

# Information Theory meets Machine Learning

---

Kedar Tatwawadi

EE376a Course

# Table of contents

1. Introduction
2. Unsupervised Learning
3. Learning Data Distribution
4. ML for Lossy Compression

# Introduction

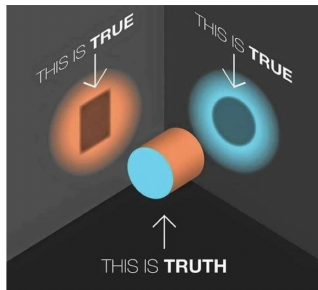
---

# ML and IT

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

## Information Theory

1. Theoretical Understanding
2. Guides the intuition



## Machine Learning

1. Algorithmic issues at the forefront
2. “Learning” stuff given data

**Figure 1: ML and IT**



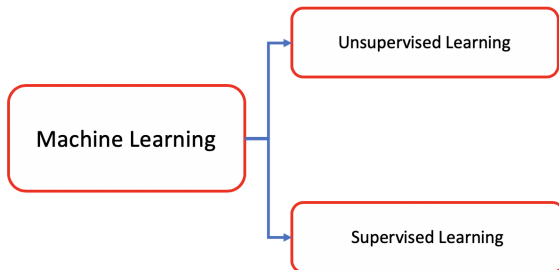
# A short (very) intro to ML



Machine Learning

**Figure 2:** Machine zoo

# A short (very) intro to ML



**Figure 3:** ML Zoo

# A short (very) intro to ML

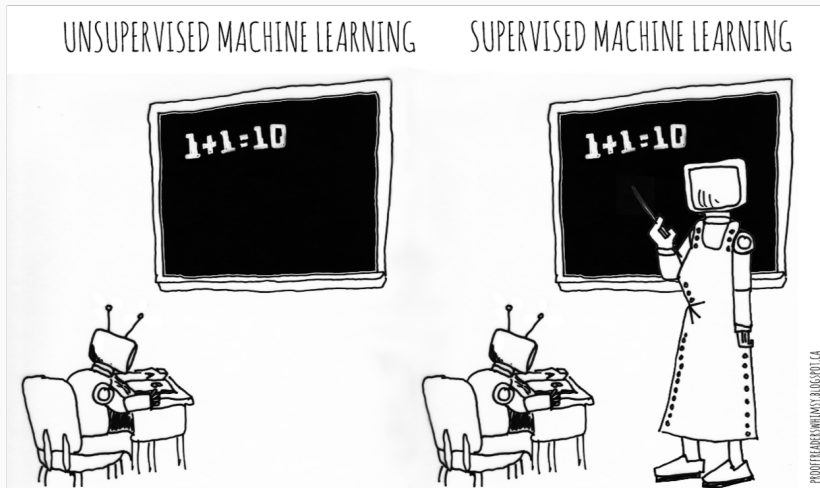


Figure 4: ML Zoo

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



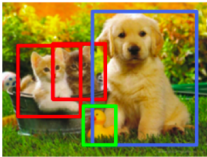

Given data tuples  $(X_1, y_1), (X_2, y_2), \dots, (X_N, y_N)$ , find a function  $F$  such that:

$$F(X) = y$$

# Supervised Learning

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

$$F(X) = y$$

Classification	Classification + Localization	Object Detection	Instance Segmentation
			
CAT	CAT	CAT, DOG, DUCK	CAT, DOG, DUCK

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

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

Take CS229, CS231n courses!



