# Information Theory meets Machine Learning

Kedar Tatwawadi

EE376a Course

# Table of contents

# Introduction

# ML and IT

ML/Statistics & Information theory are two sides of the same coin!

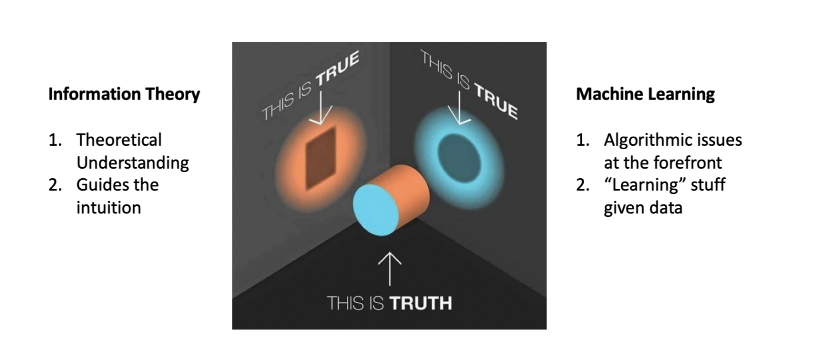



**Figure 1:** ML and IT

# A short (very) intro to ML

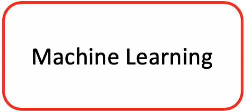



**Figure 2:** ML zoo

# A short (very) intro to ML

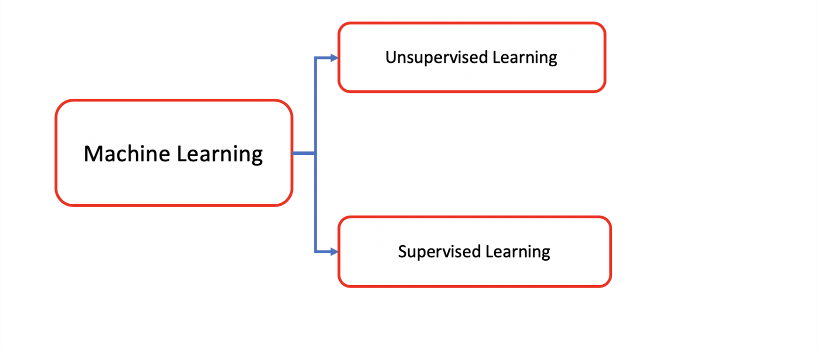



**Figure 3:** ML Zoo

# A short (very) intro to ML

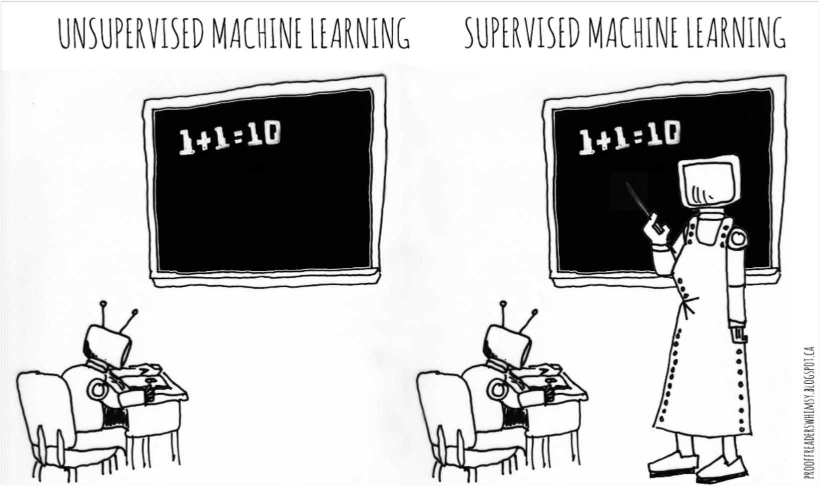



**Figure 4:** ML Zoo

## Supervised Learning

Given data tuples $(X_1, y_1), (X_2, y_2), \ldots, (X_N, y_N)$, find a function $F$ such that:

# Supervised Learning

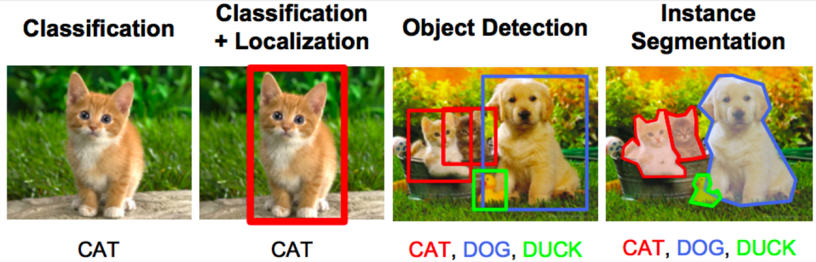
Given data tuples $(X_1, y_1), (X_2, y_2), \ldots, (X_N, y_N)$, find a function $F$ such that:

$$F(X) = y$$

# Supervised Learning

Given data tuples $(X_1, y_1), (X_2, y_2), \ldots, (X_N, y_N)$, find a function $F$ such that:

$$F(X) = y$$



# Supervised Learning

1. What is the function $F$?
2. SVM, ConvNet, Recurrent Neural Network, Decision Tree ...

# Supervised Learning

1. What is the function $F$?
2. SVM, ConvNet, Recurrent Neural Network, Decision Tree ...

Take CS229, CS231n courses!

