10-708: Probabilistic Graphical Models, Spring 2020

1: Introduction

Lecturer: Eric P. Xing *Scribe: Ini Oguntola, Vineet Jain, Samuel Levy, Zihao Chen, Sungjun Choi*

Basic Probability Concepts

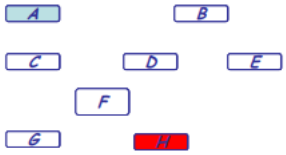


Figure 1: Multiple variables in a graph

- **Representation**: What is the joint probability distribution on multiple variables?

$$P(X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_8) \tag{1}$$

  There are 2^8 state configurations in total. But do they need to be all represented? Do we get any scientific/medical insight? One of the main benefits of graphical models is the cost savings in representing the joint distribution. Modeling the dependencies among the variables with a graph and conditionals can drastically reduce the number of parameters needed to describe the joint distribution, compared to what we would get with a full joint distribution table.

- **Learning**: Where do we get all these probabilities? Should we use maximum likelihood estimation? but how many data do we need for that? Could we use other estimation principles? Where do we incorporate domain knowledge in terms of plausible relationships between variables, and plausible values of the probabilities?

- **Inference**: If not all variables are observable, how do we compute the conditional distribution of latent variables given evidence? Computing $P(H \mid A)$ in Figure 1 would require summing over all 2^6 configurations of the unobserved variables: that requires a lot of compute power.

Multivariate Distribution in High-Dimensional Space

We start with an example from biology and represent cellular signal transduction as follows (Figure 2). Receptors A and B receive signal from cell surface, Kinases C, D and E read and decode the signal; TF F takes in the signal and triggers production of DNA with DNA template and Genes G and H are expressions of the DNA templates.

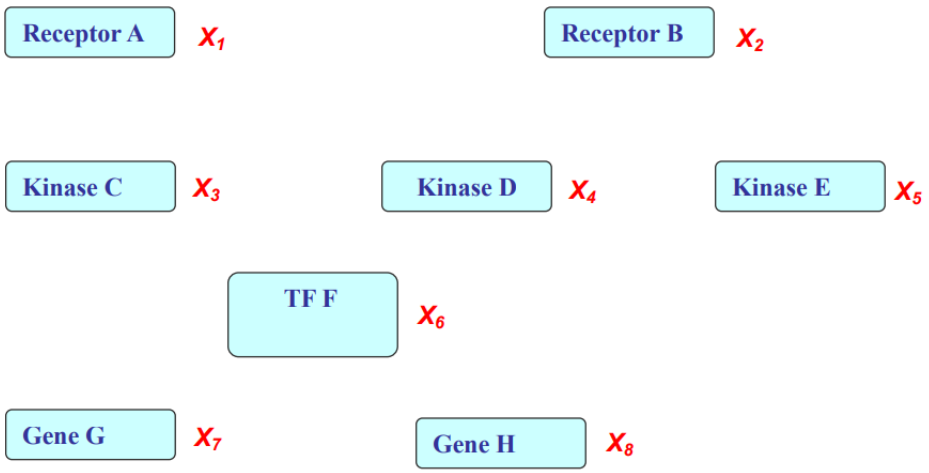


Figure 2: A possible world for cellular signal transduction

Although Figure 2 is a good start to model cellular signal transduction, the additional domain knowledge from biologist can be incorporated to impose a structure on the random variables A, B, C, D, E, F, G and H. Figure 3 partitions the random variab les into compartments they live in within a cell. The dependencies among the variables (nodes) are communication mechanisms and are modeled as edges. This representation allows us to derive the joint probabilities among the random variables using the factorization law.

A Structured View From Domain Experts

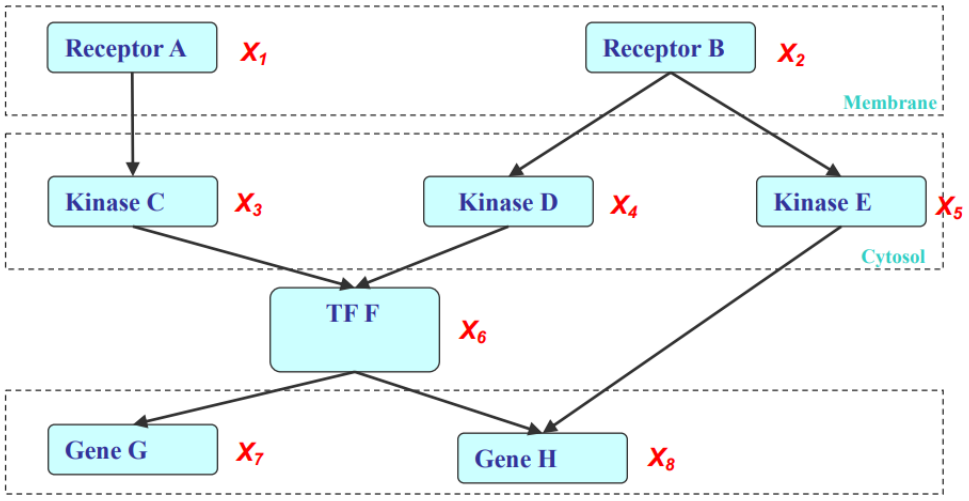


Figure 3: Dependencies among variables are represented in a directed acyclic graph

What are Graphical Models

Informally, a graphical model is a just a graph representing relationship among random variables. Nodes are random variables (features, not examples) and edges (or absence of edges) represent relationships or dependencies among random variables.

The notion of relationship varies depending on the graph. For example, in Figure 4, the graphical model is a representation of co-occurrences within a page between major Biblical figures.

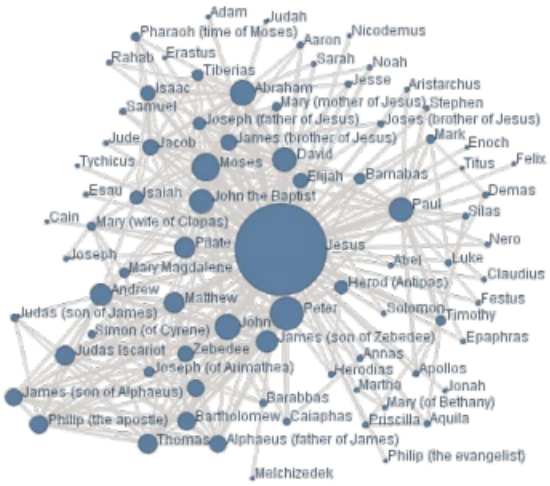


Figure 4: Graphical representation of co-occurrences of Biblical figures within a page

Relationship between Two Random Variables

Rigorously defining each component of a graphical model is crucial in avoiding multiple representations of the same phenomenon by different people, and as part of such effort we first delve into rigorously defining possible relationships between two random variables. The random variables may potentially have many types of relationships (some of which are listed in p.10), and to be rigorous we look for "one-number measures" to serve as summaries that quantitatively represent the presence, absence, or strength of such relationships.

Again, there are many such measures, some of which are listed and discussed below, and each has its adequacies and inadequacies. Choosing a measure is not a trivial task in the sense that, while one can arbitrarily choose one such measure, draw a graph out of data, and provide convincing "stories" out of the graph, unless the measure is chosen rigorously, the argument can be rather easily overthrown by counter-examples from the same data. It is thus vital to understand what each measure entails.

Pearson's Correlation

Pearson's correlation (denoted as ρ) is one of the most well-known and fundamental measures of association between random variables that is defined as follows:

$$\rho(X,Y) = \frac{\mathrm{Cov}(X,Y)}{\sqrt{\mathrm{Var}(X)}\sqrt{\mathrm{Var}(Y)}}$$

Still, there are two important caveats to note:

1. Two independent variables are uncorrelated; however, the reverse does not hold. For example, let random variables X, Y be such that $X \sim U[-1, 1]$ and $Y = X^2$. Then while it is evident that Y (deterministically) depends on X, they are uncorrelated since:

$$\begin{aligned}\text{Cov}(X, Y) &= \mathbb{E}[XY] - \mathbb{E}[X]\,\mathbb{E}[Y] \\ &= \mathbb{E}[XY] \quad (\because \mathbb{E}[X] = 0) \\ &= \mathbb{E}[X^3] = 0 \quad (\because X \text{ is not skewed})\end{aligned}$$

2. Pearson's correlation only captures linear dependence, as can be seen from the example above. This in turn means Pearson's correlation is very weak in terms of capturing independence.

Strong(er) Measures of Association

The limitations of Pearson's correlation introduced above calls for stronger measures, ones that can capture non-linear dependences and thus independences. In fact, for the following two measures we bring in the very definition of statistical independence between random variables to construct measures.

Exploiting the fact that the joint density P_{XY} of two jointly-distributed random variables X and Y can be factorized as $P_X P_Y$ if and only if they are independent of each other, we quantify the "distance" between the joint density P_{XY} and the product of marginals $P_X P_Y$. Indeed, this approach guarantees that the distance $= 0$ iff X and Y are independent.

Mutual Information

One of the most common measures of distance between two densities P and Q is **Kullback-Leibler divergence**, or KL-divergence in short:

$$KL(P, Q) = \int_{x \in \mathcal{X}} P(x) \log \frac{P(x)}{Q(x)}\, dx$$

KL divergence returns 0 when P and Q are equal, i.e. $P(x) = Q(x)$, $\forall x \in \mathcal{X}$, and a larger positive value as P and Q deviate further from each other. Since we likewise want the distance to be 0 when $P_{XY}(x, y) = P_X(x)P_Y(y)$, $\forall (x, y) \in \mathcal{X} \times \mathcal{Y}$ and positive otherwise, we can utilize KL divergence to obtain our desired measure, known as **mutual information**:

$$I(X, Y) = KL(P_{XY}, P_X P_Y)$$

This measure indeed successfully captures non-linear dependences. However, it poses **computational issues** since integration over complex combination of non-Gaussian, multi-modal, and possibly even non-parametric densities is a significant challenge.

Hilbert-Schmidt Independence Criterion (HSIC)

A recent finding that also captures non-linear dependences is HSIC(Gretton et al. 2005). It's defined as the **maximum mean discrepancy (MMD)** between joint density P_{XY} and product of marginals $P_X P_Y$.

Definition of MMD:
Let P, Q be any two densities,

$$\begin{aligned} MMD(P,Q) &= ||\mu_k(P) - \mu_k Q||_{\mathcal{H}_k} \\ \mu_k(P) &= \mathbb{E}_{Z\sim P}[\phi(Z)] \qquad \text{(kernel embedding of } P\text{)} \\ \phi(Z) &= \text{feature map of kernel k} \end{aligned}$$

One important property of this measure is $HSIC(X,Y) = 0$ if and only if $X \perp Y$. More details will be covered in future lectures.

Towards Graphical Models: Partial Correlation

The measures of association between two random variables discussed in the previous sections can be used to define a **marginal correlation/dependency** graph. This is the most primitive form of graphical model in which we connect any pair of variables with a non-trivial pairwise correlation or mutual information or HSIC.

The drawback is that this type of graphical model is not very informative due to the reason that two random variables will have non-zero measure of association very rarely. We can almost always find some statistical association between a pair of variables, either due to some underlying process that affects both variables or sometimes due to random chance.

Consider the following example: define, X = height of kid, Y = vocabulary of kid, Z = age of kid. If we compute a pairwise measure of association between these variables, we expect to find all of them to be non-zero. However, we know from 'common sense' that the height of the kid and the vocabulary has no direct relation, rather the age of the kid is the underlying variable affecting both these values. In this case, a marginal dependency graph will have edges between all pairs of variables, but we can find a more informative structural relationship.

Partial Correlation

We can define a new measure of correlation between two variables **given another variable**. We can think of it as the correlation measured between two variables X and Y after conditioning on another variable Z, or after eliminating the linear effect of Z. This is known as the **partial/conditional correlation**.

$$\begin{aligned} \rho(X,Y|Z) = \rho(e_X, e_Y) &= \frac{\text{Cov}(e_X, e_Y)}{\sqrt{\text{Var}(e_X)}\sqrt{\text{Var}(e_Y)}} \\ e_X &= X - (\beta_X^T Z + \text{intercept}_X) \\ e_Y &= Y - (\beta_Y^T Z + \text{intercept}_Y) \end{aligned}$$

It is the correlation between the residuals from regressing Z to X and Z to Y linearly. In this sense, it is similar to Pearson's correlation.

$$\begin{aligned} X \perp\!\!\!\perp Y \mid Z &\implies \rho(X,Y|Z) = 0 \\ \rho(X,Y|Z) = 0 &\not\implies X \perp\!\!\!\perp Y \mid Z \end{aligned}$$

Partial Correlation Graph

We can now construct a more meaningful graphical model than the marginal dependency graph. We connect a pair of variables if they have non-trivial partial correlation given the rest of the variables.

One possible issue with this model is that it is computationally expensive to compute the partial correlation for every pair of variables conditioned on all the rest, since we need to first fit a (linear) regression for each of the conditioned variables. However, it turns out the partial correlation matrix R has a simple form related to the inverse covariance matrix Θ.

$$R_{ij} = \rho(X_i, X_j | X_{-ij})$$
$$R_{ij} = -\frac{\Theta_{ij}}{\sqrt{\Theta_{ii}}\sqrt{\Theta_{jj}}}$$

Conditional Independence

As revealed in previous sections, it's always helpful to reduce statistical and computational complexity if we can point out conditional independence. The classical notation for conditional independence is $X \perp Y|Z$, X, Y, Z are random variables. And we have the definition:

$$X \perp Y|Z \iff P(X, Y|Z) = P(X|Z)P(Y|Z)$$

It's a hard mission to extract conditional independence if we want to use strong dependency measures or partial correlation as a tool. One shortcut is simply impose Gaussian assumption to the random variables of interest. To be detailed, suppose (X, Y, Z) are jointly Gaussian, we have $\rho(X, Y|Z) = 0$ iff $X \perp Y|Z$. Many papers rely on this fact though may not state it explicitly.

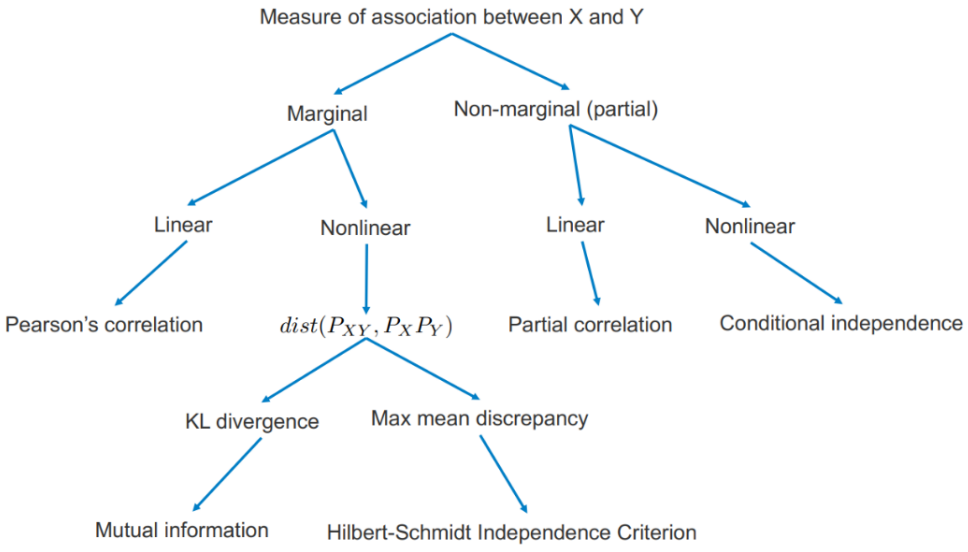


Figure 5: Summary of pairwise measures of association.

Why Graphical Models?

"Graphical models" really refer to a way of thinking, not necessarily to any particular individual model. They are a language for communication, computation, and development.

Probability theory provides the *glue* whereby the parts are combined, ensuring that the system as a whole is consistent, and providing ways to interface models to data.

The *graph theoretic* side of graphical models provides both an intuitively appealing interface by which humans can model highly-interacting sets of variables as well as a data structure that lends itself naturally to the design of efficient general-purpose algorithms.

Many of the classical multivariate probabilistic systems studied in fields such as statistics, systems engineering, information theory, pattern recognition and statistical mechanics are *special cases of the general graphical model formalism.*

The graphical model framework provides a way to view all of these systems as instances of a *common underlying formalism.*

10-708: Probabilistic Graphical Models, Spring 2020

2: Representation of undirected graphical models

Lecturer: Eric P. Xing *Scribe: T. Wörtwein, H. Lee, V. Rawal, M. Ferdosi*

Building a conditional independence graph (CIG) based on the dependencies of every possible pair of random variables quickly becomes infeasible. Therefore, today we will try something (potentially) easier than building a graph from the bottom up based on observations. Given some graph, how can we understand the model that is represented by it? How can we read the conditional independencies from the graph? What is the class of distributions represented by CIG?

1 Undirected graphical models (UGM)

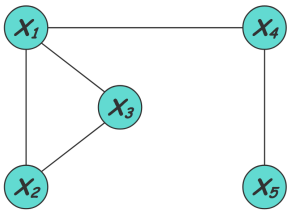


Figure 1: Example of UGM

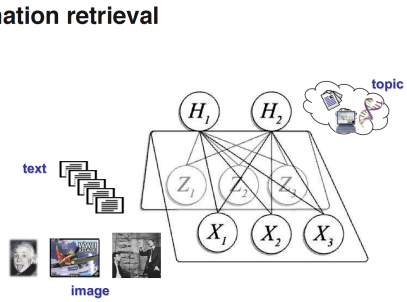


Figure 2: Example Use of UGM in Information Retrieval

Nodes correspond to random variables, while edges correspond to pairwise (non-causal) relationships. Undirected graphical models are $P(\boldsymbol{X}, \theta_G)$, i.e. probability distributions over random variables $\boldsymbol{X}$, whose parameters are determined by the graph G.

Computer vision example (is this blue patch air or water?): Build a grid model of patches in an image, and put a probability distribution on top, based on the assumption that connected nodes are likely to be of the same value.

More examples of UGM: physics model, social networks, protein interaction networks, modeling Go, ...

In domains such as information retrieval, such models can be used to describe relationships between concepts and also entities. For example, in the above figure, this graph models the correlations or alignments between images and texts and how they're aligned towards the hidden topics which give rise to certain words or certain image configurations in a dataset. This leads us to another type of graphical model.

2 Representation of undirected graphical model

An **undirected graphical model** represents a distribution $P(X_1, X_2, ..., X_n)$ defined by an undirected graph H, and a set of positive **potential functions** Ψ_c associated with the cliques of H, such that :

$$P(x_1, x_1, x_2, ..., x_n) = \frac{1}{Z} \prod_{c \in C} \Psi_c(X_c)$$

where Z is known as the partition function:

$$Z = \sum_{x_1, x_2, ..., x_n} \prod_{c \in C} \Psi_c(X_c)$$

Given a graph, identify all the "Cliques" present within the graphical model. This is also known as **Markov Random Fields, Markov Networks, ...** The **potential function** can be understood as a contingency function of its arguments assigning "pre-probabilistic" score of their joint configurations. They need to be positive to avoid producing a negative probability upon multiplication. Normalizing constant Z is used to convert it into a probability distribution. It is the sum of potential values over all possible Random Variable configurations.

2.1 Quantitative Specification : Cliques

For G = {V,E}, a complete subgraph (clique) is a subgraph $G' = \{V' \subseteq V, E' \subseteq E\}$ such that nodes in V' are fully connected. A maximal Clique is a complete subgraph such that any superset $V'' \supset V'$ is not a clique. A sub-clique is a not-necessarily-maximal clique.

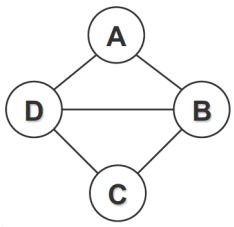


Figure 3: Example Graph

In this example, max-cliques are {A,B,C,D}, {B,C,D}, and sub-cliques are {A,B}, {C,D}, ... all edges and singletons.
Reason for using cliques : Cliques are basic units that capture all possible dependencies that are possible and not to be missed. If we start building from the sub-graphs within the cliques, and begin inter-connecting them, we may risk losing modeling some inter-dependencies.

2.2 Gibbs Distributions

The following expression in terms of potential functions is also called Gibbs Distribution.

$$P(x_1, x_1, x_2, ..., x_n) = \frac{1}{Z} \prod_{c \in C} \Psi_c(X_c)$$

2.3 Interpretation of Clique Potentials

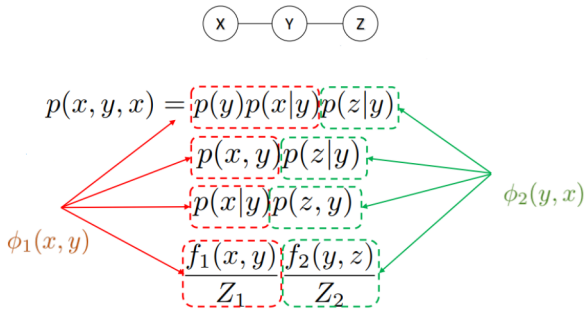


Figure 4: Interpreting Clique Potentials

Clique potentials are pre-probabilistic, contingency functions offering ways to recover or specify biases on configurations over Random Variables.

In directed graphical models, the joint distribution over the vertices could be factorized as a product of marginal and conditional distributions. In undirected graphical models however, the joint distribution can be factorized into a product of clique potentials. However, these clique potentials are not necessarily marginals or conditionals. They only represent a notion of "goodness" or "compatibility" of the variables. To illustrate this, consider the graph shown in the Figure. While the graph implies $X \perp\!\!\!\perp Z|Y$ and hence, the joint distribution can be represented as $p(x,y,z) = p(y)p(x|y)p(z|y)$, it can also be written in other forms as shown in the figure.

2.4 Examples of Undirected Graphs and Gibbs Representations

The potential functions shown here are symbolic. In order for the clique function to be implemented in a programming language, one would have to use a 3D matrix representation given that we have 3 binary variables at a time. (when using max-cliques as below). So, for discrete nodes, one can represent $P(X_{1:4})$ as two 3D tables instead of one 4D table.

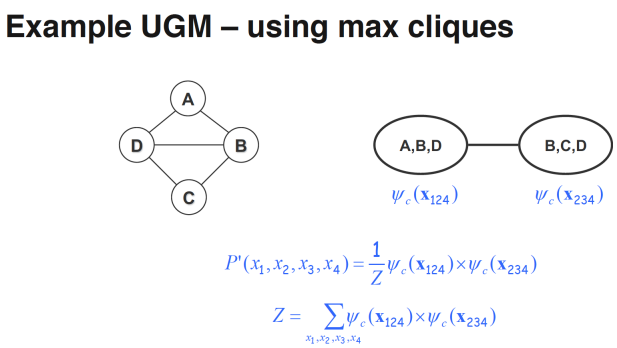


Figure 5: Example UGM : Using max-cliques

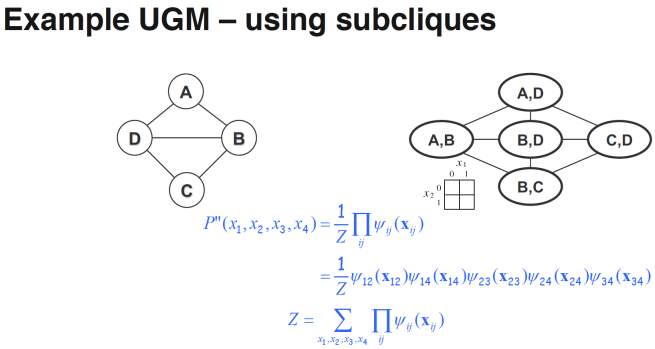


Figure 6: Example UGM : Using sub-cliques

Instead of using the max-cliques, one could also alternatively use the sub-cliques which gives us a different way of specification. However, one may lose something on doing so : it may certainly be possible for one to have some tricky configurations in the triplets representation which are not necessarily recoverable (or atleast easily recoverable) using the pairwise potential representation. So, in this way, one can represent $P(X_{1:4})$ as 5 2D tables instead of one 4D table such as using pair-MRFs, a popular and simple special case.

There is another form of canonical representation which allows one to be over-complete because it uses the potential functions from all possible cliques existent in the graph : triplet, pairwise, and also the singleton cliques. This offers us the ultimate flexibility but the model becomes a little more complicated. The canonical form is thus the most expensive and the richest form and it subsumes the previous two cases.

3 Independence Properties

Now we ask what kinds of distributions, in terms of the set of independence relationships between variables, can be represented by undirected graphs (ignoring the details of the particular parameterization). How do we define placeholders or family of graphs to represent a given set of independence assumptions?

3.1 Definition : Global Markov properties

Global Markov properties of an Undirected Graph H are

$I(H) = \{(X \perp\!\!\!\perp Z|Y) : sep(X; Z|Y)\}$

i.e. The sets X and Y of random variables are independent given the set Z if we have that the two sets are separated from each other by the set Z.

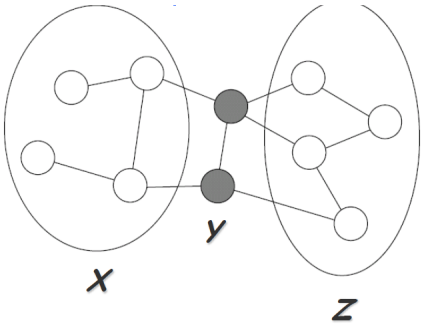


Figure 7: Global Markov Properties

4 I-maps

We define I-maps to establish a formal relationship between a graph H and a distribution P using the conditional independence definition. This will make it easier to describe all conditional independencies of P by representing them as a graph.

Definition: Let P be a distribution over X. We define I(P) as the set of all independence assertion of the form $X \perp\!\!\!\perp Z|Y$ that hold in P (independent of the parameter-values).

X	Y	P(X, Y)
1	1	0.64
1	0	0.16
0	1	0.04
0	0	0.16

Table 1: Are X and Y independent?

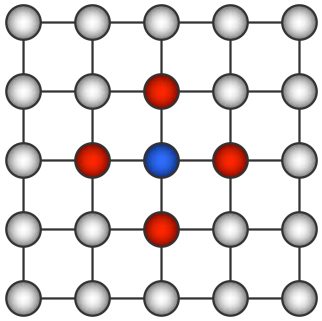


Figure 8: Nodes marked in red are the Markov blanket for the blue node.

Definition: Let H be a graph and $I(H)$ its set of independence assertions. $I(H)$ is an I-map for $I(P)$, if $I(H) \subseteq I(P)$.

If $I(H)$ is an I-map for $I(P)$, it cannot contain independence assertions which are not present in $I(P)$. The contingency Table 1 demonstrates why it is harder to determine conditional independence directly from P compared to from a graph.

4.1 Global Markov Independencies

A probability distribution P satisfies the global Markov property of the undirected graph H, if for any disjoint X, Y, and Z such that Y separates X and Z, X is independent of Z given Y:

$$I(H) = \{X \perp\!\!\!\perp Z|Y : \text{sep}_H(X; Z|Y)\}$$

$I(H)$ is a I-map of P.

4.2 Local Markov Independencies

Definition: The set of neighboring nodes of a node X_i in an undirected graph is called Markov blanked (denoted as MB_{X_i}).

Definition: The local Markov independencies associated with H are:

$$I_l(H) = \{X_i \perp\!\!\!\perp V - \{X_i\} - MB_{X_i}|MB_{X_i} : \forall i\}$$

Where B is the set of all nodes in H.

This means that X_i (the blue node in Figure 8) is independent of all other nodes in H (the white nodes in Figure 8) given its Markov blanket MB_{X_i} (the red nodes in Figure 8), i.e., all neighbors of X_i.

4.3 Soundness and Completeness of global Markov Independencies

P is a Gibbs distribution over H, if it can be represented as a normalized product over all potential functions defined on the cliques of H:

$$P(X) = \frac{1}{Z} \prod_{c \in C} \Psi_c(X_c)$$

Soundness (from graph to distribution): If P is a Gibbs distribution over H, then H is guaranteed to be an I-map of P. Given a graph H, we can find a distribution P such that H is an I-map of P, by letting H be a Gibbs distribution over H.

Completeness: If $\text{sep}_H(X; Z|Y)$ not in $I(H)$ (implies that X and Y are dependent given Z), there are still some P that factorize over H in which X is independent of Z given Y ($X \perp_P Z|Y$). Intuitively, this can happen when the concrete probability numbers are numerically independent. Despite that, the reverse (from a distribution to a graph) is 'almost always' possible.

There is no strict equivalence between graphs and distributions!

Definition: The pairwise Markov independencies associated with UG $H = (V, E)$ are

$$I_p(H) = \{X \perp Y | V \setminus \{X, Y\} : \{X, Y\} \notin E\}$$

4.4 Hammersley-Clifford Theorem

Hammersley-Clifford Theorem: Let P be a positive distribution over V, and H a Markov network graph over V. If H is an I-map for P, then P is a Gibbs distribution over H.

Sometimes H and P are perfectly equivalent.

Definition: A Markov network H is a perfect map for P if for any X, Y, Z, we have that

$$\text{sep}_H(X; Z|Y) \iff P(X \perp Z|Y)$$

Theorem: Not every distribution has a perfect map as UGM.

4.5 Exponential Form

Constraining clique potentials to be positive can be inconvenient (e.g. sometimes you want to model physical phenomenons with +1, -1). We represent a clique potential $\psi_c(X_c)$ in an unconstrained form using a real-value "energy" function $\phi_c(X_c)$: $\psi_c(x_c) = \exp(-\phi_c(x_c))$. The joint probability has a nice additive structure

$$p(x) = \frac{1}{Z} \exp\left\{-\sum_{c \in C} \phi_c(x_c)\right\} := \frac{1}{Z} \exp\{-H(x)\}$$

$H(x)$ is free energy. This is called the Bolzmann distribution in physics, and a log-linear model in statistics.

5 Boltzmann machines

Definition: A fully connected graph with parwise(edge) potentials on binary-valued nodes (-1,1).

$$P(x_1, x_2, x_3, x_4) = \frac{1}{Z} \exp\left\{\sum_{ij} \phi_{ij}(x_i, x_j)\right\} = \frac{1}{Z} \exp\left\{\sum_{ij} \theta_{ij} x_i x_j + \sum_i \alpha_i x_i + C\right\}$$

The overall energy function has the form of $H(x) = \sum_{ij}(x_i - \mu)\theta_{ij}(x_j - \mu) = (x - \mu)^\top \Theta (x - \mu)$. Why is this useful : we will cover a technique to learn this model to recover the structure of the graph from data. Graph becomes interesting especially if Θ is sparse. If $\Theta_{ij} = 0$, then there is no edge between x_i and x_j.

6 Ising Models

The concept of the Ising model comes from statistical physics as a model for modeling the energy of a physical system consisting of interactions of atoms which can be represented as nodes in an undirected graph. In the Ising model, nodes are arranged in a regular topology (often as a grid) and connected only to their immediate neighbors. It can be seen as a sparse Boltzmann Machine. We can also define a multi-state Ising model (called Potts model), in which nodes can take multiple values instead of just binary values. One example of Ising model is shown in the following figure:

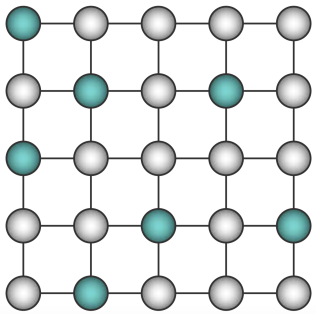


Figure 9: Example of an Ising model in which each node is connected to its neighbouring node

Its probability distribution can be written as:

$$P(X) = \frac{1}{Z} \exp\{\sum_{i,j \in N_i} \theta_{ij} X_i X_j + \sum_i \theta_{i0} X_i\}$$

7 Restricted Boltzmann Machine (RBM)

The Restricted Boltzmann Machine (RBM) is inspired by the Boltzmann Machine. We can model it as a bipartite graph consisting of visible units and hidden units. One example is shown below:

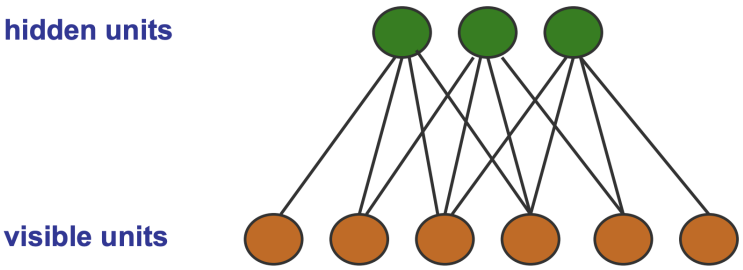


Figure 10: Example of RBM as an undirected graph wit visible and hidden nodes

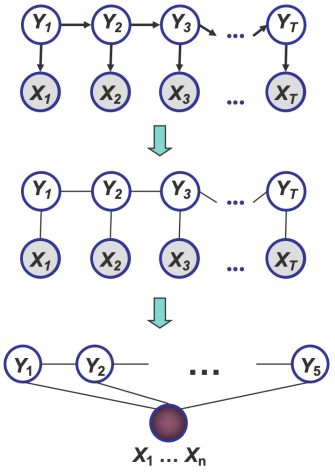


Figure 11: Example of CRF

Its probability distribution can be written as:

$$p(x,h|\theta) = \exp\big\{\sum_i \theta_i\phi_i(x_i) + \sum_j \theta_j\phi_j(h_j)\sum_{i,j}\theta_{i,j}\phi_{i,j}(x_i,h_j) - A(\boldsymbol{\theta})\big\}$$

Some of the properties of RBM: Factors are marginally dependent. Factors are conditionally independent given observations on the visible nodes. It enables iterative Gibbs sampling. There are some learning algorithms for its estimators.

The conditional probability distribution of one layer given the other layer can be factorized:

$$p(H|X) = \prod_{i=1}^{n} p(h_i|X)\ ,\ p(X|H) = \prod_{i=1}^{m} p(x_i|H)$$

The marginal distribution of hidden and visible layers can be defined as:

$$p_{ind}((H)) \propto \prod_j \exp\{\theta_j g_j(h_j)\}\ ,\ p_{ind}((X)) \propto \prod_i \exp\{\theta_i g_j(x_i)\}$$

Using f and g, we can write the joint distribution in the following form:

$$p(x,h|\theta) = \exp\big\{\sum_i \theta_i f_i(x_i) + \sum_j \lambda_j g_j(h_j)\sum_{i,j} f_i^t(x_i)\mathbf{W}_{i,j}g_j(h_j)\big\}$$

8 Conditional Random Field (CRF)

CRFs are undirected graph representation in which we can encode conditional distributions $P(Y|X)$, where Y_i are target variables and X is the set of observed variables.

The probability distribution could be written as:

$$P_\theta(y|x) = \frac{1}{Z(\theta,x)\exp\{\sum_c \theta_c f_c(x,y_c)\}}$$

10-708: Probabilistic Graphical Models, Spring 2020

3: Directed Graphical Models (Bayesian Networks)

Lecturer: Eric P. Xing Scribe: Suman Pokharel, Wendy Yang, Donghui Yan, Jingjing Tang, Wenhuan Sun

1 Two Types of Graphical Models

1.1 Directed Graphical Models

In directed graphical models, nodes that represent random variables are connected by directed edges, which represent causality relationships between nodes. This type of directed GM is called Bayesian Network or Directed Graphical Model.

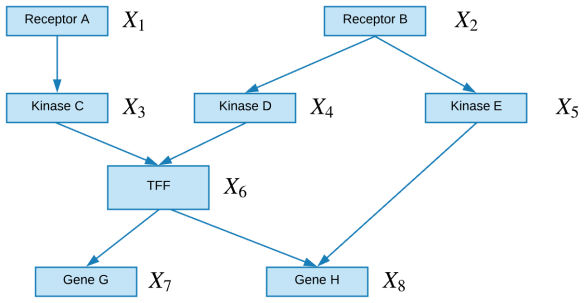


Figure 1: Directed Graph

For example, in the Directed GM above, the underlying joint probability can be written as:

$$P(X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_8) = P(X_1)P(X_2)P(X_3|X_1)P(X_4|X_2)P(X_5|X_2)P(X_6|X_3, X_4)P(X_7|X_6)P(X_8|X_5, X_6)$$

1.2 Undirected Graphical Models

In undirected graphical models, nodes are connected by undirected edges, which represents correlations between nodes/variables. This type of GM is called Markov Random Field or Undirected Graphical model.

For example, in the undirected graphical models below, the underlying joint probability can be written as $P(X_1, X_2, ..., X_7, X_8) = 1/Zexp\{E(X_1) + E(X_2) + E(X_3, X_1) + E(X_4, X_2) + E(X_5, X_2) + E(X_6, X_3, X_4) + E(X_7, X_6) + E(X_8, X_5, X_6)\}$.

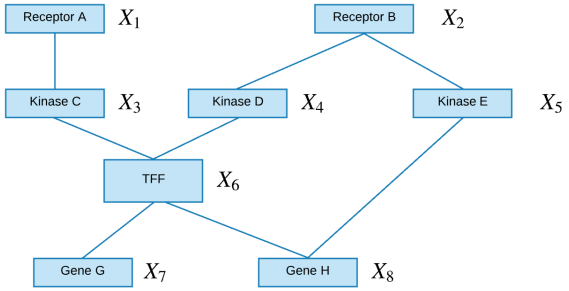


Figure 2: Undirected Graph

2 Examples

2.1 Expert Systems

Expert systems are good example of the application of directed graphical models, where the expert knowledge will be encoded as directed edges between nodes. For example, according to the ALARM network, a patient monitoring system, representing causal relationships presented by Beinlich et al. 1989, expert medical knowledge was encoded as directed edges between nodes that represent random variables (e.g. measurements (blood pressure, heart rate, respiratory rate, etc) and queries (presence of a disease)). In this case, inference is easier with such directed graphical model (e.g. P(kinked tube = true — measurements)).

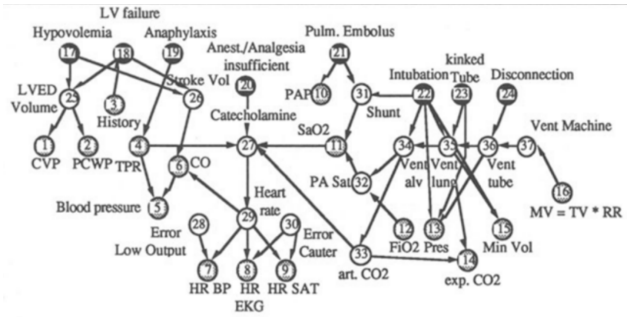


Figure 3: The ALARM network representing causal relationships, Beinlich et al. 1989

2.2 Dishonest Casino

Domain knowledge and knowledge engineering: Let discrete random variable x be the outcome of dice-rolling taking values from $[1, 2, 3, 4, 5, 6]$ and categorical random variable y be the choice of the dice taking values from [fair, loaded]. In this case, x is an observed discrete variable, y is a hidden categorical variable. There exists some causal relationships between x_i and y_i, as well as between different y_i. For example, $P(x_i|y_i = fair)$ and $P(y_{i+1}|y_i)$

Two dices were used in a casino. The fair one has equal probability for each number ($P(x = i) = \frac{1}{6}, i \in [1, 6]$), and the loaded/unfair one has the following probability distribution: $P(x = i) = \frac{1}{10}$ if $i \neq 6$, else 0.5.

Given a sequence of 50 observed rolls (e.g. 1245.....666...2344), 3 types of question could be asked:

1. Evaluation: how likely is this sequence, given our knowledge/model of how the casino works? (e.g. P(observed sequence | domain knowledge))

2. Decoding: what portion of the sequence was generated with the fair die, and what portion with the loaded die? (e.g. P(choice of dice | observed sequence))

3. Learning: how 'loaded' is the loaded die? How 'fair' is the fair die? How often does the casino player change from fair to loaded, and back? (e.g. P(choice of dice))

A Hidden Markov Model can be used to model this casino problem, where the sequence is $\mathbf{x} = x_1, x_2, ..., x_T$ and the parse $\mathbf{y} = y_1, y_2, ..., y_T$.

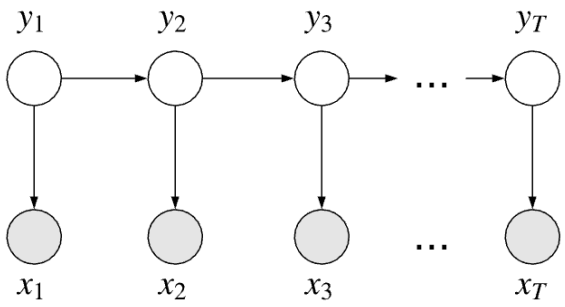


Figure 4: Hidden Markov Model for Casino

The probability of this parse can be answered as:

Joint probability $p(\mathbf{x},\mathbf{y}) = p(x_1, x_2, ..., x_T, y_1, y_2, ..., y_T) = p(y_1) \prod_{t=2}^{T} p(y_t|y_{t-1}) \prod_{t=1}^{T} p(x_t|y_t)$

$= p(y_1, y_2, ..., y_T)p(x_1, x_2, ..., x_T|y_1, y_2, ..., y_T)$

Marginal probability $p(\mathbf{x}) = \sum_y p(\mathbf{x},\mathbf{y}) = \sum_{y_1} \sum_{y_2} ... \sum_{y_N} \pi_{y_1} \prod_{t=2}^{T} a_{y_{t-1},y_t} \prod_{t=1}^{T} p(x_t|y_t))$

Posterior probability $p(\mathbf{y}|\mathbf{x}) = p(\mathbf{x},\mathbf{y})/p(\mathbf{x})$

In the case of calculating marginal probability, k^N summation operations are needed, where k represents the number of possible value of the categorical variable y. This could be improved to polynomial time.

3 Bayesian Networks

- A BN is a directed graph model whose nodes represent the random variables and whose edges represent directed influence among or between random variables.

- It is a data structure that provides the skeleton for representing **a joint distribution** compactly in a **systematic factorized** way.

- It offers a compact representation for **a set of conditional independence assumptions** about a distribution.

- We can view the graph as encoding a **generative sampling process** executed by nature, where the value for each variable is selected by the nature using a distribution that depends only on its parents.

In other words, each variable is a stochastic function of its parents.

3.1 Bayesian Network: Factorization Theorem

Theorem: Given a DAG, the most general form of the probability distribution that is **consistent with** the graph factors according to "node givens its parents":

$$P(\mathbf{X}) = \prod_{i=1}^{d} P(X_i|\mathbf{X}_{\pi_i})$$

where $\mathbf{X}_{\pi_i}$ is the set of parents of X_i, d is the number of nodes (variables) in the graph.

Example:

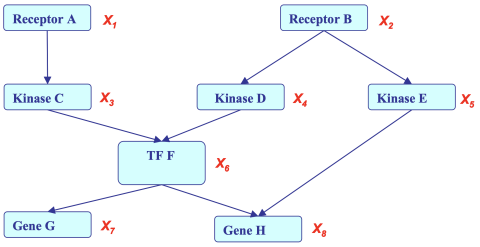


Figure 5: Directed Graph

The joint probability of the above directed graph can be written as:

$$P(\mathbf{X}_1, \mathbf{X}_2, \mathbf{X}_3, \mathbf{X}_4, \mathbf{X}_5, \mathbf{X}_6, \mathbf{X}_7, \mathbf{X}_8) =$$
$$P(\mathbf{X}_1)P(\mathbf{X}_2)P(\mathbf{X}_3|\mathbf{X}_1)P(\mathbf{X}_4|\mathbf{X}_2)P(\mathbf{X}_5|\mathbf{X}_2)P(\mathbf{X}_6|\mathbf{X}_3, \mathbf{X}_4)P(\mathbf{X}_7|\mathbf{X}_6)P(\mathbf{X}_8|\mathbf{X}_5, \mathbf{X}_6)$$

3.2 Specification of a directed GM

There are two components to any GM:

- the **qualitative** specification
- the **quantitative** specification

3.2.1 Qualitative Specification

Where does the qualitative specification come from?

- Prior knowledge of causal relationships
- Prior knowledge of modular relationships

- Assessment from experts
- Learning from data
- We simply like a certain architecture (e.g. a layered graph)
- ...

Is there a concise, unambiguous, and rigorous meaning/interpretation?

- Conditional independence between variables!

3.3 Local Structures and Independence

- Common parents
  - Fixing B **decouples** A and C
  - "Given the level of gene B, the levels of A and C are independent"
  - Expression: $P(A, C|B) = P(A|B)P(C|B)$

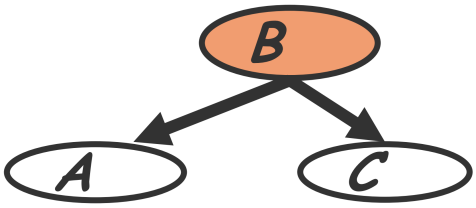


Figure 6: Common Parent Example

- Cascade
  - Knowing B **decouples** A and C
  - "Given the level of gene B, the levels of A provides no extra prediction value for the level of gene C"
  - Expression: $P(A, B, C) = P(A)P(B|A)P(C|B)$

Figure 7: Cascade Example

- V-structure
  - Knowing C **decouples** couples A and B because A can "explain away" B w.r.t C
  - "If A correlates to C, then chance for B to also correlate to B will decrease"
  - Expression: $P(A, B) = P(A)P(B)P(A, B|C)$
- The language is compact, the concept are rich!

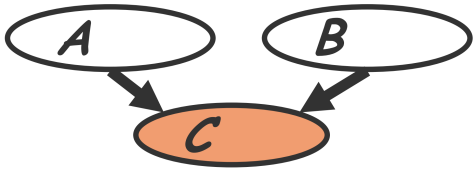
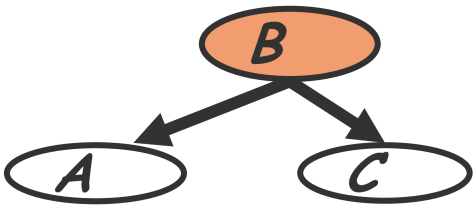


Figure 8: V-structure Example

3.4 A simple justification



Equation:
$P(A, C|B) = \frac{P(A,B,C)}{P(B)} = \frac{P(B)P(A|B)P(C|B)}{P(B)} = P(A|B)P(C|B)$

4 I-Maps

We use I-maps to establish the relationship between graph and distribution. A distribution $\mathcal{P}$ satisfies the local independencies associated with a graph $\mathcal{G}$, if and only if $\mathcal{P}$ is representable as a set of Conditional Probability Distributions (CPDs) associated with the graph $\mathcal{G}$.

Independencies associated with a distribution $\mathcal{P}$

Definition: Let $\mathcal{P}$ be a distribution over $\mathcal{X}$. We define $\mathcal{I}(\mathcal{P})$ to be the set of independence assertions of the form $(\mathbf{X} \perp \mathbf{Y}|\mathbf{Z})$ that hold in $\mathcal{P}$.

I-Map

Definition: Let $\mathcal{K}$ be any graph object associated with a set of independencies $\mathcal{I}(\mathcal{K})$. We say $\mathcal{K}$ is an I-map for a set of independencies $\mathcal{I}$ if $\mathcal{I}(\mathcal{K}) \subseteq \mathcal{I}$.

Corollary: $\mathcal{G}$ is an I-map for $\mathcal{P}$ if $\mathcal{G}$ is an I-map for $\mathcal{I}(\mathcal{P})$, where we use $\mathcal{I}(\mathcal{G})$ as set of independencies associated

4.1 Facts about I-maps

For $\mathcal{G}$ to be I-map of $\mathcal{P}$, it is necessary that $\mathcal{G}$ does not mislead us regarding independencies in $\mathcal{P}$:

any independence that $\mathcal{G}$ asserts must also hold in $\mathcal{P}$. Conversely, $\mathcal{P}$ may have additional independencies that are not reflected in $\mathcal{G}$

Example

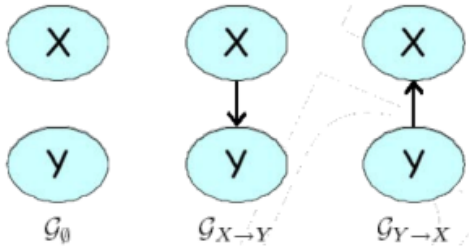


Figure 10: I-map example

X and Y are independent in graph $\mathcal{G}_\emptyset$ only.

P1

X	Y	$P(X,Y)$
x^0	y^0	0.08
x^0	y^1	0.32
x^1	y^0	0.12
x^1	y^1	0.48

P2

X	Y	$P(X,Y)$
x^0	y^0	0.4
x^0	y^1	0.3
x^1	y^0	0.2
x^1	y^1	0.1

In $P_1 : P(X,Y) = P(X)P(Y)$, i.e. $0.6 \times 0.8 = 0.48$. Hence, $I(P_1) = X \perp Y$ and is shown by graph $\mathcal{G}_\emptyset$

In $P_2 : P(X,Y) \neq P(X)P(Y)$. Hence, $I(P_1) = \emptyset$ and is shown by both graphs $\mathcal{G}_{\mathcal{X}\to\mathcal{Y}}$ and $\mathcal{G}_{\mathcal{Y}\to\mathcal{X}}$

4.2 What is $\mathcal{I}(\mathcal{G})$

4.2.1 Local Markovian Assumptions of Bayesian Network

A Bayesian Network structure $\mathcal{G}$ is a directed acyclic graph (DAG) whose nodes represent random variables $X_1, X_2, ..., X_N$.

Definition:

Let Pa_{X_i} denote the parents of X_i in $\mathcal{G}$, and $NonDescendants_{X_i}$ denote the variable in the graph that are non-descendants of X_i. Then $\mathcal{G}$ encodes the following set of local conditional independence assumptions $I_\ell(\mathcal{G})$:

$$I_\ell(\mathcal{G}) : \{X_i \perp NonDescendants_{X_i} | Pa_{X_i} : \forall i\},$$

Each node X_i is independent of its non-descendants given its parents.

Graph Separation Criterion

Directed edges separation (D-separation) criterion for Bayesian networks:

Definition:

Variables x and y are *D-separated* (conditionally independent) given z if they are separated in the *moralized* ancestral graph.

Example

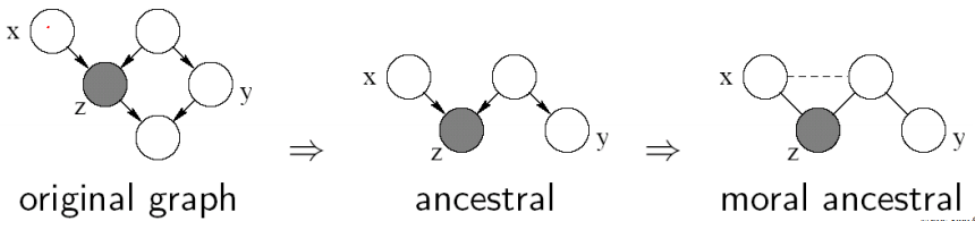


Figure 11: D-separation criterion example

Construct the ancestral graph by removing all nodes except the random variables of interest and their ancestors. Then perform moralization on ancestral graph by removing all directions on edges and connecting nodes that are originally unconnected and have a common child node.

The example shows conditional independence. If there is a way to travel from one node to another node using any path (not through the given), then those two nodes are not conditionally independent.

If $\mathbf{X} \perp \mathbf{Y}|\mathbf{Z}$, then we say $\mathbf{Z}$ D-separates $\mathbf{X}$ and $\mathbf{Y}$.

4.2.2 Global Markovian Assumptions of Bayesian Network

Practical definition of $\mathcal{I}(\mathcal{G})$

$\mathbf{X}$ is D-separated from $\mathbf{Z}$ given $\mathbf{Y}$ if we can't send a ball from any node in $\mathbf{X}$ to any node in $\mathbf{Z}$ using the *"Bayes-ball"* algorithm illustrated by the following examples (plus some boundary conditions):

Example

Causal Trail: $\mathbf{X} \rightarrow \mathbf{Z} \rightarrow \mathbf{Y}$, active $\iff$ $\mathbf{Z}$ is not observed.

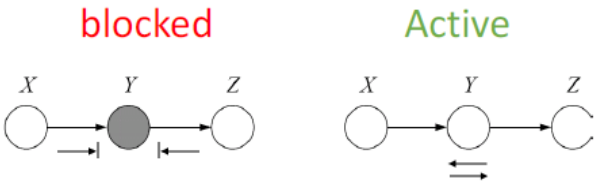


Figure 12: Causal Trail

Common cause: $\mathbf{X} \leftarrow \mathbf{Z} \rightarrow \mathbf{Y}$, active $\iff$ $\mathbf{Z}$ is not observed.

Common effect: $\mathbf{X} \rightarrow \mathbf{Z} \leftarrow \mathbf{Y}$, active $\iff$ either $\mathbf{Z}$ or one of $\mathbf{Z}$'s descendents is observed.

Definition:

All independence properties that correspond to d-separation:

$$\mathcal{I}(\mathcal{G}) = \{X \perp Z|Y : dsep_{\mathcal{G}}(X; Z|Y)\}$$

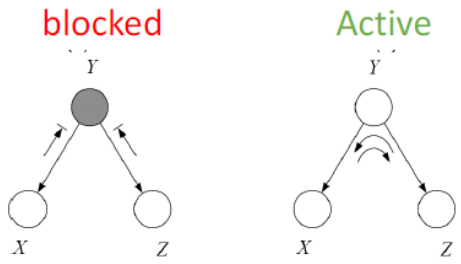


Figure 13: Common Cause

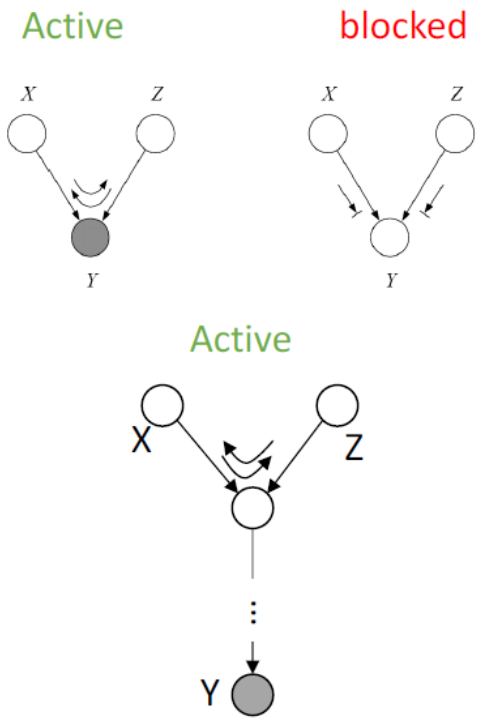


Figure 14: Common Effect

Example

To find I(G) for the graph below, we try all possible trails and see what conditional independence we can draw from the trail.

For example, for trail $X_4 \leftarrow X_1 \rightarrow X_3$, the trail is not active only if X_1 is observed based on the common cause structure. So we get $X_4 \perp X_3|X_1$, $X_4 \perp X_3|\{X_1, X_2\}$.

Similarly, we can find $\{X_1 \perp X_2, X_1 \perp X_2|X_4\}$ from trail $X_1 \rightarrow X_3 \leftarrow X_2$ and $\{X_2 \perp X_4, X_2 \perp \{X_1, X_4\}$, $X_2 \perp X_4|\{X_1, X_2\}$, $X_2 \perp X_4|\{X_1, X_3\}\}$ from trail $X_4 \leftarrow X_1 \rightarrow X_3 \leftarrow X_2$

Thus,

$$I(G) = \{X_4 \perp X_3|X_1, X_4 \perp X_3|\{X_1, X_2\}, X_1 \perp X_2, X_1 \perp X_2|X_4,$$
$$X_2 \perp X_4, X_2 \perp \{X_1, X_4\}, X_2 \perp X_4|X_1, X_2 \perp X_4|\{X_1, X_3\}\}$$

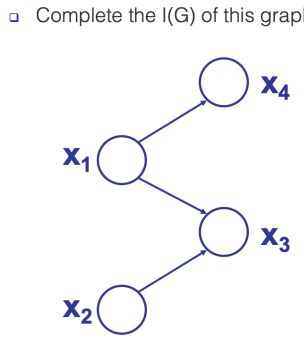


Figure 15: I(G) example

4.3 Equivalence Theorem

Separation properties in the graph imply independence properties about the associated variables.

Formally, for any graph G

let D_1 denote the family of all distributions that satisfy I(G), D_2 denote the family of all distributions that factor according to G.

Then we have $D_1 \equiv D_2$. The two families are the same.

In other words, when building the distribution, we can directly use the factorization law to assemble a distribution mechanically by $P(X) = \prod_{i=1:d} P(X_i|X_{\pi_i})$.

4.4 Conditional probability tables (CPTs)

To build the joint distribution for the graph with discrete random variables below, we can use conditional probability tables.

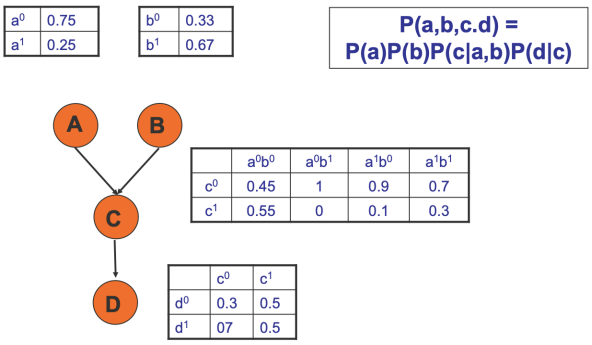


Figure 16: CPT example

4.5 Conditional probability density (CPDs)

To build the joint distribution for the graph with continuous random variables below, we can use conditional probability density functions. Here is an example of defining a continuous random variable dependent on other continuous random variables.

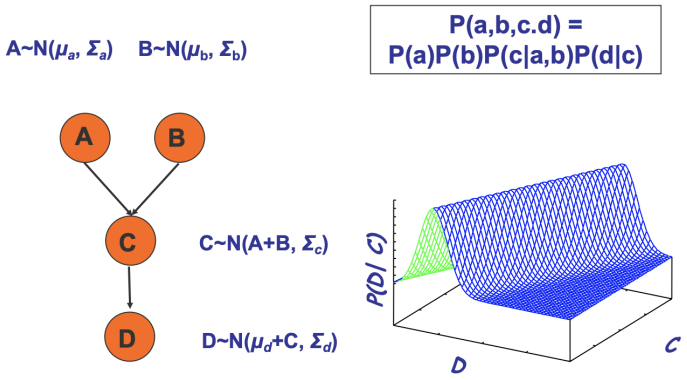


Figure 17: CPD example

4.6 Summary of BN semantics

- Conditional independencies imply factorization
- Factorization according to G implies the associated conditional independencies.

5 Soundness and completeness

- **Soundness**
  Theorem: If a distribution P factorizes according to G, then $I(G) \subseteq I(P)$ (guaranteed)

- **Completeness**
  Claim: For any distribution P that factorizes over G, if $(X \perp Y|Z) \in I(P)$ then $d - sep_G(X;Y|Z)$ (not guaranteed)

- Contrapositive of the completeness statement
  - If and Y are not d-separated given Z in G, are X and Y guaranteed to be dependent in all distributions P that factorize over G?
  - No. Even if a distribution factorizes over G, it can still contain additional independencies that are not reflected in the structure.
  - Example: Consider graph A → B, which indicates that A and B are dependent. However, the conditional distribution of B—A could be arbitrarily picked so that A and B are actually independent. (The independence can be captured by some subtle way of pasteurization)
  - **Theorem**: Let G be a BN graph. If X and Y are not d-separated given Z in G, then X and Y are dependent in **some** distribution P that factorizes over G.

A	b^0	b^1
a^0	0.4	0.6
a^1	0.4	0.6

- **Theorem**: For **almost all** distributions P that factorize over G, i.e., for all distributions except for a set of 'measure zero' in the space of CPD parameterizations, we have that $I(P) = I(G)$

10-708: Probabilistic Graphical Models, Spring 2020

Lecture 6: Case Studies: HMM and CRF

Lecturer: Eric P. Xing *Scribe: Tejas Srinivasan, Ruochi Zhang, Hongyu Zheng*

1 Hidden Markov Models (HMMs)

1.1 Introduction

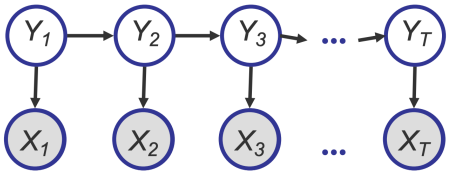


Figure 1: Hidden Markove Models (HMMs)

Hidden Markov Models (HMMs) are one of the common ways of modeling time series data. It is used in traditional speech recognition systems, natural language processing, computational biology, etc. HMMs allow us to deal with both observed and hidden events. Most of the common HMMs follow the first order markov assumption, which states that when predicting the future the past doesn't matter, only the present.

Hidden Markov Models consist of hidden and observed states ($y_{1...T}$ and $x_{1...T}$, respectively), and the dependencies are captured in the network structure seen in Figure 1. The observation and hidden space may be discrete or continuous; for the sake of simplicity, we assume that both are discrete in this case.

We define the following:

- Observation sequence: $\mathbf{x} = x_1, x_2, ..., x_T$
- Hidden sequence (unobserved): $\mathbf{y} = y_1, y_2, ..., y_T$
- Set of observations: $C = \{c_1, c_2, ..., c_K\}$
- Set of hidden states: $I = \{1, 2, ..., M\}$
- Prior probabilities: $p(y_1) \sim \texttt{Multinomial}(\pi_1, \pi_2, ..., \pi_M)$
- Transition probabilities (between hidden states): $p(y_t^j = 1 | y_{t-1}^i = 1) = a_{i,j} \implies p(y_t | y_{t-1}) = \mathbf{A}$
- Emission probabilities: $p(x_t | y_t^i = 1) \sim \texttt{Multinomial}(b_{i,1}, b_{i,2}, ..., b_{i,K})$

1.2 Joint Probability Estimation: Forward-Backward Algorithm

Given a sequence of observed states $\mathbf{x} = x_1, x_2, ..., x_T$ and hidden states $\mathbf{y} = y_1, y_2, ..., y_T$, we want to estimate the joint probability $p(\mathbf{x}, \mathbf{y})$

$$\begin{aligned} p(\mathbf{x}, \mathbf{y}) &= p(y_1)p(x_1|y_1)p(y_2|y_1)p(x_2|y_2)...p(y_T|y_{T-1})p(x_T|y_T) \\ &= \pi_{y_1} \prod_{t=2}^{T} p(y_{t-1}|y_t) \prod_{t=1}^{T} p(x_t|y_t) \\ &= p(y_1, ..., y_T)p(x_1, ..., x_T|y_1, ..., y_T) \\ &= p(\mathbf{y})p(\mathbf{x}|\mathbf{y}) \end{aligned}$$

1.2.1 Forward Algorithm

We want to estimate $p(\mathbf{x})$, the likelihood of the observed sequence, given the HMM

$p(\mathbf{x}) = \sum_y p(\mathbf{x}, \mathbf{y})$

We define $\alpha(y_t^k = 1) = \alpha_t^k = P(x_1, ..., x_t, y_t^k = 1)$

$$\begin{aligned} \alpha_t^k &= \sum_i p(x_t|y_t^k = 1)p(y_t^k = 1|y_{t-1}^i = 1)p(x_1, ..., x_{t-1}, y_{t-1}^i = 1) \\ &= \sum_i p(x_t|y_t^k = 1)a_{i,k}\alpha_{t-1}^i \\ &= p(x_t|y_t^k = 1) \sum_i a_{i,k}\alpha_{t-1}^i \\ p(\mathbf{x}) &= \sum_k p(x_1, ..., x_T, y_T^k = 1) \\ &= \sum_k \alpha_T^k \end{aligned}$$

The forward probabilities $\alpha_t^k = P(x_1, ..., x_t, y_t^k = 1)$ can be computed recursively.

