, set equal to 0 and solve for μ, we get that $\mu_{MLE} = \frac{1}{n}\sum_{i=1}^{n} x_i$
- In GMM, each cluster corresponds to a Gaussian distribution and there are k clusters
- Parameters are $\theta = \{\mu_1, \ldots, \mu_K, \sigma_1, \ldots, \sigma_K, \pi_1, \ldots, \pi_K\}$ where π_k is the mixture proportion representing probability that X_i belongs to the k-th mixture component
- MLE for GMM is different because we can't analytically solve for μ because of summation. Would be easy if we new the latent variables Z_i or cluster assignments
- EM is sensitive to initialization of parameters
- EM is used to find ML estimates for models with latent variables
- E-step use current value of parameters to find posterior distribution of latent variable: $P(Z|X, \theta^0$
- M-step determine new parameters $\hat{\theta}$
- Nice tutorial on EM

13.3.2 Parameter Estimation for Partially Observed GMs

- Often some RVs for GMs may be unobserved, some are not physically measurable, some are not observed because of faulty sensors
- Want to estimate parameters of a GMM given partially observed data using the EM Algorithm
- EM alternates between 1) computing cluster assignments at time t using the means at time t and 2) using the cluster assignments at time t to recompute mean for next iteration

13.4 Lecture #6: Case studies: HMM and CRF

13.5 Lecture #7: Variation Inference 1

- VI converts inference into optimization, approximate desired solution by relaxing intractable optimization problem
- e.g. linear systems of equation that is too large to invert
- There is some probability distribution P that we can't perform inference on, try to approximiate P with another distribution Q
- Requires some way to measure distance between P and Q (usually KL)
-

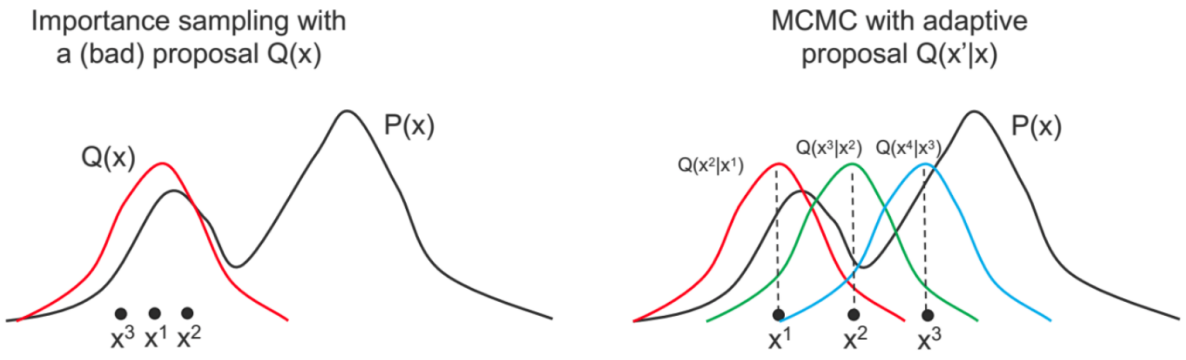
13.6 Lecture #9: Sampling 1

- Inference method for an arbitrary distribution
- Instead of manipulating $P(X)$, we estimate it using samples from the distribution
- Drawback of naive sampling is that for large models, it is difficult to get estimates for rare events
- Rejection sampling: sample from a simpler distribution $Q(x)$ and accept sample with probability
- If $Q(x)$ is not chosen well, we will get a lot of rejectd samples
  - Sample x_0 from q_x
  - Sample a number u_0 from uniform distribution over $[0, kq(x_0)]$ where k is a constant such that $kq(x) \geq \tilde{p}(x)$
  - Reject if $u_0 > \tilde{p}(x_0)$
- Importance sampling:
  -

13.6.1 Markov Chain Monte Carlo (MCMC)

- Metropolis-Hastings Algorithm
  - MH works by simulating a Markov Chain whose stationary distribution is π
  - Idea is that instead of using a fixed $Q(x)$ like likelihood weighting, we use an adaptive proposal $Q(x'|x)$

Figure 13.1: Adaptive Proposal



Comparison between using a fixed (bad) proposal and an adaptive proposal.

  - MH has a "burn-in" period where an initial number of samples are thrown away because they are not from the true distribution
- The user needs to provide a "transition kernel", Q. Q is often a continuous distribution. Random walk kernel: $Q(y|x) = \frac{1}{\sqrt{2\pi}}\exp^{-0.5(y-x)^2}$

- MH tutorial
- Gibbs Sampling
- Suppose there is some $p(x, y)$ that is difficult to sample from directly
- $p(x|y)$ and $p(y|x)$, the conditional distributions are easier to sample from
- Also has a "burn-in" period
- GS is a special case of MH
- Practical aspects:
  - Evaluate $Q(x'|x)$: acceptance rate, autocorrelation function
  - When to stop burn-in: plot the log-likelihood v.s. time

Algorithm 2: MH algorithm

Initialize $X_1 = x_1$;
for *t = 1,2,...* **do**
  Sample y from $Q(y|x_t)$;
  $A = min(1, \frac{\pi(y)Q(x_t|y)}{\pi(x_t Q(y|x_t))})$, A is the "acceptance probability";
  If Q is symmetric, the formula for $A = min(1, \frac{\pi(y)}{\pi(x_t)})$;
  Accept y with probability A and set $x_{t+1} = y$
end

Algorithm 3: Gibbs Sampling

Set x and y to some initial starting values: (x_0, y_0);
Sample $x|y$, $x_1 \sim p(x|y_0)$ and then sample $y|x$, $y_1 \sim p(y|x_1)$;
Repeat;

13.6.2 Important probability distributions

- Multinomial
  - K-outcome multinomial distribution
  - generalization of the Bernoulli distribution
  - models probability of counts for a k-sided die rolled n times
  - $k = 2$ and $n = 1$, the multinomial distribution is the Bernoulli
  - $k = 2$ and $n > 1$, this is the binomial distribution
  - $k > 2$ and $n = 1$, this is the categorical distribution

$$p(x|\pi) = \pi_1^{x_1} \pi_2^{x_2} \dots \pi_K^{x_K}$$

- Bernoulii
  - probability that flipping a coin one time will result in heads or tails
- Gaussian

