CS440/ECE448 Lecture 28: Review I
Final Exam  Mon, May 6, 9:30–10:45

Covers all lectures after the first exam.
Same format as the first exam.
Location: TBA

Conflict exam: Wed, May 8, 9:30–10:45
Location: Siebel 3403.

If you need to take your exam at DRES, make sure to notify DRES in advance
DID THE SUN JUST EXPLODE?
(IT'S NIGHT, SO WE'RE NOT SURE.)

THIS NEUTRINO DETECTOR MEASURES WHETHER THE SUN HAS GONE NOVA.
THEN, IT ROLLS TWO DICE. IF THEY BOTH COME UP SIX, IT TELLS US.
OTHERWISE, IT TELLS THE TRUTH.

LET'S TRY.
DETECTOR! HAS THE SUN GONE NOVA?

ROLL:

YES.

FREQUENTIST STATISTICIAN:
THE PROBABILITY OF THIS RESULT HAPPENING BY CHANCE IS 1/36 = 0.027.
SINCE p < 0.05, I CONCLUDE THAT THE SUN HAS EXPLODED.

BAYESIAN STATISTICIAN:
BET YOU $50 IT HASN'T.
Bayes’ Rule

• The product rule gives us two ways to factor a joint probability:
\[ P(A, B) = P(B|A)P(A) = P(A|B)P(B) \]

• Therefore,
\[ P(A|B) = \frac{P(B|A)P(A)}{P(B)} \]

• Why is this useful?
  • “A” is something we care about, but P(A|B) is really really hard to measure (example: the sun exploded)
  • “B” is something less interesting, but P(B|A) is easy to measure (example: the amount of light falling on a solar cell)
  • Bayes’ rule tells us how to compute the probability we want (P(A|B)) from probabilities that are much, much easier to measure (P(B|A)).
The More Useful Version of Bayes’ Rule

\[ P(A|B) = \frac{P(B|A)P(A)}{P(B)} \]

- **Remember**, \( P(B|A) \) is easy to measure
  (the probability that light hits our solar cell, if the sun still exists and it’s daytime).
- **Let’s assume we also know** \( P(A) \) (the probability the sun still exists).
- **But suppose we don’t really know** \( P(B) \) (what is the probability light hits our solar cell, if we don’t really know whether the sun still exists or not?)
- **However, we can compute** \( P(B) = P(B|A)P(A) + P(B|\neg A)P(\neg A) \)

\[ P(A|B) = \frac{P(B|A)P(A)}{P(B|A)P(A) + P(B|\neg A)P(\neg A)} \]

This version is what you memorize.

This version is what you actually use.
The Bayesian Decision: Loss Function

• The **query variable**, \( Y \), is a random variable.
  • Assume its pmf, \( P(Y=y) \) is known.
  • Furthermore, the true value of \( Y \) has already been determined --- we just don’t know what it is!

• The agent must **act** by saying “I believe that \( Y=a \)”.

• The agent has a **post-hoc loss function** \( L(y, a) \)
  • \( L(y, a) \) is the incurred loss if the true value is \( Y=y \), but the agent says “\( a \)”.

• The **a priori loss function** \( L(Y, a) \) is a **binary random variable**
  • \( P(L(Y, a) = 0) = P(Y = a) \)
  • \( P(L(Y, a) = 1) = P(Y \neq a) \)
The Bayesian Decision

• The observation, \( E \), is another random variable.

• Suppose the joint probability \( P(Y = y, E = e) \) is known.

• The agent is allowed to observe the true value of \( E = e \) before it guesses the value of \( Y \).
  • Suppose that the observed value of \( E \) is \( E = e \).
    Suppose the agent guesses that \( Y = a \).