# A short (very) intro to ML

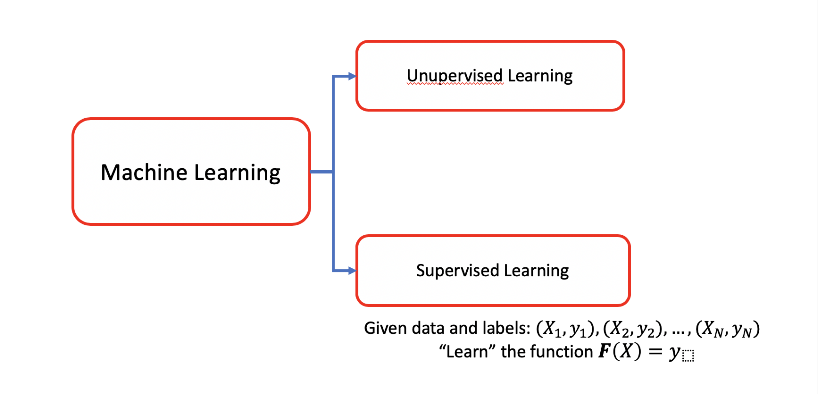



**Figure 5:** ML Zoo

# A short (very) intro to ML

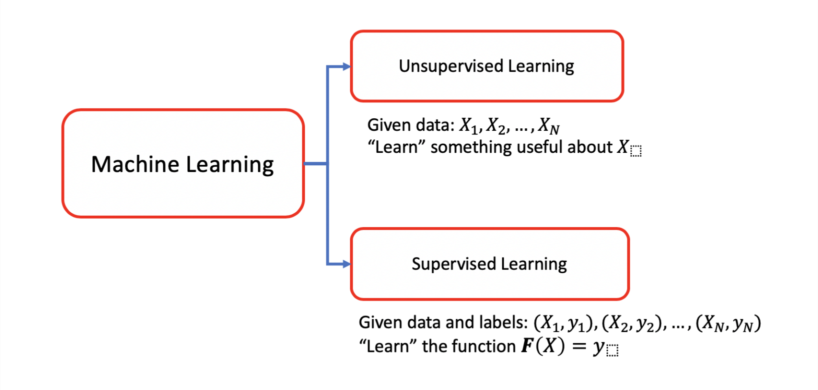



**Figure 6:** ML Zoo

# Unsupervised Learning

# Unsupervised Learning

Given data: $X_1, X_2, X_3, \ldots, X_N$
"Learn" something useful about $X$

1. Clustering
2. Data Representation
3. Distribution of the data

# Clustering

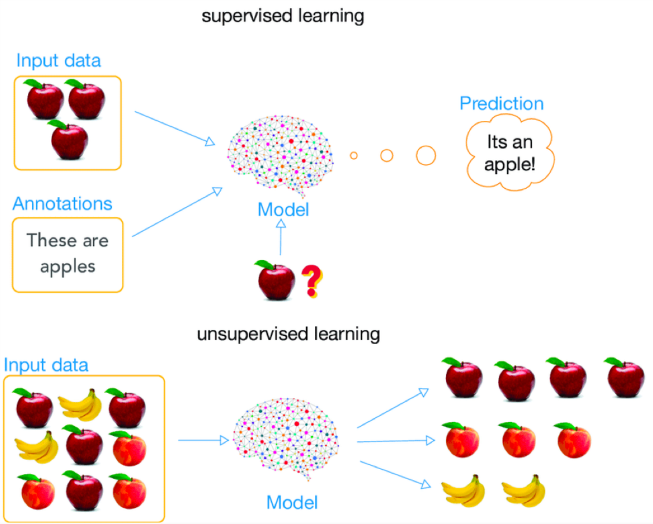



**Figure 7:** Clustering

# Data Representation

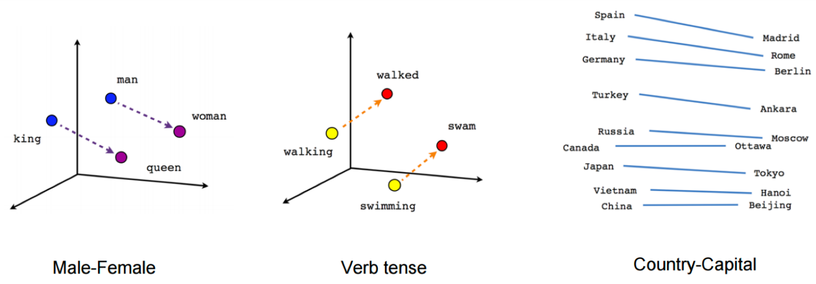
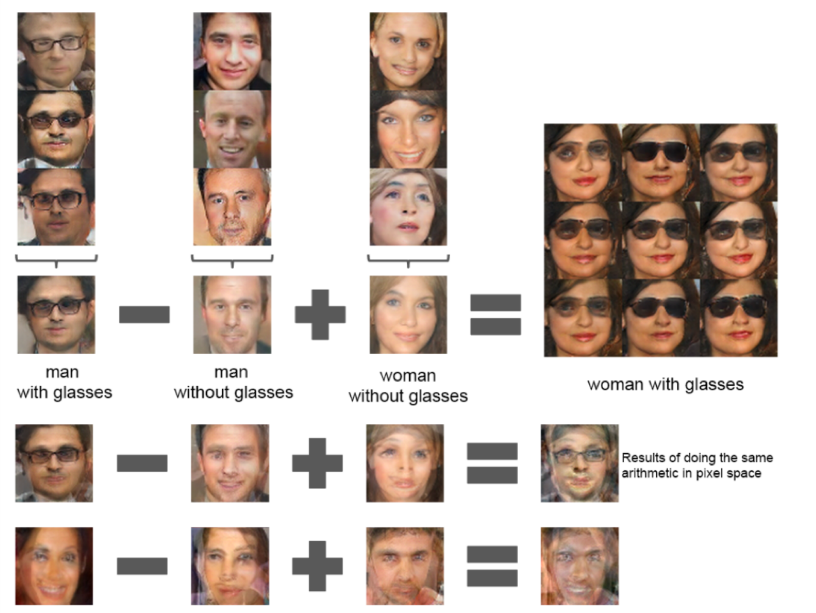



**Figure 8:** Word2Vec Representation

# Data Representation



# Learning Data Distribution

# Learning the distribution

"Learn" the underlying **Distribution of the data**
Given data: $X^{(1)}, X^{(2)}, \ldots, X^{(N)}$ with distribution $p_X(X)$, **How do we learn** $p_X(X)$**?**

# Learning the distribution

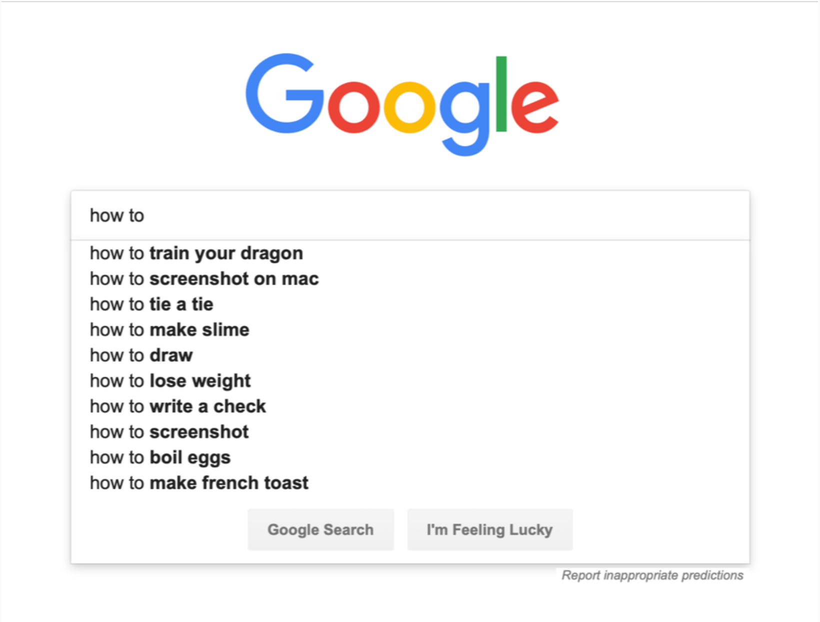
"Learn" the underlying **Distribution of the data**
Given data: $X^{(1)}, X^{(2)}, \ldots, X^{(N)}$ with distribution $p_X(X)$, **How do we learn** $p_X(X)$**?**

**Use cases**

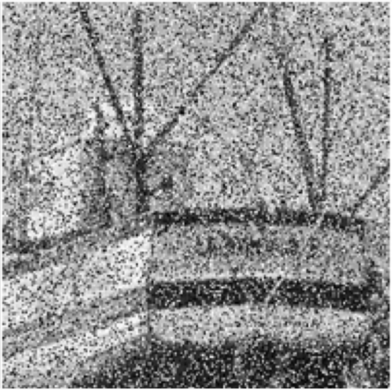
1. Sampling
2. Prediction
3. De-noising
4. Compression

# Sampling

# Prediction


Google
how to
how to train your dragon
how to screenshot on mac
how to tie a tie
how to make slime
how to draw
how to lose weight
how to write a check
how to screenshot
how to boil eggs
how to make french toast
Google Search
I'm Feeling Lucky
Report inappropriate predictions

# Denoising



# Learning the distribution

"Learn" the underlying **Distribution of the data**

1. Sampling
2. Prediction
3. De-noising
4. Compression

# Learning the distribution

Data: $X^{(1)}, X^{(2)}, \ldots, X^{(N)}$ i.i.d (independent and identically distributed) with distribution $p_X(X)$

- $X_i \in \mathcal{X}$
- Potentially $|\mathcal{X}|$ can be high

# Learning the distribution

Data: $X^{(1)}, X^{(2)}, \ldots, X^{(N)}$ i.i.d (independent and identically distributed) with distribution $p_X(X)$

- $X_i \in \mathcal{X}$
- Potentially $|\mathcal{X}|$ can be high

**How do we learn $p_X(X)$?**

# Learning the distribution

Data: $X^{(1)}, X^{(2)}, \ldots, X^{(N)}$ i.i.d (independent and identically distributed) with distribution $p_X(X)$

- $X_i \in \mathcal{X}$
- Potentially $|\mathcal{X}|$ can be high

**How do we learn $p_X(X)$?**

- We can use the Log-loss (Cross-entropy loss) to learn $p_X(X)$

$$p_X = \underset{q(X)}{\text{argmin}}\, \mathrm{E}_{p_X} \log \frac{1}{q(X)} \tag{1}$$