- k-dimensional Gaussian

$$p(x|\mu,\Sigma) = \frac{1}{(2\pi)^{k/2}|\Sigma|^{\frac{1}{2}}}\exp\{\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\}$$
$$= \frac{1}{(2\pi)^{\frac{k}{2}}}\exp\{-\frac{1}{2}\text{Tr}(\Sigma^{-1}xx^T + \mu^T\Sigma^{-1}x - \frac{1}{2}\mu^T\Sigma^{-1}\mu - log|\Sigma|\}$$

- Poisson
  - used for modeling the number of times an event occurs in an interval of time or space
  - $\text{Pois}(\lambda) = \frac{\lambda^k e^{-\lambda}}{k!}$
- Gamma

13.6.3 Markov Chain Concepts

- Markov Chain is a sequence of RVs with the Markov Property: $P(x^{(n)} = x|x^{(1)}, \ldots, x^{(n-1)}) = P(x^{(n)} = x|x^{(n-1)})$
- Transitions
- **Stationary distributions:** if it does not change under transition kernel, $\pi(x') = \sum_x \pi(x)T(x'|x)$
- Intuition: in the long run, no matter where the starting state was, the proportion of time the chain spends in state j is approximately ψ_j for all j, i.e. $\psi P = \psi$
- **Irreducible**: you can get from any state x to another other state x' with probability > 0 in a finite number of steps
- Aperiodic: you can return to any state x at any time
- **Ergodic**: irreducible and aperiodic
- Erogdicity implies you can reach the stationary distribution no matter the initial distribution
- **Reversible**: there exists a distribution $\pi(x)$ such that $\pi(x')T(x|x') = \pi(x)T(x'|x)$
- Reversible MCs **always** have a stationary distribution

Chapter 14

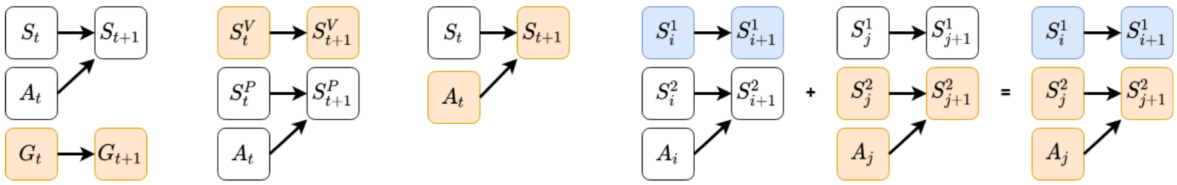
Paper Outlines

14.1 Counterfactual Data Augmentation (CoDA)

- dynamic processes involve a set of interacting subprocesses
- interactions between subprocesses are sparse
- introduces **local causal models** included from a global causal model by conditioning on a subset of state space
- local structures + experience replay to generate counterfactual experiences
- locally (during time between interaction) factor dynamics and model subprocesses independently
- underlying processes are difficult to model precisely
- factored subspaces of observed trajectory pairs are swapped
- CoDA is a data-augmentation strategy
- use attention to discover local causal structure
- improves sample-efficiency in batch-constrained and goal-constrained RL
- model time slice $(t, t+1)$ using a structural causal model (SCM)
- $\mathcal{M}_t = \langle V_t, U_t, \mathcal{F} \rangle$ with a DAG $\mathcal{G}$
- V_t is the set of state, action and next states, U_t is a set of noise variables and $\mathcal{F}$ is the set of structure equations that maps noise $\times$ current state to next state
- assume structural minimality allows us to think of edges in $\mathcal{G}$ as representing global causal dependence
- $P(S^i_{t+1}|S_t, A_t) = P(S^i_{t+1}|Pa(S^i_{t+1}))$ and S is independent of all nodes that isn't its parent
- often exists large subspace ($\mathcal{L}^{(j \perp\!\!\!\perp i)} \subset \mathcal{S} \times \mathcal{A}$) such that the next state is independent
- e.g. two-armed robot, there are many states such that the two arms are too far apart to influence each other

- consider a *local causal model* ($\mathcal{M}_t^{\mathcal{L}^{(j \perp\!\!\!\perp i)}}$) whose DAG is strictly sparser than the global DAG

Figure 14.1: Different instances of CoDA



(a) Goal relabeling **(b)** Visual augmentation **(c)** Dyna (model-based) **(d)** Counterfactual Data Augmentation by swapping locally independent subspaces **(ours)**

- Dyna augments real states with new actions and resamples the next state using a learned dynamics model
- can generate an exponential amount of data with CoDA (n independent samples from subspace $\mathcal{L}$ whose graph has m connected components $\implies n^m$ samples)
- CoDA can be used to mix and match data across timesteps

Figure 14.2: CoDA algorithm

Algorithm 1 Mask-based Counterfactual Data Augmentation (CoDA)

function CODA(transition `t1`, transition `t2`):
`s1, a1, s1' ← t1`
`s2, a2, s2' ← t2`
`m1, m2 ←` MASK(`s1, a1`), MASK(`s2, a2`)
`D1 ←` COMPONENTS(`m1`)
`D2 ←` COMPONENTS(`m2`)
`d ←` random sample from (`D1 ∩ D2`)
$\tilde{\texttt{s}}, \tilde{\texttt{a}}, \tilde{\texttt{s}}$`' ←` copy(`s1, a1, s1'`)
$\tilde{\texttt{s}}$`[d], ã[d], s̃'[d] ← s2[d], a2[d], s2'[d]`
$\tilde{\texttt{D}} \leftarrow$ COMPONENTS(MASK($\tilde{\texttt{s}}, \tilde{\texttt{a}}$))
return ($\tilde{\texttt{s}}, \tilde{\texttt{a}}, \tilde{\texttt{s}}$`'`) **if** `d` $\in \tilde{\texttt{D}}$ **else** $\emptyset$

function MASK(state `s`, action `a`):
Returns $(n+m) \times (n)$ matrix indicating if the n next state components (columns) locally depend on the n state and m action components (rows).

function COMPONENTS(mask `m`):
Using the mask as the adjacency matrix for $\mathcal{G}^{\mathcal{L}}$ (with dummy columns for next action), finds the set of connected components $C = \{C_j\}$, and returns the set of independent components $D = \{\mathcal{G}_i = \bigcup_k C_k^i \,|\, C^i \subset \text{powerset}(C)\}$.