1.2.2 Backward Algorithm

We want to compute $P(y_t^k = 1|\mathbf{x})$, the posterior disributuon of the t^{th} hidden state, given $\mathbf{x}$. We start by estimating $P(y_t^k = 1, \mathbf{x})$

$$\begin{aligned} P(y_t^k = 1, \mathbf{x}) &= P(x_1, ..., x_t, y_t^k = 1, x_{t+1}, ..., x_T) \\ &= P(x_1, ..., x_t, y_t^k = 1)P(x_{t+1}, ..., x_T|x_1, ..., x_t, y_t^k = 1) \\ &= P(x_1, ..., x_t, y_t^k = 1)P(x_{t+1}, ..., x_T|y_t^k = 1) \end{aligned}$$

$P(x_1, ..., x_t, y_t^k = 1)$ is the forward probability. We define the backward probability, $\beta_t^k = P(x_{t+1}, ..., x_T|y_t^k = 1)$.

$$\begin{aligned}\beta_t^k &= P(x_{t+1}, ..., x_T | y_t^k = 1) \\ &= \sum_i P(x_{t+2}, ..., x_T | y_{t+1}^i = 1) P(x_{t+1} | y_{t+1}^i = 1) P(y_{t+1}^i = 1 | y_t^k = 1) \\ &= \sum_i \beta_{t+1}^i P(x_{t+1} | y_{t+1}^i = 1) a_{k,i}\end{aligned}$$

We compute β_t^k recursively. $P(y_t^k = 1, \mathbf{x}) \propto \alpha_t^k \beta_t^k$

1.3 Posterior Decoding

We can now compute the posterior probabilities of the hidden state, $P(y_t^k = 1|\mathbf{x})$.

$$\begin{aligned}P(y_t^k = 1|\mathbf{x}) &= \frac{P(y_t^k = 1, \mathbf{x})}{P(\mathbf{x})} \\ &= \frac{\alpha_t^k \beta_t^k}{P(\mathbf{x})} \\ &= \frac{\alpha_t^k \beta_t^k}{\sum_k \alpha_T^k}\end{aligned}$$

1.4 Viterbi Decoding

What if we want to find the joint probability of the whole sequence (and the most likely sequence of states)?

$$\vec{y^*} = \text{argmax}_{\vec{y}} P(\vec{y}|\vec{x})$$

This differs from what posterior decoding trying to solve which is the most likely state at position t given the sequence of $\vec{x}$. And it's hard for posterior decoding to solve this problem as it's impossible to store all the probabilities which amount grows exponentially to the size of sequences.

To solve this, the Viterbi algorithm is proposed based on dynamic programming. The recursion for V_t^k which stands for the probability of the most likely sequence of states ending at state $y_t = k$ can be written as:

$$V_t^k = p(x_t | y_t^K = 1) \max_i a_{i,k} V_{t-1}^i$$

One technical issue might encounter is underflows. The probability scores become extremely small after certain times steps. The solution for that would be to take the logarithmic transformation of all values.

1.5 Computational Complexity and Implementation Details

The sum-product algorithm is polynomial. The time complexity is $O(K^2 N)$, the space complexity is $O(KN)$, where K is the number of states and N number of time steps.

For the Viterbi algorithm, as mentioned above, we can solve the underflow problem via the logarithmic transformation and use the sum of logs instead of multiplication of probabilities.

For the forward/backward algorithm, we can solve the underflow problem by re-scaling the α, β term with a constant.

1.6 Learning HMM

We first make an assumption called "parameter sharing" to further simplify the HMM learning problem. It means we only consider a time-invariant 1st-order Markov model, where we have:

$$\pi_k = p(X_1^K = 1)$$
$$A_{ij} = p(X_t^j = 1 | X_{t-1}^i) = 1$$

1.6.1 Supervised Learning

Both sequences of observation and sequences of true hidden states are known. The maximum likelihood parameters are:

$$a_{ij} = \frac{\sum_n \sum_{t=2}^T y_{n,t-1}^i y_{n,t}^j}{\sum_n \sum_{t=2}^T y_{n,t-1}^i}$$
$$b_{ik} = \frac{\sum_n \sum_{t=2}^T y_{n,t-1}^i x_{n,t}^k}{\sum_n \sum_{t=2}^T y_{n,t-1}^i}$$

where a_{ij} is the corresponding entry in the transition matrix from state i to state j and b_{ik} is the corresponding entry in the emission matrix for state i to produce observation k.

As shown above, the MLE estimation is actually a "counting scheme" which leads to its drawback. When there's little data, this may be over-fitting. For unobserved transitions or unobserved emissions the probability would be assigned as 0, which is really bad. (Sparse data problem).

This can be resolved by some "smoothing techniques". One of them is called "Pseudocounts". It would mean for each count above, we would add some pseudo counts to make sure it's non-zero. The pseudo count for each entry represents our prior belief, while the total pseudo counts represent the "strength" of prior belief. This is actually equivalent to Bayesian estimation under a uniform prior with "parameter strength" equals to the pseudo counts. Similar techniques have been used in the Bayesian language model where a Dirichlet prior is assigned to each row of the transition matrix.

1.6.2 Unsupervised Learning

When the true state path is unknown and only the observation is known. It can be solved by the Baum Welch algorithm (i.e., EM algorithm). It is guaranteed to increase the log-likelihood of the model after each iteration and it will converge to local optimum, depending on initial conditions.

Because in the unsupervised learning setting we cannot optimize over the log-likelihood directly, we optimize over the expected log-likelihood function. As compared to the log-likelihood, we replace the variables related to the unobserved hidden states with sufficient statistics of them. In brief, the E-step of the EM algorithm

calculates the sufficient statistics based on the forward and backward algorithm.

$$\gamma_{n,t}^i = p(y_{n,t}^i = 1|x_n)$$
$$\xi_{n,t}^{i,j} = p(y_{n,t-1}^i = 1, y_{n,t}^j = 1|x_n)$$

The M-step then maximize the expected log likelihood which is symbolically identical to MLE.

$$\pi_i = \frac{\sum_n \gamma_{n,1}^i}{N}$$
$$a_{ij} = \frac{\sum_n \sum_{t=2}^T \xi_{n,t}^{i,j}}{\sum_n \sum_{t=1}^{T-1} \gamma_{n,t}^i}$$
$$b_{ik} = \frac{\sum_n \sum_{t=1}^T \gamma_{n,t}^i x_{n,t}^k}{\sum_n \sum_{t=1}^{T-1} \gamma_{n,t}^i}$$

2 Conditional Random Fields (CRFs)

2.1 Caveats of HMM

There are two shortcomings of HMM in general:

- HMMs are unable to capture global knowledge. Transitions and emissions in HMMs are all local ("I saw a word last time, and I want to predict the next word with just that"). Higher-order Markov models can alleviate this problem but the cost scales up quickly. This is also prevalent in NLP researches, as emotional and sentimental analysis are likely very non-local.

- A philosophical concern is that HMM optimizes $P(X, Y)$ the joint probability, but what we are more interested in is the conditional probability $P(Y \mid X)$.

A real and challenging example is as follows.

Locally, for every state at every step, State 2 is always the preferred next step and State 1 is never more probable. However in HMM, each step is independent and we can observe a very counter-intuitive phenomenon: The most probable path under HMM is $1-1-1-1$, the path that is never favored locally at every step, with probability 0.09 (in contrast, the probability for path $2-2-2-2$ is only 0.018). This is analogous to the "rib versus rob" example in the famous Lafferty and McCallum paper [1], which was written at Carnegie Mellon.

Here is an intuitive explanation. Imagine State 1 is the introvert who only has two friends, and even $1 \to 1$ is not the most popular one, it is still a very probable choice with probability 0.4. State 2 is the popular guy who likes $2 \to 2$ most, but the probability is still only 0.3. In HMM we compare these numbers, but this is like comparing "friendship" or "stickness" from different people. Intuitively, it should not even be compared from the start.

This Label Bias problem (preference for states with a lower number of transitions) is intrinsic to HMM because we need to normalize the emission probability $P(Y \mid X)$ for every X, locally. Our solution is to normalize the probabilities globally instead, called the potentials, and use a global partition function. This step also turns the model into an undirected graphical model.

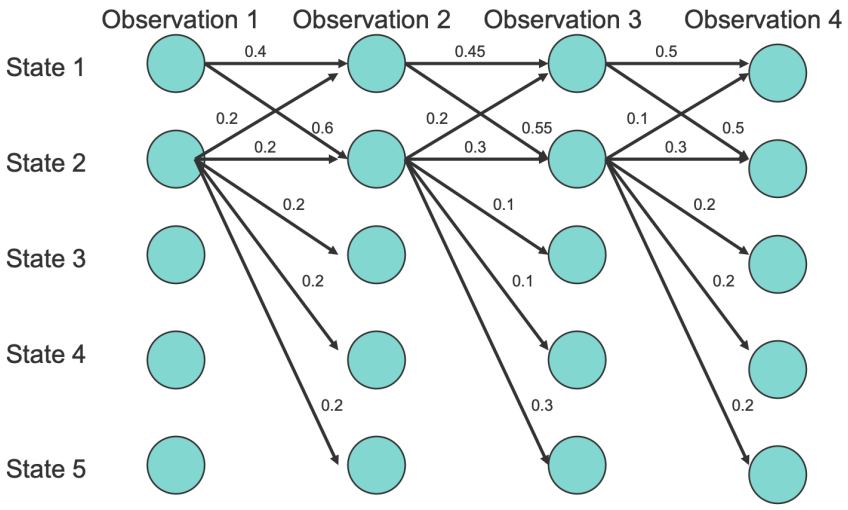


Figure 2: The Label Bias example. Edge denotes probability of transition.

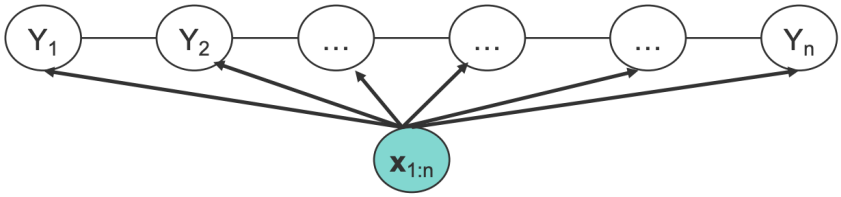


Figure 3: The Conditional Random Field (CRF) model.

2.2 The CRF Model

We directly define the conditional distribution with the weighted potential functions, without caring about local normalization, as follows.

$$\begin{aligned} P(\mathbf{y}_{1:n} \mid \mathbf{x}_{1:n}) &= \frac{1}{Z(\mathbf{x}_{1:n})} \prod_{i=1}^{n} \phi(y_i, y_{i-1}, \mathbf{x}_{1:n}) \\ &= \frac{1}{Z(\mathbf{x}_{1:n}, \mathbf{w})} \prod_{i=1}^{n} \exp(\mathbf{w}^T \mathbf{f}(y_i, y_{i-1}, \mathbf{x}_{1:n})) \end{aligned}$$

The potential functions can also be directly defined on cliques of the model, which are now single states and state pairs including the observations (now as features). It is very flexible, utilizing both local and global information.

$$P(\mathbf{y}_{1:n} \mid \mathbf{x}_{1:n}) = \frac{1}{Z(\mathbf{x}_{1:n}, \lambda, \mu)} \exp(\sum_{i=1}^{n} (\lambda^T \mathbf{f}(y_i, y_{i-1}, \mathbf{x}_{1:n}) + \mu^T \mathbf{g}(y_i, \mathbf{x}_{1:n}))$$

This formulation is usually called a feature-based model, with $\mathbf{f}$ and $\mathbf{g}$ as edge features and node features.

2.3 CRF Inference

With specified model, we are interested in inference: Given λ and μ, find the most probable states given observation $\mathbf{x}_{1:n}$:

$$\mathbf{y}^* = \arg\max_{\mathbf{y}} \exp(\sum_{i=1}^{n} (\lambda^T \mathbf{f}(y_i, y_{i-1}, \mathbf{x}_{1:n}) + \mu^T \mathbf{g}(y_i, \mathbf{x}_{1:n}))$$

This can be done by running the message passing algorithm on the junction tree of CRF (which is a chain).

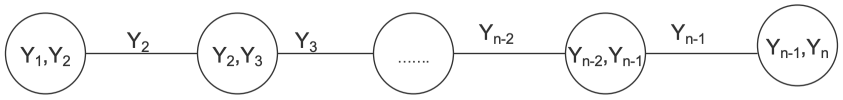


Figure 4: Junction Tree of the CRF.

Its formulation is the same as the Viterbi decoding algorithms used in HMMs.

2.4 CRF Learning

For simplicity assume the model is fully observed and we want to find the most probable λ and μ. Formally, given N observations $\{\mathbf{x}_d, \mathbf{y}_d\}$, we want to find the following:

$$\begin{aligned}
\lambda^*, \mu^* &= \arg\max_{\lambda,\mu} \prod_{i=1}^{N} P(\mathbf{y}_d \mid \mathbf{x}_d, \lambda, \mu) \\
&= \arg\max_{\lambda,\mu} \sum_{d=1}^{N} (\sum_{i=1}^{n} (\lambda^T \mathbf{f}(y_{d,i}, y_{d,i-1}, \mathbf{x}_d) + \mu^T \mathbf{g}(y_{d,i}, \mathbf{x}_d) - \log Z(\mathbf{x}_d, \lambda, \mu))
\end{aligned}$$

It appears we can do this in a one-shot manner (counting frequencies), but unfortunately it does not work for CRF. This is one of the disadvantages of CRF compared to HMM.

Next, we compute the gradient with respect to λ:

$$\nabla_\lambda L(\lambda, \mu) = \sum_{d=1}^{N} (\sum_{i=1}^{n} (\mathbf{f}(y_{d,i}, y_{d,i-1}, \mathbf{x}_d) - \sum_{\mathbf{y}} P(\mathbf{y} \mid \mathbf{x}_d) \sum_{i=1}^{n} \mathbf{f}(y_i, y_{i-1}, \mathbf{x}_d))$$

The first term here is straightforward from the observations. However to calculate the second term, we assume $\mathbf{y}$ are not observed and take expectation of the same term over all possible $\mathbf{y}$. Their difference defines the gradient. This means even if we observe all $\mathbf{y}$s already, we will pretend we do not know it in one of the steps. In general, learning undirected graphical models requires inference.

It turns out the expectation term is, in fact, tractable as we can decompose it into sums over marginal distributions of neighboring nodes:

$$\begin{aligned}
\sum_{\mathbf{y}} P(\mathbf{y} \mid \mathbf{x}_d) \sum_{i=1}^{n} \mathbf{f}(y_i, y_{i-1}, \mathbf{x}_d)) &= \sum_{i=1}^{n} \sum_{\mathbf{y}} P(\mathbf{y} \mid \mathbf{x}_d) \mathbf{f}(y_i, y_{i-1}, \mathbf{x}_d)) \\
&= \sum_{i=1}^{n} \sum_{y_i, y_{i-1}} P(y_i, y_{i-1} \mid \mathbf{x}_d) \mathbf{f}(y_i, y_{i-1}, \mathbf{x}_d))
\end{aligned}$$

In the end, this is an EM algorithm where we calculate the expectations, then update the parameters with gradient ascent.

2.5 Concluding Remarks

After all this work, do we really get a powerful model compared to HMM? In fact, the earlier papers are somewhat disappointing in experimental results. Comparison of error rates on synthetic data (where the label bias is baked into the datasets) shows that CRF is only marginally better than HMM with the presence of label bias, and it is actually slightly worse in some tests without significant label bias. If you show similar results in your ICML/(other ML conference) submissions now, it will be outright rejected. But 15 to 20 years ago people were paying more attention to the mathematical beauty, clarity, and potential of the model rather than benchmarks. Nowadays, HMMs and CRFs are still the backbones of many machine learning algorithms, but when it started the paper is a disaster: hard to read and mediocre numbers. In the end, what is the catch? People do need to publish papers, but at least for this class, to evaluate the course projects, good formulations and clear designs will be more important than "good numbers", so don't worry if you don't get a good result.

References

[1] John Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001.

10-708: Probabilistic Graphical Models, Spring 2020

5: Parameter Estimation

Lecturer: Eric P. Xing *Scribe: Matthew Ho, Danlei Zhu, Yijie Sun, John Fang, Zhichu Lu*

In general, learning graphical models involves trying to infer the best Bayesian Network from a dataset of independent samples. Graphical model learning can be broadly separated into two categories: structural learning, wherein one might try to estimate graphical connections and the implied independences, and parameter learning, where one might seek to estimate specific conditional probabilities. This lecture covers the latter.

1 Parameter Estimation for Completely Observed GMs of Given Structure

In this section, we consider learning parameters for a Bayesian Network which has a known, fixed structure $\mathcal{G}$ and is completely observable (i.e. our data samples include observations of all variables). Stated formally, we are given a dataset of N independent, identically-distributed training cases $D = \{x_1, \ldots, x_N)$. Each training case $x_n = (x_{n,1}, \ldots, x_{n,M})$ is a vector of M values, one per node.

To address our problem of learning parameters in this context, we will describe how simple, completely-observed, structure-fixed graphical models can be generalized into the *exponential family* of distributions. This generalized reparameterization will allow us to write closed-form expressions for quantities that we are interested (e.g. conditional probabilities, means, etc.). The simple graphical models that we describe in detail are building blocks for more complex models, making the exponential family parameterization useful for learning all graphical models.

1.1 Exponential Family

1.1.1 Formulation and Examples

The *exponential family* is a parametric set of probability distributions which characterize many common examples in modern statistics, including the Bernoulli, Multinomial, Gaussian, Poisson, and Gamma distributions. For a numeric random variable X described by an exponential family distribution, the PDF can be written as:

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

with natural (canonical parameter) η. The function $T(x)$ is called the *sufficient statistic* because its output is all that is required from the data to estimate η. The function $A(\eta) = \log Z(\eta)$ is the *log normalizer* and ensures the probability distribution can be integrated to unity.

We first demonstrate how a multivariate Gaussian distribution can be represented in terms of the exponential

family canonical parameters and functions. First, the classic k-dimensional Gaussian distribution:

$$\begin{aligned} p(x|\mu,\Sigma) &= \frac{1}{(2\pi)^{k/2}|\Sigma|^{1/2}} \exp\left\{\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right\} \\ &= \frac{1}{(2\pi)^{k/2}} \exp\left\{-\frac{1}{2}\operatorname{Tr}\left(\Sigma^{-1}xx^T\right) + \mu^T\Sigma^{-1}x - \frac{1}{2}\mu^T\Sigma^{-1}\mu - \log|\Sigma|\right\} \end{aligned}$$

which is fully described by the $k+k^2$ parameters that define the first two moments of the distribution, μ and Σ. $\operatorname{Tr}(\cdot)$ is the matrix trace operation. We can represent this same distribution in the exponential family representation,

$$\begin{aligned} \eta &= \left[\Sigma^{-1}\mu; -\frac{1}{2}\operatorname{vec}\left(\Sigma^{-1}\right)\right] = \left[\eta_1, \operatorname{vec}\left(\eta_2\right)\right], \qquad \eta_1 = \Sigma^{-1}\mu \text{ and } \eta_2^2 = -\frac{1}{2}\Sigma^{-1} \\ T(x) &= \left[x; \operatorname{vec}\left(xx^T\right)\right] \\ A(\eta) &= \frac{1}{2}\mu^T\Sigma^{-1}\mu + \log|\Sigma| = -\frac{1}{2}\operatorname{Tr}\left(\eta_2\eta_1\eta_1^T\right) - \frac{1}{2}\log\left(-2\eta_2\right) \\ h(x) &= (2\pi)^{-k/2} \end{aligned}$$

where $\operatorname{vec}(\cdot)$ is an operation which flattens a matrix into a 1-D vector. The $k+k^2$ parameters in the canonical η vector fully capture the variability in the k-dimensional Gaussian previously parameterized by μ and σ.

As another example, we will show how the K-outcome multinomial distribution can be written in exponential family form. We start by stating the familiar probability distribution in a form more representative of the exponential family,

$$\begin{aligned} p(x|\pi) = \pi_1^{x_1}\pi_2^{x_1}\cdots\pi_K^{x_K} &= \exp\left\{\sum_k x_k \ln \pi_k\right\} \\ &= \exp\left\{\sum_{k=1}^{K-1} x_k \ln \pi_k + \left(1 - \sum_{k=1}^{K-1} x_K\right)\ln\left(1 - \sum_{k=1}^{K-1}\pi_k\right)\right\} \\ &= \exp\left\{\sum_{k=1}^{K-1} x_k \ln\left(\frac{\pi_k}{1-\sum_{k=1}^{K-1}\pi_k}\right) + \ln\left(1 - \sum_{k=1}^{K-1}\pi_k\right)\right\}. \end{aligned}$$

Note, there are only $K-1$ parameters to fit, as $\sum_{k=1}^{K}\pi_k = 1$. We follow by stating the explicit exponential family representation,

$$\begin{aligned} \eta &= \left[\ln\left(\frac{\pi_k}{\pi_K}\right); 0\right] \\ T(x) &= [x] \\ A(\eta) &= -\ln\left(1 - \sum_{k=1}^{K-1}\pi_k\right) = \ln\left(\sum_{k=1}^{K} e^{\eta_k}\right) \\ h(x) &= 1 \end{aligned}$$

1.1.2 Moments

One particularly useful property of exponential family distributions is that we can easily compute their q-th central moments through the q-th derivatives of the log normalizer $A(\eta)$:

$$\begin{aligned}\frac{dA(\eta)}{d\eta} &= \mathbb{E}\left[T(x)\right] \equiv \mu \\ \frac{d^2A(\eta)}{d\eta^2} &= \mathrm{Var}\left[T(x)\right] \\ &\vdots\end{aligned}$$

where the expectation value of $T(x)$ is defined as the moment parameter μ. Since the log normalizer's first derivative is μ and its second derivative must be positive, then there exists some function ψ which defines a 1-to-1 relationship between canonical and moment parameters,

$$\eta \equiv \psi(\mu)$$

This property of the exponential family is particularly significant in inferring η. When performing MLE on an exponential family distribution, we can maximize the log-likelihood, ℓ, with respect to η, estimate μ directly, and then infer η using the ψ function.

$$\begin{aligned}\ell(\eta; D) &= \sum_n \log h(x_n) + \left(\eta^T \sum_n T(x_n)\right) - NA(\eta) \\ &\downarrow \\ 0 = \frac{\partial \ell}{\partial \eta} &= \sum_n T(x_n) - N\frac{\partial A(\eta)}{\partial \eta} \\ &\downarrow \\ \frac{\partial A(\eta)}{\partial \eta} = \hat{\mu}_{\mathrm{MLE}} &= \frac{1}{N}\sum_n T(x_n)\end{aligned}$$

Subsequently, we can find an estimate for the canonical parameter via $\hat{\eta}_{\mathrm{MLE}} = \psi(\hat{\mu}_{\mathrm{MLE}})$. This procedure is called *moment matching*.

1.1.3 Sufficiency

In previous sections, we have seen that most of the distributions we encounter can be expressed in the form of exponential family with appropriate $T(x)$ and η.

However, why is this interesting and practically useful? It turns out that for $p(x|\theta), T(x)$ is sufficient for θ if there is no information in X regarding θ beyond that in $T(x)$. For instance, if your boss wants you to do inference w.r.t. θ, you do not need to save all data X but $T(x)$, the sufficient statistics.

To define this property more rigorously, we need the following.

In the Bayesian view, the posterior distribution of the parameter θ is dependent of the data X. However, the posterior of the parameter is independent of data X given sufficient statistics $T(x)$, i.e. $p(\theta|T(x), x) = p(\theta|T(x))(A)$. In the Frequentist view, our data is generated from some true parameter. Yet, the distribution of our data x is dependent of the parameter θ if given the sufficient statistics $T(x)$, i.e. $p(x|T(x), \theta) = p(x|T(x))(B)$.

If we combine these two views, we obtain the Neyman factorization theorem: The statistics $T(x)$ is sufficient for the parameter θ if both (A) and B hold.

1.1.4 Examples

Here are some common distributions written in the general expoential family form.

For Gaussian, $\eta = [\Sigma^{-1}\mu; -\frac{1}{2}\vec{\Sigma^{-1}}]$, $T(x) = [x, x\vec{x^T}]$, $A(\eta) = \frac{1}{2}\mu^T\Sigma^{-1}\mu + \frac{1}{2}\log|\Sigma|$, $h(x) = (2\pi)^{-k/2}$,

$$\mu_{MLE} = \frac{1}{N}\Sigma_n T_1(x_n) = \frac{1}{N}\Sigma_n x_n$$

.

For Multinomial, $\eta = [\ln\frac{\pi_k}{\pi_K}; 0]$, $T(x) = [x]$, $A(\eta) = -\ln(1 - \Sigma_{k=1}^{K-1}\pi_k) = \ln(\Sigma_{k=1}^{K}e^{\eta_k})$, $h(x) = 1$,

$$\mu_{MLE} = \frac{1}{N}\Sigma_n x_n$$

For Possion, $\eta = \log\lambda$, $T(x) = x$, $A(\eta) = \lambda = e^{\eta}$, $h(x) = \frac{1}{x!}$,

$$\mu_{MLE} = \frac{1}{N}\Sigma_n x_n$$

.

1.2 Generalized Linear Models (GLIMs)

With the definition of Exponential Family Distribution, we can begin analyzing Generalized Linear Models(GLIMs). Suggested by its name, the definition of GLIM is very general.

1. The observed input x is assumed to enter into the model via a linear combination of its elements ξ, where $\xi = \theta^T x$.
2. The conditional mean μ is represented as a function $f(\xi)$ of ξ, where f is known as the response function.
3. The observed output y is assumed to be characterized by an exponential family distribution p with conditional mean μ.

Then the model $E_p(y) = \mu = f(\theta^T x)$ is called a GLIM. Some basic examples of GLIM include:

1. Linear Regression

   Assume the target variable y and the inputs are related by the equation: $y_i = \theta^T X_i + \epsilon_i$,where $\epsilon \sim N(0, \sigma)$. Then we have $p(y_i|x_i, \theta) = \frac{1}{\sqrt{2\pi}\sigma}\exp -\frac{(y_i - \theta^T x_i)^2}{2\sigma^2}$. To estimate θ, we can apply LMS algorithm (a gradient ascent/descent).

2. Logistic Regression(sigmoid classifier, perceptron,etc.)

   In logistic regression, the condition distribution is $p(y|x) = \mu(x)^y(1 - \mu(x))^{1-y}$, where $\mu(x) = \frac{1}{1+e^{-\theta^T x}}$ and $y \in \{0, 1\}$. To estimate parameter θ, we can either directly apply brute-force gradient method or generic laws by observing that $p(y|x)$ is a GLIM.

More advanced examples of GLIM include:

1. Markov Random Fields, where $p(X) = \frac{1}{Z}\exp(\Sigma_{i,j\in N_i}\theta_{ij}X_iX_j + \Sigma_i\theta_{i0}X_i)$

2. Restricted Boltzmann Machines, where

$$p(x, h|\theta) = \exp\{\Sigma_i \theta_i \phi_i(x_i) + \Sigma_j \theta_j \phi_j(h_j) + \Sigma_{i,j} \theta_{i,j} \phi_{i,j}(x_i, h_j) - A(\theta)\}$$

3. Conditional Random Fields, where

$$p_\theta(y|x) = \frac{1}{Z}(\theta, x) \exp\{\Sigma_c \theta_c f_c(x_c y_c)\}$$

and all X_i are assumed as features that are inter-dependent.

A more formal view of GLIMs is the following:

parameter θ and data $x \to \xi \overset{f}{\to} \mu \overset{\Psi}{\to} \eta \overset{\text{EXP}}{\to} y$, where f is some response function, Ψ is some reversible transformer corresponding to the T operator before, and EXP is some exponential family distribution we use.

Notice that the choice of exp family distribution is constrained by the nature of the data y. For example, if y is a continuous vector, then multivariate Gaussian is a reasonable choice. However, if y is a class label, using Bernouli or multinomial is more favorable.

We also have some mild constraints for the choice of response function, such as positivity. There also exists some canonical response functions for different models.

- Gaussian, $\mu = \eta$
- Bernoulli, $\mu = \frac{1}{1+e^{-\eta}}$
- multinomial, $\mu_i = \frac{\eta_i}{\Sigma_j e^{\eta_j}}$
- Poisson, $\mu = e^\eta$
- gamma, $\mu = -\eta^{-1}$.

1.3 Learning GLIMs

1.3.1 MLE for GLIMs with natural response

For example for log-likelihood,

$$l = \sum_n \log h(y_n) + \sum_n (\theta^T x_n y_n - A(\eta_n))$$

note that here $\theta^T x_n$ acts as the natural parameters for the exponential family distribution. Take derivative of the log-likelihood,

$$\begin{aligned}
\frac{dl}{d\theta} &= \sum_n (x_n y_n - \frac{dA(\eta_n)}{d\eta_n} \frac{d\eta_n}{d\theta}) \\
&= \sum_n (y_n - \mu_n) x_n \\
&= X^T (y - \mu)
\end{aligned}$$

Note that this is a fixed point function since μ is a function of θ. This can be used as a generic learning rule in online learning for canonical GLIMs. The stochastic gradient ascent is given by

$$\theta^{t+1} = \theta^t + \rho(y_n - \mu_n^t)x_n$$
$$\text{where } \mu_n^t = (\theta^t)^T x_n, \rho \text{ is a step size}$$

Alternative to stochastic gradient descent to speed up is batch learning algorithm:

$$\begin{aligned} H = \frac{dl^2}{d\theta d\theta^T} &= \frac{d}{d\theta^T}\sum_n (y_n - \mu_n)x_n \\ &= \sum_n x_n \frac{d\mu_n}{d\theta^T} \\ &= -\sum_n x_n \frac{d\mu_n}{d\eta_n}\frac{d\eta_n}{d\theta^T} \\ &= -\sum_n x_n \frac{d\mu_n}{d\eta_n} x_n^T \\ &= -X^T W X \end{aligned}$$

where the second but last equality is due to $\eta_n = \theta^T x_n$. $X = [x_n^T]$ is the design matrix and $W = diag(\frac{d\mu_i}{d\eta_i})_{i=1}^N$ can be computed via the second derivative of $A(\eta_n)$.

After obtaining Hessian $H = -X^T W X$ together with jacobian, we can apply Iteratively Reweighted Least Squares (IRLS).

Recall Newton-Raphson methods

$$\begin{aligned} \theta^{t+1} &= \theta^t + H^{-1}\nabla\theta l \\ &= H^{-1}(H\theta^t + \nabla\theta l) \\ &= (X^T W^t X)^{-1} X^T W^t z^t \end{aligned}$$

where $z^t = X\theta^t + (W^t)^{-1}(y - \mu^t)$ is the adjusted response. This can be understood as solving the the iteratively reweighted least squares problem

$$\theta^t = \arg\min_\theta (x - X\theta)^T W (z - X\theta)$$

1.3.2 Examples: Logistic and Linear regression

For logistic regression, conditional distribution is given by a Bernoulli

$$p(y|x) = \mu(x)^y (1 - \mu(x))^{1-y}$$

where $\mu(x) = \frac{1}{1+e^{-\eta(x)}}$. We know $p(y|x)$ is an exponential family function with mean $E(y|x) = \mu = \frac{1}{1+e^{-\eta(x)}}$ and canonical response function $\eta = \theta^T x$. Hence from previous section we know IRLS updates with

$$\frac{d\mu}{d\eta} = \mu(1 - \mu)$$

$$W = diag(\mu_i(1 - \mu_i)_{i=1}^N)$$

For linear regression, condition distribution is a Gaussian

$$\begin{aligned} p(y|x,\theta,\Sigma) &= \frac{1}{(2\pi)^{k/2}|\Sigma|^{1/2}} \exp\{-\frac{1}{2}(y-\mu(x))^T\Sigma^{-1}(y-\mu(x))\} \\ &= h(x)\exp\{-\frac{1}{2}\Sigma^{-1}(\eta^T(x)y - A(\eta))\} \end{aligned}$$

where $\mu(x) = \theta^T x = \eta(x)$. Thus $p(y|x)$ is an exponential family function with $E(y|x) = \mu = \theta^T x$ and canonical response function $\eta_1 = \theta^T x$. Since $\frac{d\mu}{d\eta} = 1$ thus in IRLS $W = I$ hence

$$\theta^{t+1} = \theta^t + (X^T X)^{-1} X^T (y - \mu^t)$$

which is reduced to steepest descent. If further take $t \to \infty$ we get the normal equation

$$\theta = (X^T X)^{-1} X^T Y$$

Remember that simple GMs (with one or two nodes) are the building blocks of complex GMs.

1.4 MLE for General BNs

1.4.1 Example

Assume the parameters for each CPD are globally independent and all nodes are fully observed then the log-likelihood function decomposes into a sum of local terms, one per node.

$$\begin{aligned} l(\theta, D) &= \log p(D|\theta) \\ &= \log \Pi_n (\Pi_i p(x_{n,i}|\mathbf{x}_{n,\pi_i}, \theta_i)) \\ &= \sum_i (\sum_n \log p(x_{n,i}|\mathbf{x}_{n,\pi_i}, \theta_i)) \end{aligned}$$

which allows us to utilize what we have learned from the small GM to instantiate each term in the above equation and get the result for general GM.

1.4.2 Decomposable Likelihood of a BN

Distribution defined by the DAG GM

$$p(x|\theta) = p(x_1|\theta_1)p(x_2|x_1,\theta_2)p(x_3|x_1,\theta_3)p(x_4|x_2,x_3,\theta_4)$$

leads us to learn four separate small BNs, each of which consists of a node and its parents.

Suppose now each CPD is represented as a table (multinomial) where

$$\theta_{ijk} := p(X_i = j | X_{\pi_i} = k)$$

and the sufficient statistics are counts of family configurations

$$n_{ijk} = \sum_n x_{n,i}^j x_{n,\pi_i}^k$$

and the log likelihood is

$$l(\theta, D) = \log \Pi_{ijk} \theta_{ijk}^{n_{ijk}} = \sum_{i,j,k} n_{ijk} \log \theta_{ijk}.$$

Using a Lagrange multiplier to enforce $\sum_j \theta_{ijk} = 1$ we get

$$\theta_{ijk}^{ML} = \frac{n_{ijk}}{\sum_{j'} n_{ij'k}}$$

In summary, learning BN with more nodes rely on local operations which build off that.

2 Parameter Estimation for Partially Observed GMs: EM Algorithm

2.1 Unobserved Variables

In previous sections we have seen parameter estimation for fully observed graphical models. However, often in practice some random variables in a graphical model may be unobserved, and for various reasons. Some random variables are imaginary quantities designed to capture the abstract data generation process and thus are not physically measurable (e.g. the latent variables in speech recognition models and mixture models); some others may be unobserved because of faulty sensors. As we will see, the fact that there are unobserved random variables makes parameter estimation trickier. Nonetheless, partially observed graphical models models remain useful in practice, and we will see how to use Expectation-Maximization to estimate their parameters.

2.1.1 Why is Learning Harder for Partially Observed GMs?

Let's consider the case of Gaussian Mixture Models, where the data is sampled from a mixture of Gaussian distributions by first sampling Z, the class indicator vector, and then sampling X from a Gaussian distribution with a class specific mean and co-variance matrix.

We can estimate the parameters of the graphical model using MLE. In a fully observed setting where Z is observed, we maximize the log likelihood function

$$\ell_c(\theta; x, z) = \log p(x, z|\theta) = \log p(x, z|\theta_z) + \log p(x|z, \theta_x)$$

which factors nicely into two terms with decoupled parameters θ_z and θ_x which can be optimized individually.

If we do not observe Z, then the likelihood function is the following,

$$\ell_c(\theta; x) = \log p(x|\theta) = \log \sum_z p(x, z|\theta)$$

where parameters θ_x and θ_z become coupled via marginalization. This objective is much harder to optimize than the objective in the fully observed setting. In the following sections, we will see the EM algorithm and how it can be seen as optimizing a surrogate of this objective.

2.2 Expectation-Maximization

Again let's consider Gaussian Mixture Models. Our goal is to estimate parameters of the GMM given partially observed data. We have just seen that the fully observed objective for GMM is much easier to optimize than the partially observed objective. What we are missing in the partially observed setting is the value for the latent variable Z. Hence, the goal of the E-Step step is to compute the expectation of Z so that the M-step can perform parameter estimation similar to how it is done in the fully observed setting, but using the expected value of Z (and in general its sufficient statistics) rather than the value of Z.

This can be formulated as follows, in the **E-step**, we compute the expected value of the the hidden variables (i.e. z_n^k) given the current estimate of the parameters,

$$\tau_n^{k(t)} = \langle z_n^k \rangle_{q(t)} = p(z_n^k = 1 | x, \mu^{(t)}, \Sigma^{(t)}) = \frac{\pi_k^{(t)} N(x_n | \mu^{(t)}, \Sigma^{(t)})}{\sum_i \pi_i^{(t)} N(x_n | \mu_i^{(t)}, \Sigma_i^{(t)})}$$

In the **M-step**, we perform parameter estimation using the current expected values for the hidden variables. We are optimizing the *expected complete log likelihood* which is discussed with more detail in Section 2.3.

$$\begin{aligned}
\pi_k^* &= \frac{\sum_n \langle z_n^k \rangle_{q(t)}}{N} = \frac{\sum_n \tau_n^{k(t)}}{N} = \frac{\langle n_k \rangle}{N} \\
\mu_k^{(t+1)} &= \frac{\sum_n \tau_n^{k(t)} x_n}{\sum_n \tau_n^{k(t)}} \\
\Sigma_k^{(t+1)} &= \frac{\sum_n \tau_n^{k(t)} (x_n - \mu_k^{(t+1)})(x_n - \mu_k^{(t+1)})^T}{\sum_n \tau_n^{k(t)}}
\end{aligned}$$

2.3 Complete and Incomplete Log Likelihoods

Using MLE, we want to learn the model parameters that maximize the likelihood of the data. In other words, we want to maximize the *complete log likelihood*, defined as

$$\ell_c(\theta; x, z) := \log p(x, z | \theta)$$

Maximizing this would be easy if all the variables were observed. However, when z is not observed, we instead have an *incomplete log likelihood*,

$$\ell_c(\theta; x) = \log p(x | \theta) = \log \sum_z p(x, z | \theta)$$

This objective doesn't decouple, so we cannot maximize it directly. Instead, we try to maximize a surrogate that lower bounds the objective we want. For any distribution $q(z)$ we define the *expected complete log likelihood* as

$$\langle \ell_c(\theta; x, z) \rangle_q := \sum_z q(z | x, \theta) \log p(x, z | \theta)$$

We can see that this is a lower bound using Jensen's inequality. The proof is as follows

$$\begin{aligned}
\ell(\theta; x) &= \log p(x|\theta) \\
&= \log \sum_z p(x, z|\theta) \\
&= \log \sum_z q(z|x) \frac{p(x, z|\theta)}{q(z|x} \\
&\geq \sum_z q(z|x) \log \frac{p(x, z|\theta)}{q(z|x)} \\
&= \langle \ell_c(\theta; x, z) \rangle_q + H_q
\end{aligned}$$

2.4 EM as Maximizing Free Free Energy

We define the *free energy* as follows:

$$F(q, \theta) := \sum_z q(z|x) \log \frac{p(x, z|\theta)}{q(z|x)} = \langle \ell_c(\theta; x, z) \rangle_q + H_q$$

Note that this is the second-to-last expression in the previous proof. Then we can view the EM algorithm as coordinate ascent on F.

2.4.1 E-Step

In the E-step, we maximize over q, and we can show that the solution is

$$q^{t+1} = \text{argmax}_q F(q, \theta^t) = p(z|x, \theta^t)$$

We can prove this by showing that this choice of q^{t+1} achieves the upper bound on F that we derived in the previous section

$$\begin{aligned}
F(p(z|x, \theta^t), \theta^t) &= \sum_z p(z|x, \theta^t) \log \frac{p(x, z|\theta^t}{p(z|x, \theta^t} \\
&= \sum_z p(z|x, \theta^t) \log p(x|\theta^t) \\
&= \log p(x|\theta^t) \\
&= \ell(\theta^t; x)
\end{aligned}$$

Given this result, we assume WLOG that $p(x, z|\theta)$ is a generalized exponential family distribution. Then the expected complete log likelihood is

$$\begin{aligned}
\langle \ell_c(\theta^t; x, z) \rangle_{q^{t+1}} &= \sum_z q(z|x, \theta^t) \log p(x, z|\theta^t) - A(\theta) \\
&= \sum_i \theta_i^t \langle f_i(x, z) \rangle_{q(z|x, \theta^t)} - A(\theta) \\
&\overset{p \sim \text{GLIM}}{=} \sum_i \theta_i^t \langle \eta_i(z) \rangle_{q(z|x, \theta^t)} \xi_i(x) - A(\theta)
\end{aligned}$$

2.4.2 M-Step

In the M-step, we now maximize over θ. We note that in the definition of free energy, the H_q term does not depend on θ, so we are just optimizing the first term.

$$\theta^{t+1} = \text{argmax}_\theta \langle \ell_c(\theta; x, z) \rangle_{q^{t+1}} = \text{argmax}_\theta \sum_z q(z|x) \log p(x, z|\theta)$$

Under optimal q^{t+1}, this is equivalent to solving a standard MLE of the fully observed model $p(x, z|\theta)$ with the sufficient statistics involving z replaced by their expectations w.r.t $p(z|x, \theta)$.

2.5 EM Algorithm for K-Means

In K-means, we assume there are K clusters and we want to find the parameters of the model (i.e. means of the K clusters) and the hidden variable (i.e. cluster assignments of data points). We can estimate the means iteratively by alternating between 1) computing cluster assignments at time t using the means at time t, and 2) using cluster assignments at time t to recompute the mean for the next iteration. This can be formalized as follows, in E-step

$$z_n^{(t)} = \text{argmin}_k (x_n - \mu_k^{(t)})^T \Sigma_k^{-1(t)} (x_n - \mu_k^{(t)})$$

and in M-step

$$\mu_k^{(t+1)} = \frac{\sum_n \delta(z_n^{(t)}, k) x_n}{\sum_n \delta(z_n^{(t)}, k)}$$

.

2.6 EM Algorithm for Gaussian Mixture Models (GMMs)

In GMMs, we assume that we have some data that is sampled from a mixture of k Gaussian distributions. The data $\{x_n\}$ are observed, but we do not know the parameters for the Gaussian distributions $\{\mu_k, \Sigma_k\}$. Let z_n be a latent class indicator vector and suppose we have a prior $\pi_k = p(z_n^k = 1)$ on the class labels. Then we can write its likelihood as

$$p(z_n) = \prod_k (\pi_k)^{z_n^k}$$

If we knew the class label for a data point, then the likelihood is

$$\begin{aligned} p(x_n|z_n^k = 1, \mu, \Sigma) &= N(x_n; \mu_k \Sigma_k) \\ &= \frac{1}{(2\pi)^{m/2} |\Sigma_k|^{1/2}} \exp\left(-\frac{1}{2} (x_n - \mu_k)^T \Sigma_k^{-1} (x_n - \mu_k) \right) \end{aligned}$$

Thus, we can combine the previous two equations and utilize the fact that z_n is a binary indicator vector to write the likelihood of x_n

$$\begin{aligned} p(x_n|\mu, \Sigma) &= \sum_k p(x_n, z_n^k = 1|\mu, \Sigma) \\ &= \sum_k p(z_n^k = 1|\pi) p(x_n|z_n^k = 1, \mu, \Sigma) \\ &= \sum_k \pi_k \cdot N(x_n; \mu_k \Sigma_k)) \end{aligned}$$

The complete log-likelihood is thus

$$\begin{aligned}\langle \ell_c(\theta; x, z)\rangle &= \sum_n \langle \log p(z_n|\pi)\rangle_{p(z|x)} + \sum_n \langle \log p(x_n|z_n, \mu, \Sigma)\rangle_{p(z|x)} \\ &= \sum_n \sum_k \langle z_n^k\rangle \log \pi_k - \frac{1}{2}\sum_n \sum_k \langle z_n^k\rangle \left((x_n - \mu_k)^T \Sigma_k^{-1}(x_n - \mu_k) + \log|\Sigma_k| + C\right)\end{aligned}$$

10-708: Probabilistic Graphical Models, Spring 2020

7: Variational Inference I

Lecturer: Eric P. Xing *Scribe: Yihang Shen, Yizhou He, Alex Gaudio, Amrit Singhal*

Over the last a few lectures we have learned exact inference, but when is exact inference is expensive or even impossible? This motivates methods for approximate inference.

1 Overview of Variational Methods

- 15 years ago, a third of papers at NIPS were on this topic. The field has evolved tremendously.
- Variational Inference converts inference into optimization. - **Variational** is fancy name for optimization based formulations to approximate the desired solution by relaxing/approximating the intractable optimization problem.

Example: Linear system of equations when A is too large to invert:

$$Ax = b, A \succ 0$$

The solution is $x^\star = A^{-1}b$, however this is intractable for very large system as solving A^{-1} become very expensive.

This problem can be rewritten in variational formulation,

$$x^\star = \text{argmin}_x \frac{1}{2} x^T Ax - bx$$

turns to be a optimization problem where conjugate gradient can be applied to the problem.

High level idea of Variational Inference:

There is a true distribution P that we wish to perform inference on, but we can't realistically do it. Therefore, we want to approximate P with another distribution Q. Variational Inference assumes the existence of P and Q, and also requires a way to measure distance between P and Q (usually KL Divergence).

- Inference: The distribution P
- Challenge: Direct inference on P is often intractable
- Variational approach: Project P to a tractable family of distributions Q.
- Perform inference on projected Q.
- Measure of distance: Test how good the approximation is, for example KL divergence.

Next class, we will cover the mathematics of variational inference in more detail. Now, we focus on applications.

2 Applications

2.1 The generalized algorithm design problem

More generally, how do we get started modeling a task? This is a design problem with many parts. These parts can be solved one step at a time.

- What is the task? Classification? Clustering?
- Data inputs and outputs: Is it continuous, binary or counts?
- Model: We need to choose one.
- Inference: Also need to choose one. For instance: Exact, Variational, MCMC
- Evaluation: How do we measure success? Visualize the results, score them, interpret them.

We will apply these steps to a task of text processing:

3 Probabilistic Topic Models

3.1 The Task and Data: Document Embedding on Text Documents

We have a stack of text documents. We want to represent relationships between documents for the purpose of grouping or categorizing them, as well as classifying and comparing similarity. We also want to be able to explore how documents change over time.

If we could meaningfully reduce each document to a point in some high dimensional space, we may be able to find a solution that is useful to all these problems. Our task is to create a document embedding. The inputs will a set of documents, each document contains text. The outputs will be a set of points in space.

3.2 Data Representation: A review of document embedding models

We consider various models that have been used to approach the document embedding task.

Bag of words representation.

Each document is a vector in the word space. This approach has significant disadvantage that it is too high dimensional, sparse, and ignores the order of words in a document.

Topic models

Topics, instead of words, intuitively tag, label or characterize a text. And a text may contain many topics. The idea is to recover for each document a single vector containing the mixing proportion of topics. In a PGM setting, do topic discovery on unstructured collection of texts to get a structured topic network, then generate a topic simplex. We will briefly present LSI and then connect it topic models.

Latent Semantic Index (LSI)

This is an old approach from the 70s, very similar to topic models. The basic idea is to perform singular value decomposition to decompose a word-document matrix into three sub-matrices. Primarily an algebraic solution rather than probabilistic one.

$$X = W\Lambda D^T$$

A singular value decomposition. Here X is the contexts collections, the x axis are documents, and the y axis represents word vectors. W relates topics to words, Λ relates topics to topics (the diagonal has singular values), and D^T relates documents to topics.

Relation to Topic models. Topic models have a very similar form. Algebraically, the only difference is the absence of a Λ matrix:

$$X = WD^T$$

where $X = P(w)$, $W = P(w \mid z)$, and $D^T = P(z)$.

Admixture models:

Description: Objects are bags of elements, mixtures are distributions over elements, mixing vector θ represents each mixtures' contributions for objects. Objects are generated from picking mixture components from θ and picking elements from that component.

Advantage: Allowing much richer mechanism for mixture of sources.

Differences between admixture models (similar to topic models) and mixture models:

For the mixture model, we assume that each document D is generated from a latent variable Z, the graph is represented as: $Z \rightarrow D$, but for the admixture model, each word W_i in a document is generated by a latent variable Z_i, and these latent variables are parameterized by θ, the graph can be constructed as: $\theta \rightarrow Z_i \rightarrow W_i$.

Figure 1 shows the graphical representation of topic models. The parameter θ is drawn from a prior distribution (in LDA it is Dirichlet distribution), then z_i is drawn from multinomial distribution with parameter θ, after that $w_i \mid z_i$ is drawn from multinomial distribution with parameter β, and β represents possible topics of the document.

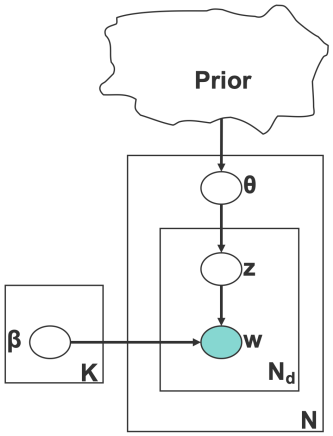


Figure 1: The graphical representation of a general topic model

LoNTAM

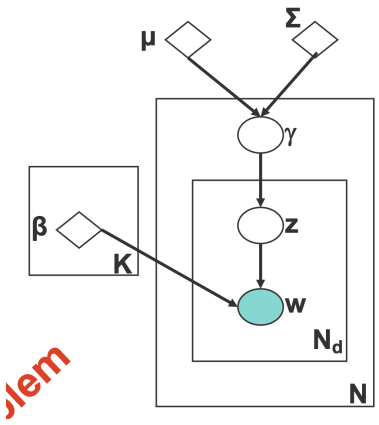


Figure 2: LoNTAM

There is another similar model called LoNTAM (Figure 2), the differences between Figure 2 and Figure 1 is that in LoNTAM, the prior used for sampling θ (γ in Figure 2) is multivariate Gaussian distribution.

Pros and cons of LDA and LoNTAM:

LDA:

Pros: Conjugate prior means more efficient inference.

Cons: Can only capture variations in each topic's intensity independently.

LoNTAM:

Pros: Capture the intuition that some topics are highly correlated and can rise up in intensity together.

Cons: Not a conjugate prior implies more difficult inference.

The joint distribution of Figure 1 is:

$$p(\beta, \theta, z, w) = \prod_{k=1}^{K} p(\beta_k \mid \eta) \prod_{d=1}^{D} p(\theta_d \mid \alpha) \prod_{n=1}^{N} p(z_{dn} \mid \theta_d) p(w_{dn} \mid z_{dn}, \beta)$$

However, it is very hard to do the exact inference according to that distribution, therefore we need methods for approximate inference.

3.3 Variational Inference

During class, slide 28 was missing many details. We fill them in here.

We have that:

$$\begin{aligned}
\log p(x) &= \log \frac{p(x,z,\theta)}{p(z,\theta \mid x)} = \log \frac{p(x,z,\theta)q(z,\phi \mid x)}{q(z,\phi \mid x)p(z,\theta \mid x)} \\
&= \int_z q(z,\phi \mid x) \log \frac{p(x,z,\theta)q(z,\phi \mid x)}{q(z,\phi \mid x)p(z,\theta \mid x)} dz \\
&= \int_z q(z,\phi \mid x) \log \frac{q(z,\phi \mid x)}{p(z,\theta \mid x)} dz + \int_z q(z,\phi \mid x) \log \frac{p(x,z,\theta)}{q(z,\phi \mid x)} dz \\
&= KL(q(z,\phi \mid x) \| p(z,\theta \mid x)) + \int_z q(z,\phi \mid x) \log \frac{p(x,z,\theta)}{q(z,\phi \mid x)} dz \\
&\geq \int_z q(z,\phi \mid x) \log \frac{p(x,z,\theta)}{q(z,\phi \mid x)} dz := L(\theta,\phi \mid x)
\end{aligned}$$

Instead of maximizing $\log p(x)$ we can maximize the lower bound $L(\theta, \phi \mid x)$, or equivalently, minimize free energy:

$$F(\theta,\phi \mid x) = -\log p(x) + KL(q(z,\phi \mid x) \| p(z,\theta \mid x))$$

We can use EM algorithm to maximize $L(\theta, \phi \mid x)$, for the E-step, fix θ, maximize L with respect to ϕ, if closed solution exists, we have

$$q^\star(z,\phi|x) \propto \exp(\log p(x,z,\theta))$$

For the M-step, fix ϕ, maximize L with respect to θ.

In the practical case of Figure 1, it is hard to do inference on the true posterior:

$$p(\beta,\theta,z \mid w) = \frac{p(\theta,\beta,z,w)}{p(w)}$$

Mean Field Approximation With mean field approximation, we assume q is fully factorized, that is:

$$q(\beta,\theta,z \mid \lambda,\gamma,\phi) = \prod_{k=1}^{K} q(\beta_k \mid \lambda_k) \prod_{d=1}^{D} q(\theta_d \mid \gamma_d) \prod_{n=1}^{N} q(z_{dn} \mid \phi_{dn})$$

We can have parametric form for each marginal factor:

$q(\beta_k \mid \lambda_k) = \text{Dirichlet}(\beta_k \mid \lambda_k)$

$q(\theta_d \mid \gamma_d) = \text{Dirichlet}(\theta_d \mid \gamma_d)$

$q(z_{dn} \mid \phi_{dn}) = \text{Multinomial}(z_{dn} \mid \phi_{dn})$

Therefore, we can learn parameters as the E-step:

$$\gamma^*, \lambda^*, \phi^* = argmin_{\gamma,\lambda,\phi} KL(q(\beta,\theta,z \mid \lambda,\gamma,\phi) \| p(\beta,\theta,z \mid w,\alpha,\eta))$$

Latent Dirichlet Allocation (LDA): For LDA, the optimal MF approximation can be computed in a closed form. Specifically, we define the following distributions:

- $p(\theta_d|\alpha) \propto \exp\left\{\sum_{k=1}^{K}(\alpha_k - 1)\log\theta_{dk}\right\}$ is the topic proportion distribution, which is Dirichlet.
- $p(z_{dn}|\theta_d) = \exp\left\{\sum_{k=1}^{K} \mathbf{1}[z_{dn}=k]\log\theta_{dk}\right\}$ is the word-label frequency distribution, which is Multinomial.

And plug them into the following update step, the result of which is a Dirichlet distribution:

$$\begin{aligned} q(\theta_d) &\propto \exp\left\{\mathbb{E}_{\prod_n q(z_{dn})}\left[\log p(\theta_d|\alpha) + \sum_n \log p(z_{dn}|\theta_d)\right]\right\} \\ &\propto \exp\left\{\sum_{k=1}^{K}\left(\sum_{n=1}^{N} q(z_{dn}=k) + \alpha_k - 1\right)\log\theta_{dk}\right\} \end{aligned}$$

Also update the marginals:

$$q(z_{dn}=k|\phi_{dn}) = \phi_{dn}(k) = \beta_k(w_{dn})\exp\left\{\Psi(\gamma_d(k)) - \Psi(\sum_{j=1}^{K}\gamma_d(j))\right\}$$

$$\lambda_k(j) = \eta(j) + \sum_{d=1}^{D}\sum_{n=1}^{N_d}\phi^*(k)\mathbf{1}[w_{dn}=j]$$

The $\lambda_k(j)$ term gives you a frequency (count) of the j^{th} word frequency in topic k, which is clear from the summation over every document and every word and the delta function $\mathbf{1}[...]$.

Iterating on these equations to convergence results in the MF approximation to the posterior. The algorithm formed by this iteration is coordinate ascent for LDA.

Variational Learning: the maximization step

For the M-step, usually people do not learn θ, as for the Bayesian, θ are integrated out. But one can still have MAP of θ, $MAP_{\theta,\beta}L(\theta,\phi \mid x)$, and one can even perform estimation of the hyper-parameters $MAP_{\alpha,\eta}L(\theta,\phi \mid x)$.

4 More on Mean Field Approximation

4.1 Naive Mean Field

The naive mean field algorithm approximates $p(X)$ by a fully factorized $q(X) = \prod_i q_i(X_i)$.

In the case of Boltzmann distribution

$$p(X) = \frac{1}{Z}\exp\left(\sum_{i<j} q_{ij}X_iX_j + q_{i0}X_i\right)$$

The mean field update equation for this is

$$\begin{aligned} q_i(X_i) &= \exp\left(\theta_{i0}X_i + \sum_{j\in N_i}\theta_{ij}X_i\langle X_j\rangle_{q_j} + A_i\right) \\ &= p(X_i|\{\langle X_j\rangle_{q_j} : j\in N_i\}) \end{aligned}$$

where $\langle X_j\rangle$ resembles a message sent from node j to node i. $\{\langle X_j\rangle_{q_j} : j\in N_i\}$ forms the "mean field" applied to X_i from its neighbourhood, as shown in Figure 3a.

(a) MF update (b) GMF update

Figure 3: Messages passed while Mean field update

4.2 Generalised Mean Field

We can also apply more general forms of the mean field approximations, i.e. clusters of disjoint latent variables are independent, while the dependencies of latent variables in each clusters are preserved shown in Figure 3b.

4.2.1 From mean field approximation to Gibbs free energy

Assume that we are given a disjoint clustering $\{C_1, \ldots, C_K\}$ of all variables. Let

$$q(X) = \prod_{i=1}^{K} q_i(X_{C_i})$$

Then, we get the mean field energy as

$$G_{MF} = \sum_i \sum_{X_{C_i}} \prod_i q_i(X_{C_i}) E(X_{C_i}) + \sum_i \sum_{X_{C_i}} q_i(X_{C_i}) \ln q_i(X_{C_i})$$

For example, in the case of naive mean field, we get

$$G_{MF} = \sum i < j \sum_{x_i, x_j} q(x_i) q(x_j) \phi(x_i, x_j) + \sum_i \sum_{x_i} q(x_i) \phi(x_i) + \sum_i \sum_{x_i} q(x_i) \ln q(x_i)$$

Using clustering, no matter what clustering we use, we will **never** be able to be equal to the exact Gibbs free energy, but it **always** defines a lower bound of the likelihood.

So, we optimize each $q_i(X_{C_i})$, and do inference in each $q_i(x_c)$ using any tractable algorithm.

4.2.2 Generalised Mean Field Theorem

The optimum GMF approximation to the cluster marginal is isomorphic to the cluster posterior of the original distribution given the internal evidence and its generated mean fields

$$q_i^*(X_{H,C_i}) = p(X_{H,C_i} | X_{E,C_i}, \langle X_{H,MB_i} \rangle_{q_{j \neq i}})$$

The GMF algorithm is simply iterate over each q_i.

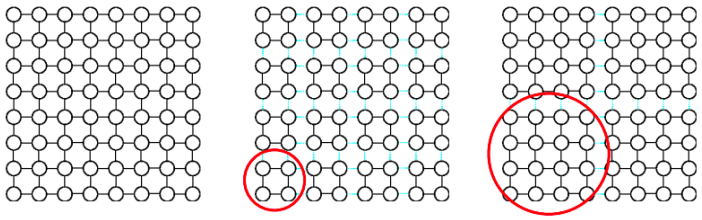


Figure 4: Generalised Mean Field for Ising models

4.2.3 Convergence Theorem

The GMF algorithm in guaranteed to converge to a local optimum, and provides a lower bound for the likelihood of the evidence(or partition function) of the model.

Example: GMF approximation to Ising models

Figure 4 shows two possible clusterings for the Ising model, one where we form 2×2 clusters, and the other where we form 4×4 clusters.

We obtain the cluster marginal of a square block C_k as

$$q(X_{C_k}) \propto \exp\left(\sum_{i,j \in C_k} \theta_{ij} X_i X_j + \sum_{i \in C_k} \theta_{i0} X_i + \sum_{i \in C_k, j \in MB_k, k' \in MB_{C_k}} \theta_{ij} X_i \langle X_j \rangle_{q(X_{C_{k'}})}\right)$$

Note that this is virtually a reparameterized Ising model of small size.

Key takeaway of class today

Know the Variational Inference principle, and in particular, the Mean Field Approximation principle. MFA is very powerful. You are effectively removing all/most of the edges resulting in a product of all singletons/clusters.

10-708: Probabilistic Graphical Models, Spring 2020

8: Variational Inference 2

Lecturer: Xun Zheng *Scribe: Zhaoqi Cheng, Eric Dexheimer, Yifei Li, Chiyu Wu, Junwoong Yoon*

1 Practice of Variational Inference

1.1 Stochastic Variational Inference

1.1.1 Drawback of Coordinate Ascent

In the previous lecture on variational inference with coordinate ascent, there were two types of parameters to update: the local variables, which were documents, and the global parameters λ. The algorithm is summarized in a high level in Algorithm 1:

Algorithm 1: Coordinate Ascent

```
Initialize global parameters λ;
repeat
    for each document d ∈ {1, 2, ..., D} do
        Update document-specific variational distributions;
    end
    Update global parameters λ;
until Lower bound L(λ) converges;
```

As in the algorithm, all documents are looped through first before updating the global parameters. This becomes a problem when there are too many documents, so the updating of the model will be infrequent.

To combat this, the lower bound can be separated into a per-data point term parameterized by local parameters ϕ_i, and a global term parameterized by the global parameter λ:

$$\mathcal{L}(\lambda, \phi_{1:n}) = \underbrace{\mathbb{E}_q\left[\log p(\beta) - \log q(\beta|\lambda)\right]}_{\text{global contribution}} + \sum_{i=1}^{n} \underbrace{\left\{\mathbb{E}_q\left[\log p(w_i, z_i|\beta] - \log q(z_i|\phi_i)\right\}\right.}_{\text{per-data point contribution}}. \tag{1}$$

Let

$$f(\lambda) \coloneqq \mathbb{E}_q\left[\log p(\beta) - \log q(\beta|\lambda)\right] \tag{2}$$

be the global contribution, and

$$g_i(\lambda, \phi_i) \coloneqq \mathbb{E}_q\left[\log p(w_i, z_i|\beta] - \log q(z_i|\phi_i)\right. \tag{3}$$

be the per-data point contribution of the i-th data point. Then the lower bound can be simplified as

$$\mathcal{L}(\lambda, \phi_{1:n}) = f(\lambda) + \sum_{i=1}^{n} g_i(\lambda, \phi_i). \tag{4}$$

To optimize our objective, we can maximize it w.r.t. parameters $\phi_{1:n}$ first, which results in a one-parameter lower bound

$$\mathcal{L}(\lambda) = f(\lambda) + \sum_{i=1}^{n} \max_{\phi_i} g_i(\lambda, \phi_i). \tag{5}$$

Let the optimizer for each data point be

$$\phi_i^* = \arg\max_{\phi} g_i(\lambda, \phi_i), \tag{6}$$

then the gradient of the one-parameter lower bound $\mathcal{L}(\lambda)$ has the following form,

$$\frac{\partial \mathcal{L}(\lambda)}{\partial \lambda} = \frac{\partial f(\lambda)}{\partial \lambda} + \sum_{i=1}^{n} \frac{\partial g_i(\lambda, \phi_i)}{\partial \lambda}. \tag{7}$$

Since the per-data point term is a sum of each data point contribution, the derivative of each data-point can be summed to compute the derivative of the per-data point term as well, as shown in the second term of RHS of equation 7. This allows us to use stochastic gradient algorithms and update the model more frequently for better convergence. And once the global parameter λ is estimated, each ϕ_i can be estimated online if needed.