• Then its loss, \( L(Y, a) \), is a conditional random variable:
  \[
P(L(Y, a) = 0 | E = e) = P(Y = a | E = e) \\
P(L(Y, a) = 1 | E = e) = P(Y \neq a | E = e) \\
= \sum_{y \neq a} P(Y = y | E = e)
\]
MAP decision

The action, “a”, should be the value of C that has the highest posterior probability given the observation X=x:

\[ a^* = \arg\max_a P(Y = a | E = e) = \arg\max_a \frac{P(E = e | Y = a) P(Y = a)}{P(E = e)} = \arg\max_a P(E = e | Y = a) P(Y = a) \]

**Maximum A Posterior (MAP) decision:**

\[ a_{MAP}^* = \arg\max_a P(Y = a | E = e) = \arg\max_a P(E = e | Y = a) P(Y = a) \]

**Maximum Likelihood (ML) decision:**

\[ a_{ML}^* = \arg\max_a P(E = e | Y = a) \]
The Bayesian Terms

• \( P(Y = y) \) is called the “prior” (a priori, in Latin) because it represents your belief about the query variable \textit{before} you see any observation.

• \( P(Y = y|E = e) \) is called the “posterior” (a posteriori, in Latin), because it represents your belief about the query variable \textit{after} you see the observation.

• \( P(E = e|Y = y) \) is called the “likelihood” because it tells you how much the observation, \( E = e \), is like the observations you expect if \( Y = y \).

• \( P(E = e) \) is called the “evidence distribution” because \( E \) is the evidence variable, and \( P(E = e) \) is its marginal distribution.

\[ P(y|e) = \frac{P(e|y)P(y)}{P(e)} \]
Naïve Bayes model

Suppose we have many different types of observations (symptoms, features) $E_1, \ldots, E_n$ that we want to use to obtain evidence about an underlying hypothesis $Y$

MAP decision:

$$a = \arg\max_p(Y = a | E_1 = e_1, \ldots, E_n = e_n)$$

$$= \arg\max_p(Y = a)p(E_1 = e_1, \ldots, E_n = e_n | Y = a)$$

$$\approx \arg\max_p(Y = a)p(y_1 | a)p(y_2 | a) \cdots p(y_n | a)$$
Parameter estimation

- Model parameters: feature likelihoods $p(\text{word} \mid \text{class})$ and priors $p(\text{class})$
  - How do we obtain the values of these parameters?

<table>
<thead>
<tr>
<th>prior</th>
<th>P(\text{word} \mid \text{spam})</th>
<th>P(\text{word} \mid \neg\text{spam})</th>
</tr>
</thead>
<tbody>
<tr>
<td>spam: 0.33</td>
<td>the: 0.0156</td>
<td>the: 0.0210</td>
</tr>
<tr>
<td></td>
<td>to: 0.0153</td>
<td>to: 0.0133</td>
</tr>
<tr>
<td></td>
<td>and: 0.0115</td>
<td>of: 0.0119</td>
</tr>
<tr>
<td>¬spam: 0.67</td>
<td>of: 0.0095</td>
<td>2002: 0.0110</td>
</tr>
<tr>
<td></td>
<td>you: 0.0093</td>
<td>with: 0.0108</td>
</tr>
<tr>
<td></td>
<td>a: 0.0086</td>
<td>from: 0.0107</td>
</tr>
<tr>
<td></td>
<td>with: 0.0080</td>
<td>and: 0.0105</td>
</tr>
<tr>
<td></td>
<td>from: 0.0075</td>
<td>a: 0.0100</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Bayesian Learning

- The “bag of words model” has the following parameters:
  - $\lambda_{cw} \equiv P(W = w | C = c)$
  - $\pi_c \equiv P(C = c)$
- Each document is a sequence of words, $D_i = [W_{1i}, \ldots, W_{ni}]$.
- If we assume that each word is conditionally independent given the class (the naïve Bayes a.k.a. bag-of-words assumption), then we get:

$$P(E, Y) = \prod_{i=1}^{m} P(D_i | C_i) P(C_i)$$

$$= \prod_{i=1}^{m} P(C_i = c_i) \prod_{j=1}^{n} P(W_{ji} = w_{ji} | C_i = c_i) = \prod_{i=1}^{m} \pi_{c_i} \prod_{j=1}^{n} \lambda_{c_i w_{ji}}$$
Parameter estimation

• ML (Maximum Likelihood) parameter estimate:

\[
P(\text{word} \mid \text{class}) = \frac{\text{# of occurrences of this word in docs from this class}}{\text{total # of words in docs from this class}}
\]

• Laplacian Smoothing estimate

  • How can you estimate the probability of a word you never saw in the training set? (Hint: what happens if you give it probability 0, then it actually occurs in a test document?)

