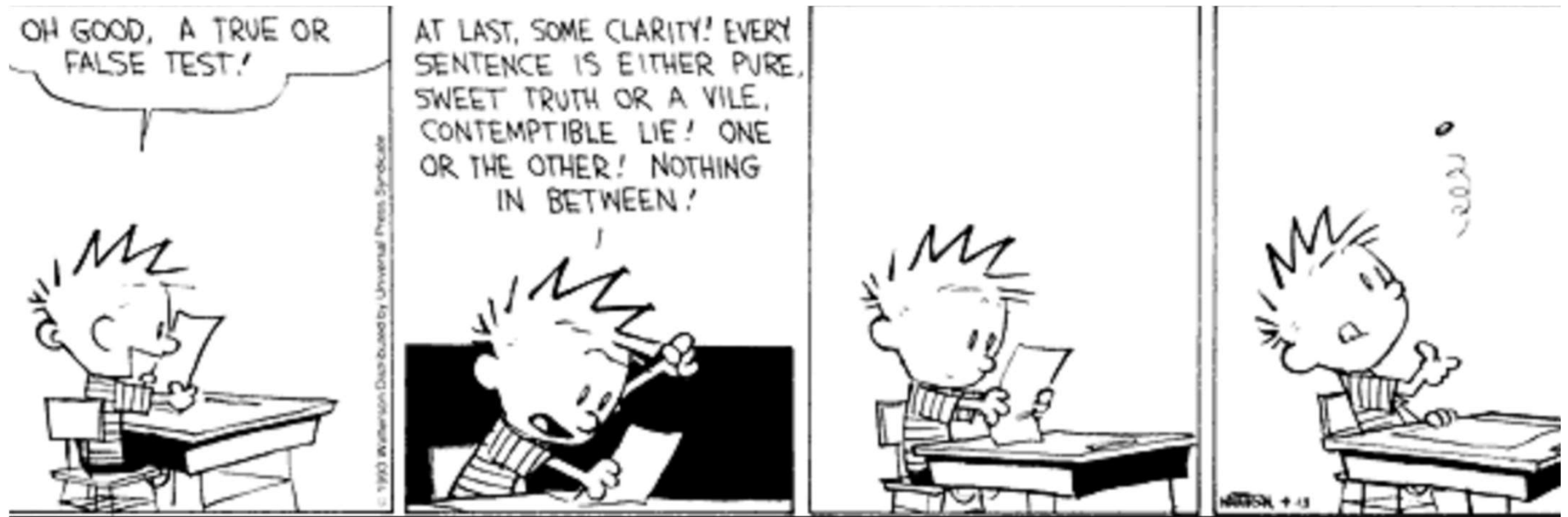


Machine Learning

EN.601.475/675

DR. PHILIP GRAFF



Final Exam Review

The Final

Released on Gradescope at 2:00 PM EDT, Wednesday May 6

3 hours permitted

- I believe Gradescope measures this from the time you start

Is designed to be 2 hours or less, so you should have plenty of time

Content (250 pts total):

- 20 T/F with explanation (6 pts each, 3+3)
 - Explanation of 1 sentence is sufficient, no more than 2 are needed
- 8 multiple choice (5 pts each)
- 5 multi-part questions with short to medium-length written answers
 - 8-28 pts per question
 - Each has 2-7 sub-questions

Will primarily focus on “new material” (i.e. not covered by midterm)



Old material



Linear Regression



Logistic Regression



Perceptron



SVM



Kernel methods



Decision trees



Boosting



Deep learning (Part 1)

New material



Deep Learning (Part 2)



Clustering



Expectation-Maximization



Dimensionality Reduction



Graphical Models



Structured Prediction



Practical Machine Learning

Deep Learning – Part 2

Methods to prevent over-fitting

- Regularization
- Large data sets
- Early stopping
- Training with dropout
 - “Thinned” networks

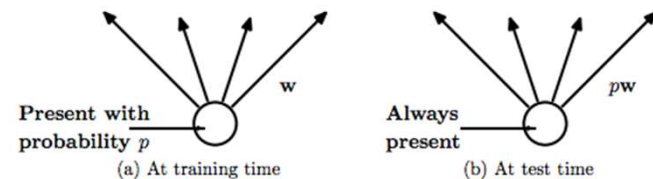
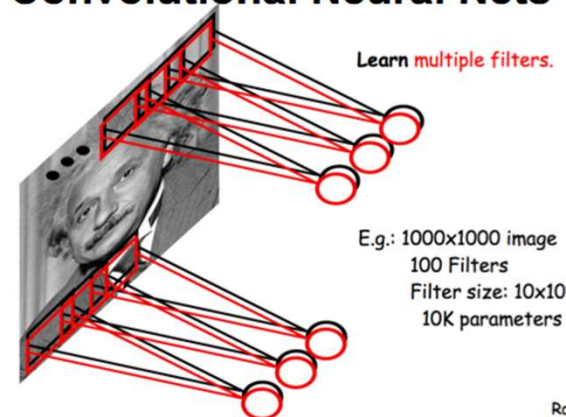


Figure 2: **Left:** A unit at training time that is present with probability p and is connected to units in the next layer with weights w . **Right:** At test time, the unit is always present and the weights are multiplied by p . The output at test time is same as the expected output at training time.

Convolutional networks

- Kernels to identify local structure
- Pooling to introduce spatial invariance

Convolutional Neural Nets



LeCun et al. "Gradient-based learning applied to document recognition" IEEE 1998

Clustering

Clustering

Data

$$\{(\mathbf{x}_i)\}_{i=1}^N \quad \mathbf{x}_i \in \mathbb{R}^M$$

Input: number of clusters, k

Algorithm: partition data into k clusters

- Each \mathbf{x} belongs to a cluster

Cluster: similar group of examples

Algorithm: K-Means

Given data

$$\{(\mathbf{x}_i)\}_{i=1}^N \quad \mathbf{x}_i \in \mathbb{R}^M$$

Initialize

$$\boldsymbol{\mu}_k$$

Iteratively update until convergence

$$r_{nk} = \begin{cases} 1 & \text{if } k = \operatorname{argmin}_j \|\mathbf{x}_n - \boldsymbol{\mu}_j\|^2 \\ 0 & \text{otherwise} \end{cases} \quad \boldsymbol{\mu}_k = \frac{\sum_n r_{nk} \mathbf{x}_n}{\sum_n r_{nk}}$$

Issues with K-Means

Computational complexity

- Re-assignment step
 - Vector distance: M operations
 - Find best cluster for each example: $K*N$ distances
 - Total: $O(KNM)$
- Compute new means
 - Each example added to a cluster once: $O(NM)$
- For T iterations, total is $O(TKNM)$
 - Linear in each variable

- Still slow compared to some supervised methods

Difficulty of finding optimal assignment

- NP-hard in Euclidean space (certainly non-convex)
- Solution: random restarts
 - How do we pick random starts?

Gaussian Mixture Models

Cluster Responsibilities:

$$\gamma(z_{nk}) = \frac{\pi_k \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^K \pi_j \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}$$
$$\pi_k = \frac{N_k}{N} = \frac{N_k}{\sum_k N_k}$$

Cluster means:

$$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{n=1}^N \gamma(z_{nk}) \mathbf{x}_n$$
$$N_k = \sum_n \gamma(z_{nk})$$

Cluster covariances:

$$\boldsymbol{\Sigma}_k = \frac{1}{N_k} \sum_{n=1}^N \gamma(z_{nk}) (\mathbf{x}_n - \boldsymbol{\mu}_k)(\mathbf{x}_n - \boldsymbol{\mu}_k)^T$$

Algorithm: GMMs

Given data

$$\{(\mathbf{x}_n)\}_{n=1}^N \quad \mathbf{x}_n \in \mathbb{R}^M$$

Initialize

$$\boldsymbol{\mu}_k \quad \boldsymbol{\Sigma}_k \quad \pi_k$$

Iteratively update until convergence

$$\gamma(z_{nk}) \quad \boldsymbol{\mu}_k \quad \boldsymbol{\Sigma}_k \quad \pi_k$$

Same Algorithm!

The maximization algorithm is the same for both!

Iterate two steps:

1. Compute the **expected** cluster assignments according to the current model
2. Then **maximize** the model parameters according to the current cluster assignments

Expectation-Maximization Algorithm (EM)



Expectation- Maximization

EM Algorithm

A general technique for maximizing likelihood when you have latent variables

- Latent variables are those you do not observe
- We never get to see examples of cluster assignments (unsupervised!)