1.1.2 Stochastic Variational Inference using Natural Inference

However, the gradient of the lower bound with respect to the parameters will not be the steepest direction, since the parameters represent a distribution. For example, a pair of Gaussian distributions with the same two means will have the same "distance" in the mean parameter space, but the KL divergence is very different, as seen by the overlap in Figure 1. In addition, parameterizing the spread using variance or the inverse variance would lead to different gradients due to the difference in representation.

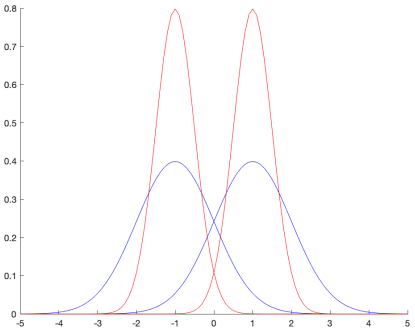


Figure 1: Two pairs of Gaussian distributions with the same two means

The reason is, using gradient algorithms, the objective function $f(\theta)$ at θ_0 is approximated by a local quadratic second-order approximation

$$f(\theta) \approx f(\theta_0) + \nabla f(\theta_0)^\top (\theta - \theta_0) + \frac{1}{2t}(\theta - \theta_0)^\top (\theta - \theta_0), \tag{8}$$

where $\nabla^2 f(\theta_0)$ is replaced by $\frac{1}{t} I$. But this approximation will not work for distributions. Instead, we use the Fisher Information Matrix to for the gradient since the variational distribution is in the exponential family.

Now, we can perform stochastic variational inference by sampling data points and updating the parameters using the natural gradient as in Algorithm 2.

Algorithm 2: Stochastic Variational Inference using Natural Inference

Initialize global parameters λ_0, $t = 0$;
Set step-size schedule ρ_t;
for $t = 1, \ldots, \infty$ **do**
  Sample a data point $i \sim \text{Uniform}(1, \ldots, n)$;
  Compute the optimal local parameter $\phi_i^*(\lambda_t)$;
  Perform natural gradient ascent on global parameter λ,
    $\lambda_{t+1} \leftarrow \lambda_t + \rho_t g(\lambda_t) = (1 - \rho_t)\lambda_t + \rho_t \left(\eta + nt_{\phi_i^*}(x_i)\right)$;
end

1.2 Black-box Variational Inference (BBVI)

We have derived variational inference specific for Latent Dirichlet allocation (LDA) above. But there are innumerable conjugate/non-conjugate models. We want to have a general solution which does not entail model-specific work. This solution will serve like a black box, which outputs a variational distribution when input any model and massive data. It is called Black-box Variational Inference (BBVI).

There are generally two types of BBVI: BBVI with the score gradient, and BBVI with the reparameterization gradient. The latter is the foundation of Variational AutoEncoder (VAE), which will be discussed more in lecture 12.

1.2.1 BBVI with the Score Gradient

Consider a probabilistic model where x is the observed variable and z is the latent variable. The corresponding variational distribution is $q(z|\lambda)$. The evidence lower bound (ELBO) is

$$\mathcal{L}(\lambda) \coloneqq \mathbb{E}_{q_\lambda(z)} \left[\log p(x, z) - \log q(z|\lambda)\right]. \tag{9}$$

Its gradient w.r.t. λ is

$$\nabla_\lambda \mathcal{L} = \mathbb{E}_q \left[\nabla_\lambda \log q(z|\lambda)(\log p(x, z) - \log q(z|\lambda))\right], \tag{10}$$

where $\nabla_\lambda \log q(z|\lambda)$ is called the *score function*.

To derive equation 10, we need two essential facts:

1. $\nabla_\lambda q_\lambda(z) = \frac{1}{q_\lambda(z)} \nabla_\lambda q_\lambda(z) = q_\lambda(z) \nabla_\lambda q_\lambda(z)$;
2. $\mathbb{E}_q \left[\nabla_\lambda \log q_\lambda(z)\right] = 0$, i.e. the expectation of the gradient of log-likelihood (score function) is zero.

Based on these two facts, we can derive the score gradient of our ELBO,

$$
\begin{aligned}
\nabla_\lambda \mathcal{L} &= \nabla_\lambda \int_z \left[q_\lambda(z) \log p(x,z) - q_\lambda(z) \log q_\lambda(z)\right] \mathrm{d}z \\
&= \int_z \left\{ \log p(x,z) \nabla_\lambda q_\lambda(z) - \left[\nabla_\lambda q_\lambda(z) \log q_\lambda(z) + q_\lambda(z) \frac{1}{q_\lambda(z)} \nabla_\lambda q_\lambda(z) \right] \right\} \mathrm{d}z \\
&= \int_z \nabla_\lambda q_\lambda(z) \left[\log p(x,z) - \log q_\lambda(z) - 1\right] \mathrm{d}z \\
&= \int_z q_\lambda(z) \nabla_\lambda q_\lambda(z) \left[\log p(x,z) - \log q_\lambda(z) - 1\right] \mathrm{d}z \\
&= \mathbb{E}_{q_\lambda} \left[\nabla_\lambda q_\lambda(z) \left(\log p(x,z) - \log q_\lambda(z)\right)\right] - \mathbb{E}_{q_\lambda} \left[\nabla_\lambda q_\lambda(z)\right] \\
&= \mathbb{E}_{q_\lambda} \left[\nabla_\lambda q_\lambda(z) \left(\log p(x,z) - \log q_\lambda(z)\right)\right].
\end{aligned}
$$

Using the score gradient in equation 10, We can take advantage of Monte Carlo to compute the noisy unbiased gradient of the ELBO with samples from the vaiational distribution,

$$
\nabla_\lambda \mathcal{L} \approx \frac{1}{S} \sum_{s=1}^{S} \nabla_\lambda \log q(z_s|\lambda)(\log p(x,z_s) - \log q(z_s|\lambda)), \tag{11}
$$

where $z_s \sim q(z|\lambda)$.

2 Theory of Variational Inference

We have seen how variational inference can be viewed as an optimization problem in which we approximate the true solution by relaxing/approximating the intractable optimization problem. Now, a unified view based on the variational principle is presented where the mean-field approximation is an inner approximation of the set of canonical parameters, while loopy belief propagation is an outer approximation.

Thus far, common inference problems include marginal distributions and normalization constants, but now we want to represent these quantities in a variational form via exponential families and convex analysis.

2.1 Graphical Models as Exponential Families

The exponential family described previously is:

$$
p_\theta(x_1, ..., x_n) = \exp\left(\theta^T \phi(x) - A(\theta)\right), \tag{12}
$$

where θ is the set of canonical parameters, $\phi(x)$ are the sufficient statistics, and $A(\theta)$ is the log partition function for normalization,

$$
A(\theta) = \log \int \exp\left\{\theta^\top \phi(x)\right\} \mathrm{d}x.
$$

Note that whatever distribution p_θ, the log partition function $A(\theta)$ is a convex function, which can be proved via its Hessian. There is also a constraint for domain of the effective canonical parameters that the value of the log partition function $A(\theta)$ must be finite:

$$
\Omega \coloneqq \left\{\theta \in \mathbb{R}^d | A(\theta) < +\infty\right\}.
$$

Both Gaussian MRFs and discrete MRFs can be represented in this way. The Gaussian MRF has a joint distribution

$$p(\mathbf{x}) = \exp\left\{\frac{1}{2}\langle\Theta, \mathbf{x}\mathbf{x}^T\rangle - A(\Theta)\right\}, \text{ where } \Theta = -\Lambda, \tag{13}$$

with sufficient statistics

$$\{x_s^2, s \in V; x_s x_t, (s,t) \in E\}. \tag{14}$$

The discrete MRF joint distribution can be parameterized as

$$p(\mathbf{x};\theta) = \exp\left\{\underbrace{\sum_{s\in V}\sum_{j}\theta_{s;j}\mathbb{I}_j(x_s)}_{\text{marginal potential}} + \underbrace{\sum_{(s,t)\in E}\theta_{st;jk}\mathbb{I}_j(x_s)\mathbb{I}_k(x_t)}_{\text{pair-wise potential}}\right\}, \tag{15}$$

where $\mathbb{I}$ denotes the binary indicator function. The indicators can also be regarded as one-hot variables. One example of such distribution is the Ising model.

The exponential family is useful because computing the expectation over sufficient statistics given canonical parameters yields the marginals, which is called the marginal/pair-wise inference:

$$\mu_{s;j} = \mathbb{E}_p[\mathbb{I}_j(X_s)] = \mathbb{P}[X_s = j] \quad \forall j \in \mathcal{X}_s, \tag{16}$$

$$\mu_{st;jk} = \mathbb{E}_p[\mathbb{I}_{st;jk}(X_s, X_t)] = \mathbb{P}[X_s = j, X_t = k] \quad \forall (j,k) \in \mathcal{X}_s \in \mathcal{X}_t. \tag{17}$$

The sufficient statistics and canonical parameters are duals, in that one can be computed from the other. In addition, the normalizer yields the log partition function:

$$\log Z(\theta) = A(\theta). \tag{18}$$

The above duality is presented for the Bernoulli distribution whose joint distribution is

$$p(x;\theta) = \exp\left(\theta x - A(\theta)\right), x \in \{0,1\}, A(\theta) = \log(1 + e^\theta), \tag{19}$$

and the inference (computing the mean parameter) is

$$\mu(\theta) = \mathbb{E}_\theta[X] = 1 \cdot p(X = 1;\theta) + 0 \cdot p(X = 0;\theta) = \frac{e^\theta}{1 + e^\theta}. \tag{20}$$

Now we want to formulate the inference in a variational manner, i.e. as an optimization problem.

2.2 Conjugate Dual

First, we define the conjugate dual function for a function $f(\theta)$:

$$f^*(\mu) := \sup_\theta\{\langle\theta, \mu\rangle - f(\theta)\} \tag{21}$$

A visual representation of the conjugate dual can be seen in Figure 2. The conjugate dual is always a convex function since it is a point-wise supremum of a class of linear functions. Note that the dual is the dual of itself if f is convex and lower semi-continuous, i.e.

$$f(\theta) = \sup_\mu\{\langle\theta, \mu\rangle - f^*(\mu)\}. \tag{22}$$

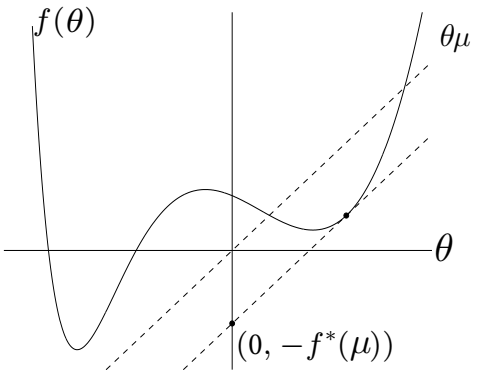


Figure 2: Example of conjugate dual function

In this sense the log partition function can be written this way under its domain:

$$A(\theta) = \sup_{\mu}\{\langle\theta, \mu\rangle - f^*(\mu)\}, \quad \theta \in \Omega. \tag{23}$$

This way, the dual variable μ has a natural interpretation as the mean parameter.

2.2.1 Example: Computing Mean Parameter of Bernoulli Distribution

Firstly consider a simple example of the Bernoulli distribution. Using its partition function in equation 19, its conjugate dual is

$$A^*(\mu) \coloneqq \sup_{\theta\in\mathbb{R}} \{\mu\theta - \log\left[1 + \exp(\theta)\right]\}. \tag{24}$$

It's kind of cheating because we have already known the form of $A(\theta)$. Solving the equation we write down the stationary condition which the optimal θ will satisfy

$$\mu = \frac{e^{\theta^*}}{1 + e^{\theta^*}}. \tag{25}$$

Inversing this condition we know the θ^* is

$$\theta^* = \log\frac{\mu}{1-\mu}, \ \mu \in (0, 1). \tag{26}$$

Plug this optimal θ^* into equation 24,

$$A^*(\mu) = \begin{cases} \mu\log\mu + (1-\mu)\log(1-\mu), & \text{if } \mu \in (0,1), \\ +\infty, & \text{otherwise.} \end{cases} \tag{27}$$

As we see here, $A^*(\mu)$ can only be defined on a restricted domain of μ. Using the dual of dual, we can solve the original partition function $A(\theta)$,

$$A(\theta) = \sup_{\mu\in(0,1)} \{\mu\cdot\theta - A^*(\mu)\}. \tag{28}$$

The optimum is achieved at

$$\mu(\theta) = \frac{e^{\theta}}{1 + e^{\theta}}, \tag{29}$$

which is exactly the mean.

2.2.2 Computation of Conjugate Dual

The example of Bernoulli is by no means practical. It's just for purpose of understanding. But in general, the process of the variational representation is similar. Given a distribution of exponential family

$$p(x;\theta) = \exp\left\{\sum_{i=1}^{d}\theta_i\phi_i(x) - A(\theta)\right\}, \tag{30}$$

we firstly write down its dual function

$$A^*(\theta) \coloneqq \sup_{\theta\in\Omega}\left\{\langle\mu,\theta\rangle - A(\theta)\right\}. \tag{31}$$

In order to represent this dual function, the optimal θ^* should satisfy the stationary condition

$$\mu - \nabla A(\theta^*) = 0. \tag{32}$$

Note that the derivatives of A essentially yields the mean parameters, i.e.

$$\frac{\partial A}{\partial\theta_i}(\theta) = \mathbb{E}_\theta\left[\phi_i(x)\right] = \int\phi_i(x)p(x;\theta)\mathrm{d}x. \tag{33}$$

Thus the stationary condition becomes

$$\mu = \mathbb{E}_{\theta^*}\left[\phi(x)\right]. \tag{34}$$

However, we should be careful that μ is restricted to a domain Ω. Now assume there is a solution $\theta(\mu)$ s.t. $\mu = \mathbb{E}_\theta\left[\phi(X)\right]$, then the dual takes the form

$$A^*(\mu) = \langle\theta(\mu),\mu\rangle - A(\theta(\mu)) = \mathbb{E}_{\theta(\mu)}\left[\langle\theta(\mu),\phi(x)\rangle - A(\theta(\mu)\right] \tag{35}$$

$$= \mathbb{E}_{\theta(\mu)}\left[\log p(x;\theta(\mu))\right], \tag{36}$$

which is derived by plugging in the optimal $\theta(\mu)$ and the definition of the joint probability of this exponential distribution.

Recall that the entropy is defined as,

$$H(p(x)) = -\int p(x)\log p(x)\mathrm{d}x, \tag{37}$$

so the dual is

$$A^*(\mu) = -H(p(x;\theta(\mu))), \tag{38}$$

when there is such a solution $\theta(\mu)$.

2.3 Marginal Polytope

Though we provide a framework for general exponential families/graphical models above, there are still several problems. Computing the entropy is generally intractable, and the constrain set Ω of mean parameter is hard to characterize. For the latter one, the answer is the marginal polytope.

For any distribution $p(x)$ (which does not necessarily belong to exponential family) and a set of sufficient statistics $\phi(x)$, define a vector of mean parameters as

$$\mu_i = \mathbb{E}_p\left[\phi_i(X)\right] = \int\phi_i(x)p(x)dx. \tag{39}$$

The set of all realizable mean parameters $\mathcal{M} := \left\{\mu \in \mathbb{R}^d \mid \exists p \text{s.t.} \mathbb{E}_p[\phi(X)] = \mu\right\}$ is a convex set.

When $p(x)$ belongs to exponential families, $\mathcal{M}$ is called marginal polytope and has the following convex hull representation:

$$\mathcal{M} = \left\{\mu \in \mathbb{R}^d \mid \sum_{x \in \mathcal{X}^m} \phi(x)p(x) = \mu, \text{ for some } p(x) \geq 0, \sum_{x \in \mathcal{X}^m} p(x) = 1\right\} = \text{conv}\left\{\phi(x), x \in \mathcal{X}^m\right\} \tag{40}$$

By Minkowski-Weyl Theorem, any non-empty convex polytope can be characterized by a finite collection of linear inequality constraints

$$\mathcal{M} = \left\{\mu \in \mathbb{R}^d | a_j^\top \mu \geq b_j, \forall j \in \mathcal{J}\right\} \tag{41}$$

where $|\mathcal{J}|$ is finite.

2.4 Variational Principle and Approximation

The marginal polytope is constrained by a set of half-planes. The number of such half-planes is called *facet complexity*. For a tree graphical model, the facet complexity grows only linearly in the graph size. However, for a general graph it grows so fast that it's extremely hard to characterize the marginal polytope. Thus we want to introduce some variational method.

As shown before, the dual function takes the form:

$$A^*(\mu) = \begin{cases} -H\left(p_{\theta(\mu)}\right) & \text{if } \mu \in \mathcal{M}^\circ \\ +\infty & \text{if } \mu \notin \overline{\mathcal{M}} \end{cases}, \tag{42}$$

where $\theta(\mu)$ satisfies $\mu = \mathbb{E}_{\theta(u)}[\phi(X)]$.

The exact variational formulation is

$$A(\theta) = \sup_{\mu \in \mathcal{M}} \left\{\theta^T \mu - A^*(\mu)\right\}, \tag{43}$$

where $\mathcal{M}$ is the marginal polytope and A^* is the conjugate dual (negative entropy function).

As discussed, the marginal polytope $\mathcal{M}$ is difficult to characterize, while the conjugate dual has no explicit form. To approximate, there are 2 common methods:

1. Mean field approximation method: non-convex inner bound and exact form of entropy;
2. Bethe approximation and loopy belief propagation: polyhedral outer bound and non-convex Bethe approximation.

Here we will focus on the former one.

2.4.1 Mean Field Approximation

For an exponential family with sufficient statistics ϕ defined on an arbitrary graph G, the set of realizable mean parameter set is:

$$\mathcal{M}(G; \phi) = \left\{\mu \in \mathbb{R}^d \mid \exists p \text{ s.t. } E_p[\phi(X)] = \mu\right\} \tag{44}$$

The idea is to restrict p to a subset of distributions associated with a tractable subgraph. For instance, we can transform a general graph with mean parameter set $\Omega = \left\{\theta \in \mathbb{R}^d \mid A(\theta) < +\infty\right\}$ to subgraph F_0 with $\Omega\left(F_0\right) = \left\{\theta \in \Omega \mid \theta_{(s,t)} = 0 \forall (s,t) \in E\right\}$ or T with $\Omega(T) = \left\{\theta \in \Omega \mid \theta_{(s,t)} = 0 \forall (s,t) \in E(T)\right\}$, as illustrated in Figure 3.

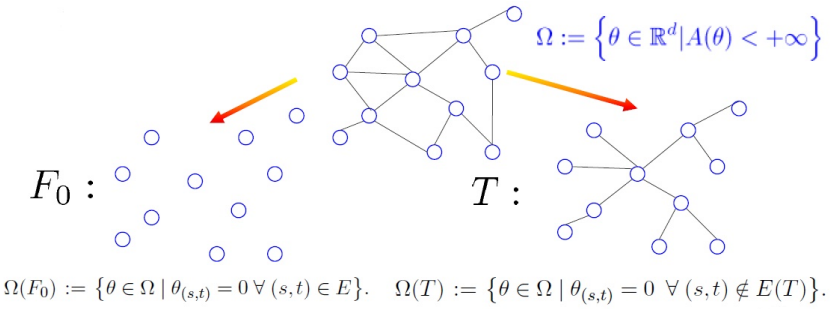


Figure 3: Transform a general graph to tractable subgraph

2.4.2 Geometry of Mean Field

Mean field optimization is always **non-convex** for any exponential family in which the state space $\mathcal{X}^m$ is finite. Recall the marginal polytope $\mathcal{M}(G)$ is a convex hull,

$$\mathcal{M}(G) = \text{conv}\left\{\phi(e); e \in \mathcal{X}^m\right\} \tag{45}$$

and $\mathcal{M}_F(G)$ contains all the extreme points of this polytope. This implies that the convex hull $\mathcal{M}(G)$ must be non-convex if $\mathcal{M}_F(G)$ is a strict subset of the convex hull.
For example, let's consider a two-node Ising model:

$$\mathcal{M}_F(G) = \{\tau_1, \tau_2 \in [0,1] s.t. \tau_{12} = \tau_1\tau_2\} \tag{46}$$

This model has a parabolic cross section along $\tau_1 = \tau_2$, hence it is non-convex.

10-708: Probabilistic Graphical Models, Spring 2020

1: Monte Carlo Methods

Lecturer: Eric P. Xing *Scribe: Aditi Chaudhary, Neha Cheemalavagu, Ke Ni*

1 Approach to Inference

In the previous class, we discussed methods for graph inference. For graphs of manageable size, exact inference algorithms such as elimination algorithm, message passing can be used. However, for complex graphs we turn to approximate inference method such as Mean Field Approximation (MFE) and Loopy Belief Propagation (BP). Both methods pose the inference problem as optimization. The key idea behind MFE is that it is a product of singleton marginals. In Loopy BP, message passing is applied over the any graph, even though it might not be converge to the exact solution. Today we are going to talk about another inference method for arbitrary distribution.

Closed-form representation For a given closed-form distribution say Gaussian, inference is straightforward as we can simply integrate the probability distribution.

Sample-based representation For an arbitrary distribution, we can draw samples $X_n \sim P((X)$ where $n = 1, N$ and use the following approximation $E_p f(x) = \sum_i \frac{1}{N} f(x_i)$ to get the sample-based representation. The key idea is that instead of manipulating the $P(X)$, we instead base it solely on the samples from the distribution.

Naive Sampling We first consider the naive sampling, where we sample according to the probabilities specified by the Bayesian Network. We traverse the graph in the topological order and draw samples for each random variable based on its local marginal distribution. Inference for a given query is then done on the counting the samples drawn which satisfy the query. The major drawback is that for large models, it might be difficult to get good estimates for rare events as we might not have enough samples. So, we might not have a good estimate of the approximate if the sample size is too small.

Monte Carlo Methods For distributions which have irregular or arbitrary distribution, it might be difficult to draw samples using the naive method. This is where Monte Carlo methods come in the picture. The key idea is that how to make use of the examples, because not all examples are useful for a given inference query. The main challenges that these methods attempt to answer are:

1. How to draw samples for irregular or arbitrary distributions?
2. How to know which samples are useful?
3. How do we know how much to sample?

Rejection Sampling Suppose the true distribution $\Pi(X)$ is difficult to sample form. Instead, we consider the un-normalized distribution $\Pi'(X)$ to evaluate since that doesn't involve computing the partition function. We instead sample from a simpler distribution $Q(x)$. The procedure goes as follows:

1. Draw x_0 from $Q(x)$
2. Accept x_0 with probability $p = \frac{\Pi'(x_0}{kQx}$ such that $0 <= p <= 1$

Figure 1 explains this as selecting a sample which falls under the white region of the un-normalized distribution and rejecting the sample if it falls under the grey region.

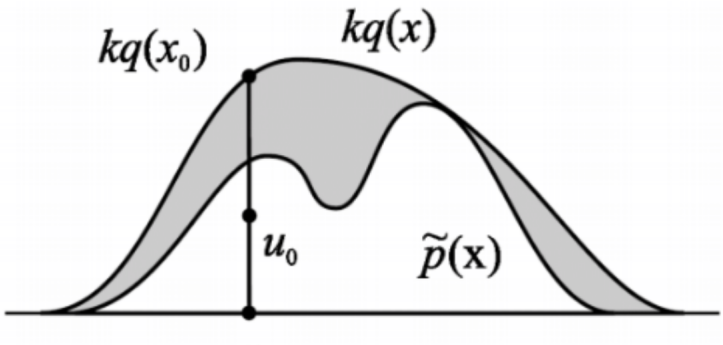


Figure 1: Figure of Rejection Sampling explained pictorically.

We check the correctness of the approach. Based on the algorithm we have: $P(x_0) = \frac{\frac{Q(x_0)*\Pi'(x_0)}{kQ(x_0)}}{\int Q(x)*\Pi'(x)kQ(x)dx}$. The $Q(x)$ terms being common across numerator and denominator get cancelled and give us: $P(x_0) = \frac{\Pi'(x_0)}{\int \Pi'(x)dx} = \Pi(x)$

One major pitfall with this method is that if the $Q(x)$ distribution is not chosen well, we end up with lot of rejected samples, as many as the shaded area in the Figure 1. In the class, we discussed a thought example, where we have $Q \sim N(\mu, \sigma_p^{\frac{2}{d}})$ and $P \sim N(\mu, \sigma_p^{\frac{2}{d}})$ where d is the dimention. However, even for a very similar distribution, in high-dimentional space the optimal acceptance rate is roughly only $\frac{1}{20000}$. We can do adaptive rejection samping which covers the envelope with piecewise functions.

Importance Sampling Suppose sampling from the target distribution $P(x)$ is hard, we therefore sample from a simpler distribution $Q(x)$. We first look at the un-normalized case:

Unnormalized importance sampling

1. Suppose we draw M samples $x_m \sim Q(x)$.
2. We then calculate the respective weights $w_m = \frac{P(x_m)}{Q(x_m)}$ to re-weight the $f(X)$.
3. We then compute $< f(X) >= \frac{1}{M} \sum_m f(x_m) w_m$

Since we re-weight every example, there are no samples wasted. This is called un-normalized because w_m are just weights and need not necessarily sum to 1. We check the correctness of the expectation of the target distribution. $< f(X) >= \int f(x)P(x)dx$
$< f(X) >= \int f(x)\frac{P(x)}{Q(x)}Q(x)dx$
We approximate the integral using the samples we draw from $Q(x)$ which gives us: $< f(X) >\sim \frac{1}{M}\sum_m f(x_m)\frac{P(x_m)}{Q(x_m)}$ where $x_m \sim Q(x)$
$< f(X) >\sim \frac{1}{M}\sum_m f(x_m)w_m$

Normalized importance sampling To get a better estimate of the target distribution, we now consider $P(x) = \frac{P'(x)}{\alpha}$. It turns out we can also obtain α from the samples. Let $R(x) = \frac{P'(X)}{Q(X)}$. We note thet this $R(x)$ is different from the w_m used earlier. w_m was the true ratio of distributions whereas R is the ratio of un-normazlied distribution and the proposed distribution. We show the correctness of that: $< R(x) >_Q= \int \frac{P'(x)}{Q(x)}Q(x)dx$
$< R(x) >_Q= \int P'(x)dx = \alpha$ which is the normazliation constant. Then to calculate the expectation for true distribution: $< f(x) >_P= \int f(x)P(x)dx = \frac{1}{\alpha}\int f(x)\frac{P'(x)}{Q(x)}Q(x)dx$
$< f(x)_P >= \frac{\int f(x)r(x)Q(x)dx}{\int r(x)Q(x)dx}$
$< f(x)_P >\sim \frac{\sum_m f(x_m)r_m}{\sum_m r_m}$ where $x_m \sim Q(x)$
$< f(x)_P >\sim \sum_m f(x_m)w_m$ where $w_m = \frac{r_m}{\sum_m r_m}$

The procedure for sampling is same as before, with the weights now computed as shown above. In the un-normalized sampling, since we don't have to compute the normalization constant, the w_m are available immediately. However, in this case the weights can be computed only after enough samples have been drawn using which the normalization constant can be computed.

The major pitfall for importance sampling is again if the $P$and Q distributions do not match. Consider the example shown in Figure 2: If more samples from the high-density region of $P(x)$ but which falls under the

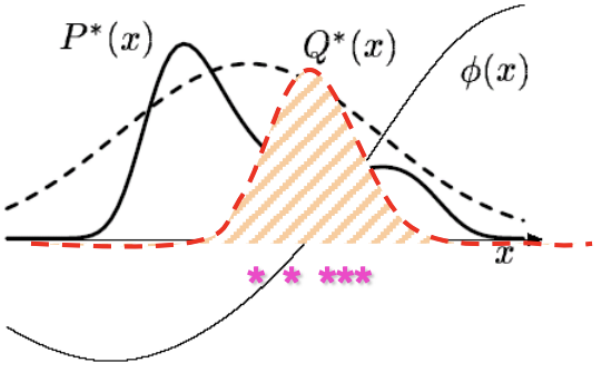


Figure 2: Figure of importance Sampling explained pictorically.

tail of $Q(x)$, we will end up getting less of these high-weight samples as they are under the tail of $Q(x)$. Similarly, samples from the high-density $Q(x)$ if they fall under tail of $P(x)$, we will end with many low-weight samples which will be less important. And since the convergence criterion is based on the change in the $< f(x) >$ value, we might get stuck in the local estimate.

Weighted Re-Sampling There are two possible solutions. The first is to use heavy tail $Q(x)$, but it might lead to lot of samples wasted. The other idea is to use weighted re-sampling. The idea is that we stop using previous examples, but we do another round of re-sampling. The procedure is as follows:

1. Draw samples from the Q
2. Construct weights $w_m = \frac{\frac{P(x_m)}{Q(x_m)}}{\sum_l \frac{P(x_l)}{Q(x_l)}}$
3. Sub-sample x from with probability as w_m .

This re-weights the samples and amplifies the high-importance samples. The low-probability will get suppressed.

Limitations of Monte Carlo Methods There are some fundamental issues with these methods. The weighted re-sampling method lets us re-shape the proposed distribution $Q(x)$. But since it is not an online algorithm, it requires us to store large number of samples and again re-draw samples from those. If the samples are not good, then the procedure needs to repeated again, so it is not very handy in practice. Online algorithms' advantage is that there is no storage cost but the main problem is that the $Q(x)$ function might not match the $P(x)$.

2 Markov Chain Monte Carlo (MCMC)

- MCMC algorithms have adaptive proposals
- Instead of $Q(x')$ they use $Q(x'|x)$ in which x' is the new state being sampled and x is the previous sample
- As, x changes, $Q(x'|x)$ can also change as a function of x'

2.1 Metropolis-Hastings Algorithm

1. Initialize starting state $x^{(0)}$ and set $t = 0$
2. Burn-in: while samples have not converged, draw samples...
   - $x = x^{(t)}$
   - $t = t + 1$
   - sample $x^* \sim Q(x^*|x)$ # draw from proposal
   - sample $u \sim Uniform(0, 1)$ # draw acceptance threshold
     - if $u < A(x^*|x) = min\left(1, \frac{P(x^*)Q(x|x^*)}{P(x)Q(x^*|x)}\right)$, then $x^t = x^*$ # transition
     - else $x^t = x$ # stay in current state
   - Take samples from $P(x)$: reset $t = 0$, for $\mathbf{t} = 1 : N$
     - $x(t + 1) \leftarrow$ Draw sample $x(t)$

2.2 Markov Chains Overview

- **Markov Chain:** a sequence of random variables $x^{(1)}, x^{(2)}, ..., x^{(n)}$ with the Markov Property
- **Markov Property:** $P(x^{(n)} = x|x^{(1)}, ..., x^{(n-1)}) = P(x^{(n)} = x|x^{(n-1)})$
- **Transition kernel:** $P(x^{(n)} = x|x^{(n-1)})$
  - The next state depends only on the previous state
- Random variables $x^{(i)}$ can be vectors
  - $x^{(t)}$ is defined as the t-th sample of **all** variables in a graphical model
  - $X^{(t)}$ represents the entire state of the graphical model at time, t
- We study homogeneous Markov Chains, in which the transition kernel is fixed with time
  - We will call the kernel $T(x'|x)$, where x is the previous state and x' is the next state $P(x^{(t)} = x|x^{(t-1)})$

2.3 Markov Chains Key Concepts

- **Probability distributions over states:** $\pi^{(t)}(x)$ is a distribution over the state of the system x, at time t
  - When considering MCs, we do not think of the system as being in one state, but rather as having a distribution over states
  - For graphical models, x represents **all** variables
- **Transitions:** states transition from $x^{(t)}$ to $x^{(t+1)}$ according to transition kernel $T(x'|x)$
  - We can also transition entire distributions:

$$\pi^{(t+1)}(x') = \sum_x \pi^{(t)}(x)T(x'|x)$$

  - At time t, state x has a probability mass of $\pi^{(t)}(x)$ and the transition probability redistributes this mass to other states x'
- **Stationary distributions:** $\pi(x)$ is a stationary distribution if it does not change under the transition kernel:

$$\pi(x') = \sum_x \pi(x)T(x'|x), \text{ for all x'}$$

- To understand why stationary distributions are important in MCMC, we need to define the following notions:
  - **Irreducible:** an MC is irreducible if you can get form any state x to any other state x' with a probability > 0 in a finite number of steps
    - No states are unreachable in the defined state space
  - **Aperiodic:** an MC is aperiodic if you can return to any state x at any time
    - Periodic MCs have states that need ≥ 2 time steps to return to (cycles)
  - **Ergodic (or regular):** an MC is ergodic if it is irreducible and aperiodic
    - Ergodicity is important and implies that you can reach the stationary distribution $\pi_{st}(x)$ no matter what the initial distribution $\pi^{(0)}(x)$

  - All good MCMC algorithms must satisfy ergodicity so that you cannot have an initialization that will never converge
- **Reversible (detailed balance):** an MC is reversible if there exists a distribution $\pi(x)$ such that the detailed balance condition is satisfied:

$$\pi(x')T(x|x') = \pi(x)T(x'|x)$$

  - The probability of $x' \to x$ is the same as $x \to x'$
  - Reversible MCs **always** have a stationary distribution
  - Proof:

$$\begin{aligned}
\pi(x')T(x|x') &= \pi(x)T(x'|x) \\
\sum_x \pi(x')T(x|x') &= \sum_x \pi(x)T(x'|x) \\
\pi(x')\sum_x T(x|x') &= \sum_x \pi(x)T(x'|x) \\
\pi(x') &= \sum_x \pi(x)T(x'|x)
\end{aligned}$$

  - The last line is the definition of the stationary proof

2.4 Why MH Works

- In MH, we draw samples x' according to $Q(x'|x)$ and then accept or reject the sample according to $A(x'|x)$
- So, the transition kernel is:

$$T(x'|x) = Q(x'|x)A(x'|x)$$

- Proof that MH satisfies detailed balance. Recall that...

$$A(x'|x) = min\left(1, \frac{P(x')Q(x|x')}{P(x)Q(x'|x)}\right)$$

This implies that if $A(x'|x) < 1$, then $\frac{P(x)Q(x'|x)}{P(x')Q(x|x')} > 1$ and thus $A(x|x') = 1$
if $A(x'|x) < 1$, then $\frac{\pi(x)Q(x'|x)}{\pi(x')Q(x|x')} > 1$ and thus $A(x|x') = 1$

  - Suppose $A(x'|x) < 1$ and $A(x|x') = 1$ Then,

$$\begin{aligned}
A(x'|x) &= \frac{P(x')Q(x|x')}{P(x)Q(x'|x)} \\
P(x)Q(x'|x)A(x'|x) &= P(x')Q(x|x') \\
P(x)Q(x'|x)A(x'|x) &= P(x')Q(x|x')A(x|x') \\
P(x)T(x'|x) &= P(x')T(x|x')
\end{aligned}$$

  - The last line is the definition of detailed balance

- This means that the MH algorithm leads to the stationary distribution $P(x)$
- Since we defined $P(x)$ to be the true distribution of x, this suggests that the algorithm eventually converges to the true distribution

2.5 Caveats

- We know MH eventually converges to the true distribution $P(x)$, but we have no guarantees as to when this will occur
  - **Burn-in period:** represents the un-converged part of the Markov Chain, which we throw away
  - Knowing when to halt burn-in is an art and there are techniques to help determine this

2.6 Gibbs Sampling Overview

- **Gibbs Sampling:** an MCMC algorithm that samples each random variable of a graphical model, one at a time
  - GS is a special case of the MH algorithm
- GS algorithms are relatively simple to derive for many graphical models, such as mixture models and Latent Dirichlet allocation
- They have reasonable computation and memory requirements because they sample a single random variable at a time
- They also can be Rao-Blackwellized (integrate out some random variables) to decrease the sampling variance

2.7 Gibbs Sampling Algorithm

1. Let variables $x_1, ..., x_n$ represent random variables in our graphical model
2. Initialize the starting values of $x_1, ..., x_n$
3. Repeat the following until convergence:
   (a) Choose an ordering of the n variables (can be fixed or random)
   (b) For each variable x_i in the selected order:
      i. Sample x form $P(x_i|x_1, ..., x_{i-1}, x_{i+1}, ..., x_n)$ (the conditional distribution of x_i given the current values of all other variables)
      ii. Update $x_i \leftarrow x$

As soon as we update the current variable x_i, we **immediately** use the updated value for sampling the subsequent variables x_j

2.8 Recall Markov Blankets

- The conditional $P(x_i|x_1, ..., x_{i-1}, x_{i+1}, ..., x_n)$ in the GS algorithm can be simplified using Markov Blankets
  - Let $MB(x_i)$ be the Markov Blanket of x_i, then,
$$P(x_i|x_1, ..., x_{i-1}, x_{i+1}, ..., x_n) = P(x_i|MB(x_i))$$
- For a Bayesian network, the Markov Blanket of x is the set of parents, children and co-parents
- For a Markov Random Field, the Markov Blanket of x is its immediate neighbors

3 Topic Models: Collapsed Gibbs

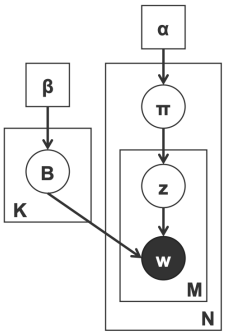


Figure 3: Graph representation of topic model (LDA) from Prof. Xing's slide

3.1 Graph Model Review

LDA graphical model mentioned in previous lectures can utilize collapsed Gibbs to complete inference task. Since previous scribe should have concluded LDA well, we review the model graph briefly here: we have V-dimensional topic vectors for K topics (denoted as B node). Each topic vector is sampled from a conjugate Dirichlet prior distribution, $Dirichlet(\beta)$. Similarly π_i is sampled from $Dirichlet(\alpha_i)$, $i \in \{1, ..., M\}$ where M is the number of documentation. z_{ij} is a topic index sampled from $Multinomial(\pi_i)$, with i, j denoting documentation ID and j^{th} token in a document. Finally, $w_{ij} \sim Multinomial(TopicVector_{z_{ij}})$.

3.2 Algorithm Overview

Similar to Gibbs sampling, we sample all z at every iteration until convergence, for all documents and all tokens. A simplified core algorithm pseudocode for inferring z is below.

1. Initialize the starting values of $z_{1,\cdot}, ..., z_{n,\cdot}$
2. Repeat the following until convergence:
   (a) For each variable z_{ij} in the token order and some documentation order
      i. Sample z from $P(z_{ij}|z_{-ij}, w)$
      ii. Update $z_{ij} \leftarrow z$

3.3 Compute $P(z_{ij}|z_{-ij}, w)$

We can derive this probability density function from two Dirichlet-Multinomial conditional distributions. The result is shown below.

$$P(z_{ij}) \propto \frac{n^{w_i}_{-i,j} + \beta}{n^{\cdot}_{-i,j} + W \cdot \beta} \cdot \frac{n^{d_i}_{-i,j} + \alpha}{n^{d_i}_{-i} + T \cdot \alpha} \tag{1}$$

$n_{-i,j}^{w_i}$ is the number of word positions a excluding i such that $w_a = w_i, z_{aj} = z_{ij}$; $n_{-i,j}^{\cdot}$ is the number of word positions a with $z_{aj} = z_{ij}$ in all documents excluding w_i; $n_{-i,j}^{d_i}$ is the number of word positions a in the current document excluding i such that $z_{aj} = z_{ij}$; $n_{-i}^{d_i}$ is the number of word positions a in the current document, which maybe ignored with α terms.

After we obtain the formula proportional to $P(z_{ij}|z_{-ij}, w)$, we can normalize probabilities for all z_{ij} and obtain the distribution. Note that now we would like to estimate new π and topic word vectors according to our new z sample, if the process does not converge.

4 Proof: Gibbs Sampling is MH

- Gibbs Sampling proposal distribution: $Q(x_i', x_{-i}|x_i, x_{-i}) = P(x_i'|x_{-i})$
- apply MH to this proposal

$$\begin{aligned} A(x_i', x_{-i}|x_i, x_{-i}) &= min(1, \frac{P(x_i', x_{-i}) \cdot Q(x_i, x_{-i}|x_i', x_{-i})}{P(x_i, x_{-i}) \cdot Q(x_i', x_{-i}|x_i, x_{-i})}) \\ &= min(1, \frac{P(x_i', x_{-i}) \cdot P(x_i|x_{-i})}{P(x_i, x_{-i}) \cdot P(x_i'|x_{-i})}) \\ &= min(1, \frac{P(x_i'|x_{-i}) \cdot P(x\_i) \cdot P(x_i|x_{-i})}{P(x_i|x_{-i}) \cdot P(x\_i) \cdot P(x_i'|x_{-i})}) \\ &= min(1, 1) \\ &= 1 \end{aligned} \tag{2}$$

As we can observe, GS is a special case of MH with a proposal accepting everything.

5 Practical Aspects of MCMC

- Evaluate $Q(x'|x)$
  - acceptance rate
  - autocorrelation function
- When to stop burn-in
  - plot the sample values vs. time
  - plot log-likelihood vs. time

5.1 Acceptance Rate

- Definition of acceptance Rate: the fraction of samples that MH accepts.
  - General guideline: 0.5 is a good acceptance rate.
  - If $P(x)$ and $Q(x'|x)$ are both Gaussian, the optimal acceptance rate is around 0.45 for $D = 1$ and around 0.23 as D goes to infinity.

- Tradeoff regarding choosing $Q(x'|x)$
  - Low-variance (narrow) proposals have high acceptance rate and proposed samples are close leading to more iterations to explore P(x) fully.
  - High-variance proposals explore P(x) faster, but rejects many samples which slows the algorithm.

5.2 Autocorrelation(AC) Function

AC indicates how correlated adjacent samples are, which means we want lower AC. AC function of a random variable x is:

$$R_x(k) = \frac{\sum_{t=1}^{n-k}(x_t - \hat{x}) \cdot (x_{t+k} - \hat{x})}{\sum_{t=1}^{n-k}(x_t - \hat{x})^2} \tag{3}$$

Sample Size Inflation Factor (SSIF) can be estimated by $R_x(1)$

$$S_x = \frac{1 + R_x(1)}{1 - R_x(1)} \tag{4}$$

The effective sample size is n/S_x, where n is the number of samples.

5.3 Monitor burn-in: Sample vs. Time

We can monitor convergence by plotting all samples from multiple MH runs. Well-mixed chains in the sample-time plots indicate good convergence. However, using this method needs us to visualize random variables, which may be high dimensional data.

5.4 Monitor burn-in: Likelihood vs. Time

To solve high-dimensional data issue mentioned above, we can plot $\log P(x_i|x_{-i}$, meaning we plot complete log-likelihood of a random variable x depending all other random variables in the model.

6 Summary

- **Direct Sampling:** is difficult to populate in high dimensional space
- **Rejection Sampling:** Creates sampling like in direct sampling, but only counts samples which are consistent with evidences
- **Likelihood Weighting:** Sample variables and calculate evidence weight, but only create samples which support evidences
- **MCMC:** use adaptive proposals $Q(x'|x)$ to sample from the true distribution $P(x)$
  - **Metropolis-Hastings:** allows you to specify $Q(x'|x)$, but choosing a good one is important
  - **Gibbs:** set $Q(x'|x)$ to the conditional distribution $P(x'|x)$; the acceptance rate is always 1, but that results in slow exploration of the space
  - Knowing when to halt burn in is an art

10-708: Probabilistic Graphical Models, Spring 2020

10: Sampling 2

Lecturer: Eric P. Xing Scribe: Joshua Uyheng, Geyang Zhang, Mike Chen, Wenyu Huang, Niles Christensen

1 Challenges with Markov Chain Monte Carlo

There are several challenges to using Markov Chain Monte Carlo (MCMC). First of all, performing random walks in MCMC may have poor acceptance rates, which means that we need to increase the total number of samples drawn. Moreover, the effective number of samples is small because samples can have high correlation between each other given the method that is used to draw samples in MCMC. Finally, it can be unclear from first principles when to stop burn-in.

1.1 Poor Acceptance Rates

There are tradeoffs in choosing $Q(x'|x)$ that have to do with whether the distribution has low or high variance. Proposals have to be selected to balance sufficient variance (ability to explore distribution $P(x)$) with high enough acceptance rates for speeding up sampler.

- **Low-variance** (narrow) proposals might have **high acceptance rates** but do not explore $P(x)$ efficiently.
- **High-variance** (wide) proposals explore $P(x)$ well but have **low acceptance rates**.

1.2 Autocorrelation

Due to the design of the sampling process, MCMC chains will show **autocorrelation**, which means temporally close samples feature high correlation. We want **low autocorrelation** for a better **effective sample size**.

These concerns can be monitored using the autocorrelation function $R_x(k)$ (Equation 1) and the sample size inflation factor s_x (Equation 2). The effective sample size is computed as $\frac{n}{s_x}$, which is increased with low autocorrelation but decreased with high autocorrelation.

$$R_x(k) = \frac{\sum_{t=1}^{n-k} (x_t - \bar{x})(x_{t+k} - \bar{x})}{\sum_{t=1}^{n-k} (x_t - \bar{x})^2} \tag{1}$$

$$s_x = \frac{1 + R_x(1)}{1 - R_x(1)} \tag{2}$$

1.3 Stopping Burn-In

Though there are no hard rules for stopping burn-in, we can make use of meaningful heuristics to decide that sampling chains have converged.

1. **Monitor samples directly**: If chains look well-mixed, it is likely that convergence has occurred. See Figure 1.

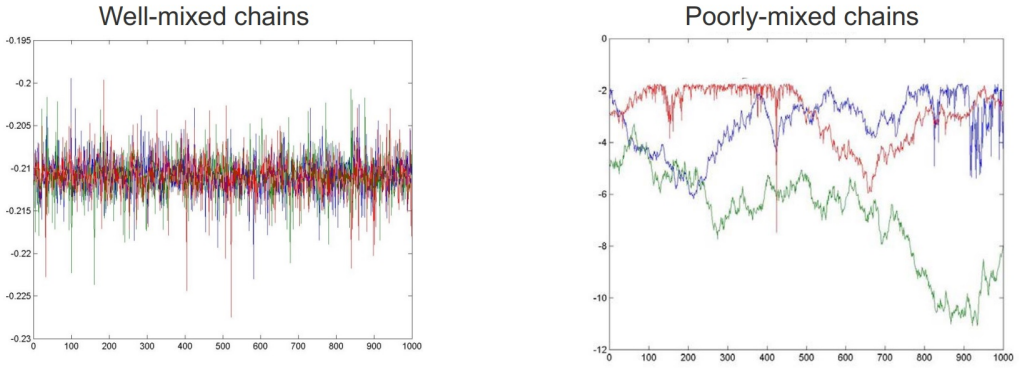


Figure 1: **Left:** Well-mixed chains indicate convergence has occurred. **Right:** Poorly mixed chains indicate that convergence has yet to occur. Figures from lecture slides.

2. **Monitor log-likelihood**: If data are high-dimensional (as is often the case), the complete log-likelihood can be visualized instead. Convergence is often suggested by the log-likelihood first climbing, then eventually plateauing. See Figure 2.

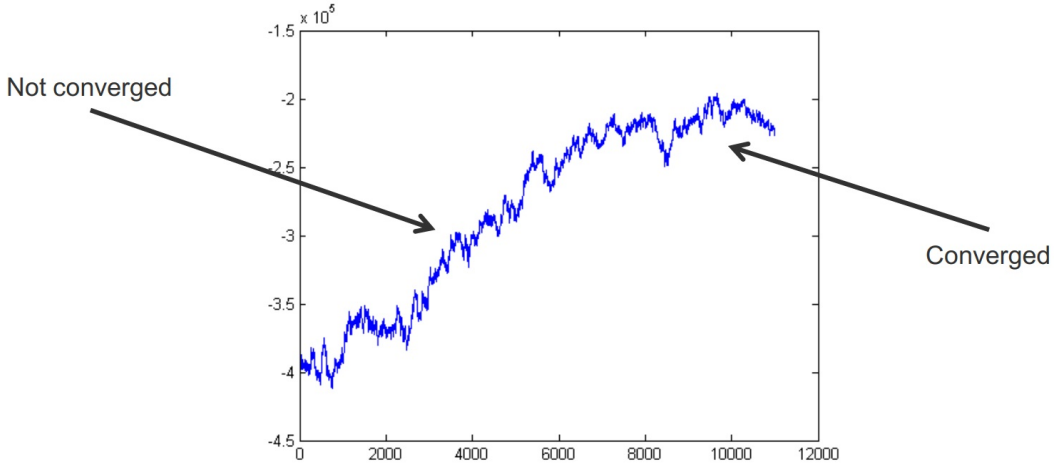


Figure 2: Visualization of log-likelihood often useful for high-dimensional samples. Plateauing behavior suggests convergence. Figure from lecture slides.

2 Hamiltonian Monte Carlo

2.1 Motivation

In Random walk in MCMC, the proposal is at risk of slipping into low probability region and therefore $Q(x_{new}|x_{old})$ may have low acceptance rate. Also, when the variance of proposal is small, samples may be highly correlated with each other.

We want to a way to control the direction of the next sample so we can avoid low acceptance rate and keep low correlation. HMC is using gradient information to control the direction.

2.2 Hamiltonians and Operators

Hamiltonians are a way of describing the state of subatomic particles. A Hamiltonian is a function of two vectors: a position vector, x, and a momentum vector, p. We say that

$$H(p, x) = K(p) + U(x)$$

where H is the Hamiltonian, K is a function that yields kinetic energy given momentum, and U is a function that yields potential energy given position.

These notions allow us to explore Hamiltonian dynamics. In general, information about the Hamiltonian at a specific time allows us to reason about future time steps. Specifically, we say that

$$\frac{dx_i}{dt} = \frac{dH}{dp_i}$$

$$\frac{dp_i}{dt} = \frac{dH}{dx_i}$$

3 Computing Updates

3.1 Euler's Method

We can use Euler's method to compute a sequence of samples. Given Euler's equation:

$$p_i(t+\epsilon) = p_i(t) + \epsilon \frac{dp_i}{dt}(t) = p_i(t) - \epsilon \frac{dU}{dq_i}(q(t)) \tag{3}$$

$$q_i(t+\epsilon) = q_i(t) + \epsilon \frac{dq_i}{dt}(t) = q_i(t) + \epsilon \frac{p_i(t)}{m_i} \tag{4}$$

We can use p_i to compute $q_t, q_{t+1}, ...q_N$.

3.1.1 Shortcomings of Euler's Method

Euler's method has one main shortcoming. This is the fact that an update can be expressed as

$$\begin{pmatrix} p(t+\epsilon) \\ q(t+\epsilon) \end{pmatrix} = \begin{pmatrix} 1 & a \\ b & 1 \end{pmatrix} \begin{pmatrix} p(t) \\ q(t) \end{pmatrix} \tag{5}$$

It can be seen that, depending on the values of a and b, the volume of the space described by $\begin{pmatrix} p(t) \\ q(t) \end{pmatrix}$ can potentially change with repeated updates. As such, Euler's method describes a divergent series. This is undesirable, as the points we sample from it can start to get arbitrarily far from our original series. Furthermore, this is not possible in the original physics interpretation of a Hamiltonian, which can interfere with intuitive understanding of the system.

3.2 Leapfrog Method

In the leapfrog method, we compute the next step slightly differently from before. Now, we compute the following

$$\begin{aligned} p_i(t+\epsilon/2) &= p_i(t) - (\epsilon/2)\frac{dU}{dq_i}(q(t)) \\ q_i(t+\epsilon) &= q_i(t) + \epsilon\frac{p_i(t+\epsilon/2)}{m_i} \\ p_i(t+\epsilon) &= p_i(t+\epsilon/2) - (\epsilon/2)\frac{dU}{dq_i}(q(t+\epsilon)) \end{aligned}$$

3.2.1 Compared to Euler's Method

The leapfrog method solves Euler's Method's problem of being a divergent series. When applying a similar process to the one used to generate equation 5, we instead get the following:

$$\begin{pmatrix} p(t+\epsilon) \\ q(t+\epsilon) \end{pmatrix} = \begin{pmatrix} 1 & a \\ 0 & 1 \end{pmatrix} \begin{pmatrix} p(t+\epsilon/2) \\ q(t+\epsilon) \end{pmatrix} \tag{6}$$

Notably, the matrix $\begin{pmatrix} 1 & a \\ 0 & 1 \end{pmatrix}$ corresponds to a shear transformation, meaning that it is volume-preserving. This transformation can also seen to be volume-preserving because its determinant is 1. As such, this series is not divergent, and is better behaved.

3.3 Langevin MCMC

Main idea is to do one step Leapfrog. Compared with Leapfrog method, Langevin MCMC algorithm can be aggragated in a more compact form.

3.3.1 Pros against Leapfrog method

Given a large datasets, stocahstic Langevin has control over computing the gradient using subsets of samples instead of whole gradient.

4 More Ideas on Approximate Inference

Previously, we have seen a lot of methods on computing $Q(\cdot)$, the proposed target distribution. For example, it can be defined as a Gaussian, or a moving Gaussian depends on your previous sample. To make it fancier, we can draw a sample from the Gaussian and then compute the gradient to find our next sample.

To compute the building block of a good proposal $Q(\cdot)$, we have to investigate the original target distribution $P(\cdot)$. In MC, it is used to calculate the weight r, but not used to compute $Q(\cdot)$; in MCMC, it is used to compute A; in HMC, we use it to compute the gradient (write as $\frac{\partial U}{\partial q}$ in equation, where U is original distribution and q is our target x).

Now, we try to make a better proposal of $Q(x_{new}|x_{old})$ directly from the knowledge of P.

4.1 Variational MCMC

Recall Jensen's inequality:

$$log(p(x|\theta)) \geq E_{q(z)}[log(p(x|\theta))] - E_{q(z)}[log(q(z)]$$

Where we are interested in posterior of parameters $p(\theta|x)$ but transform to find the expectation of hidden variable $q(z|\lambda)$, where λ is the variational parameter, and use it as a lower bound to approach P. However, the calculation of Q could still be hard. So we ask: if we can introduce another variational parameter ξ to replace $p(x|\theta)$ as $p(x|\theta, \xi)$, so to make the computing of Q easier? The idea here is to find a middle place between $q(z|\lambda)$ and $p(x|\theta, \xi)$ as $P^{est}(\theta|x, \lambda, \xi)$, and let this estimation be as close as to $P(\theta|x)$.

$$Q(x_{new}|x_{old}) = P^{est}(\theta|x, \lambda, \xi) \leq P(\theta|x)$$

In this process, finding $P^{est}(\theta|x, \lambda, \xi)$ can be taken as a optimization problem, and approaching P with P^{est} can be done with MCMC. Generally, this method helps increasing mixing rate, and acceptance rate in lower dimension; it also decrease the convergence time and get uncorrelated information. Also, compare to Hamiltonian MCMC which also combined optimization and variatioanl inference, this method is more general.

4.2 Sequential MC with Particle Filters

Recall weighted resampling:
1. Draw N samples from proposal Q;
2. Get samples weight $w^m = \frac{P(x^m)|Q(x^m)}{\sum_l P(x^l)|Q(x^l)} = \frac{r^m}{\sum_m r^m}$;
3. Sub-sample those N samples w.r.t weights.

In this way, you can have the samples' distribution as close as to the original distribution.

Similarly, this idea can be adapted by sequential sampling. Suppose we have a HMM but with very complex transition probability, if we have $P(X_t|\mathbf{Y}_{1:t})$, how can we show $P(X_t + 1|\mathbf{Y}_{1:t+1})$? Observe the expression of $P(X_t|\mathbf{Y}_{1:t})$:

$$P(X_t|\mathbf{Y}_{1:t}) = P(X_t|\mathbf{Y}_{1:t-1}, Y_t) = \frac{P(X_t|\mathbf{Y}_{1:t-1})P(Y_t|X_t)}{\int P(X_t|\mathbf{Y}_{1:t-1})P(Y_t|X_t)dX_t}$$

where $P(X_t|\mathbf{Y}_{1:t-1})$ is the sampling step and $\frac{P(Y_t|X_t)}{\int P(X_t|\mathbf{Y}_{1:t-1})P(Y_t|X_t)dX_t}$ is the weight of current sampling. So $P(X_t|\mathbf{Y}_{1:t})$ can be sampled by:

$$X_t^m \sim P(X_t|\mathbf{Y}_{1:t-1}), w_t^m = \frac{P(Y_t|X_t^m)}{\sum_m P(Y_t|X_t^m)}$$

Thus, we can iteratively do:

$$P(X_{t+1}|\mathbf{Y}_{1:t}) = \int P(X_t|\mathbf{Y}_{1:t})P(X_{t+1}|X_t)dX_t = \sum_m w_t^m P(X_{t+1}|X_t^m)$$

$$P(X_t + 1|\mathbf{Y}_{1:t+1}) = \frac{P(X_{t+1}|\mathbf{Y}_{1:t})P(Y_{t+1}|X_{t+1})}{\int P(X_{t+1}|\mathbf{Y}_{1:t})P(Y_{t+1}|X_{t+1})dX_{t+1}}$$

$$\Rightarrow X_{t+1}^m \sim P(X_{t+1}|\mathbf{Y}_{1:t}), w_{t+1}^m = \frac{P(Y_{t+1}|X_{t+1}^m)}{\sum_m P(Y_{t+1}|X_{t+1}^m)}$$

e.g switching SSM
Each hidden state $S_t = \{X_t^1, X_t^2\}$. Knowing $S_t^m \sim P(S_t|S_{t-1}^m)$, approximate $P(X_t|Y_{1:t}, S_{1:t}^m)$.

10-708: Probabilistic Graphical Models, Spring 2020

11: Foundations of Deep Learning

Lecturer: Eric P. Xing *Scribe: Michael Kronovet, Shreyas Chaudhari, Ahmed Shah*

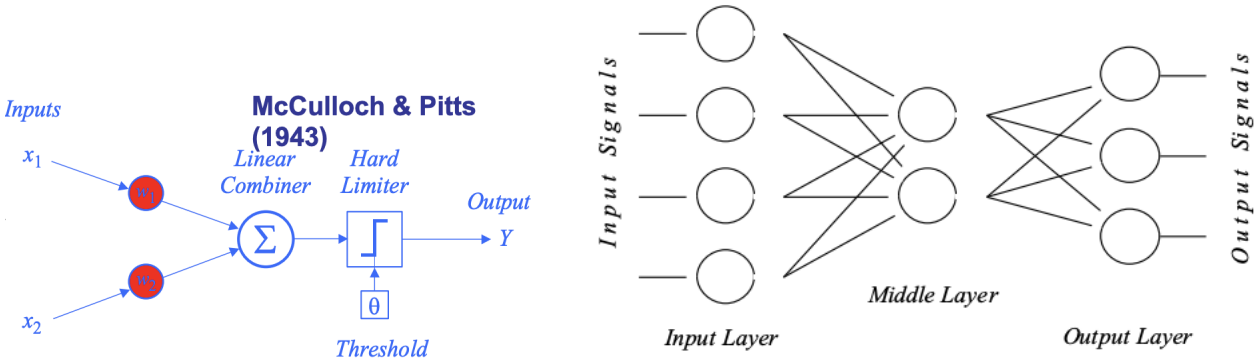
1 An overview of DL components

Many people used to view machine learning as more of a black box. You throw data in a sort of "learning machine", where you tweak some tuning parameters, and get an output. Now we are much more familiar with the underlying math of machine learning models and the underlying mechanisms of how they work.

1.1 Historical Remarks: Early Days of Neural Networks

In the human brain we have roughly 10 billion neurons. Important components of the neuron include dendrites, axons, and synapses. Dendrites are extensions from the neuron cell body that take information to the cell body. They receive stimulation in order for the cell to become active. Axons are the extension from the neuron cell body that takes information away from the cell body. The axon conducts electrical impulses known as action potentials away from the nerve cell body. The function of the axon is to transmit information. Synapses are junctions between nerve cells, and they control the aggregation of signals.

Neural networks were originally an attempt to replicate neurons. In 1943, McCulloch and Pitts published a paper that detailed an artificial neuron (really a perceptron) that worked by mimicking the functionality of a biological neuron. In their model, the output is a hard threshold of a linear combination of inputs. In order to replicate an entire biological neural network, people adapted the McCulloch-Pitts model to have multiple linear combinations of inputs.



1.1.1 Perception

$$\hat{f}(x_i; \vec{w}) = \frac{1}{1 + \exp(-\sum_{i=0}^{n} w_i x_i)}$$

We can imagine that every dendrite will have different conductivity properties, hence the different weights for each input. We seek the weights that minimize the squared residual loss. So, $\vec{w} = \arg\min_{\vec{w}} \sum_i \frac{1}{2}(y_i - \hat{f}(x_i; \vec{w}))^2$

$$\begin{aligned}
\frac{\partial E_D[\vec{w}]}{\partial w_j} &= \frac{\partial}{\partial w_i}\frac{1}{2}\sum_l (t_d - o_d)^2 \\
&= \frac{1}{2}\sum_l 2(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - o_d) \\
&= \sum_l (t_d - o_d)(-\frac{\partial o_d}{\partial w_i}) \\
&= -\sum_l (t_d - o_d)\frac{\partial o_d}{\partial net_i}\frac{\partial net_d}{\partial w_i} \\
&= -\sum_l (t_d - o_d) o_d(1 - o_d) x_d^i
\end{aligned}$$

where x_d = input, t_d = target output, w_i = weight i, net = $\sum_i w_i x_i$, o_d = observed output = $\frac{1}{1+\exp(\text{- net})}$.

The perceptron learning algorithm is essentially gradient descent. We can either batch update the weights (compute gradient over the entire dataset) or incrementally update the weights (compute gradient for random points). It is better to use incremental updates when the dataset is large, because then compute the gradient over the entire dataset is computationally expensive.

Batch update rule: $\vec{w} = \vec{w} - \eta \nabla E_D[\vec{w}]$ since it calculated across the entire dataset D.

Incremental update rule: $\vec{w} = \vec{w} - \eta \nabla E_d[\vec{w}]$ since it is calculated stochastically for different datapoints $d \in D$.

1.2 Reverse-Mode Automatic Differentiation (Backpropagation)

Similar to the perceptron learning rule, we use gradient descent to solve for the optimal weights in a neural network. Gradient descent in a neural network relies on backpropagation, which is essentially just applying the chain rule from calculus and using reverse accumulation.

We can apply the chain rule and sum over all states of the intermediate variable indefinitely until we reach the value in question we wish to take the derivative with respect to (let's say its x in this case.

$$\frac{\partial f_n}{\partial x} = \sum_{i_1 \in \pi(n)} \frac{\partial f_n}{\partial f_{i_1}}\frac{\partial f_{i_1}}{\partial x} = \sum_{i_1 \in \pi(n)} \frac{\partial f_n}{\partial f_{i_1}} \sum_{i_2 \in \pi(i_1)} \frac{\partial f_{i_1}}{\partial f_{i_2}}\frac{\partial f_{i_2}}{\partial x} = \ldots$$

This process is fully automated in modern programs like tensorflow.

1.3 Modern Building Blocks: Units, Layers, Activation Functions, Loss Functions, Etc.

1.3.1 Activation Functions

Activation functions define the output of a node in a neural network given a linear combination of the previous layer's outputs. Common activation functions are the linear activation $f(x) = x$, the ReLu activation $f(x) = \max\{0, x\}$, the sigmoid activation $f(x) = \frac{1}{1+\exp(-x)}$, the tanh activation $\frac{e^x - e^{-x}}{e^x + e^{-x}}$, etc.

1.3.2 Structure of Networks

The structure of a neural network defines the underlying computation of what that network is doing. There are many different network structures that have been adapted and specialized for specific tasks. Convolutional neural networks (CNNs) have been used for image processing, recurrent neural networks (RNNs) have often been used to work with time series data, and there are many other formulations of neural networks that are optimized for certain use cases.