# Learning the distribution

Data: $X^{(1)}, X^{(2)}, \ldots, X^{(N)}$ with distribution $p_X(X)$

$$
\begin{aligned}
\mathrm{E}_{p_X} \log \frac{1}{q(X)} &= \sum_{x \in \mathcal{X}} p(x) \log \frac{1}{q(x)} \\
&= \sum_{x \in \mathcal{X}} p(x) \log \frac{1}{p(x)} \frac{p(x)}{q(x)} \\
&= \sum_{x \in \mathcal{X}} p(x) \log \frac{1}{p(x)} + \sum_{x \in \mathcal{X}} p(x) \log \frac{p(x)}{q(x)} \\
&= H_p(X) + D_{KL}(p_X || q)
\end{aligned}
$$

# Learning the distribution

Data: $X^{(1)}, X^{(2)}, \ldots, X^{(N)}$ with distribution $p_X(X)$

$$
\begin{aligned}
\mathrm{E}_{p_X} \log \frac{1}{q(X)} &= \sum_{x \in \mathcal{X}} p(x) \log \frac{1}{q(x)} \\
&= \sum_{x \in \mathcal{X}} p(x) \log \frac{1}{p(x)} \frac{p(x)}{q(x)} \\
&= \sum_{x \in \mathcal{X}} p(x) \log \frac{1}{p(x)} + \sum_{x \in \mathcal{X}} p(x) \log \frac{p(x)}{q(x)} \\
&= H_p(X) + D_{KL}(p_X || q)
\end{aligned}
$$

$$
p_X = \underset{q(X)}{\operatorname{argmin}} \, \mathrm{E}_{p_X} \log \frac{1}{q(X)}
$$

# Learning the distribution

$$p_X = \operatorname*{argmin}_{q(X)} \mathrm{E}_{p_X} \log \frac{1}{q(X)}$$

- In practice we consider empirical expectation instead:

$$\operatorname*{argmin}_{q(X)} \mathrm{E}_{p_X} \log \frac{1}{q(X)} \approx \operatorname*{argmin}_{q(X)} \frac{1}{N} \sum_{i=1}^{N} \log \frac{1}{q(X^{(i)})}$$

# Learning the distribution

- In practice we consider empirical expectation instead:

$$\begin{aligned}
\operatorname*{argmin}_{q(X)} \frac{1}{N} \sum_{i=1}^{N} \log \frac{1}{q(X^{(i)})} &= \operatorname*{argmin}_{q(X)} \frac{1}{N} \log \frac{1}{q(X_1)q(X_2)\dots q(X_N)} \\
&= \operatorname*{argmin}_{q(X)} \sum_{x \in \mathcal{X}} \frac{n_x}{N} \log \frac{1}{q(x)} \\
&= \operatorname*{argmin}_{q(X)} \mathrm{E}_{\hat{p}_X} \log \frac{1}{q(x)}
\end{aligned}$$

# Learning the distribution

$$\operatorname*{argmin}_{q(X)} \frac{1}{N} \sum_{i=1}^{N} \log \frac{1}{q(X^{(i)})} = \operatorname*{argmin}_{q(X)} \mathrm{E}_{\hat{p}_X} \log \frac{1}{q(x)}$$

$$= \hat{p}_X(x) = \frac{n_x}{N}$$

# Learning the distribution

$$\begin{aligned}\underset{q(X)}{\operatorname{argmin}} \frac{1}{N} \sum_{i=1}^{N} \log \frac{1}{q(X^{(i)})} &= \underset{q(X)}{\operatorname{argmin}} \mathrm{E}_{\hat{p}_X} \log \frac{1}{q(x)} \\ &= \hat{p}_X(x) = \frac{n_x}{N}\end{aligned}$$

- When $X = (Y_1, Y_2, \ldots, Y_d)$, $|\mathcal{X}| = k^d$
- For high $|\mathcal{X}|$, $\hat{p}_X$ is not useful!

# Learning the distribution

$$
\begin{aligned}
\operatorname*{argmin}_{q(X)} \frac{1}{N} \sum_{i=1}^{N} \log \frac{1}{q(X^{(i)})} &= \operatorname*{argmin}_{q(X)} \mathrm{E}_{\hat{p}_X} \log \frac{1}{q(x)} \\
&= \hat{p}_X(x) = \frac{n_x}{N}
\end{aligned}
$$

- When $X = (Y_1, Y_2, \ldots, Y_d)$, $|\mathcal{X}| = k^d$
- For high $|\mathcal{X}|$, $\hat{p}_X$ is not useful!
- We need more data, or ... some regularization.

# Data Example

$F$ features

$n$ records

| UserID | Age | Locatio | Device | Time | DocID | … |
|---|---|---|---|---|---|---|
| 4324234 | 25 | 90210 | iPhone 7 | 9pm | 33221 | … |
| 1223231 | 49 | 94087 | iPad pro | 10am | 66543 | … |
| … | … | … | … | … | … | … |

- $X = (Y_1, Y_2, \ldots, Y_d)$, $|\mathcal{X}| = k^d$, $N \approx$ number of dimensions.

# Regularization

$$\begin{aligned}\operatorname*{argmin}_{q(X)} \mathrm{E}_{p_X} \log \frac{1}{q(x)} &= \operatorname*{argmin}_{q(X)} \mathrm{E}_{\hat{p}_X} \log \frac{1}{q(x)} \\ &\approx \operatorname*{argmin}_{q(X)\in Q} \mathrm{E}_{\hat{p}_X} \log \frac{1}{q(x)}\end{aligned}$$

- $q(X) = q(Y_1, Y_2, \ldots, Y_d) =$
  $q_1(Y_1)q_2(Y_2|Y_1)q_3(Y_3|Y_2, Y_1)\ldots q_d(Y_d|Y_1, \ldots, Y_{d-1})$
- $Q$ restricts some distributions
  e.g.: $q(Y_1, Y_2, \ldots, Y_d) = q_1(Y_1)q_2(Y_2)q_3(Y_3)\ldots q_d(Y_d)$

# $Q_I$ independent distributions

- $Q_I$ restricts the distribution over the $d$ dimensions to be independent
  e.g.: $q(Y_1, Y_2, \ldots, Y_d) = q_1(Y_1)q_2(Y_2)q_3(Y_3)\ldots q_d(Y_d)$

$$\operatorname*{argmin}_{q(X)\in Q_I} \mathrm{E}_{\hat{p}_X} \log \frac{1}{q(x)} = (\hat{q}_1(y_1), \ldots, \hat{q}_d(y_d))$$

- $q(Y_1, Y_2, \ldots, Y_d) = \hat{q}_1(y_1)\hat{q}_2(y_2)\ldots\hat{q}_d(y_d)$
  is not very useful for the tabular dataset

# Tabular Example

$F$ features

$n$ records

| UserID | Age | Locatio | Device | Time | DocID | … |
|---|---|---|---|---|---|---|
| 4324234 | 25 | 90210 | iPhone 7 | 9pm | 33221 | … |
| 1223231 | 49 | 94087 | iPad pro | 10am | 66543 | … |
| … | … | … | … | … | … | … |

# Tree-based distributions

- We restrict distributions to $\mathcal{T}$:
  e.g.: $\mathcal{T} = \{q | q(Y_1, Y_2, \dots, Y_d) = q_1(Y_1) q_2(Y_2 | Y_{i_2}) q_3(Y_3 | Y_{i_3}) \dots q_d(Y_d | Y_{i_d})\}$

# Tree-based distributions

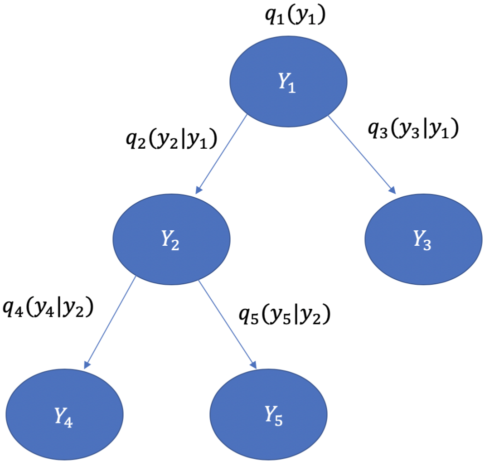
- We restrict distributions to $\mathcal{T}$:
  e.g.: $\mathcal{T} = \{q | q(Y_1, Y_2, \dots, Y_d) = q_1(Y_1) q_2(Y_2 | Y_{i_2}) q_3(Y_3 | Y_{i_3}) \dots q_d(Y_d | Y_{i_d})\}$
- For every $Y_i$, we allow dependence on one of the other variables $Y_{i_j}, i_j < i$