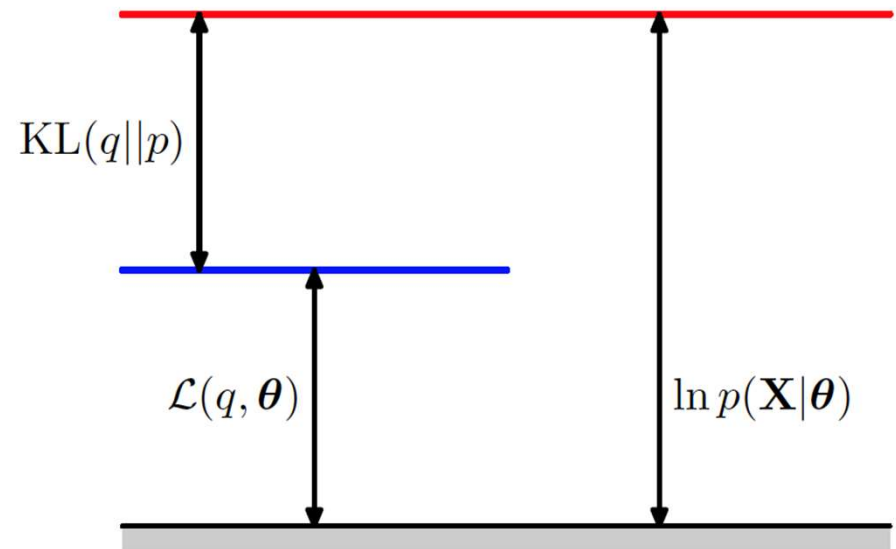
E-M allows us to write objective functions without seeing these variables

- Maximization step is familiar
 - Find the best parameters given the observations
- Expectation step is new
 - Pretend we see the latent variables

Pictorial View of EM

The log-likelihood is the sum of the two components

$KL(q||p) \geq 0$ implies that $\mathcal{L}(q, \theta)$ is a lower bound on $\log p(\mathbf{X}|\theta)$



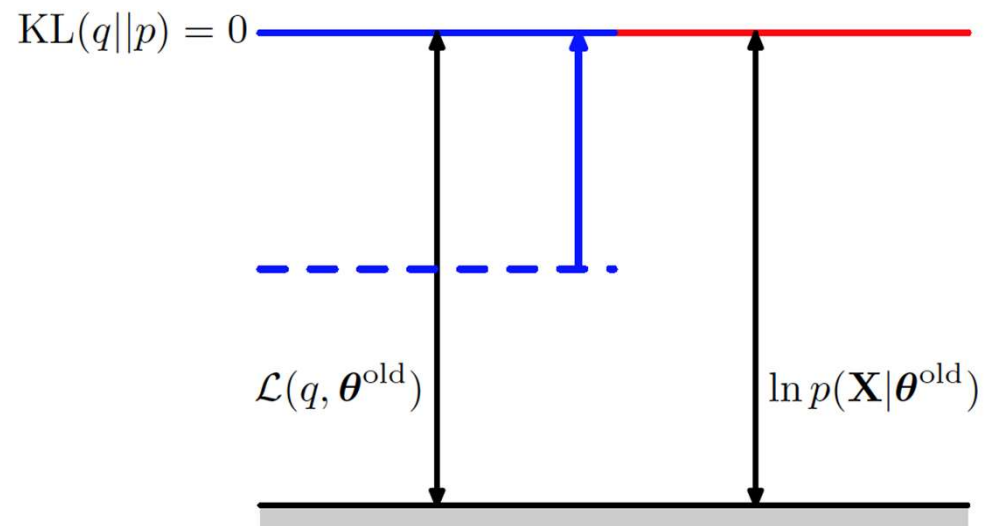
Pictorial View

In the E step, we set
 $q(\mathbf{Z}) = p(\mathbf{Z}|\mathbf{X}, \theta^{old})$

The “observed” variables \mathbf{Z} match
what we **expect** given parameters

$$\log p(\mathbf{X}|\theta) = \mathcal{L}(q, \theta) + \boxed{\text{KL}(q||p)}$$

Goes to 0



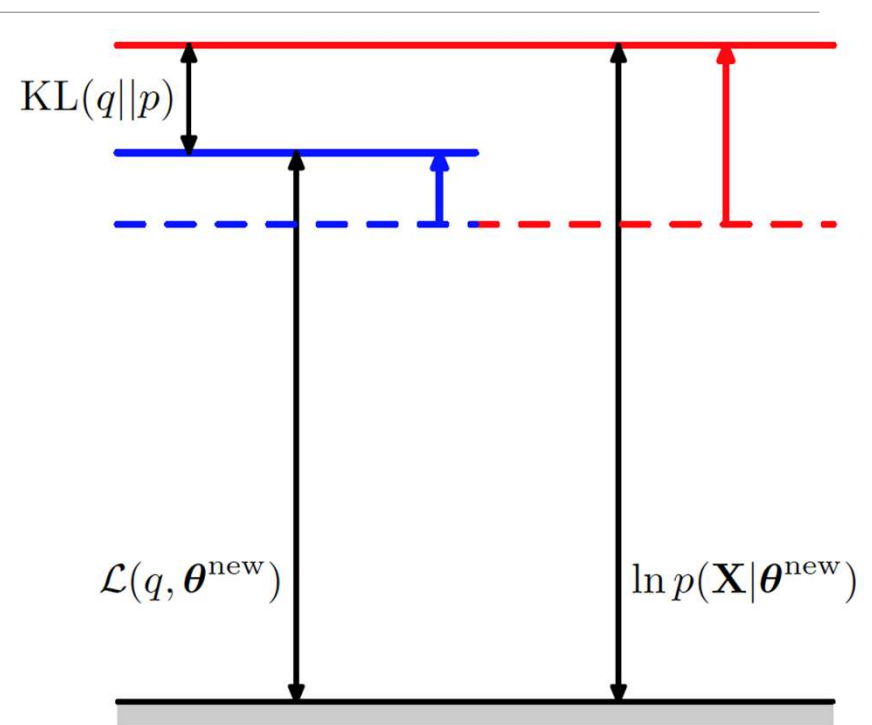
Pictorial View

In the M step, we maximize $\mathcal{L}(q, \theta)$

Find θ^{new} such that
 $\mathcal{L}(q, \theta^{new}) \geq \mathcal{L}(q, \theta^{old})$

Because $KL(q||p) \geq 0$ and θ changes,
KL will increase as well

$$\log p(\mathbf{X}|\theta) = \underbrace{\mathcal{L}(q, \theta)}_{\text{Increases}} + \underbrace{KL(q||p)}_{\text{Can only increase}}$$



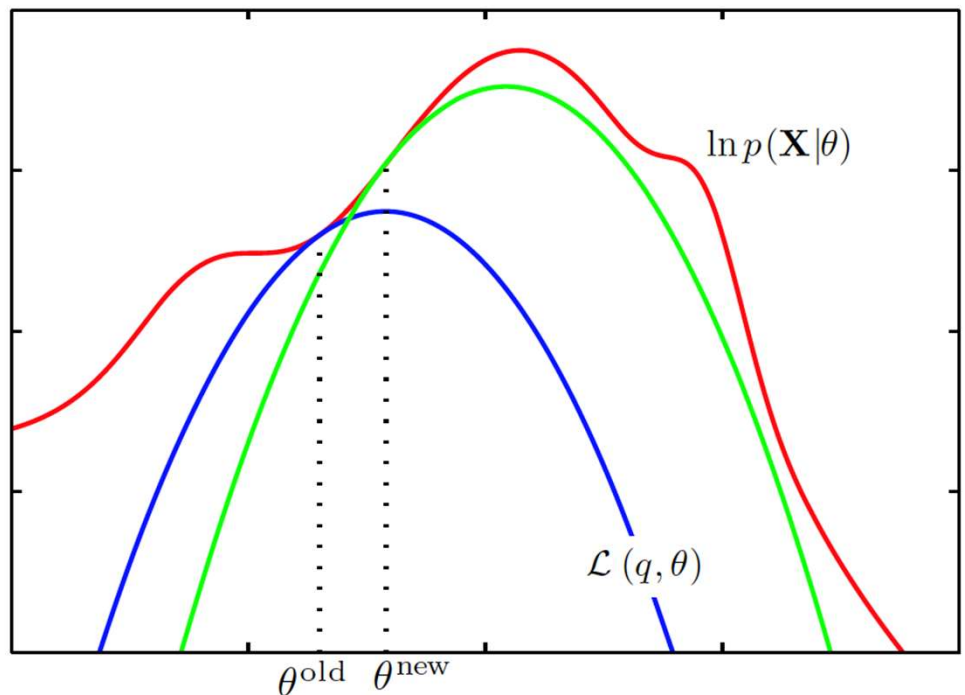
Pictorial view

Likelihood function in red

Using old parameters lower bound the likelihood using \mathcal{L} (blue)

Maximize \mathcal{L} to get new parameters

Next E step gives new lower bound (green)



Convergence

We now see why EM converges in general

- We are always increasing the likelihood function
- At some point we won't be able to increase it anymore

Very powerful result

- For any problem with latent variables, if you can write the complete data likelihood, you can use EM.
- The algorithm will always converge!

The General EM Algorithm

Goal: maximize a likelihood function $p(\mathbf{X}|\theta)$

- Write a joint distribution over the complete data $p(\mathbf{X}, \mathbf{Z}|\theta)$

Choose an initial setting for θ^{old}

E Step: compute $q(\mathbf{Z})$ as $p(\mathbf{Z}|\mathbf{X}, \theta^{old})$

M Step: Compute θ^{new} given by $\theta^{new} = \operatorname{argmax}_{\theta} Q(\theta, \theta^{old})$

Let $\theta^{old} = \theta^{new}$

Repeat until convergence

$$\mathcal{L}(q, \theta) = Q(\theta, \theta^{old}) + \text{constant}$$

where

$$Q(\theta, \theta^{old}) = \sum_{\mathbf{Z}} p(\mathbf{Z}|\mathbf{X}, \theta^{old}) \log(p(\mathbf{X}, \mathbf{Z}|\theta))$$

GMM with EM:

See lecture 12 for full description!