1.3.3 Loss Functions

Loss functions are imperative for the training of neural networks because they tell your weights how to update during backpropagation. Different loss functions are preferable in different problem settings. For example, if we were interested in training a neural network to predict a class label for a set of inputs, then cross-entropy could be a good loss function for that network. When modeling a classification problem where we are interested in mapping input variables to a class label, we can model the problem as predicting the probability of an example belonging to each class, which is why cross entropy, which measures the error between two probability distributions, could be a good choice. On the other hand, using a loss function like mean squared error that would take the squared difference between output classes may lead the network to perform more poorly. Suppose you choose a loss function for your neural network that optimizes for the squared difference between your predicted label and the actual label. Then, if you incorrectly classify something as belonging to class 5 instead of class 1, that would have a much more impactful error on your model than if you were to predict class 2 instead of class 1. Depending on how you would want to weight these errors, you would want to tweak your loss function.

Neural networks can use any arbitrary combinations of activation functions, layer structures, and loss functions. Interestingly, it seems that given enough data deeper network structures keep improving their predictive abilities. We believe this is the case because deeper networks with more training allow the networks to learn increasingly more abstract representations of the data. For example, when a neural network is trained on identifying faces, the weights on the deeper network layers when observed start to take on facial features. In fact, the deeper into the network you go the more complex and detailed the faces and patterns in the weights become. Since neural networks are capable of "disentangling" the data to learn complex representations of features from the data, they are often referred to as performing representation learning.

2 Similarities and differences between GMs and NNs

Both graphical models and GMs have similarities in their empirical goals, structure (graphical) and objective (aggregated from local functions). But the similarities end there and seemingly similar tasks of inference and learnign have different meanings under the two lenses. These differences have been tabulated below:

	Graphical Models	Deep Neural Networks
Representation	Encode meaningful knowledge and the associated uncertainty in a graphical form	Representations facilitate computation and performance on the end-metric and intermediate representations are not guaranteed to be meaningful
Learning and Inference	Based on a rich toolbox of well-studied and structure-dependent techniques like EM, message passing, VI, MCMC, etc.	Learning is predominantly based on the gradient descent method (backpropagation); Inference is often trivial and done via a "forward pass"
Structure	Graphs represent models	Graphs represent computation

As the above table summarizes, though both GMs and NNs have a graphical structure the graphs have different utilities. Utility of the graph for **graphical models**:

- Synthesizing a global loss function from local structure
- Designing sound and efficient inference algorithms
- Inspire approximation and penalization
- Monitoring theoretical and empirical behavior and accuracy of inference

Utility of the graph for **deep neural networks**:

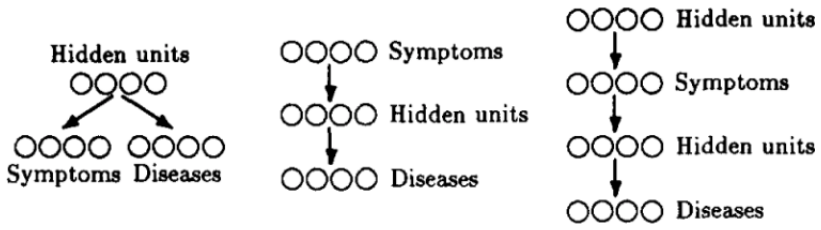
- Conceptually synthesize complex decision hypothesis
- Organizing computational operations
- Designing processing steps and computing modules
- Inference algorithms in DL have no obvious utility

This comparison can be summed up by these two statements - Graphical models are representations of probability distributions. Neural networks are function approximators (with no necessary probabilistic meaning).

The following section considers some of the neural nets that are in fact proper graphical models.

3 Neural Networks That Are Graphical Models

3.1 Sigmoid Belief Networks



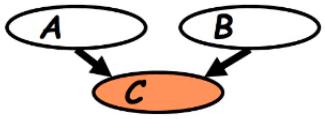
Sigmoid belief networks have been around for around 30 years and in the simplest form, they have one hidden layer with conditional probabilities represented by sigmoid functions. This does not allow for complex combination of features/nodes. Adding a or multiple hidden layers can allow for such combinations (the figure).

Sigmoid belief nets are simply Bayesian networks over binary variables with conditional probabilities represented by sigmoid functions:

$$P\left(x_i|\pi\left(x_i\right)\right) = \sigma\left(x_i \sum_{x_j \in \pi(x_i)} w_{ij} x_j\right)$$

3.1.1 Explaining away effect

Bayesian networks exhibit a phenomenon called "explain away effect". This is a consequence of conditional dependence in directed graphical models for V-structures. When conditioned on visible variables, the hidden variables are dependent on each other.



This coupling or correlation creates a computational difficulty for sigmoid belief networks.

3.1.2 Learning and Inference

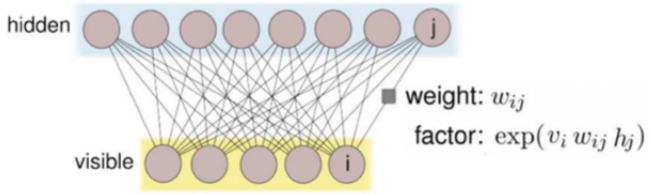
The learning in sigmoid belief networks has the update rule given in the equations that follow (proposed by Neal, 1992).

$$\begin{aligned}
\frac{\partial L}{\partial w_{ij}} &= \sum_{\tilde{v}\in\mathcal{T}} \frac{1}{P(\tilde{V}=\tilde{v})} \frac{\partial P(\tilde{V}=\tilde{v})}{\partial w_{ij}} \\
&= \sum_{\tilde{v}\in\mathcal{T}} \frac{1}{P(\tilde{V}=\tilde{v})} \sum_{\tilde{h}} \frac{\partial P(\tilde{S}=\langle\tilde{h},\tilde{v}\rangle)}{\partial w_{ij}} \\
&= \sum_{\tilde{v}\in\mathcal{T}} \sum_{\tilde{h}} P(\tilde{S}=\langle\tilde{h},\tilde{v}\rangle|\tilde{V}=\tilde{v}) \cdot \frac{1}{P(\tilde{S}=(\tilde{h},\tilde{v}))} \frac{\partial P(\tilde{S}=\langle\tilde{h},\tilde{v}\rangle)}{\partial w_{ij}} \\
&= \sum_{\tilde{v}\in\mathcal{T}} \sum_{\tilde{s}} P(\tilde{S}=\tilde{s}|\tilde{V}=\tilde{v}) \frac{1}{P(\tilde{S}=\tilde{s})} \frac{\partial P(\tilde{S}=\tilde{s})}{\partial w_{ij}} \\
&= \sum_{\tilde{v}\in\mathcal{T}} \sum_{\tilde{s}} P(\tilde{S}=\tilde{s}|\tilde{V}=\tilde{v}) \frac{1}{\sigma\left(s_i^* \sum_{k<i} s_k w_{ik}\right)} \frac{\partial \sigma\left(s_i^* \sum_{k<i} s_k w_{ik}\right)}{\partial w_{ij}} \\
&= \sum_{\tilde{v}\in\mathcal{T}} \sum_{\tilde{s}} P(\tilde{S}=\tilde{s}|\tilde{V}=\tilde{v}) s_i^* s_j \sigma\left(-s_i^* \sum_{k<i} s_k w_{ik}\right)
\end{aligned}$$

where the last expression is approximated with Gibbs sampling. Due to the *explaining away effect*, this sampling will be hard. This leads to slow convergence.

3.2 Restricted Boltzmann Machines

RBM is a Markov Random Field represented with a bi-partite graph. All nodes in one layer of the graph are connected to all nodes in the other. There are no intra-layer connections, which induced the conditional independence properties for undirected graphical models.



Usually the visible and hidden variables (v, h) are binary variables, though it can be generalised to Gaussian Bernoulli RBMs. The joint distribution is represented by:

$$P(v,h) = \frac{1}{Z} \exp\left\{ \sum_{i,j} w_{ij} v_i h_i + \sum_i b_i v_i + \sum_j c_j h_j \right\}$$

3.2.1 Learning in RBMs

The log-likelihood of a single data point where the unobservables marginalized out in the last term is:

$$\log L(v) = \log \sum_h \exp\left\{ \sum_{i,j} w_{ij} v_i h_i + \sum_i b_i v_i + \sum_j c_j h_j - \log(Z) \right\}$$

Gradient of the log-likelihood w.r.t. the model parameters, written as expectations is:

$$\frac{\partial}{\partial w_{ij}} \log L(v) = \underbrace{E_{P(h|v)}\left[\frac{\partial}{\partial w_{ij}} P(v,h) \right]}_{\text{over the posterior}} - \underbrace{E_{P(v,h)}\left[\frac{\partial}{\partial w_{ij}} P(v,h) \right]}_{\text{over the joint}}$$

Both expectations are approximated via sampling. This leads to two 'phases':

- Sampling from the posterior is exact (RBM factorizes over h given v): called the clamped / wake / positive phase as the network is "awake" since it conditions on the visible variables
- Sampling from the joint is done via MCMC (e.g., Gibbs sampling): called the unclamped / sleep / free / negative phase as the network is "asleep" since it samples the visible variables from the joint. In some sense it, is "dreaming" the visible inputs

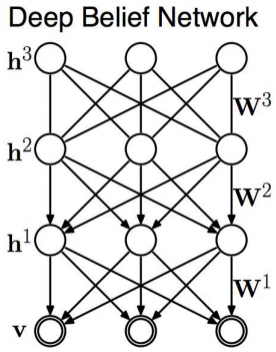
Learning is done by optimizing the log-likelihood of the model for a given data via gradient descent. Estimation in the sleep phase heavily relies on the mixing properties of the Markov chain which causes slow convergence and requires extra computation.
The other way round applies to directed latent variable models, as the following section describes.

3.2.2 RBMs are Infinite Belief Networks

Recall that the gradient of the training objective of RBMs, i.e. the log likelihood of the visible units, contains an expectation over the joint probability, $p(v, h)$. To compute this expectation we sample the joint distribution using block Gibbs sampling, which involves alternatively clamping v and sampling h, and clamping h and sampling v until the samples converge to a stationary distribution. Observe that this process is exactly equivalent to top-down propagation in an *infinitely* deep sigmoid network in which the weights are shared across all the layers. Further observe that a model that stacks a RBM on top of a sigmoid which also shares the same weights is strictly equivalent to the RBM itself. If the weights of the sigmoid network and the RBM are then untied we get a Deep Belief Network, which is explained in detail in the following section.

4 Deep Belief Network

Deep Belief Networks (DBNs) are hybrid graphical models that comprise of a RBM stacked on top of a sigmoid network. Importantly, the weights of the RBM and the sigmoid networks are untied. If they were tied the overall model, as mentioned earlier, would be equivalent to just the RBM.



A DBN represents the following joint probability distribution

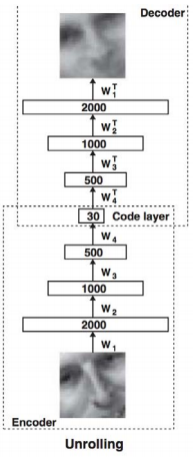
$$P(v, h_1, h_2, ..., h_{k-1}, h_k) = P(v|h_1) \left(\prod_{i=1}^{k-2} P(h_i|h_{i+1}) \right) P(h_{k-1}, h_k)$$

where $P(h_{k-1}, h_k)$ is modeled by an RBM and $P(h_i|h_{i+1})$ are modeled by sigmoid networks. The model is trained by maximizing, $\log P(v)$, the log-likelihood of the given data. Training is done stage-wise, with a greedy layer-wise pretraining step followed by an ad-hoc, possibly task dependent, fine-tuning step.

The layer-wise pretraining proceeds as follows. First, a RBM is trained. The weights of the RBM are then frozen and another RBM is stacked on top of it. The second RBM is trained on the output of the first RBM. Note that the weights of the first and second RBMs are *untied*. Proceeding in this manner we can incrementally add *pretrained* layers to the DBN. Observe that this process is equivalent to untying and tuning the weights of an infinite sigmoid network layer by layer, starting from the bottom and moving up. Common variants of this training process either tune the weights of each RBM until convergence by repeated iterations of the wake and sleep steps, or may perform only one weight update before moving to the next RBM.

Since the pre-training procedure is mostly ad-hoc and does not guarantee convergence to a good probabilistic model, a fine-tuning step is performed to tune the model for some downstream task. The hope is that

the representations learned by the model in pretraining step capture information that can be used in the downstream task. One such task could be to produce a feature representation for images. For this task we can unroll the pretrained DBN to create an *autoencoder* as in the diagram below.



Then we can fine-tune all the weights by backpropagating the reconstruction error between the original image and the reconstructed image. In the diagram above, the code layer provides a representation of the image.

A closely related model to DBNs is the Deep Boltzmann Machine (DBM), which consists of multiple RBMs stacked together. DBMs can be trained like boltzmann machines, using MCMC. Weights from pretrained DBMs can be used to initialize other networks for downstream tasks.

5 Graphical Models vs. Deep Networks

The primary goal of deep networks is to derive a representation of the distribution of random variables (features) using a vast interconnected set of latent variables. Like GMs, adding more layers of hidden/latent variables allows the model to represent more complex distributions, however, unlike GMs, DNs are not concerned with correctly learning and inferring the values of these latent variables. In fact, the representation produced by the hidden variables in a DN is evaluated solely by the model's performance on a downstream task. The hidden variables can take on any value as long as the performance on the downstream task remains optimal.

5.1 Study of Belief Networks as GMs

In [1], the authors compare the performance of mean-field belief propagation and other approximate inference algorithms when applied to deep sigmoid belief networks. The values of the hidden variables were simulated and the error was measured between the approximated values and the true values of the hidden variables. The study found that belief propagation (i.e. back-propagation) performs the worst, while mean-field approximations algorithms perform much better, in terms of, both, time and accuracy.

6 Optimizing Optimization

Recall that a RBM (or any other undirected GM), when unrolled yields infinite instantiations of another GM such that each instantiation shares the same weights. During training, we perform one weight update on each GM instantiation and move to the next until the weights converge. This process is akin to gradient descent, in that we have theoretical guarantees of converging to an optimal point in the limit. In contrast, deep learning models keep updating the weights of each sub-model until convergence before moving on to the next. Therefore, deep learning models are not designed to asymptotically converge and can be seen as optimizing the optimization steps of an undirected GM using backpropagation.

7 Dealing with Structured Prediction in Deep Learning

Structured graphical models, such as CRFs, can be converted to deep learning models by urolling them for a fixed number of steps and optimizing each step using backpropagation. Similarly, message passing based inference algorithms can be trucated and converted into computational graphs.

References

[1] Eric P Xing, Michael I Jordan, and Stuart Russell. A generalized mean field algorithm for variational inference in exponential families. *arXiv preprint arXiv:1212.2512*, 2012.

10-708: Probabilistic Graphical Models, Spring 2020

12:Deep generative models: overview of the theoretical basis and connection

Lecturer: Eric P. Xing *Scribe: Shangbang Long Ankit Shrivastava, Prakhar Gupta*

1 Deep generative models

Deep generative models (DGMs) are probably the most popular research topic nowadays (CVPR, ICML, NeurIPS, etc.). This lecture is about a unifying theoretical perspective of DGMs.

DGMs can perform: style transfer/fusion, music/image generations, and etc..

DGMs are probabilistic distributions of a set of variables. *Deep* means having multiple layers of hidden variables.

1.1 Early forms of deep generative models

1.1.1 Sigmoid belief nets

One example is the Sigmoid Belief Networks in which lower layers connect to upper layers via sigmoid functions:

$$p(X_{n,k} = 1|\theta_{n,k}, X_{n-1}) = \sigma(\theta_{n,k}^T X_{n-1})$$

where X_n is a vector representing the hidden variables of the $n-$th layer, $\theta_{n,k}$ is a vector that represents how the $k-$th unit of layer n is connected to the lower layer.

These are authentic and genuine PGMs, based on the Bayesian rules.

1.1.2 Helmholtz machines

Helmholtz machines express a dual, alternative process that unifies inference and generative process:

$$X_n = G_\theta(X_{n-1})$$

and

$$X_{n-1} = F_\phi(X_n)$$

1.1.3 Predictability minimization

PM defines a training procedure. The encoding models represent the input data as latent variables. There is an extra predictability layer on top of the latent variables that learns to predict the latent variables. At the same time, PM learns with a constraint that minimizes the predictability of the latent variables.

1.1.4 Learning procedure

The training of such models follows an EM style framework, vias sampling and inference alternatively.

Another option is to optimize a variational lower bound:

$$log(p(x)) \geq E_{q_\phi(z|x)}[log(p_\theta(x,z))] - KL(q_\phi(z|x)||p(z)) := L(\theta,\phi;x)$$

and optimization target:

$$max_{\theta,\phi} L(\theta,\phi;x)$$

1.2 Resurgence of deep generative models

Variational autoencoders (VAEs) is the first modern, actively used deep generative models. VAEs consist of a generative model and an inference model that are trained in a variational way. Nearly at the same period, Generative adversarial networks (GANs) are proposed. GANs consist of a generator that transforms latent variables into data domain and a discriminator that learns to distinguish real data and generated (fake) data.

2 Theoretical basis of deep generative models

2.1 Recap: Variational Inference

Consider a generative model $p_\theta(x|z)$ and prior $p(z)$. The joint distribution is computed as: $p_\theta(x,z) = p_\theta(x|z)p(z)$. Assume variational distribution $q_\phi(z_x)$. Then the objective is to maximize lower bound for log likelihood:

$$\begin{aligned}
&\log p(x) \\
&= KL\left(q_\phi(\mathbf{z}|x)\|p_\theta(\mathbf{z}|x)\right) + \textstyle\int_z q_\phi(\mathbf{z}|x)\log\frac{p_\theta(x,z)}{q_\phi(z|x)} \\
&\geq \textstyle\int_z q_\phi(z|x)\log\frac{p_\theta(x,z)}{q_\phi(z|x)} \\
&:= L(\theta,\phi,x)
\end{aligned}$$

which is equivalently to minimize free energy:

$$F(\boldsymbol{\theta},\phi;x) = -\log p(x) + KL\left(q_\phi(z|x)\|p_\theta(z|x)\right)$$

During training, the following two EM steps are taken alternatively:

•$E-step:$ maximize $\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x})$ wrt. $\boldsymbol{\phi}$, with $\boldsymbol{\theta}$ fixed .

If closed form solutions exist, then $q_\phi^*(z|x) \propto \exp[\log p_\theta(x, z)]$.

•$M-step:$ maximize $\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x})$ wrt. $\boldsymbol{\theta}$, with $\boldsymbol{\phi}$ fixed

2.2 Wake sleep algorithm

Wake sleep algorithm maximizes data log-likelihood with two steps of loss relaxation:

2.2.1 *Wake* phase

Maximize the variational lower bound of log-likelihood, or equivalently, minimize the free energy: $F(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}) = -\log p(\boldsymbol{x}) + KL\left(q_\phi(\boldsymbol{z}|\boldsymbol{x}) \| p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})\right)$

In the wake phase, the algorithm gets samples from $q_\phi(z|x)$ through inference on hidden variables, and then use the samples as targets for updating the generative model $p_\theta(\mathbf{Z}|\boldsymbol{x})$. This phase corresponds to the variational M step, i.e.:

$$\max_{\boldsymbol{\theta}} \mathrm{E}_{\boldsymbol{q}_{\boldsymbol{\phi}}(\mathbf{z}|\boldsymbol{x})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}, \mathbf{z})\right]$$

2.2.2 *Sleep* phase

Minimize a different objective (reversed KLD) wrt) to ease the optimization: $F'(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}) = -\log p(\boldsymbol{x}) + KL\left(p_\theta(\boldsymbol{z}|\boldsymbol{x}) \| q_\phi(\boldsymbol{z}|\boldsymbol{x})\right)$.

The sleep phase corresponds to the E step, i.e.:

$$\max_{\phi} \mathrm{E}_{q_\phi(\mathbf{z}|x)}\left[\log p_\theta(x, z)\right]$$

However, one major difficulty is that the gradient for the distribution q_ϕ is difficult to compute. Instead, we use the log-derivative trick:

$$\nabla_\phi E_{q_\phi}\left[\log p_\theta\right] = \int \nabla_\phi q_\phi \log p_\theta = \int q_\phi \log p_\theta \nabla_\phi \log q_\phi = E_{q_\phi}\left[\log p_\theta \nabla_\phi \log q_\phi\right]$$

Then we can use Monte Carlo to estimate the gradients.

2.3 Variational autoencoders

Variational lower bound:

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}) = \mathrm{E}_{q_\phi(\mathbf{Z}|\boldsymbol{x})}\left[\log p_\theta(\boldsymbol{x}, \mathbf{z})\right] - \mathrm{KL}\left(q_\phi(\mathbf{z}|\boldsymbol{x}) \| p(\mathbf{z})\right)$$

Similar to Wake-Sleep algorithm, the procedure optimizes $\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x})$ w.r.t. θ and ϕ alternatively. VAEs also use reparametrization trick to reduce variance.

Recall:

$$\nabla_\phi E_{q_\phi}\left[\log p_\theta\right] = E_{q_\phi}\left[\log p_\theta \nabla_\phi \log q_\phi\right]$$

The scale factor $\log p_\theta$ has high variance.

Instead of sampling with $\mathbf{z} \sim q_\phi(\mathbf{z}|\boldsymbol{x})$, we parametrize z as a noise added to a function, $\mathbf{z} = \mathbf{g}_\phi(\boldsymbol{\epsilon}, \boldsymbol{x}), \quad \boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon})$.

Then the estimated gradients then become: $\nabla_\phi \mathrm{E}_{q_\phi(\mathbf{z}|\boldsymbol{x})}\left[\log p_\theta(\boldsymbol{x}, \mathbf{z})\right] = \mathrm{E}_{\epsilon\sim p(\epsilon)}\left[\nabla_\phi \log p_\theta\left(\boldsymbol{x}, \mathbf{z}_\phi(\boldsymbol{\epsilon})\right)\right]$. Empirically, it has lower variance.

2.4 Variational autoencoders Algorithm

One of the Variational Autoencoder algorithm is a minibath Auto-encoding variational Bayes proposed by [1]

Algorithm 1: Minibatch version of the Auto-encoding VB (AEVB) algorithm.

Result: Write here the result
$(\theta, \phi) \leftarrow$ initialize parameters;
while *repeat until convergence of parameters* (θ, ϕ) **do**
  $\boldsymbol{X}^M \leftarrow$ Random minibatch of M datapoints (drawn from full dataset) ;
  $\epsilon \leftarrow$ Random samples from noise distribution $p(\epsilon)$;
  $g \leftarrow \widetilde{\mathcal{L}}^M(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{X}^M, \boldsymbol{\epsilon})$ (Gradient of minibatch estimator);
  $(\theta, \phi) \leftarrow$ Update parameters using gradients g (e.g. SGD or Adagrad)
end

3 Generative adversarial networks

Generative adversarial networks are another type of model complementary to variational autoencoders. It does not have any particular inference model but consist to two main components a Generative model and a Discriminator. The generative model map latent variable $\mathbf{z}$ coming from some prior i.e. $\mathbf{z} \sim p(\mathbf{z})$ to data space $\mathbf{x}$ using a transformation function G_θ, that is $\mathbf{x} = G_\theta(\mathbf{x})$. The discriminator component $D_\phi(\mathbf{x})$ outputs the probability that $\mathbf{x}$ came from data rather than the generator.

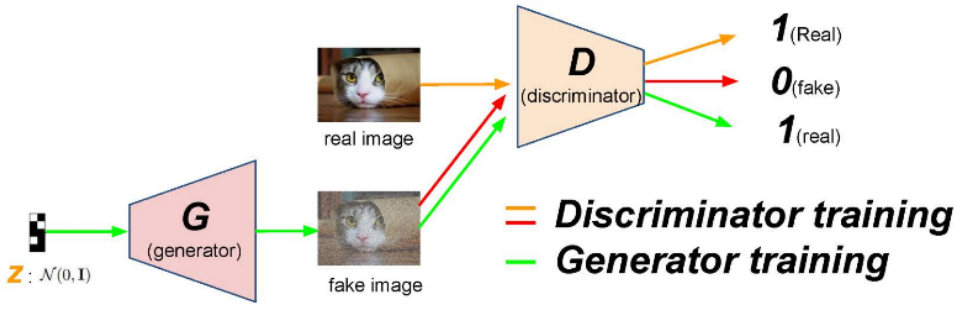


Figure 1: General architecture of GANS

Figure 1 shows a general GANS model. The model is trained by competing generator and discriminator against each other.

- It trains Discriminator to maximize the probability of correctly classifying real training examples and fake generated examples by

$$\max_D \mathcal{L}_D = \mathrm{E}_{\mathbf{x}\sim p_{data(\mathbf{x})}}[\log D(\mathbf{x})] + \mathrm{E}_{\mathbf{x}\sim G(\mathbf{z}),\mathbf{z}\sim p(\mathbf{z})}[\log 1 - D(\mathbf{x})]$$

- At the same time train Generator to beat the discriminator by

$$\min_G \mathcal{L}_G = \mathrm{E}_{\mathbf{x}\sim G(\mathbf{z}),\mathbf{z}\sim p(\mathbf{z})}[\log 1 - D(\mathbf{x})]$$

  Since the original loss suffers from vanishing gradient when Discriminator is too strong, in practice the Generator is trained by

$$\max_G \mathcal{L}_G = \mathrm{E}_{\mathbf{x}\sim G(\mathbf{z}),\mathbf{z}\sim p(\mathbf{z})}[\log D(\mathbf{x})]$$

The goal while training GANs is to achieve optimal state when generator and discriminator are competing among each other i.e.

- Generator produces the data using the distribution which is same as of the data distribution

$$p_G(\mathbf{x}) = p_{data}(\mathbf{x})$$

- And, discriminator assigning the probability randomly with probability of .5

$$D(\mathbf{x}) = \frac{p_{data}(\mathbf{x})}{p_{data}(\mathbf{x}) + p_G(\mathbf{x})} = \frac{1}{2}$$

3.1 GANs objective in variational-EM format

Since the first GANs model has been proposed there has been multiple proposed GANs to improve desired results but the practice of designing of models are often compared to alchemy as it is difficult to explain why certain model is working. Hence, it is needed to view these models in different way. One of the ways to view GANs is using variational format and try to connect them with other deep generative models such as VAE and its variant

To understand GANs in terms of variational inference format, we first redefine the model such that the goal is to find implicit distribution over $\mathbf{x} \sim p_\theta(\mathbf{x}|y)$ where,

$$p_\theta(\mathbf{x}|y) = \begin{cases} p_{g_\theta}(\mathbf{x}) & y = 0 \quad \textbf{(distribution of generated images)} \\ p_{data}(\mathbf{x}) & y = 1 \quad \textbf{(distribution of data)} \end{cases}$$

Also, $\mathbf{x} \sim p_{g_\theta}(\mathbf{x}) \Leftrightarrow \mathbf{x} = G_\theta(\mathbf{z}), \mathbf{z} \sim p(\mathbf{z}|y = 0)$.

Now recall the conventional GANs formualtion as

$$\max_\phi \mathcal{L}_\phi = \mathrm{E}_{\mathbf{x}\sim p_{data(\mathbf{x})}}[\log D_\phi(\mathbf{x})] + \mathrm{E}_{\mathbf{x}=G_{\boldsymbol{\theta}(\mathbf{z})},\mathbf{z}\sim p(\mathbf{z}|y=0)}[\log 1 - D_\phi(\mathbf{x})]$$

$$\max_\theta \mathcal{L}_\theta = \mathrm{E}_{\mathbf{x}=G_{\boldsymbol{\theta}(\mathbf{z})},\mathbf{z}\sim p(\mathbf{z}|y=0)}[\log D_\phi(\mathbf{x})]$$

By rewriting formulation of GANs, it can be reviewed as

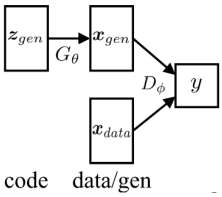


Figure 2: General architecture of GANS

- Introduce implicit distribution over $\mathbf{x} \sim p_\theta(\mathbf{x}|y)$, where

$$\mathbf{x} = G_{\boldsymbol{\theta}(\mathbf{z}),\mathbf{z}\sim p(\mathbf{z}|y)}$$

  We can call it as generative model since the data is generated based on hidden variable $\mathbf{z}$

- And Discriminator distribution $q_\phi(y|\mathbf{x})$ and $q^r_\phi(y|\mathbf{x}) = q_\phi(1-y|\mathbf{x})$ The discriminator model does not exist in variational inference literature, hence it should not be confused with inference model used in earlir vm

- the objective functions of GANs then becomes
  Learn parameters of discriminator model

$$\max_\phi \mathcal{L}_\phi = E_{p\theta(\mathbf{x}|y)p(y)}[\log q_\phi(y|\mathbf{x})]$$

  this kind of look like a posterior inference of hidden states because y is always observed as we need to determine whether it is true or false label. Now next task is to learn generative model parameters but with the loss function is reversed

$$\max_\theta \mathcal{L}_\theta = E_{p\theta(\mathbf{x}|y)p(y)}[\log q^r_\phi(y|\mathbf{x})]$$

  Hence the formulation look similar to wake sleep. This is further discussed n section 3.3 and section 3.4

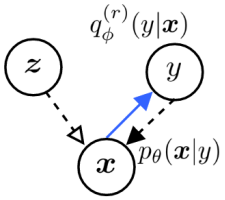


Figure 3: GANS viewed in form of Variational format

3.2 GANs: Minimizing KLD

Recall that in variational EM we minimize $-\log p(\mathbf{x}) + \mathrm{KL}(q_\phi)(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x}))$, that is, we minimize the KLD from the inference model to the posterior. We can similarly rewrite the objective of GAN in the form of minimizing KLD.

For each optimization step of $p_{\boldsymbol{\theta}(\mathbf{x}|y)}$ starting from an initial point $(\boldsymbol{\theta}_0, \boldsymbol{\phi}_0)$, let $p(y)$ be a uniform prior distribution, and

$$p_{\boldsymbol{\theta}=\boldsymbol{\theta}_0}(\mathbf{x}) = E_{p(y)}[p_{\boldsymbol{\theta}=\boldsymbol{\theta}_0}(\mathbf{x}|y)]$$
$$q^{\mathrm{r}}(\mathbf{x}|y) \propto q^{\mathrm{r}}(y|\mathbf{x}) p_{\boldsymbol{\theta}=\boldsymbol{\theta}_0}(\mathbf{x})$$

The update rule for $\boldsymbol{\theta}$ is in Lemma 1:

Lemma 1:

$$\nabla_\theta(-E_{p_\theta(\mathbf{x}|y)p(y)}[\log q^{\mathrm{r}}_{\phi=\phi_0}(y|\mathbf{x})]) \mid_{\theta=\theta_0} =$$
$$\nabla_\theta(E_{p(y)}[\mathrm{KL}(p_\theta(\mathbf{x}|y) \parallel q^{\mathrm{r}}(\mathbf{x}|y))] - \mathrm{JSD}(p_\theta(\mathbf{x}|y=0) \parallel p_\theta(\mathbf{x}|y=1))) \mid_{\theta=\theta_0} .$$

Here the generative model $p_\theta(\mathbf{x}|y)$ is equivalent to the variational distribution for the posterior $q^{\mathrm{r}}(\mathbf{x}|y)$. $p_{\boldsymbol{\theta}=\boldsymbol{\theta}_0}(\mathbf{x})$ is the prior.

Now, minimizing the KLD drives the generator $p_{g_{\boldsymbol{\theta}}(\mathbf{x})}$ to the true data distribution $p_{\mathrm{data}}(\mathbf{x})$. For a uniform y (being equally real or generated), by definition,

$$p_{\boldsymbol{\theta}=\boldsymbol{\theta}_0}(\mathbf{x}) = E_{p(y)}[p_{\boldsymbol{\theta}=\boldsymbol{\theta}_0}(\mathbf{x}|y)] = \frac{p_{g_{\boldsymbol{\theta}=\boldsymbol{\theta}_0}}(\mathbf{x}) + p_{\mathrm{data}}(\mathbf{x})}{2},$$

and thus we could break the KLD into two terms:

$$\mathrm{KL}(p_\theta(\mathbf{x}|y=1) \parallel q^{\mathrm{r}}(\mathbf{x}|y=1)) = \mathrm{KL}(p_{\mathrm{data}}(\mathbf{x}) \parallel q^{\mathrm{r}}(\mathbf{x}|y=1)) = \text{const.},$$

and

$$\mathrm{KL}(p_\theta(\mathbf{x}|y=0) \parallel q^{\mathrm{r}}(\mathbf{x}|y=0)) = \mathrm{KL}(p_{g_\theta}(\mathbf{x}) \parallel q^{\mathrm{r}}(\mathbf{x}|y=0)),$$

where

$$q^{\mathrm{r}}(\mathbf{x}|y=0) \propto q^{\mathrm{r}}(y=0|\mathbf{x}) p_{\boldsymbol{\theta}=\boldsymbol{\theta}_0}(\mathbf{x})$$

This can be seen as a mixture of $p_{g_{\boldsymbol{\theta}}(\mathbf{x})}$ and $p_{\mathrm{data}}(\mathbf{x})$ weighted by $q^{\mathrm{r}}(y=0|\mathbf{x})$. During the update we drive the generator $p_{g_{\boldsymbol{\theta}}(\mathbf{x})}$ towards $q^{\mathrm{r}}(\mathbf{x}|y=0)$ by minimizing the KLD, and drive it towards the true distribution $p_{\mathrm{data}}(\mathbf{x})$, as shown in Figure 4.

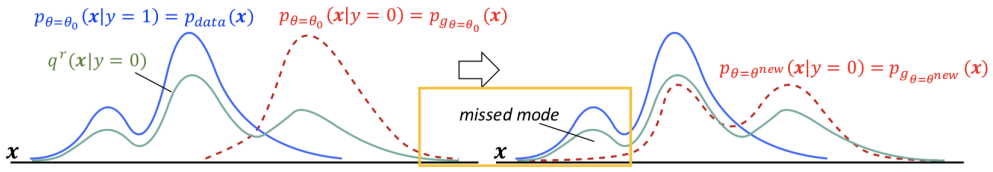


Figure 4: Minimizing KLD in GAN

The KL formulation tends to be large in the regions of x space where $q^{\mathrm{r}}(\mathbf{x}|y=0)$ is small unless $p_{g_{\boldsymbol{\theta}}(\mathbf{x})}$ is also small. Thus, $p_{g_{\boldsymbol{\theta}}(\mathbf{x})}$ tends to be small where $q^{\mathrm{r}}(\mathbf{x}|y=0)$ is small to avoid the penalization. Therefore, this KLD formulation allows GAN to recover the major modes, but also leads GAN to miss some minor modes of $p_{\mathrm{data}}(\mathbf{x})$, where $p_{g_{\boldsymbol{\theta}}(\mathbf{x})}$ and $q^{\mathrm{r}}(\mathbf{x}|y=0)$ are small and already give a small KLD. This phenomenon where some modes are missed but other captured well is what leads GAN to create sharp images.

	GANs (InfoGAN)	VAEs
Generative distribution	$p_\theta(\boldsymbol{x}\|y) = \begin{cases} p_{g_\theta}(\boldsymbol{x}) & y=0 \\ p_{data}(\boldsymbol{x}) & y=1. \end{cases}$	$p_\theta(\boldsymbol{x}\|\boldsymbol{z},y) = \begin{cases} p_\theta(\boldsymbol{x}\|\boldsymbol{z}) & y=0 \\ p_{data}(\boldsymbol{x}) & y=1. \end{cases}$
Discriminator distribution	$q_\phi(y\|\boldsymbol{x})$	$q_*(y\|\boldsymbol{x})$, perfect, degenerated
$\boldsymbol{z}$-inference model	$q_\eta(\boldsymbol{z}\|\boldsymbol{x},y)$ of InfoGAN	$q_\eta(\boldsymbol{z}\|\boldsymbol{x},y)$
KLD to minimize	$\min_\theta \mathrm{KL}\left(p_\theta(\boldsymbol{x}\|y) \,\|\| \, q^r(\boldsymbol{x}\|\boldsymbol{z},y)\right)$ $\sim \min_\theta \mathrm{KL}(P_\theta \,\|\| \, Q)$	$\min_\theta \mathrm{KL}\left(q_\eta(\boldsymbol{z}\|\boldsymbol{x},y)q_*^r(y\|\boldsymbol{x}) \,\|\| \, p_\theta(\boldsymbol{z},y\|\boldsymbol{x})\right)$ $\sim \min_\theta \mathrm{KL}(Q \,\|\| \, P_\theta)$

Figure 5: GAN vs VAE

3.3 GAN vs VAE

Lets recap the VAE objective

$$\max_{\boldsymbol{\theta},\boldsymbol{\eta}} \mathcal{L}^{\text{vae}}_{\theta,\eta} = E_{p_{\text{data}}(\boldsymbol{x})}\left[E_{\tilde{q}_\eta(\boldsymbol{z}|\boldsymbol{x})}\left[\log \tilde{p}_\theta(\boldsymbol{x}|\boldsymbol{z})\right] - \mathrm{KL}\left(\tilde{q}_\eta(\boldsymbol{z}|\boldsymbol{x})\|\tilde{p}(\boldsymbol{z})\right)\right]$$

We now assume a perfect discriminator $q_\star(y|\mathbf{x})$ telling whether $\mathbf{x}$ is real or generated, and $q^{\text{r}}_\star(y|\mathbf{x}) = q_\star(1 - y|\mathbf{x})$. Notice that q_* is degenerate since we are always generating fake $\mathbf{x}$. We can write $\mathcal{L}^{\text{vae}}_{\boldsymbol{\theta},\boldsymbol{\eta}}$ as

Lemma 2:

$$\begin{aligned}\mathcal{L}^{\text{vae}}_{\theta,\eta} &= 2 \cdot E_{p_{\theta_0}(\boldsymbol{x})}\left[E_{q_\eta(\boldsymbol{z}|\boldsymbol{x},y)q^r_*(y|\boldsymbol{x})}\left[\log p_\theta(\boldsymbol{x}|\boldsymbol{z},y)\right] - \mathrm{KL}\left(q_\eta(\boldsymbol{z}|\boldsymbol{x},y)q^r_*(y|\boldsymbol{x})\|p(\boldsymbol{z}|y)p(y)\right)\right] \\ &= 2 \cdot E_{p_{\theta_0}(\boldsymbol{x})}\left[-KL\left(q_\eta(\boldsymbol{z}|\boldsymbol{x},y)q^r_*(y|\boldsymbol{x})\|p_\theta(\boldsymbol{z},y|\boldsymbol{x})\right)\right]\end{aligned}$$

where the posterior is given as

$$p_\theta(\mathbf{z}|\mathbf{x},y) \propto p_\theta(\mathbf{x}|\mathbf{z},y)p(\mathbf{z}|y)p(y)$$

and is determined by the generative model $p_{\boldsymbol{\theta}(\mathbf{z}|\mathbf{x},y)}$ and the other two terms are fixed priors. In this way, VAE has the generative model on the right side of KLD, different to GAN.

Therefore, GAN while provides a "sharp" distribution of covering major modes while missing minor modes of the true distribution p_{data}, VAE provides a blurred or approximate distribution to cover all the modes while less sharp approximation of modes where p_{data} is small.

The Figure 5 summarizes the comparison between GAN and VAE's new formulation.

3.4 Linking GAN and VAE to Wake-sleep

Lets look back at the wake-sleep algorithm:

$$\text{Wake} : \max_{\theta} E_{q_\lambda(\mathbf{h}|\mathbf{x})p_{\text{data}}(\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{h})],$$
$$\text{Sleep} : \max_{\theta} E_{p_\theta(\mathbf{x}|\mathbf{h})p(\mathbf{h})}[\log q_\lambda(\mathbf{h}|\mathbf{x})].$$

We now discuss how VAE and GAN relate to the wake-sleep procedure.

VAE only deals with the wake phase and also learns the inference model $\boldsymbol{\eta}$, That is, the KLD term in the original variational free energy:

$$\max_{\theta,\eta} \mathcal{L}^{\text{VAE}}_{\theta,\eta} = E_{q_\eta(\mathbf{z}|\mathbf{x})p_{\text{data}}(\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - E_{p_{\text{data}}(\mathbf{x})}[\text{KL}(q_\eta(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z}))]$$

It does not involve the sleep-phase objective.

GAN only deals with the sleep phase and extends it by learning the generative parameter $\boldsymbol{\theta}$:

$$\max_{\phi} \mathcal{L}_\phi = E_{p_\theta(\mathbf{x}|y)p(y)}[\log q_\phi(y|\mathbf{x})],$$
$$\max_{\phi} \mathcal{L}_\theta = E_{p_\theta(\mathbf{x}|y)p(y)}[\log q^{\text{r}}_\phi(y|\mathbf{x})].$$

It does not involve the wake-phase objective.

3.5 Conclusion

- GANs and VAEs are essentially minimizing KLD in opposite directions and extend two phases of classic wake sleep algorithm, respectively
- This lecture discussed a general formulation useful for analyzing a broad class of existing DGM modelsm and can inspire new models and algorithms.

References

[1] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

10-708: Probabilistic Graphical Models, Spring 2020

13 : Deep Generative Models II

Lecturer: Eric P. Xing *Scribe: Cheng Cheng, Chiyu Wu, Xinhe Zhang*

1 Generative Adversarial Networks (GANs)

1.1 Vanilla GAN [GPAM$^+$14]

The goal of Vanilla GAN in general is to generate new images via learning from the original image data. The model primarily consists of two parts, i.e. a generator which produces image from noise z and a discriminator which tries to distinguish between z and the true images x. The Vanilla GAN is performing an minimax game between a generator G and a discriminator D. G tries to generate samples as close to the true data as possible whereas D tries to distinguish them apart. The loss objective is therefore formulated as:

$$\min_G \max_D V(D,G) = E_{x\sim p_x(x)}[\log D(x)] + E_{z\sim p_z(z)}[\log(1 - D(G(z)))]$$

where $z \sim p_z(z)$ is a random noise (usually standard Gaussian) and $x \sim p_x(x)$ is the true data distribution. Images generated from GAN are pretty good without any traditional feature engineering.

1.2 Wasserstein GAN [GAA$^+$17]

Vanilla GAN relies on KL-divergence which causes instability during training. For instance, if the data and model's manifolds are different, there may exist x such that $P_g(x) = 0$ but $P_d(x) > 0$, in which the KL divergence is infinite.

In order to solve this problem, Wasserstein GAN (WGAN) is proposed. WGAN relies on Wasserstein distance from the optimal transport literature. It measures the minimum transportation cost for transforming one distribution into another distribution.

For WGAN to work, the Lipschitz continuity property $|D|_L \leq K$ need to be ensured. The loss term is therefore formulated as:

$$W(p_d, p_g) = \frac{1}{K} \sup_{||D||_L \leq K} E_{x\sim p_d}[D(x)] - E_{x\sim p_g}[D(x)]$$

Compared to vanilla GAN, WGAN provides more stable gradients at the place where vanilla GAN has vanishing gradients (Fig.1).

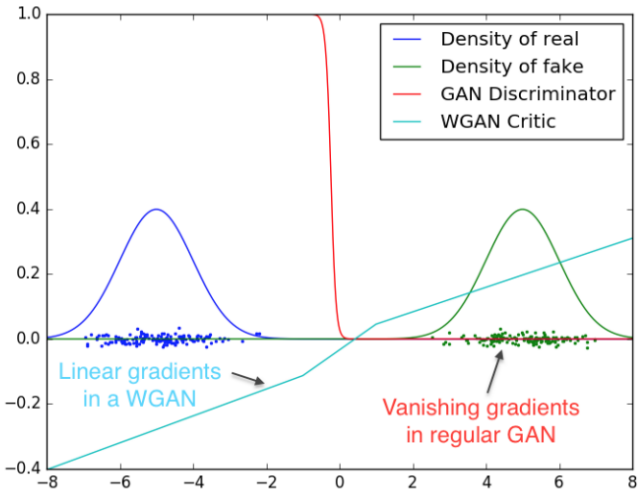


Figure 1: WGAN vs. Vanilla GAN on Gaussian data [GAA+17]

1.3 Progressive GAN [KALL18]

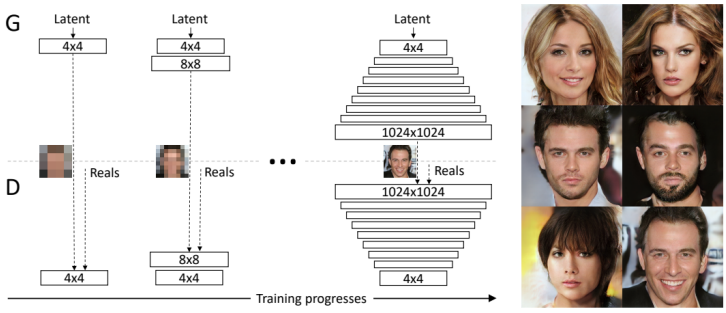


Figure 2: Progressive GAN for generating facial images [KALL18]

The classic GAN-based methods can only generate images with low resolution. Therefore, the progressive GAN is proposed to enhance the quality of training and generating large images.

Intuitively, progressive GAN works to first extract structure of the image using lower layers and then attend to details on the image. Therefore during training, low resolution images are first generated, and then their resolutions are increased by adding additional layers.

Progressive GAN is shown to generate high resolution facial images (Fig.2).

1.4 Big GAN [BDS19]

The authors of BigGAN finds that by up-scaling the Vanilla GAN, significantly better learning results could be achieved. In the paper, $2x - 4x$ more parameters as well as $8x$ larger batch size are experimented. Also,

model collapse is avoided by utilizing a strong discriminator at the initial training stage, and then gradually relax it. High quality images are generated by Big GAN (Fig.3)

Figure 3: Images generated by Big GAN [BDS19]

2 Normalizing Flow (NF)

2.1 Overview

Normalizing Flows can transform a simple distribution into a complex one by applying sequence of invertible transformation functions. Through a chain of transformation, we replaced the current variable by the new one and eventually obtain a probability distribution for the target variable. Figure 4 shows the whole chain of the normalizing flow.

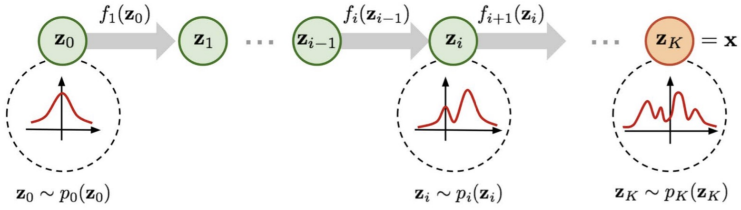


Figure 4: The process of a normalizing flow, transforming a simple distribution $\mathbf{p}_0(\mathbf{z}_0)$ to a complex distribution $\mathbf{p}_K(\mathbf{z}_K)$. Figure courtesy: Lilian Weng

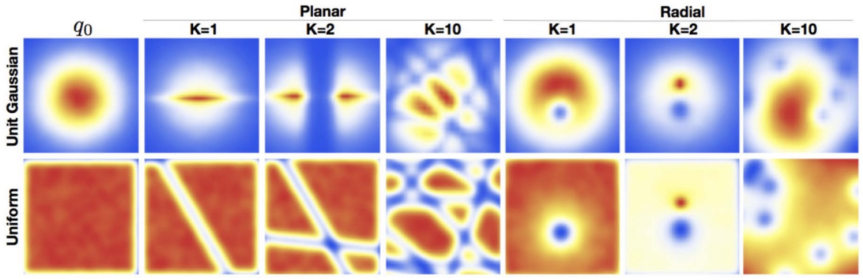


Figure 5: Effect of normalizing flow on two distributions.

Figure 5 shows the effect of the normalizing flows. We can see that a spherical Gaussian distribution can be transformed into a bimodal distribution through two successive transformations [RM15].

Now given a random variable $\mathbf{z}$ from a simple distribution $\mathbf{p}(\mathbf{z})$, we apply a invertible transformation function

$\mathbf{f}$ to obtain a new variable $\mathbf{x}$ from a more complex distribution.

$$\begin{aligned}\mathbf{z} &\sim \mathbf{p}(\mathbf{z}) \\ \mathbf{x} &= \mathbf{f}(\mathbf{z}) \\ \mathbf{z} &= \mathbf{f}^{-1}(\mathbf{x})\end{aligned}$$

According to Change of Variable Theorem, we have:

$$\begin{aligned}\mathbf{p}(\mathbf{x}) &= \mathbf{p}(\mathbf{z}) \left|\det \frac{\mathbf{dz}}{\mathbf{dx}}\right| \\ &= \mathbf{p}(\mathbf{f}^{-1}(\mathbf{x})) \left|\det \frac{\mathbf{df}^{-1}}{\mathbf{dx}}\right|\end{aligned}$$

where $\left|\det \frac{\mathbf{df}^{-1}}{\mathbf{dx}}\right|$ is the Jacobian determinant of the function $\mathbf{f}^{-1}$.

Now considering the normalizing flow in the figure 4, we need to obtain $\mathbf{z}_K$ through chain of transformations $\mathbf{f}_1$, $\mathbf{f}_2$, ..., $\mathbf{f}_K$. That is,

$$\begin{aligned}\mathbf{z}_0 &\sim \mathbf{p}(\mathbf{z}_0) \\ \mathbf{x} = \mathbf{z}_K &= \mathbf{f}_K \cdot \mathbf{f}_{K-1} \ldots \mathbf{f}_0(\mathbf{z}_0) \\ \mathbf{z}_i &= \mathbf{f}_i^{-1}(\mathbf{z_{i-1}}) \\ \mathbf{p}(\mathbf{z_i}) &= \mathbf{p}(\mathbf{z_{i-1}}) \left|\det \frac{\mathbf{dz_{i-1}}}{\mathbf{dz_i}}\right|\end{aligned}$$

Then the training objective is to maximize the data log-likelihood:

$$\log \mathbf{p}(\mathbf{x}) = \log \mathbf{p}(\mathbf{z}_0) + \sum_{i=1}^{K} \log \left|\det \frac{\mathbf{dz_{i-1}}}{\mathbf{dz_i}}\right|$$

2.2 Case Study: GLOW

GLOW is a flow based generative model, consisting of a series of steps of flow and combined in a multi-scale architecture [KD18], as shown in figure 6. Each step of flow consists of actnorm, an invertible 1×1 convolution and a coupling layer.

For the actnorm layer, it is similar to batch normalization [IS15]. Any channel after the actnorm activation layer will have zero mean and unit variance. The key difference between actnorm layer and batch normalization layer is that actnorm can work reasonably well when batch size is only 1. 1×1 convolutional layer is used to perform permutation operation. And LU Decomposition is applied to reduce the computational cost of calculating the determinant of weight matrix $\det(W)$. The affine Coupling Layer is a powerful reversible transformation.

GLOW demonstrates a significant improvement performance in terms of log-likelihood on standard image modeling benchmarks.

3 Integrating Domain Knowledge into Deep Learning

Deep learning has been proven very successful in lots of areas. However, deep learning still suffers from several limitations. It heavily rely on massive labeled data which is really expensive. Also, deep network can

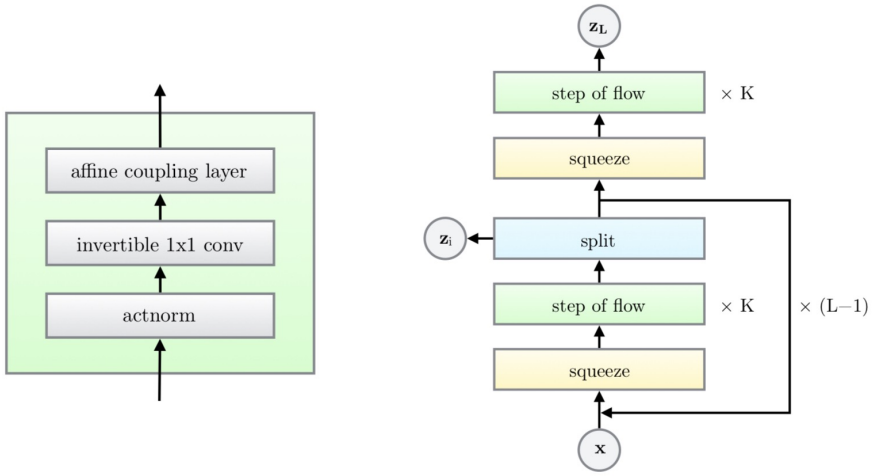


Figure 6: The architecture of GLOW

be regarded as a black-box, trained end-to-end, which is uninterpretable. Furthermore, it is hard to encode human intention and domain knowledge into the deep neural network. For human, we not only learn from concrete examples as deep neural networks, but also learn from abstract knowledge such as logic rules.

Now we hope to integrate domain knowledge into deep neural network. Consider a statistical model $\mathbf{p}_\theta(\mathbf{x})$ and a constraint function $\mathbf{f}_\phi(\mathbf{x})$:

$$\mathbf{x} \sim \mathbf{p}_\theta(\mathbf{x})$$
$$\mathbf{f}_\phi(\mathbf{x}) \in \mathbf{R}$$

where higher $\mathbf{f}_\phi(\mathbf{x})$ value means our generated x is better with regard to the prior knowledge.

Let's consider a image generation problem, as shown in Figure 7. Here we hope to generate images which has the consistent pose with the input pose template. We first use $\mathbf{p}_\theta$ as our generative model, which has two inputs, one is the source image, and the other is the target pose template. Then we apply a constraint function $\mathbf{f}_\phi$ to ensure the generated image having the consistent structure with the true target. Here $\mathbf{f}_\phi$ can be regarded as a human part parser, able to extract poses from an image. By this way, we can generate new images with desired structures.

4 Learning with Constraints

Just like what we do with GAN training, we fold the constraint function $f_\phi(x)$ under expectation of the generation distribution p_θ.

4.1 Objective

$$\min_\theta \mathcal{L}(\theta) - \alpha \mathbb{E}_{p_\theta}[f_\phi(x)]$$

Where the $\mathcal{L}(\theta)$ is the regular objective, such as the cross-entropy loss, etc. and $\alpha \mathbb{E}_{p_\theta}[f_\phi(x)]$ is the regularization, i.e. the imposed constraints, which is difficult to compute because when taking the derivative of

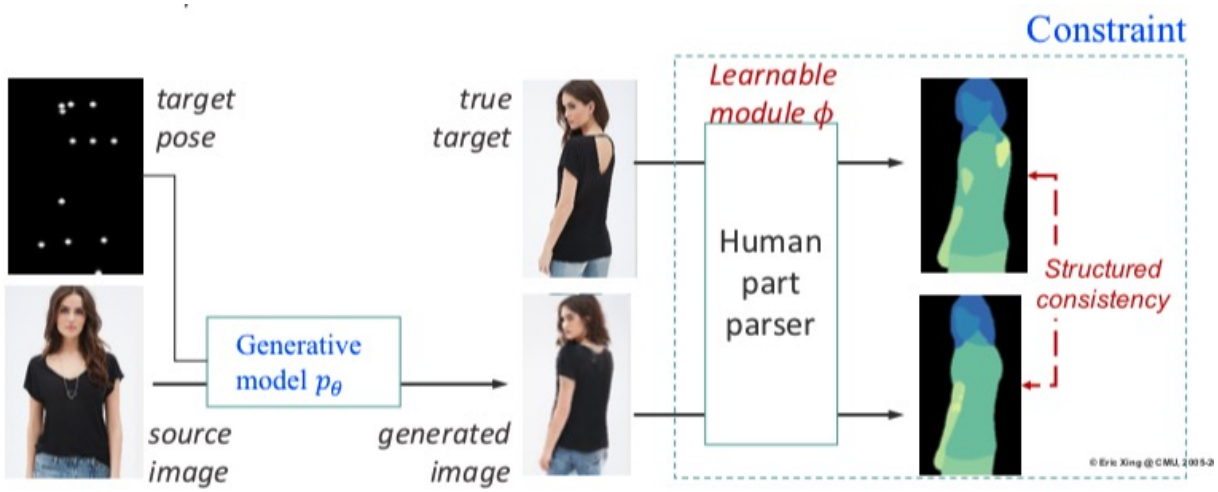


Figure 7: Pose guided person image generation

the expectation, we use the log probability trick. However, in this case, the magnitude of the log term will explode as explained in the weak-sleep algorithm.

Due to the difficulty of computing the regularization, we introduce a variational approximation q to the true distribution. We have a minimax situation: on the one hand, we would love to maximize the loss under the approximating distribution; on the other hand, we want to minimize the discrepancy between the true distribution and the approximation. As a result, we compute the KL divergence term minus the expected loss instead:

$$\mathcal{L}(\theta, q) = \mathrm{KL}(q(x) \| p_\theta(x)) - \lambda \mathbb{E}_q[f_\phi(x)]$$

This method is addressed as the posterior regularization method [GGT+10].

We then introduce a scaling term α to control the relative contribution of the two parts, our revised objective is:

$$\min_{\theta, q} \mathcal{L}(\theta) + \alpha \mathcal{L}(\theta, q)$$

4.2 Learning

An EM-like approach is applied.

E-step:

$$q^\star(x) = p_\theta(x) \exp\{\lambda f_\phi(x)\} / Z$$

M-step:

$$\min_\theta \mathcal{L}(\theta) - \mathbb{E}_{q^\star}[\log p_\theta(x)]$$

4.3 Logical Rule Constraints

Putting everything together, the set up now is we consider a supervised learning $p_\theta(y|x)$, our input-targe space is (X, Y), and our first-order logic rules (r, λ).

Given l rules, we can train the model by alternatively training for the variational distribution $q^\star(y|x)$ and the true generative model:

1. E-step

$$q^\star(y|x) = p_\theta(y|x)\exp\left\{\sum_l \lambda_l r_l(y, x)\right\}/Z$$

2. M-step

$$\min_\theta \mathcal{L}(\theta) - \mathbb{E}_{q^\star}[\log p_\theta(y|x)]$$

Note in the M-step we are using the variational which is easier to compute.

4.4 Rule Knowledge Distillation

Instead of learning a difficult target $p_\theta(y|x)$one-shot, we iterate with an auxiliary $q(y|x)$, which is called the "teacher" and it is often an ensemble. The target $p_\theta(y|x)$ is called the student. We match the soft predictions of the teacher network and the student network. This interaction between the teacher(s) and the student will ultimately get the student closer to the teachers.

With our setup, at each iteration t, we update the $\theta^{(t+1)}$ with:

$$\arg\min_{\theta\in\Theta} \frac{1}{N}\sum_{n=1}^{N}(1-\pi)\ell(y_n, \sigma_\theta(x_n)) + \pi\ell(s_n^{(t)}, \sigma_\theta(x_n))$$

where π is the balancing parameter, y_n is the true hard label, $\sigma_\theta(x_n)$ is the soft prediction of $p_\theta(y|x)$ and $s_n^{(t)}$ is the soft prediction of the teacher network.

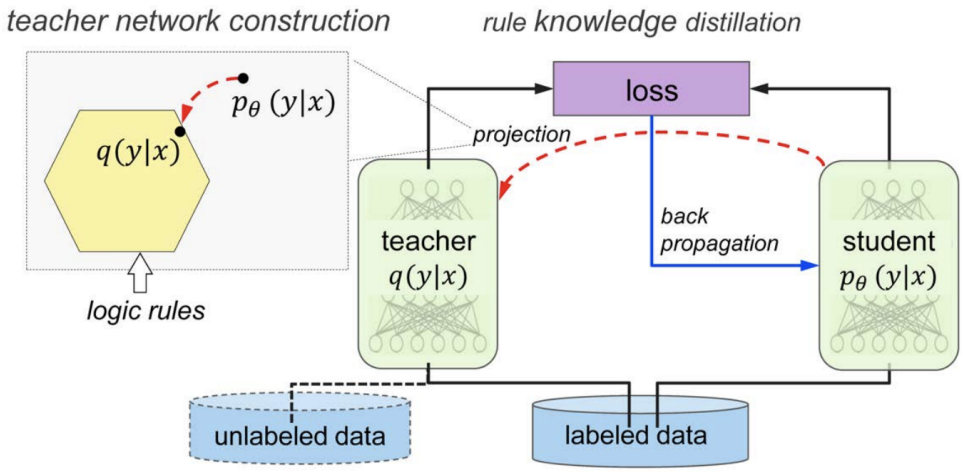


Figure 8: Graphical illustration of knowledge distillation.

There are a few catches with this approach:

1. The rule function f required a few properties: many occasions required the functions to be differentiable.
2. How to expand the input space beyond the logical rules is interesting. For example, if we phrase the reward function in reinforcement learning as a teacher model, then reinforcement learning is an instance of our approach as well.
3. The architecture could be modified and this approach could be extended to different domains of tasks as well. For instance, we could involve LSTM and solve language problems.

References

[BDS19] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale GAN training for high fidelity natural image synthesis. In *International Conference on Learning Representations*, 2019.

[GAA+17] Ishaan Gulrajani, Faruk Ahmed, Martín Arjovsky, Vincent Dumoulin, and Aaron C. Courville. Improved training of wasserstein gans. *CoRR*, abs/1704.00028, 2017.

[GGT+10] Kuzman Ganchev, Jennifer Gillenwater, Ben Taskar, et al. Posterior regularization for structured latent variable models. *Journal of Machine Learning Research*, 11(Jul):2001–2049, 2010.

[GPAM+14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

[IS15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[KALL18] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of GANs for improved quality, stability, and variation. In *International Conference on Learning Representations*, 2018.

[KD18] Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems*, pages 10215–10224, 2018.

[RM15] Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows. *arXiv preprint arXiv:1505.05770*, 2015.

10-708: Probabilistic Graphical Models, Spring 2020

14: Deep Sequence Models

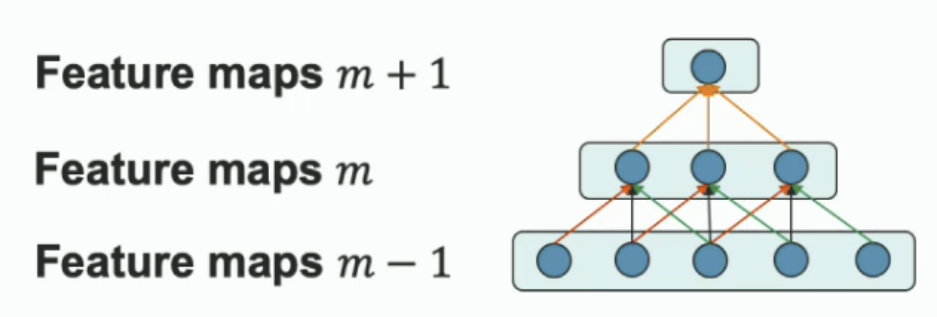
Lecturer: Zhiting Hu *Scribe: Karmesh Yadav, Vivek Pandit, Chirag Pabbaraju, Laavanye Bahl*

This lecture is about four fundamental and popular model architectures.

1 CNN

CNNs are biologically-inspired variants of MLPs that exploit the strong spatial local correlations present in images. The biological concept of the receptive field states that the visual cortex contains a complex arrangement of cells that are sensitive to small sub-regions that are tiled to cover the visual field.