  • Laplacian smoothing: pretend you have seen every vocabulary word one more time than you actually did

\[
P(\text{word} \mid \text{class}) = \frac{\text{# of occurrences of this word in docs from this class} + 1}{\text{total # of words in docs from this class} + V}
\]

(V: total number of unique words)
Garlic halved horizontally = nature's Voronoi diagram?
Learning \( P(C = c) \)

- This is the probability that a randomly chosen document from our data has class label \( c \).
- \( P( C ) \) is a categorical random variable over \( k \) outcomes \( c_1 \ldots c_k \).
- How do we set the parameters of this distribution?
- Given our training data of labeled documents, we can simply set \( P(C = c_i) \) to the fraction of documents that have class label \( c_i \).
- This is a **maximum likelihood estimate**: Among all categorical distributions over \( k \) outcomes, this assigns the highest probability (likelihood) to the training data.
Documents as random variable

• We assume a **fixed vocabulary** $V$ of $M$ word types: $V = \{\text{apple, ...}, \text{zebra}\}$.

• A **document** $d_i = \text{“The lazy fox...”}$ is a **sequence of $n$ word tokens**
  
  $d_i = w_{i1}...w_{iN}$

  The same word type may appear multiple times in $d_i$.

• Choice 1: We model $d_i$ as a **set of word types**:
  
  $\forall v_j \in V$: what’s the probability that $v_j$ occurs/doesn’t occur in $d_i$?
  
  We treat $P(v_j)$ as a Bernoulli random variable

• Choice 2: We model $d_i$ as a **sequence of word tokens**:
  
  $\forall n_{n=1...N}$: what’s the probability that $w_{in} = v_j$ (rather than any other $v_{j'}$)
  
  We treat $P(w_{in})$ as a categorical random variable (over $V$)
Linear Classifiers in General

Consider the classifier

\[ y = 1 \quad \text{if} \quad \beta_c + \sum_{w=1}^{V} \alpha_{cw}f_{cw} > 0 \]

\[ y = 0 \quad \text{if} \quad \beta_c + \sum_{w=1}^{V} \alpha_{cw}f_{cw} < 0 \]

This is called a “linear classifier” because the boundary between the two classes is a line. Here is an example of such a classifier, with its boundary plotted as a line in the two-dimensional space \( f_1 \) by \( f_2 \):
Linear Classifiers in General

Consider the classifier

\[ y = \arg \max_c \left( \beta_c + \sum_{w=1}^{V} \alpha_{cw} f_{cw} \right) \]

- This is called a “multi-class linear classifier.”
- The regions \( y = 0, y = 1, y = 2 \) etc. are called “Voronoi regions.”
- They are regions with piece-wise linear boundaries. Here is an example from Wikipedia of Voronoi regions plotted in the two-dimensional space \( f_1 \) by \( f_2 \):
Linear Classifiers in General

When the features are binary ($f_w \in \{0,1\}$), many (but not all!) binary functions can be re-written as linear functions. For example, the function

$$y = (f_1 \lor f_2)$$

can be re-written as

$$y = 1 \text{ iff } f_1 + f_2 - 0.5 > 0$$

Similarly, the function

$$y = (f_1 \land f_2)$$

can be re-written as

$$y = 1 \text{ iff } f_1 + f_2 - 1.5 > 0$$
Perceptron model:
action potential = signum(affine function of the features)

\[ y = \text{sgn}(\alpha_1 f_1 + \alpha_2 f_2 + \ldots + \alpha_V f_V + \beta) = \text{sgn}(\vec{w}^T \vec{f}) \]

Where \( \vec{w} = [\alpha_1, \ldots, \alpha_V, \beta]^T \)
and \( \vec{f} = [f_1, \ldots, f_V, 1]^T \)

Can incorporate bias as component of the weight vector by always including a feature with value set to 1
Perceptron

For each training instance $\mathbf{f}$ with label $y \in \{-1,1\}$:

- Classify with current weights: $y' = \text{sgn}(\mathbf{w}^T \mathbf{f})$
  - Notice $y' \in \{-1,1\}$ too.
- Update weights:
  - if $y = y'$ then do nothing
  - if $y \neq y'$ then $\mathbf{w} = \mathbf{w} + \eta y \mathbf{f}$
    - $\eta$ (eta) is a “learning rate.” More about that later.
Perceptron: Proof of Convergence

• If the data are linearly separable (if there exists a $\vec{w}$ vector such that the true label is given by $y' = \text{sgn}(\vec{w}^T \vec{f})$), then the perceptron algorithm is guaranteed to converge, even with a constant learning rate, even $\eta=1$.