We can write the complete-data likelihood function for a dataset as:

$$p(\mathbf{X}, \mathbf{Z} | \theta) = \prod_{n=1}^N \prod_{k=1}^K \pi_k^{z_{nk}} \mathcal{N}(x_n | \mu_k, \Sigma_k)^{z_{nk}} \quad \theta = \{\mu_k, \Sigma_k, \pi_k\}_{k=1}^K$$

With the latent z variables, we have only products and can easily take the log of this function

$$\log p(\mathbf{X}, \mathbf{Z} | \theta) = \sum_{n=1}^N \sum_{k=1}^K z_{nk} (\log \pi_k + \log \mathcal{N}(x_n | \mu_k, \Sigma_k))$$

General EM

E step

- Finding the values for the hidden variables may not be easy
- We may need to approximate the values

M step

- Maximization may require multiple steps, optional constraints

Latent Variables

E-M is useful for latent variables

- Variables that you do not observe

What is the structure of these latent variables?

- How do they influence the observed variables?
- Can you have multiple latent variables in a complex structure?

We need some way to talk about these variables formally



Dimensionality Reduction

Dimensionality Reduction

Do we really need thousands or millions of dimensions to represent an example?

We believe that examples can accurately be represented using fewer dimensions for learning

This corresponds to the assumption that data points live close to a lower-dimensional manifold (subspace) of the data.

Dimensionality reduction:

- Reduce high dimensional data to fewer dimensions

Benefits

Easier learning

- Fewer parameters to learn
- Collapse multiple useful features into one useful feature
- Remove unnecessary features

Faster learning

Learning without labels

- Reduce dimensionality without labels
- Uncovers latent structure in the data

Linear Projections

Examples given by $\{\mathbf{x}_i\}_{i=1}^N \quad \mathbf{x}_i \in \mathbb{R}^D$

Select a basis of M vectors: $\{\mathbf{u}_m\}_{m=1}^M \quad \mathbf{u}_m \in \mathbb{R}^D$

- Basis vectors are orthonormal (perpendicular and unit)

$$\mathbf{u}_i \cdot \mathbf{u}_j = 0, \quad i \neq j \quad \mathbf{u}_i \cdot \mathbf{u}_j = 1, \quad i = j$$

Center of space $\bar{\mathbf{x}}$ defines the offset

Examples in low dimensional space:

$$\mathbf{z}_i = [\mathbf{z}_i^1, \mathbf{z}_i^2, \dots, \mathbf{z}_i^M] \quad \mathbf{z}_i^m = (\mathbf{x}_i - \bar{\mathbf{x}}) \cdot \mathbf{u}_m \quad \mathbf{z}_i = \mathbf{U} (\mathbf{x}_i - \bar{\mathbf{x}})$$

PCA

Given an $N \times D$ matrix X

- Each row of X is a D -dimensional point

Re-center data by subtracting mean from each row $\hat{\mathbf{x}}_i = \mathbf{x}_i - \bar{\mathbf{x}}$

Compute covariance matrix of data ($D \times D$ matrix) $\Sigma = \frac{1}{N} \hat{\mathbf{X}}^T \hat{\mathbf{X}}$

Find eigenvalues and eigenvectors of Σ

Selected principal components

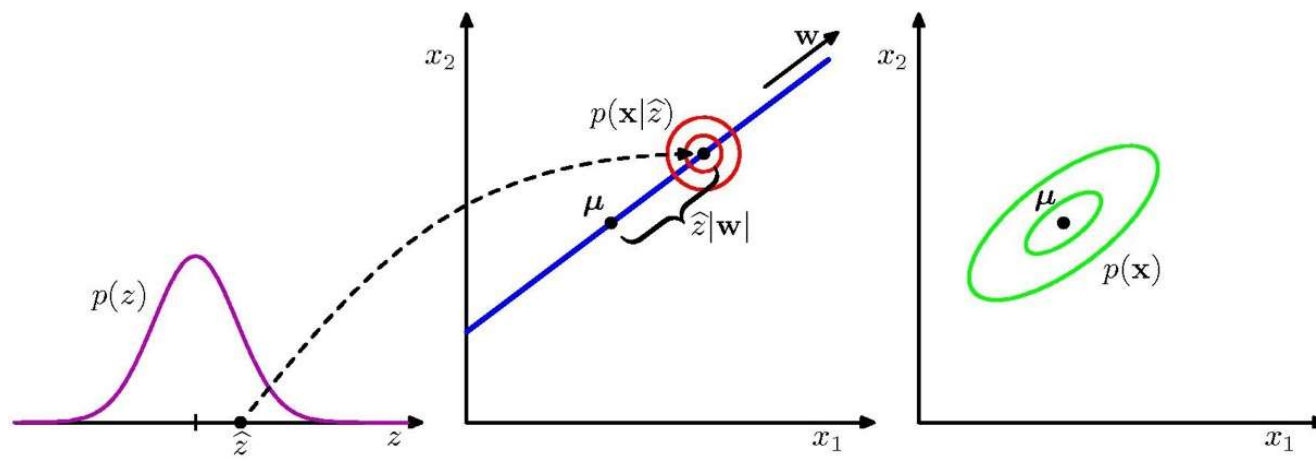
- M eigenvectors with highest eigenvalues

Maximum Variance

PCA selects the dimensions of the data that yield the maximum variance

- The eigenvector with the largest eigenvalue gives the direction of maximum variance
- We want the maximum variance dimensions so as to lose as little as possible in our reconstruction
- Removing small eigenvalues removes dimensions of smaller variances

Generative Story



Generate z from $p(z)$

Generate \mathbf{x} from Gaussian with mean $Wz + \mu$

Probabilistic PCA

Same form as PCA (Principal Component Analysis)

- Finds the M eigenvectors with the highest eigenvalues
- We can easily find this using singular value decomposition
 - Python (numpy), MATLAB, and other packages

Data Reconstruction Error

Distortion of the reconstructed points: $J = \frac{1}{N} \sum_{i=1}^N \|\mathbf{x}_i - \tilde{\mathbf{x}}_i\|^2$

We can show this is equivalent to:

$$J = \frac{1}{N} \sum_{i=1}^N \sum_{j=M+1}^D \|\mathbf{x}_i^T \mathbf{u}_j - \bar{\mathbf{x}}^T \mathbf{u}_j\|^2$$

This is minimized when we select the $D-M$ smallest eigenvectors to use in the distortion

- This leave the M eigenvectors with largest eigenvalues for the principal subspace
- This is the same as PCA

Advantages of PPCA

Why probabilistic PCA?

- Leads to a Bayesian PCA which finds the optimal value for M (number of dimensions)
- Can be used to model class-conditional densities, leading to classification problems
- Generative model that provides high dimensional samples

Summary: PCA

Finds a low dimensional representation of the data using linear projections

Many ways to understand PCA

- Find maximum variance projection
- Find projection with minimum reconstruction error
- Graphical model (probabilistic PCA)
 - EM for high dimensional data

Extensions

- Factor analysis- removes assumptions about shared variance
- Kernel PCA- non-linear version of PCA

Graphical Models

Outline

Representation

- What is a graphical model?
- What does it represent?
- Conditional Independence
- Types of probabilistic models

Inference

- How can we compute probabilities?
- Message Passing

Examples

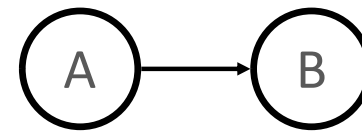
- Learning and inference

Graph Types

Edge type determines graph type

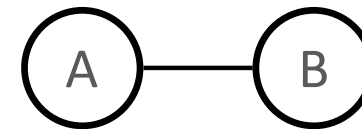
Directed graphs

- Edges have directions ($A \rightarrow B$)
- Assume DAGs (no cycles)
- Typically called Bayesian Networks
 - Popular in AI and stats



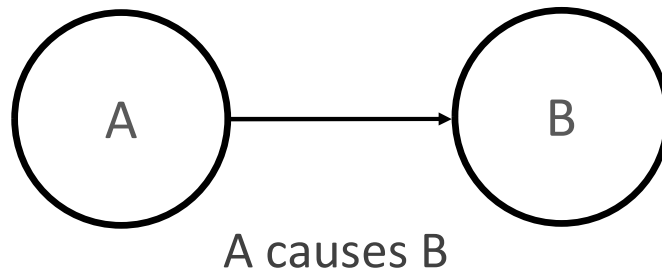
Undirected graphs

- Edges don't have directions ($A - B$)
- Typically called Markov Random Fields (MRFs)
 - Popular in physics and vision



Directed Graphs

The direction of the edge indicates causation



Causation can be very intuitive

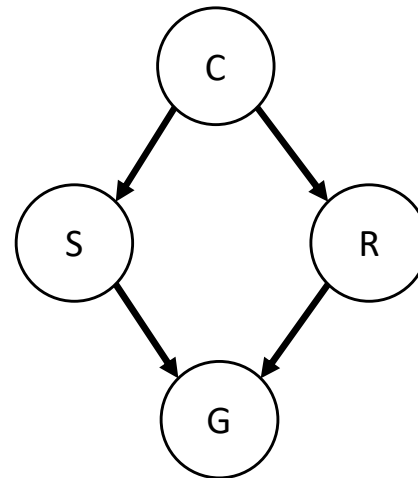
- We may know which random variable causes the other
- Use this intuition to create a graph structure

Factorization

For any graphical model we can write the joint distribution using conditional probabilities