- specify a ratio between observed to counterfactual data to control for selection bias
- inferring local fatorization, how to approximate causal model
- use a global network mask (MADE) for autoregressive distribution modeling and GraN-DAG for causal discovery
- locally conditioned network mask by taking matrix product of locally conditioned layer masks
- MLP and single-head set transformer
- Inferring local factorization
  - SANDy (Sparse Attention Neural Dynamics)

  - it learns a function or mask that represents the adjacency matrix of the local causal graph
  - SANDy transformer performs better than MLP in AUC score
  - SANDy mixture is only sufficient for the simple synthetic MP environments, does not work for Spriteworld
  - the transformer has stronger inductve bias and more reliably infer local interaction patterns
- Some experiment details and notes:
  - working with the original TD3 codebase
  - mask function trained using 42k samples from random policy
  - increase agent batch size from 256 to 1000, more batch size allows agent to see more of its on-policy data in the face of many off-policy CoDA samples
  - batch-RL experiment in the Pong environment
  - trained a transformer to learn a mask function
  - goal-conditioned RL experiments on OpenAI gym FetchPush environment
  - these experiments use a hand-coded heuristic designed with domain knowledge
  - e.g. action is entangled with gripper and gripper + objects are disentangled when they are more than 10cm apart
- Other experimental insights:
  - tried augmenting the buffer by sampling from a dynamics model similar to model-based RL
  - good at next-state prediction, but fail to capture collisions and long-term dependencies

14.2 Differentiable Causal Discovery from Interventional Data (DCDI)

- learning a causal directed acyclic graph from data
- reformulates as continuous constrained optimization problem, solved using the augmented Lagrangian method
- this work proposes a NN model that can leverage interventional data
- using only observational data is challenging because the *faithfulness assumption* states that the true DAG is only identifiable up to a *Markov equivalence class*
- this can be improved by considering interventional data
- when observing enough interventions, the DAG is exactly identifiable
- there are score-based and constraint-based optimization methods, but they are computationally expensive and rely on parametric assumption

- perfect interventions remove the dependencies of a node on its parents, e.g. gene knockout in biology
- two classes of methods in causal structure learning
  - constraint-based methods
    - PC algorithm works with observational data
    - rely on conditional independence
    - COmbINE and HEJ support interventional data
    - JCI supports latent cofounders and can deal with interventions with unknown targets
  - score-based methods
    - formulate problem of estimating DAG G^* by optimizing a score function $\mathcal{S}$ over the space of DAGs

$$\mathcal{G} \in \underset{\mathcal{G} \in \text{DAG}}{\text{argmax}} \mathcal{S}(\mathcal{G})$$

    - common choice in the purely observational setting is the regularized ML

$$\mathcal{S}(\mathcal{G}) := \max_{\theta} \mathbb{E}_{X \sim P_X}[log f_\theta(X) - \lambda |\mathcal{G}|]$$

    - the space of DAGs is super-exponential in the number of nodes
    - these methods rely on greedy combinatorial search algorithms
    - e.g. GIES (adaptation of GES), assumes a *linear* gaussian model
    - CAM uses greedy search, nonlinear, additive noise model
  - hybrid-methods
    - combines both score-based and constraint, e.g. IGSP
- *weighted adjacency matrix* and acyclicity constraint: $\text{Tr}e^{A_\theta} - d = 0$
- shown that graph is acyclic iff the above constraint is satisfied
- problem is solved approximately using an augmented Lagrangian procedure
- performance is assessed by two metrics comparing estimated graph to the ground-truth graph
- *structural Hamming Distance* (SHD) = number of edges that are different between the two DAGs
- *structural interventional distance* (SID) how the two DAGs differ wrt to their causal inference statements
- DCDI relies on stochastic gradient descent and thus is scalable to graphs with hundreds of nodes even though the augmented Lagrangian procedure requires computing marix exponential which is $O(d^3)$

- Synthetic datasets
  - first sample a DAG using the Erdos-Renyi scheme
  - perfect v.s. imperfect interventions
  - different types of causal mechanisms: linear, additive noise model (ANM), and nonlinear with non-additive noise (NN)
-

14.3 Off-Policy Deep Reinforcement Learning without Exploration (BCQ)

- Extrapolation error: unseen state-action pairs are estimated to have unrealistic values
- Error can be attributed to mismatch in distribution of data induced by policy and data contained in the batch
- BCQ uses state-conditioned generative model to produce only previously seen actions
- Extrapolation error can be attributed to several causes:
  - Absent data: (s, a) is unavailable
  - Model bias
  - Training mismatch
- Off-policy algorithms (e.g. DDPG) deteriorate in perfomance when data is uncorrelated and value estimate produced by the Q-net diverges
- Off-policy algorithms are ineffective when learning *truly off-policy*
- Three different batch tasks: final buffer, concurrent, and imitation
- Behavioral agent consistently outperformed the off-policy agent trained with DDPG
- Batch-constrained policies trained to select actions with respect to three objectives:
  - Minimize distance of selected actions to data in the batch
  - Lead to states where familiar data can be observed
  - Maximize the value function
- BCQ generates plausible candidate actions with high similarity to the batch and selects the highest valued action using a learned Q-network
- Actions outputted using a generative model $G_w(s)$
- A perturbation model $\xi_\phi(s, a, \Phi)$ which is used to increase the diversity of the actions
- The choice of the number of actions to sample n and Φ creates a trade-off between imitation learning and RL

- Also uses a modified Clipped Double Q-Learning to estimate value by taking a convex combination of two Q-networks