# Tree-based distributions

- We restrict distributions to $\mathcal{T}$:
  e.g.: $\mathcal{T} = \{q | q(Y_1, Y_2, \ldots, Y_d) = q_1(Y_1) q_2(Y_2 | Y_{i_2}) q_3(Y_3 | Y_{i_3}) \ldots q_d(Y_d | Y_{i_d})\}$
- For every $Y_i$, we allow dependence on one of the other variables $Y_{i_j}, i_j < i$
- This exactly corresponds to a "tree distribution"

# Tree-based distributions — Examples

- Example tree distribution:
  $q(Y_1, Y_2, \ldots, Y_5) = q_1(Y_1)q_2(Y_2|Y_1)q_3(Y_3|Y_1)q_4(Y_4|Y_2)q_5(Y_5|Y_2)$



- Example tree distribution:
  $q(Y_1, Y_2, Y_3) = q_1(Y_1)q_2(Y_2|Y_1)q_3(Y_3|Y_2)$

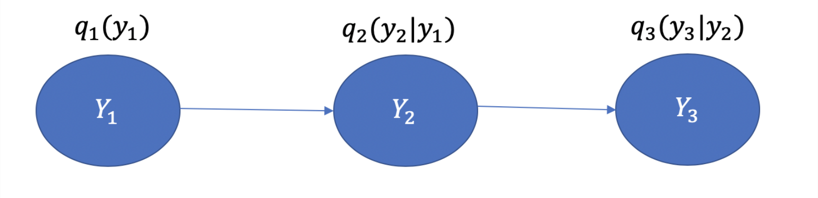



**Figure 11:** Graph example

# Tree-based distributions

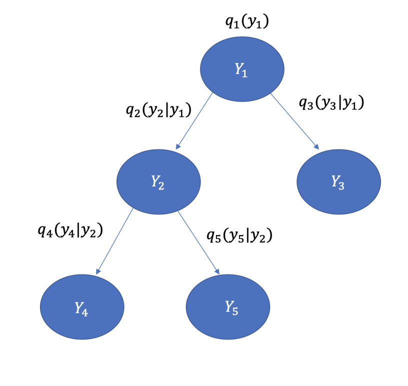



**Figure 12:** Graph example

- Tree distributions are practical! No of parameters $= dk^2$

# Tree-based distributions

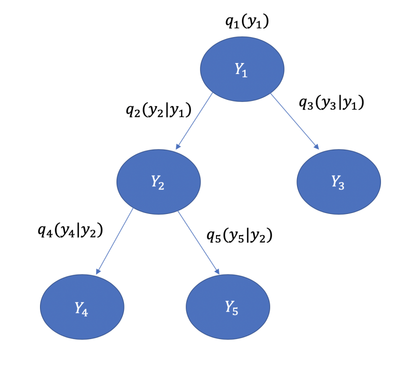



**Figure 12:** Graph example

- Tree distributions are practical! No of parameters $= dk^2$
- Sampling is easy (in a breadth-first search order): $Y_1 \to Y_2 \to Y_3 \to Y_4 \to Y_5$

# Tree-based distributions

- Can be used for compression, using Arithmetic coding:
- $q(Y_1, Y_2, \dots, Y_5) = q_1(Y_1)q_2(Y_2|Y_1)q_3(Y_3|Y_1)q_4(Y_4|Y_2)q_5(Y_5|Y_2)$

(f) Suppose both the encoder and the decoder have a prediction algorithm (say a neural network) that provides probabilities $q_i(x|x^{i-1})$ for all $i$'s and all $x \in \mathcal{X}$. How would you modify the scheme such that you achieve

$$l(x^n) \leq \log \frac{1}{q_1(x_1)q_2(x_2|x_1)\dots q_n(x_n|x^{n-1})} + 1$$

Thus, if you have a prediction model for your data, you can apply arithmetic coding on it - high probability translating to short compressed representations.

**Figure 13:** HW3 Q3(f)

# Chow-Liu Tree Algorithm

- Let $\hat{I}(Y_i; Y_j)$ be the mutual information computed using the "empirical" distribution: $\hat{p}_X(X) = \hat{p}_X(Y_1, Y_2, \ldots, Y_d)$
  The best tree graph representing the data can be found by:

$$G = \operatorname{argmax} \sum_{edges(i,j)} \hat{I}(Y_i; Y_j) \tag{2}$$

# Chow-Liu Tree Algorithm

- Let $\hat{I}(Y_i; Y_j)$ be the mutual information computed using the "empirical" distribution: $\hat{p}_X(X) = \hat{p}_X(Y_1, Y_2, \dots, Y_d)$
  The best tree graph representing the data can be found by:

$$G = \operatorname{argmax} \sum_{edges(i,j)} \hat{I}(Y_i; Y_j) \tag{2}$$

- Intuition:: Add edges which have "high' correlation.

# Data Example

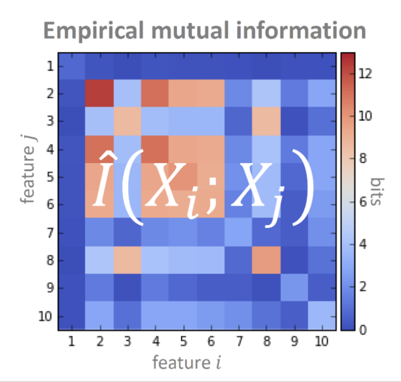
$F$ features

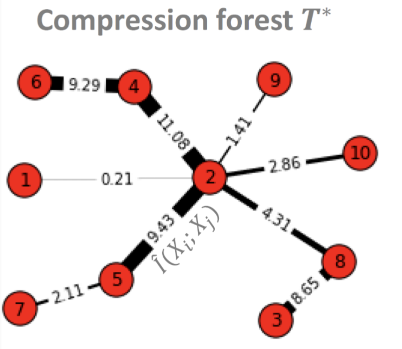
$n$ records

| UserID | Age | Locatio | Device | Time | DocID | ... |
|---|---|---|---|---|---|---|
| 4324234 | 25 | 90210 | iPhone 7 | 9pm | 33221 | ... |
| 1223231 | 49 | 94087 | iPad pro | 10am | 66543 | ... |
| ... | ... | ... | ... | ... | ... | ... |

- $X = (Y_1, Y_2, \ldots, Y_d)$, $|\mathcal{X}| = k^d$, $N \approx$ number of dimensions.

# Data Example



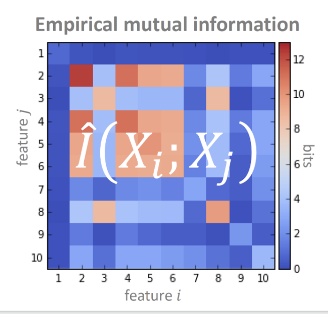
# Chow-Liu Tree Algorithm

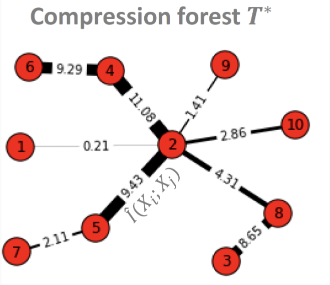
- Let $\hat{I}(Y_i; Y_j)$ be the mutual information computed using the "empirical" distribution: $\hat{p}_X(X) = \hat{p}_X(Y_1, Y_2, \ldots, Y_d)$
  The best tree graph representing the data can be found by:

$$G = \text{argmax} \sum_{edges(i,j)} \hat{I}(Y_i; Y_j) \tag{3}$$

- Intuition:: Add edges which have "high' correlation.
- We will proove this in the class!