# A short (very) intro to ML

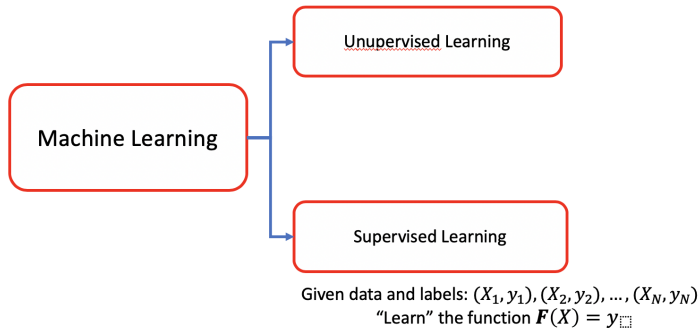


Figure 5: ML Zoo

# A short (very) intro to ML

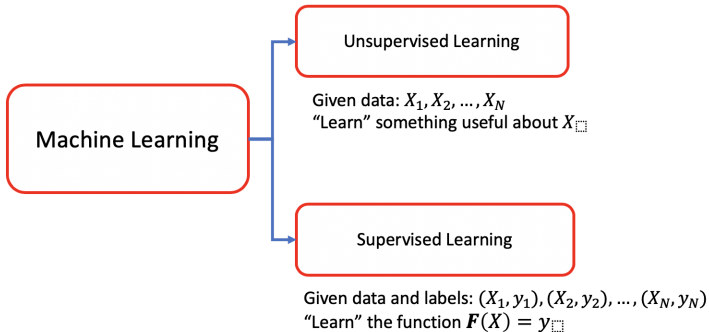


Figure 6: ML Zoo

# Unsupervised Learning

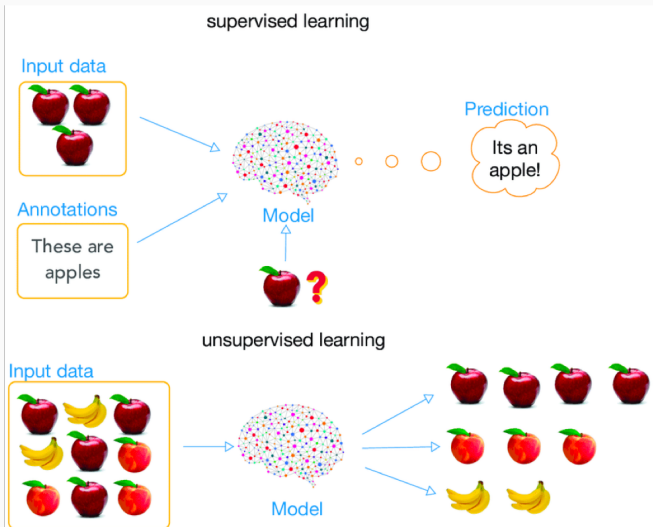
---

Given data:  $X_1, X_2, X_3, \dots, X_N$

"Learn" something useful about  $X$

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

# Clustering



**Figure 7:** Clustering

# Data Representation

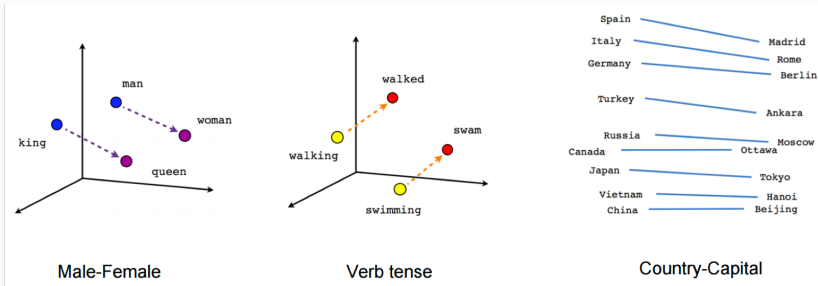
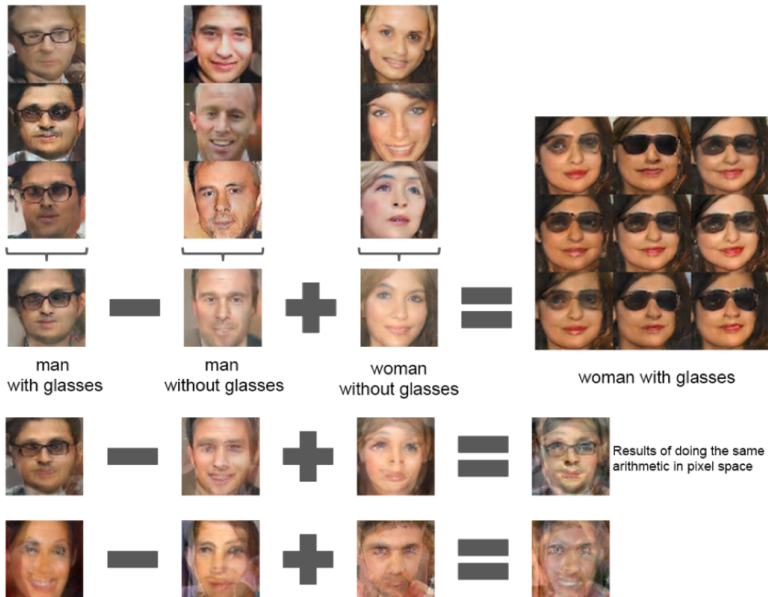


Figure 8: Word2Vec Representation

# Data Representation



# Learning Data Distribution

---



# Learning the distribution

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

Given data:  $X^{(1)}, X^{(2)}, \dots, X^{(N)}$  with distribution  $p_X(X)$ , **How do we learn  $p_X(X)$ ?**