Algorithm 1 BCQ

Input: Batch $\mathcal{B}$, horizon T, target network update rate τ, mini-batch size N, max perturbation Φ, number of sampled actions n, minimum weighting λ.
Initialize Q-networks $Q_{\theta_1}, Q_{\theta_2}$, perturbation network ξ_ϕ, and VAE $G_\omega = \{E_{\omega_1}, D_{\omega_2}\}$, with random parameters θ_1, θ_2, ϕ, ω, and target networks $Q_{\theta'_1}, Q_{\theta'_2}$, $\xi_{\phi'}$ with $\theta'_1 \leftarrow \theta_1, \theta'_2 \leftarrow \theta_2, \phi' \leftarrow \phi$.
for $t = 1$ **to** T **do**
  Sample mini-batch of N transitions (s, a, r, s') from $\mathcal{B}$
  $\mu, \sigma = E_{\omega_1}(s, a), \quad \tilde{a} = D_{\omega_2}(s, z), \quad z \sim \mathcal{N}(\mu, \sigma)$
  $\omega \leftarrow \text{argmin}_\omega \sum (a - \tilde{a})^2 + D_{\text{KL}}(\mathcal{N}(\mu, \sigma) || \mathcal{N}(0, 1))$
  Sample n actions: $\{a_i \sim G_\omega(s')\}_{i=1}^n$
  Perturb each action: $\{a_i = a_i + \xi_\phi(s', a_i, \Phi)\}_{i=1}^n$
  Set value target y (Eqn. 13)
  $\theta \leftarrow \text{argmin}_\theta \sum (y - Q_\theta(s, a))^2$
  $\phi \leftarrow \text{argmax}_\phi \sum Q_{\theta_1}(s, a + \xi_\phi(s, a, \Phi)), a \sim G_\omega(s)$
  Update target networks: $\theta'_i \leftarrow \tau\theta + (1 - \tau)\theta'_i$
  $\phi' \leftarrow \tau\phi + (1 - \tau)\phi'$
end for

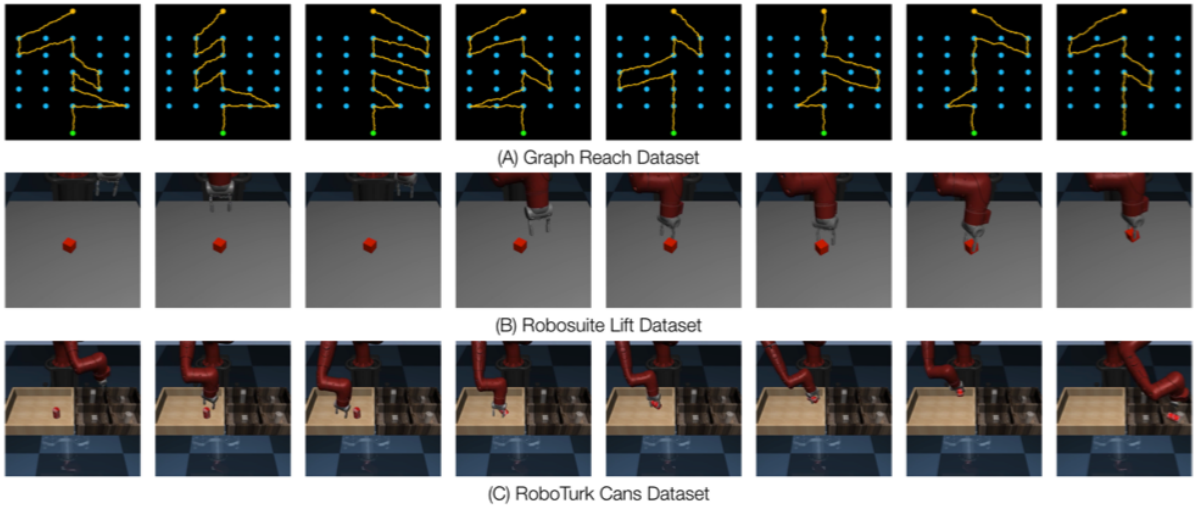
- Experiments and results
  - MuJoCo environments in OpenAI gym
  - Baselines: DDPG, DQN, BC, and VAE-BC
  - Tasks: HalfCheetah, Hopper, Walker2d
  - BCQ is the only one that succeeds at all tasks, matching or outperforming BC
  - Achieves high performance in very few iterations, able to disentangle poor and expert actions
  - Suggests that extrapolation error has been successfully mitigated, BCQ is able to accurately estimate the true Q-value

14.4 IRIS: Implicit Reinforcement without Interaction at Scale for Learning Control from Offline Robot Manipulation Data

- IRIS is a framework that addresses the problem of offline policy learning from a large set of diverse and suboptimal demonstrations
- Use demonstrated data in lieu of reward function, avoids the problem of exploration
- Conventional IL methods assume that demonstration data is near-optimal and unimodal
- High-level controller selects a new goal state s_g for T timesteps, low-level controller is conditioned on s_g and tries to reach that state in the T timesteps

- High-level controller has two parts
  - conditional VAE (cVAE): predicts the distribution of states $p(s_{t+T}|s_t)$, used to sample goal proposals
  - value function $V(s)$ used to select the most promising goal proposal
  - value function is trained using a simple variant of BCQ
- Low-level controller:
  - a goal-conditioned RNN that outputs an action a_t given s_t and s_g
  - trained on trajectory sequences from dataset, the last observation in each sequence is treated as the goal
  - trained using a simple Behavioral Cloning loss
  - learns to copy action sequence that resulted in a particular observation
- This induces *selective imitation*
- Experiments:
  - Graph reach is 2D navigation domain, contains 250 demonstrations, demonstration paths can take detours
  - Robosuite Lift: actuate Sawyer robot to grasp and lift cube from table
  - RoboTurk Can Pick and Place task and Can Image task

Figure 14.3: Tasks



  - Baselines: BC, BC-RNN, BCQ, IRIS w/o cVAE, IRIS w/o Value Network
  - All baselines achieve perfect performance on Graph Reach, IRIS 81.3% on Lift, and 28.3% on Cans dataset