• In fact, training a perceptron is often the fastest way to find out if the data are linearly separable. If $\vec{w}$ converges, then the data are separable; if $\vec{w}$ diverges toward infinity, then no.

• If the data are not linearly separable, then perceptron converges iff the learning rate decreases, e.g., $\eta=1/n$ for the $n$’th training sample.
Lecture 17: More on binary vs. multi-class classifiers  
(Polychotomizers: One-Hot Vectors, Softmax, and Cross-Entropy)

Mark Hasegawa-Johnson, 3/9/2019. CC-BY 3.0: You are free to share and adapt these slides if you cite the original.  
Modified by Julia Hockenmaier
The supervised learning task

Given a **labeled training data set** of $N$ items $x_n \in X$ with labels $y_n \in Y$

$$D_{\text{train}} = \{(x_1, y_1), ..., (x_N, y_N)\}$$

($y_n$ is determined by some unknown target function $f(x)$)

Return a model $g: X \mapsto Y$ that is a good approximation of $f(x)$

($g$ should assign correct labels $y$ to unseen $x \notin D_{\text{train}}$)
Classifiers in vector spaces

Binary classification:
We assume $f$ separates the positive and negative examples:

Assign $y = 1$ to all $x$ where $f(x) > 0$
Assign $y = 0$ (or -1) to all $x$ where $f(x) < 0$
Linear classifiers

Many learning algorithms restrict the hypothesis space to **linear classifiers**:

\[ f(x) = w_0 + wx \]
Linear Separability

• Not all data sets are linearly separable:

• Sometimes, feature transformations help:
Linear classifiers: \( f(x) = w_0 + wx \)

**Linear classifiers** are defined over vector spaces.

Every hypothesis \( f(x) \) is a hyperplane:

\[
f(x) = w_0 + wx
\]

\( f(x) \) is also called the decision boundary.

- Assign \( \hat{y} = +1 \) to all \( x \) where \( f(x) > 0 \)
- Assign \( \hat{y} = -1 \) to all \( x \) where \( f(x) < 0 \)

\[
\hat{y} = \text{sgn}(f(x))
\]
With a separate bias term \( w_0 \):

\[
    f(x) = w \cdot x + w_0
\]

The **instance space** \( \mathcal{X} \) is a \( d \)-dimensional vector space (each \( x \in \mathcal{X} \) has \( d \) elements).

The **decision boundary** \( f(x) = 0 \) is a \((d-1)\)-dimensional **hyperplane** in the instance space.

The **weight vector** \( w \) is **orthogonal** (normal) to the decision boundary \( f(x) = 0 \):

For any two points \( x^A \) and \( x^B \) on the decision boundary \( f(x^A) = f(x^B) = 0 \):

\[
    w(x^B - x^A) = f(x^B) - w_0 - f(x^A) + w_0 = 0
\]

The **bias term** \( w_0 \) determines the **distance of the decision boundary** from the origin:

For \( x \) with \( f(x) = 0 \), the distance to the origin is

\[
    \frac{w \cdot x}{\|w\|} = -\frac{w_0}{\|w\|} = -\frac{w_0}{\sqrt{\sum_{i=1}^{d} w_i^2}}
\]
Canonical representation: getting rid of the bias term

With \( \mathbf{w} = (w_1, \ldots, w_N)^T \) and \( \mathbf{x} = (x_1, \ldots, x_N)^T \):

\[
f(x) = w_0 + \mathbf{wx} = w_0 + \sum_{i=1}^{N} w_i x_i
\]

\( w_0 \) is called the bias term.