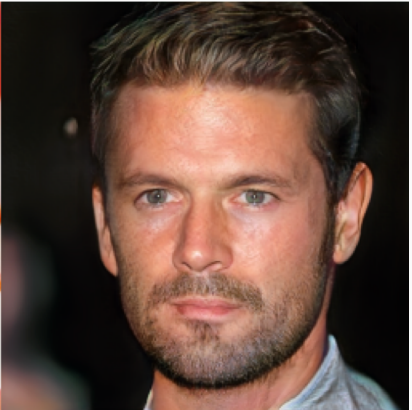
"Learn" the underlying **Distribution of the data**

Given data:  $X^{(1)}, X^{(2)}, \dots, X^{(N)}$  with distribution  $p_X(X)$ , **How do we learn  $p_X(X)$ ?**

**Use cases**

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

# Sampling





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**

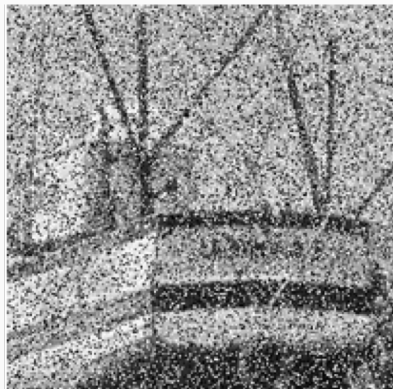
how to **make french toast**

Google Search

I'm Feeling Lucky

*Report inappropriate predictions*

# Denoising



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

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

# Learning the distribution

Data:  $X^{(1)}, X^{(2)}, \dots, X^{(N)}$  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:  $X^{(1)}, X^{(2)}, \dots, X^{(N)}$  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:  $X^{(1)}, X^{(2)}, \dots, X^{(N)}$  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  $p_X(X)$

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

# Learning the distribution

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

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

# Learning the distribution

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

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

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

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

- In practice we consider empirical expectation instead:

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

- In practice we consider empirical expectation instead:

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

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

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

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

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

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



# Data Example

$\longleftrightarrow F \text{ features} \longrightarrow$

$n \text{ records} \updownarrow$

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

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

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

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

## $Q_I$ independent distributions

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

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

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

# Tabular Example

$n$  records

$F$  features

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

- 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  $Y_{i_j}, i_j < i$

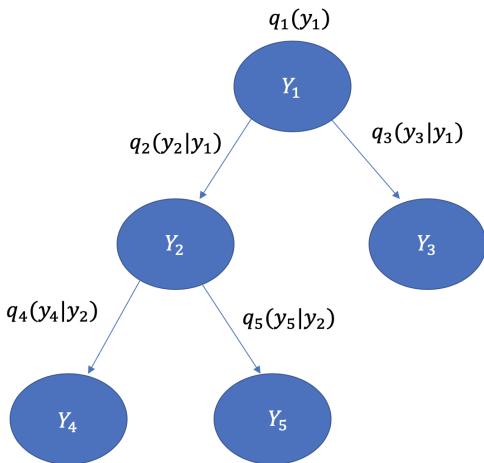
# Tree-based distributions

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

## Tree-based distributions — Examples

- Example tree distribution:

$$q(Y_1, Y_2, \dots, Y_5) = q_1(Y_1)q_2(Y_2|Y_1)q_3(Y_3|Y_1)q_4(Y_4|Y_2)q_5(Y_5|Y_2)$$

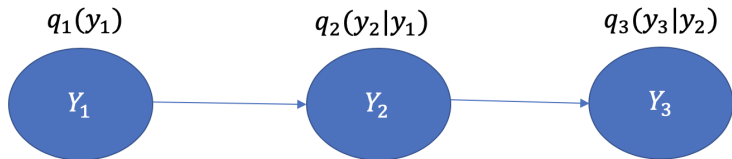




## Tree-based distributions — Examples

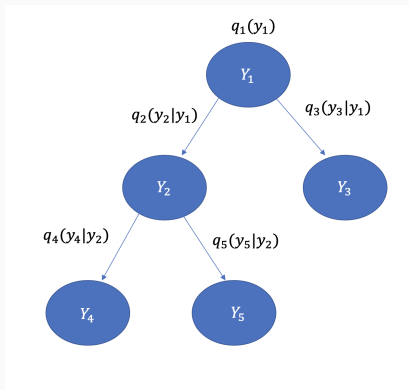
- Example tree distribution:

$$q(Y_1, Y_2, Y_3) = q_1(Y_1)q_2(Y_2|Y_1)q_3(Y_3|Y_2)$$



**Figure 11:** Graph example

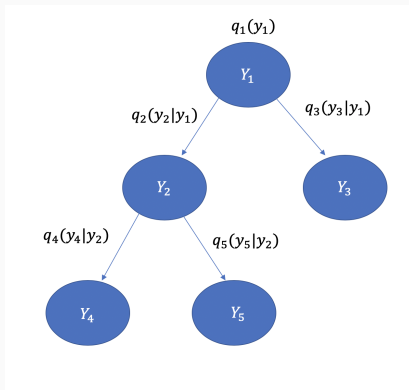
# Tree-based distributions



**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):  
 $Y_1 \rightarrow Y_2 \rightarrow Y_3 \rightarrow Y_4 \rightarrow Y_5$

# 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)

- 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:

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

- 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:

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

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

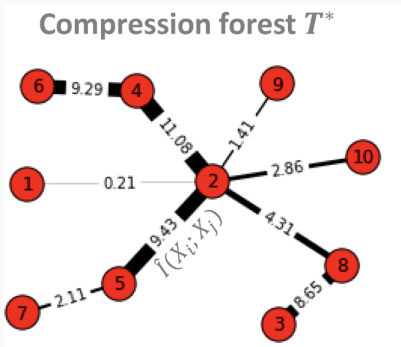
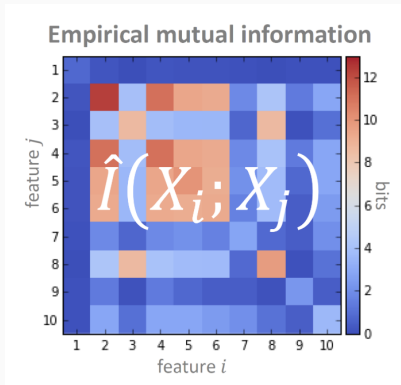
# Data Example

$\longleftrightarrow F \text{ features} \longrightarrow$

$n \text{ records} \updownarrow$	UserID	Age	Locatio	Device	Time	DocID	...
	4324234	25	90210	iPhone 7	9pm	33221	...
	1223231	49	94087	iPad pro	10am	66543	...
	...	...	...	...	...	...	...

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

# Data Example





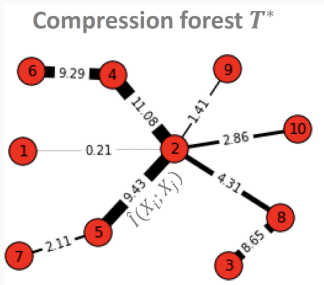
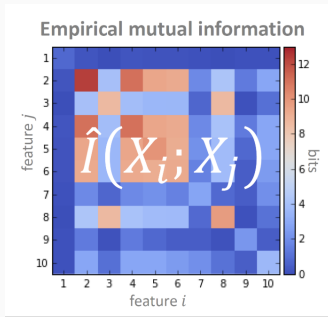
# Chow-Liu Tree Algorithm

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

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

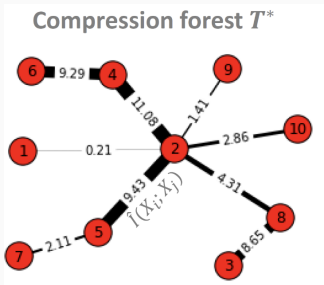
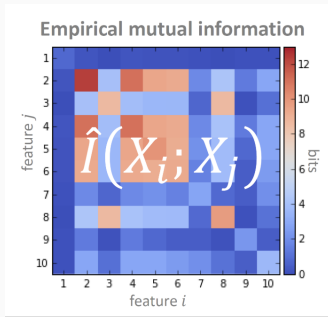
- **Intuition::** Add edges which have "high" correlation.
- **We will prove this in the class!**

# Practical Considerations



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

# Practical Considerations



- Exhaustive 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

- 

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

$G$  is a solution to the problem:

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

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

# Practical Considerations

- 

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

$G$  is a solution to the problem:

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

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

- We really want to solve the problem:

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

# Practical Considerations

- 

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

$G$  is a solution to the problem:

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

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

- We really want to solve the problem:

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

- 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.