14.5 Building Machines That Learn and Think

- desire to build systems that learn and think like people
- machines should build causal models of the world that support explanation and understanding and not just pattern recognition
- ground learning in intuitive physics
- harness compositionality and learning-to-learn, rapidly acquire knowledge to new tasks and situations
- deep learning models may be solving the problems differently than people do
- Prediction versus explanation
- Developmental start-up software, cognitive capabilities that are present early in development
  - 1) intuitive physics
  - primitive object concepts that allow them to track objects over time and discount physically implausible trajectories
  - new task, but physics still works the same way
  - 2) intuitive psychology
  - understanding that people have mental states like goals and beliefs
- model-building is the basis of human-level learning
- compositionality and learning-to-learn can make rapid model learning possible
- we are incredibly fast at perceiving and acting
- neural networks are designed for pattern-recognition rather than model-building
- integrate NN with rich model-building mechanisms can explain how human minds understand the world so well, so quickly
- humans can learn a lot more from a lot less
- single example of a new visual concept can be enough information to support: classification of new examples, generation of new examples, parsing object into parts and relations, and generation of new concepts
- DQN, simple model-free RL algorithm, that learns to play the game Frostbite
- visual system and policy are highly specialized to the games that it was trained on
- DQM plays games at human-level performance, but it is doing so in a way different than humans
- DQN trained on 200 million frames, 924 days, about 500 times more training experience as a human, not sample efficient
- Fundamental differences in representation and learning between people and DQN
- DQN relies on some reward function, otherwise take random actions

- DQN is inflexible to changes in inputs and goals, e.g. changing color of object will be harmful to performance
- People can understand the game + goals quickly. Moreover, people understand enough to invent new goals, generalize changes to input, and explain the game to others
- How do we bring to bear rich prior knowledge to learn new tasks and solve new problems quickly?
- Intuitive physics-engine approach to scene understanding
- PhysNet used DCNN to predict stability of block towers from simulated images
- Could neural networks be trained to emulate a general-purpose physics simulator?
- Integrating intuitive physics and DL could be important to more human-like learning algorithms
- Intuitive psychology can allow us to infer the beliefs, desires, and intentions of others (e.g. avoiding bird in Frostbite game)
- Injecting inductive bias can boostrap reasoning about abstract concepts
- One-shot learning is innate characteristic of humans from a young age
- Compositionality is the idea that new representations can be constructed from the combination of primitive elements
- Object-oriented reinforcement learning, representing a scene as a composition of objects
- compositionality is important at the level of goals and subgoals
- causality is a subclass of generative models that resemble how the data are actually generated
- causality can glue features together by relating them to a deep underlying cause
- perception without key ingredients and absence of causal glue can lead to errors
- network can get the objects correctly but fail to understand the physical forces at work
- learning to learn, learning a new task can be accelerated through previous or parallel learning of related tasks
- people transfer knwoledge at multiple-levels, learn compositionally structured causal models of a game
- there is evidence that suggests our brain has a model-based learning system, building a map of the environment and using it to plan action sequences for complex tasks
- Intrinsically motivated learning
- Responses to common questions
  - It is unfair to compare learning speeds of human and machine because of apriori knowledge / experience

  - Neuroscience in the long run will place more constraints on theories of intelligence
  - Language is essential to human intelligence and goes hand in hand with other key ingredients
  - Language facilitates more powerful learning-to-learn and compositionality, allow people to learn more quickly and flexibly
- AI has made incredible progress: beat chess masters, Jeopardy champions, facial recognition, speech understanding
- More exciting applications to come: self-driving, medicine, drug design, genetics, and robotics
- Recent advancements in incorporating psychological ingredients with deep networks: selective attention, augmented memory, experience replay
- Attention allows the model to focus on smaller sub-tasks rather than solving whole problem in one-shot
- Memory incorporates classical data structures into gradient-based learning systems
- AI systems like AlphaGo are trained on millions of self-play games whereas world champion probably only played 50,000 games in his lifetime
- Proposed goal: systems should see objects rather than features, build causal models and not just recognize patterns, recombine representations without retraining, and learning-to-learn rather than starting from scratch

14.6 A Review of Robot Learning for Manipulation

- Autonomous manipulation has manyyy applications: hospitals, elder and child care, factories, outer space, restaurants, service, and home
- robots perceive latent object properties by observing outcome of manipulation - interactive perception
- interactive perception is the basis for self-supervised learning
- manipulation tasks exhibits highly hierarchical structure!!
- this structure enables modularity for skills to be mixed and matched together
- tasks that are similar enough to not be considered unique skills is known as a task family
- exploit similarity between tasks to perform them more efficiently
- object-centric representations
- ability to handle novel concepts and unforeseen situations, robot must generalize knowledge
- robot learning can be formulated as an MDP
- common to model environment as a collection of object states

- skills are modelled after the options framework (HRL framework)
- option, $o = I_o, \beta_o, \pi_o$

  I_o = initiation set, indicator function describing when the option may be executed
  β_o = termination condition, describes the probability that an option finishes when reaching stat
  π_o = option policy, maps states to actions, motor skill controller

- discovering reusable skills
- robot needs to learn policies as a function of a context vector τ which encodes extra task-specific information (possibly factored into object)
- representations should capture within- and across task variations
- types of object variations: pose, shape, material properties, interactions / relative properties
- object models can be hierarchical, geometric and non-geometric properties
- point-level representations: point cloud, pixel, voxels
- they are flexible, each element can be associated with additional info
- part-level representations: can lead to better generalization, objects have similar parts that afford similar interactions
- object-level representations: define relative poses, forces, and constraints between objects
- can also be used to define different types of interactions or relations between objects
- passive perception (perceiving environment without exploiting physical interactions), useful for estimating position, shape, and material properties
- interactive perception (robot physically interacts with surroundings), e.g. push object to estimate its constraints or weight, estimate dynamic properties
- allows robot to reduce uncertainty
- robots combine multiple sensor modalities (vision + touch and haptics) and incorporate task information (e.g. instructions)
- transition models can be reused between different tasks, but the new task needs to share the same state, action, and context spaces
- transfer depends on overlap between data distributions
- covariate shift (input varies) and dataset shift (input and output varies)

14.7 Lessons from Amazon Picking Challenge

- Amazon Picking Challenge tests ability of robotic system to fulfill orders by picking items from a warehouse shelf