There are three properties of convolutional neural networks:

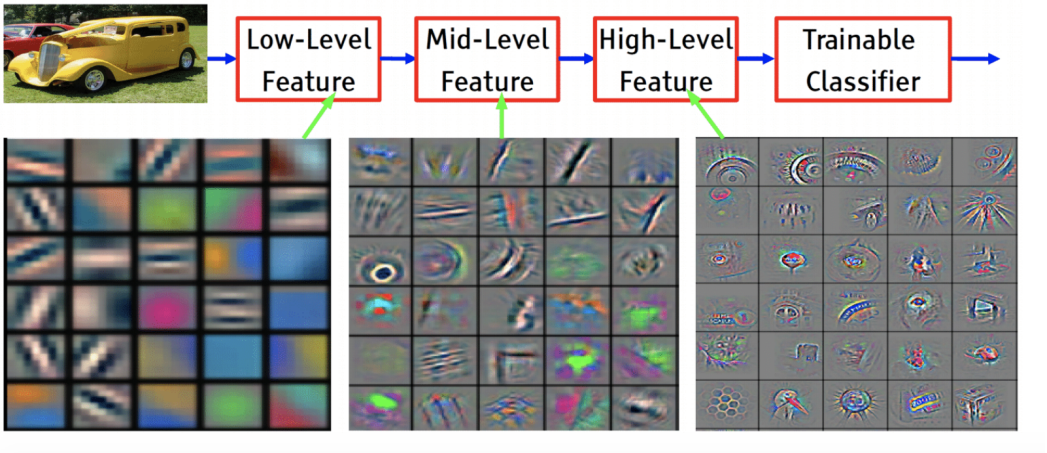


- **Sparse connectivity**
  Each neuron is connected to a subset of neurons from the preceding layer.

- **Shared weights**
  Weights are shared in the form of filters which helps in traversing the image.

- **Increasingly 'global' receptive fields**
  Using the biological analogue, simple cells detect local features while complex ones pool the outputs of simpler cells.

- **Heirarchical Representation Learning**
  Stacking multiple layers can result in lower layers learning low-level features like edges and corners, while upper layers learn high-level representations like textures.



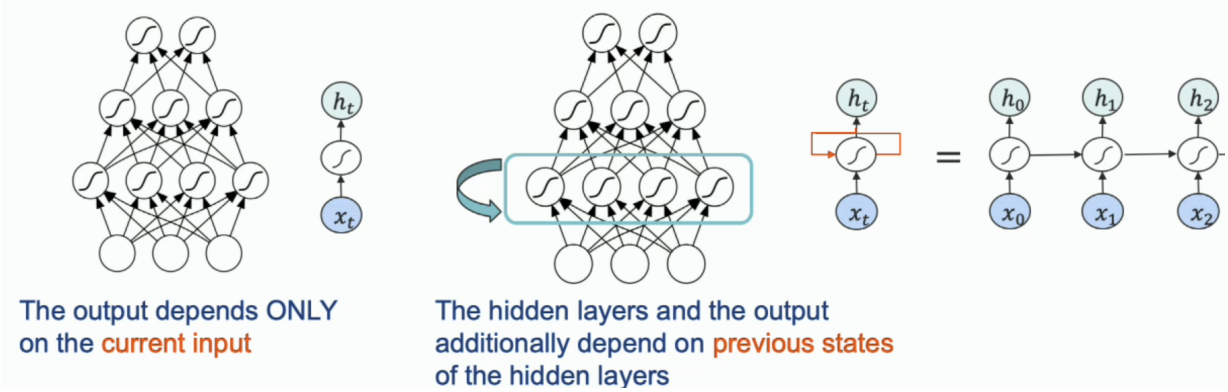
Evolution of ConvNets

1. AlexNet, 8 layers, 2012 [1]
2. VGG, 19 layers, 2015 [9]
3. GoogLeNet, 22 layers, 2014 [18]
4. ResNet, 152 layers, 2015 [6]

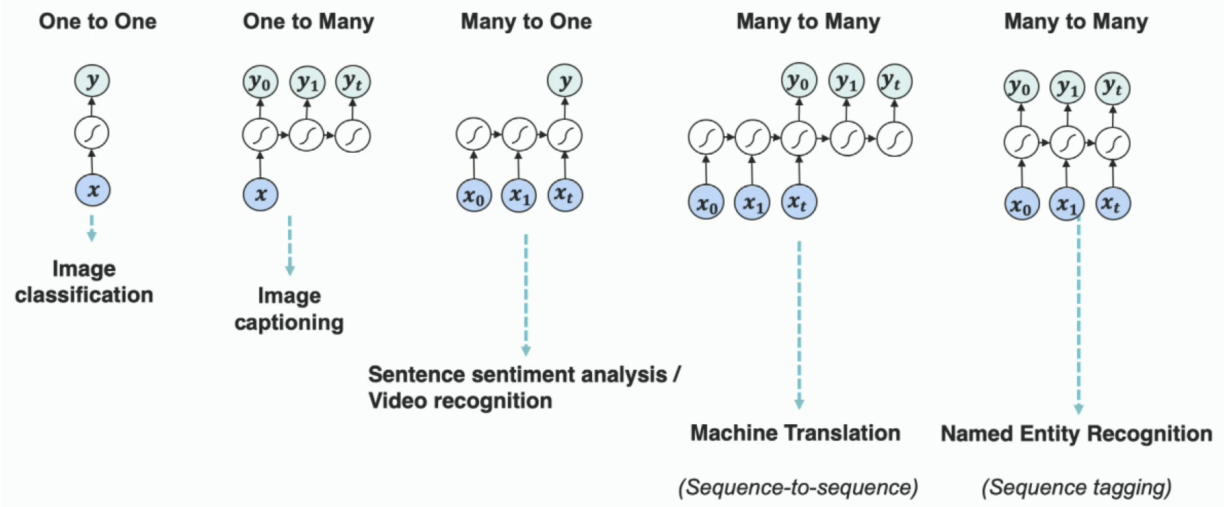
2 RNN

Following are the differences between CNNs and RNNs:

- CNNs are used for spatial modelling. The temporal (or sequential) analogue to the CNN is the RNN.
- RNNs can have a variable number of computation steps unlike CNNs.
- Unlike MLPs and CNNs, RNN outputs depend not only on the current input, but also on the previous states of hidden layers as shown in figure below.

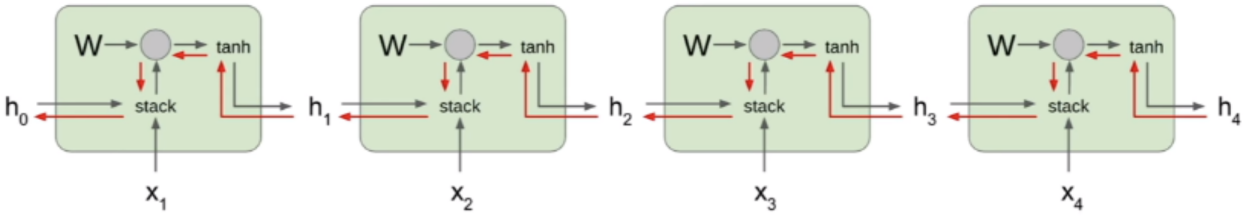


2.1 Various forms of RNNs with applications



2.2 Vanishing/Exploding Gradients in RNNs [3, 14]

Following is an illustration of the flow of computation in an RNN:

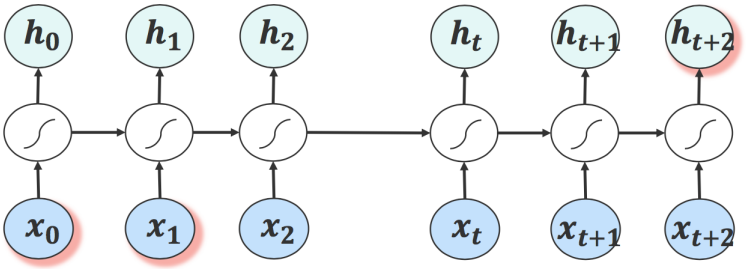


where

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

As we can see, computing the gradient of any loss function w.r.t. say h_0 will involve repeated multiplication of the matrix W along with repeated tanh functions. Thus, if the largest singular value of W is greater than 1, the chained multiplication of W will lead to an exploding gradient w.r.t. h_0. Similarly, if the largest singular value of W is less than 1, this will lead to a vanishing gradient w.r.t. h_0. To remedy the problem of exploding gradients, we can apply gradient clipping i.e. if the norm of the gradient is larger than some predetermined threshold, we can rescale the gradient to have norm equal to the threshold. However, we do not have such suitable ad-hoc fixes for the vanishing gradient problem.

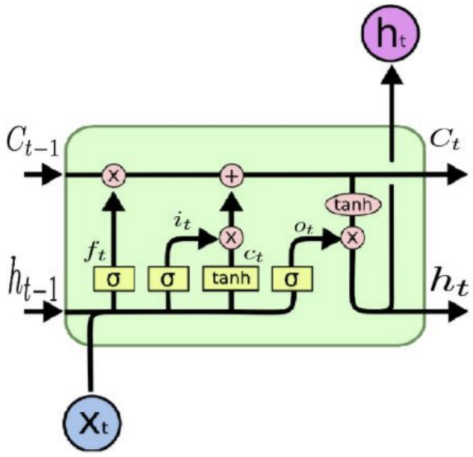
2.3 Long term dependency problem



This optimization issue leads to problems in modelling long-term dependencies in sequences. For example, suppose we are fed the sequence "I live in France and I know -" and our sequence model is tasked to predict the next word. Here, since the word "France" was just a few words past, the gradient information should not have been killed during optimization and the model should have learned to output "French". However, if we instead feed the sequence "I live in France, a beautiful country, and I know -", the gradient information from a state that is further behind would likely have been killed and the model would have difficulty in outputting the desired word.

3 Long Short Term Memory (LSTM) [7]

To alleviate the problems of modelling long-term dependencies due to vanishing/exploding gradients, the LSTM model was introduced in 1997. This variant of RNNs involves gating mechanisms to prevent loss of gradient information over time. Specifically an LSTM cell structure consists of forget gates, input gates and output gates. These gates control the operations of reading, writing and resetting information into the maintained cell state.



Here, x_t denotes the input, h_t denotes the hidden state and c_t denotes the cell state at time step t and σ denotes the sigmoid activation. Also, the pink circles with $+$ and $\times$ denote coordinate-wise addition and multiplication of vectors.

- Forget Gate: This gate decides what must be removed from the existing cell state. Specifically, the forget gate performs the following computation, w.r.t. the cell parameters W_f and b_f. Here, $[h_{t-1}, x_t]$ denotes the concatenation of vectors h_{t-1} and x_t.

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

- Input Gate: This gate determines the new information to be written into the cell state. Specifically, the input gate performs the following computation

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

- Update Cell State: Then, we perform the following computation, which updates the outputs of the forget gate and the input gate into the cell state.

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

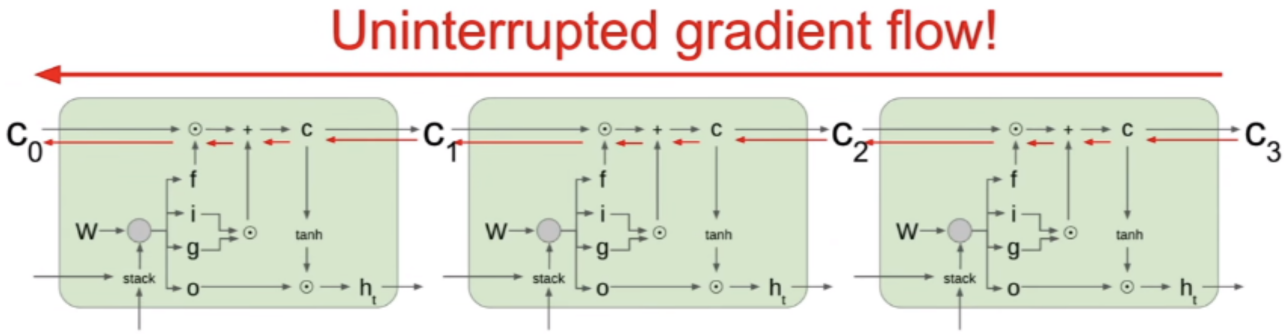
  Specifically, the first product removes unnecessary information from the cell state and the second product introduces new information into the cell state scaled appropriately by the values in $\tilde{C}_t$.

- Output Gate: Finally, we perform the following computation to decide what to output (h_t) from the current cell state. Specifically, we perform the following computation

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

  Here, the sigmoid controls the parts of the cell state that the cell outputs.

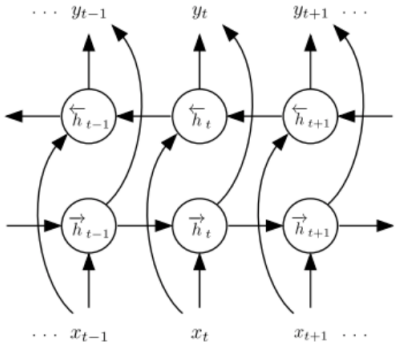
3.1 Backpropagation in LSTM



With this gating mechanism, it can be seen that while backpropagating gradients, repeated multiplication of a matrix W is avoided. Further, different values of the forget gate activations at each time step redress the problem of exploding/vanishing gradients. A detailed working out of this can be found at [2].

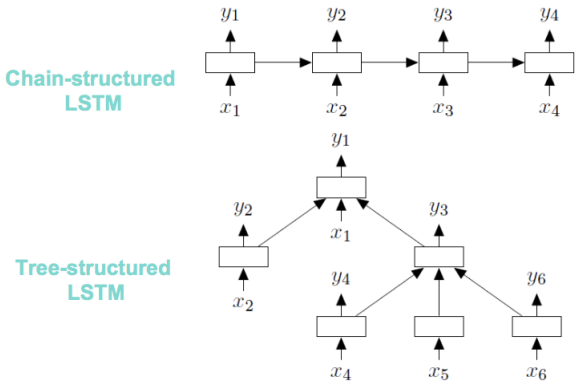
4 Other RNN variants

4.1 Bi-directional RNNs [5]



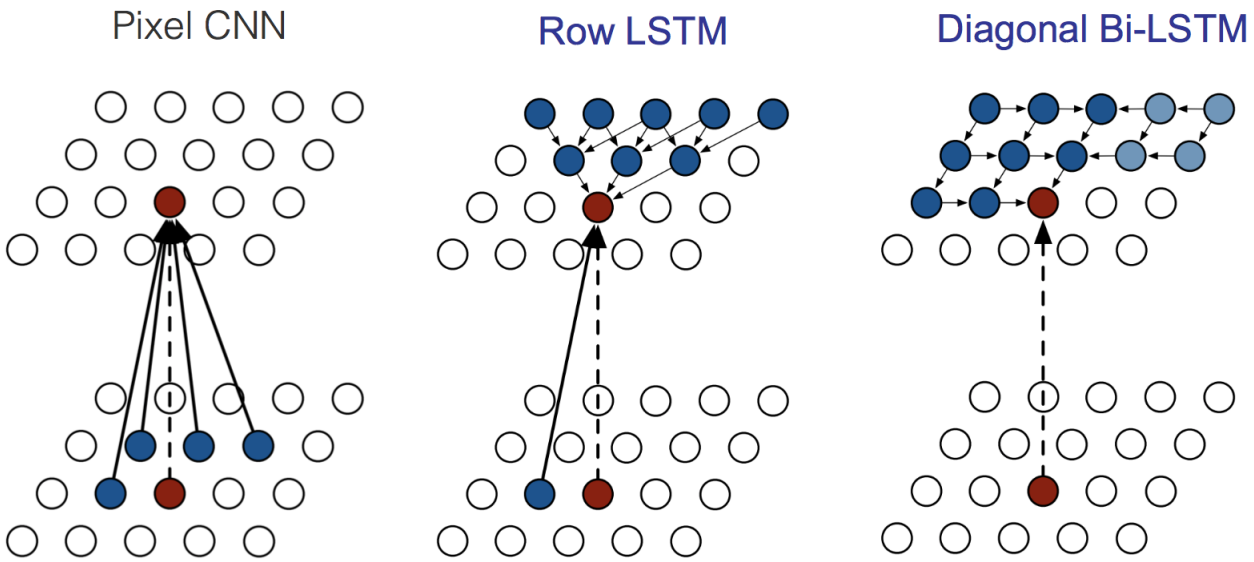
- In bi-directional RNNs, the hidden state is the concatenation of both forward and backward hidden states
- This allows the hidden state to capture both past and future information

4.2 Tree-structured RNNs [19]



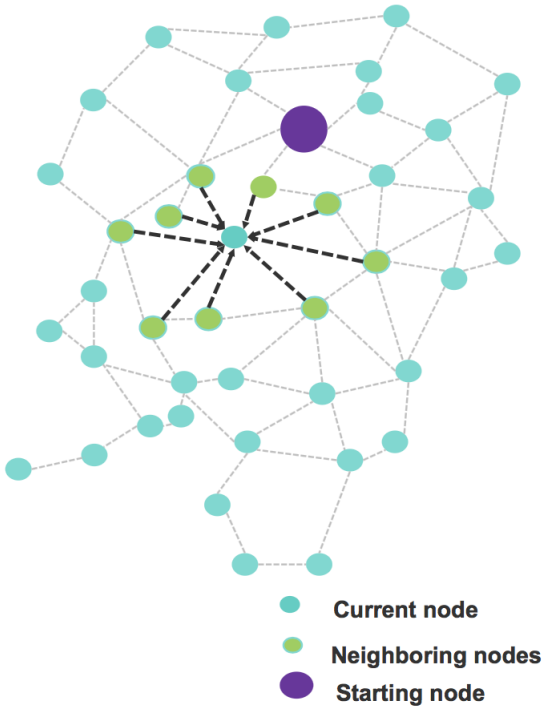
- In tree-structured RNNs, the hidden states condition on both an input vector and the hidden states of arbitrarily many child units
- Thus, standard LSTMs are in fact special cases of tree-LSTMs where each internal node has exactly one child.

4.3 RNNs for 2-D sequences [13]



- Instead of using CNNs for extracting features in 2-D data, we can use LSTMs over the rows/diagonals of the data as shown above to extract quality features

4.4 RNNs for Graph structures [11]



- We can also have an LSTM operate on graph-structured data e.g. in image segmentation applications, we can model the pixels to be nodes in a graph where the features extracted at each node also depend on the features extracted for neighbouring nodes in the graph

5 Attention

Attention mechanisms are techniques used to choose which feature to focus on, in the input data. They have improve performance in tasks such as machine translation, image captioning and speech recognition, to name a few.

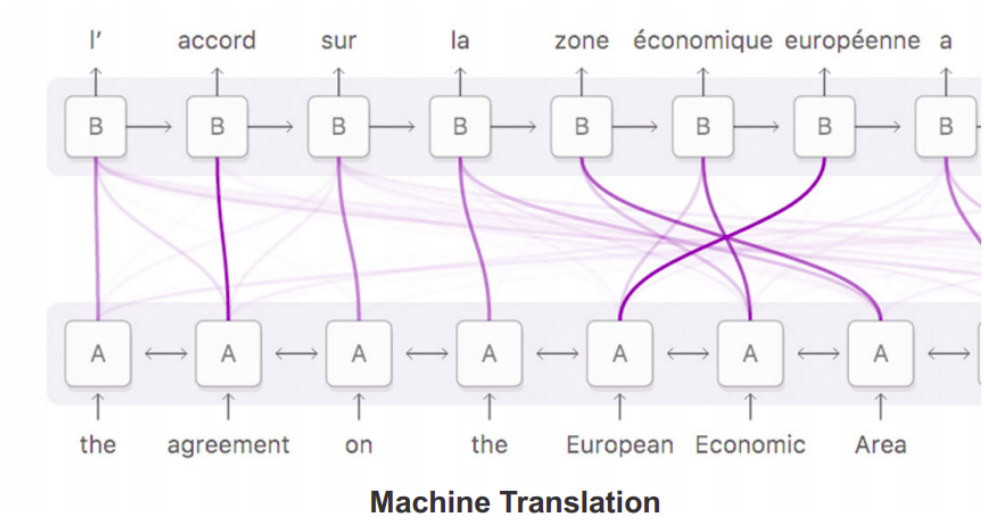


Figure 1: Example of attention used in machine translation

They are used primarily to handle for long-range dependencies and hence dealing with vanishing gradients, a problem typically seen in RNNs. By allowing for fine-grained localized representations of portions of data, like patches in images or words in sentences, attention improves model interpretability.

5.1 Attention Computation

Attention can be computed for a machine translation task using the following procedure:

- Encode each token in the input sentence into a key vector. This corresponds to what is available in the input
- When decoding, compare the query vector with the encoder states, and generate alignment scores corresponding to each key vector. Query vector corresponds to what is required to be generated, depending on what tokens have already been generated.
- Compute the attention weights by normalizing the alignment scores.
- Treat the encoder states as value vectors and compute a weighted sum, using the attention weights.
- Use this weighted sum in the decoder to generate the next token.

5.2 Attention Variants

There are a number of different alignment score functions used to generate alignment scores. Some of these are shown in the table below:

Soft and hard attention are variants of attention that respectively use deterministic and stochastic methods in computing the weights for each token. The computation described above is for soft attention. Instead of

Name	Alignment score function	Citation
Content-base attention	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \text{cosine}[\boldsymbol{s}_t, \boldsymbol{h}_i]$	Graves2014
Additive(*)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a[\boldsymbol{s}_t; \boldsymbol{h}_i])$	Bahdanau2015
Location-Base	$\alpha_{t,i} = \text{softmax}(\mathbf{W}_a \boldsymbol{s}_t)$ Note: This simplifies the softmax alignment to only depend on the target position.	Luong2015
General	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \mathbf{W}_a \boldsymbol{h}_i$ where $\mathbf{W}_a$ is a trainable weight matrix in the attention layer.	Luong2015
Dot-Product	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \boldsymbol{h}_i$	Luong2015
Scaled Dot-Product(^)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \frac{\boldsymbol{s}_t^\top \boldsymbol{h}_i}{\sqrt{n}}$ Note: very similar to the dot-product attention except for a scaling factor; where n is the dimension of the source hidden state.	Vaswani2017

Figure 2: Popular alignment score functions

using the attention weights to compute a weighted average, hard attention uses these as probabilities and samples from the corresponding features using this distribution. A comparison for attention used on images can be illustrated below. Notice how soft attention can be diffuse, and assign nonzero weights to significant weights to large portions of the image at times, while hard attention focuses on a particular equally-sized part of the image in each case. Soft attention is presently the more popular variant, primarily because it allows for simpler backpropagation in the network.

5.3 Applications in Computer Vision

Attention are widely used in computer vision tasks such as Image captioning, Image paragraph generation, etc. [10] introduce a technique using attention over both image and text. The entire pipeline can be seen in Fig. 3, and proceeds as follows:

- The image is first segmented into semantic regions, and each region is captioned with local phrases.
- Attention is applied on both visual regions and the text phrases and the resulting features fed to a generator which generates sentences using hierarchical text generation.
- These sentences are then fed to a sentence discriminator and a recurrent topic-transition discriminator for assessing sentence plausibility and topic coherence respectively
- A paragraph description corpus is adopted to provide linguistic knowledge about paragraph generation, which depicts the true data distribution of the discriminators

Another application is Image captioning. [21] introduce a technique, described as follows:

- The entire image is given as input to a Convolutional Neural Network (CNN).
- Multiple feature maps are used to extract visual features from the images.
- RNNs are used to attend to the images. Both hard and soft attention is used. Hard attention samples regions directly from the image. Soft attention computes the expected attention over the entire image.
- LSTM use the attention weights to generate captions word by word.

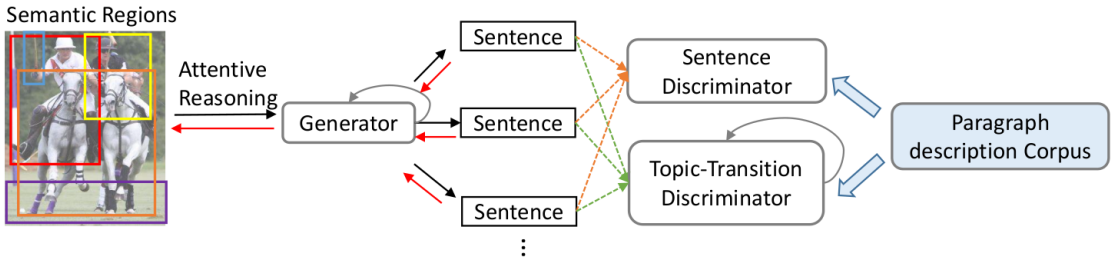


Figure 3: Pipeline for Image Paragraph Generation

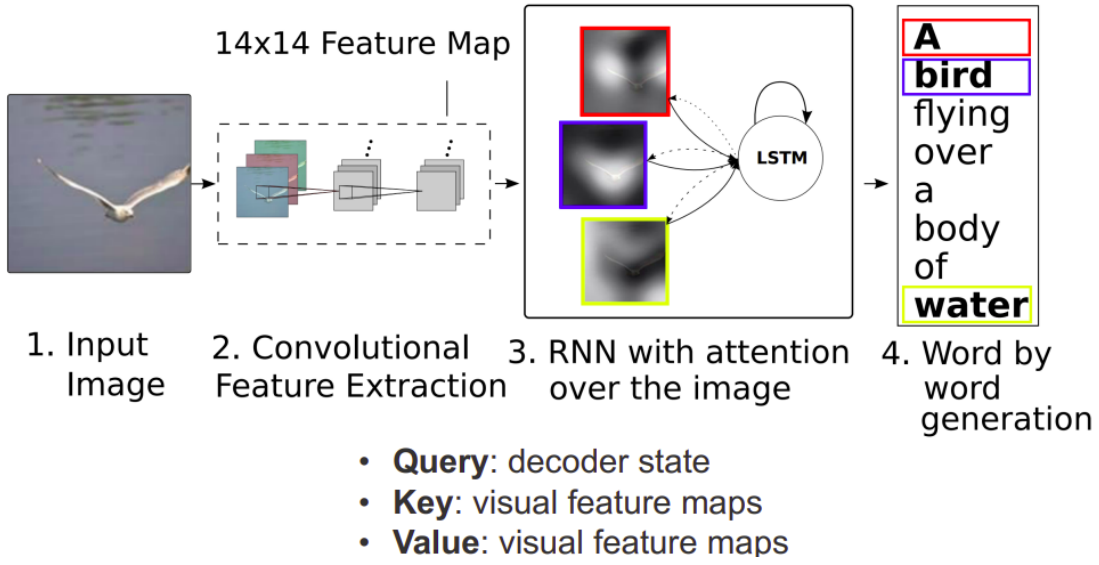


Figure 4: Pipeline for Image captioning

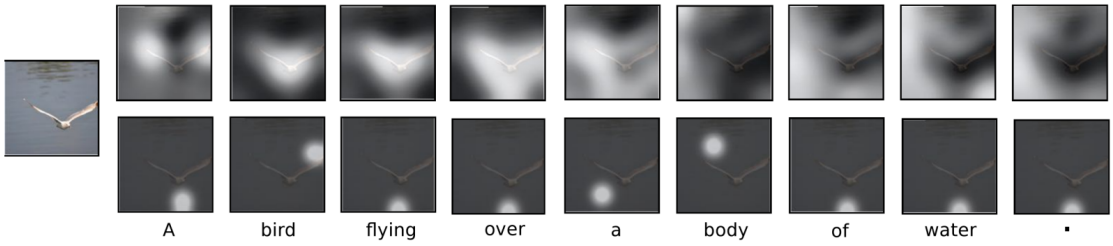


Figure 5: Above: Soft Attention, Below: Hard Attention in Image Captioning

6 Transformers: Multi-Headed Attention [20]

Recently, [20] introduced a novel, non-recurrent neural network architecture composed purely of self-attention called the Transformer. The Transformer has attained state-of-the-art results in many sequence-to-sequence natural language processing tasks, such as machine translation. Since the Transformer architecture lacks recurrent networks, it can be more amenable to learning long-range dependencies over sequences while also

improving training and inference speed.

As shown below, the Transformer employs multi-headed self-attention, in which multiple attention layers run in parallel. Intuitively, this can enable different heads to focus on different parts of the sequence.

Formally, multiple heads of Queries Q, Keys K of dimension d_k, and Values V can be packed together into separate matrices to allow for attention to be computed efficiently using the scaled dot-product variant, which they suggest prevents diminished gradients during training:

$$\text{Attention } (Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right) V \tag{1}$$

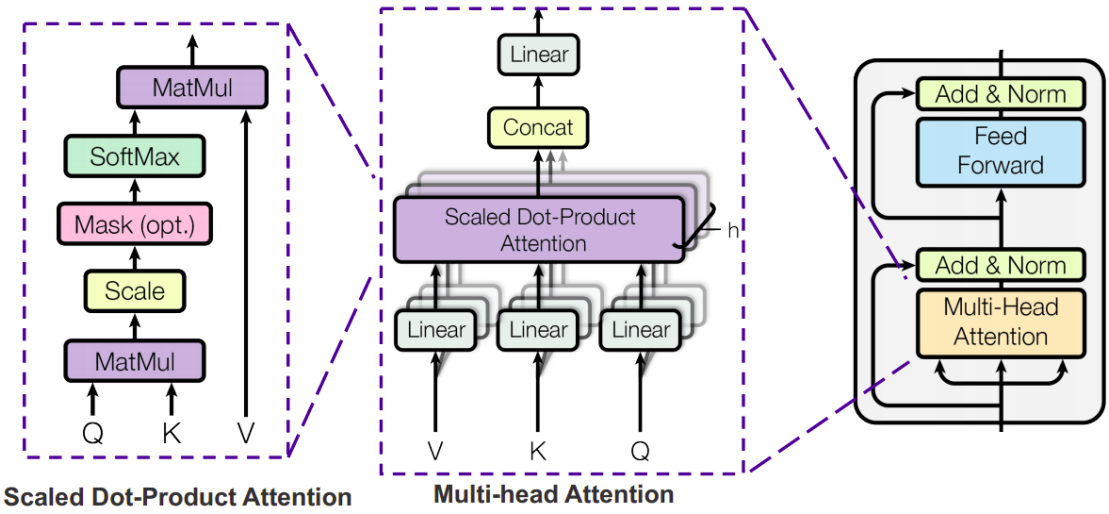


Figure 6: Multi headed attention

Multi-headed attention can then jointly attend to information from multiple different representations at different positions by:

$$\begin{aligned} \text{MultiHead } (Q, K, V) &= \text{ Concat } (\text{head}, , \ldots, \text{ head }_h) W^O \\ \text{where } head_i &= \text{Attention } \left(QW_i^Q, KW_i^K, VW_i^V\right) \end{aligned} \tag{2}$$

6.1 Unsupervised Language Model Pre-training

Recently, language representation via unsupervised language model pre-training has revolutionized the field of natural language processing. Parameters learned during the training of large language models using self-supervision have been shown to be extremely effective when transferred to other NLP prediction tasks. [16] [15] [4] [17]. Since language modeling requires the resolving of long-term dependencies, hierarchical relations, and sentiment, it can be seen as an ideal source task for transfer learning in NLP. [8]

ELMo [15] introduce deep contextualized word representations, which are learned functions of the internal states of a deep bidirectional LSTM language model trained to predict both the next word in a sentence given its history and the previous word in a sentence given its future words. These contextualized representations can then be frozen and used as embeddings for other downstream tasks like question answering, textual

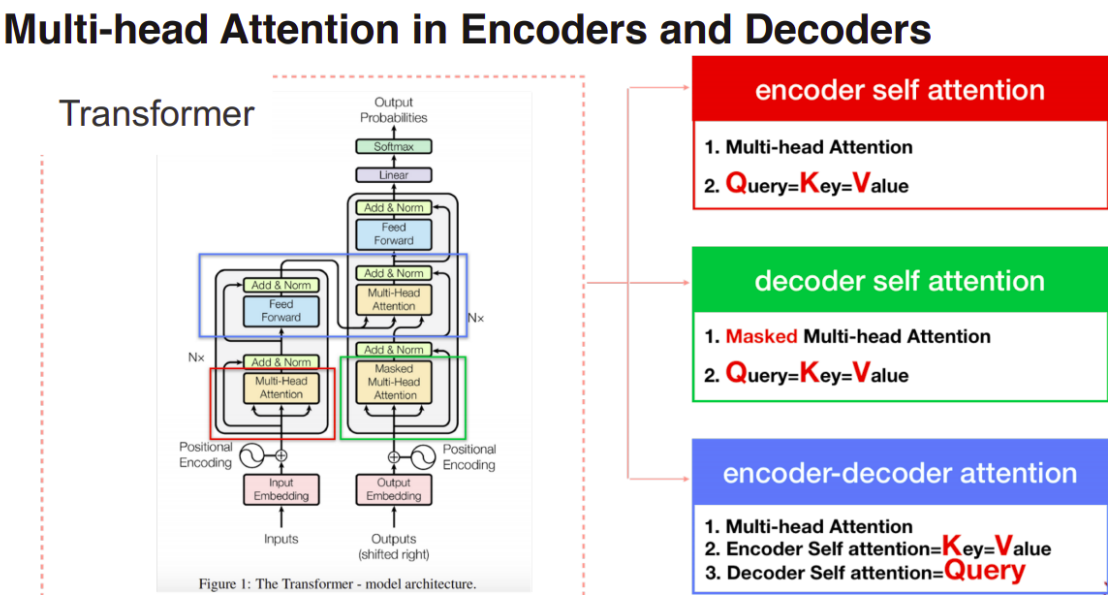


Figure 7: Pipe Line for Multi headed attention in encoders and decoders

entailment, and sentiment analysis. Instead of just transferring word embeddings for a new task, Howard and Ruder's (2018) ULMFiT aims to transfer the language model itself for new tasks. Particularly, the authors train an AWD-LSTM [12] language model on 103 million words of Wikipedia data, fine-tune on a smaller amount of task-specific data using different learning rates for different layers of the model, and add a final classifier on the end of the network for the target task. Thus, while ELMo requires task-specific architectures when transferring to new tasks, ULMFiT simply adds a classifier on top of the language model to obtain state-of-the-art results on six benchmarks. OpenAI then adapted this method (dubbed Generative Pre-training or GPT) to work with the popular Transformer [20] architecture, an auxiliary language modeling loss during fine-tuning to obtain even better results, and adaptation to more difficult tasks such as machine translation. [16] Just this year, OpenAI followed up their work with GPT2, which is the highest performing language model to date. They use a similar but much higher capacity model to GPT as they find that capacity improves performance log-linearly. Due to its high, human-like language generation performance, they controversially decided to not release their largest model: a 1.5B parameter Transformer trained on 8 million documents of web text.

7 BERT: Pre-trained Text Representation Model

In BERT (Bidirectional Encoder Representations from Transformers), Devlin et al. (2018) [4] used a bidirectional Transformer architecture to obtain improved contextualized word embeddings in an extension to OpenAI GPT [16]. Before BERT, there were some conventional word embedding methods like Word2Vec and Glove, which had a pretrained matrix with each row denoting the embedding vector vector of a particular word. The matrix was pretrained with a large corpus of text using the above mentioned methods.

The paper introduces two new objectives to adapt the traditional task of predicting the next word in language modeling to benefit from bidirectionality. After encoding each word in a given sentence into a contextualized representation, they have the model both predict a random masked word from the original sentence and perform a binary classification on two sentences to identify whether or not one sentence follows the other.

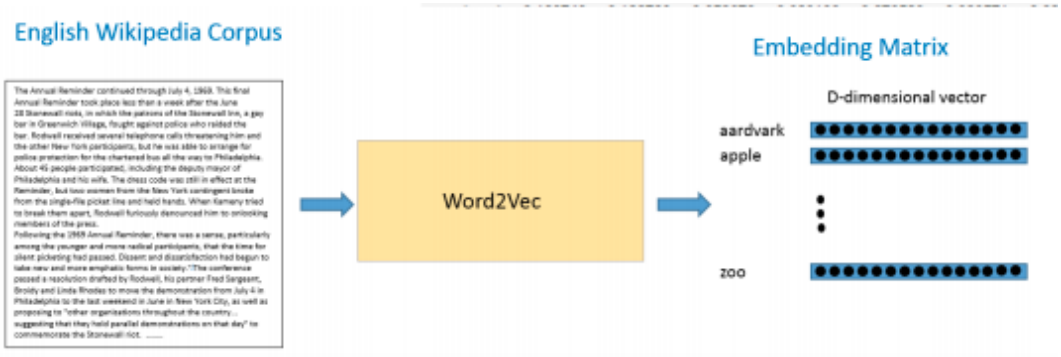


Figure 8: Word2Vector

Although the masked language model objective requires more pre-training steps since each prediction is no longer sequential, they find that performance increases over the traditional objective are immediate. They find that the next sentence classification objective is particularly beneficial to tasks like natural language inference and question answering since they require multi-sentence reasoning.

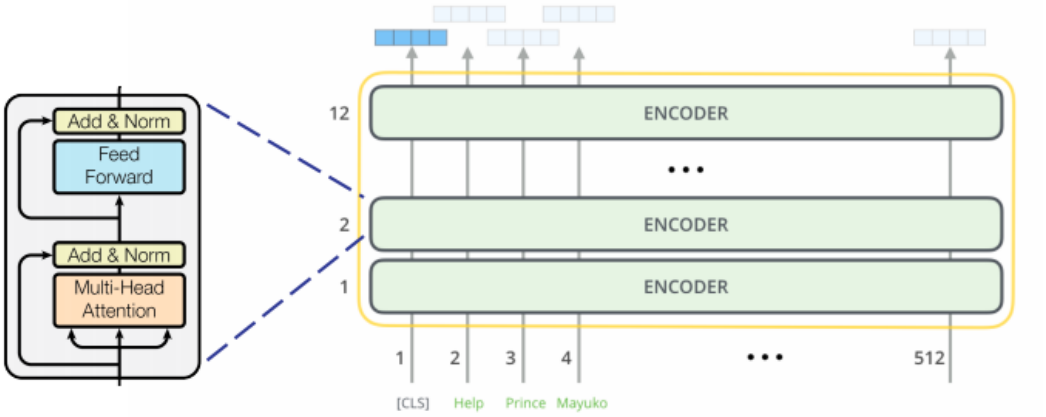


Figure 9: The BERT encoder

In the paper, the authors use both Wikipedia data (2.5B words) and free online eBook data (800M words) for training a Transformer encoder with hundreds of millions of parameters. An ablation study on model size empirically shows that extreme model sizes lead to large improvements on even very small scale tasks, provided that the model has been sufficiently pre-trained. Although training takes many more steps to converge than a traditional language model objective, BERT, with only single output layer modifications, performs at the state-of-the-art for eleven NLP tasks including sentiment, question answering, and natural language inference. The authors report that they were able to train BERT in just 4 days on 4 Google TPU pods (256 TPU Chips). In comparison to this, it will take 5.3 days to train it on a cluster having 64 V100 GPU and 99 days on a standard 4 GPU desktop with an RTX 2080Ti.

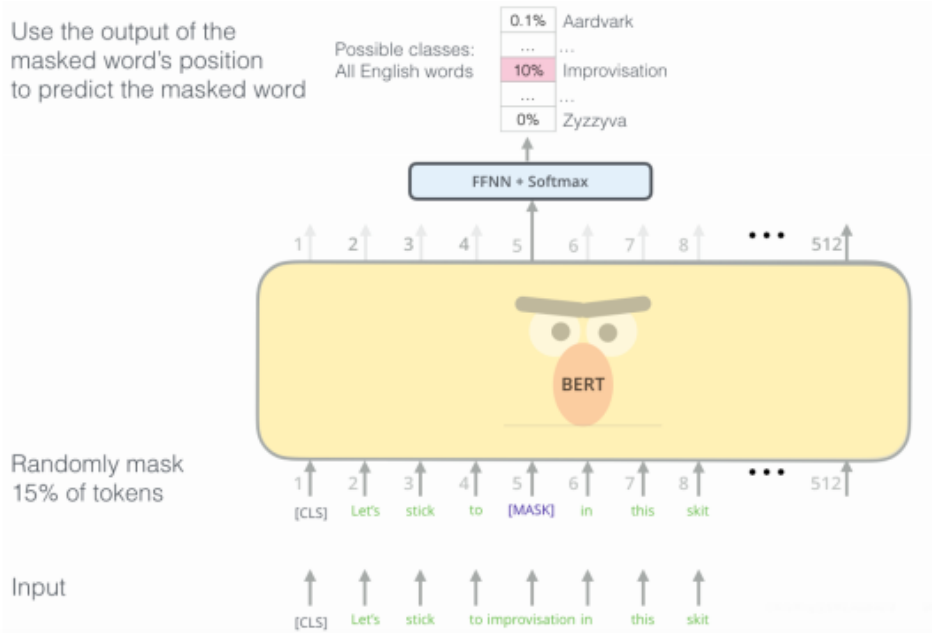


Figure 10: The masked language model objective

References

[1] G.E. Hinton. A. Krizhevsky, I. Sutskever. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems 25, pp. 1097–1105*, 2012.

[2] Nir Arbel. How lstm networks solve the problem of vanishing gradients. `https://medium.com/datadriveninvestor/how-do-lstm-networks-solve-the-problem-of-vanishing-gradients-a6784971a577`, Feb 2020.

[3] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[5] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. IEEE, 2013.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[7] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[8] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. *arXiv preprint arXiv:1801.06146*, 2018.

[9] A. Zisserman K. Simonyan. Very deep convolutional networks for large-scale image recognition. 2014.

[10] Xiaodan Liang, Zhiting Hu, Hao Zhang, Chuang Gan, and Eric P Xing. Recurrent topic-transition gan for visual paragraph generation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3362–3371, 2017.

[11] Xiaodan Liang, Xiaohui Shen, Jiashi Feng, Liang Lin, and Shuicheng Yan. Semantic object parsing with graph lstm. In *European Conference on Computer Vision*, pages 125–143. Springer, 2016.

[12] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. Regularizing and optimizing lstm language models. *arXiv preprint arXiv:1708.02182*, 2017.

[13] Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. *arXiv preprint arXiv:1601.06759*, 2016.

[14] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318, 2013.

[15] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.

[16] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. *URL https://s3-us-west-2. amazonaws. com/openai-assets/researchcovers/languageunsupervised/language understanding paper. pdf*, 2018.

[17] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9, 2019.

[18] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[19] Kai Sheng Tai, Richard Socher, and Christopher D Manning. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*, 2015.

[20] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[21] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057, 2015.

10-708: Probabilistic Graphical Models, Spring 2020

15:Case Study of Deep Generative Models: Text Generation

Lecturer: Eric P. Xing *Scribe: hel1, mingjie1, yukaih2, wenhaod*

1 Text Generation Tasks and Center Goals

The goal of text generation is to generate natural language from input data or machine representations. It spans a broad set of natural language processing (NLP) tasks. These tasks includes dialog system, machine translation, summarization, description generation, captioning and speech recognition.

There are basically two center goals in all these tasks. The first center goal is generating human-like, grammatical, and readable text, or so called natural language. The second goal is generating text that contains desired information inferred from inputs. For example, in machine translation the generated target sentence should has the same meaning with the source input sentence. In data description tasks, the generated report should describe the input data table. And for other tasks like attribute control the generated sentence should also contain the same kind of information as input like generating "I like this restaurant" for a positive sentiment input. Same goals is also in conversation control that we should control conversation strategy and topic to the specified input.

2 Common Model for Text Generation

We will first take a look at the first center goal: how to generate natural language.

2.1 Language Model

One of the most basic model for text generation tasks is language model. In language model, the sentence $\mathbf{y}$ is modeled as a series of tokens y_t, and the probability of the sentence $p_\theta(\mathbf{y})$ is computed by decomposing the probability of the whole sentence across times steps, or tokens. And the probability of each token y_t is modeled by a conditioned probability to its previous sequence$\mathbf{y}_{1:t-1}$. The expression is as below:

$$\mathbf{y} = y(y_1, y_2, ..., y_T) \tag{1}$$

$$p_\theta(\mathbf{y}) = \prod_t p_\theta(y_t|\mathbf{y}_{1:t-1}) \tag{2}$$

To implement this model, we can use recurrent neural networks like LSTM.

2.2 Conditional Language Model

In conditional language model, we can specify the context x to incorporate the input data . For example, in machine translation, x is the input sentence. During the inference, the probability is conditioned on input

x. Expression is as below:

$$\mathbf{y} = y(y_1, y_2, ..., y_T) \tag{3}$$

$$p_\theta(\mathbf{y}|\mathbf{x}) = \prod_t p_\theta(y_t|\mathbf{y}_{1:t-1}, \mathbf{x}) \tag{4}$$

3 Common Learning Algorithm

3.1 Maximum Likelihood Estimation (MLE)

One of the most popular and simplest training method is Maximum Likelihood Estimation (MLE). In MLE, we will maximize the data log-likelihood $L(\theta)$ from the given ground truth data $\mathbf{y}^*$. The expression for training is as below:

$$\mathbf{y}^* = (y_1^*, y_2^*, ..., y_T^*) \tag{5}$$

$$\theta = \arg\max_\theta L(\theta) = \arg\max_\theta p_\theta(\mathbf{y}^*|x) = \arg\max_\theta (\log \prod_t p_\theta(y_t^*|\mathbf{y}_{1:t-1}^*, \mathbf{x})) \tag{6}$$

For the evaluation, we have different metrics for different tasks to evaluate similarity between the output and ground truth. For example, BLEU is used for machine translation and ROUGE is used for summarization.

3.1.1 Two Issues of MLE

There are two issues of MLE. The first issue is called exposure bias [Ranzato et al., 2015]. In the training process, the next token is predicted given the previous ground-truth sequence: $(y_t^*|\mathbf{y}_{1:t-1}^*, \mathbf{x})$. But in evaluation process, the ground truth is not exposed to the model. The next token is predicted given previous sequence that are generated by the model it self: $(\hat{y}_t|\hat{\mathbf{y}}_{1:t-1}, \mathbf{x})$. So if there is an error in the model, the error will affect subsequent tokens. The second issue is the mismatch between training and evaluation criteria. As described in the training and evaluation process, we use log-likelihood for training metrics but use BLUE for evaluation in machine translation.

3.1.2 Possible Solutions

Reinforcement learning [e.g., Ranzato et al., 2015] is one of the solutions to deal with the mismatch between training and evaluation criteria. In reinforcement learning, a reward function $R(\mathbf{y}, \mathbf{y}^*)$ is defined on sequence $\mathbf{y}$ and ground truth $\mathbf{y}^*$. And the training process is to maximize the expected reward. For example, if we use BLEU as reward function R for translation tasks, then we will have the same metrics for training and evaluation. Expression is as below:

$$\max_\theta \mathbb{E}_{p_\theta(\mathbf{y})}[R(\mathbf{y}, \mathbf{y}^*)] \tag{7}$$

But there are also problems for reinforcement learning. For example, the sequence space is extremely large (50000^{50} for a sentence with length of 50 and vocabulary of 50000). So there will be high variance and poor exploration efficiency during training process. This means most of the time, the model may generate something not very meaningful.

There are also many recent works to make training more practical. For example, Reward Augmented Maximum Likelihood (RAML) [Norouzi et al.,16] adds reward-aware perturbation to the MLE data examples to close the gap between training and evaluation criteria. RAML can also let the model see mistakes in the data rather than just ground truth data. In Softmax Policy Gradient (SPG) [Ding & Soricut, 17], reward

distribution is used for effective sampling and estimating policy gradient. There are also other algorithms like Data noising [Xie et al.,17] that adds random noise to data so that model is exposed to noise and mistake during training time.

All these algorithms are special instances of a generalized entropy regularized policy optimization (ERPO) framework. The differences are in the choice of rewards and the values of hyperparameters α and β.

4 Generalized Entropy Regularized Policy Optimization (ERPO)

4.1 General Framework

The objective of the generalized ERPO is:

$$L(q, \theta) = \mathbb{E}_q[R(\mathbf{y}|\mathbf{y}^*)] - \alpha KL(q(\mathbf{y}|\mathbf{x})||p_\theta(\mathbf{y}|\mathbf{x})) + \beta H(q) \tag{8}$$

where $p_\theta(\mathbf{y}|\mathbf{x})$ is the sequence generation model, $R(\mathbf{y}|\mathbf{y}^*)$ is the reward function and $q(\mathbf{y}|\mathbf{x})$ is the variational distribution. Therefore, in ERPO, the objective is to maximize the reward function under q, minimize the KL divergence between q and the model $p\theta$, and regularized by the entropy of q.

The objective can be solved with an EM-style procedure.

In the E-step:

$$q^{n+1}(\mathbf{y}|\mathbf{x}) \propto \exp(\frac{\alpha \log p_\theta^n(\mathbf{y}|\mathbf{x}) + R(\mathbf{y}|\mathbf{y}^*)}{\alpha + \beta}) \tag{9}$$

In the M-step:

$$\theta^{n+1} = \text{argmax}_\theta \mathbb{E}_{q^{n+1}}[\log p_\theta(\mathbf{y}|\mathbf{x})] \tag{10}$$

α and β have implications for the algorithm. As $\alpha \to \infty$, $q^{n+1} = p_\theta^n$ and the objective corresponds to minimizing KL divergence. As $\beta \to \infty$, q^{n+1} becomes a uniform distribution and the objective corresponds to maximizing the entropy of q.

4.2 MLE

MLE can be interpreted as a ERPO. Specifically, if the reward function is:

$$R = R_\delta(\mathbf{y}|\mathbf{y}^*) := \begin{cases} 1 & \text{if} \quad \mathbf{y} = \mathbf{y}^* \\ -\infty & \text{otherwise} \end{cases}$$

and $\alpha \to \infty$, $\beta = 1$.

Then the E-step becomes:

$$q(\mathbf{y}|\mathbf{x}) = \begin{cases} 1 & \text{if} \quad \mathbf{y} = \mathbf{y}^* \\ 0 & \text{otherwise} \end{cases}$$

and the M-step becomes:

$$\theta^{n+1} = \text{argmax}_\theta \log p_\theta(\mathbf{y}^*|\mathbf{x})$$

Therefore, MLE can be viewed as a policy optimization with a δ-function reward. In this formulation, any exploration beyond the training data will not be exposed to the model and this will result in exposure bias, which is one of the drawback of MLE. On the other hand, due to the restriction, q becomes the empirical distribution, so the implementation of the algorithm is very simple and efficient

4.3 RAML

In RAML [Norouzi et al., 16], the reward function can be a common reward such as the BLEU$(\mathbf{y}|\mathbf{y}^*)$. $\alpha \to 0$ and $\beta = 1$.

In this scenario, the E-step becomes:

$$q(\mathbf{y}|\mathbf{x}) \propto \exp(R(\mathbf{y}, \mathbf{y}^*))$$

and the M-step becomes:

$$\max_\theta \mathbb{E}_q[\log p_\theta(\mathbf{y}|\mathbf{x})]$$

Compared to MLE, RAML use a smoother reward function and it can be exposed to a larger exploration space.

4.4 SPG

Softmax Policy Gradient (SPG) [Ding & Soricut, 17] is another special case of ERPO. The E-step and M-step are the same as previous two methods, the reward function could be a common reward such as $BLEU(\mathbf{y}, \mathbf{y}^*)$. The only difference is that in SPG we set $\alpha = 1$ and $\beta = 0$, which is contrary to the MLE and RAML methods. The final updating equation is:

$$\text{E-step:} \quad q(\mathbf{y}|\mathbf{x}) \propto p_\theta(\mathbf{y}|\mathbf{x}) \exp(R(\mathbf{y}, \mathbf{y}^*))$$

$$\text{M-step:} \quad \max_\theta E_q \left[\log p_\theta(\mathbf{y}|\mathbf{x})\right]$$

SPG uses both the model distribution and the reward for exploration, therefore leading to the largest exploration space. The advantage is that the chance to find the optimal solution is larger, while the disadvantage is that learning difficulty is highly increased. Some training tricks are needed to resolve this difficulty.

4.5 Data Noising

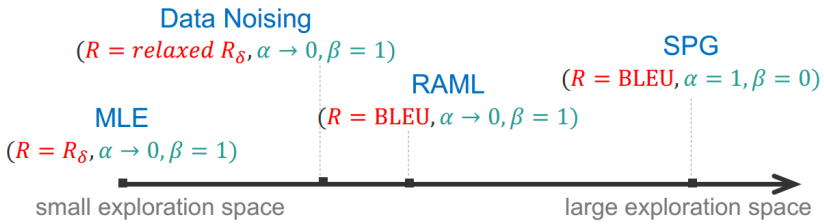
Data Noising is also a special case of ERPO with $\alpha \to 0$ and $\beta = 1$. The difference is that the reward function is a locally relaxed variant of $R_\delta(\mathbf{y}|\mathbf{y}^*)$. For example, we can use this reward function:

$$R'_\delta(\mathbf{y}|\mathbf{y}^*) := \begin{cases} 1 & \text{if} diff(\mathbf{y}, \mathbf{y}^*) = 1 \\ -\infty & \text{otherwise} \end{cases} \tag{11}$$

To implement this reward function, we can randomly replace a single token with another uniformly picked token, which is much easier than adding reward-aware noise in RAML.

5 Interpolation algorithm

After introducing previous four algorithms, we can summarize them in a hyperparameter space. the exploration space can be used to compare these methods, and the relation is shown below:



Every algorithm corresponds to a point in the hyperparameter space and the position is defined by the selection of reward function, α, and β. From left to right, the exploration space increases. There exists a trade-off here: when the exploration space is large, we will have better test performance in theory, but more difficult for training.

An intuitive idea to balance this trade-off is interpolating among the algorithms. We can start from MLE hyperparameter values, and then gradually anneal to the SPG hyperparameter values. This procedure will make the training easy at the beginning and gradually increase the exploration space later.

6 Unsupervised Controlled Generation of Text

Recall that we have two central goals in text generation tasks. In previous sections, we have introduced how to generate human-like, grammatical, and readable text. Next, we will focus on generating text that contains desired information inferred from inputs.

Usually, we need lots of supervision data to achieve the second goal. Though tasks such as machine translation and data description can easily acquire millions of data, for some tasks, we cannot access to these amounts of data. For example, in attribute control task (i.e. modify sentiment from positive to negative) and conversation control task (i.e. control conversation strategy and topic), there is nearly no existing supervision data. Therefore, we have to consider unsupervised controlled generation method.

There are two kinds of generative level in general: sentence-level control and conversation-level control. Two representative works of the sentence-level control are text attribute transfer and text content manipulation, and target-guided open-domain conversation is a typically conversation-level control. In the following part, we will mainly introduce these three tasks.

6.1 Text Attribute Transfer

The task is that, given a sentence, we want to modify this sentence to have a desired attribute value while keeping all other aspects unchanged. Let's say we want to transfer sentiment from negative to positive. Here is an input sentence for example:

"It was super dry and had a weird taste to the entire slice."

we want to modify the sentence to have a positive sentiment like:

"It was super fresh and had a delicious taste to the entire slice."

The goal here is we want to change the content from "dry" to "fresh" and from "weird" to "delicious", but still keep all other aspects unchanged. The application of this kind of tasks can be like the personalized article writing, conversation systems, and authorship obfuscation.

With formal formulation, given an input sentence x and the original attributes a_x. For the target, we have target sentence y and target attribute a_y. Now we want to generate a target sentence y, where y has the desired attributes a_y and keeps all attribute-independent properties of x:

$$Task : (x, a_y) \rightarrow y$$

In the training setting, we only have (x, a_x), but no ((x, a_x), (y, a_y)) for training. So the next is how we can use this data available to train a particular conditional generation model in this case.

The solution is that we design an encoder, which encodes the input sentence as a feature vector z. And we concatenate this feature vector with the attribute, where the value is a_y for the positive sentiment or 0 for the negative sentiment. Then we feed the concatenated input into a decoder to generate y, where the model can be represented as:

$$p_\theta(y|x, a_y)$$

The key intuition for learning is we decompose the task into competitive sub-objectives and use direct supervision for each of the sub-objectives.

The first objective is we want y to keep all attribute-independent properties of x, which means y must be fairly close to the input of x. So a very straightforward and simple way is to enforce the similarity between x and y, where we just use the auto-encoding loss:

$$(x, a_x) \rightarrow x$$

The second objective is we want y has the desired attributes a_y, where we use a classification loss given the produced y sample to a pre-trained sentiment classifier f:

$$\hat{y} \sim p_\theta(y|x, a_y), f(\hat{y}) \rightarrow a_y$$

So there are basically two loss functions and we will optimize these two loss functions jointly.

6.2 Text Content Manipulation

The goal here is that we want to generate a sentence to describe the contents in a given data record like controlling the writing style by using the writing style of a reference sentence.

The method is pretty similar to the one in "Text Attribute Transfer", where we also decompose the task into competitive sub-objectives and use direct supervision for each of the sub-objectives.

6.3 Target-guided Open-domain Conversation

There are basically three sets of conversation systems.

The first set is the Task-oriented dialog, where we use this dialogue system to address the specific tasks like booking a flight or reserving a restaurant. However, the task-oriented dialog is a close domain conversation, which means the conversation system can only do a specific task, not any anything else.

The second set is Open-domain chit-chat conversation. The goal is to improve the user engagement, and the metric is how long this chat bot can keep it conversing with a human, where the conversation is random without any control of topics or other aspects.

The third set is Target-guided conversation, which combines previous two types of conversation. It is still an open-domain conversation that basically can converse with human about whatever topic, but it controls the conversation strategy to reach a desired topic at the end of the conversation at the same time.

There are two goals for target-guided open-domain conversation. The first goal is to reach a desired topic in the end of conversation starting from any topic. The second goal is to achieve a smooth transition in natural conversation.

The challenge is that we have no supervised data for the task. The solution to this problem is using competitive sub-objectives and partial supervision. More specifically, we use rich chit-chat data to learn smooth single-turn transition for generating natural conversation. We can also use rule-based multi-turn planning to reach desired target, that means use keyword of each response to guide the topic closer to the target at each step. The basic step is as follows. At first, we extract keyword from the input sentence. Then we use the learned kernel-based topic transition and target-guided rule to transit to keywords that are close in the word embedding space and make sure the next keywords must get closer to the target keyword. At last, keyword conditional response retrieval is done to generate the conversation.

10-708: Probabilistic Graphical Models, Spring 2020

16: Modeling networks: Gaussian graphical models and Ising models

Lecturer: Eric P. Xing *Scribe: Yu-Hsuan Wang, Ta Chung Chi, Ayush Raina, Yijia Sun*

1 Structure learning

1.1 Introduction

In previous lectures, we assume we have enough domain knowledge to design the structure of our models which could be illustrated with the following examples:

1. We know there are multiple clusters in the data and we know each cluster could be represented with a Gaussian distribution ⇒ We use Gaussian mixture model to model our data.
2. We know the relation between different time steps of data and time step t provides sufficient information for us to model time step $t+1$ ⇒ We use hidden Markov model to model our data.
3. We know the dependency between concepts and encode the dependency with a knowledge graph or expert system like Figure 1.

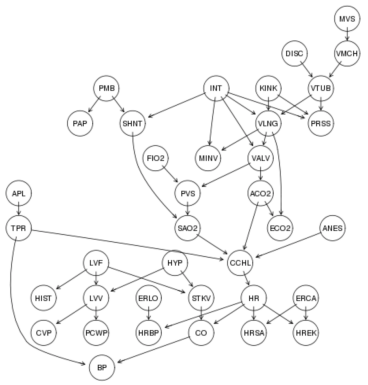


Figure 1

In this lecture, we are doing the other way around which is called *structure learning*. We are going to learn the dependency, correlations or the causal structures of entities from our data. There are different categories of structure learning:

- Tree v.s. Non-Tree: The learned structure could be constrained to be a tree or not. If we constrain the learned structure to be a tree, there would be a nice mathematical guarantee that we could achieve it in polynomial time [1]. On the other hand, without such constraint the problem becomes NP-hard while there are many heuristics and approximations could be applied to solve it.

- Directed v.s. Undirected: The learned structures could also be categorized by whether they are directed or undirected. If the learned structure is a directed structure, we could apply causal discovery approach to solve it. The approach would be introduced in the following lectures. On the other hand, in this lecture, we are going to introduce how to learn an undirected structure in the following sections.

1.2 Network learning

There are many problems that require undirected network learning. For example, the dependency between different users in a social network or the dependency between web pages on the internet. However, the problem could be more complicated with the fact that the network is dynamic. For example, Figure 2 shows the relations between members of congress over time. The dynamic nature increase the difficulty of learning.

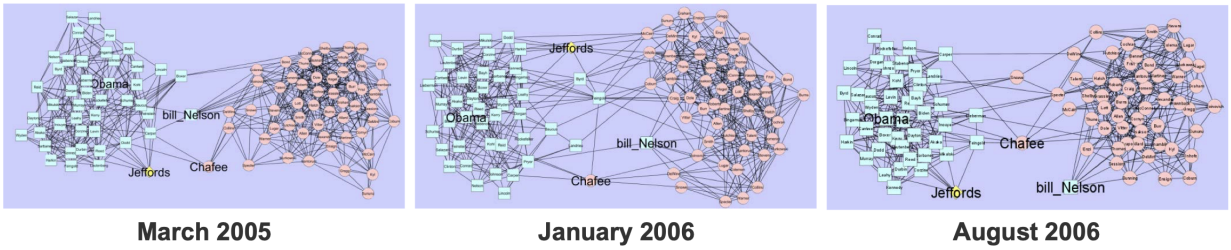


Figure 2

In the good old days, structure learning is ad hoc. For example, when we are trying to capture the relations between figures in the bible, we would use the co-occurrence of their names within a sentence. However, we could also use the co-occurrence within passage or different window and all of them would give us very different results. As a result, we need a more standarized approach to perform structure learning. In the following sections, we are going to introduce these approaches.

2 Network inference as parameter estimation

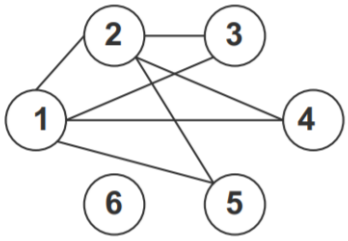


Figure 3

Markov random field is used to model data set with discrete, continuous, or heterogeneous nodal values by defining potential functions on cliques in a graph. Every pair-wise weight is attribute to an edge that connects the pair-wise random variables of interest. Intuitively, if the weight has zero value, there does not exist an edge that connect the pair-wise random variables. Consider a graph as shown in Figure 3 and the corresponding joint probability distribution in Eq.(1)

$$p(x_1, x_2, x_3, x_4) = \frac{1}{Z} \exp\{\theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4 + \theta_{12} x_1 x_2 + \theta_{13} x_1 x_3 + \theta_{23} x_2 x_3 + \theta_{34} x_3 x_4\} \quad (1)$$

We are able to obtain a parameter matrix which encodes the graph structure as shown in 4 The matrix

$$\begin{pmatrix} * & * & * & * & * & 0 \\ * & * & * & * & * & 0 \\ * & * & * & 0 & 0 & 0 \\ * & * & 0 & * & 0 & 0 \\ * & * & 0 & 0 & * & 0 \\ 0 & 0 & 0 & 0 & 0 & * \end{pmatrix}$$

Figure 4

creates a bridge between parameter learning and structure learning.

One example of Markov Random Filed model is a multivariate Gaussian with has a probability density function as shown in Eq.(2)

$$p(\mathbf{x}|\mu, \Sigma) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\{-\frac{1}{2}(\mathbf{x} - \mu)^T \Sigma^{-1} (\mathbf{x} - \mu)\} \quad (2)$$

Without loss of generality, we assume $\mu = 0, Q = \Sigma^{-1}$.

$$p(x_1, x_2, ...x_n|\mu = 0, Q) = \frac{|Q|^{1/2}}{(2\pi)^{n/2}} \exp\{-\frac{1}{2} \sum_i q_{ii}(x_i)^2 - \sum_{i<j} q_{ij} x_i x_j\} \quad (3)$$

The values in precision matrix Q corresponds to the weights of the singleton and pair-wise potentials in a continuous Markov Random Field. Therefore, Gaussian graphical model $\mathbf{X}^{(n)} \sim \mathcal{N}(\mathbf{0}, \Sigma^{(n)})$ is pair-wise *Markov Network* where the precision matrix carries structural information of the graph. The conditional independence and partial correlation coefficients are a more sophisticated dependence measure. For a given precision matrix Q, recall that

$$Q_{i,j} = 0 \implies X_i \perp\!\!\!\perp X_j | \mathbf{X}_{-ij}$$

or

$$p(X_i, X_j|\mathbf{X}_{-ij}) = p(X_i|\mathbf{X}_{-ij}) p(X_J|\mathbf{X}_{-ij})$$

We need to distinguish *Markov Network* from *Correlation network* which is based on the covariance matrix

$$\Sigma_{i,j} = 0 \implies X_i \perp\!\!\!\perp X_j, p(X_i, X_j) = p(X_i) p(X_j) \quad (4)$$

The former represents conditional independence among the variables, while the latter represents marginal independence. However, the conversion between correlation network and markov network is nontrivial ,i.e. sometimes the precision matrix Q cannot be obtained by just inverting the covariance matrix. Therefore, the goal is to bypass inverting the covariance matrix and to learn the preciion matrix directly. One common assumption is that the precision matrix is structurally sparse. The sparsity assumption makes empirical sense. For example, genes are only assumed to interface with small groups of other genes. The assumption also make statistical sense as learning is now feasible in high dimensions with small sample size.