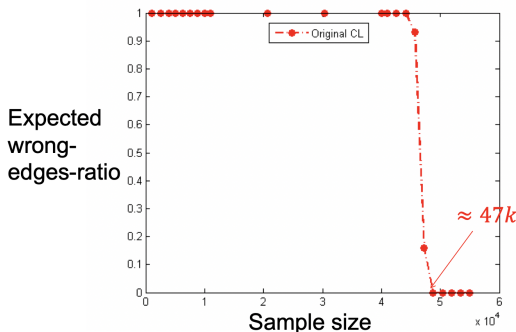
For details about how it works, please refer to our paper 'Minimax Estimation of Functionals of Discrete Distributions', IEEE Transactions on Information Theory, Vol.61, Issue 5, pp 2835-2885, May 2015. For details about how to use it in Matlab or Python, please checkout our Github repo below:

[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!

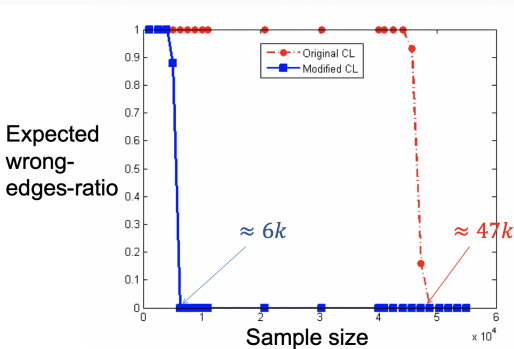


Setting: star graph, 7 nodes, each node alphabet size 300



# Practical Considerations

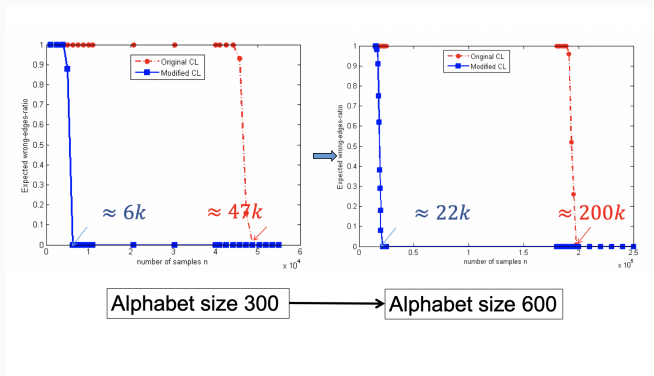
- Using  $N$  samples we can have better estimators for  $I(Y_i, Y_j)$  than the empirical plug-in estimator  $\hat{I}(Y_i, Y_j)$
- Information theory helps us get better estimators!



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:

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

$G$  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:

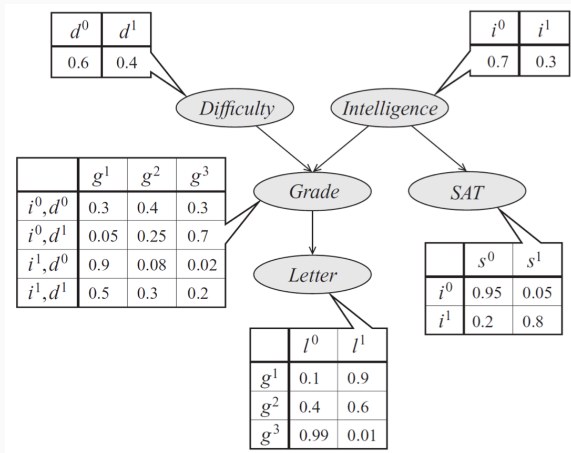
$$G = \operatorname{argmax} \sum_{\text{edges}(i,j)} \hat{I}(Y_i; Y_j) \quad (5)$$

$G$  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

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

# General Bayesian Networks



# Chow-Liu Algorithm for Bayesian Networks

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

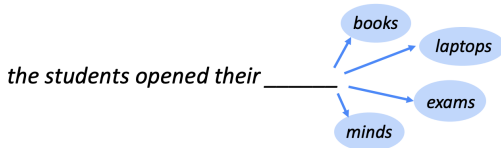
For general bayesian networks:

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

- 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

# Learning Distributions — Language model

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



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

$$P(x^{(t+1)} | x^{(t)}, \dots, x^{(1)})$$

where  $x^{(t+1)}$  can be any word in the vocabulary  $V = \{w_1, \dots, w_{|V|}\}$

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

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

## 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  $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(T)}$ , then the probability of this text (according to the Language Model) is:

$$\begin{aligned} P(\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(T)}) &= P(\mathbf{x}^{(1)}) \times P(\mathbf{x}^{(2)} | \mathbf{x}^{(1)}) \times \dots \times P(\mathbf{x}^{(T)} | \mathbf{x}^{(T-1)}, \dots, \mathbf{x}^{(1)}) \\ &= \prod_{t=1}^T P(\mathbf{x}^{(t)} | \mathbf{x}^{(t-1)}, \dots, \mathbf{x}^{(1)}) \end{aligned}$$