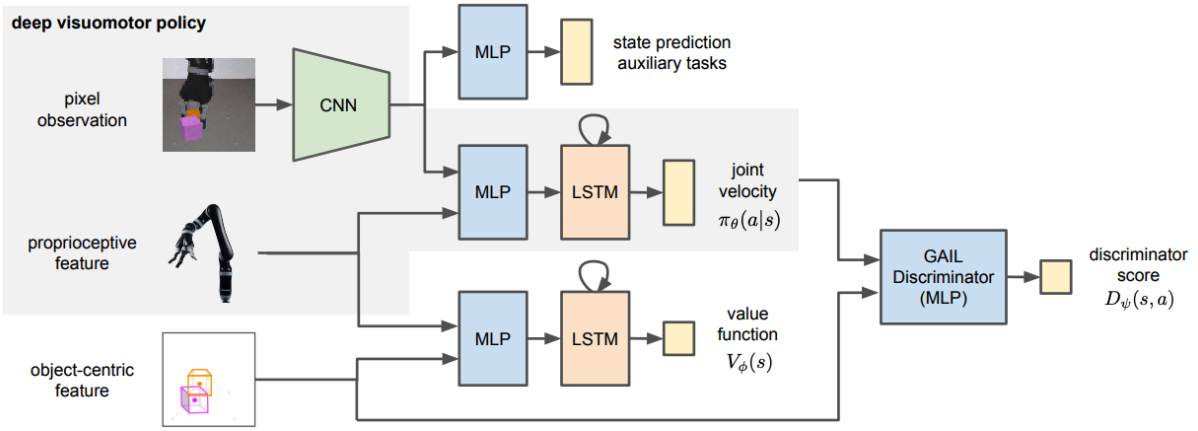
- Build tightly integrated systems and modularize the system by breaking it down into simpler subproblems
- In manipulation it is important to consider alternative embodiments
- Integrate planning with feedback from physical interactions, interactions help reduce uncertainty
- Finding general solutions is desirable but may be infeasible. Try to search for reasonable and useful assumptions to simplify problem
- Challenges: narrow bins and objects were barely visible and partially obstructed, floor made of reflective metal, poor lighting conditions
- Use joint- and task- space feedback controllers
- Use a mobile base to allow the arm to reposition itself to generate easier grasps, but increased the dimensionality of the configuration space
- Simple end-effector that using a suction cup, can pick up most objects, thin shape reduces need to consider complex collision avoidance
- Proper embodiment simplifies the overall solution
- Use a hybrid automaton where states correspond to a feedback controller and state transitions are triggered by sensor events
- Most of the failure cases occurered because of perception inability to discriminate objects properly
- There is recurring underlying structure in robotics problems, making suitable assumptions helps alleviate difficulties of general purpose solutions
- Adding explicit knowledge about physics makes problem more tractable

14.8 Reinforcement and Imitation Learning for Diverse Visuomotor Skills

- Model-free DRL method that leverages small amount of demonstration data to assist RL agent
- Applied to robotic manipulation tasks, end-to-end visuomotor policies that map RGB camera inputs to joint velocities
- RL for robotics, policies must map multi-modal and partial information to control of many DOFs
- Real tasks have contact-rich dynamics and vary along many dimensions, generalization challenge
- Exploration is challenging due to high-dimensional and continuous action space
- Techniques for exploiting priviledged + task specific information to accelerate + stabilize training
- Combine IL with RL using a hybrid reward (imitation reward based on GAIL)

- Also use demonstrated data to create a curriculum by randomizing the start state distribution
- Learn policy and value in separate modalities
- Value function is used in PPO for estimating the advantage to compute policy gradient, instead of using pixels, they use low-level physical states to train value function
- Auxiliary tasks for visual modules
  - improve learning efficiency
  - state-prediction layer used to predict locations of objects from camera observation
  - use fully-connected layer to regress 3D coordinates, minimize l_2 loss
- GAIL discriminator uses object-centric representations (positions of objects), requires some domain knowledge
- Diversify training conditions such as visual appearance, object geometry, and system dynamics to sim2real transfer
- Deep visuomotor policy takes as input RGB observation and proprioceptive features (joint positions and angular velocities)
- GAIL has two networks: a policy network and a discriminator network and uses a min-max objective
- Policy is trained using policy gradient methods to maximum discounted sum of reward
- Employing shaping reward as a trick to facilitate exploration. Task rewards given as a sparse reward at different stages of the tasks, e.g. block stacking - reaching, lifting, stacking
- This is better than hand-crafting a dense shaping reward
- Hybrid reward: $r(s_t, a_t) = \lambda r_{gail}(s_t, a_t) + (1 - \lambda) r_{task}(s_t, a_t)$
- There is a balanced contribution between RL and IL rewards

Figure 14.4: Deep Visuomotor Policy



- Experiments
  - Block lifting, block stacking, pouring liquid, order fulfillment, clearing table
  - Kinova Jaco arm has 9 DOF: 6 arm joints and 3 actuated fingers
  - Used various objects ranging from basic geometric shapes to 3D objects made from primitive shapes
  - Sim2real is still a challenge, large domain gap, transfer is hindered by visual discrepancies, arm dynamics, and physical properties of the environment
  - Certain level of degradation when running on a real robot, zero-shot sim2real transfer
  - On a real robot, there is often a delay in execution of action which is detrimental to robot's performance
  - Fine-tuned agent in simulation subjected to a random chance of dropping actions

14.9 Making Sense of Vision and Touch

- Contact-rich manipulation tasks require haptic and visual feedback
- Use self-supervision to learn multimodal representation of inputs that are used to improve sample efficiency for policy learning
- Evaluated method on peg insertion for different geometry, configurations, and clearances
- Contact-rich tasks: peg insertation, block packing, edge following
- Diverse set of modalities including vision, range, audio, haptic, proprioceptive data and language and often these are complementary of each other
- DL usually requires lot of high-dimensional training data and self-supervision does not rely on having human annotated data
- Their model encodes 3 types of data: RGB, haptic feedback from F/T sensor, and proprioceptive data from joint encoders
- Heterogenuous nature requires domain-specific encoders for each modality
- Generate labels automatically through self-supervision
- The model has to predict 1) the optical flow generated by action and 2) whether end-effector will make contact with environment
- To exploit concurrency between data streams, use a third objective that predicts whether two streams are temporally aligned (binary classification)