- We just need conditional probabilities for a node given its parents

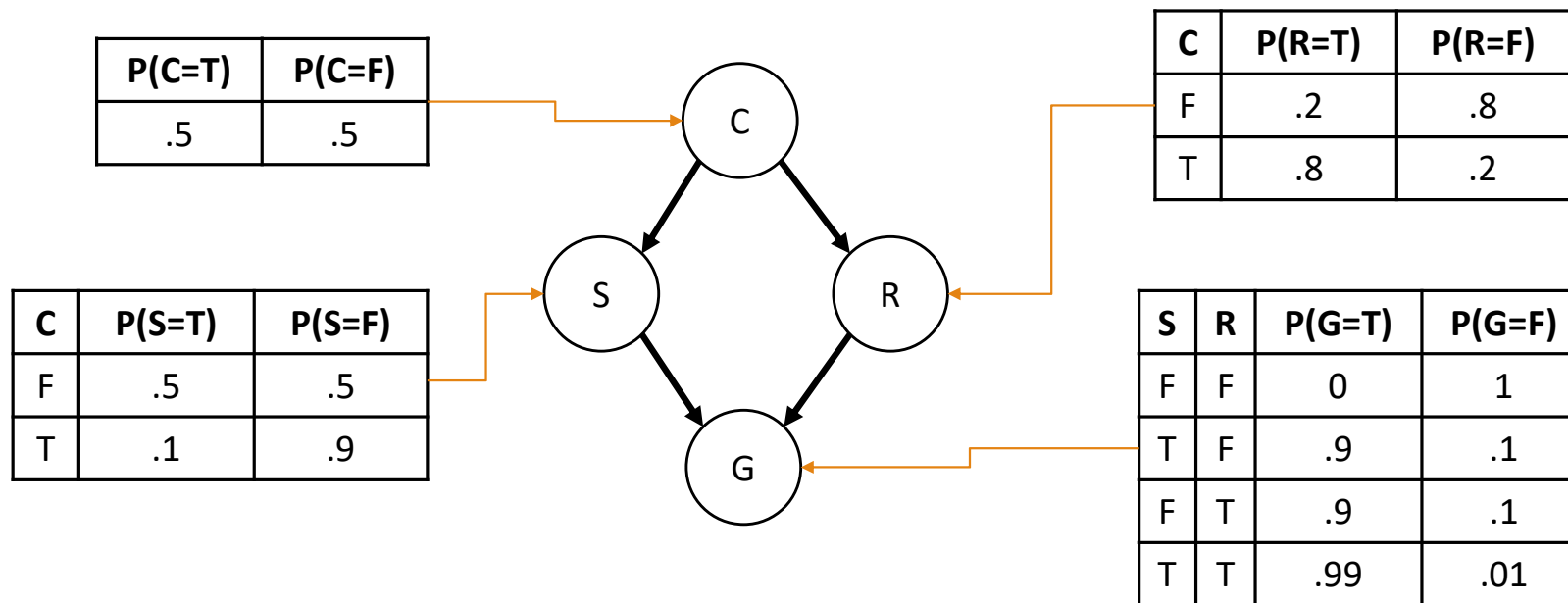
$$p(\mathbf{x}) = \prod_{k=1}^K p(x_k | \text{parents}_k)$$



Conditional Probability Tables

The CPTs specify the conditional probability at each node

- Reflect only local information



Naïve Bayes

Generative story

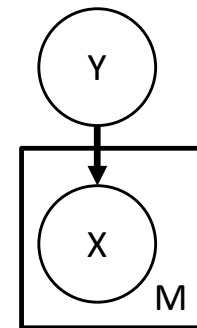
- Generate a label Y
- Given Y , generate each feature X_i independently

Learning

- We observe X and Y , find maximum likelihood solution

Prediction

- Compute most likely value for Y given X



$$\begin{aligned} p(y, \mathbf{x}) &= p(\mathbf{x}|y)p(y) \\ &= \prod_{i=1}^M p(x_i|y)p(y) \end{aligned}$$

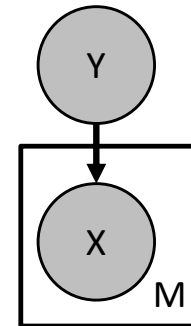
Learning

We assumed both examples, X , and labels, Y , for learning Naïve Bayes

- Maximum likelihood solution
- Each entry in the table is based on counts of the data

What if we only have X ?

- General purpose method for maximizing likelihood where we have missing variables
- $\max_X p(X) = \sum_{y \in Y} p(y, X)$
- Label is a hidden (latent) variable \rightarrow E-M!
- Unsupervised Naïve Bayes = clustering
- Some labels: semi-supervised Naïve Bayes



Conditional Independence

Definition: Random variable (RV) **X** is conditionally independent of RV **Y** given RV **Z** if the probability of each is independent given **Z**

$$p(x, y|z) = p(x|z)p(y|z)$$

Example:

- Probability that I need an umbrella and the ground is wet
- Not independent! If it's wet I probably need an umbrella because it's raining
- I am told it is raining
- Given this, the probability that I need an umbrella is independent of the ground being wet
- I gain no new information knowing that the ground is wet

D-Separation

Two sets of nodes A and B are **d-separated** given observed set C if all paths between A and B are blocked

- 1) The arrows on the path meet head-to-tail or tail-to-tail at a node in set C
- 2) The arrows meet head-to-head at a node and neither the node, nor any of its descendants, is in set C

If sets of nodes are d-separated they are conditionally independent

Same notation applies to sets of nodes: $A \perp\!\!\!\perp B \mid C$

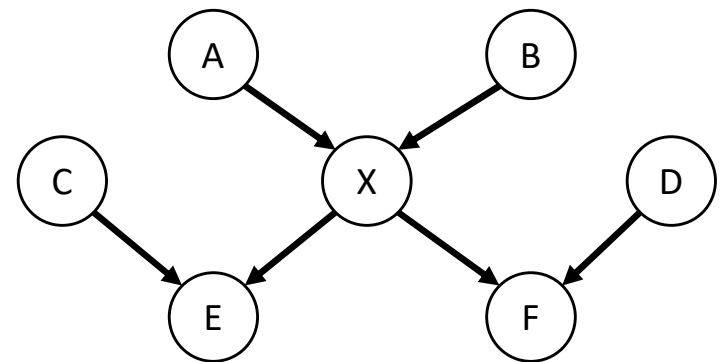
Markov Blanket

The **Markov Blanket** of a node is the minimal set of nodes that isolates it from the graph

- A node conditioned on its Markov blanket is independent from all other nodes in the graph

What are the nodes in the blanket for X?

- Think about d-separation
- All of them!
- A Markov blanket depends on the parents, children, and co-parents



Undirected Networks

What happens when we use undirected edges?



Called undirected graphical models

a.k.a. Markov Random Fields (MRFs), Markov Networks

Conditional Independence

D-separation used directionality of the edges

Can we define it without that?

- Yes!

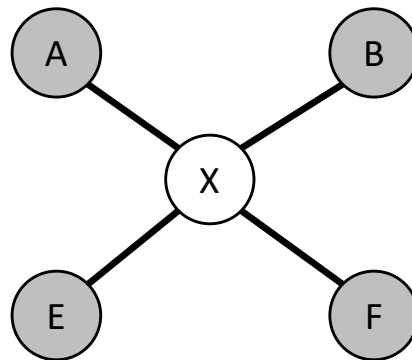
If all paths from set A to set B pass through set C, then A is d-separated from B given C

- Notice no distinction between head-to-head and tail-to-tail
- “Explaining away” not an issue since no causation
- Actually easier to check
- Use the same notation: $A \perp\!\!\!\perp B \mid C$

Markov Blanket

The absence of “explaining away” makes the Markov blanket simple as well

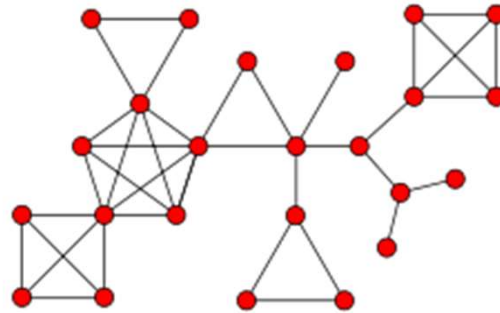
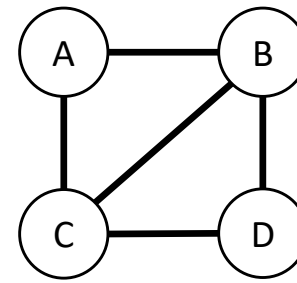
The Markov blanket of a node contains the neighbors of the node



Cliques

Cliques in the graph

- AB, AC, BC, BD, CD
- Maximal cliques: ABC, BCD
- ABCD is not a clique since no edge A—D



We just need to use maximal cliques since they contain all other cliques

Factorization

Define

- x_C is all nodes in clique C
- $\psi(x_C)$ is a potential function over clique C

We can define a joint distribution of the graph as a product of potential functions over maximal cliques

$$p(\mathbf{x}) = \frac{1}{Z} \prod_C \psi(x_C)$$

- Cliques have taken the place of nodes and CPTs

Partition Function

Notice that $\psi(x_c)$ is not a probability

- Must be that $\psi(x_c) \geq 0$
- Will not sum to 1

$$p(\mathbf{x}) = \frac{1}{Z} \prod_c \psi(x_c)$$

We need to normalize to get a probability

$$Z = \sum_{\mathbf{x}} \prod_c \psi(x_c)$$

Z is called the *partition function*

Logistic Regression

No generative story

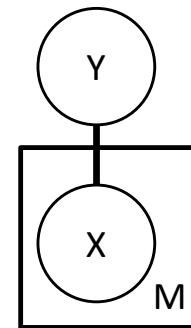
- Graph does not encode causality
- We can say that X and Y are related