This is what our LM provides

Figure 15: Slides borrowed from CS224n lecture, Jan 22



# Learning Distributions — Language model

- First we make a **simplifying assumption**:  $\mathbf{x}^{(t+1)}$  depends only on the preceding  $n-1$  words.

$$P(\mathbf{x}^{(t+1)} | \mathbf{x}^{(t)}, \dots, \mathbf{x}^{(1)}) = P(\mathbf{x}^{(t+1)} | \mathbf{x}^{(t)}, \dots, \mathbf{x}^{(t-n+2)}) \quad (\text{assumption})$$

*n-1 words*

prob of a  $n$ -gram

prob of a  $(n-1)$ -gram

$$\begin{aligned} &= \boxed{P(\mathbf{x}^{(t+1)}, \mathbf{x}^{(t)}, \dots, \mathbf{x}^{(t-n+2)})} \\ &\quad \boxed{P(\mathbf{x}^{(t)}, \dots, \mathbf{x}^{(t-n+2)})} \end{aligned}$$

(definition of  
conditional prob)

- Question:** How do we get these  $n$ -gram and  $(n-1)$ -gram probabilities?
- Answer:** By **counting** them in some large corpus of text!

$$\approx \frac{\text{count}(\mathbf{x}^{(t+1)}, \mathbf{x}^{(t)}, \dots, \mathbf{x}^{(t-n+2)})}{\text{count}(\mathbf{x}^{(t)}, \dots, \mathbf{x}^{(t-n+2)})} \quad (\text{statistical approximation})$$

Figure 16: Slides borrowed from CS224n lecture, Jan 22

## Storage Problems with $n$ -gram Language Models

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

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

Increasing  $n$  or increasing corpus increases model size!

Figure 17: Slides borrowed from CS224n lecture, Jan 22

## Generating text with a n-gram Language Model

- You can also use a Language Model to **generate text**.

*today the price of gold per ton , while production of shoe lasts and shoe industry , the bank intervened just after it considered and rejected an imf demand to rebuild depleted european stocks , sept 30 end primary 76 cts a share .*

Surprisingly grammatical!

...but **incoherent**. We need to consider more than three words at a time if we want to model language well.

But increasing  $n$  worsens sparsity problem,  
and increases model size...

17

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

## A RNN Language Model

### RNN Advantages:

- Can process **any** length input
- Computation for step  $t$  can (in theory) use information from **many** steps back
- Model size **doesn't increase** for longer input
- Same weights applied on every timestep, so there is **symmetry** in how inputs are processed.

### RNN Disadvantages:

- Recurrent computation is **slow**
  - In practice, difficult to access information from **many** steps back
- More on these later in the course

24

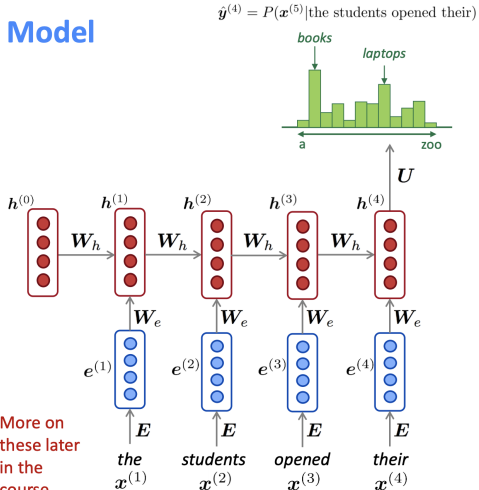


Figure 19: Slides borrowed from CS224n lecture, Jan 22

## Training a RNN Language Model

- Get a **big corpus of text** which is a sequence of words  $x^{(1)}, \dots, x^{(T)}$
- Feed into RNN-LM; compute output distribution  $\hat{y}^{(t)}$  **for every step  $t$** .
  - i.e. predict probability dist of *every word*, given words so far
- **Loss function** on step  $t$  is **cross-entropy** between predicted probability distribution  $\hat{y}^{(t)}$ , and the true next word  $y^{(t)}$  (one-hot for  $x^{(t+1)}$ ):

$$J^{(t)}(\theta) = CE(y^{(t)}, \hat{y}^{(t)}) = - \sum_{w \in V} y_w^{(t)} \log \hat{y}_w^{(t)} = - \log \hat{y}_{x_{t+1}}^{(t)}$$

- Average this to get **overall loss** for entire training set:

$$J(\theta) = \frac{1}{T} \sum_{t=1}^T J^{(t)}(\theta) = \frac{1}{T} \sum_{t=1}^T - \log \hat{y}_{x_{t+1}}^{(t)}$$