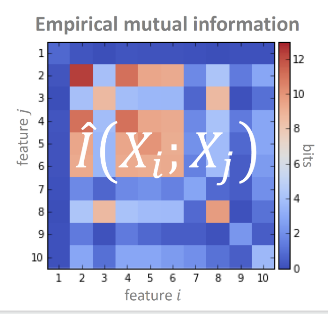
# Practical Considerations

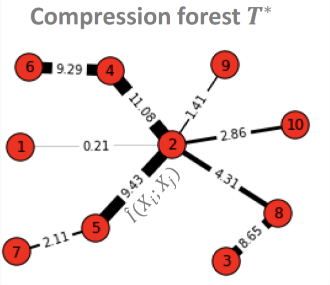



- Exaustive search over all trees is not possible, use $O(d \log d)$ algorithm such as Kruskal's or Prim's algorithm

# Practical Considerations




- Exaustive search over all trees is not possible, use $O(d \log d)$ algorithm such as Kruskal's or Prim's algorithm
- Need to compute $O(d^2)$ mutual informations, which is the more costly part

## Practical Considerations

- $$G = \operatorname{argmax} \sum_{edges(i,j)} \hat{I}(Y_i; Y_j) \tag{4}$$

  $G$ is a solution to the problem:

  $$\underset{q(X)}{\operatorname{argmin}} \mathrm{E}_{\hat{p}_X} \log \frac{1}{q(x)} \approx \underset{q(X)}{\operatorname{argmin}} \mathrm{E}_{p_X} \log \frac{1}{q(x)}$$

  and thus "approximates" $\hat{p}_X(X)$.

## Practical Considerations

- $$G = \operatorname{argmax} \sum_{edges(i,j)} \hat{I}(Y_i; Y_j) \tag{4}$$

  $G$ is a solution to the problem:

  $$\underset{q(X)}{\operatorname{argmin}} \mathrm{E}_{\hat{p}_X} \log \frac{1}{q(x)} \approx \underset{q(X)}{\operatorname{argmin}} \mathrm{E}_{p_X} \log \frac{1}{q(x)}$$

  and thus "approximates" $\hat{p}_X(X)$.
- We really want to solve the problem:

  $$\operatorname{argmax} \sum_{edges(i,j)} I(Y_i; Y_j)$$

# Practical Considerations

- $$G = \operatorname{argmax} \sum_{edges(i,j)} \hat{I}(Y_i; Y_j) \tag{4}$$

  $G$ is a solution to the problem:

  $$\underset{q(X)}{\operatorname{argmin}} \mathrm{E}_{\hat{p}_X} \log \frac{1}{q(x)} \approx \underset{q(X)}{\operatorname{argmin}} \mathrm{E}_{p_X} \log \frac{1}{q(x)}$$

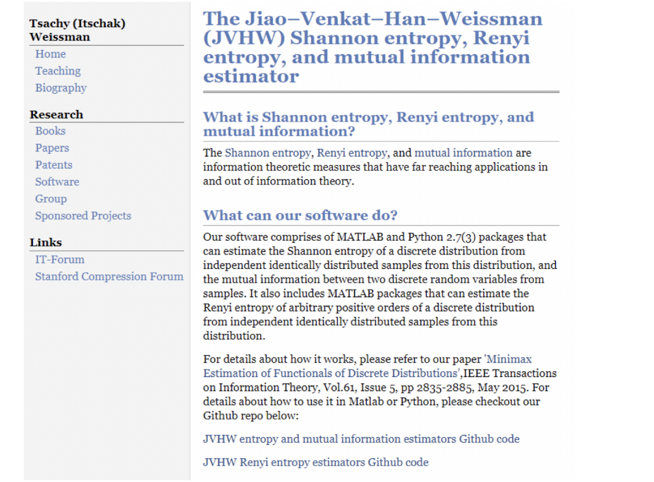
  and thus "approximates" $\hat{p}_X(X)$.
- We really want to solve the problem:

  $$\operatorname{argmax} \sum_{edges(i,j)} I(Y_i; Y_j)$$

- Using $N$ samples we can have better estimators for $I(Y_i, Y_j)$ than the empirical plug-in estimator $\hat{I}(Y_i, Y_j)$
- Information theory helps us get better estimators!

# Practical Considerations

- Using $N$ samples we can have better estimators for $I(Y_i, Y_j)$ than the empirical plug-in estimator $\hat{I}(Y_i, Y_j)$
- Information theory helps us get better estimators!



**Tsachy (Itschak) Weissman**

- Home
- Teaching
- Biography

**Research**

- Books
- Papers
- Patents
- Software
- Group
- Sponsored Projects

**Links**

- IT-Forum
- Stanford Compression Forum

## The Jiao–Venkat–Han–Weissman (JVHW) Shannon entropy, Renyi entropy, and mutual information estimator

### What is Shannon entropy, Renyi entropy, and mutual information?

The Shannon entropy, Renyi entropy, and mutual information are information theoretic measures that have far reaching applications in and out of information theory.

### What can our software do?

Our software comprises of MATLAB and Python 2.7(3) packages that can estimate the Shannon entropy of a discrete distribution from independent identically distributed samples from this distribution, and the mutual information between two discrete random variables from samples. It also includes MATLAB packages that can estimate the Renyi entropy of arbitrary positive orders of a discrete distribution from independent identically distributed samples from this distribution.

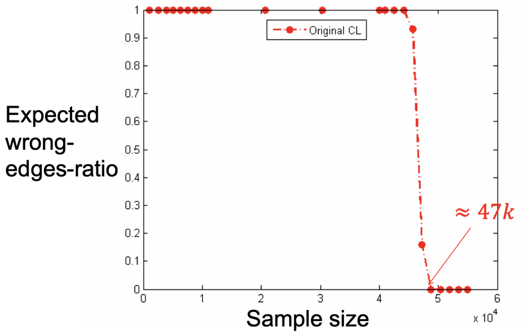
For details about how it works, please refer to our paper 'Minimax Estimation of Functionals of Discrete Distributions',IEEE Transactions on Information Theory, Vol.61, Issue 5, pp 2835-2885, May 2015. For details about how to use it in Matlab or Python, please checkout our Github repo below:

JVHW entropy and mutual information estimators Github code

JVHW Renyi entropy estimators Github code

# Practical Considerations

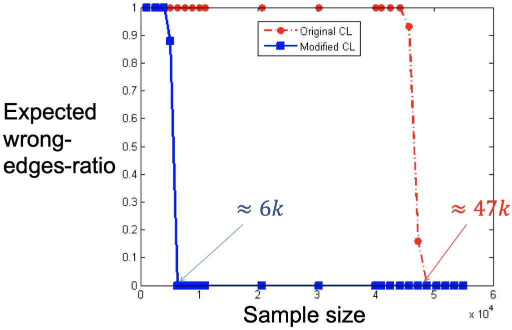
- Using $N$ samples we can have better estimators for $I(Y_i, Y_j)$ than the empirical plug-in estimator $\hat{I}(Y_i, Y_j)$
- Information theory helps us get better estimators!




Setting: star graph, 7 nodes, each node alphabet size 300

# Practical Considerations

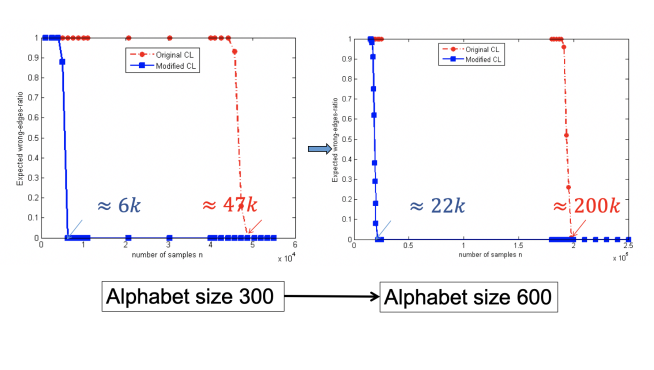
- Using $N$ samples we can have better estimators for $I(Y_i, Y_j)$ than the empirical plug-in estimator $\hat{I}(Y_i, Y_j)$
- Information theory helps us get better estimators!




Setting: star graph, 7 nodes, each node alphabet size 300

# Practical Considerations

- Using $N$ samples we can have better estimators for $I(Y_i, Y_j)$ than the empirical plug-in estimator $\hat{I}(Y_i, Y_j)$
- Information theory helps us get better estimators!