Learning

- We observe X and Y, maximum likelihood solution

Prediction

- Compute most likely value for Y given X

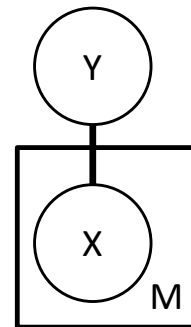


Factorization

$$p(y|\mathbf{x}) = \frac{1}{Z} \prod_m \psi_m(x_m, y)$$

$$\psi_m(x_m, y) = e^{w_m f(x_m, y)}$$

$$Z = \sum_y \prod_m \psi_m(x_m, y)$$



MRF: Pros and Cons

Pros

- Define arbitrary potential functions
- Much more flexibility than directed models
- Easier to compute condition independence
- Don't need to express causation relationship

Cons

- $Z!!!$
- Sum over all states x
- M discrete nodes, each with K discrete states, KM
 - However, we just need Z for learning
 - To evaluate we need most likely option, so Z cancels

Graph Moralization

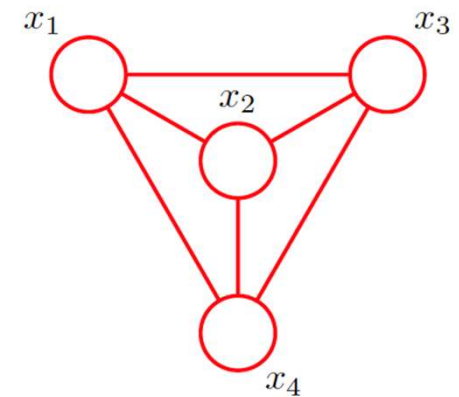
Add links between the parents of a node (if one does not already exist)

“Marrying the parents” \rightarrow *moralization*

Resulting undirected graph is the *moral graph*

Take each conditional distribution and multiply into one of the clique potentials (initialize each at 1)

Partition function still stays at $Z = 1$



Note: We had to discard some conditional independence properties in this process

Inference

Computing probabilities of network configurations

- We may know some values of the network (observed variables)
- How do we compute the posterior of one or more nodes?

Previously, we worked out the probabilities explicitly

Is there an efficient and general way we can do this?

Factor Graphs

Both directed and undirected models allow the global function to be represented as a product of local factors

- This is what makes them powerful

We can make this explicit by representing these factors in the graph

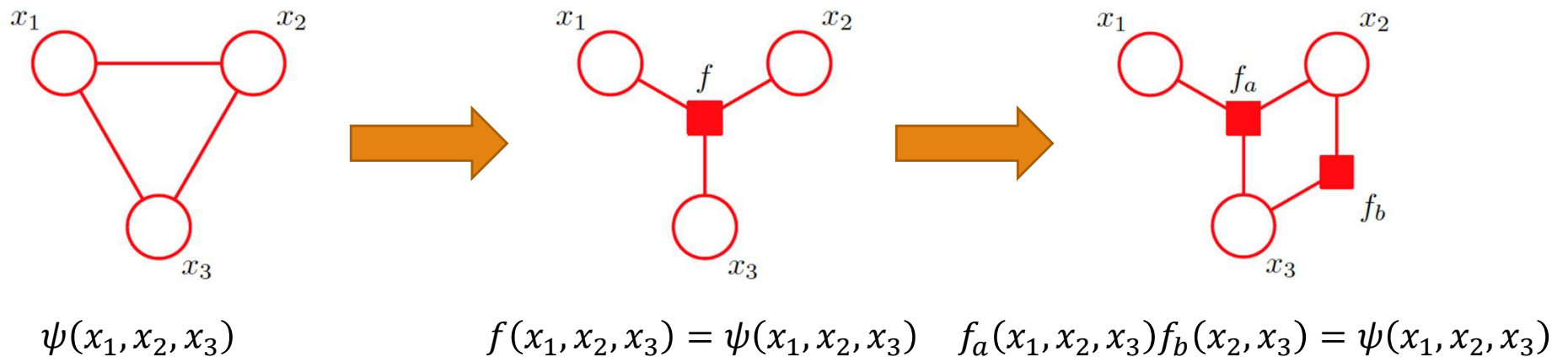
Write the joint distribution as a product of the factors

- \mathbf{x}_s is a subset of the variables
- f_s is a function on the subset \mathbf{x}_s
- $p(\mathbf{x}) = \prod_s f_s(\mathbf{x}_s)$

Factor Graphs

We can construct a factor graph to represent a given graphical model

- For an undirected model, the factors are potential functions over the maximal cliques



Sum-Product Algorithm

Factor graphs are used to derive the sum-product algorithm

- A powerful class of efficient exact inference algorithms for tree graphs
- *Belief propagation*: an algorithm for exact inference on directed graphs without loops, equivalent to special case of sum-product

How do we find the marginal

$$p(x) = \sum_{\mathbf{x} \setminus x} p(\mathbf{x})$$

- $\mathbf{x} \setminus x$ is the set of variables in \mathbf{x} with x removed

We will replace $p(\mathbf{x})$ with a factor graph to obtain an efficient algorithm

Sub-Factors

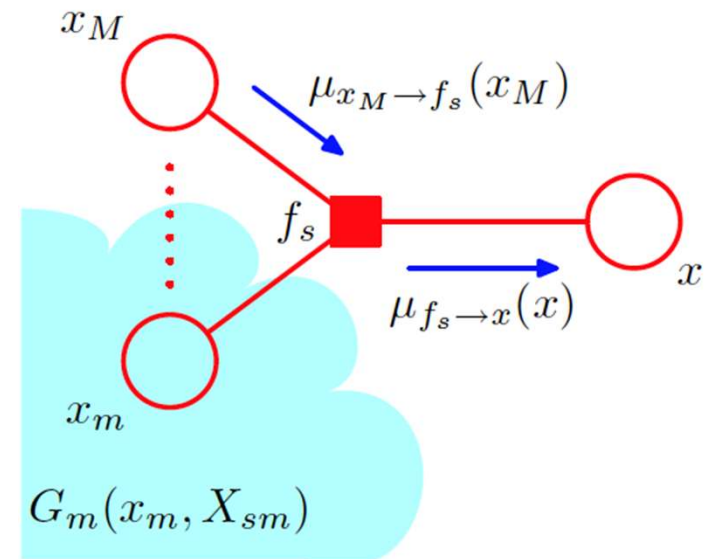
Notice that the factors themselves can be broken into factor sub-trees

- Can itself be factorized:

$$F_s(x, X_s) = f_s(x, x_1, \dots, x_M) G_1(x_1, X_{s1}) \dots G_M(x_M, X_{sM})$$

The set of variables associated with f_s , in addition to x , are $\{x, x_1, \dots, x_M\}$

- This can also be denoted as \mathbf{x}_s



Sub-Factors

Each message is written as:

$$\begin{aligned}\mu_{f_s \rightarrow x}(x) &= \sum_{x_1} \dots \sum_{x_M} f_s(x, x_1, \dots, x_M) \prod_{m \in \text{ne}(f_s) \setminus x} \left[\sum_{X_{xm}} G_m(x_m, X_{sm}) \right] \\ &= \sum_{x_1} \dots \sum_{x_M} f_s(x, x_1, \dots, x_M) \prod_{m \in \text{ne}(f_s) \setminus x} \mu_{x_m \rightarrow f_s}(x_m)\end{aligned}$$

G_m is the product of all factors associated with x_m

Can replace the last term by a message from variable x_m to factor f_s

Messages

Using this definition, we can write messages from variables to factors:

$$\begin{aligned}\mu_{x_m \rightarrow f_s}(x_m) &= \prod_{l \in \text{ne}(x_m) \setminus f_s} \left[\sum_{X_{ml}} F_l(x_m, X_{ml}) \right] \\ &= \prod_{l \in \text{ne}(x_m) \setminus f_s} \mu_{f_l \rightarrow x_m}(x_m)\end{aligned}$$

If only two neighbors, then just pass the message

Requires input from all other factors before the message can be sent

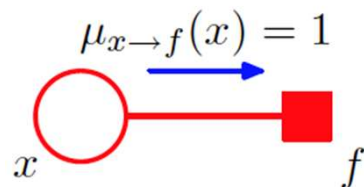
Recursion

These messages are all based on recursion

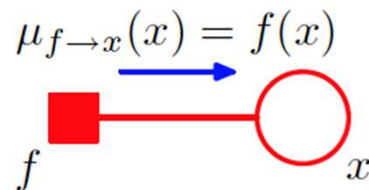
- Derive a message from other input messages

Requires a base case to be used

- The leaves of the network can serve as starting points



$$\mu_{x \rightarrow f}(x) = 1$$



$$\mu_{f \rightarrow x}(x) = f(x)$$

Summary: Sum-Product

We want to evaluate marginal $p(x)$ in a tree factor graph

1. Assume x is root, start messages at the leaves
2. Propagate messages until received by x
 - Wait until all neighbors except one send message, then create new message
3. When x receives all messages, compute $p(x)$

$$p(x) = \prod_{s \in ne(x)} \mu_{f_s \rightarrow x}(x)$$

As before, we can compute the messages over the whole graph and use them to compute every marginal

Normalization

If we started from an undirected graph, then our result is distribution that is not normalized

- If we started from a directed graph, then it is already normalized

Normalization is easy now

- Recall with chains we computed Z by summing over all configurations for the node(s) of interest
- We have the messages to compute this here