2.1 Network learning with the LASSO

For a linear regression problem $y = \beta \mathbf{X} + \epsilon, \epsilon \sim \mathcal{N}(\mu, \sigma^2)$, LASSO regression has the form

$$\min_{\beta_0, \beta} \sum_{i=1}^{N} (y_i - \beta_0 - x_i^T \beta)^2 \text{ s.t. } \sum_{j=1}^{p} |\beta_j| \leq t \quad (5)$$

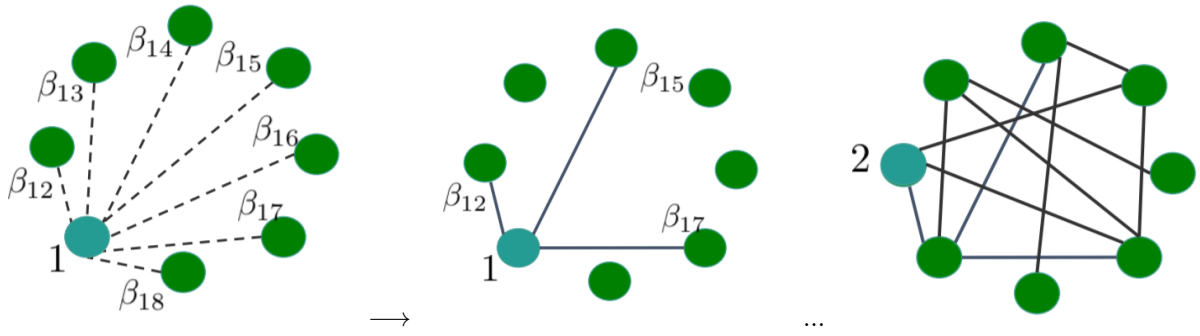
L1 regularization in the LASSO guarantees sparsity under the following assumptions

- Dependency condition: relevant covariates are not overly dependent
- Incoherence condition: large number of irrelevant covariates cannot be too correlated with relevant covariates
- Strong concentration bounds: sample quantities converge to expected values quickly

If we project linear regression to a graphical representation and perform LASSO regression of all nodes to a target node, LASSO can select the neighborhood of each node by solving

$$\hat{\beta}_i = \text{argmin}_{\beta_i} ||\mathbf{Y} - \mathbf{X}\beta_i||^2 + \lambda||\beta_i||_1$$

Repeat the LASSO regression for every node will form the total edge set



$$\hat{\varepsilon} = (u, v) : \max(|\hat{\beta_{uv}}|, |\hat{\beta_{vu}}|) > 0 \tag{6}$$

This algorithm is called the *Graphical LASSO* where LASSO is performed on every node with respect to all the other nodes in the whole collection. It is been proved in [2, 3] that Graphical LASSO has consistent structure recovery, i.e. this procedure has high probability of recovering the true structure of the graph. If $\lambda_s > C\sqrt{\frac{\log p}{S}}$, then with high probability $S(\hat{\beta}) \to S(\beta^*)$.

2.2 Graphical LASSO regression

Equation 2 for a bi-variate Gaussian distribution case can be written as follows:

$$p(\mathbf{x}|\mu, \Sigma) = \mathcal{N}(\begin{bmatrix}\mathbf{x_1}\\ \mathbf{x_2}\end{bmatrix} | \begin{bmatrix}\mu_1\\ \mu_2\end{bmatrix}, \begin{bmatrix}\Sigma_{11} & \Sigma_{12}\\ \Sigma_{21} & \Sigma_{22}\end{bmatrix}$$

The conditional probabilities hence can be written as:

$$p(\mathbf{x_2}) = \mathcal{N}(\mathbf{x_2}|m_2^m, V_2^m) \text{ where } m_2^m = \mu_2 \text{ and } V_2^m = \Sigma_{22} \tag{7}$$

$$p(\mathbf{x_1}|\mathbf{x_2}) = \mathcal{N}(\mathbf{x_1}|m_{1|2}, V_{1|2}) \text{ where } m_{1|2} = \mu_1\Sigma_{12}\Sigma_{22}^{-1}(\mathbf{x_2} - \mu_2) \text{ and } V_{1|2} = \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21} \tag{8}$$

Expanding the covariance and precision matrices:

$$\Sigma = \begin{bmatrix}\sigma_{11} & \sigma_1^T\\ \sigma_1 & \Sigma_{-1}\end{bmatrix}$$

Note that here σ_{11} is the variance of the first variable and Σ_{-1} is the covariance matrix for the rest $n-1$ variables.

The precision matrix can be represented as below following the matrix inversion lemma:

$$\mathcal{Q} = \Sigma^{-1} = \begin{bmatrix} q_{11} & -q_{11}\sigma_1^T\Sigma_{-1}^{-1} \\ -q_{11}\Sigma_{-1}^{-1}\sigma_1 & \Sigma_{-1}^{-1}\left(\mathbf{I} + q_{11}\sigma_1\sigma_1^T\Sigma_{-1}^{-1}\right) \end{bmatrix} = \begin{bmatrix} q_{11} & -q_1^T \\ q_1 & \mathcal{Q}_{-1} \end{bmatrix} \tag{9}$$

This is used to simplify the conditional distributions of a particular random variable in the multi-variate gaussian from Eq 7-8.

$$p(X_i|\mathbf{X_{-i}}) = \mathcal{N}\left(\mu_i + \Sigma_{X_i\mathbf{X_{-i}}}\Sigma^{-1}_{\mathbf{X_{-i}}\mathbf{X_{-i}}}\left(\mathbf{X_{-i}} - \mu_{\mathbf{x_{-i}}}\right), \Sigma_{X_iX_i} - \Sigma_{X_i\mathbf{X_{-i}}}\Sigma^{-1}_{\mathbf{X_{-i}}\mathbf{X_{-i}}}\Sigma_{\mathbf{X_{-i}}X_i}\right)$$

Plugging values from Eq 9 the expression is simplified as a single-variate gaussian for conditional auto-regression of the random variable:

$$p(X_i|\mathbf{X_{-i}}) = \mathcal{N}\left(\frac{\vec{q_i}^T}{-q_{ii}}\mathbf{X_{-i}}, q_{i|-i}\right)$$

Similar expressions can be written for each node in the graph. X comes from a gaussian distribution and can be represented as $X = \frac{\vec{q_i}^T}{-q_{ii}}\mathbf{X_{ii}} + \epsilon$ this is similar to a standard regression formulation where $X_i = \beta_i X_{-i} + \epsilon$. Where β_i represents the coefficients of the regression. Here i are the different columns of the matrix Q. For every value of i a LASSO regression is solved and the values are learned. The beta coefficients are dependent on the zero and non-zero values of the Q matrix given if a particular variable lies in the neighbourhood (is connected) of the others. Hence an estimate of the neighborhood s_i, we have:

$$p(X_i|\mathbf{X_{-i}}) = p(X_i|\mathbf{X_s}) \text{ where } s_i \text{ defines the Markov blanket of node } i$$

Iterating this process over all random variables gives the Meinshausen-Buhlmann(MB) algorithm for structure learning. An alternate method is called L_1 -regularized maximum likelihood learning and it directly learns the value of the Q matrix. First step is to compute the sample covariance matrix from data and an optimization problem is formed based on the maximum likelihood estimator as shown below:

$$\mathcal{Q}^* = argmax_Q \left\{\log det\mathcal{Q} - tr(SQ) - \rho||Q||_1\right\}$$

The algorithm iterates over the different columns of the $\mathcal{Q}$ matrix and minimize the above loss function. This leads to a more stable solution as there is a single loss function and the solution is achieved iteratively converging to a final solution.

3 Learning Ising Model

If the node x_i of graph we intend to learn is discrete, for example, voting outcome of a person, we have:

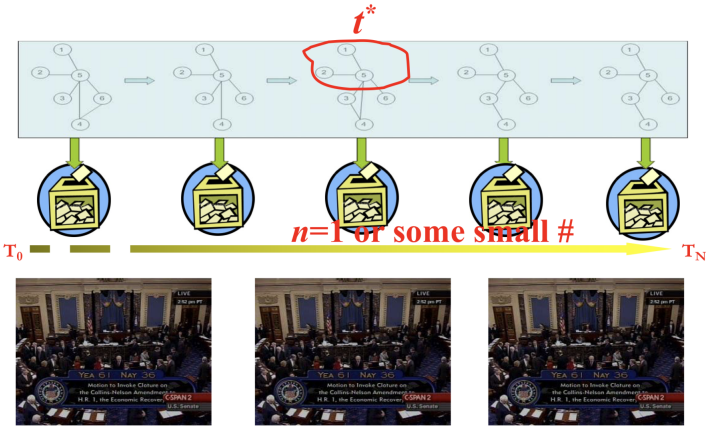
$$P(x|\theta) = \exp\left(\sum_{i\in v}\theta_{ii}^T x_i + \sum_{(i,j)\in E}\theta_{ij}x_i x_j - A(\Theta)\right) \tag{10}$$

Note that this is a pairwise MRF, where θ_{ij} reveals the graph structure. However, we cannot use simple linear regression to do graphical lasso due to the discrete nature of the random variable. Therefore, the conditional is no longer gaussian but logistic:

$$P_\theta(x_k|x_{-k}) = logistic(2x_k \cdot {\theta_{-k}}^T x_{-k}) \tag{11}$$

If each node is a variable length vector instead, partial correlation computing may be useful for us to draw connections between different dimensional nodes.

4 Evolving Social Networks



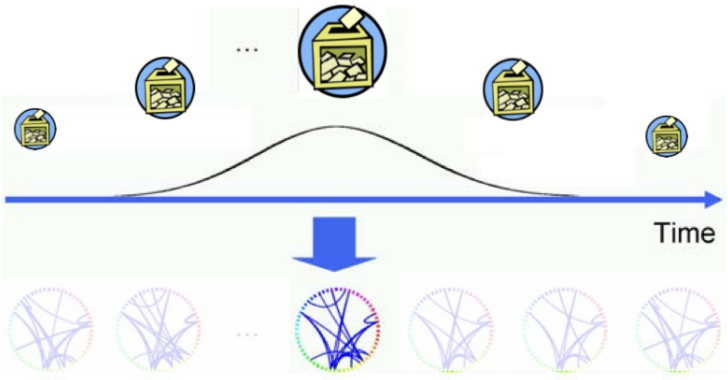
Evolving social graphs are interesting and hard to estimate because in real world, graphs are evolving and are not i.i.d generated. The difficulty is that we may have only one sample at each particular time step, which is not good for statistical estimation. However, the idea is that graphs are highly dependent across time steps, so we can come up with an algorithm to estimate the graph structure using all the samples over time.

4.1 kernel weighted L1 regularized Logistic Regression

The formal equation is given by:

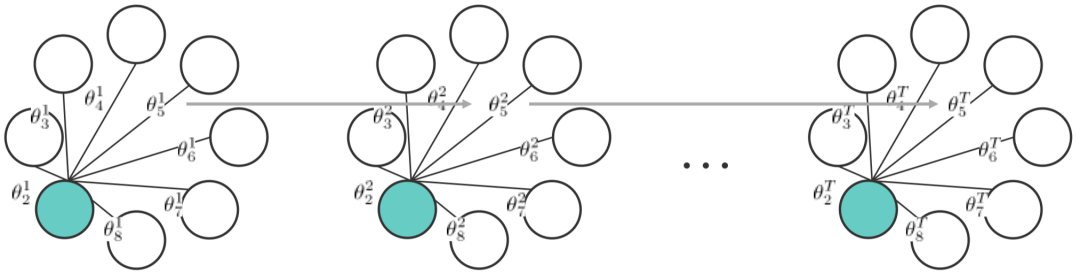
$$\hat{\theta}_i^t = \arg\min_{\theta_i^t} l_w(\theta_i^t) + \lambda_1 \|\theta_i^t\|_1, \tag{12}$$

where $l_w(\theta_i^t) = \sum_{t'=1}^{T} w(x^{t'}, x^t) \log p(x_i^{t'} | x_{-i}^{t'}, \theta_i^t)$.



Note that we introduce a weight function, which is defined for two time steps that measures the distance of graphs of two time steps. This operation is done over all time steps. The closer it is to the time of interest, the higher the weight should be. The good thing is we can utilize all the samples instead of only that for the specific time step to perform structure estimation.

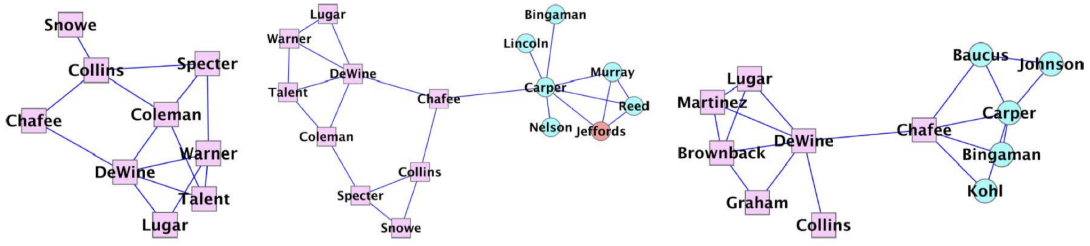
4.2 TESLA: Temporally Smoothed L_1 regularized logistic regression



$$\hat{\theta}_i^1, \cdots, \hat{\theta}_i^T = \arg\min_{\theta_i^1, \cdots, \theta_i^T} \sum_{+} t = 1^T l_{avg}(\theta_i^t) + \lambda_1 \sum_{t=1}^{T} \|\theta_{-i}^t\|_1 + \lambda_2 \sum_{t=2}^{T} \|\theta_i^t - \theta_i^{t-1}\|_q^q, \tag{13}$$

where $l_{avg}(\theta_i^t) = \frac{1}{N^t} \sum_{d=1}^{N^t} \log p(x_{d,i}^t, \theta_i^t)$. Note that we are estimating all the graphs together, therefore, the samples that define the conditional likelihood are not going to be re-weighted. Further, we are able to measure the difference of all graphs. The high level intuition of this objective is that the evolving is smooth and gentle over time. For a particular edge, we hypothesize that the difference should be small. In other words, if an edge is presented in a graph previously, it should be presented in the next graph with high chance.

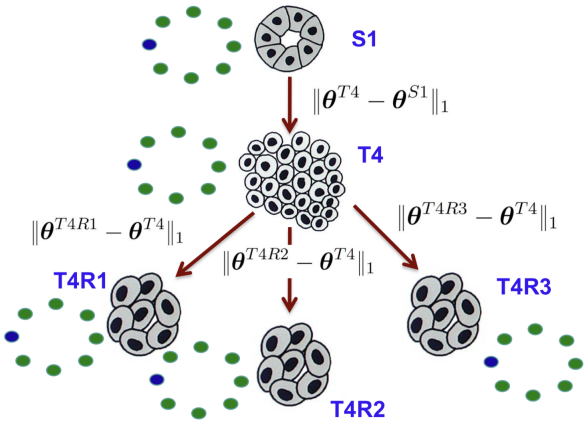
4.3 Senate Network



(a) t = 0.1 (b) t = 0.4 (c) t = 0.8

We use the data that consists of voting records on 542 bills of 100 senators, where each vote is a binary outcome. We can see that senator Chafee gradually changed his preference over time.

4.4 Breast Cancer Cells



The breast cancer cell develops over different stages such that scientists can observe different structures over time. The goal is to determine if the cell is normal given its structure. Specifically, we want to penalize the total variation of the graphs that are adjacent in the network.

References

[1] C Chow and Cong Liu. Approximating discrete probability distributions with dependence trees. *IEEE transactions on Information Theory*, 14(3):462–467, 1968.

[2] N. Meinshausen and P. Bühlmann. High-dimensional graphs and variable selection with the lasso. *Ann. Statist.*, 34(3):1436–1462, 06 2006.

[3] M. J. Wainwright. Sharp thresholds for high-dimensional and noisy sparsity recovery using ℓ_1 -constrained quadratic programming (lasso). *IEEE Transactions on Information Theory*, 55(5):2183–2202, May 2009.

10-708: Probabilistic Graphical Models, Spring 2020

17: Causality 1

Lecturer: Kun Zhang *Scribe: Ni Zhan, Tianqin Li, Carmel Fiscko, Xuecong Fu*

1 Intro

We want to learn why causality is important, how causal models relate to graph models, how to discover causal info, infer causal effects given causal structure and data, and benefits of knowing causal info.

Causality vs. Dependence

In daily life and scientific discovery, we have to distinguish between causal connections and associations. For example "couples who share housework are more likely to divorce" seems astonishing. However, association doesn't mean causality. In many real world application, we want to see causal connection, not just association.

Causality implies dependence. The idea "causality implies correlation but correlation does not imply causality" is basically correct. Fortunately, we can make use of dependence structure or pattern, and using multiple distributions/variables, we can discover causation from correlation under some assumptions.

Association - if two variables are not independent, one is useful to predict another.

X and Y are **associated** iff $\exists x_1 \neq x_2, P(Y|X = x_1) \neq P(Y|X = x_2)$

Causality

To say x is cause of y, need to go another level. We define causality based on intervention. Let's say you do intervention on x, and for different values of x, the corresponding distribution for y is different, then x causes y.

Def. Intervention (on x) - only change x without changing any other variable.

X is a **cause** of Y iff $\exists x_1 \neq x_2, P(Y|do(X = x_1)) \neq P(Y|do(X = x_2))$

Since all other variables have same value, we can say the difference in y is because of x.

Examples

- Causal relation between hot weather and high sales of ice cream, are they causally related? If you can make it very hot, you can see high sales of ice cream. However you can intervene on B (ice cream sales) without changing A (hot weather), by hiring a lot of people to buy ice cream. We can have noise in the variables, and A and B not necessarily deterministically related in this case.
- Ex 2. A guy goes to work by bus around 8:30. At the same time, the bus is coming. Variable-1 is guy leaving home 0 or 1, variable-2 is bus coming. Variables are clearly correlated, but are they causally related? Intuitively, no because the man does not see the bus coming. Now we do intervention by changing target variable, jump in front of bus while it is incoming to make bus stop, action will not change anything else in system. In this case, man would probably still leave. There could be a common cause such as bus timetable. This example shows we really have to pay attention to definition of intervention. If we change bus timetable to change if bus is coming, it is not intervention because it will probably also change if guy leaves home at that time.

We can use directed graphs for causal relations. In a DAG, direction from a to b means a causes b.

1.1 Outline

- why benefit from causal thinking.
- difference between causal graph model and graph models.
- identification of causal effects, counterfactual reasoning.

2 Causal thinking

We have to distinguish between dependence and causality. If you want to make prediction, dependence is enough. If you want to change the system, then must care about causality.

Examples

- If we want to change incidence of cough, knowing dependence with yellow fingers does not help. We need to go to cause, which is smoking.
- **Simpson's paradox:** (based on a real dataset published in 1970s, Figure 1) There is a hospital with two groups of patients with kidney stones: large and small, and two treatments A and B. For each group, treatment A is better. If you merge the data, treatment B seems better. If you are doctor and want to maximize recovery, would you recommend A or B? In principle, even if you don't know the size of kidney stone, should still pick A. Stone size influences treatment and recovery, stone size is a common cause. Because common cause is there, you cannot correctly see causal inference of treatment on recovery from the dependence pattern. We will come back to this example later.

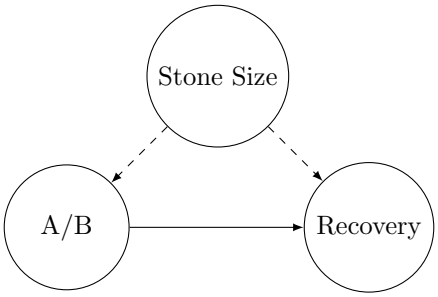


Figure 1: Simpson's Paradox.

- Cholesterol vs. exercise per age group. Separating by age, cholesterol decreases when exercise increases. If you ignore the age, more exercise = more cholesterol, different conclusion than when you observe ages. If you want to make recommendation about exercise and cholesterol, have to go to causal level.
- **Dependencies in data.** Going back 60 years in data, female college students smarter than male. This happened essentially because of selection. Admittance into college was dependent on gender and IQ. If you look at whole population, the "female smarter than male" pattern will disappear. Need to look for and correct selection bias.
- **Survivorship bias.** In world war 2, we put armor on airplanes to increase chance of survival, and looked at holes on returning flights. Where should I put armor? "Being hit" and "where holes are" are causes of survival. We can observe $probability(holes|survived)$. We want to infer pattern of holes that did not survive.
- **Monty hall problem.** We want to increase chance of winning money. Location of money is independent of your initial choice. "Which door is opened by host" is common effect. Two originally

independent variables ("initial choice" and "where money is") are now dependent. Now initial choice has some info about location of money, so I should change my mind of door selection.

Causal thinking makes a difference. If we know causality, we can do:

- active manipulation
- generalization / adaptation in new environments
- info integration. x can cause z through y. Contrast this with dependence: If you only know dependence, you cannot integrate information in this way. (For example, x and y are dependent, and y and z dependent, x and z could be correlated, identical, independent, etc. Only knowing dependence, you do not know relationship between x and z.)

Causality is creativity problem, we need to ask "what if", and integrate information from different aspects.

Ex. Causality for prediction

- **Figure and shadow.** In first scenario, you only see shadow, can you say anything about figure? Second scenario: you only see figure, can you say anything about shadow? Which one is easier?
- Suppose you write down 5, and ask someone to predict if it is a 5? Easy. Then ask someone to predict how I handwrite 5? Hard.
- Prediction is not symmetric, because there is an environment change.
- The figure is cause of shadow.
- Effect contains information about cause.
- If I have no idea about environment, I cannot predict effect from cause, but it is easier to predict cause from effect (This is not rigorous).
- We see that it is easier to predict the figure given shadow, than the shadow given figure, if we do not know about the environment (light source, time of day, etc.).

We can make use of some dependence pattern in data to make predictions.

Change of environment is related to a distribution shift problem. Using causality, we can understand system and causal process, identify source of change, then make prediction.

Problem with google photos

Google photos classified African people as gorillas. Two and half years later, google claimed they solved problem by removing "gorillas" from set of possible labels.

Adversarial Attack

Add a little noise, the prediction changes from panda to gibbon. Machines use a different process than humans to make prediction. If human and machine processes are different, then it is always possible to change something so that the machine decision changes but ours stays the same, or vice versa. This is called adversarial attack. Are human and machine processes consistent in some way? Human process does not look at pixels, we look at high level features and our experience. We want to enable machines to think in a human-like manner.

Artificial Intelligence

We usually assume a fixed distribution of data. We train then apply same model to test. In most cases distribution is not fixed. If you play badminton well, you can play ping pong well. A human driver can make right decision in new scenario. But this is hard for machines.

Desribe intelligence:

- able to understand system
- because of understanding, can do control, intervention

- decompose complex task into small task
- information theory: combine small pieces into big picture
- learn from a small number of examples because of understanding
- interpolation and extrapolation

How can we achieve these things for machines?

How to achieve intelligence?

One way is to examine how brain works, and develop some system. We want to find principles underlying intelligence. From evolution perspective, there are two properties: humans survived and prospered. We have a good ability to make prediction across scenarios (otherwise hard to survive). We have an inner compact representation, allowed us to explain observations, explain connection between scenarios, tasks, etc. We are creative, we can change the world (intervention). We used causal info about world.

3 Causal Graphical Models

D-separation

review on your own.

What does d mean in d-separation? originally means directional. Separation criteria is different for undirected graphs.

Local and global markov condition: They are same for DAGs.

Causal Bayesian Networks Interpreting causality using graph models and conditional distributions.

The causal DAG describes effects of intervention, and you can get the distribution of variables resulting from intervention. Given a DAG, you may modify DAG to represent an intervention. How can you know when a DAG represents causal relationship? DAG is causal if 3 conditions are true:

- 1. Is markov: intervention does not change conditional independence relations
- 2. If you do intervention, then you set the value. Ex. If you set x3 to "on", then p(x3 is on) = 1
- 3. If you do intervention on x3, conditional distribution of other causal modules does not change because of intervention. To only change this module, you cut off edges going into x3 and set x3 to on, without changing anything else.

Formally, Let $P_x(V)$ be distribution of V resulting from intervention $do(X = x)$. A DAG G is a causal bayesian network if:

- 1. $P_x(V)$ is Markov relative to G
- 2. $P_x(V_i = v_i) = 1$ for all $V_i \in V$ and v_i consistent with $X = x$
- 3. $P_x(V_i|PA_i) = P(V_i|PA_i)$ for all $V_i \notin X$ (PA_i means parents of V_i)

If the three conditions are met, then the graph representation is causal, and you can do inference.

Structural Causal Models Alternatively, we can use structural causal models, which is a set of equations. The value of xi is a function of parents and ei (exogenous / error).

$$X_i = f_i(PA_i, E_i)$$

If system is causal, then each equation represents autonomous mechanism, i.e. each equation is not relevant to other equations. The variables can be dependent, but equations are not related, which is known as **modularity**. Modularity is essential property of causal system. When modularity holds, I can locate where

changes are. Each equation describes how you assign value to left hand side, and describes structure that underlies the variables.

Three types of problems in AI

- Prediction: Do not need causal representation, only care about conditional distribution $p(x3|x2 = 1)$
- Intervention: Predict effect of intervention. "Would person cough if we make sure he has yellow fingers?" We want to see $p(x3|do(x2 = 1))$ (probablility of x3 given intervention on x2). We need the causal picture. If x2 is cause of x3, we need $p(x3|do(x2 = 1))$. If x3 is cause of x2, $p(x3|do(x2 = 1)) = p(x3)$.
- Counterfactual: "Would George cough had he had yellow fingers, given that he does not have yellow fingers and coughs?" This question has more information, we want to see effect of intervention given observation that he does not have yellow fingers and coughs. $p(x3_{x2=1}|x2 = 0, x3 = 1)$ Couterfactuals relate to a particular person, day, etc.

4 Identify causal effects

Randomized control experiments Ex. There are two groups of patients and two treatments. Apart from treatment, other variables need to have same value or distribution. If I observe difference in recovery, it must be caused by treatment. It is hard to do this experiment, hard to constrain patients to have the same value/distribution of other features. Sometimes you do not know what other variables need to be considered. Instead of randomized control experiments, we can use causal understanding and observation data, called causal inference.

Def. Causal discovery: aims to discover causal info by analyzing data

Causal inference and causal discovery are two different problems in causality.

Causal effect Ex. Simpson's paradox. We want to see the effect of treatment on recovery.

$$P(R|T) = \sum_S P(R|T, S)P(S|T)$$

$$P(R|do(T)) = \sum_S P(R|T, S)P(S)$$

Top equation is conditional distribution, and from chain and sum rule in probability theory. If we want to see effect of treatment on recovery, need to find $P(R|do(T))$ (the probability distribution of recovery given intervention of treatment), this describes causal link (we do not care about other factors). How are two equations different? $P(S|T)$ (stone size given treatment (conditional distribution)) becomes $P(S)$ (marginal of stone size), because when you do intervention, you cut off arrow from stone size to treatment, then stone size and treatment become independent.

Origin of paradox: A is better than B. Blue line is for small stones. Many patients with small stone received treatment B. Many patients with large stone received teatment A. For conditional distribution, take average across all patients, and mean is close to where more patients are. Then you get "B better than A" for all patients. When use intervention, you get "A better than B". Intervention is manipulation, we cut some edges off, so the calculated expression is different.

Def. Causal effect probability of target variable given intervention on variable you want to change. To understand this quantity intuitively, in an intervention, we cut off edges into x (the variable we are changing),

set x value, infer probability distribution of target variable y. The average effect is expectation of this distribution. We can also derive other measures based on this distribution.

Identifiability of causal effects A fundamental question in causuality. Statisticians and computer scientists do research in this direction. What is meant by identifiable? Given a causal graph and the same distribution for observed variables, if $P(y|do(x))$ is same for different models, then causal effect of X on Y is identifiable. Identifiable implies unique solution. Ex. We do not observe stone size data, but do observe treatment, recovery data, can we recover influence of treatment on recovery?

Key issue is to control confounding bias. Ex. If you do not observe age, then cholesterol-exercise casual relation is not identifiable.

Two criteria for identifiability:

- **Back door criterion** For a set of variables Z, Z satisfies back door criterion relative to ordered pair (Xi, Xj) if:
  - No node in Z is descendant of Xi.
  - Z blocks every path between Xi and Xj that contains arrow into Xi. Here the back door paths are paths into Xi.

  Z = {X4, X3} satisfies.
  Z = {X4} does not. If you condition on X4, the M-shaped back door path is open (according to d-separation).
  If Z satisfies back door criterion relative to (X,Y), then

$$P(y|do(x)) = \sum_z P(y|x,z)P(z)$$

- **Front door criterion** A set of variables Z satisfies front door criterion relative to ordered pair (X, Y) if:
  - Z intercepts all directed paths from X to Y.
  - There is no back door path from X to Z.
  - All back door paths from Z to Y are blocked by X.

  If Z satisfies front door criterion relative to (X,Y), then

$$P(y|do(x)) = \sum_z P(z|x) \sum_{x'} P(y|x',z)P(x')$$

Ignorability Compare backdoor criterion with ignorability. Ignorability condition is

$$Y(x) \perp\!\!\!\perp X|Z.$$

What is small x random variable according to potential outcome framework? Y is function of x, u. x is value of some r.v., u is specification of unit (person, day, scenario). $Y(x,u)$ is value of y for unit u on intervention x. We can ignore the unit u, and then $Y(x)$ is counterfactual random variable. Y(x) is description of causal mechanism from x to y. The ignorability condition means: how Y is generated from X will be independent from X, given Z. If back door (graphical condition) is satisfied, then ignorability is satisfied (mostly, we will not discuss the subtle differences.)

Next lecture We will discuss unification of criteria, why do counterfactual reasoning, and how discover causal information from observational data.

10-708: Probabilistic Graphical Models, Spring 2020

18: Causality II

Lecturer: Kun Zhang Scribe: Xueying Ding, Naveen Shankar, Koyoshi Shindo, Zeyu Tang, Yilin Yang

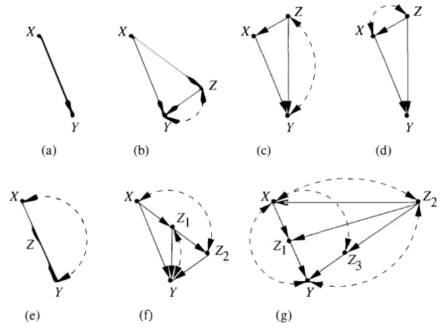
1 Introduction

In this lecture, we will first continue the discussion about causal inference; then we will be talking about causal discovery.

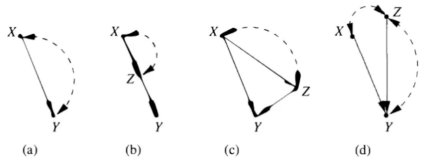
2 Identifying a Causal Effect

A sufficient condition for identifying the causal effect $P(y|do(x))$ from a graph is that there exists no bi-directed path (a path composed entirely of bi-directed arcs) between X and any of its children (Tian and Pearl [2002]).

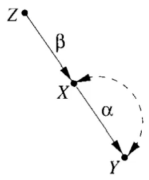
For example, this condition holds in the following graphs:



And does not hold in these graphs:



However, if you assume the system is linear, you can often recover the causal effect even if the condition does not hold. For example, in the following graph:



$\beta = r_{XZ}$ (the regression coefficient of regressing X on Z) and $\alpha\beta = r_{YZ}$. As such, you can recover the true

causal effect α by $\alpha = \frac{r_{YZ}}{r_{XZ}}$.

3 Propensity Scores

Assuming the back-door criterion (or conditional ignorability) condition holds, the average causal effect (ACE) can be calculated as

$$\text{ACE} = \mathbf{E}[Y|do(x)] - \mathbf{E}[Y|do(x')]$$

Where e.g. x is the treatment condition, x' is the baseline, and $P(Y|do(x)) = \sum_c P(Y|x,c)P(c)$, for some confounding covariate(s) c.

When you have randomized controlled experiments, $P(c)$ is the same across the treatment and baseline conditions, so the ACE is simple to calculate. However, without randomized controlled experiments, there is no such guarantee. Instead, you have to do something like match $P(C|x)$ to $P(C|x')$. Unfortunately, C is usually high-dimensional, which makes such a matching problem difficult. One alternative is to use *propensity scores*.

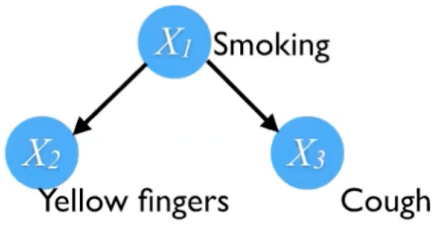
Let the propensity score $h(C) = P(X = 1|C)$, then $X \perp\!\!\!\perp C|h(C)$, and we call $h(C)$ and C confounding-equivalent because:

$$\begin{aligned}\text{ACE} &= \sum_c P(Y|x,c)P(c) = \sum_c \sum_h P(Y|x,c)p(h)p(c|h) = \sum_c \sum_h P(Y|x,c,h)p(h)p(c|h,x) \\ &= \sum_c \sum_h P(Y,c|x,h)P(h) = \sum_h P(Y|x,h)P(h)\end{aligned}$$

In other words, the ACE can be calculated using $P(h)$ instead of $P(c)$. This is extremely useful, since regardless of how high-dimensional C is, $h(C)$ is always a scalar. Thus, the task of matching $P(h)$ across different groups is feasible even when C is high-dimensional, leading to more accurate estimations of the ACE.

4 Counterfactual Reasoning

Consider a simple three variable graph, where each variable is binary:



There are three types of questions we might want to answer about the graph:

- Prediction: Would George cough if we *find* he has yellow fingers? This is $P(X_3|X_2 = 1)$
- Intervention: Would George cough if we *make sure* that he has yellow fingers? This is $P(X_3|do(X_2 = 1))$
- Counterfactual: Would George cough *had* he had yellow fingers, *given that he does not have yellow fingers and coughs*? This is $P(X_{3_{X_2=1}}|X_2 = 0, X_3 = 1)$

In general, statements about counterfactual reasoning are more powerful than statements about intervention, which are more powerful than statements about prediction.

For example, consider the situation where you have two variables X and Y, where $Y = \log(X + E + 3)$; E is some exogenous error, and X causes Y. Furthermore, say we observe for George $x = -1$ and $y = 0.4$, and we are interested in what Y would have been had X been 0 instead. With prediction, the "best" answer we can give is $\mathbf{E}[Y|X = 0]$, or $\log(0 + 3) = 1.10$.

However, with counterfactual inference, we can make claims *specific* to George, rather than general claims about people with $X = 0$. We do this by following three steps of counterfactual inference to calculate $P(Y_{X=0}|X = -1, Y = 0.4, E = e)$:

- Abduction: Find $P(E|\text{evidence}) = P(E|X = x, Y = y)$
- Action: Replace the equation for X by $X = x'$
- Prediction: Use the modified model to predict Y

Here, the process is:

- Abduction: $e = \exp(y) - x - 3 = \exp(0.4) + 1 - 3 = -0.51$
- Action: Set $X = x' = 0$ (instead of $X = x = -1$)
- Prediction: $Y = \log(x' + e + 3) = \log(0 - 0.51 + 3) = 0.91$

As a result, we provide the counterfactual inference value of $Y = 0.91$ instead of the prediction value $Y = 1.10$. Counterfactual inference allows us to make a claim about a specific person, and gives an answer that is both more informative and notably different from the best answer given by prediction. In this case, we can make such a claim by leveraging the fact that once observed, the exogenous error E is specific to George, and would have remained the same in counterfactual scenarios.

5 Causal Discovery (Overview)

Life abounds with examples when causal reasoning could be quite helpful: finding the causal relationship between agriculture, culture, and climate; investigating how certain variables would influence the shape of human skeleton in the long run; distinguishing causes from effects by analysing distribution information; reasoning about the latent variable when we do not have access to such variables; and so on.

Traditional causal discovery methods fall into two categories: constraint-based and score-based methods. Before we look at various causal discovery algorithms, let's first consider the connection between *causal structure* and *statistical properties of data*, and suitable assumptions are needed.

5.1 Modularity

Within a causal system, the whole causal process can be divided into small modules based on the *modularity* property.

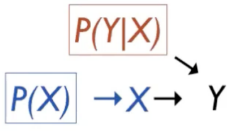


Figure 1: Modularity property

As illustrated in fig. 1, we can see that, if there is no confounder between X and Y, the data generating process of the cause (here is X) and the data generating process of effect from cause (here is generating Y from X) are independent.

5.2 Causal Sufficiency

We say a set of random variable $\mathbf{V}$ is causally sufficient, if $\mathbf{V}$ contains every direct cause (with respect to $\mathbf{V}$) of any pair of variables in $\mathbf{V}$.

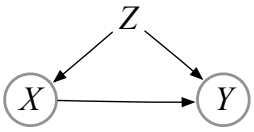


Figure 2: Causal sufficiency illustration

As illustrated in fig. 2, Z is the unobserved common cause (hidden confounder) of X and Y, according to our definition of causal sufficiency, $\mathbf{V}_1 = \{X, Y, Z\}$ is causally sufficient, while $\mathbf{V}_2 = \{X, Y\}$ is not.

5.3 Causal Markov Condition vs. Faithfulness

In the previous lectures, we have learnt the d-separation, namely, "read" the conditional independence relation from the graph. We know that if X and Y are d-separated by Z in graph $\mathcal{G}$, then the conditional independence $X \perp\!\!\!\perp Y \mid Z$ holds true.

Notice that the contrapositive of this proposition, namely, conditional dependence implies d-connection, does not tell us what would happen to the underlying graph if we actually found some conditional independence relations in the data. Therefore, beyond (global) Markov condition, we need something else to connect the conditional independence in data back to the property in the causal graph, that is, faithfulness.

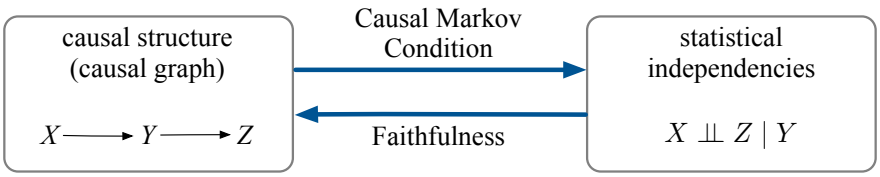


Figure 3: Caption

In the figure above, Causal Markov Condition is saying that, each variable is conditional independent with its non-descendants (non-effect variables) conditional on its parents (direct causes); Faithfulness is saying that, all observed (conditional) independencies are entailed by Markov condition in the graph. Notice that faithfulness is a rather strong assumption, and not every causal discovery algorithm makes this assumption, as will see later when we discuss causal discovery algorithms.

With Causal Markov Condition and faithfulness assumption, we can recover the skeleton (only edges) of the causal graph. After orienting the derived skeleton, we can get the causal graph using the following PC algorithm.

6 PC Algorithm

An example of the constraint-based causal discovery methods is the PC algorithm (Spirtes and Glymour [1991]). This contains two steps:

Step1: X and Y are adjacent iff they are dependent conditional on every subset of the remaining variables.
Step2: Orientation propogation.

More details can be found in Algorithm 1.

Algorithm 1 PC Algorithm

A.) Form the complete undirected graph C on the vertex set **V**.
B.) n=0.
repeat
repeat
select an ordered pair of variables X and Y that are adjacent in C such that **Adjacenies**(C,X)\{Y} has cardinality greater than or equal to n, and a subset **S** of **Adjacenies**(C,X)\{Y} of cardinality n, and if X and Y are d-separated given **S** delete edge X-Y from C and record **S** in **Sepset**(X, Y) and **Sepset** (Y, X);
until all ordered pairs of adjacent variables X and Y such that **Adjacenies**(C,X)\{Y} has cardinality greater than or equal to n and all subset **S** of **Adjacenies**(C,X)\{Y} of cardinality n have been tested or d-separated;
n = n+1.
until for each ordered pair of adjacent vertices X, Y, **Adjacenies**(C,X)\{Y} is of cardinality less than n.
C.) For each triple of vertices X, Y, Z such that the pair X, Y and the pair Y, Z are each adjacent in C but the pair X, Z are not adjacent in C, orient X-Y-Z as X→Y←Z if and only if Y is not in **Sepset**(X, Z).
D.) repeat
If A→B, B and C are adjacent, A and C are not adjacent, and there is no arrowhead at B, then orient B-C as B → C.
If there is a directed path from A to B, and an edge between A and B, then orient A-B as A → B.
until no more edges can be oriented.

7 Equivalent Classes: Patterns

We can define the equivalanet classes: Two DAGs are (independence) equivalent if and only if they have the same skeletons and the same v-structures (Pearl and Verma [1991]). We can only recover such class based on the data using the conditional independent relations. Such (independence) equivalent classes can be represented by pattern, which is defined by: Patterns or CPDAG(Completed Partially Directed Acyclic Graph): graphical representation of (conditional) independence equivalence among models with no latent common causes (i.e., causally sufficient models).

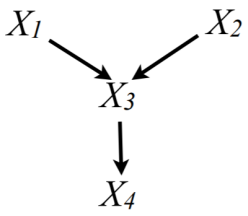
8 FCI

FCI (Fast Causal Inference) can validly infer causal relationships from conditional independence statements even when there are confounders. FCI is quite complex so the detail of the algorithm was not covered in the lecture, but some rules from FCI are covered in the lecture to showcase examples of when we can be certain that there must be no confounders between two variables, and we can be certain that there must be confounders between two variables.

8.1 Example 1. Y-structure implies no confounder

The three conditional independence statements: $X_1 \perp X_2$; $X_1 \perp X_4|X_3$; $X_2 \perp X_4|X_3$ imply that X_1, X_2, X_3, X_4 follow the so-called Y-structure shown in the figure below.

Then it is impossible for X_3 and X_4 to have a confounder. Suppose for contradiction that there is a confounder C that directly causes X_3 and X_4. Then the path $X_1 - X_3 - C - X_4$ will cause X_1 and X_4 to be d-connected given X_3, and the condition $X_1 \perp X_4 | X_3$ no longer holds, hence a contradiction.



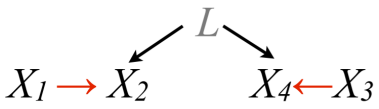
8.2 Example 2. If $A \perp C|B$ and B does not cause A, then there is no confounder between B and C.

Since B does not cause A, then it must hold that either A causes B or A and B have a confounder. In both cases, B causes C must holds and there must be no confounder between B and C. Otherwise, B would be a collider and A and C cannot be conditionally independent given B.

A real life example would be $A = raining$, $B = slippery$, $C = falling\ down$. Since *raining* and $falling\ down$ are conditionally independent given *slippery*, and *slippery* does not cause *raining*, we can be certain that *slippery* causes $falling\ down$ and there is no confounder between *slippery* and $falling\ down$. Another example would be $A = geographical\ background$, $B = economic\ conditions$, $C = emergence\ of\ science$.

8.3 Example 3. If $X_1 \perp X_3; X_1 \perp X_4; X_2 \perp X_3$, then X_2 and X_4 must have a confounder.

This is illustrated in the figure below. Suppose for contradiction that there are no counfounder between X_2 and X_4. Then applying PC rule, X_1, X_2 and X_4 should follow v-structure which implies X_4 causes X_2. Similarly, X_2, X_3 and X_4 should follow v-structure which implies X_2 causes X_4, then we have a contradction, so there must be a confounder (denoted as L in the figure below) between X_2 and X_4.

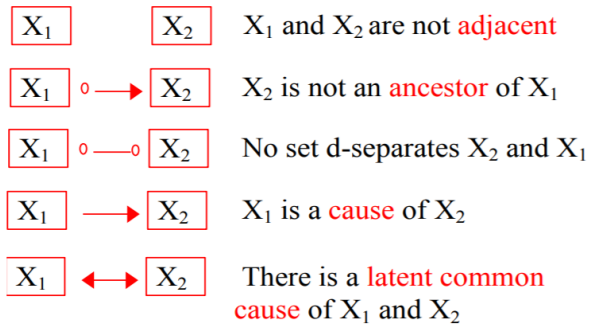


8.4 PAG

The above three examples showcase some rules that are integrated in the FCI algorithm. As can be seen from the example 3, result provided by FCI would make graph over observed variables not necessarily a DAG without the help of latent variables, so FCI returns output called PAG instead of DAG. There are 5 kinds of relationships between two variables in PAG shown in the figure below. A circle denotes that it could be an arrow head or arrow tail.

9 GES

PC algorithm and FCI are constraint-based causal discovery algorithms. Another branch of causal discovery algorithms is score-based. GES (Greedy Equivalence Search) is one kind of score-based causal discovery.



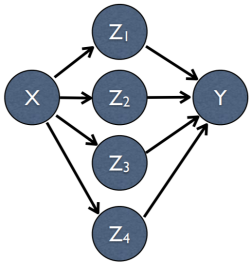
The score function satisfies 3 properties: score equivalent (assigning the same score to equivalent DAGs), locally consistent (score of a DAG increases when adding any edge that eliminates a false independence constraint and vice versa), and decomposable (score of a DAG can be written as summation over a function of node and its parents in the DAG). An example of a score function that satifies above three properties are BIC: $S_B(G, D) = \log p(D|\hat{\theta}, G^h) - \frac{d}{2}\log m$.

The GES algorithm consists of Forward Greedy Search and Backward Greedy Search through the sapce of DAG equivalence classes, and the detailed algorithms are shown below:

Algorithm 2 GES

1. Forward Greedy Search (FGS)
   Start from some (sparse) pattern (usually the empty graph)
   Evaluate all possible patterns with one more adjacency that entail strictly fewer CI statements than the current pattern
   Move to the one that increases the score most
   Iterate until a local maximum
2. Backward Greedy Search (BGS) Start from the output of the Forward stage
   Evaluate all possible patterns with one fewer adjacency that entail strictly more CI statements than the current pattern
   Move to the one that increases the score most
   Iterate until a local maximum

For example, if the data was generated by the DAG shown below, then it is likely that FGS will add an edge between X and Y since they are highly correlated. However, BGS will likely remove this edge between X and Y for a higher score.



10 Linear,Non-Gaussian Models

For classical models, the problem is defined as identifying the casual structures such as $X \longleftarrow Y \longrightarrow Z$, $X \longrightarrow Y \longrightarrow Z$, and $X \longleftarrow Y \longleftarrow Z$, which give rise to the conditional independence between the data as $X \perp\!\!\!\perp Z \mid Y$. However, recovering the causal relations from conditional independences is bounded by the equivalence class. Since the mapping between causal structures and conditional independence is not one-to-one, we cannot reconstruct the exact causal relationship. Furthermore, the classical methods cannot directly characterize and recover the cause-effect relationships for two-variable cases, such as $X \longrightarrow Y$, $X \longleftarrow Y$ and $X \longleftarrow Z \longrightarrow Y$ (where Z is an unobserved cause). Additional weak and reasonable assumptions are needed.

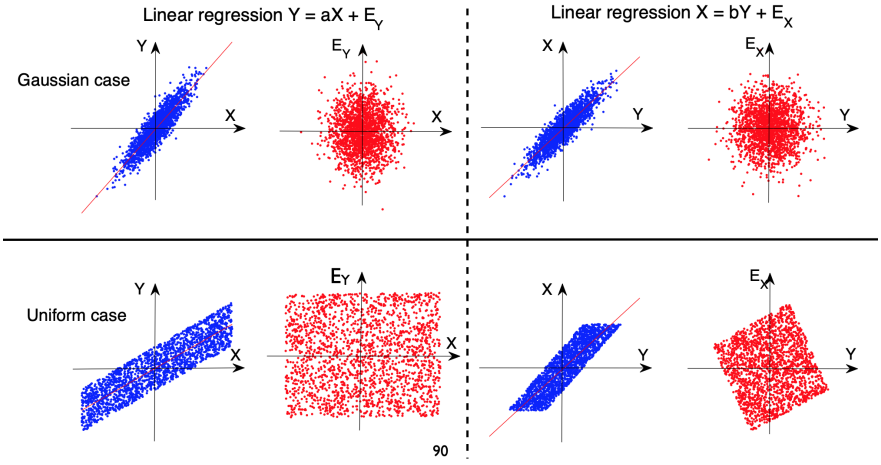
10.1 Functional Causal Model

A functional causal model represents the effect as a function of direct causes and noise: $Y = f(X, E)$, with $X \perp\!\!\!\perp E$.The following are some typical casual models for discovering the causal relations(Shimizu et al. [2006], Hoyer et al. [2008], Zhang and Hyvärinen [2009a], Zhang and Hyvärinen [2009b]):

- Linear non-Gaussian acyclic causal model: $Y = a \cdot X + E$.
- Additive noise model: $Y = f(X) + E$.
- Post-nonlinear causal model: $Y = f_2(f_1(X) + E$.

It is noted that even if we do not give any constraints on the function f, given two random variables X and Y, we can always represent a variable as a function of another variable and an independence noise. However, it is not the case if we have the constraints on the function.

In the linear case, we can easily go from correlation to asymmetry causal relation if we assumed the noise is non-Gaussian. An example is for data generated by $Y = aX + E$ i.e, $X \longrightarrow Y$ The upper case explains regressing Y on X and X on Y when the X and noise E are gaussian. In this case, the error term is uncorrelated from the predictor, which also implies independence. For the bottom cause, X and the noise is generated from uniform distribution. We can observe that the noise term is not independent from Y.



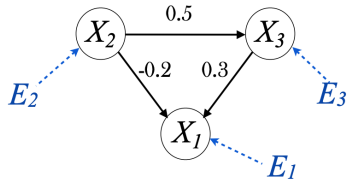
10.2 LiNGAM Model

In the more general case, we have the linear non-Gaussian acyclic model. The Linear non-Gaussian acyclic model is defined as the following:

$$X_i = \sum_{\text{j: parents of i}} b_{ij} X_j + E_i$$
$$\mathbf{X} = \mathbf{BX} + \mathbf{E}$$

with the disturbances(errors) E_i are non-Gaussian(or at most one is Gaussian) and mutually independent. An example is to represent the causal structure of the graph as:

$$X_2 = E_2$$
$$X_3 = 0.5X_2 + E_3$$
$$X_1 = -0.2X_2 + 0.3X_3 + E_1$$



The matrix form for the above LiNGAM model is given as the following. Note that the diagonal entries of the matrix are zero because we assume no self-loops, for simplicity.

$$\begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix} = \begin{bmatrix} 0 & -0.2 & 0.3 \\ 0 & 0 & 0 \\ 0 & 0.5 & 0 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix} + \begin{bmatrix} E_1 \\ E_2 \\ E_3 \end{bmatrix}$$

Thus, the task is reduced to recover the matrix B given the data. With the above LiNGAM model setting, we can reconstruct a unique solution $\mathbf{B}$ by the model and independent component analysis(ICA).

ICA aims to find a rotation transformation $\mathbf{Y} = \mathbf{WX}$, and to make each Y_i independent, maximum likelihood and mutual information minimization are used.

Thus, $\mathbf{B}$ is achieved from $\mathbf{W}$ by permutation and rescaling (make all diagonal entries non-zero, divide the row by its diagonal entry, and subtract the resulting matrix from the identity matrix).

10.3 Gaussianity or Non-Gaussianity?

The gaussian is widely used in the research area because of its "simplicity" of the form: marginal and conditionals are also gaussian, the mean and covariance matrix are easily computed, central limit theorem, and etc.

However, in practice, the non-gaussianity is actually ubiquitous. By Cramer's theorem, we observe the linear closure property of Gaussian distribution: if the sum of any finite independent variables is Gaussian, then all summands must be gaussian. But gaussian is only special in the linear case.

Other issues in causal discovery includes nonlinearities, categorical variables or mixed cases, measurement error, selection bias, causality in time series, non-stationary and heterogeneous data, and etc. One important issue is to face the confounding, or find the hidden causes or parents in the graph.

References

P. Hoyer, D. Janzing, J. Mooij, J. Peters, and B. Schölkopf. Nonlinear causal discovery with additive noise models. pages 689–696, 01 2008.

J. Pearl and T. Verma. A theory of inferred causation, 1991.

S. Shimizu, P. O. Hoyer, A. Hyvärinen, and A. Kerminen. A linear non-gaussian acyclic model for causal discovery. *J. Mach. Learn. Res.*, 7:2003–2030, Dec. 2006. ISSN 1532-4435.

P. Spirtes and C. Glymour. An algorithm for fast recovery of sparse causal graphs. *Social science computer review*, 9 (1):62–72, 1991.

J. Tian and J. Pearl. A general identification condition for causal effects. In *In Eighteenth National Conference on Artificial Intelligence*, pages 567–573, 2002.

K. Zhang and A. Hyvärinen. On the identifiability of the post-nonlinear causal model. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, UAI '09, page 647–655, Arlington, Virginia, USA, 2009a. AUAI Press. ISBN 9780974903958.

K. Zhang and A. Hyvärinen. Causality discovery with additive disturbances: An information-theoretical perspective. In *Proceedings of the 2009th European Conference on Machine Learning and Knowledge Discovery in Databases - Volume Part II*, page 570–585, Berlin, Heidelberg, 2009b. Springer-Verlag. ISBN 3642041736.

10-708: Probabilistic Graphical Models, Spring 2020

19: RL as Inference 1

Lecturer: Maruan Al-Shedivat Scribe: Harshit Sikchi, Yufei Wang, Mengdi Xu, Tianwei Ni, Yash Oza

1 Intro to Reinforcement Learning

In supervised learning we have a collection of data $D = [x_i, y_i]_{i=1}^n$ where our aim is to learn a model that approximates $P(y|x)$. In unsupervised learning we have a collection of data $D = [(x_1, x_2, x_3..., x_d)_i]_{i=1}^n$ where we seek learn a model which approximates $P(x_1, x_2..., x_d)$. Reinforcement learning sorts of closes the loop where the agent can interact with the world, obtain samples and learn a policy where it can maximize a reward function in the given environment. Reinforcement learning is useful as ultimately we want to build autonomous intelligent machines that can perceive and interact with the world, exhibit purposeful goal directed behavior and learn from interactions. Recently, Reinforcement learning have seen a number of successes where an RL agent was able to beat the world-master in game of GO, also in robotics where it was demonstrated that an robot trained with RL was able to learn how to manipulate a Rubik's cube to solve it.

Reinforcement learning can be specified as a Markov Decision Process(MDP). A MDP is specified by a set of states(S), a set of possible actions(A), environment dynamics($P(s_{t+1}|s_t, a_t)$) and a reward function r(s,a). The environment dynamics specify the transition probability of an agent from state s_t to state s_{t+1} after it takes an action a. The reward function provides a scalar feedback specifying a utility of the action. A trajectory in this MDP is represented as

$$\tau = (s_1, a_1, r_1, s_2, a_2, r_2, ..., s_H) \tag{1}$$

Using this framework we can choose to solve two common problems. The first is to find a policy $\pi : S \to A$ that outputs actions for each given state such that the cumulative reward along the trajectory is maximized. Alternatively we might be interested to find out the underlying MDP given a set of optimal trajectories. The first problem is the standard RL objective whereas the second one is known as Inverse Reinforcement learning.

1.1 Definitions

The **cumulative return** from timestep t is defined as the rewards accumulated starting from timestep t

$$G_t = r_{t+1} + r_{t+2} + .. + r_T \tag{2}$$

If $t = \infty$ the sum can become diverge and we can use the notion of discount factor γ, where $0 < \gamma < 1$ to get a finite sum.

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... \tag{3}$$

$$= r_{t+1} + \gamma G_{t+1} \tag{4}$$

A **policy** is a mapping from state to action. It can be deterministic as well as stochastic. In most general form, at any state s,

$$a \sim \pi(a|s) \tag{5}$$

Value function of a state s is defined as the expected cumulative reward obtained when starting from state s and following policy π.

$$V_\pi(s) := E_\pi[G_t|s_t = s] = E_\pi[\sum_{k=0}^{T} \gamma^k r_{t+k+1}|s_t = s] \tag{6}$$

Value function of a state-action pair or more commonly known as **Q function** of a state-action pair (s,a) is defined as the expected cumulative reward obtained when starting from state s, taking an action a and following policy π thereafter.

$$Q_\pi(s,a) := E_\pi[G_t|s_t = s, a_t = a] = E_\pi[\sum_{k=0}^{T} \gamma^k r_{t+k+1}|s_t = s, a_t = a] \tag{7}$$

1.2 Bellman Equations for Value and Q functions

Given the definition of value and Q functions, it is natural to derive the following Bellman Equations:

$$\begin{aligned} V_\pi(s) &:= \mathbb{E}_\pi[G_t|s_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{T} \gamma^k r_{t+k+1}|s_t = s\right] \\ &= \mathbb{E}_\pi[r_{t+1} + \gamma G_{t+1}|s_t = s] \\ &= \sum_a \pi(a|s) \sum_{s'} p(s'|s,a)[r(s,a) + \gamma \mathbb{E}_\pi[G_{t+1}|s_{t+1} = s']] \\ &= \sum_a \pi(a|s) \sum_{s'} p(s'|s,a)[r(s,a) + \gamma V_\pi(s')] \end{aligned} \tag{8}$$

$$\begin{aligned} Q_\pi(s,a) &:= \mathbb{E}_\pi[G_t|s_t = s, a_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{T} \gamma^k r_{t+k+1}|s_t = s, a_t = a\right] \\ &= r(s,a) + \gamma \mathbb{E}_\pi[G_{t+1}|s_t = s, a_t = a] \\ &= r(s,a) + \gamma \sum_{s'} p(s'|s,a) \sum_{a'} \pi(a'|s') \mathbb{E}_\pi[G_{t+1}|s_{t+1} = s', a_{t+1} = a'] \\ &= r(s,a) + \gamma \sum_{s'} p(s'|s,a) \sum_{a'} \pi(a'|s') Q_\pi(s', a') \end{aligned} \tag{9}$$

1.3 Optimal Policies and Value Functions

Goal of RL: find the optimal policy that achieves the highest expected returns. A policy π is better or equal to π' ($\pi \geq \pi'$) if its expected return is greater than that of π' in all states:

$$\pi \geq \pi' \Leftrightarrow V_\pi(s) \geq V_{\pi'}(s) \forall s \in S \tag{10}$$

Given this, we can define the optimal value and Q functions, and the Bellman Optimality Equations:

$$V_*(s) := \max_\pi V_\pi(s) = \max_a \sum_{s'} p(s'|s,a)[r(s,a) + \gamma V_*(s')] \tag{11}$$

$$Q_*(s,a) := \max_\pi Q_\pi(s,a) = \sum_{s'} p(s'|s,a)[r(s,a) + \gamma \max_{a'} Q_*(s',a')] \tag{12}$$

If we can compute the optimal Q values $Q_*(s,a)$, then we can recover the optimal policy $\pi_*(a|s)$ as:

$$\pi_*(a|s) = \delta\left(a = \arg\max_a Q_*(s,a)\right) \tag{13}$$

To recover a set of optimal trajectories, we just need to execute the optimal policy:

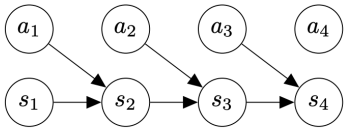
$$\begin{aligned} \tau_* &= (s_1^*, a_1^*, r_1^*, s_2^*, a_2^*, r_2^*, ...) \\ s_{t+1}^* &\sim p(s_{t+1}|s_t, a_t^* = \arg\max_a Q_*(s,a)) \end{aligned} \tag{14}$$

2 RL and Control as Inference: The GM framework

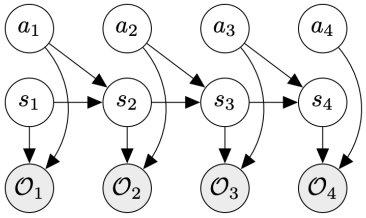
2.1 MDP as a Graphical Model

The graphical model for a standard MDP is shown on the left of Fig.1. The state is a Markov Chain and the states and actions are both random variables.

In MDP some transitions are rewarded with high rewards, and we hope to up weight the trajectories with high rewards and down weight the suboptimal ones. Therefore we augment the graphical model with an optimality variable $\mathcal{O}_t$ which is observable and makes it a Hidden Markov Process. The conditional distribution of the optimality variable is $p(\mathcal{O}_t = 1|s_t, a_t) = \exp(r(s_t, a_t))$. High rewards means the high probability of being optimal at time point t. Note that here we assume the reward are adjusted to make sure $p(\mathcal{O}_t = 1|s_t, a_t)$ is a probability distribution.



Initial state	$s_0 \sim p_0(s)$
Transition	$s_{t+1} \sim p(s_{t+1} \mid s_t, a_t)$
Policy	$a_t \sim \pi(a_t \mid s_t)$
Reward	$r_t = r(s_t, a_t)$



Initial state	$s_0 \sim p_0(s)$
Transition	$s_{t+1} \sim p(s_{t+1} \mid s_t, a_t)$
Policy	$a_t \sim \pi(a_t \mid s_t)$
Reward	$r_t = r(s_t, a_t)$
Optimality	$p(\mathcal{O}_t = 1 \mid s_t, a_t) = \exp(r(s_t, a_t))$

Figure 1: MDP as a Graphical Model

Why the optimality variable $\mathcal{O}_t$ is important?

- The auxiliary variable $\mathcal{O}_t$ allows us to incorporate the reward information into a probabilistic generative process for sampling the trajectories. We can solve control and planning problems using probabilistic inference algorithms in this Hidden Markov Model.
- It allows us to probabilistically specify a model of optimal behavior, is importance for inverse RL.

- It also provides an explanation for why stochastic behavior might be preferred (for the explanation and transfer learning point of view).

Given the graphical model, we can

- Given a reward, determine how likely a trajectory to be optimal. Mathematically, we can compute $p(\tau, \mathcal{O}_{1:T})$, the probability of a trajectory τ given acting optimally throughout the trajectory.

$$\begin{aligned} p(\tau, \mathcal{O}_{1:T}) &\propto p(s_1) \prod_{t=1}^{T} p(a_t|s_t) p(s_{t+1}|s_t, a_t) p(\mathcal{O}_t|s_t, a_t) \\ &= p(s_1) \prod_{t=1}^{T} p(s_{t+1}|s_t, a_t) \exp(r(s_t, a_t) + \log p(a_t|s_t)) \\ &= \left[p(s_1) \prod_{t=1}^{T} p(s_{t+1}|s_t, a_t) \right] \exp\left(\sum_{t=1}^{T} r(s_t, a_t) + \log p(a_t|s_t) \right) \end{aligned}$$

- Given a collection of optimal trajectories, infer the reward and priors, which is basically an inverse RL question.

$$\begin{aligned} p(\tau, \mathcal{O}_{1:T}, \theta, \phi) &\propto \left[p(s_1) \prod_{t=1}^{T} p(s_{t+1}|s_t, a_t) \right] \exp\left(\sum_{t=1}^{T} r_\phi(s_t, a_t) + \log p_\theta(a_t|s_t) \right) \\ &= \left[p(s_1) \prod_{t=1}^{T} p(s_{t+1}|s_t, a_t) \right] \exp\left(\sum_{t=1}^{T} \phi^T f_r(s_t, a_t) + \log \theta^T f_p(a_t|s_t) \right) \end{aligned}$$

  The problem is a featurized CRF. By recovering the parametric potential functions f_r and f_p, we can learn the reward recovered from the trajectories. Note that CRF is undirected and does not preserve the casual structure; this model is more restrictive and known as MEMM.