## Practical Considerations

- When we solve the optimization problem, we are not penalizing model complexity:

$$G = \text{argmax} \sum_{edges(i,j)} \hat{I}(Y_i; Y_j) \tag{5}$$

$G$ is a solution to the problem:

- Practically this is important. For example in compression, we also need space to store the distributions $\hat{p}(Y_i|J_j)$ themselves! (along with arithmetic coding).

# Practical Considerations

- When we solve the optimization problem, we are not penalizing model complexity:

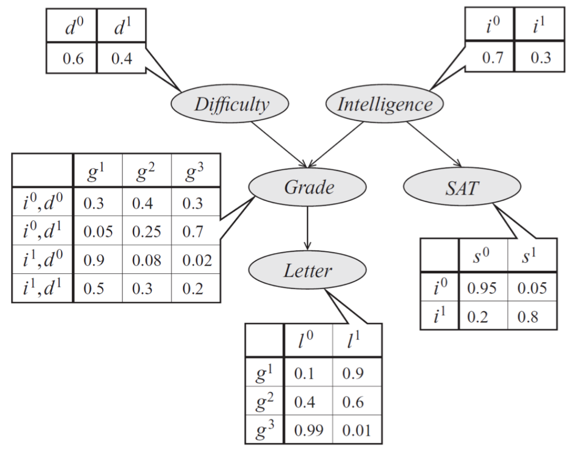
$$G = \text{argmax} \sum_{edges(i,j)} \hat{I}(Y_i; Y_j) \tag{5}$$

  $G$ is a solution to the problem:

- Practically this is important. For example in compression, we also need space to store the distributions $\hat{p}(Y_i|J_j)$ themselves! (along with arithmetic coding).
- The **BIC Criteria** (Bayesian Information Criteria), alters the optimization by adding a penalty function for model complexity

$$\text{argmax} \sum_{edges(i,j)} \left( \hat{I}(Y_i; Y_j) + \frac{1}{2} \log N |\mathcal{Y}_i||\mathcal{Y}_j| \right)$$

# General Bayesian Networks




| $d^0$ | $d^1$ |
|---|---|
| 0.6 | 0.4 |

| $i^0$ | $i^1$ |
|---|---|
| 0.7 | 0.3 |

| | $g^1$ | $g^2$ | $g^3$ |
|---|---|---|---|
| $i^0, d^0$ | 0.3 | 0.4 | 0.3 |
| $i^0, d^1$ | 0.05 | 0.25 | 0.7 |
| $i^1, d^0$ | 0.9 | 0.08 | 0.02 |
| $i^1, d^1$ | 0.5 | 0.3 | 0.2 |

| | $s^0$ | $s^1$ |
|---|---|---|
| $i^0$ | 0.95 | 0.05 |
| $i^1$ | 0.2 | 0.8 |

| | $l^0$ | $l^1$ |
|---|---|---|
| $g^1$ | 0.1 | 0.9 |
| $g^2$ | 0.4 | 0.6 |
| $g^3$ | 0.99 | 0.01 |

# Chow-Liu Algorithm for Bayesian Networks

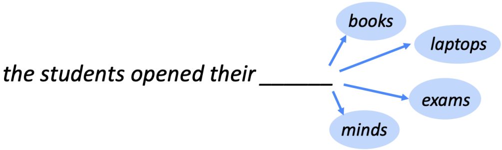
- Let $\hat{I}(Y_i; Y_j)$ be the mutual information computed using the "empirical" distributions.

For general bayesian networks:

$$G = \operatorname{argmax} \sum_{edges(i,j)} \hat{I}(Y_i; Y_j) \tag{6}$$

- Chow-liu algorithm for Bayesian networks is an approximation based on the intuition for tree-based algorithms
- Exact solutions no more possible. Apply heuristic greedy schemes

- **Language Modeling** is the task of predicting what word comes next.




- More formally: given a sequence of words $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(t)}$, compute the probability distribution of the next word $\boldsymbol{x}^{(t+1)}$ :

$$P(\boldsymbol{x}^{(t+1)} \mid \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})$$

where $\boldsymbol{x}^{(t+1)}$ can be any word in the vocabulary $V = \{\boldsymbol{w}_1, ..., \boldsymbol{w}_{|V|}\}$

- A system that does this is called a **Language Model**.

**Figure 14:** Slides borrowed from CS224n lecture, Jan 22

# Learning Distributions — Language model

## Language Modeling

- You can also think of a Language Model as a system that assigns probability to a piece of text.

- For example, if we have some text $x^{(1)}, \ldots, x^{(T)}$, then the probability of this text (according to the Language Model) is:

$$
\begin{aligned}
P(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}) &= P(\boldsymbol{x}^{(1)}) \times P(\boldsymbol{x}^{(2)} | \, \boldsymbol{x}^{(1)}) \times \cdots \times P(\boldsymbol{x}^{(T)} | \, \boldsymbol{x}^{(T-1)}, \ldots, \boldsymbol{x}^{(1)}) \\
&= \prod_{t=1}^{T} \underbrace{P(\boldsymbol{x}^{(t)} | \, \boldsymbol{x}^{(t-1)}, \ldots, \boldsymbol{x}^{(1)})}_{\text{This is what our LM provides}}
\end{aligned}
$$

**Figure 15:** Slides borrowed from CS224n lecture, Jan 22

# Learning Distributions — Language model

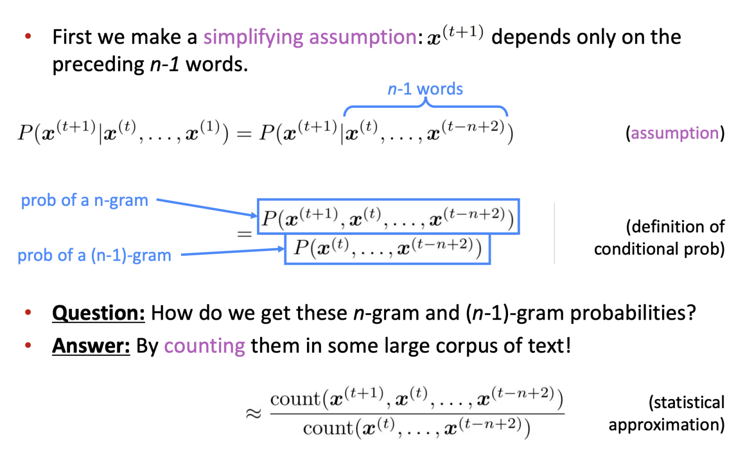



**Figure 16:** Slides borrowed from CS224n lecture, Jan 22

# Learning Distributions — Language model

## Storage Problems with n-gram Language Models

**Storage**: Need to store count for all *n*-grams you saw in the corpus.

$$P(\boldsymbol{w}|\text{students opened their}) = \frac{\boxed{\text{count}(\text{students opened their } \boldsymbol{w})}}{\text{count}(\text{students opened their})}$$

Increasing *n* or increasing corpus increases model size!