Max-Product Algorithm

Computes the configuration of the graph that gives the highest total probability

Replace the sum in sum-product with a max

$$\mathbf{x}^{\max} = \operatorname{argmax}_{\mathbf{x}} p(\mathbf{x})$$

Key trick in sum-product: $ab + ac = a \times (b + c)$

Key trick in max-product: $\max(ab, ac) = a \times \max(b, c)$



True because we are working
with factors and $a \geq 0$

Message Passing

To avoid underflow, we instead consider $\log p(\mathbf{x})$

- $\log(\max_{\mathbf{x}} p(\mathbf{x})) = \max_{\mathbf{x}} \log p(\mathbf{x})$ and $\max(a + b, a + c) = a + \max(b, c)$

Using the message passing formulation from sum-product, we obtain new messages for max-product (now max-sum)

$$\begin{aligned}\mu_{f \rightarrow x}(x) &= \max_{x_1, \dots, x_M} \left[\ln f(x, x_1, \dots, x_M) + \sum_{m \in \text{ne}(f_s) \setminus x} \mu_{x_m \rightarrow f}(x_m) \right] \\ \mu_{x \rightarrow f}(x) &= \sum_{l \in \text{ne}(x) \setminus f} \mu_{f_l \rightarrow x}(x).\end{aligned}$$

Base Cases and Root

Take the log of the previous base cases

$$\mu_{x \rightarrow f}(x) = 0 \quad \mu_{f \rightarrow x}(x) = \log f(x)$$

Root node

$$x^{\max} = \operatorname{argmax}_x \left[\sum_{s \in ne(x)} \mu_{f_s \rightarrow x}(x) \right]$$

Messages from Root

We can't pass messages back as before

- There may be multiple configurations that have max value for $p(\mathbf{x})$
- Need to distinguish between these

Solution: keep track of which states correspond to the max configuration

- Store quantities
- The argument at x_n that gave the highest probability
- To find the best path we follow the values for $\phi(x)$

$$\phi(x_n) = \arg \max_{x_{n-1}} [\ln f_{n-1,n}(x_{n-1}, x_n) + \mu_{x_{n-1} \rightarrow f_{n-1,n}}(x_n)]$$

Inference Review

Message passing algorithms for exact inference

- Each node computes a local message to send to its neighbors
- Sum-Product
 - Compute marginals for nodes in the graph
 - Efficient way to compute all marginals
- Max-Product/Max-Sum
 - Find the highest probability configuration
 - Back track through graph to decode entire configuration

Exact inference algorithms exist for certain types of non-tree structured graphs

Structured Prediction

What is Structured Prediction?

Input: \mathbf{x}

- Typically a structured input
- Maintain structure of input in \mathbf{x}
- Do not flatten into list of features in an instance

Output: y

- y is now from a large set of possible outputs
- Outputs defined based on input
- Often exponential in size of input

Structured Prediction Challenges

Scoring

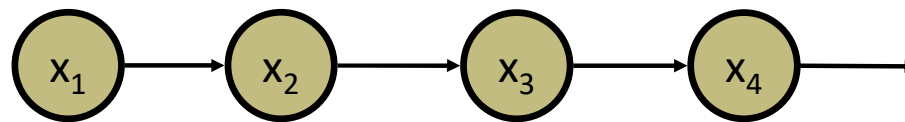
- How do we assign a score/probability to a possible output structure?

Search/Inference

- Find the best scoring output structure
- How do we search through an exponential number of options?

Markov Chains

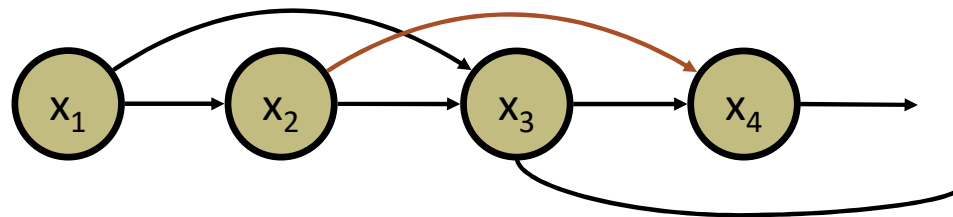
First order Markov chain:



$$p(\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N) = \prod_{n=1}^N p(\mathbf{x}_n | \mathbf{x}_{n-1})$$

$K(K - 1)$ parameters

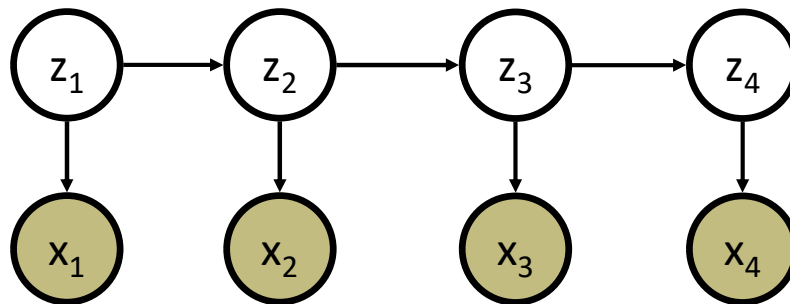
Second order Markov chain:



$$p(\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N) = \prod_{n=1}^N p(\mathbf{x}_n | \mathbf{x}_{n-1}, \mathbf{x}_{n-2})$$

$K^2(K - 1)$ parameters

Hidden Markov Model (HMM)



HMMs: Pictorial Representation

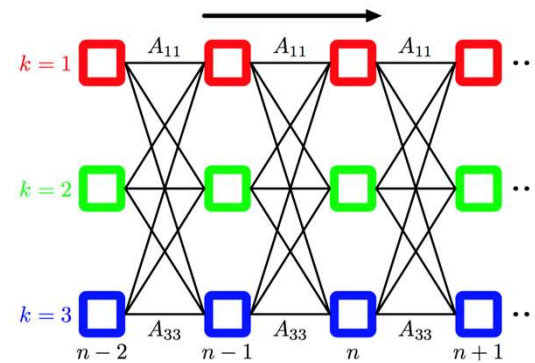
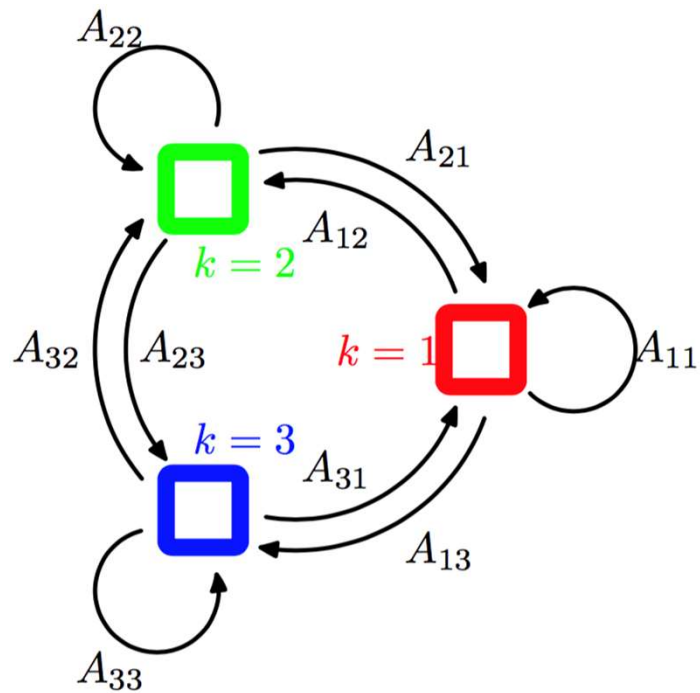


Figure credit: Bishop

Joint Probability of HMM

The joint probability of an HMM is:

$$p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta}) = p(\mathbf{z}_1|\boldsymbol{\pi}) \left[\prod_{n=2}^N p(\mathbf{z}_n|\mathbf{z}_{n-1}, \mathbf{A}) \right] \prod_{m=1}^N p(\mathbf{x}_m|\mathbf{z}_m, \boldsymbol{\phi})$$

A: Matrix of transition probabilities

- A_{ij} is the probability of moving from state i to state j

π : Vector with starting probabilities

ϕ : Matrix of emission probabilities

- ϕ_{ij} is the probability of state i and emitting observation j


Unsupervised Training

How do we train a probabilistic model?