Figure 14.5: Multimodal fusion model

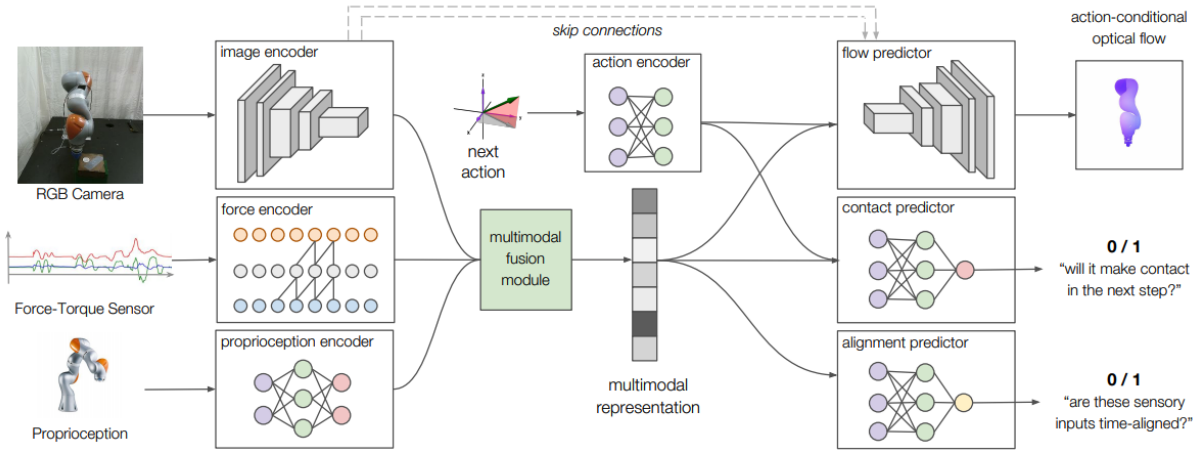
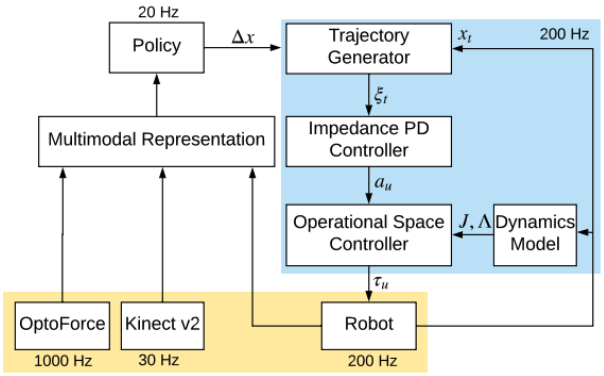


Figure 14.6: Controller

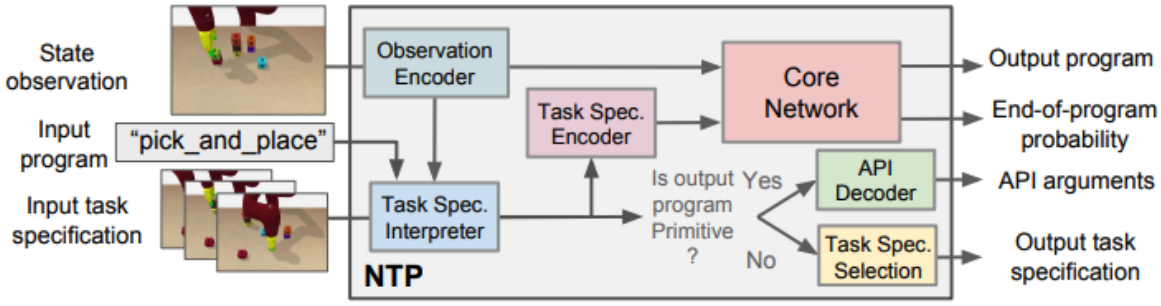


- Model-free removes need for an accurate dynamics model (hard to obtain in presence of rich contacts), use TRPO for policy learning
- Policy network is 2-layer MLP that takes in 128-d multimodal rep and produces a 3D displacementof end-effector
- Policy outputs Cartesian-control commands instead of joint-space commands
- Use direct torque control because it gives robot compliance making it safer
- Also make use of a staged-reward function for subtasks to simplify the challenge of exploration
- Conduct ablation study to learn about importance of each modality, also study robustness of policy in presence of sensor noise and external perturbation (e.g. pushing robot arm)
- Lots of other challenges in real-world: sensor synchronization, variable delays, real-world dynamics, etc

14.10 Neural Task Programming: Learning to Generalize Across Hierarchical Tasks

- few-shot learning from demonstration and neural program induction
- inputs a task specification (e.g. video specification) and recursively decomposes it into finer sub-tasks
- Complex manipulation tasks: object sorting, assembly, de-cluttering, etc
- NTP interprets a task specification and instantiates a hierarchical policy as a neural program
- Task specification can either be a task demonstration recorded as a state-trajectory or even a list of language instructions
- NTP generalizes to 3 kinds of variations in task structure: task length, task topology, and task semantics

Figure 14.7: NTP



- NTP is a meta-learning algorithm, decomposes final objective into simpler objectives recursively and each subtask is assigned a neural program
- NTP extends upon NPI (Neural Programmer-Interpreter)
- NPI is a type of neural program induction in, core of NPI is an LSTM which selects at every timestep the next program to run
- NTP has three parts: Task Specification Interpreter (f_{TSI}), Task Specification Encoder (f_{TSE}), and a core network (f_{CN})

Figure 14.8: NTP Algorithm

Algorithm 1 NTP Inference Procedure

Inputs: task specification ψ, program id i, and environment observation o
function RUN(i, ψ)
  $r \leftarrow 0$, $p \leftarrow M^{prog}_{i,:}$, $s \leftarrow f_{ENC}(o)$, $c \leftarrow f_{TSE}(\psi)$
  while $r < \alpha$ **do**
    $k, r \leftarrow f_{CN}(c, p, s)$, $\psi_2 \leftarrow f_{TSI}(\psi, p, s)$
    $i_2 \leftarrow \arg\max_{j=1...N}(M^{key}_{j,:} k)$
    if program i_2 is primitive **then** ▷ if i_2 is an API
      $\mathbf{a} \leftarrow f_{TSI}(\psi_2, i_2, s)$ ▷ decode API args
      RUN_API(i_2, $\mathbf{a}$) ▷ run API i_2 with args $\mathbf{a}$
    else
      RUN(i_2, ψ_2) ▷ run program i_2 w/ task spec ψ_2
    end if
  end while
end function