**Figure 17:** Slides borrowed from CS224n lecture, Jan 22

# Learning Distributions — Language model

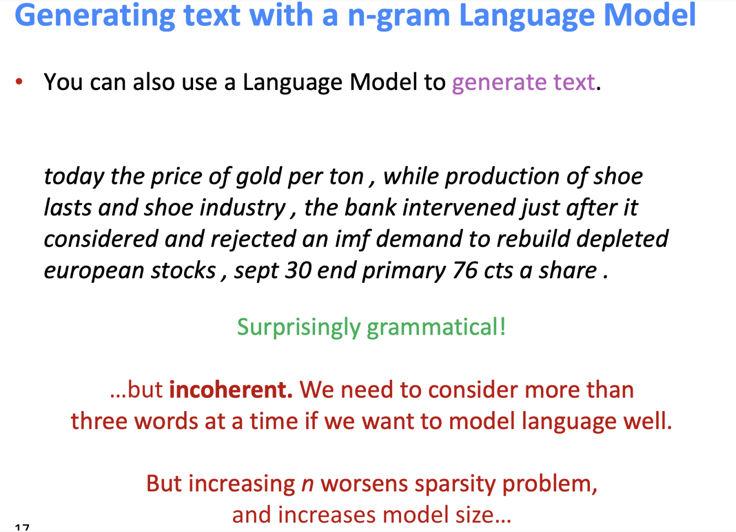



**Figure 18:** Slides borrowed from CS224n lecture, Jan 22

# Learning Distributions — Language model

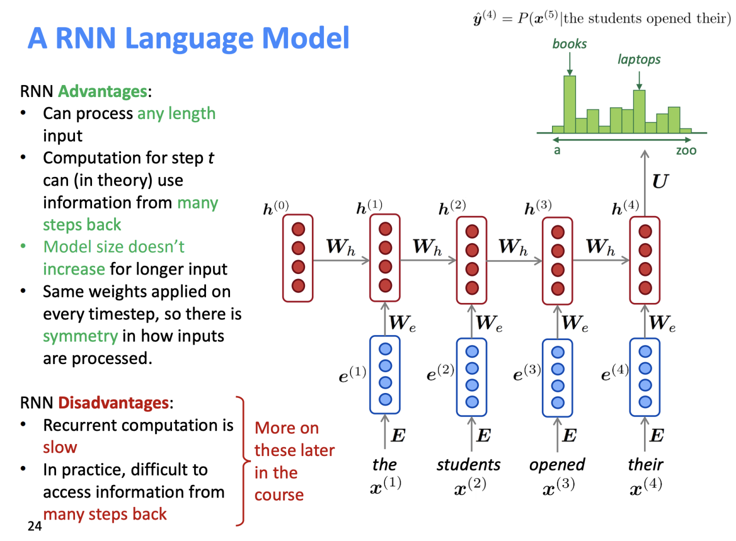



**Figure 19:** Slides borrowed from CS224n lecture, Jan 22

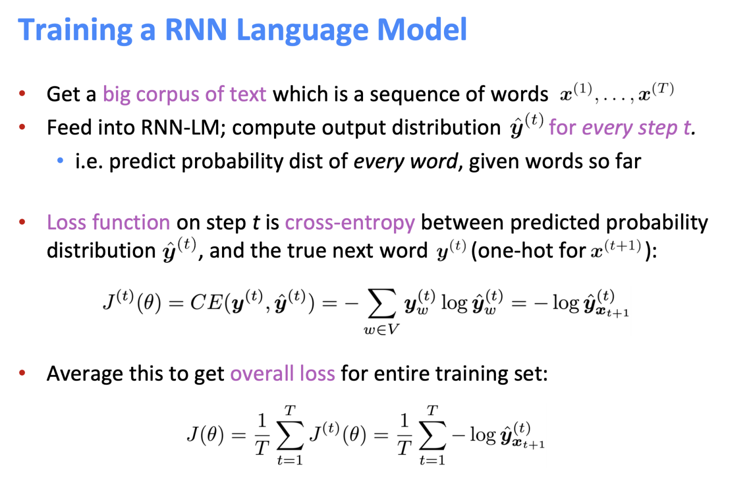



**Figure 20:** Slides borrowed from CS224n lecture, Jan 22

# Learning Distributions — Language model

## Evaluating Language Models

- The standard evaluation metric for Language Models is perplexity.

$$\text{perplexity} = \underbrace{\prod_{t=1}^{T} \left( \frac{1}{P_{\text{LM}}(\boldsymbol{x}^{(t+1)} | \, \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})} \right)}_{\text{Inverse probability of corpus, according to Language Model}}^{1/T}$$

Normalized by number of words

- This is equal to the exponential of the cross-entropy loss $J(\theta)$:

$$= \prod_{t=1}^{T} \left( \frac{1}{\hat{\boldsymbol{y}}^{(t)}_{\boldsymbol{x}_{t+1}}} \right)^{1/T} = \exp\left( \frac{1}{T} \sum_{t=1}^{T} -\log \hat{\boldsymbol{y}}^{(t)}_{\boldsymbol{x}_{t+1}} \right) = \exp(J(\theta))$$

$\boxed{\textbf{Lower perplexity is better!}}$

41

**Figure 21:** Slides borrowed from CS224n lecture, Jan 22

## Why should we care about Language Modeling?

- Language Modeling is a benchmark task that helps us measure our progress on understanding language

- Language Modeling is a subcomponent of many NLP tasks, especially those involving generating text or estimating the probability of text:
    - Predictive typing
    - Speech recognition
    - Handwriting recognition
    - Spelling/grammar correction
    - Authorship identification
    - Machine translation
    - Summarization
    - Dialogue
    - etc.

**Figure 22:** Slides borrowed from CS224n lecture, Jan 22

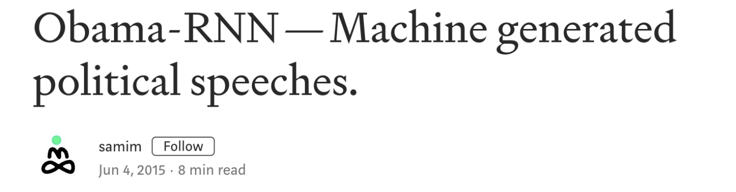
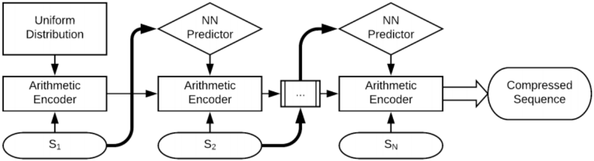



**Figure 23:** Slides borrowed from CS224n lecture, Jan 22

# Learning Distributions — Language model

**DeepZip: Lossless Data Compression using Recurrent Neural Networks**

Mohit Goyal[†,γ], Kedar Tatwawadi[*], Shubham Chandak[*] and Idoia Ochoa[γ]




a) Encoder Framework

**Figure 24:** DeepZip Language compressor based on Language models

# ML for Lossy Compression

# Representation learning for Compression

- More recently, ML is being used for lossy compression of data
- **Why use ML?**
    1. Unclear, high-dimensional data models. Eg: natural images.

# Representation learning for Compression

- More recently, ML is being used for lossy compression of data
- **Why use ML?**
    1. Unclear, high-dimensional data models. Eg: natural images.
    2. Unclear Loss functions for lossy compression
    3. EG: Image Compression $\rightarrow$ "Human perception loss"