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

## Evaluating Language Models

- The standard **evaluation metric** for Language Models is **perplexity**.

$$\text{perplexity} = \prod_{t=1}^T \left( \frac{1}{P_{\text{LM}}(\mathbf{x}^{(t+1)} | \mathbf{x}^{(t)}, \dots, \mathbf{x}^{(1)})} \right)^{1/T}$$

← Normalized by number of words

Inverse probability of corpus, according to Language Model

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

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

**Lower perplexity is better!**

## 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

## Obama-RNN — Machine generated political speeches.



samim

Follow

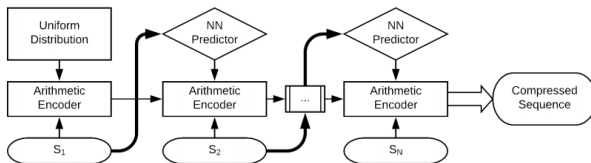
Jun 4, 2015 · 8 min read

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



## DeepZip: Lossless Data Compression using Recurrent Neural Networks

Mohit Goyal<sup>†,γ</sup>, Kedar Tatwawadi<sup>\*</sup>, Shubham Chandak<sup>\*</sup> and Idoia Ochoa<sup>γ</sup>



a) Encoder Framework

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

# ML for Lossy Compression

---

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

- 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"

Extreme 2000:1 Compression of 512x512 Faces



JPEG\* (398 bytes)



JPEG 2000 (384 bytes)



WebP (398 bytes)



BPG (389 bytes)



WaveOne Faces (379 bytes)



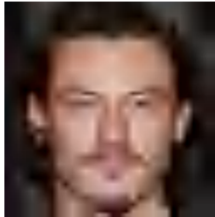
Original (786,432 bytes)

Figure 25: Waveone image compression using neural networks

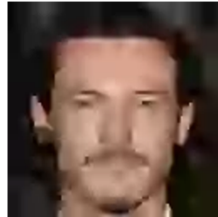
Extreme 2000:1 Compression of 512x512 Faces



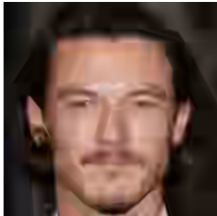
JPEG\* (330 bytes)



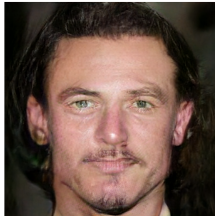
JPEG 2000 (332 bytes)



WebP (336 bytes)



BPG (337 bytes)

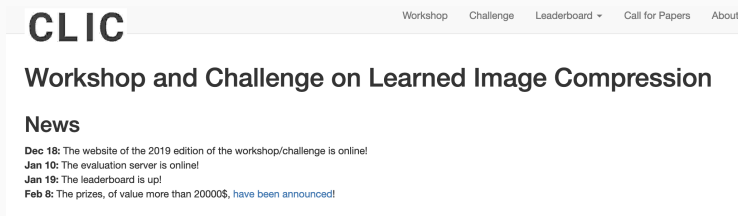


WaveOne Faces (327 bytes)



Original (786,432 bytes)

Figure 26: Waveone image compression using neural networks



The screenshot shows the top portion of the CLIC website. At the top left is the logo 'CLIC' in large, bold, black letters. To its right is a horizontal navigation bar with links: 'Workshop', 'Challenge', 'Leaderboard' (with a dropdown arrow), 'Call for Papers', and 'About'. Below the navigation bar is a large heading 'Workshop and Challenge on Learned Image Compression'. Underneath this heading is a section titled 'News'. The 'News' section contains four entries, each with a date and a brief announcement: 'Dec 18: The website of the 2019 edition of the workshop/challenge is online!', 'Jan 10: The evaluation server is online!', 'Jan 19: The leaderboard is up!', and 'Feb 8: The prizes, of value more than 20000\$, have been announced!'.

**CLIC**

Workshop Challenge Leaderboard ▾ Call for Papers About

## Workshop and Challenge on Learned Image Compression

### News

**Dec 18:** The website of the 2019 edition of the workshop/challenge is online!

**Jan 10:** The evaluation server is online!

**Jan 19:** The leaderboard is up!

**Feb 8:** The prizes, of value more than 20000\$, [have been announced!](#)

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

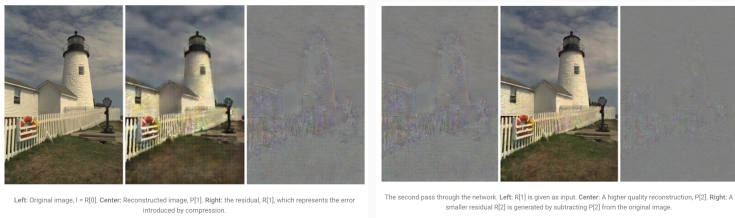
- Lots of very different types of techniques used. The core idea is:
  1. Learn a "smooth" representation for the Image
  2. Quantize the representation
  3. Entropy coding of the representation



- 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

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.