- Given a reward, infer the optimal policy by calculating $p(a_t|s_t, \mathcal{O}_{t:T})$. Instead of solving the optimization problem, we now can solve the inference problem.

2.2 Optimal Policy via Inference

Now we aim to infer the optimal policy $p(\mathbf{s}_t|\mathbf{a}_t, \mathcal{O}_{t:T})$ by standard message passing algorithm. It is sufficient to compute the backward message $\beta_t(\mathbf{s}_t, \mathbf{a}_t) = p(\mathcal{O}_{t:T}|\mathbf{s}_t, \mathbf{a}_t)$, which denotes the probability of a trajectory to be optimal from t to T starting from the state and action at time t. Also we introduce the message $\beta_t(\mathbf{s}_t) = p(\mathcal{O}_{t:T}|\mathbf{s}_t)$. Then the messages can be computed recursively:

$$\begin{aligned} \beta_t(\mathbf{s}_t) = p(\mathcal{O}_{t:T}|\mathbf{s}_t) &= \int_{\mathcal{A}} p(\mathcal{O}_{t:T}|\mathbf{s}_t, \mathbf{a}_t) p(\mathbf{a}_t|\mathbf{s}_t) d\mathbf{a}_t = \int_{\mathcal{A}} \beta_t(\mathbf{s}_t, \mathbf{a}_t) p(\mathbf{a}_t|\mathbf{s}_t) d\mathbf{a}_t \\ \beta_t(\mathbf{s}_t, \mathbf{a}_t) = p(\mathcal{O}_{t:T}|\mathbf{s}_t, \mathbf{a}_t) &= \int_{\mathcal{A}} \beta_{t+1}(\mathbf{s}_{t+1}) p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) p(\mathcal{O}_t|\mathbf{s}_t, \mathbf{a}_t) d\mathbf{s}_{t+1} \end{aligned} \tag{15}$$

Then the optimal action distribution can be derived by two backward messages:

$$\begin{aligned} \pi(\mathbf{a}_t|\mathbf{s}_t) := p(\mathbf{a}_t|\mathbf{s}_t, \mathcal{O}_{t:T}) &= \frac{p(\mathbf{s}_t, \mathbf{a}_t|\mathcal{O}_{t:T})}{p(\mathbf{s}_t|\mathcal{O}_{t:T})} = \frac{p(\mathbf{s}_t, \mathbf{a}_t, \mathcal{O}_{t:T})}{p(\mathbf{s}_t, \mathcal{O}_{t:T})} = \frac{p(\mathcal{O}_{t:T}|\mathbf{s}_t, \mathbf{a}_t) p(\mathbf{s}_t) p(\mathbf{a}_t|\mathbf{s}_t)}{p(\mathcal{O}_{t:T}|\mathbf{s}_t) p(\mathbf{s}_t)} \\ &\propto \frac{p(\mathcal{O}_{t:T}|\mathbf{s}_t, \mathbf{a}_t)}{p(\mathcal{O}_{t:T}|\mathbf{s}_t)} = \frac{\beta_t(\mathbf{s}_t, \mathbf{a}_t)}{\beta_t(\mathbf{s}_t)} \end{aligned} \tag{16}$$

Here we assume the action prior is a uniform distribution $p(\mathbf{a}_t|\mathbf{s}_t) = 1/|\mathcal{A}|$.

Then in order to get more intuition in RL, we introduce the message in log-space:

$$\begin{aligned} Q(\mathbf{s}_t, \mathbf{a}_t) &= \log \beta_t(\mathbf{s}_t, \mathbf{a}_t) \\ V(\mathbf{s}_t) &= \log \beta_t(\mathbf{s}_t) \\ \pi(\mathbf{a}_t|\mathbf{s}_t) &\propto \exp(Q(\mathbf{s}_t, \mathbf{a}_t) - V(\mathbf{s}_t)) = \exp(A_t(\mathbf{s}_t, \mathbf{a}_t)) \end{aligned} \tag{17}$$

Actually, the log-messages Q, V correspond to the state-action and state value function in a soft version. The action distribution is proportional to advantage value. Moreover, by the relationship in 15, we can derive the following relationship for Q, V:

$$\begin{aligned} V(\mathbf{s}_t) &= \log \int_{\mathcal{A}} \exp(Q(\mathbf{s}_t, \mathbf{a}_t)) d\mathbf{a}_t \approx \max_{\mathbf{a}_t} Q(\mathbf{s}_t, \mathbf{a}_t) \\ Q(\mathbf{s}_t, \mathbf{a}_t) &= \log p(\mathcal{O}_t|\mathbf{a}_t, \mathbf{s}_t) + \log \int \beta_{t+1}(\mathbf{s}_{t+1}) p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) d\mathbf{s}_{t+1} = r(\mathbf{s}_t, \mathbf{a}_t) + \log \mathbb{E}_{\mathbf{s}_{t+1}}[\exp(V(\mathbf{s}_{t+1})] \end{aligned} \tag{18}$$

Thus V can be seen as the soft-max of Q. When the dynamic is deterministic, the second relationship is exactly Bellman equation backup:

$$Q(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + V(\mathbf{s}_{t+1}) \tag{19}$$

However, when the dynamic is stochastic, the update is optimistic, because it will be largely determined by the max of next state value, which creates risk-seeking behavior. This issue will be mitigated by variational inference in the next section.

In conclusion, with the PGM augmented by optimality variables, we reduced the optimal control to inference in a HMM-like model, and make its connections with dynamic programming, value iteration in RL field.

3 Connections to Variational Inference

3.1 Which objective does the inference optimize?

Recall that the optimal trajectory distribution:

$$p(\tau) \propto \left[p(\mathbf{s}_1) \prod_{t=1}^{T} p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) \right] \exp\left(\sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t) \right) \tag{20}$$

Now we aim to optimize an approximate policy to be optimized to be closed to this trajectory distribution. Let the policy be $\pi(\mathbf{a}_t|\mathbf{s}_t)$, then its trajectory distribution under **deterministic** dynamics (where $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \mathcal{O}_{t:T}) = p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$):

$$\begin{aligned} \hat{p}(\tau) &= p(\mathbf{s}_1|\mathcal{O}_{1:T}) \prod_{t=1}^{T} p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \mathcal{O}_{t:T}) \pi(\mathbf{a}_t|\mathbf{s}_t) \\ &= p(\mathbf{s}_1|\mathcal{O}_{1:T}) \prod_{t=1}^{T} p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) \pi(\mathbf{a}_t|\mathbf{s}_t) \end{aligned} \tag{21}$$

In case of exact inference derived in the last section, the match is exact, i.e. $D_{KL}(\hat{p}(\tau)||p(\tau)) = 0$. Therefore we can view the optimization objective as maximizing the negative KL divergence:

$$\begin{aligned}
&\max_{\pi} - D_{KL}(\hat{p}(\tau)||p(\tau)) \\
&\quad = \mathbb{E}_{\tau\sim\hat{p}}[\log p(\tau) - \log \hat{p}(\tau)] \\
&\quad = \mathbb{E}_{\tau\sim\hat{p}}\left[\log \frac{p(\mathbf{s}_1)}{p(\mathbf{s}_1)} + \sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t) - \log \pi(\mathbf{a}_t|\mathbf{s}_t) + \log \frac{p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)}{p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)}\right] \\
&\quad = \mathbb{E}_{\tau\sim\hat{p}}\left[\sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t) - \log \pi(\mathbf{a}_t|\mathbf{s}_t)\right] \\
&\quad = \sum_{t=1}^{T} \mathbb{E}_{\mathbf{s},\mathbf{a}\sim\hat{p}}[r(\mathbf{s}_t, \mathbf{a}_t) - \log \pi(\mathbf{a}_t|\mathbf{s}_t)] \\
&\quad = \sum_{t=1}^{T} \mathbb{E}_{\mathbf{s},\mathbf{a}\sim\hat{p}}[r(\mathbf{s}_t, \mathbf{a}_t)] + \mathbb{E}_{\mathbf{s}\sim\hat{p}}[\mathcal{H}(\pi(\mathbf{a}_t|\mathbf{s}_t))]
\end{aligned} \tag{22}$$

Now, the dynamics under deterministic conditions are given as -
$Q(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + V(\mathbf{s}_{t+1})$
and under stochastic conditions are given as -
$Q(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \log E_{\mathbf{s}_{t+1}\sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)}[\exp(V(\mathbf{s}_{t+1}))]$

Here, rather than having the optimistic term (which assumes that if any of the future states have a high reward regardless of the intermediary states that lead to there, the exponential term will favor that high-reward-state only), we would like to ask the question - given that a high reward was obtained in the past, what is the action probability, given that the transition probability did not change ?

3.2 Control via variational inference

To address the above mentioned issues, we would use Variational Inference, where the goal is to find $q(s_{1:T}, a_{1:T})$ such that it approximates $p(s_{1:T}, a_{1:T}|\mathcal{O}_{1:T})$ while the dynamics stays fixed to $p(s_{t+1}|s_t, a_t)$.

The distribution over optimal trajectories is given as $p(\tau) = \left[p(\mathbf{s}_1)\prod_{t=1}^{T} p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)\right] \exp\left(\sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t)\right)$ and the policy induced distribution is given as

$q(\tau) = q(\mathbf{s}_1)\prod_{t=1}^{T} p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)\, q(\mathbf{a}_t|\mathbf{s}_t)$

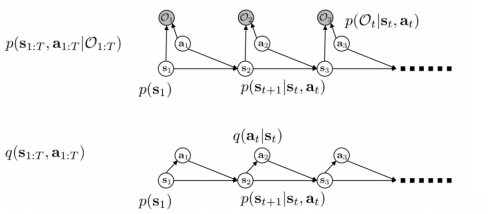


Figure 2: Graphical models with and without the optimality variables. Using variational inference, we try to find a variational distribution(bottom) that approximates the original distribution(above) well.

Hence, we can compute the Evidence Lower Bound as -

$$\begin{aligned}
\log \mathrm{p}\left(\mathcal{O}_{1:\mathrm{T}}\right) &= \log \iint \mathrm{p}\left(\mathcal{O}_{1:\mathrm{T}}, \mathbf{s}_{1:\mathrm{T}}, \mathbf{a}_{1:\mathrm{T}}\right) \mathrm{d}\mathbf{s}_{1:\mathrm{T}} \mathrm{d}\mathbf{a}_{1:\mathrm{T}} \\
&= \log \iint \mathrm{p}\left(\mathcal{O}_{1:\mathrm{T}}, \mathbf{s}_{1:\mathrm{T}}, \mathbf{a}_{1:\mathrm{T}}\right) \frac{\mathrm{q}\left(\mathbf{s}_{1:\mathrm{T}}, \mathbf{a}_{1:\mathrm{T}}\right)}{\mathrm{q}\left(\mathbf{s}_{1:T}, \mathbf{a}_{1:T}\right)} \mathrm{d}\mathbf{s}_{1:\mathrm{T}} \mathrm{d}\mathbf{a}_{1:T} \\
&= \log \mathbb{E}_{\left(\mathbf{s}_{1:T}, \mathbf{a}_{1:T}\right) \sim \mathbf{q}\left(\mathbf{s}_{1:T}, \mathbf{a}_{1:T}\right)}\left[\frac{p\left(\mathcal{O}_{1:T}, \mathbf{s}_{1:T}, \mathbf{a}_{1:T}\right)}{q\left(\mathbf{s}_{1:T}, \mathbf{a}_{1:T}\right)}\right] \\
&\geq \mathbb{E}_{\left(\mathrm{s}_1\right)}\left[\log \mathrm{p}\left(\mathcal{O}_{1:\mathrm{T}}, \mathbf{s}_{1:\mathrm{T}}, \mathbf{a}_{1:\mathrm{T}}\right) - \log \mathrm{q}\left(\mathbf{s}_{1:\mathrm{T}}, \mathbf{a}_{1:\mathrm{T}}\right)\right] \\
&= \mathbb{E}_{\tau \sim q}\left[\sum_{t=1}^{T} r\left(\mathbf{s}_t, \mathbf{a}_t\right) - \log q\left(\mathbf{a}_t | \mathbf{s}_t\right)\right] \\
&= \sum_{t=1}^{T} \mathbb{E}_{\left(s_t, \mathbf{a}_t\right) \sim q}\left[\mathbf{r}\left(\mathbf{s}_t, \mathbf{a}_t\right)\right] + H\left(\mathbf{q}\left(\mathbf{a}_t | \mathbf{s}_t\right)\right)
\end{aligned}$$

using Jensen's inequality to lower bound the log-probability of the observable variables. Hence, now the objective is composed of two components just like the deterministic case, but in terms of the variational distribution. The first term is the expected return induced by the variational policy, and the second term is the entropy of the variational policy. Now, to obtain the optimal policy, we have

$$q\left(\mathbf{s}_{1:T}, \mathbf{a}_{1:T}\right) = p\left(\mathbf{s}_1\right) \Pi_t p\left(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t\right) q\left(\mathbf{a}_t | \mathbf{s}_t\right) \tag{23}$$

$$\log p\left(\mathcal{O}_{1:T}\right) \geq \sum_t E_{\left(\mathbf{s}_t, \mathbf{a}_t\right) \sim q}\left[r\left(\mathbf{s}_t, \mathbf{a}_t\right) + \mathcal{H}\left(q\left(\mathbf{a}_t | \mathbf{s}_t\right)\right)\right] \tag{24}$$

Solving further, we have

$$q\left(\mathbf{a}_T | \mathbf{s}_T\right) = \arg\max E_{\mathbf{s}_T \sim q\left(\mathbf{s}_T\right)}\left[E_{\mathbf{a}_T \sim q\left(\mathbf{a}_T | \mathbf{s}_T\right)}\left[r\left(\mathbf{s}_T, \mathbf{a}_T\right)\right] + \mathcal{H}\left(q\left(\mathbf{a}_T | \mathbf{s}_T\right)\right)\right] \tag{25}$$

$$\arg\max E_{\mathbf{s}_T \sim q\left(\mathbf{s}_T\right)}\left[E_{\mathbf{a}_T \sim q\left(\mathbf{a}_T | \mathbf{s}_T\right)}\left[r\left(\mathbf{s}_T, \mathbf{a}_T\right) - \log q\left(\mathbf{a}_T | \mathbf{s}_T\right)\right]\right] \tag{26}$$

This is minimized when $q\left(\mathbf{a}_T | \mathbf{s}_T\right) \propto \exp\left(r\left(\mathbf{s}_T, \mathbf{a}_T\right)\right)$

$$q\left(\mathbf{a}_T | \mathbf{s}_T\right) = \frac{\exp\left(r\left(\mathbf{s}_T, \mathbf{a}_T\right)\right)}{\int \exp\left(r\left(\mathbf{s}_T, \mathbf{a}\right)\right) d\mathbf{a}} = \exp\left(Q\left(\mathbf{s}_T, \mathbf{a}_T\right) - V\left(\mathbf{s}_T\right)\right) \tag{27}$$

And the value function is given as -

$$V\left(\mathbf{s}_T\right) = \log \int \exp\left(Q\left(\mathbf{s}_T, \mathbf{a}_T\right)\right) d\mathbf{a}_T \tag{28}$$

10-708: Probabilistic Graphical Models, Spring 2020

Lecture 20: Reinforcement Learning and Control Through Inference in GM

Lecturer: Eric P. Xing *Scribe: Ankur Deka, Zuxin Liu, Yash Belhe, Tejus Gupta, Rohit Jena*

1 Control as Inference Recap

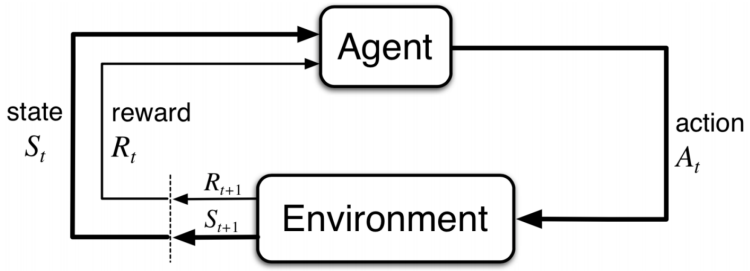


Figure 1: Reinforcement learning framework

Reinforcement learning is usually modeled as a Markov Decision Process(MDP), just as Fig. 1 shows. A typical MDP has 4 major components:

- The initial state distribution: $s_0 \sim p_0(s)$
- Transition probability: $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$
- Policy: $a_t \sim \pi(a_t|s_t)$
- Reward: $r_t = r(s_t, a_t)$

To represent the MDP with graphical model, we introduce an auxiliary variable $\mathcal{O}$ to define the distribution over optimal trajectories. The graphical model representation is shown in Fig. 2. We have:

- The initial state distribution: $s_0 \sim p_0(s)$
- Transition probability: $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$
- Policy: $a_t \sim \pi(a_t|s_t)$
- Reward: $r_t = r(s_t, a_t)$
- Optimality: $p(\mathcal{O}_t = 1|s_t, a_t) = \exp(r(s_t, a_t))$

The introduced auxiliary variable $\mathcal{O}$ allows us to model sub-optimal behavior and can be used to solve inverse reinforcement learning problem. The graphical model representation of RL provides us an alternative to solve control and planning problems via inference algorithms.

In the classical RL setup, we have the following equations for value function and Q-function:

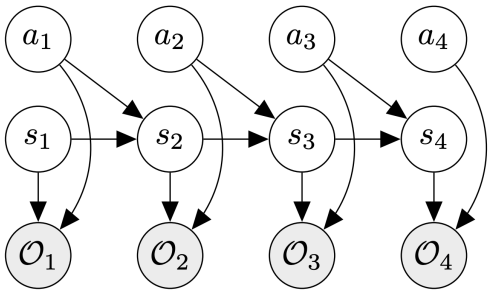


Figure 2: Graphical model representation of RL

$$V_\pi(s) := \mathbb{E}_\pi \sum_{k=0}^{T} \left[\gamma^k r_{t+k+1} \mid s_t = s\right] \tag{1}$$

$$Q_\pi(s,a) := \mathbb{E}_\pi \sum_{k=0}^{T} \left[\gamma^k r_{t+k+1} \mid s_t = s, a_t = a\right] \tag{2}$$

where γ is the discount factor, which represents the importance of rewards in future states.

The optimal value function (Bellman optimality) are:

$$V_*(s) := \max_\pi V_\pi(s) = \max_a \sum_{s'} p(s'|s,a)[r(s,a) + \gamma V_*(s')] \tag{3}$$

$$Q_*(s,a) := \max_\pi Q_\pi(s,a) = \sum_{s'} p(s'|s,a)[r(s,a) + \gamma \max_{a'} Q_*(s',a')] \tag{4}$$

The above Bellman update procedure can also be illustrated by Fig. 3.

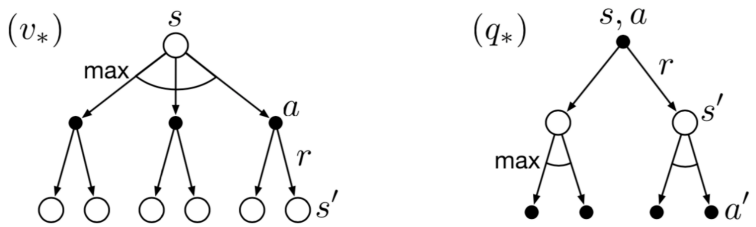


Figure 3: Backup diagrams for v_* and q_*

Given the optimal value function $Q_*(s,a)$, the optimal policy can be represented by:

$$\pi_*(a \mid s) := \delta\left(a = \underset{a}{\text{argmax}} Q_*(s,a)\right) \tag{5}$$

Let $V_t(s_t) = \log \beta_t(s_t), Q_t(s_t,a_t) = \log \beta_t(s_t,a_t)$. Denote $\tau = (s_1, a_1, \ldots, s_T, a_T)$ as the full trajectory. Denote $p(\tau) = p(\tau \mid \mathcal{O}_{1:T})$. Running inference in the graphical model as the Fig. 2 shows, we can compute:

$$p(\tau \mid \mathcal{O}_{1:T}) \propto p(s_t) \prod_{t=2}^{T} p(s_{t+1} \mid s_t, a_t) \times \exp(\sum_{t=1}^{T} r(s_t, a_t)) \tag{6}$$

Also, we know the following softmax relation:

$$V(s_t) = \log \int \exp(Q(s_t, a_t) + \log p(a_t \mid s_t)) da_t \tag{7}$$

We can also get:

$$p(a_t \mid s_t, \mathcal{O}_{1:T}) = exp(Q_t(s_t, a_t) - V_t(s_t)) \tag{8}$$

where we usually call $A_t(s_t, a_t) = Q_t(s_t, a_t) - V_t(s_t)$ as the advantage function.

The objective that the inference procedure want to optimize is the following KL divergence:

$$-D_{KL}(\hat{p}(\tau) \mid\mid p(\tau)) = \sum_{t=1}^{T} \mathbb{E}_{(s_t,a_t)\sim\hat{p}(s_t,a_t)} \left[r(s_t, a_t)\right] + \mathbb{E}_{s_t\sim\hat{p}(s_t)} \left[\mathcal{H}(\pi(a_t \mid s_t))\right] \tag{9}$$

where $\mathcal{H}(\pi(a_t \mid s_t))$ is the entropy of the policy. The first term is just the standard RL objective, while the second entropy term is used for regularization.

For deterministic dynamics, we can get this objective directly. For stochastic dynamics, we obtain it from the evidence lower bound.

2 Policy Gradients

In this section, we'll be looking at directly optimizing the standard RL objective function $E_{\tau\sim p_\theta(\tau)} \left[\sum_t r\left(\mathbf{s}_t, \mathbf{a}_t\right)\right]$. Here, θ are the parameters of our policy function i.e $\pi_\theta\left(\mathbf{a}|\mathbf{s}\right)$, so this amounts to finding the optimal policy function.

Let's first start off by defining the probability distribution over trajectories. The probability of any trajectory τ is given by:

$$p_\theta(\tau) = p_\theta\left(\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T\right) = p\left(\mathbf{s}_1\right) \prod_{t=1}^{T} \pi_\theta\left(\mathbf{a}_t|\mathbf{s}_t\right) p\left(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t\right) \tag{10}$$

Now, the optimal value of θ is that which maximizes our expected reward, i.e:

$$\theta^\star = \arg\max_\theta E_{\tau\sim p_\theta(\tau)} \left[\sum_t r\left(\mathbf{s}_t, \mathbf{a}_t\right)\right] \tag{11}$$

Now, we define the objective function $J(\theta)$ as

$$J(\theta) = E_{\tau\sim p_\theta(\tau)} \left[\sum_t r\left(\mathbf{s}_t, \mathbf{a}_t\right)\right] \tag{12}$$

Hence, the optimal θ is just that θ which maximizes the objective function

$$\theta^\star = \arg\max_\theta J(\theta) \tag{13}$$

We can estimate this objective function by drawing trajectories $\tau \sim p_\theta(\tau)$ and computing a Monte-Carlo estimate of this expectation

$$J(\theta) = E_{\tau \sim p_\theta(\tau)} \left[\sum_t r\left(\mathbf{s}_t, \mathbf{a}_t\right) \right] \approx \frac{1}{N} \sum_i \sum_t r\left(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}\right) \tag{14}$$

Now, we simply perform gradient ascent on the objective function $J(\theta)$ to optimize it.

In the approximate form of the objective function, there is no explicit dependence of $J(\theta)$ on the parameters θ. This may naively lead us to believe that the $\nabla_\theta J(\theta) = 0$.

This of course is not true, dependence on θ has instead been absorbed into the Monte-Carlo approximation of the estimation. To make this dependence explicit, we may write $\nabla_\theta J(\theta)$ as follows:

$$\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta E_{\tau \sim p_\theta(\tau)} \left[\sum_t r\left(\mathbf{s}_t, \mathbf{a}_t\right) \right] && (15)\\
&= \nabla_\theta \int p_\theta(\tau) \left[\sum_t r\left(\mathbf{s}_t, \mathbf{a}_t\right) \right] d\tau && (16)\\
&= \int \nabla_\theta p_\theta(\tau) \left[\sum_t r\left(\mathbf{s}_t, \mathbf{a}_t\right) \right] d\tau && (17)\\
&= \int \left[\nabla_\theta p_\theta(\tau)\right] \left[\sum_t r\left(\mathbf{s}_t, \mathbf{a}_t\right) \right] d\tau && (18)\\
& && (19)
\end{aligned}$$

Where the second step follows from the definition of the expectation, the third one is due to the linearity of the integration and the gradient operator and the fourth one from the fact that $[\sum_t r\left(\mathbf{s}_t, \mathbf{a}_t\right)]$ does not depend on θ.

In its current form, $\nabla_\theta J(\theta)$ seems hard to compute since there is no obvious Monte-Carlo estimate of this integral and $\nabla_\theta p_\theta(\tau)$ depends on the dynamics of the environment $p\left(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t\right)$ which we may not know.

However, we can use the log-gradient trick to easily estimate $J(\theta)$. More specifically:

$$\nabla_\theta \log p_\theta(\tau) = \frac{\nabla_\theta p_\theta(\tau)}{p_\theta(\tau)} \tag{20}$$

Hence, we can write $\nabla_\theta p_\theta(\tau)$ as:

$$\nabla_\theta p_\theta(\tau) = p_\theta(\tau) \nabla_\theta \log p_\theta(\tau) \tag{21}$$

Now, we can substitute this expansion of $\nabla_\theta p_\theta(\tau)$ into our expression for $\nabla_\theta J(\theta)$

$$\nabla_\theta J(\theta) = \nabla_\theta E_{\tau \sim p_\theta(\tau)} \left[\sum_t r\left(\mathbf{s}_t, \mathbf{a}_t\right) \right] \tag{22}$$

$$= \int \left[\nabla_\theta p_\theta(\tau)\right] \left[\sum_t r\left(\mathbf{s}_t, \mathbf{a}_t\right) \right] d\tau \tag{23}$$

$$= \int \left[p_\theta(\tau) \nabla_\theta \log p_\theta(\tau)\right] \left[\sum_t r\left(\mathbf{s}_t, \mathbf{a}_t\right) \right] d\tau \tag{24}$$

$$\tag{25}$$

Hence, we have written $\nabla_\theta J(\theta)$ in terms of an expectation over $p_\theta(\tau)$

$$\nabla_\theta J(\theta) = E_{\tau \sim p_\theta(\tau)} \left[\nabla_\theta \log p_\theta(\tau) \sum_t r\left(\mathbf{s}_t, \mathbf{a}_t\right) \right] \tag{26}$$

$$\tag{27}$$

Let's try to evaluate $\nabla_\theta \log p_\theta(\tau)$ by first writing out $\log p_\theta(\tau)$

$$\log p_\theta(\tau) = \log p\left(\mathbf{s}_1\right) + \sum_t \log \pi_\theta\left(\mathbf{a}_t | \mathbf{s}_t\right) + \log p\left(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t\right) \tag{28}$$

Which gives us $\nabla_\theta \log p_\theta(\tau)$ as:

$$\nabla_\theta \log p_\theta(\tau) = \sum_t \nabla_\theta \log \pi_\theta\left(\mathbf{a}_t | \mathbf{s}_t\right) \tag{29}$$

Finally, substituting the expression for $\nabla_\theta \log p_\theta(\tau)$ into the expression for $\nabla_\theta J(\theta)$, we have:

$$\nabla_\theta J(\theta) = E_{\tau \sim p_\theta(\tau)} \left[\left(\sum_t \nabla_\theta \log \pi_\theta\left(\mathbf{a}_t | \mathbf{s}_t\right) \right) \left(\sum_t r\left(\mathbf{s}_t, \mathbf{a}_t\right) \right) \right] \tag{30}$$

$$\tag{31}$$

Once again, we can estimate this expectation using a Monte-Carlo average by drawing sample trajectories $\tau \sim p_\theta(\tau)$:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_t \nabla_\theta \log \pi_\theta\left(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}\right) \right) \left(\sum_t r\left(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}\right) \right) \tag{32}$$

We can now update our estimate of θ by performing gradient ascent:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta) \tag{33}$$

Looking at this expression, it is evident that the update rule is trying to up-weight those trajectories with higher total rewards (as $\sum_t r\left(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}\right)$ is higher for them) and suppress those trajectories with lower total rewards (as $\sum_t r\left(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}\right)$ is lower for them).

Putting all of these steps together, we have the REINFORCE algorithm:

Algorithm 1 REINFORCE Algorithm

1. sample $\left\{\tau^i\right\}$ from $\pi_\theta\left(\mathbf{a}_t | \mathbf{s}_t\right)$
2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_t \nabla_\theta \log \pi_\theta\left(\mathbf{a}_t^i | \mathbf{s}_t^i\right)\right) \left(\sum_t r\left(\mathbf{s}_t^i, \mathbf{a}_t^i\right)\right)$
3. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

3 Value Based Reinforcement Learning

Instead of explicitly learning the policy of a reinforcement learning agent, we can learn the optimum value function and retrieve the optimal policy from it. If we the value of $Q^*(s,a)\forall s,a$, we can obtain the opitmal policy as $\pi(a|s) = \delta(a = \text{argmax}_{a'}Q(s,a'))$.

3.1 Policy Iteration

Policy iteration iterates over two steps as shown in Algotihm 4. The first step, policy evaluation, iteratively evaluates the Q_π function of the policy using bellman update.

$$Q_\pi(s,a) \leftarrow r(s,a) + \sum_{s'} \gamma p(s'|s,a)V(s')$$
$$\equiv Q_\pi(s,a) \leftarrow r(s,a) + \sum_{s'} \gamma p(s'|s,a) \sum_{a'} \pi(a'|s')Q_\pi(s',a')$$

The second step greedily updates the policy by taking the action with the highest Q_π value.

Algorithm 2 Policy Iteration

Initialize random $\pi(a|s), Q(s,a), V(s)\forall s,a$
Repeat until convergence:

1. Policy Evaluation
   Repeat until convergence: $Q_\pi(s,a) \leftarrow r(s,a) + \sum_{s'} \gamma p(s'|s,a) \sum_{a'} \pi(a'|s')Q_\pi(s',a')$
2. Policy Improvement
   $\pi(a|s) \leftarrow \delta\left(a = \text{argmax}_{a'}Q(s,a')\right)$

Policy improvement step is guaranteed to be at least as good as the current policy. This can be intuitively understood as follows. Let's say we take action $a_1 = \text{argmax}_a(Q_\pi(s,a))$ in the first step and follow policy π from there on. Doing so is better than or at least as good following policy π from the beginning because we took the action with highest Q_π value. We can then extend this argument when we transition to state s_2 from s_1. That is, we choose action $a_2\text{argmax}_a(Q_\pi(s_2,a))$ from state s_2 and follow policy π from there on. Likewise, following the updated policy at every step is guaranteed to be at least as good as the current policy.

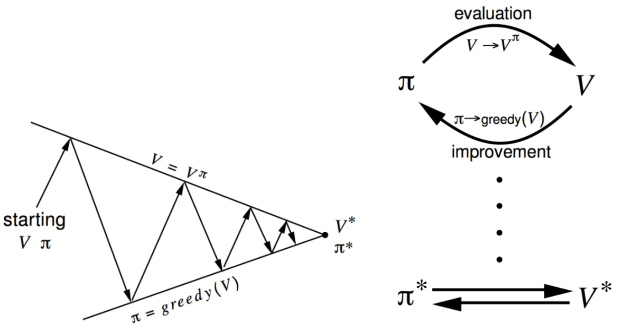


Figure 4: Left: Value function and policy are iteratively updated and at the end they converge to the optimal values. *Source: UCL Course on RL* [Link]

Fig. 4 shows policy iteration pictorially. Policy evaluation step evaluates the correct value function. A greedy update on the policy improves the policy but the value function is no longer correct. After multiple iterative steps both of these converge to the true values.

3.2 Value Iteration

Is there a way we could avoid explicitly representing the policy and perform reinforcement learning based only on value functions. The answer is a simple extension to the policy iteration algorithm we have just seen. We can combine the policy evaluation and policy improvement steps into a single step.

In the bellman update, we substitute $V(s')$ with $\max_{a'} Q(s', a')$

Algorithm 3 Value Iteration

Initialize random $Q(s, a) \forall s, a$
Repeat until convergence:
$Q(s, a) \leftarrow r(s, a) + \sum_{s'} \gamma p(s'|s, a) \max_{a'} Q(s', a')$

Due to the substitution, we can represent the update equation purely using value function. There is one key difference between policy iteration and value iteration. Policy iteration updates the Q value for multiples steps until convergence and then performs one single greedy update of the policy. Value iteration does a single update on the value function for every greedy update of the policy.

3.3 Fitted Q Iteration

Policy iteration and value iteration are applicable only for small sized discrete state-spaces. For an environment with S states and A actions per state, we need to store and update $S \times A$ Q values.

For large or continuous state spaces, we can approximate the value function Q with a function approximator with parameters θ. We can minimize the error:

$$\min_{\theta} \mathbb{E}_{a \sim \pi} ||Q^{\theta}(s, a) - y||$$
$$\text{where, } y = (r(s, a) + \gamma \max_{a'} Q^{\theta}(s', a')), \text{ is called the target}$$

Here $\pi(a|s) = \delta(a = \text{argmax}_{a'} Q(s, a'))$. We can minimize this objective with stochastic gradient descent. While updating the parameters we don't take the gradient of the target with respect to the parameters θ. We can them perform a greedy update on the policy as in policy iteration.

Algorithm 4 Fitted Q-Iteration

Initialize random $\pi(a|s), Q(s, a), V(s) \forall s, a$
Repeat until convergence:
1. Policy Evaluation
   $\min_{\theta} \mathbb{E}_{a \sim \pi} ||Q^{\theta}(s, a) - (r(s, a) + \gamma \max_{a'} Q(s', a'))||$
2. Policy Improvement
   $\pi(a|s) \leftarrow \delta\left(a = \text{argmax}_{a'} Q(s, a')\right)$

In practice, fitted Q-learning is very unstable and there are numerous tricks used to stabilize it. In particular, Deep Q Network (DQN, Mnih et al.) introduces a replay buffer and delayed parameters for the target network.

4 Soft Policy Gradient and Soft Q-learning

The soft-policy gradient is written as:

$$J(\theta) = \sum_{t=1}^{T} \mathbb{E}_{(s_t,a_t)\sim p(s_t,a_t)} \left[r(s_t,a_t)\right] + \mathbb{E}_{s_t\sim p(s_t)} \left[\mathcal{H}(\pi_\theta(a_t|s_t))\right] \tag{34}$$

$$= \sum_{t=1}^{T} \mathbb{E}_{(s_t,a_t)\sim p_\theta(s_t,a_t)} \left[r(s_t,a_t) - \log(\pi_\theta(a_t|s_t))\right] \tag{35}$$

To calculate the gradient of the second term, we compute the following (using an expectation over trajectories):

$$\begin{aligned}
&\nabla_\theta \sum_{t=1}^{T} \mathbb{E}_{(s_t,a_t)\sim p_\theta(s_t,a_t)} \left[\log(\pi(a_t|s_t))\right] \\
&= \int \nabla_\theta \left[p(\tau) \sum_{t=1}^{T} \log(\pi_\theta(a_t|s_t))\right] d\tau \\
&= \int \left[\nabla_\theta p(\tau) \sum_{t=1}^{T} \log(\pi_\theta(a_t|s_t)) + p(\tau)\nabla_\theta \sum_{t=1}^{T} \log(\pi_\theta(a_t|s_t))\right] d\tau \quad \text{(Using chain rule)} \\
&= \int \left[p(\tau)\nabla_\theta \log p(\tau) \sum_{t=1}^{T} \log(\pi_\theta(a_t|s_t)) + p(\tau) \sum_{t=1}^{T} \nabla_\theta \log(\pi_\theta(a_t|s_t))\right] d\tau \\
&= \int p(\tau)\nabla_\theta \log(p(\tau)) \left[\sum_{t=1}^{T} \log \pi_\theta(a_t|s_t) + 1\right] d\tau
\end{aligned}$$

From the backward messages in RL (previous lecture), recall that

$$Q_t(s_t,a_t) = \log(\beta_t(s_t,a_t)) = r(s_t,a_t) + \log(\mathbb{E}_{s_{t+1}\sim p(s_{t+1},a_{t+1})} \left[\exp(V_{t+1}(s_{t+1}))\right])$$

$$V_t(s_t) = \log(\beta_t(s_t)) = \log\left(\int \exp(Q_t(s_t,a_t))da_t\right)$$

From these two equations, it is easy to see that

$$\pi(a_t|s_t) = \frac{\beta_t(s_t,a_t)}{\beta_t(s_t)} = \exp\left(Q_t(s_t,a_t) - V(s_t)\right)$$

To get rid of the $p(\tau)$ term over trajectories, we approximate it using sampling N trajectories. Expanding Equation (35) and taking derivatives with respect to the parameters θ we get:

$$\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta \sum_{t=1}^{T} \mathbb{E}_{(s_t,a_t)\sim p_\theta(s_t,a_t)} \left[r(s_t,a_t) - \log \pi_\theta(a_t|s_t)\right] \\
&\approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \left[r(s_t,a_t) + \left(\sum_{t'=t+1}^{T} r(s_{t'},a_{t'}) - \log(\pi_\theta(a_{t'}|s_{t'}))\right) - \log(\pi_\theta(a_t|s_t)) - 1\right] \\
&= \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \left(\nabla_\theta Q_\theta(s_t,a_t) - \nabla_\theta V_\theta(s_t)\right) \left[r(s_t,a_t) + Q_\theta(s_{t+1},a_{t+1}) - Q_\theta(s_t,a_t) + V(s_t)\right]
\end{aligned}$$

Since the term inside the inner brackets containing the terms of t' becomes $Q_\theta(s_{t+1}, a_{t+1})$. Expanding the terms related to V_θ we get:

$$= \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\left(\nabla_\theta Q_\theta(s_t,a_t) - \nabla_\theta V_\theta(s_t)\right)\left[r(s_t,a_t) + Q_\theta(s_{t+1},a_{t+1}) - Q_\theta(s_t,a_t) + V(s_t)\right]$$
$$= \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta Q_\theta(s_t,a_t)\left[r(s_t,a_t) + \text{softmax}_{a_{t+1}} Q_\theta(s_{t+1},a_{t+1}) - Q_\theta(s_t,a_t)\right]$$

This equation results in a very similar update to Q-learning:

$$\theta \leftarrow \theta + \alpha\nabla_\theta Q_\theta(s,a)\left(r(s,a) + \gamma V(s') - Q_\theta(s,a)\right)$$

where the value function $V(s')$ is defined as:

$$V(s') = \log\int \exp(Q_\theta(s',a'))da'$$

We can technically add a temperature parameter β inside the exp of the Q-function such as $\exp\left(\frac{Q(s,a)}{\beta}\right)$. Higher values of β correspond to more 'random' policies and β close to 0 means a less stocastic policy.

Soft optimality has many benefits. Some of these benefits are listed below:

- Improves exploration and prevents entropy collapse.
- Easier to finetune policies for specific tasks. This is an empirical observation.
- Better robustness (due to wider coverage of states).
- Reduced to hard optimalty by increasing magnitude of rewards.
- This is a good model of human behavior.

10-708: Probabilistic Graphical Models, Spring 2020

21: Gaussian Processes

Lecturer: Eric Xing *Scribe: Aman Jakhar, Chang Shi, Adam Smoulder, Maxwell Wang, Ni Zhan*

1 Introduction

Assumption-based model approach vs. Probabilistic approach Assumption-based model approaches directly fit a funtion from the data without worrying about how the data is generated, while probailistic approaches provide more explicit guidance on how the data is generated and how the errors behave.

Assumption-based model approach: We guess the parametric form of a function that could fit the data

- $f(x, \boldsymbol{w}) = \boldsymbol{w}^{\mathrm{T}} x$ [Linear function of $\boldsymbol{w}$ and x]
- $f(x, \boldsymbol{w}) = \boldsymbol{w}^{\mathrm{T}} \phi(x)$ [Linear function of $\boldsymbol{w}$] (Linear Basis Function Model)
- $f(x, \boldsymbol{w}) = g\left(\boldsymbol{w}^{\mathrm{T}} \phi(x)\right)$ [Non-linear in x and $\boldsymbol{w}$] (E.g., Neural Network)

$\phi(x)$ is a vector of basis functions. For example, if $\phi(x) = \left(1, x, x^2\right)$ and $x \in \mathbb{R}^1$, $f(x, \boldsymbol{w}) = w_0 + w_1 x + w_2 x^2$ is a quadratic function.

Then choose an error measure $E(\boldsymbol{w})$, minimize with respect to $\boldsymbol{w}$ to estimate the parameters

$$E(\boldsymbol{w}) = \sum_{i=1}^{N} \left[f\left(x_i, \boldsymbol{w}\right) - y\left(x_i\right)\right]^2$$

Probabilistic approach: We could explicitly account for noise in our model.

$$y(x) = f(x, \boldsymbol{w}) + \epsilon(x), \text{ where } \epsilon(x) \text{ is a noise function.}$$

One commonly takes $\epsilon(x) = \mathcal{N}\left(0, \sigma^2\right)$ for i.i.d. additive Gaussian noise, in which case

$$p\left(y(x) | x, \boldsymbol{w}, \sigma^2\right) = \mathcal{N}\left(y(x); f(x, \boldsymbol{w}), \sigma^2\right) \qquad \text{Observation Model}$$

$$p\left(\boldsymbol{y} | x, \boldsymbol{w}, \sigma^2\right) = \prod_{i=1}^{N} \mathcal{N}\left(y\left(x_i\right); f\left(x_i, \boldsymbol{w}\right), \sigma^2\right) \qquad \text{Likelihood}$$

Then maximize the likelihood of the data $p\left(\boldsymbol{y} | x, \boldsymbol{w}, \sigma^2\right)$ with respect to $\sigma^2, \boldsymbol{w}$ tp estimate the parameters.

The probabilistic approach helps us interpret the error measure in a deterministic approach, and gives us a sense of the noise level σ^2 Probabilistic methods thus provide an intuitive framework for representing uncertainty, and model development. In fact, it is also possible to model how x is distributed, thus more flexibility is offered.

Regularization Both assumption-based model approach and probabilistic approach are prone to *over-fitting* for flexible $f(x, \boldsymbol{w})$: low error on the training data, high error on the test set. More complex model gives

more flexibility to accomodate the irregular patterns in data, therefore increases the risk of over-fitting on data, thus regularization is introduced to penalize the complexity of the models.

Use a penalized log likelihood(or error function), such as

$$\log p(\boldsymbol{y}|X,\boldsymbol{w}) \propto \overbrace{-\frac{1}{2\sigma^2}\sum_{i=1}^{n}\left(f\left(x_i,\boldsymbol{w}\right)-y\left(x_i\right)^2\right)}^{\text{model fit}} \quad \overbrace{-\lambda \boldsymbol{w}^{\mathrm{T}}\boldsymbol{w}}^{\text{complexity penalty}}$$

However, there are ambiguity about how we define complexity and how much we should penalize complexity. While choosing the penalty form is very ad-hoc, λ can be set using *cross-validation.*

Bayesian Inference Since these heuristics are ad-hoc, a more rational way of fitting models with the data while taking care of model complexity and penalty would be Bayesian Inference.

According to the Bayes' Rule,

$$p(a|b) = p(b|a)p(a)/p(b) \qquad (\text{posterior} = \frac{\text{likelihood} \times \text{prior}}{\text{marginal likelihood}})$$

we get $p(a|b) \propto p(b|a)p(a)$,

$$p\left(\boldsymbol{w}|\boldsymbol{y},X,\sigma^2\right) = \frac{p\left(\boldsymbol{y}|X,\boldsymbol{w},\sigma^2\right)p(\boldsymbol{w})}{p\left(\boldsymbol{y}|X,\sigma^2\right)}$$

Then the predictive distribution is

$$p\left(y|x_*,\boldsymbol{y},X\right) = \int p\left(y|x_*,\boldsymbol{w}\right)p(\boldsymbol{w}|\boldsymbol{y},X)d\boldsymbol{w}$$

Because it's an average of infinitely many models weighted by their posterior probabilities, thereis no over-fitting,and complexity is automatically calibrated. Typically we are more interested in distribution over functions than in parameters $\boldsymbol{w}$.

2 Nonparametric Bayesian Inference and Gaussian Processes

Gaussian Processes are nonparametric Bayesian inference models under particular conditions. In this section, we first review the differences between parametric and nonparametric frameworks, develop intuition of Bayesian (non)parametric inference, then define the Gaussian Process.

Parametric vs. Nonparametric models Recall that parametric models work by assuming that the present data can be explained by a fixed and finite number of parameters, with residuals being explained by noise terms. For example, simple linear regression assumes that the output is a linear combination of the inputs with Gaussian noise added. While the complexity of parametric models can greatly vary (logistic regression versus artifical neural networks), the principle of a fixed parameter set remains.

Conversely, in nonparametric models, the number of parameters typically grows with sample size. Consider kernel regression, where the expected output for a given input is a weighted average of the data outputs, where the weights are determined by distance of the given input from the data inputs. The more data that become available, the more parameters are present, the more complex the operation gets. While this increased complexity with more data can pose computational problems, nonparametric models avoid making the many assumptions enforced by parametric models.

Parametric vs. Nonparametric Bayesian Inference Parametric Bayesian inference specifically refers to modeling distributions for variables (as opposed to just variables) over a finite parameter set using Bayes Rule. Consider most examples, where parameters follow some prior distribution and the likelihood depends on these parameters.

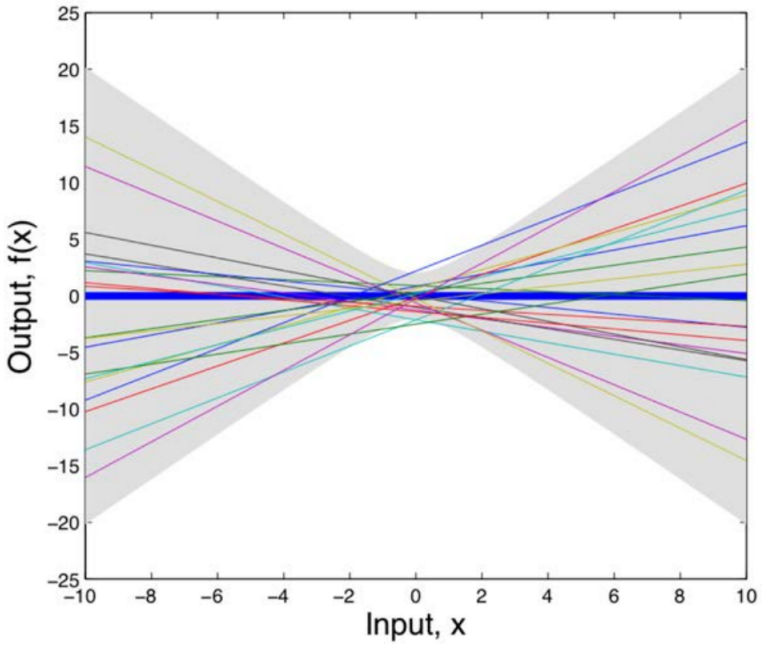
What if, however, the model has an infinite set of parameters? Bayesian nonparametrics are Bayesian models where these finite-dimensional parameters are replaced by a stochastic process. Finite datasets will ultimately only need to use a finite number of parameters, and the rest will be integrated out.

Weight-Space and Function-Space Views Consider a single linear regression model:

$$f(x) = a_0 + a_1 x$$

$$a_0, a_1 \sim N(0, 1)$$

In this case, we can see many possible lines that construct a "weight space", as seen in the figure below:



However, we are more interested in the distribution over functions that is caused by the parameter distributions (a "function-space" view). In the case of this example, we can very clearly determine this distribution. Namely, since both parameters are standard normal, we know:

$$E[f(x|a_0, a_1)] = 0 + 0x = 0$$

$$cov[f(x_b|a_0, a_1), f(x_c|a_0, a_1)] = 1 + x_b x_c$$

This means that we can model our output as a joint Gaussian distribution:

$$[f(x_1), ..., f(x_n)] \sim N(0, K)$$

$$K_{ij} = cov(f(x_i), f(x_j)) = k(x_i, x_j) = 1 + x_i x_j$$

Crucially, we are assuming x is given and not making any statements about the randomness of x. This distribution is due to the randomness in the function. This is a very specific example of a Bayesian Nonparametric approach, but we can generalize this to define Gaussian Processes.

Gaussian Process Definition A Gaussian Process is a collection of random variables, where any finite number of them have a joint Gaussian distribution. A function f is a Gaussian Process with mean function $m(x)$ and covariance kernel $k(x_i, x_j$ if:

$$[f(x_1), ..., f(x_n)] \sim N(\mu, K)$$
$$\mu_i = m(x_i)$$
$$K_{ij} = k(x_i, x_j)$$

Linear Basis Function Models A slightly more complicated example of a Gaussian Process than the one above is case of linear basis functions. Consider:

$$f(x, \mathbf{w}) = \mathbf{w}^T \phi(x)$$
$$p(\mathbf{w}) = N(0, \Sigma_w)$$

Recall that we assume x is given, and hence not random. To define our Gaussian process, determine the mean and covariance for the induced distributions over functions:

$$m(x) = E[f(x, \mathbf{w})] = E[\mathbf{w}^T]\phi(x) = 0$$
$$k(x_i, x_j) = cov(f(x_i), f(x_j)) = E[f(x_i)f(x_j)] - E[f(x_i)]E[f(x_j)]$$
$$= \phi(x_i)^T E[\mathbf{w}\mathbf{w}^T]\phi(x_j) - 0$$
$$k(x_i, x_j) = \phi(x_i)^T \Sigma_w \phi(x_j)$$

3 Gaussian Processes

3.1 Gaussian Process Graphical model

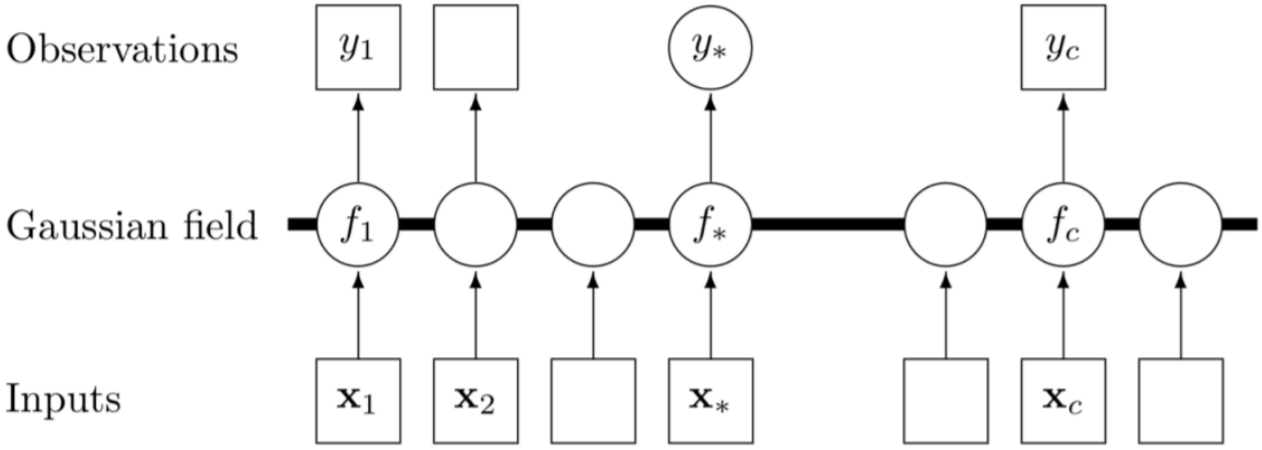


Figure: Gaussian process graphical model

In the above chart y_i represent the observations and x_i represent the inputs. The functions f_i belong to the Gaussian field. When posterior inference is done f_is act as random variables and are integrated out, which means that every prediction y_* depends on all the other inputs and observations.

3.2 Example: RBF kernel

Radial Basis Function (RBF) kernel is by far the most used and popular kernel. It expresess the intuition that the data points which are close together in some distance measure are more correlated. The kerner's hyperparameters a and l control the amplitude and wiggliness (variance) of these functions. These parameters can be learned by optimization through cross validation. If we have extremely large support of x we can approximate any function, i.e. GP with RBF kernel are universal approximators. However such a large covariance matrix presents computational problems. The RBF kernel can be formulated as:

$$k_{RBF}(x_1, x_2) = cov(f(x_1), f(x_2)) = a^2 exp(-\frac{||x_1 - x_2||^2}{2l^2})$$

The hyperparameter l controls how quickly covariance decays with L_2 distance. The impact of length scale on the covariance is shown in the below figure. When the window is very small the covariance drops very quickly.

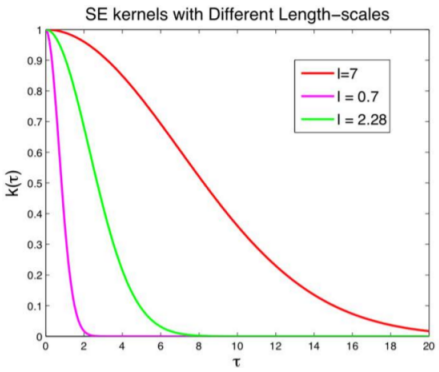


Figure: RBF kernel covariance as a function of distance with varying length scale

This RBF kernel can then be used to create prior distributions which can be used subsequently for modelling and predictions. A Gaussian prior with a mean of 0 and covariance matrix as identity matrix may be constructed as follows:

$$(X_1, X_2 ..... X_n) \sim \ GP(0, \Sigma)$$

Various sample of this joint distribution look like the below figure. The shaded region represents the uncertainity associated with each sample, whereas the blue line represents the mean function.

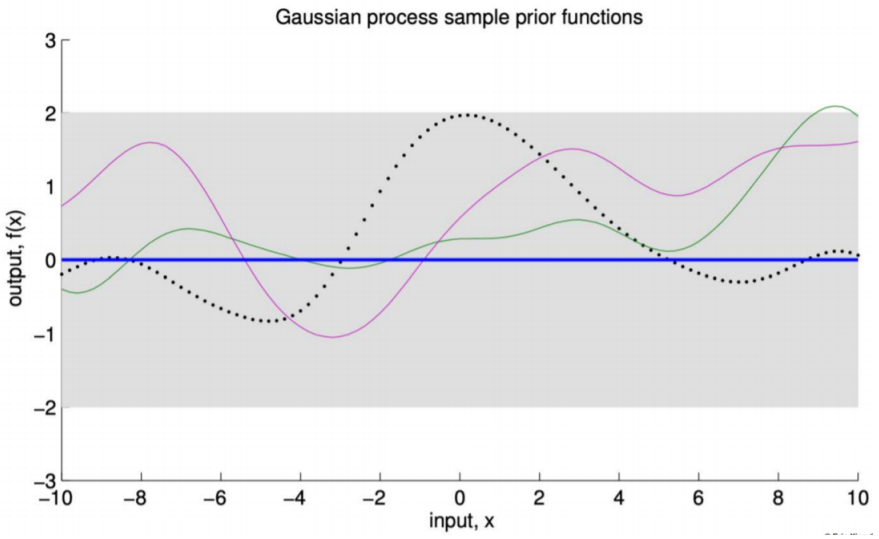


Figure: Different samples of $GP(0, \Sigma)$

3.3 Gaussian Process Inference

The process for inference for a Gaussian Process can be summarized as:

1. Observe noisy data $\mathbf{y} = (y(x_1), y(x_2)....y(x_N))^T$ at input locations X
2. Begin with the standard regression assumption:

$$y(x) = f(x) + \epsilon^2$$

   where ϵ is normally distributed

3. Place a Gaussian process distribution over noise free functions $f(x) \sim GP(0, k_\theta)$. The kernel is parameterized by θ
4. For a new observation X_* infer $p(\mathbf{f}_*|\mathbf{y}, X, X_*)$ which may be decomposed as:

$$p(\mathbf{y}_*|\mathbf{f}_*, X, X_*) = p(\mathbf{y}_*|\mathbf{f}_*)p(\mathbf{f}_*|X, X_*)$$

The conditional distribution for a multivariate Gaussian respresenting the joint ditribution of training data and some extra data point can be written as:

$$\begin{bmatrix}\mathbf{y}\\ \mathbf{f}_*\end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix}K_\theta(X,X) + \sigma^2 I & K_\theta(X, X_*)\\ K_\theta(X_*, X) & K_\theta(X_*, X_*)\end{bmatrix}\right)$$

Using this joint distribution one may extract the conditional distribution for f_* which may be written as:

$$\begin{gathered}\mathbf{f}_*|X_*, X, \mathbf{y}, \theta \sim \mathcal{N}(\bar{\mathbf{f}_*}, cov(\mathbf{f}_*))\\ \bar{\mathbf{f}_*} = K_\theta(X_*, X)[K_\theta(X,X) + \sigma^2 I]^{-1}\mathbf{y}\\ cov(\mathbf{f}_*) = K_\theta(X_*, X_*) - k_\theta(X_*, X)[K_\theta(X,X) + \sigma^2 I]^{-1}K_\theta(X, X_*)\end{gathered}$$

This seems like an expensive calculation with an $N \times N$ matrix inversion.The below figure shows how the posterior of the GP is updated as we observe new data points. Note that variance at the data points is exactly zero.

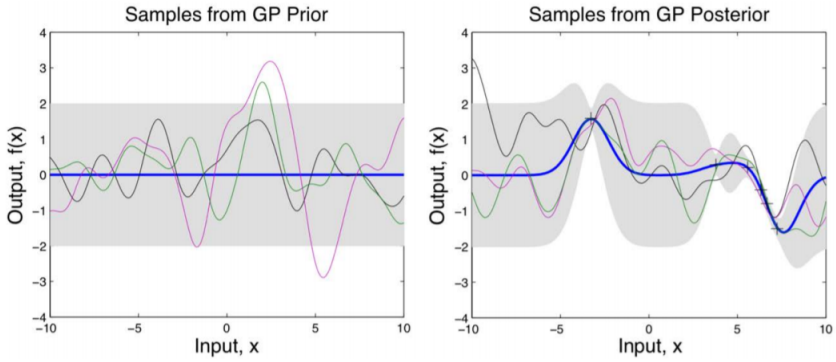


Figure: Change in posterior after introduction of data points

Increasing the length scale has effects similar to those we described in an earlier figure of the RBF kernel. Larger window size of the kernel function allows the points to influence more distant points. The figure below shows that:

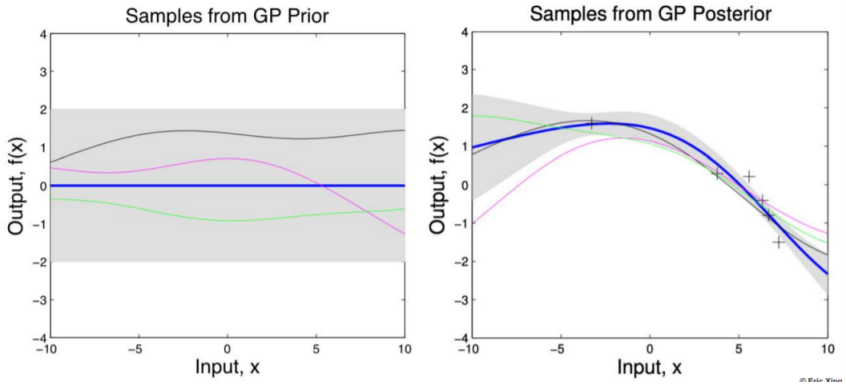


Figure: GP dependence on increasing the length scale l

We realize that we are not specifying the paramateric form of $f(x)$ and are able to utilize the entire training data which are the main advantages of the Gaussian process.

3.4 Gaussian Process Learning

As mentioned before the entire Gaussian process $f(x)$ can be integrated away to obtain the marginal likelihood as a function of kernel hyperparameters alone. This may be written as:

$$p(\mathbf{y}|\theta, X) = \int p(\mathbf{y}|\mathbf{f}, X)p(\mathbf{f}|\theta, X)d\mathbf{f}$$

The log likelihood of this may be written as:

$$logp(\mathbf{y}|\theta, X) = -\frac{1}{2}\mathbf{y}^T(K_\theta + \sigma^2\mathbf{I})^{-1} - \frac{1}{2}log|K_\theta + \sigma^2\mathbf{I}| - \frac{N}{2}log(2\pi)$$

This log likelihood may be minimized over θ to obtain the optimal value. Note the extra complexity penalty in the above term. Various other types of kernels are also used. Some of which are listed below:

1. $k_{SE}(\tau) = exp(-0.5\frac{\tau^2}{l2})$
2. $k_{MA}(\tau) = a(1 + \frac{\sqrt{3}\tau}{l})exp(-\frac{\sqrt{3}}{l})$
3. $k_{RQ}(\tau) = (1 + \frac{\tau^2}{2\alpha l^2})^{-\alpha}$
4. $k_{PE}(\tau) = exp(-2\frac{\sin^2(\pi\tau\omega)}{l^2})$

4 Deep Kernel Learning

We can use GP in conjunction with other techniques. GP itself is a particular way of transforming an input to y. GP gives uncertainty and transforms to a whole distribution, and the function is implicit and has rich representation. We can treat GP as a regular transformer and use it inside an existing transformer framework. For example, in deep kernel learning, we insert a GP layer into a neural network [1], Fig. 1. The GP layer uses a RBF kernel, which achieves the effect of having an infinite dimension NN layer. In NN theory, wider dimensional layers are more expressive for latent representations and allow more influence and correspondences between dimensions. (Here we are referring to dimension of a single data point.) We do not need to plug in an infinite dimension layer but we achieve this effect with the GP layer.

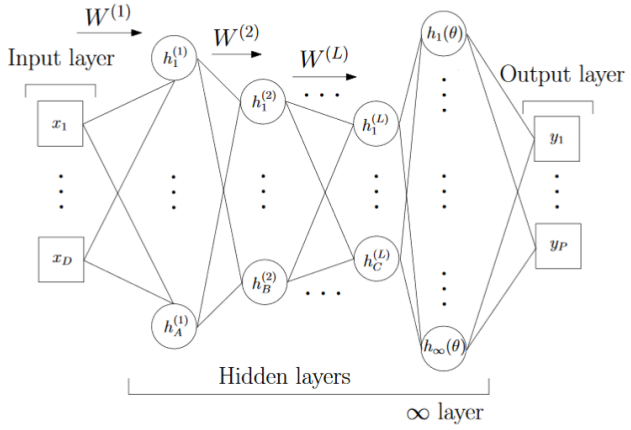


Figure 1: NN with GP layer

We jointly learn the parameters using back-propagation and chain rule. We take derivative of loss function w.r.t. NN parameters and hyperparameters inside GP.

$$\frac{\partial L}{\partial \theta} = \frac{\partial L}{\partial K_\gamma}\frac{\partial K_\gamma}{\partial \theta}, \quad \frac{\partial L}{\partial \mathbf{w}} = \frac{\partial L}{\partial K_\gamma}\frac{\partial K_\gamma}{\partial g(\mathbf{x}, \mathbf{w})}\frac{\partial g(\mathbf{x}, \mathbf{w})}{\partial \mathbf{w}}$$

where θ are parameters of base kernel, $\mathbf{w}$ are weights of network, and L is log marginal likelihood of the GP.