# ML for CompressionI

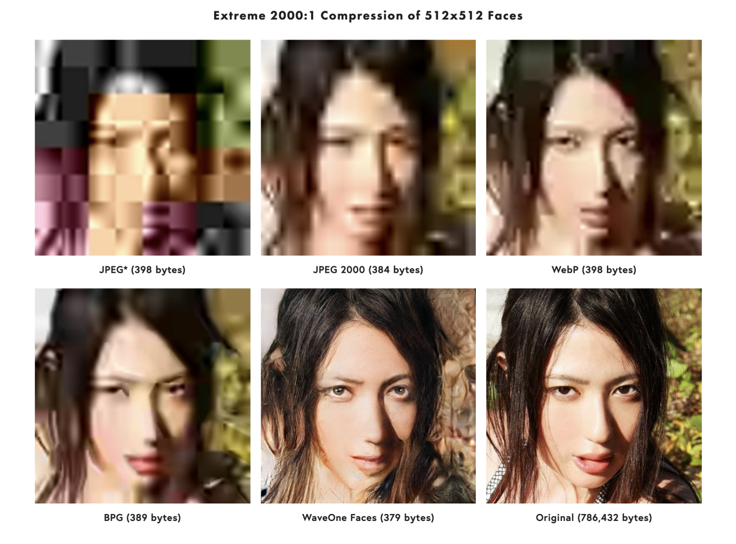



**Figure 25:** Waveone image compression using neural networks

# ML for CompressionI

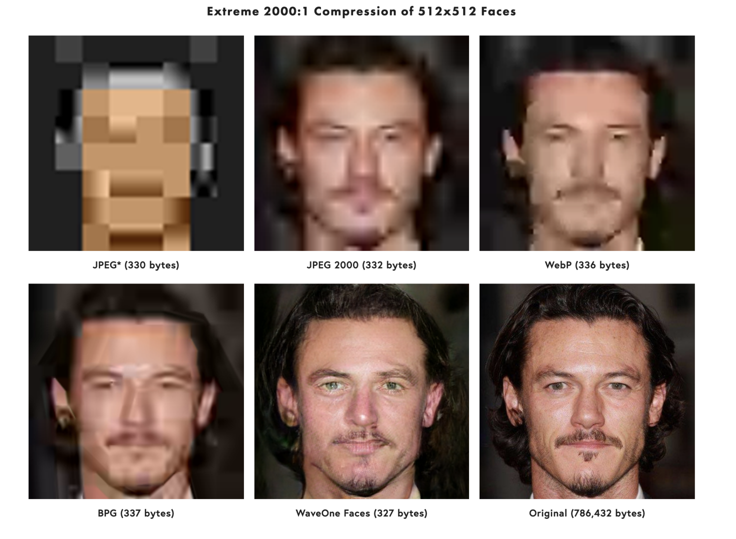



**Figure 26:** Waveone image compression using neural networks

# ICLR Compression Contest

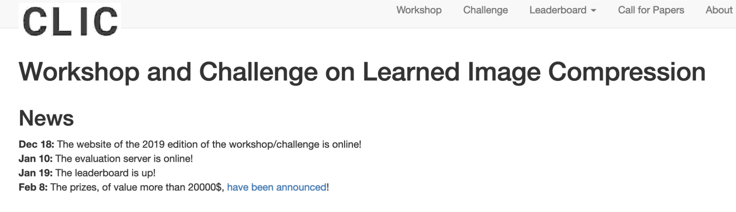



**Figure 27:** More details: https://www.compression.cc

# ML for Compression

- Lots of very different types of techniques used. The core idea is:
  1. Learn a "smooth" representation for the Image
  2. Quantize the representation
  3. Entropy coding of the representation

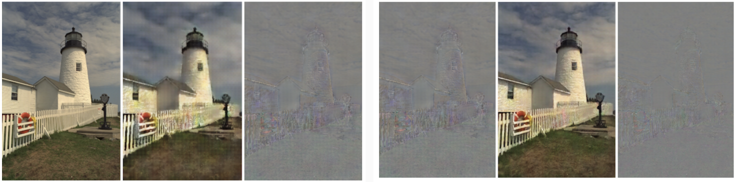
# ML for Compression

- Lots of very different types of techniques used. The core idea is:
    1. Learn a "smooth" representation for the Image
    2. Quantize the representation
    3. Entropy coding of the representation
- The "smoothness" of the representation is the key to good lossy compression.
- Different techniques used for compression: Autoencoders, VAE, GANs

# ML for Compression

Some notable papers:

- Toderici, George, et al. "Full resolution image compression with recurrent neural networks." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2017.



**Left:** Original image, I = R[0]. **Center:** Reconstructed image, P[1]. **Right:** the residual, R[1], which represents the error introduced by compression.

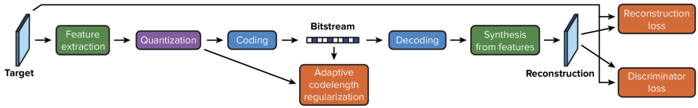
The second pass through the network. **Left:** R[1] is given as input. **Center:** A higher quality reconstruction, P[2]. **Right:** A smaller residual R[2] is generated by subtracting P[2] from the original image.

**Figure 28:** Compression over multiple iterations

# ML for Compression

Some notable papers:

- Rippel et.al, "Real-time adaptive Image compression" ICML'17 Proceedings



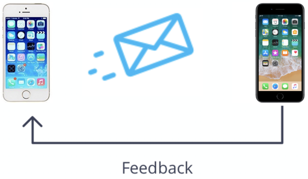

*The overall architecture of our model. The* ***feature extractor*** *discovers structure and reduces redundancy via the pyramidal decomposition and interscale alignment modules. The lossless* ***coding scheme*** *further compresses the quantized tensor via bitplane decomposition and adaptive arithmetic coding. The* ***adaptive codelength regularization*** *modulates the expected codelength to a prescribed target bitrate. Distortions between the target and its reconstruction are penalized by the* ***reconstruction loss****. The* ***discriminator loss*** *encourages visually pleasing reconstructions by penalizing discrepancies between their distributions and the targets'.*

**Figure 29:** Using a GAN based loss

# ML for Channel Coding

## Learning a code

## for channels with feedback




H. Kim, Y. Jiang, S. Kannan, S. Oh, P. Viswanath, "***Discovering feedback codes via deep learning***", 2018

**Figure 30:** Using a RNN for channel coding

- AWGN channel from transmitter to receiver
- Output fed back to the transmitter

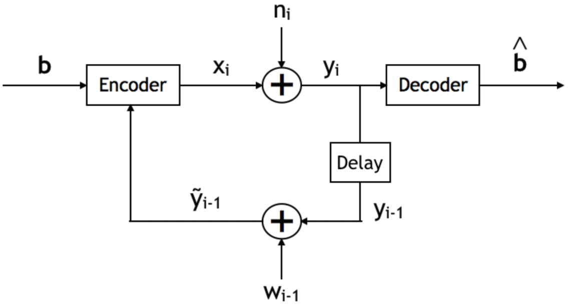



**Figure 31:** Using a RNN for channel coding

# ML for Channel Coding

- Robust to noise in the feedback

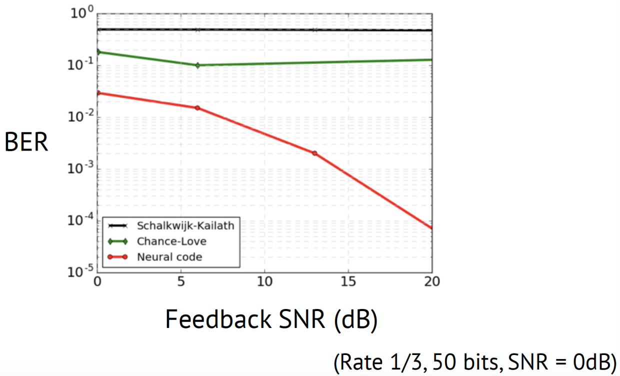



**Figure 32:** Using a RNN for channel coding

# ML for Joint Source-Channel Coding

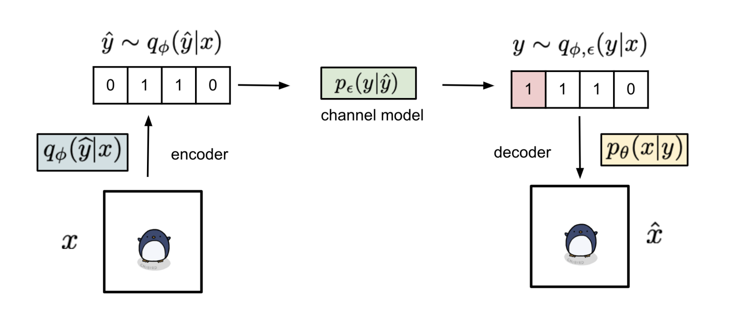



**Figure 33:** NECST: Neural Joint Source-Channel Code, Choi et.al. arxiv

# Conclusion?

ML/Statistics & Information theory are two sides of the same coin!

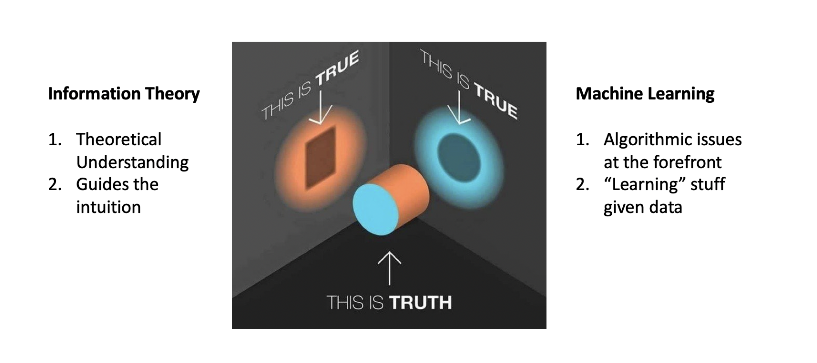



**Figure 34:** ML and IT

**Thank You!**