- NTP vs NPI: NTP can interpret task specifications and perform hierarchical decomposition, NTP uses APIs as primitive actions and it uses a reactive core network instead of a RNN
- APIs are subroutines for learning at an abstract level, APIs take in specific arguments (e.g. object category or end-effector position)
- APIs used were move_to, grip, and release

14.11 ALFRED: A Benchmark for Interpreting Grounded Instructions for Everyday Tasks

- Action Learning From Realistic Environments and Directives (ALFRED)
- Mapping natural language instructions and vision to actions for household tasks
- Contains long, compositional tasks to bridge gap between research and real-world applications
- Language directives have high-level goals and low-level language instructions
- Baseline model performs poorly on the ALFRED dataset, more room for grounded visual language models
- Lot of platforms for language-driven navigation, embodied QA
- 8k demonstrations, average of 50 steps
- Interact with objects by specifying a pixelwise interaction mask, unlike other settings where they treat localization as a solved problem
- Evaluate on seq2seq model like VLN tasks which is not effective achieving less than 5% success rate
- Need models that can address challenges of language, action, and vision for accomplishing household tasks

- Objects contain multiple visual variations different shapes, textures, and colors
- Tasks are parameterized by object of focus
- Generate expert demonstrations by encoding agent and environment dynamics into PDDL rules
- Split validation and test into *seen* and *unseen* environments to test models generalization to new spaces and novel object classes
- Baseline model: seq2seq architecture
  - CNN encodes visual input
  - bi-LSTM encodes the language directive
  - decoder LSTM infers low-level actions and attends over encoded language
- Model is trained using imitation learning, DAgger-style is non-trivial
- High-level language is concatenated with low-level language with a seperator
- Tasks require reasoning over long sequences of images and instructions, proposed two auxillary losses to add temporal information (progress monitoring)
- Progress monitoring is where the agent maintains an internal estimate of their progress towards the goal and subgoals
- Evaluate both Task Success and Goal-Condition success
- Seq2seq model only achieved 8 percent goal-conditioned success rate
- Also conducted some ablations to investigate unimodal ablation, found that both language and vision was necessary and a single modal is not sufficient to complete the task
- Possible directions: models that exploit hierarchy, are modular, and support structured reasoning and planning

14.12 Causal Discovery with Reinforcement Learning

- Propose to use RL to search for a DAG with the best scoring
- Encoder-decoder takes observable data and generates a graph adjacency matrix
- Reward = predefined score function + two penalty terms to enforce acyclicity
- GES checks acyclicity one edge at a time
- Problem is to use observed data X to infer the underlying causal DAG $\mathcal{G}$
- Modified BIC for score function, used same acyclicity constraint as in NOTEARS
- Overall reward function is $-[\mathcal{C}(\mathcal{G}) + \lambda_1 I(\mathcal{G} \notin \text{DAGs} + \lambda_2 h(A)]$
- Used actor-critic algorithm

Chapter 15

Glossary

15.1 Glossary

Deep Learning Artificial Intelligence Optimization

Self-supervised learning MoCo Contrastive learning

Unsupervised Learning

Learning from demonstration Imitation Learning

Metalearning Dataset bias Ethics and fairness Domain adaptation Graph neural networks

Statistics Dirchlet Processes Chinese Restaurant Process Priors Posterior Likelihood Bayesian inference Bayesian networks Variance Expectation Stick-breaking construction Casual inference

Robotics Robotic perception Filters State estimation Planning Motion control Grasping Kinematics Dynamics Affordances Partial observability Haptics Occlusion Linear v.s. nonlinear control Active perception Manipulation Noise Sensors LiDAR

Extra: Contact dynamics Frictional contact

Vision 3D geometry Representation Learning Detection Segmentation 3D implicit functions multiview geometry Videos Temporal data, reasoning about time Sound Autoencoders Variational Autoencoders Beta-variational Autoencoders

Embodied Learning Navigation Vision and dialog visual navigation interactive navigation

Generative Adversarial Networks

Reinforcement Learning Batch Reinforcement Learning Counterfactual Data Value Iteration Q-learning Deep Deterministic Policy Gradient TD3 SAC Inverse RL Behavioral Cloning Imitation Learning

Variational AEs c-VAEs

Natural Language Processing Conversational Entailment Commonsense Question Answering Everyday Actions in Text Transformers Attention Multihead attention

Probability distributions Gaussians Dirichlet distribution

Clustering Kmeans Spherical Kmeans Nonparametric clustering Bayesian nonparametric clustering

Uncategorized Reward-shaping Auxilary task learning Curiosity-driven learning Hierarchical Reinforcement Learning

ML concepts Precision / recall Entropy Information Gain Decision Trees ROC Curve KNN vs K-means clustering Confusion matrix Regression L1/L2 reg generative vs discriminative models F1-score imbalanced dataset: collect more data, resample, different alg Logistic regression Kernel trick

Causal Discovery score-based methods constraint-based methods constrained-optimization framework Dags with no tears

Good books A Book of Why An Introduction to Probabilistic Graphical Models Michael Jordan Course of probabilistic graphical models - Eric Xing 10-708 (PGM)

Machine Learning: A Probabilistic Perspective CS391R robot learning class

Bibliography

[1] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *CoRR* abs/1311.2524 (2013). arXiv: 1311.2524. URL: http://arxiv.org/abs/1311.2524 (page 19).

[2] Scott Fujimoto, David Meger, and Doina Precup. "Off-Policy Deep Reinforcement Learning without Exploration". In: *CoRR* abs/1812.02900 (2018). arXiv: 1812.02900. URL: http://arxiv.org/abs/1812.02900 (page 31).