**Figure 28:** Compression over multiple iterations

Some notable papers:

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

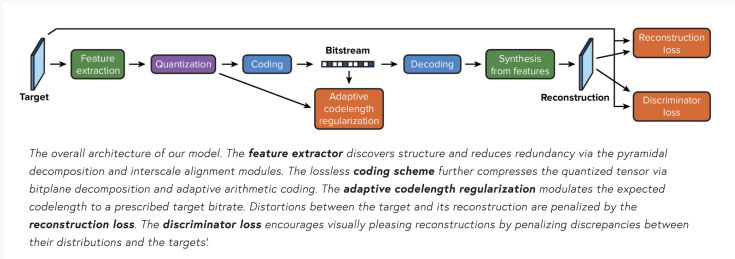
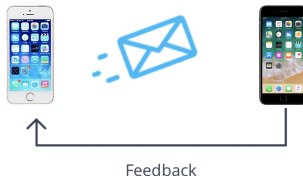


Figure 29: Using a GAN based loss

## Learning a code for channels with feedback

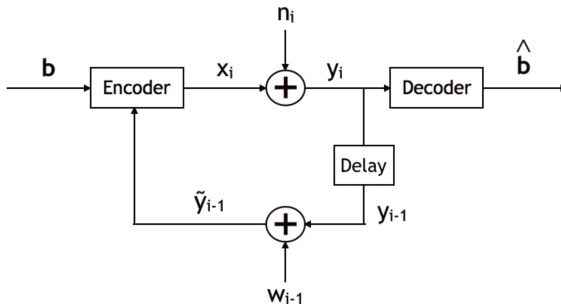


H. Kim, Y. Jiang, S. Kannan, S. Oh, P. Viswanath, “*Discovering feedback codes via deep learning*”, 2018

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

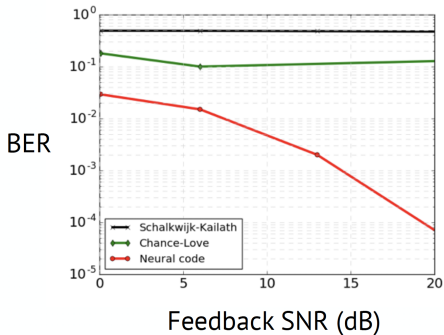
# ML for Channel Coding

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



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

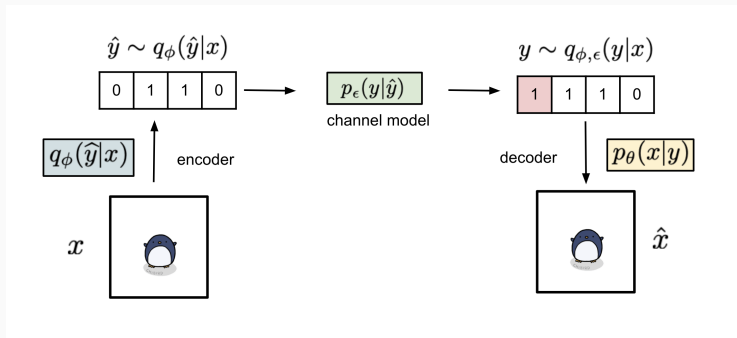
- Robust to noise in the feedback



(Rate 1/3, 50 bits, SNR = 0dB)

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

# ML for Joint Source-Channel Coding



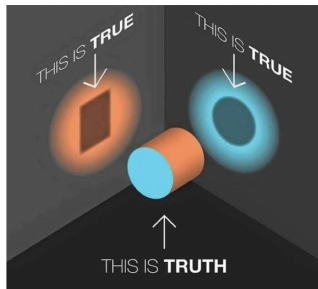
**Figure 33:** NECST: Neural Joint Source-Channel Code, Choi et.al. arxiv

# Conclusion?

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

## Information Theory

1. Theoretical Understanding
2. Guides the intuition



## Machine Learning

1. Algorithmic issues at the forefront
2. “Learning” stuff given data

**Figure 34:** ML and IT



**Thank You!**