- Maximum likelihood! $\max_{\theta} p(\mathbf{X}, \mathbf{Z}|\theta)$
- Problem: we don't know \mathbf{Z}

Solution: EM

- Step 1: Write the complete data likelihood

$$p(\mathbf{X}|\theta) = \sum_{\mathbf{Z}} p(\mathbf{X}, \mathbf{Z}|\theta)$$
$$p(\mathbf{X}, \mathbf{Z}|\theta) = p(\mathbf{z}_1|\boldsymbol{\pi}) \left[\prod_{n=2}^N p(\mathbf{z}_n|\mathbf{z}_{n-1}, \mathbf{A}) \right] \prod_{m=1}^N p(\mathbf{x}_m|\mathbf{z}_m, \boldsymbol{\phi})$$


EM for HMMs

E-Step

- Find the expected values for the hidden variables Z given the model parameters
 - The most likely Z given X and current model parameters

M-Step

- Pretend to observe the values for Z
- Update model parameters A, π, φ to maximize complete data likelihood

Prediction

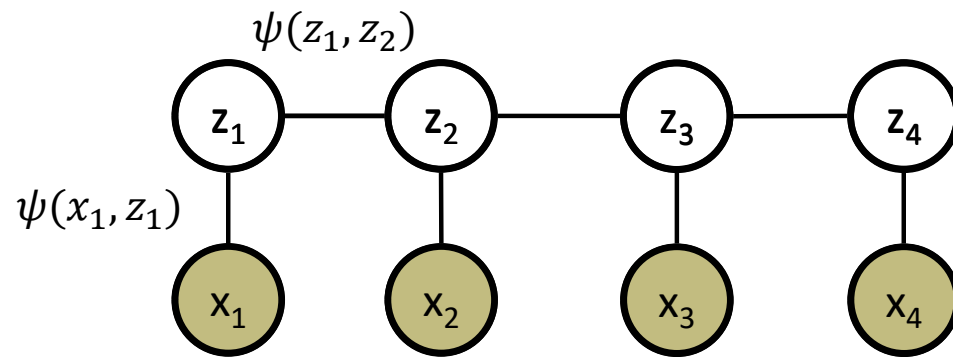
How do we find the most likely state?

$$\operatorname{argmax}_{\mathbf{Z}} p(\mathbf{Z}|\mathbf{X}, \theta)$$

Inference

- Max Product Algorithm
- Viterbi Decoding

Conditional Random Fields



$$p(\mathbf{X}, \mathbf{Z}) = \frac{1}{Z} \prod_A \psi_A(\mathbf{x}_A)$$

Replace conditional probabilities with potential functions

The joint probability is a product of potential functions

Joint Probability

The joint probability of the HMM can be written using potential functions:

- HMM:

$$p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta}) = p(\mathbf{z}_1|\boldsymbol{\pi}) \left[\prod_{n=2}^N p(\mathbf{z}_n|\mathbf{z}_{n-1}, \mathbf{A}) \right] \prod_{m=1}^N p(\mathbf{x}_m|\mathbf{z}_m, \boldsymbol{\phi})$$

- CRF:

$$p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta}) = \frac{1}{Z} \psi(\mathbf{z}_1) \left[\prod_{n=2}^N \psi(\mathbf{z}_n, \mathbf{z}_{n-1}) \right] \prod_{m=1}^N \psi(\mathbf{x}_m, \mathbf{z}_m)$$

$$p(\mathbf{X}, \mathbf{Z}) = \frac{1}{Z} \prod_A \psi_A(\mathbf{x}_A) \qquad Z = \sum_{\mathbf{x}, \mathbf{z}} \prod_A \psi_A(\mathbf{x}_A, \mathbf{z}_A)$$

Model Parameters

HMM: model parameters are CPTs

- CRF: CPTs replaced with potential functions

Parameterize the potential functions

- Learn the parameters

$$\psi(\mathbf{x}, \mathbf{z}) = \exp \left(\sum_k \theta_k f_k(\mathbf{x}, \mathbf{z}) \right)$$

- Parameters θ determine value of potential function
- f_k is a feature function
 - Genomics example: $f_k = 1$ if x is the base pair “G”
- Notice: linear combination of features (linear model!)

Discriminative Training

Solution: don't learn $p(\mathbf{X})$!

Maximize the conditional likelihood of the data

$$p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}) = \frac{1}{Z} \prod_A \psi_A(\mathbf{X}_A, \mathbf{Z}_A) \quad Z = \sum_{\mathbf{Z}} \prod_A \psi_A(\mathbf{X}_A, \mathbf{Z}_A)$$

$$\psi_A(\mathbf{X}_A, \mathbf{Z}_A) = \exp \left\{ \sum_k \theta_{Ak} f_{Ak}(\mathbf{X}_A, \mathbf{Z}_A) \right\}$$

CRF Conditional log likelihood of all examples for sequence model:

$$\log p(\mathbf{z}|\mathbf{x}) = \sum_{i=1}^N \sum_{t=1}^T \sum_{k=1}^K \theta_k f_k(\mathbf{z}_{it}, \mathbf{z}_{it-1}, \mathbf{x}_{it}) - \sum_{i=1}^N \log Z(\mathbf{x}_i)$$

Training a CRF

The conditional log likelihood is convex

- Take the derivative and solve for θ

$$\frac{\partial L}{\partial \theta_k} = \sum_{i=1}^N \sum_{t=1}^T f_k(\mathbf{z}_{it}, \mathbf{z}_{it-1}, \mathbf{x}_{it}) - \sum_{i=1}^N \sum_{t=1}^T \sum_{\mathbf{z}, \mathbf{z}'} f_k(\mathbf{z}, \mathbf{z}', \mathbf{x}_{it}) p(\mathbf{z}, \mathbf{z}' | \mathbf{x}_{it}) - \frac{\theta_k}{\sigma^2}$$

- The derivative is 0 when
 - The last term (regularizer) is 0
 - The first term and the second term cancel each other
 - First term: the expected value for f_k under the empirical distribution (from the data)
 - Second term: expectation for f_k given model distribution

Computing Probabilities

What do we need to compute the values in the derivative?

- Marginal probability of $p(\mathbf{z}, \mathbf{z}' | \mathbf{x}_{it})$
- The normalization constant Z
 - Total score for all possible labeling of the sequence
 - Sum Product Algorithm (Forward-Backward)

Prediction

- Sequence of states with max probability
- Prediction
 - How do we find the highest probability sequence?
 - Max Product Algorithm (Viterbi decoding)

CRF Summary

CRFs are

- Markov Random Fields
- The MRF equivalent of a supervised HMM
- Discriminatively trained using conditional log likelihood
- Linear models (recall linear potential functions)

CRF training contains versions of

- Sum Product Algorithm
- Max Product Algorithm
- Convex optimization

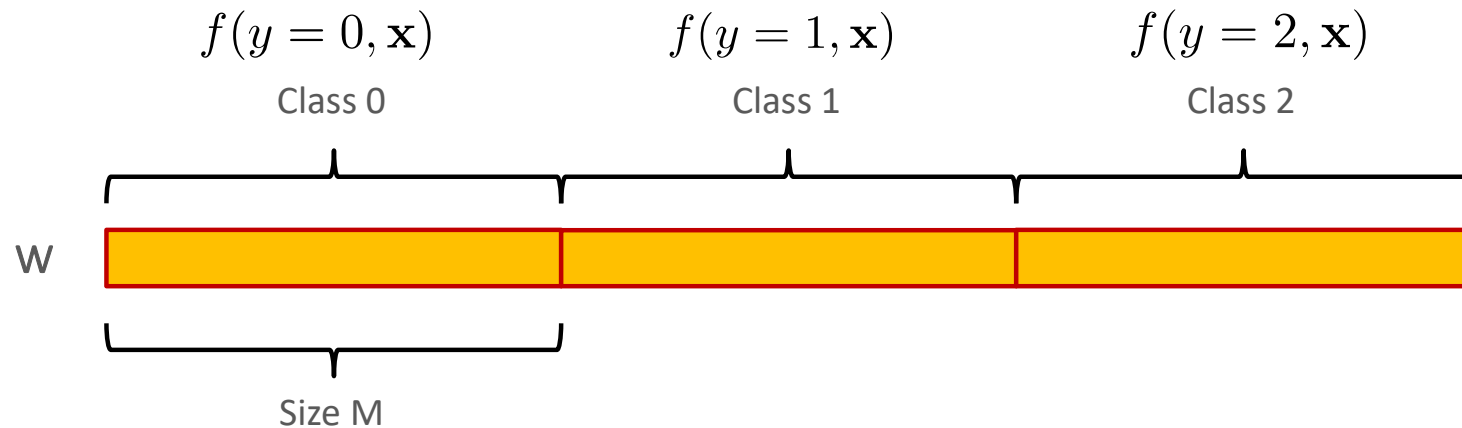
Multi-class Encodings

- Encode multiple labels into a single weight vector
- Update the entire vector at once in a single model

1 of K Encoding

Encode all K labels in a single linear classifier

- Lengthen the weight vector to $M \cdot K$
- A segment of length M learns weights for class k

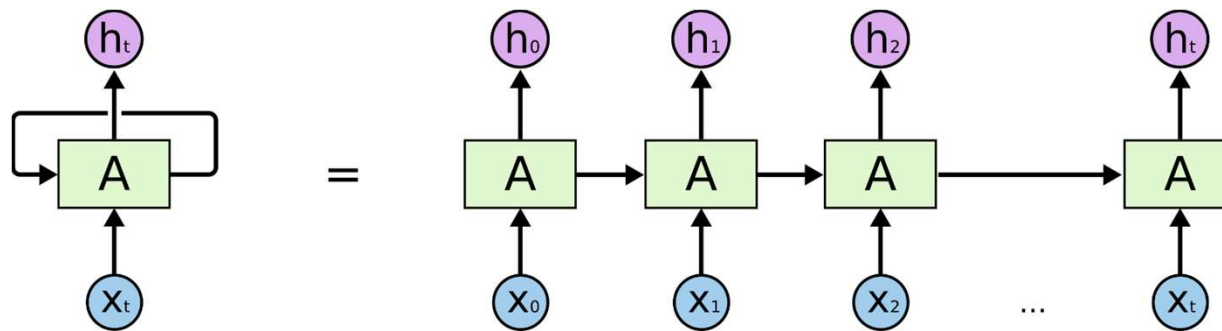


Algorithm: Structured Perceptron

On each round:

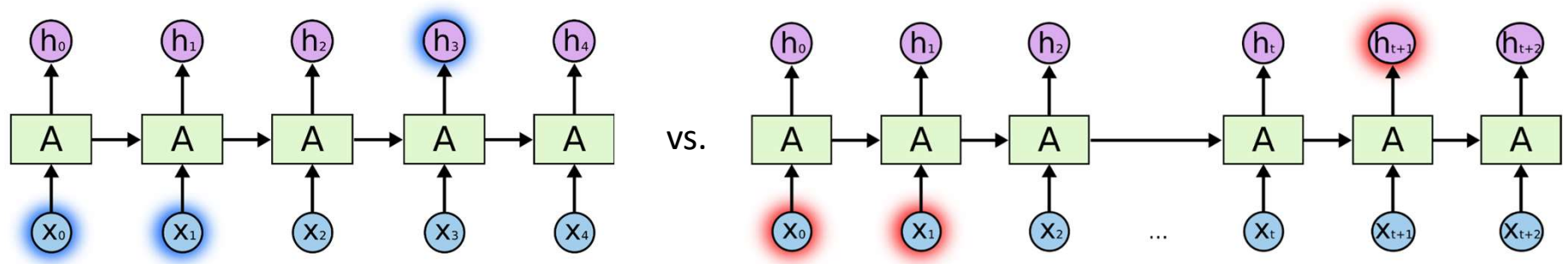
1. Receive example \mathbf{x}
2. Predict: $\hat{y} = \operatorname{argmax}_{y \in L} \mathbf{w} \cdot f(\mathbf{x}, y)$ Need efficient procedure since L may be exponential!
3. Receive correct label: $y \in L$
4. Suffer loss: $l(y, \hat{y})$ Loss sensitive to comparing structured objects
5. Update \mathbf{w} : $\mathbf{w}^{i+1} = \mathbf{w}^i + f(y_i, \mathbf{x}_i) - f(\hat{y}, \mathbf{x}_i)$ General representations for each prediction

Recurrent Neural Networks (RNNs)



- Recurrent neural networks repeat the same processing for each input, preserving some “state” information between iterations.
- This allows them to store information and capture dependencies between multiple inputs.

The Problem of Long-Term Dependencies



- It can be difficult to train an RNN to use information along different time scales, and, in particular, over long time scales.
- LSTMs are a flavor of RNN that address this concern well, and as a result are widely used.

Long Short Term Memory architecture

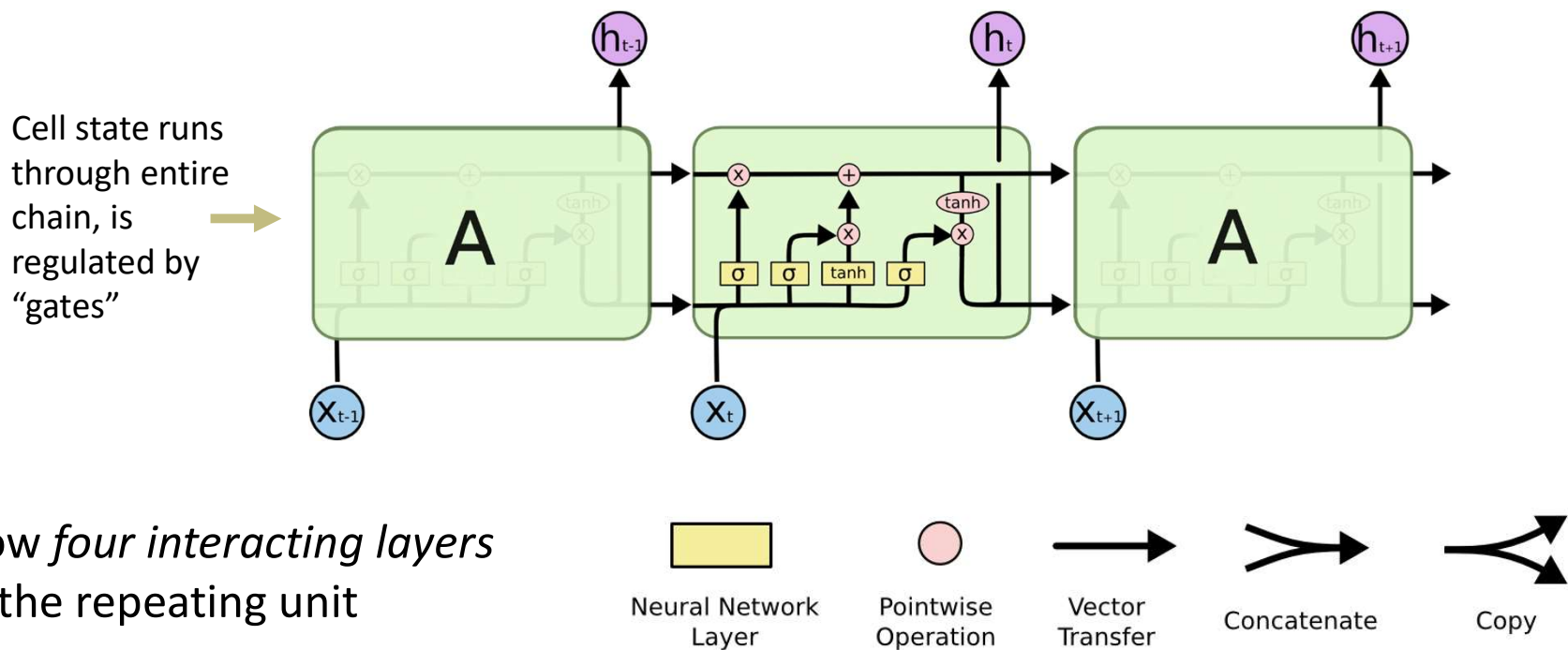
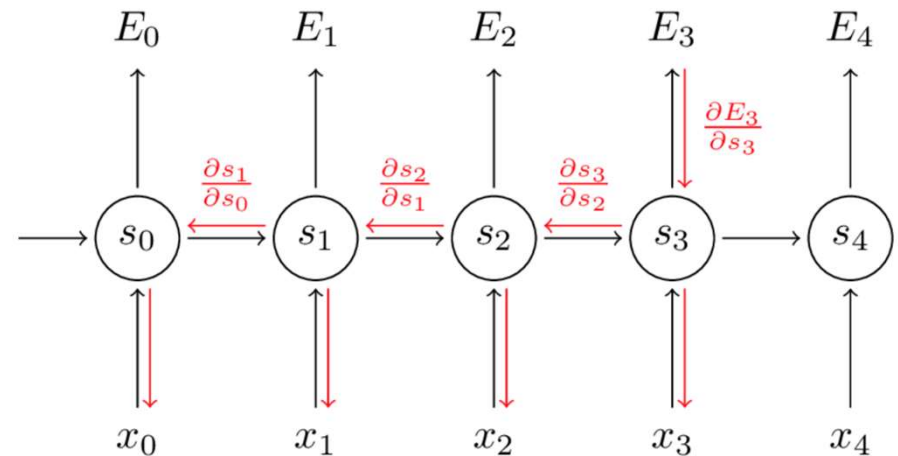


Image credit: <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>


Training RNNs

- Use Backpropagation Through Time (BPTT)
- Unroll network as shown
 - Compute derivatives along paths between current output and each prior input
- Loss function will likely be non-convex in parameter space
 - Use stochastic gradient descent



Practical machine learning

7 Easy Steps


1. Understand the problem
 2. Should you use machine learning?
 3. Designing a ML system
 4. Implementing a ML system
 5. Why doesn't it work?
 6. How to make it better
 7. Deploying the system
- 

Formulate the Problem

Machine learning is formal; it requires formal problems.

- Clear goal
- Consistent input (data)
- Consistent output (predictions)

People often come up with vague problems

- Are they suggesting one task? Several related tasks? Is the task learnable?
 - Your job: formalize the problem
- 

The Machine Learning Checklist

A well formed problem

Data suitable for learning

The right resources

- Do you have the resources for development and deployment?

Will it scale?

Maintenance plan

Does it make sense to invest in this solution?

Baselines

Good baselines are the key to evaluation

- They help you understand the data
- Questions:
 - Will a rule-based system do better?
 - How will you know when ML works?
 - Will the improvement over baseline matter?
 - How well can humans do?

Answers to the above needed to convince your boss to invest in ML

A solid orange horizontal bar spanning the width of the slide, located at the bottom.

Feature Design

You are only as good as your representation

Finding the right features is often the key to success

- Understanding how algorithms work influences feature design
 - For example, choosing a linear vs. a non-linear model

Train/Dev/Test

Take test data seriously

- Set it aside and don't use it until you are done
- Easy to hill climb on a data set

How much data to use?

- Save enough test data so tests are meaningful
- Does a 1% improvement mean 1 example, 10, 100?

Use cross validation on training data or tune on dev data

A solid orange horizontal bar spanning the width of the slide, located at the bottom.

No bugs, but it doesn't work

What should I do?

- More/less features?
- Regularization?
- More data?
- New algorithm?
- New optimization?
- Different hyper-parameters?
- ...

Diagnostics

- Problem 1: Bias vs. Variance
- Problem 2: Optimization Algorithm

Testing Components

Test each component separately

Approach #1: Gold labels

- Simulate better input by using hand labeled data
 - For example, how do perfect web page topic predictions improve ranking?
- How much did it help?
 - You can artificially add error to see how worse performance impacts the overall system

Testing Components

Approach #2: Compare to baseline

- Add a new feature. How much did it help?
- Ablative feature analysis: remove feature and measure impact

Key idea: Don't spend time improving the system unless you know the improvement will help

- You rarely need 100% accuracy for informative output