The deep kernel has advantages in regression. In a comparison between GP, deep NN, and deep NN with infinity layer, across multiple datasets, many of the best performing models (in terms of error and variance) were deep kernel learning models. (In Table 1 in paper [1], bold numbers were considered best performance.)

DKL on Sequential Data We can combine GP with sequential model such as RNN or LSTM. In a regular sequential model, we have fixed length input that generates one output. When inputing forward through the sequence, we must drop the earlier part of sequence because the model takes fixed length input. The outputs do not use information from the entire sequence. We can add a GP layer that uses latent representations from h as inputs and maps to the output, as shown in Fig. 2. With the added GP layer, we make use of all inputs up to that point in the sequence. The predictions are derived using conditional GP with mean and covariance as previously described. We get benefits from both sequential NN and GP (which gives correlation and uncertainty). Details in paper [2].

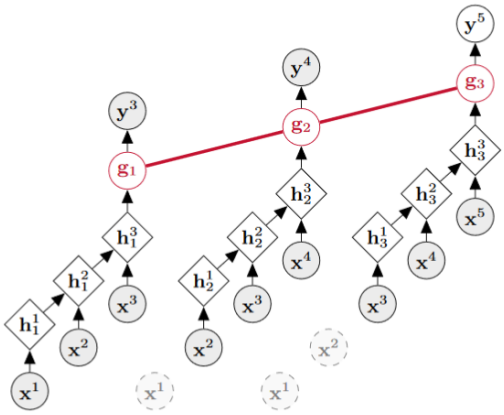


Figure 2: LSTM with GP layer

An example application is autonomous driving.

- We want to predict the lane location in the front, shown in Fig. 3. There is a visible difference in prediction quality/accuracy, with GP-LSTM having higher accuracy. The GP-LSTM also gives error bound (uncertainty) on prediction, not just point prediction.
- Predict lead vehicle distance in Fig. 4. Comparing LSTM and GP-LSTM, the GP-LSTM is higher accuracy and provides uncertainty. The uncertainty is useful because we can take additional actions based on uncertainty. When uncertainty is large, we may want to stay away from the driver.

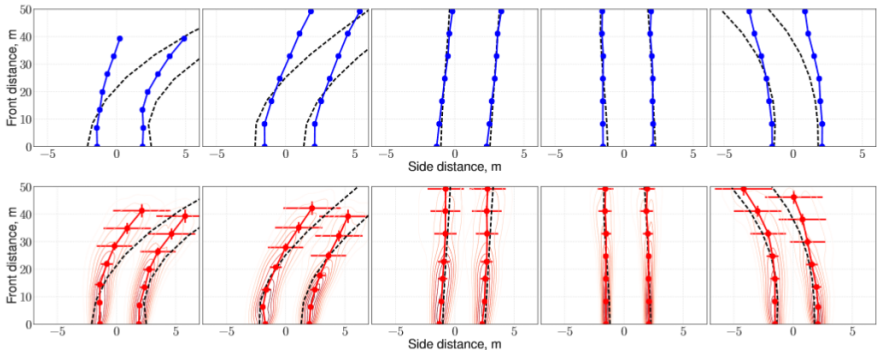


Figure 3: Lane prediction: Blue is standard LSTM, and red is GP-LSTM. Dots indicate prediction.

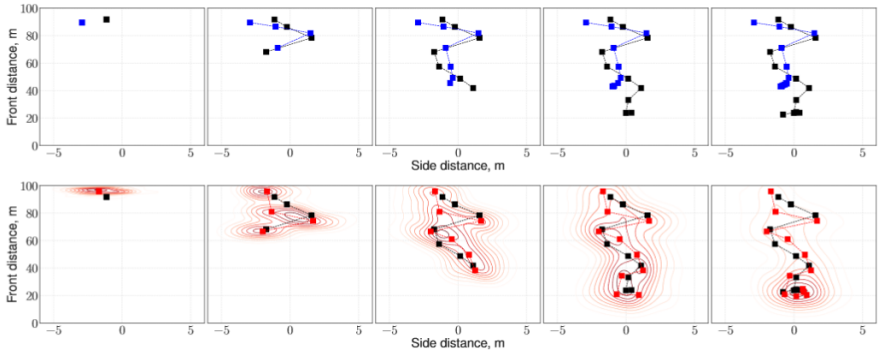


Figure 4: Lead vehicle prediction: Blue is standard LSTM, and red is GP-LSTM. GP-LSTM curves represent uncertainty.

5 Scalability Issues and Solutions

As previously mentioned, Gaussian Processes are a form of nonparametric models, meaning that complexity will increase with the number of data. In this section, we identify scalability issues and identify possible methods of circumventing these.

The Scalability Issue There are bottlenecks in both inference and learning for Gaussian Processes. In inference, the limiting step comes due to the $n \times n$ kernel matrix inversion involved in predicting both the mean and covariance, namely $(K_\theta(X,X) + \sigma^2 I)^{-1}$. This requires $\mathcal{O}(n^3)$ operations and $\mathcal{O}(n^2)$ storage. Similarly for learning, the same complexity is achieved due to the necessity of calculating $log|K_\theta(X,X)+\sigma^2 I|$ for the marginal likelihoods needed to learn θ. After this calculation, the predictive mean and variance cost $\mathcal{O}(n)$ and $\mathcal{O}(n^2)$ per test point.

There are three different families of approaches one can take to avoid the explosion of operations with data:

- Approximate nonparametric kernels in a finite basis "dual space". For m basis functions, this only requires $\mathcal{O}(m^2 n)$ computations and $\mathcal{O}(m)$ storage. E.g. SSGP, Random Kitchen Sinks, Fastfood, A la Carte
- Exploit existing structure in K to exactly (and quickly) solve linear systems and log-determinants. E.g. Toeplitz Method, Kronecker Method. This can reduce time complexity to $\mathcal{O}(1)$ mean and variance predictions and $\mathcal{O}(n)$ for inference and learning in the case of Massively Scalable Gaussian Processes (See Wilson, Dann, Nickisch (2015), Thoughts on Massively Scalable Gaussian Processes).
- Inducing point based sparse approximations. E.g. SoR, FITC, KISS-GP

We will focus on the latter of these.

Inducing Point Methods The goal of inducing point methods is to approximate the Gaussian Process through m inducing points, $\bar{\mathbf{f}}$. These points are used to obtain a Sparse Pseudo-input Gaussian process (SPGP) prior, defined as:

$$p(\bar{\mathbf{f}}) = \int d\bar{\mathbf{f}} \prod_n p(f_n|\bar{\mathbf{f}}) p(\bar{\mathbf{f}})$$

Inverting the covariance for a SPGP requires $\mathcal{O}(m^2 n)$ operations.

Typically, selecting inducing points is as simple as making a grid spanned by m selected points. However, higher-dimensional grids would require many inducing points, potentially hurting the efficiency gained by using this method.

Finally, recent efforts have been put forth to parallelize learning and inference for Gaussian Processes such that they can be performed using multiple GPUs. For information, see Wang et al. (2019). Exact Gaussian Processes on a Million Data Points (arXiv:1903.08114).

References

1. Wilson, A. G., Hu, Z., Salakhutdinov, R. & Xing, E. P. Deep Kernel Learning. *CoRR*. arXiv: 1511.02222 [cs.LG]. http://arxiv.org/abs/1511.02222v1 (2015).
2. Al-Shedivat, M., Wilson, A. G., Saatchi, Y., Hu, Z. & Xing, E. P. Learning Scalable Deep Kernels With Recurrent Structure. *CoRR*. arXiv: 1610.08936 [cs.LG]. http://arxiv.org/abs/1610.08936v3 (2016).

10-708: Probabilistic Graphical Models, Spring 2020

Lecture 23: Bayesian Nonparametrics: Dirichlet Proccesses

Lecturer: Eric P. Xing *Scribe: Hai Phan, Magesh Kanna, Stephen Tsou*

1 Introduction

Data can be represented using Parametric and non-parametric models. Parametric models are defined by a countable and fixed number of parameters. These models can be employed in the settings where the exact number of parameters to be used are known. Mixture of K-Gaussians, polynomial regression are few examples of parametric models. For non-parametric models, the number of parameters grows with the sample size. One example for a non-parametric model is Kernel density estimation. Here, the number of parameters can be random.
On the other hand, Bayesian nonparametrics models allow an infinite number of parameters *a priori* leading to infinite capacity. However, a finite dataset uses only a finite set of parameters and hence rest of the unused parameters are integrated out.

1.1 Mixture Models

Mixture Models are a class of parametric models which fixed number of parameters. In the scenario of clustered data, we can fit the data with K- Gaussian Mixture Models.

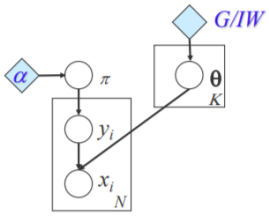


Figure 1: Bayesian Finite Mixture Model

$$p(x_1, x_2, ..., x_N|\pi, \{\mu_k\}, \{\Sigma_k\}) \;=\; \Pi_{n=1}^{\infty}\, \Sigma_{k=1}^{K}\, \pi_k\, \mathcal{N}(x_k|\mu_k, \Sigma_k)$$

We can choose the mixture weights and mixing parameters by using a Bayesian finite Mixture Modelling approach by putting a prior on them and integrating them out.

$$p(x_1, x_2, .., x_N) \;=\; \iiint (\Pi_{n=1}^{\infty}\, \Sigma_{k=1}^{K}\, \pi_k\, \mathcal{N}(x_k|\mu_k, \Sigma_k))\; p(\pi)\; p(\mu_{1:K})\; p(\Sigma_{1:K})\; d\pi\; d\mu_{1:K}\; d\Sigma_{1:K}$$

2 Dirichlet Distribution

The Dirichlet distribution is a distribution over the (K-1)-dimensional simplex. It is parametrized by a K-dimensional vector $(\alpha_1, \alpha_2, ..., \alpha_K)$ such that $\alpha_k \geq 0$ and $\sum_k \alpha_k > 0$. The Dirichlet Distribution is defined as,

$$Dirichlet(\alpha_1, \alpha_2, ..., \alpha_K) = \frac{\Pi_{k=1}^K \Gamma(\alpha_k)}{\Gamma(\Sigma_{k=1}^K \alpha_k)} \Pi_{k=1}^K \pi_k^{\alpha_k - 1}$$

If $\pi \sim$ Dirichlet$(\alpha_1, \alpha_2, ..., \alpha_K)$, then $\pi_k \geq 0$ for all k, and $\sum_k \pi_k = 1$. The expectation is given by :

$$\mathbb{E}[\pi_1, \pi_2, .., \pi_k] = \frac{(\alpha_1, \alpha_2..., \alpha_k)}{\Sigma_k \, \alpha_k}$$

Figure 2: 3-Component Dirichlet Distribution for different configurations.

2.1 Conjugacy to the multinomial distribution

Dirichlet distribution is the conjugate of multinomial distribution.Let $\pi \sim$ Dirichlet$(\alpha_1, \alpha_2, ..., \alpha_K)$ and $x_n \sim$ Multinomial(π), then we have,

$$\begin{aligned} p(\pi|x1, ..., x_n) &\propto p(x_1, x_2, .., x_n|\pi) \; p(\pi) \\ &= (\frac{\Pi_{k=1}^K \Gamma(\alpha_k)}{\Gamma(\Sigma_{k=1}^K \alpha_k)} \Pi_{k=1}^K \pi_k^{\alpha_k - 1})(\frac{n!}{m_1!, .., m_K!} \pi_1^{m_1} ....\pi_K^{m_K}) \\ &\propto \frac{\Pi_{k=1}^K \Gamma(\alpha_k + m_k)}{\Gamma(\Sigma_{k=1}^K \alpha_k + m_k)} \Pi_{k=1}^K \pi_k^{\alpha_k + m_k - 1} \\ &= Dirichlet(\alpha_1 + m_1, ..., \alpha_K + m_K) \end{aligned} \tag{1}$$

Here, m_k corresponds to the number of occurances of $x_n = k$ in the dataset.

Dirichlet Distribution is a distribution over positive vectors that sum to one. We can associate each entry with a set of parameters. For instance, in the case of Gaussian Mixture Models, the parameters would be mean and covariance of each cluster. In a Bayesian setting, these parameters are random. We can combine the distribution over probability vectors with a distribution over parameters to get a distribution over distributions over parameters.

2.2 Properties of Dirichlet Distribution

Collapsing Property: Lets examine the relationship between the Dirichlet distribution and Gamma distributions. Let $\eta_k \sim$Gamma$(\alpha_k, , 1)$ represent K independent Gamma distributed variables. The normalized

sum is represented by a Dirichlet distribution as follows:

$$\frac{(\eta_1, ..., \eta_K)}{\Sigma_k \eta_k} \sim \text{ Dirichlet}(\alpha_1, \alpha_2, ..., \alpha_K)$$

If $\eta_1 \sim$ Gamma$(\alpha_1, 1)$ and $\eta_2 \sim$ Gamma$(\alpha_2, 1)$, then $\eta_1 + \eta_2 \sim$ Gamma($\alpha_1 + \alpha_2$,1).
Therefore, if $(\pi_1, .., \pi_K) \sim$ Dirichlet$(\alpha_1, .., \alpha_K)$ then,

$$(\pi_1 + \pi_2, .., \pi_K) \sim \text{ Dirichlet}(\alpha_1 + \alpha_2, .., \alpha_K)$$

This property is called the collapsing property and is used to reduce the dimensionality of the Dirichlet distribution.

Splitting Property: The beta distribution is a Dirichlet distribution on the 1-simplex. Let $(\pi_1, .., \pi_K) \sim$ Dirichlet$(\alpha_1, .., \alpha_K)$ and $\theta \sim Beta(\alpha_1\, b, \alpha_1\, (1-b))$ for $0 < b < 1$.Then,

$$(\pi_1\theta, \pi_1(1-\theta), \pi_2.., \pi_K) \sim \text{ Dirichlet}(\alpha_1 b_1, \alpha_1(1-b_1), \alpha_2., \alpha_K)$$

More generally, if $\theta \sim$ Dirichlet$(\alpha_1 b_1, \alpha_1 b_2, .., \alpha_1 b_N)$, $\Sigma_i b_i = 1$, then,

$$(\pi_1\theta_1, ..., \pi_1\theta_N, \pi_2, .., \pi_K) \sim \text{Dirichlet}(\alpha_1\theta_1, ..., \alpha_1\theta_N, ..., \alpha_K)$$

This property is called the splitting property.

Re-normalization Property: Dirichlet Distribution follows the renormalization property.
Let $(\pi_1, .., \pi_K) \sim$ Dirichlet$(\alpha_1, .., \alpha_K)$,

$$\frac{(\pi_2, ..\pi_K)}{\Sigma_{k=2}^K \pi_k} \sim \text{Dirichlet}(\alpha_2, ..., \alpha_K)$$

2.3 Infinite Dimensional Prior

In clustering, determining the number of clusters a priori is very hard. Therefore, we allow infinite number of clusters as priors thereby ensuring to have more clusters than required.
An infinite mixture model is defined by:

$$p(x_1|\pi, \{\mu_k\}, \{\Sigma_k\}) \;=\; \Sigma_k\; \pi_k\, \mathcal{N}(x_k|\mu_k, \Sigma_k)$$

We start with Dirichlet distribution with two components $\pi^{(K)} = (\pi_1^{(2)}, \pi_2^{(2)}) \sim Dirichlet(\frac{\alpha}{2}, \frac{\alpha}{2})$ and split each component with the help of the splitting property.

On splitting according to this rule,

$$\theta_1^{(2)}, \theta_2^{(2)} \sim \text{ Beta}(\frac{\alpha}{2}.\frac{1}{2}, \frac{\alpha}{2}.\frac{1}{2})$$

On repeating this process, we get,

$$\pi^{(K)} \sim \text{Dirichlet}(\frac{\alpha}{K}, .., \frac{\alpha}{K})$$

As K in limit tends to infinity, we get a vector with infinitely many components

3 Dirichlet Process

Let define a Dirichlet process. Let H be a base measure distribution on some space Ω. This space can be a Gaussian distribution. We define the following distribution.

$$\pi \sim \lim_{K \to \infty} \text{Dirichlet}\left(\frac{\alpha}{K}, \cdots, \frac{\alpha}{K}\right)$$

For $k = 1 \cdots \infty$, denote

$$\theta_k \sim H$$

Then:

$$G := \sum_{k=1}^{\infty} \pi_k \delta_{\theta_k}$$

is an infinite distribution over base measure H. We can re-write Dirichlet process as follow.

$$G \sim \text{DP}(\alpha, H)$$

Samples from the DP are discrete. We call the point masses in the distribution outcome, atoms. Figure 3 illustrate this. Intuitively speaking, The values of point masses are defined by the length of the bars. The

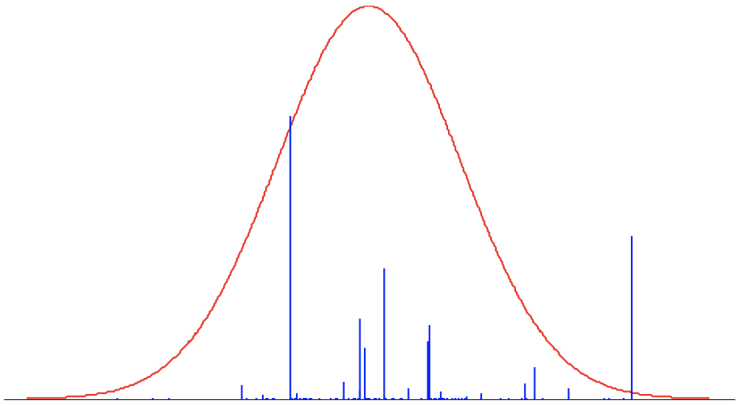


Figure 3: The point masses, atoms, in the resulting distribution. H determines the positions of these atoms concentration parameter α determines the distribution over atom sizes. The smaller value of α is, the sparser distribution is.

Properties of the Dirichlet process. For partitions $A_1, \cdots, A_K$ of Ω, e.g. different color values in an infinite color space. The total mass for each partition is presented in Figure 4.

A Dirichlet process is the unique distribution over probability distributions on spaces Ω for any finite partition $A_1, \cdots, A_K \in \Omega$.

$$(P(A_1), \cdots, P(A_K)) \sim \text{Dirichlet}(\alpha H(A_1), \cdots, \alpha H(A_K))$$

Conjugacy of the Dirichlet process. Denote $P(A_k)$ the mass assigned by $G \sim \text{DP}(\alpha, H)$. Again, we have as follow.

$$(P(A_1), \cdots, P(A_K)) \sim \text{Dirichlet}(\alpha H(A_1), \cdots, \alpha H(A_K))$$

If we see an observation in the J^{th} segment, then:

$$(P(A_1), \cdots, P(A_K) | X_1 \in A_j) \sim \text{Dirichlet}(\alpha H(A_1), \cdots, \alpha H(A_j) + 1, \cdots, \alpha H(A_K))$$

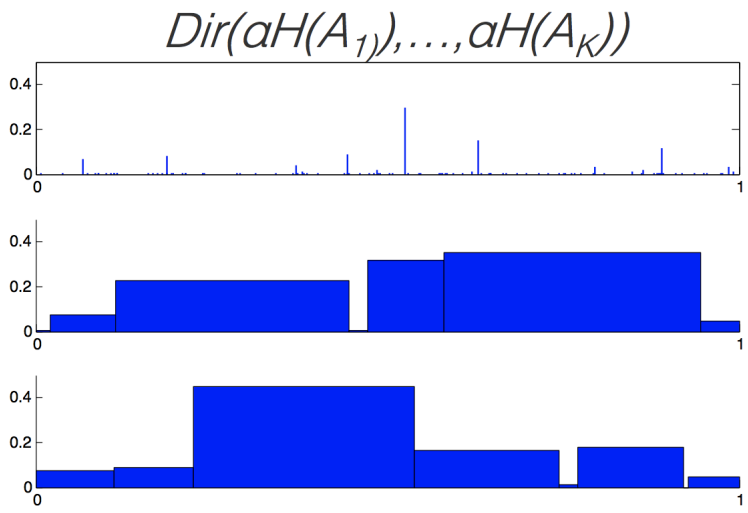


Figure 4: The point masses for each partition.

This must be true for all possible partitions of Ω. This is only possible if the posterior of G, given an observation x, is given by

$$G|X_1 = x \sim \text{DP}\left(\alpha + 1, \frac{\alpha H + \delta_x}{\alpha + 1}\right)$$

Predictive distribution. The Dirichlet process clusters observations because of being a prior in mixture models. A new data point can either join an existing cluster, or start a new cluster.

Assume H is a continuous distribution on Ω. This means for every point θ in Ω, $P_H(\theta) = \theta$. With first data point, Dirichlet process starts a new cluster samples a cluster parameter θ_1. We have now split our parameter space in two: the singleton θ_1, and everything else. Let π_1 be the size of atom at θ_1. The combined mass of all the other atoms is $\pi^* = 1 - \pi_1$.

$$\text{priori, } (\pi_1, \pi_*) \sim \text{Dirichlet}(0, \alpha)$$

$$\text{posteriori, } \quad (\pi_1, \pi_*)|X_1 = \theta_1 \sim \text{Dirichlet}(1, \alpha)$$

If we integrate out π_1 we get.

$$\begin{aligned} P(X_2 = \theta_k | X_1 = \theta_1) &= \int P(X_2 = \theta_k | (\pi_1, \pi_*)) P((\pi_1, \pi_* | X_1 = \theta_1) d\pi_1 \\ &= \int \pi_k \text{Dirichlet}((\pi_1, 1 - \pi)|1, \alpha) d\pi_1 \\ &= \mathbb{E}_{\text{Dirichlet}(1,\alpha)}[\pi_k] \\ &= \begin{cases} \frac{1}{1+\alpha} & \text{if } k = 1 \\ \frac{\alpha}{1+\alpha} & \text{for new k} \end{cases} \end{aligned} \tag{2}$$

The probability $\frac{1}{1+\alpha}$ is for the old cluster (e.g. parameter θ stay at cluster $k = 1$) while $\frac{\alpha}{1+\alpha}$ is probability for a new cluster. Lets say we choose to start a new cluster, and sample a new parameter $\theta_2 \sim H$. Let π_2 be the size of the atom at θ_2.

$$\text{posteriori, } (\pi_1, \pi_2, \pi_*)|X_1 = \theta_1, X_2 = \theta_2 \sim \text{Dirichlet}(1, \alpha)$$

If we integrate out $\pi = (\pi_1, \pi_2, \pi_*)$, we achieve.

$$\begin{aligned} P(X_3 = \theta_k | X_1 = \theta_1, X_2 = \theta_2) &= \int P(X_3 = \theta_k|\pi) P(\pi | X_1 = \theta_1, X_2 = \theta_2) d\pi \\ &= \mathbb{E}_{\text{Dirichlet}(1,1,\alpha)}[\pi_k] \\ &= \begin{cases} \frac{1}{2+\alpha} & \text{if } k = 1, 2 \\ \frac{\alpha}{2+\alpha} & \text{for new k} \end{cases} \end{aligned} \tag{3}$$

In general, if m_k is the number of times we have seen $X_i = k$, and K is the total number of observed values.

$$\begin{aligned} P(X_{n+1} = \theta_k | X_1, \cdots, X_n) &= \int P(X_{n+1} = \theta_k|\pi) P(\pi | X_1, \cdots, X_n) d\pi \\ &= \mathbb{E}_{\text{Dirichlet}(m_1, m_2, \cdots, m_K, \alpha)}[\pi_k] \\ &= \begin{cases} \frac{m_k}{n+\alpha} & \text{if } k \leq K \\ \frac{\alpha}{n+\alpha} & \text{for new cluster} \end{cases} \end{aligned} \tag{4}$$

This is a general form for predictive distribution for the next observation. This distribution have rich-get-richer property, which involve more rich observations.

3.1 Interpretations Of Dirichlet Process

DP – a Pólya urn/ Hoppe Urn Process.

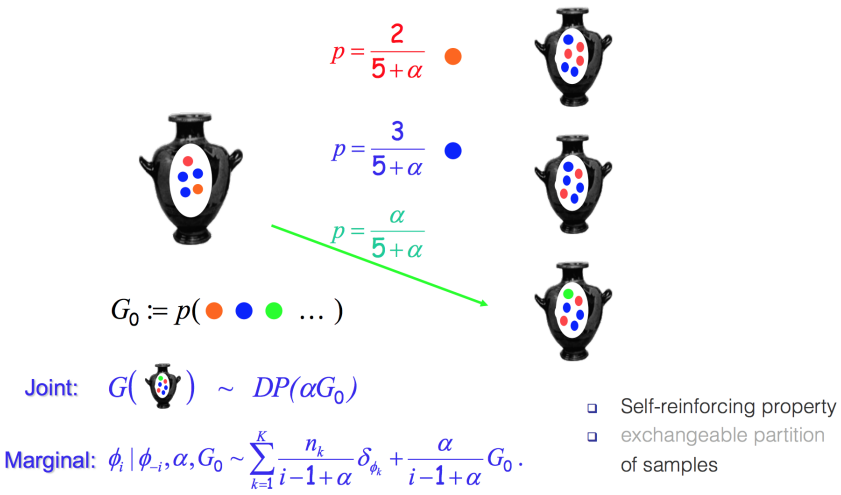


Figure 5: Visualization of pólya urn

The pólya urn scheme is a way of imagining the predictive distribution of new samples under the dirichlet process. It is one way which can be used to represent a dirichlet process. It is seen as an opposite model of sampling without replacement. In this version, they represent the dirichlet as a visual metaphor of a non-transparent urn that contains colored balls that can be drawn randomly. The example in class is actually the hoppe urn or a mutator urn which is more closely related to the Chinese Restaurant Process. In it, a singular ball is initially placed into the urn. It has a fixed $\frac{\alpha}{n+\alpha}$ probability of being drawn where n is the number of other balls in the urn. Every time this initial colored ball is drawn, a new colored ball is put into the urn. If the second colored ball is drawn, a second ball of the same color is put into the urn. If the first

colored ball is drawn again, a ball of a third color is put into the urn. For any color that is not the first, the probability of drawing that color (i.e. blue) is $\frac{k}{n+\alpha}$ where k is the number of ball of that color (i.e. that are already in the urn. Here, the joint distribution of all of the filled in urns after sampling n balls is known as the Dirichlet Process. In this example, the DP Prior is a prior distribution over the colors. The pólya urn is the procedure that defines how to draw colors for every new ball given the previous ones. The classic example of pólya urn has two balls of two different colors initially in the urn.

DP - Stick Breaking

In the non-parametric model case, stick-breaking is a way of generating a dirichlet process by generating a recursive series of beta distributions on a remaining finite mass. It is defined by a Griffiths Engen McCloskey Distribution (GEM). $\forall k$, a_k= 1 and $b_k = \alpha$, with clusters k = $1 : \infty$, atoms $\pi = (\pi_1, \pi_2, ...\pi_k) \sim GEM(\alpha)$:

$$V_k = Beta(a_k, b_k)$$

$$\pi_k = [\Pi_{j=1}^{k-1}(1 - V_j)]V_k$$

When the GEM distribution above is coupled with a parameter distribution $\mu_k \sim \mathcal{N}(\mu_0, \sigma_0)$ for instance, a dirichlet process can be defined as

$$G = \sum_{k=1}^{\infty} \pi_k \delta_{\mu,k} = DP(\alpha, \mathcal{N}(\mu_0, \sigma_0)$$

DP - Chinese Restaurant Process

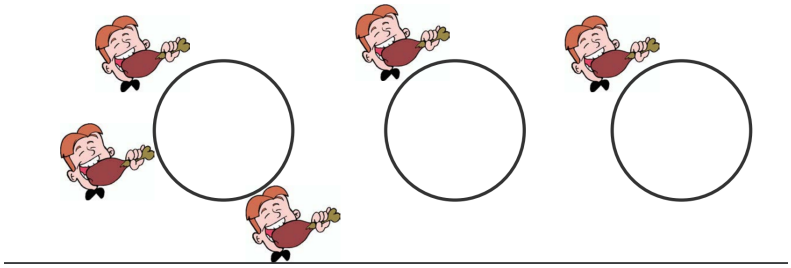


Figure 6: Visualization of Chinese Restaurant Process of 5 Customers

If one integrates all of the π_k out of the above dirichlet process, this is called the Chinese Restaurant Process. Metaphorically, it is akin to seating up to an infinite amount of customers at an up to an infinite amount of tables. It is an index set on a set of permutations.

One interesting aspect of it is both the hoppe urn and Chinese Restaurant Process are invariant to permutation. In other words, the clustering of N customers does not depend on the order in which they arrive and hence exchangeable.

We can define CRP(α, N) as a distribution over all partitions of the labeled set [N] := $\{1, 2, ..., N\}$. For example, $\pi_{[5]} = \{\{1, 3, \}, \{2\}, \{4, 5\}\}$ is a partition of [5]. This distribution is defined recursively. Given a partition $\pi_{[n]}$, the destination of the next person n + 1 has the following distribution.

$$P(n + 1 \text{ joins table c } |\pi_{[n]}) = \frac{|c|}{n + \alpha}$$

$$P(n+1 \text{ starts a new table } |\pi_{[n]} = \frac{\alpha}{n+\alpha}$$

The probability of the example partition $\{\{1,3,\},\{2\},\{6,4,5\}\}$ under CRP(α, 6) is:

$$\frac{\alpha}{\alpha} \cdot \frac{\alpha}{\alpha+1} \cdot \frac{1}{\alpha+2} \cdot \frac{\alpha}{\alpha+3} \cdot \frac{1}{\alpha+4} \cdot \frac{2}{\alpha+5}.$$

Without loss of generality, we can see that the probability of a given partition $\pi_{[N]} \sim$ CRP(α, N) is:

$$P(\pi_{[N]}) = \frac{1}{\alpha(\alpha+1)\cdots(\alpha+N-1)} \Pi_{c \epsilon \pi_{[N]}} \alpha(|c|-1)!)$$

$$= \frac{\alpha^k}{\alpha(\alpha+1)\cdots(\alpha+N-1)} \Pi_{c \epsilon \pi_{[N]}} (|c|-1)!$$

where K represents the number of clusters c in $\pi_{[N]}$. From the above we can see that CRP is exchangeable because only the sizes of the clusters affect the probability.

4 Graphical Model Representations of DP

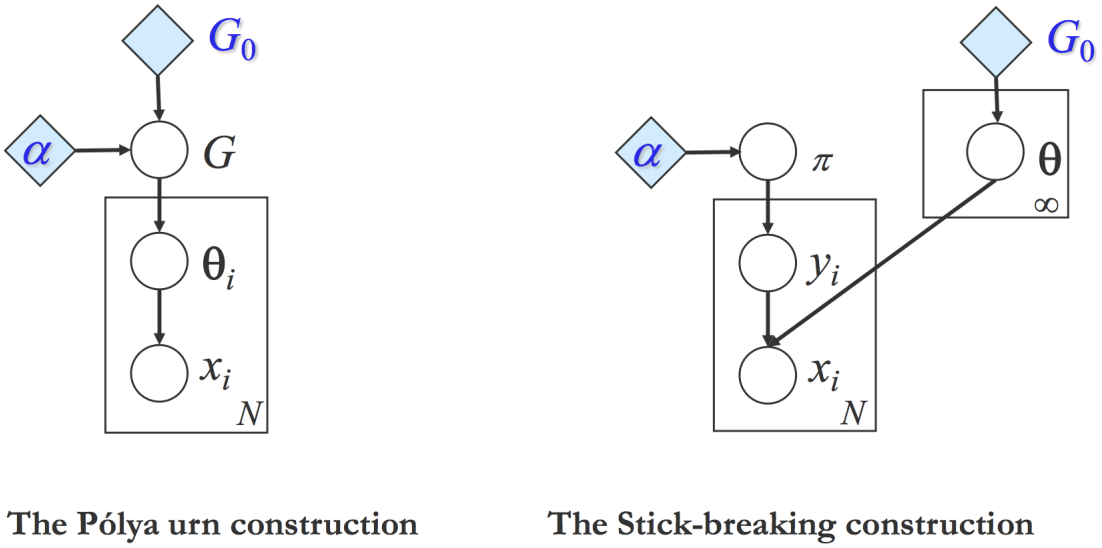


Figure 7: Two typical graphical models of Dirichlet process.

There are two different way to present the same distribution as graphical models. For Pólya urn construction, samples x_i are directly sampled from centroid θ_i, which is sampled from prior of a discrete Dirichlet Process.

In the stick-breaking construction, sample x_i is an indicator function of the cluster y_i , which is computed from multinomial distribution π.

5 Inference

The inference process is to find the potential association between points with infinite number of parameters. Here is the Dirichlet Process mixture model for this.

$$G := \sum_{k=1}^{\infty} \pi_k \delta_{\theta_k} \sim \mathrm{DP}(\alpha, H)$$

$$\phi_n \sim G$$

$$x_n \sim f(\phi_n)$$

Collapsed sampler. The collapsed sampler is integrated to Chinese restaurant process (CRP). When sampling any data point, we can always rearrange the ordering so that it is the last data point. Denote z_n be the cluster allocation of n^{th} data point and K be the total number of instantiated clusters. We have as follow.

$$p(z_n = k | x_n, z_{-n}, \phi_{1:K} \propto \begin{cases} m_k f(x_n|\phi_k) & k \leq K \\ \alpha \int_\Omega f(x_n|\phi) H(d\phi) & k = K+1 \end{cases}$$

There are several problems with collapsed sampler. First of all, we can only sample one data point at a time, leading to the slow performance of mixing. Second, if the likelihood is not conjugate, integrating out parameter values for new features can be difficult.

6 Topic models

Topic models describe documents using a distribution over features. Each feature is a distribution over words. Each document is represented as a collection of words (usually unordered – "bag of words" assumption). The words within a document are distributed according to a document-specific mixture model. The features are shared among documents. The features learned tend to give high probability to semantically related words – "topics".

6.1 Latent Dirichlet allocation

For each topic $k = 1, \cdots, K$, we sample a distribution over words, $\beta \sim \mathrm{Dir}(\eta_1, \cdots, \eta_v)$. For each document $m = 1, \cdots, M$, we sample a distribution over topics, $\theta_m \sim \mathrm{Dir}(a_1, \cdots, a_K)$. For each word, $n = 1, \cdots, N_m$, we sample a topic $z_{nn} \sim \mathrm{Discrete}(\theta_m)$, a word $w_{mk} \sim \mathrm{Discrete}(\beta_z)$.

6.2 Constructing a topic model with infinitely many topics

Latent Dirichlet Allocation (LDA) is one of popular topic models used to classify text in document for a specific topic. In LDA, each distribution is associated with a distribution over K topics. The problem is that

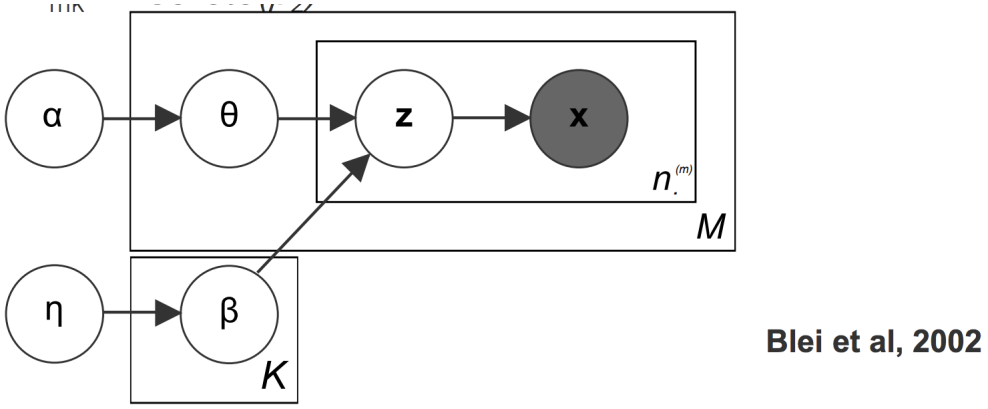


Figure 8: Topic model.

how can we choose the number of topics? So we can apply a Dirichlet process to find number of clusters which is topics. Also, we want to make sure the topics are shared between documents.

Sharing topics. In LDA, we have M independent samples from a Dirichlet distribution. The weights are different, but the topics are fixed to be the same. If we replace the Dirichlet distributions with Dirichlet processes, each atom of each Dirichlet process will pick a topic independently of the other topics. When we sample the same two topics, the probability is zero because of continuous property of the base measure. So, we need to use a discrete base feature, e.g. consider a base measure $H = \sum_{k=1}^{K} \alpha_k \delta_{\beta_k}$, we have LDA to choose number of topics. With infinite number of topics and the locations, we want an infinite, discrete base measure.

7 Hierarchical Dirichlet Process (Teh et al, 2006)

Sample the base measure from a Dirichlet process.

$$G_0 \sim \mathrm{DP}(\gamma, H)$$

$$G_m \sim \mathrm{DP}(\alpha, G_0)$$

7.1 Chinese restaurant franchise

Chinese restaurant franchise is an analog for Hierarchical Dirichlet Process. Imagine a franchise of restaurants, serving an infinitely large, global menu. Each table in each restaurant orders a single dish. We denote the following terms.

- Let n_{rt} be the number of customers in restaurant r sitting at table t.
- Let m_{rd} be the number of tables in restaurant r serving dish d.
- Let m_d be the number of tables, across all restaurants, serving dish d.

The steps in Chinese restaurant franchise are as follow.

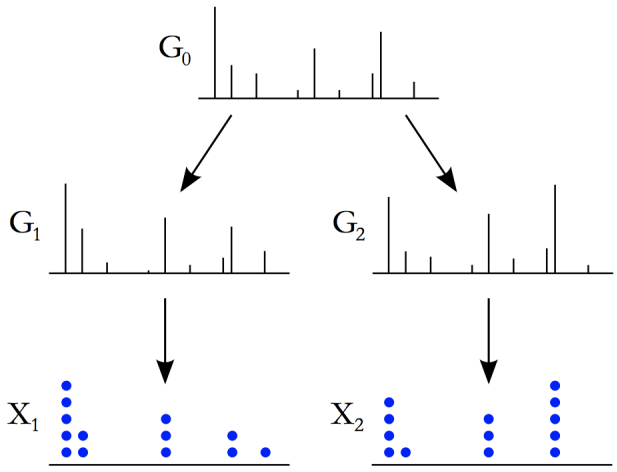


Figure 9: Hierarchical Dirichlet Process.

- The first customer enters a restaurant, and picks a table.
- The n^{th} customer enters the restaurant. He sits at an existing table with probability $\frac{m_k}{(n-1+\alpha)}$, where m_k is the number of people sat at table k. He starts a new table with probability $\frac{\alpha}{n-1+\alpha}$

Each table in each restaurant picks a dish, with probability proportional to the number of times it has been served across all restaurants.

$$p(\text{table } t \text{ chooses dish } d|\text{previous tables}) = \begin{cases} \frac{m_d}{T+\gamma} & \text{for an existing table} \\ \frac{\gamma}{T+\gamma} & \text{for a new table} \end{cases}$$

8 An infinite topic model

In the above mentioned metaphors, to relate to the problem of topic model we can consider restaurants, dishes as documents, topics respectively. Let H be a V-dimensional Dirichlet distribution, so a sample from H is a distribution over a vocabulary of V words.

$$G_0 := \sum_{k=1}^{\infty} \pi_k \delta_{\beta_k} \sim \text{DP}(\alpha, H)$$

For each document $m = 1, \cdots, M$,

- Sample a distribution over topics, $G_m \sim \text{DP}(\gamma, G_0)$.
- For each word $n = 1, \cdots, N_m$, we sample a topic $\phi_{mn} \sim \text{Discrete}(G_0)$ and a word $w_{mk} \sim \text{Discrete}(\phi_{mn}$

9 Experiment: The "right" number of topics

In Figure 10 (left), the perplexity is evaluated by mixture models range between 10 and 120. In Figure 10 (right) the posterior over the number of topics is computed by the hierarchical Dirichlet Process mixture models of the optimal LDA models.

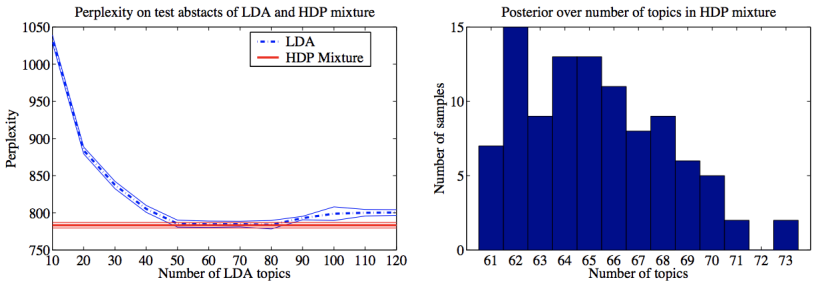


Figure 10: (Left) Comparison of latent Dirichlet allocation and the hierarchical Dirichlet process mixture.Results are averaged over 10 runs; the error bars are one standard error. (Right) Histogram of the number oftopics for the hierarchical Dirichlet process mixture over 100 posterior samples.

10 Summary

Bayesian approach only follow finite-component clustering while nonparametric Bayesian approach involves infinite number of clusters. Dirichlet Process provides a conjugate infinite-dimensional prior. DP can be considered as an infinite limit of a Dirichlet distribution. DP is very powerful in particular metaphor applications: Stick-breaking process, Polya urn process, and Chinese restaurant process.

Hierarchical DP utilize discrete base measure, allowing to reuse atoms. This leads to a powerful application for topic model: topic sharing.

10-708: Probabilistic Graphical Models, Spring 2020

24: Indian Buffet Process

Lecturer: Eric P. Xing *Scribes: Stefan Andjelkovic, David Bick, Prithvi Gudapati*

1 Recap

1.1 Motivating Examples

To understand why we would be interested in these techniques, it is best to start with a motivating example. We see concretely how we could apply the method, and can keep it in mind when observing the technical details.

Imagine we have four ground truth features that we can combine in images, shown below

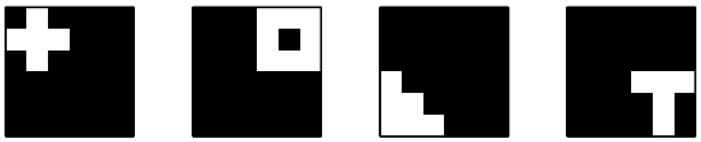


Figure 1: Ground truth features

and we observe combinations of these features that have been corrupted with noise, as shown below

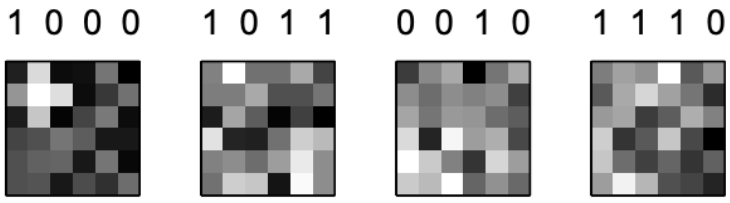


Figure 2: Noisy Combinations of Features

where the 1's and 0's above the image show which of the four latent features are present. In reality, we would observe the corrupted images and not have any idea what the ground truth features are. We would

not even know how many features to expect that there would be. This is the motivation behind the Indian Buffet Process. We would start with the second set of images, and produce the following

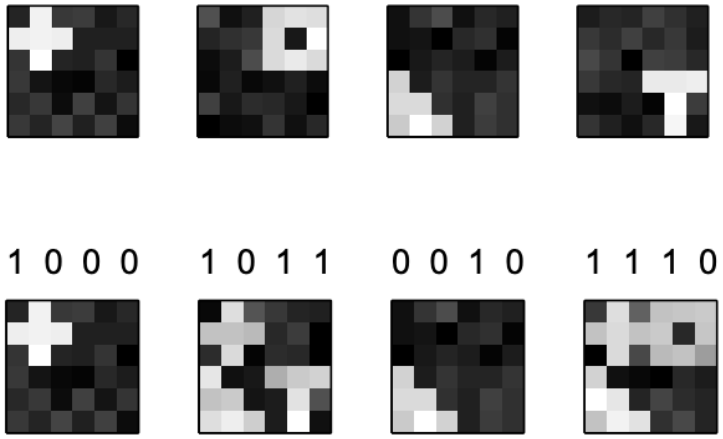


Figure 3: IBP Outputs

where the first set of images is the posterior mean for the latent features that our images are comprised of, and the second set of images is the set of estimates of which features the observed data are composed of. Thus the IBP learned both how many latent features there are in the data, and what these features look like. It also gives an estimate of which features are present in a given observation, the vector above the second set of images.

The images and much of the information in these notes is taken from Griffith and Ghahramani [1].

1.2 General Bayesian Nonparametrics

The general motivation for Bayesian Nonparametrics is to allow an arbitrary number of elements for a desired quantity. In the above example, the finite version is a fixed latent feature model, which was developed before the infinite version. Another classical example is mixture model clustering, where before the Dirichlet process mixture model was developed, one had to specify a number of clusters and iterate over different quantities for this value to see a metric showed diminishing returns.

The bayesian nonparametrics key contribution is allowed the model to add clusters (in the Dirichlet mixture model) or latent variables (IBP) as needed according to the data. To do this, you must specify an unbounded prior distribution on clusters or latent variables. This is generally referred to as an infinite distribution, however once you have finite observed data, which is always the case, it reduces the infinite distribution is limited to a finite instantiation.

2 Latent variable models

We assume that we have a collection of features $\mathbf{F} = [f_1^T f_2^T \dots f_N^T]^T$ [1]. We are concerned then with both $P(X|F)$ and P(F), where $P(X|F)$ is the likelihood for each observation given its set of features, and P(F) is the prior over different matrices of features. The difference between finite and infinite latent variable models comes from P(F), and whether it places a bound on the number of variables.

We decompose $F = Z \otimes V$, where Z is a binary matrix where $z_{ik} = 1$ if object i has feature k, and 0 otherwise [1]. We use $\otimes$ to represent element-wise product, also called Hadamard product. V represents the values that the feature takes if present, and Z determines the presence in F. We can thus decompose p(F) = p(F)p(V).

We break F into the $Z \otimes V$ because it simplifies the problem to finding a infinite prior on binary matrices. Then we can worry about the other values of the features, V, separately. The binary matrices are useful in many applications and so reducing the scope to Z does not reduce the relevance of the study. Also, and most importantly, (finite) binary matrices can be specified by a Beta-Bernoulli process, which has closed form due to the conjugacy of the Beta prior with the Bernoulli likelihood. This allows us to take the limit of the closed form integral, which is not generally possible for any combination of likelihood and prior.

2.1 Finite Variant

We can define the probability of a binary matrix as follows:

$$P(Z|\pi) = \Pi_{k=1}^{K} \Pi_{i=1}^{N} P(z_{ik}|\pi_k) \tag{1}$$

$$= \Pi_{k=1}^{K} \pi_k^{m_k} (1-\pi_k)^{N-m_k} \tag{2}$$

where π_k is the probability of an object containing the i-th feature, and m_k is the number of objects that contain feature k.

The conjugate prior for this likelihood is the beta distribution as a prior on each π_k. We can isolate $P(Z)$, rather than $P(Z|\pi)$, by multiplying by the prior to get the joint $P(Z, \pi)$ and integrating out π. This is shown below, with a $Beta(\frac{\alpha}{K}, 1)$ prior (see [1] for details on the parameters of this prior):

$$P(Z) = \Pi_{k=1}^{K} \int (\Pi_{i=1}^{N} P(z_{ik}|\pi_k)) p(\pi_k) d\pi_k \tag{3}$$

$$= \Pi_{k=1}^{K} \frac{B(m_k + \frac{\alpha}{K}, N - m_k + 1)}{B(\frac{\alpha}{K}, 1)} \tag{4}$$

$$= \Pi_{k=1}^{K} \frac{\frac{\alpha}{K}\Gamma(m_k + \frac{\alpha}{K})\Gamma(N - m_k + 1)}{\Gamma(N + 1 + \frac{\alpha}{K})} \tag{5}$$

2.2 Infinite Limit

To define the infinite limit of eq (5), we must note briefly that we actually consider the equivalence class of binary matrices, not individual binary matrices. The essence of the idea is that if we reorder the columns, we have not actually changed anything about the matrix because the columns are independent. We denote the equivalence class of matrices as $[Z]$ (see [1] for more details).

We break equation (5) into two components, where we denote K_0 as the number of features with $m_k = 0$, and K_+ as the number of features with $m_k > 0$. The result of the decomposition is (7), where (5) is repeated for convenience,

$$\Pi_{k=1}^{K} \frac{\frac{\alpha}{K}\Gamma(m_k + \frac{\alpha}{K})\Gamma(N - m_k + 1)}{\Gamma(N + 1 + \frac{\alpha}{K})} \tag{6}$$

$$= (\frac{N!}{\Pi_{j=1}^{N}(j + \frac{\alpha}{K})})^K (\frac{\alpha}{K})^{K_+} (\Pi_{k=1}^{K} \frac{(N - m_k)!\Pi_{j=1}^{m_k - 1}(j + \frac{\alpha}{K})}{N!}) \tag{7}$$

We then take the limit of equation (7) to get the infinite latent variable model

$$\lim_{K\to\infty} (\frac{N!}{\Pi_{j=1}^{N}(j + \frac{\alpha}{K})})^K (\frac{\alpha}{K})^{K_+} (\Pi_{k=1}^{K} \frac{(N - m_k)!\Pi_{j=1}^{m_k - 1}(j + \frac{\alpha}{K})}{N!}) \tag{8}$$

$$= (\frac{\alpha^{K_+}}{\Pi_{h=1}^{2^N - 1} K_h!}) \exp(-\alpha H_N) \Pi_{k=1}^{K_+} \frac{(N - m_k)!(m_k - 1)!}{N!} \tag{9}$$

The main takeaway is that we have used the closed form of the Beta-Bernoulli model that was facilitated by conjugacy, and taken the limit as number of columns goes to infinity, which was also facilitated by the the equivalence classes to simplify the number of binary matrices. We end with a result that only depends on K_+, so our infinite model depends only on what we observed in our finite dataset, although we allowed an infinite number of features a priori. There is an added term that we get by taking $P([Z])$ rather than $P(Z)$, which counts the number of matrices per equivalence class, see [1] for further details.

3 Predictive distribution: Indian Buffet Process

3.1 The Indian Buffet Process

We can describe a model using an analogy to an Indian restaurant with an infinitely large buffet. In this process, each customer besides the first customer chooses dishes partially based on the customers who came before them. It begins with the first customer helping himself to Poisson(α) dishes. Each subsequent customer helps himself to each of the previously chosen dishes with a probability of $\frac{m_k}{n}$, where m_k is the number of customers before him who took kth dish and n is the number of customers so far including him. Once he has taken dishes that have already been taken by other customers, he proceeds to try Poisson($\frac{\alpha}{n}$) new dishes. Figure 4 is an illustration of what this process would look like with the columns representing the different dishes and the rows representing the customers.

4 Properties of the Indian Buffet Process

The Indian Buffet Process has a rich getting richer property as popular dishes become more popular. This is due to the fact that the probability that a dish gets picked goes up when it is popular. We can see this effect in Figure 5. Once the first few customers pick dishes on the left side, those dishes continue to get picked. However, there continues to be some exploration.

In addition, the number of nonzero entries for each row is distributed according to Poisson(α) due to exchangeability, which means that changing the ordering of the rows and columns in the matrix does not change the joint probability. This allows any row or column to be treated as the last row or column in the matrix.

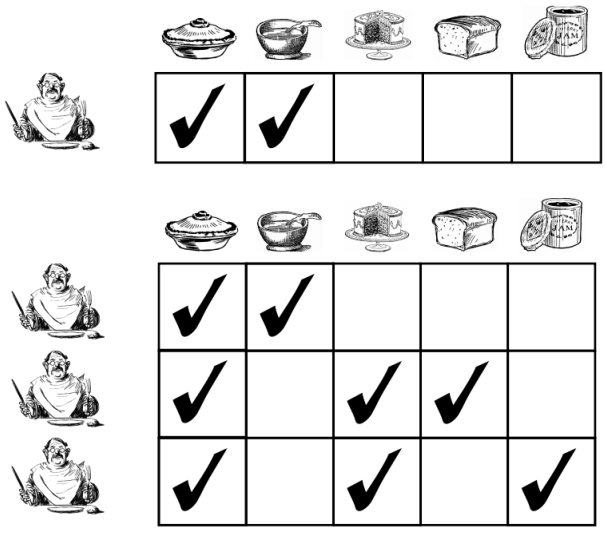


Figure 4: IBP Example

Furthermore, given that if $x_1 \sim \text{Poisson}(\alpha_1)$ and $x_2 \sim \text{Poisson}(\alpha_2)$ then $(x_1 + x_2) \sim \text{Poisson}(\alpha_1 + \alpha_2)$, the number of nonzero entries for the whole matrix is distributed according to Poisson($N\alpha$). Furthermore, the number of non-empty columns is distributed according to Poisson($\alpha H N$)

4.1 Relation to Infinite Beta-Bernoulli Model

Now the goal is to show that the Indian Buffet Process is lof-equivalent to the infinite beta Bernoulli model described earlier.

The probability of a matrix Z generated using this process can be computed to be

$$p(\mathbf{Z}) = \prod_{n=1}^{N} p(\mathbf{z}_n | \mathbf{z}_{1:(n-1)}) \tag{10}$$

$$= \prod_{n=1}^{N} \text{Poisson}\left(K_1^{(n)}\right) \prod_{n-1}^{K_+} \left(\frac{\sum_{i=1}^{n-1} z_{ik}}{n}\right)^{z_{nk}} \left(\frac{n - \sum_{i=1}^{n-1} z_{ik}}{n}\right)^{1-z_{nk}} \tag{11}$$

$$= \frac{\alpha^{K_+}}{\prod_{n=1}^{N} K_1^{(n)}!} \exp\{-\alpha H_N\} \prod_{k=1}^{K_+} \frac{(N - m_k)!(m_k - 1)!}{N!} \tag{12}$$

If we include the cardinality of $\mathbf{Z}$, we can see that this is the same as Equation (9). Thus, we can see that that matrix can be sampled using the Indian Buffet Process.

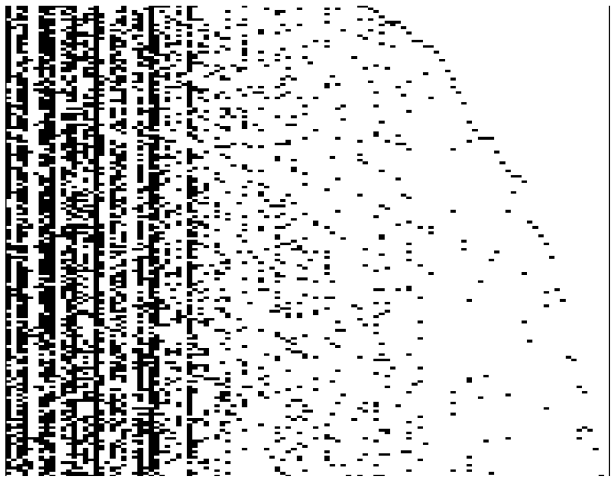


Figure 5: Indian Buffet Process Left Ordering Bias

5 Building latent feature models using the IBP

Now the goal is to use the Indian Buffet Process to build latent feature models.

A simple model that can be developed using the Indian Buffet Process is the linear Gaussian model.

The general form of any latent factor model is

$$\mathbf{X} = \mathbf{W}\mathbf{A}^T + \epsilon, \tag{13}$$

where $\mathbf{W} = \mathbf{Z} \odot \mathbf{V}$. For a linear Gaussian model, $\mathbf{W} = \mathbf{Z}$, so $\mathbf{W}$ is just a binary matrix. $\mathbf{Z}$ is sampled using the Indian Buffet Process, meanwhile $\mathbf{a}_k \sim \mathcal{N}(\mathbf{0}, \sigma_a^2\mathbf{I})$ and $\epsilon_{nk} \sim \mathcal{N}(0, \sigma_\epsilon^2)$

The linear Gaussian model is quite constrained as it is an all-or-nothing model due to the binary "loading matrix" W.

As a result, it is better to not set $\mathbf{W} = \mathbf{Z}$, and instead, make $\mathbf{W}$ a weight matrix. This can be done by making $\mathbf{V}$ a weight matrix that is created using some distribution, for example a Gaussian.

6 Inference in the IBP

Exchangeability, which was mentioned previously as an important property for the Indian Buffet Process, plays an important role in inference.

With K_+ being the total number of used features, excluding the current data point, and Θ being the set of parameters associated with the likelihood, the prior probability of choosing one of these features if $\frac{m_k}{N}$. This is because we can treat the current data point as the last one using exchangeability.

In addition, the posterior probability is proportional to

$$p(z_{nk} = 1|\mathbf{x}_n, \mathbf{Z}_{-nk}, \Theta) \propto m_k f(\mathbf{x}_n|z_{nk} = 1, \mathbf{Z}_{-nk}, \Theta) \tag{14}$$
$$p(z_{nk} = 0|\mathbf{x}_n, \mathbf{Z}_{-nk}, \Theta) \propto (N - m_k) f(\mathbf{x}_n|z_{nk} = 0, \mathbf{Z}_{-nk}, \Theta), \tag{15}$$

where f is the likelihood of the sample given how the features are chosen.

Now, new features must be added as well. This can be done using the Metropolis Hastings method.

1. Propose $K^*_{new} \sim \text{Poisson}(\frac{\alpha}{N})$, and let Z^* be the matrix with K^*_{new} features appearing only in the current data point.
2. Accept the proposed matrix with probability

$$\min\left(1, \frac{f(\mathbf{x}_n|Z^*, \Theta)}{f(\mathbf{x}_n|Z, \Theta)}\right)$$

7 Beta processes and the IBP

In the finite Beta-Bernoulli process, we had $\pi_k \sim \text{Beta}\left(\frac{\alpha}{K}, 1\right)$ and $z_{nk} \sim \text{Bernoulli}\left(\pi_k\right)$. Integrating π_k out leaves exchangeable z_{nk}. We defined Indian Buffet Process as infinite limit of the beta random variables, when $K \to \infty$.

Beta process was defined by Hjort [2], and more on its extension for IBP can be found here [3]. In the scope of this topic, and for the sake of simplicity, we can define beta process as a process characterized by distribution over discrete measures, like the one shown above.

Posterior distribution of the column probabilities can be obtained in the closed form. Beta process atoms (π_k) are drawn from beta distribution, and their counts from Binomial(π_k, N). Because beta distribution is conjugate to binomial, the posterior for each k is Beta $\left(\frac{\alpha}{K} + m_k, N + 1 - m_k\right)$.

To combine all the samples into joint posterior, we can use the beta process stick-breaking construction, as presented in [4].

7.1 Stick-breaking construction

The stick-breaking construction for the beta process is as follows:

- Start with a unit length stick and define $\pi_0 = 1$.
- For each k, sample $\mu_k \sim \text{Beta}(\alpha, 1)$.
- Break off a stick part of length μ_k and throw away the rest.
- Store the value $\pi_k = \pi_{k-1}\mu_k$.

This process is shown in Fig. (6)

Unlike Dirichlet process, in beta process atom values do not necessarily sum up to 1.

Inference in beta process can be performed by sampling distributions $\mathbf{Z} \mid \pi, \mathbf{\Theta}$ and $\pi \mid \mathbf{Z}$. Posteriors for atoms with $m_k > 0$ are beta distributed and posteriors for atoms with $m_k = 0$ can be sampled with stick-breaking procedure.

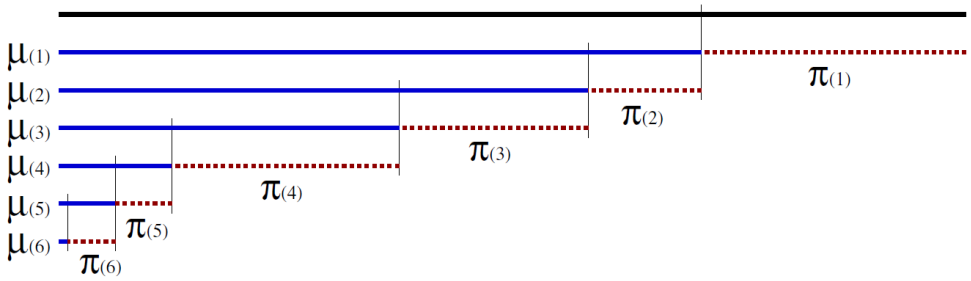


Figure 6: Stick-breaking construction for beta processes, adapted from [4]. Note that red dotted lines correspond to Dirichlet process π_k values, while blue solid lines correspond to beta process μ_k values

7.2 Two-parameter extension

In the previous examples, parameter α determines both *the number of non-empty columns* and *the number of features per data point*. These wouldn't necessarily be equal, so they could be decoupled. The proposed solution is to, instead of sampling $\pi_k \sim \text{Beta}\left(\frac{\alpha}{K}, 1\right)$, sample weights from:

$$\pi_k \sim B\left(\frac{\alpha\beta}{K}, \beta\right)$$

In this extension restaurant scheme is as follows:

- A customer walks into the restaurant and orders Poisson(α) dishes.
- The n-th customer walks into the restaurant and orders previous dishes, each with probability $\frac{m_k}{\beta+n-1}$ of being ordered.
- Then, he orders Poisson($\frac{\alpha\beta}{\beta+n-1}$) new dishes.

This procedure guarantees that marginal number of features follows $N_{\text{features}} \sim \text{Poisson}(\alpha)$, and number of non-empty columns:

$$N_{\text{non-empty columns}} \sim Poisson\left(\alpha \sum_{n=1}^{N} \frac{\beta}{\beta+n-1}\right)$$

When $\beta = 1$, we recover Indian Buffet Process.

8 The infinite gamma-Poisson process

In analogy to the IBP as infinitesimal limit of beta-Bernoulli process, we can define gamma process as infinitesimal limit of gamma random variables, drawn from Poisson distribution. If D is Dirichlet process, drawn from $D \sim \text{DP}(\alpha, H)$, and γ from gamma distribution $D \sim \text{Gamma}(\alpha, 1)$, then gamma process $G = \gamma D$ is distributed according to the $G \sim \text{GaP}(\alpha, H)$.

Gamma distribution is conjugate to the Poisson distribution, so for each atom v_k of gamma process we can sample $z_{nk} \sim \text{Poisson}(v_k)$

Alternatively, to construct matrix $\mathbf{Z}$, row by row, we can modify IBP. For each row n:

- For each of the previous features sample a count $z_{nk} \sim \text{NegBinom}\left(m_k, \frac{n}{n+1}\right)$.
- Sample total count of new features $K_n^* \sim \text{NegBinom}\left(\alpha, \frac{n}{n+1}\right)$.
- Partition K_n^* according to the Chinese Restaurant Process, and assign these counts to the new features.

More on the infinite gamma-Poisson process and how it applies to computer vision can be found in [5].

References

[1] Thomas L. Griffiths and Zoubin Ghahramani. The indian buffet process: An introduction and review. *J. Mach. Learn. Res.*, 12:1185–1224, July 2011.

[2] Nils Lid Hjort et al. Nonparametric bayes estimators based on beta processes in models for life history data. *the Annals of Statistics*, 18(3):1259–1294, 1990.

[3] Romain Thibaux and Michael I Jordan. Hierarchical beta processes and the indian buffet process. In *Artificial Intelligence and Statistics*, pages 564–571, 2007.

[4] Yee Whye Teh, Dilan Grür, and Zoubin Ghahramani. Stick-breaking construction for the indian buffet process. In *Artificial Intelligence and Statistics*, pages 556–563, 2007.

[5] Michalis K Titsias. The infinite gamma-poisson feature model. In *Advances in Neural Information Processing Systems*, pages 1513–1520, 2008.