The canonical representation redefines \( \mathbf{w}, \mathbf{x} \) as

\[
\mathbf{w} = (w_0, w_1, \ldots, w_N)^T
\]

and

\[
\mathbf{x} = (1, x_1, \ldots, x_N)^T
\]

\[
=> f(x) = \mathbf{w} \cdot \mathbf{x}
\]
Batch versus online training

**Batch learning:**
The learner sees the complete training data, and only changes its hypothesis when it has seen the entire training data set.

**Online training:**
The learner sees the training data one example at a time, and can change its hypothesis with every new example.

**Compromise: Minibatch learning (commonly used in practice)**
The learner sees small sets of training examples at a time, and changes its hypothesis with every such minibatch of examples.
Multi-class perceptrons

• *One-vs-others* framework: Need to keep a weight vector \( \mathbf{w}_c \) for each class \( c \)

• Decision rule: \( y = \text{argmax}_c \mathbf{w}_c \cdot \mathbf{f} \)

• Update rule: suppose example from class \( c \) gets misclassified as \( c' \)
  • Update for \( c \): \( \mathbf{w}_c \leftarrow \mathbf{w}_c + \eta \mathbf{f} \)
  • Update for \( c' \): \( \mathbf{w}_{c'} \leftarrow \mathbf{w}_{c'} - \eta \mathbf{f} \)
  • Update for all classes other than \( c \) and \( c' \): no change
Review: Multi-class perceptrons

- *One-vs-others* framework: Need to keep a weight vector $w_c$ for each class $c$
- Decision rule: $y = \arg\max_c w_c \cdot f$
Differentiable Perceptron

• Also known as a “one-layer feedforward neural network,” also known as “logistic regression.” Has been re-invented many times by many different people.

• Basic idea: replace the non-differentiable decision function

\[ y' = \text{sign}(\vec{w}^T \vec{f}) \]

with a differentiable decision function

\[ y' = \tanh(\vec{w}^T \vec{f}) \]
Differential Perceptron

The weights get updated according to

\[ \overrightarrow{w} = \overrightarrow{w} - \eta \nabla_{\overrightarrow{w}} L \]
Differentiable Multi-class perceptrons

Same idea works for multi-class perceptrons. We replace the non-differentiable decision rule $c = \arg\max_c \mathbf{w}_c \cdot \mathbf{f}$ with the differentiable decision rule $c = \text{softmax}_c \mathbf{w}_c \cdot \mathbf{f}$, where the softmax function is defined as

\[ p(c | \mathbf{f}) = \frac{e^{\mathbf{w}_c \cdot \mathbf{f}}}{\sum_{k=1}^{\# \text{classes}} e^{\mathbf{w}_k \cdot \mathbf{f}}} \]
Differentiable Multi-Class Perceptron

• Then we can define the loss to be:

\[ L(y_1, ..., y_n, \vec{f}_1, ..., \vec{f}_n) = -\sum_{i=1}^{n} \ln p(c = y_i | \vec{f}_i) \]

• And because the probability term on the inside is differentiable, we can reduce the loss using gradient descent:

\[ \vec{w} = \vec{w} - \eta \nabla_{\vec{w}} L \]
Training a Softmax Neural Network

All of that differentiation is useful because we want to train the neural network to represent a training database as well as possible. If we can define the training error to be some function $L$, then we want to update the weights according to

$$w_{mk} = w_{mk} - \eta \frac{\partial L}{\partial w_{mk}}$$

So what is $L$?
Training: Maximize the probability of the training data

Remember, the whole point of that denominator in the softmax function is that it allows us to use softmax as

$$\hat{y}_{ij} = \text{Estimated value of } P(\text{class } j \mid \vec{f}_i)$$

Suppose we decide to estimate the network weights $w_{mk}$ in order to maximize the probability of the training database, in the sense of

$$w_{mk} = \arg\max_{w} P(\text{training labels} \mid \text{training feature vectors})$$
Training: Maximize the probability of the training data

Remember, the whole point of that denominator in the softmax function is that it allows us to use softmax as

$$\hat{y}_{ij} = \text{Estimated value of } P(\text{class } j \mid \vec{f}_i)$$

If we assume the training tokens are independent, this is:

$$w_{mk} = \arg\max_w \prod_{i=1}^{n} P(\text{reference label of the } i^{th}\text{ token} \mid i^{th}\text{ feature vector})$$
Training: Maximize the probability of the training data

Remember, the whole point of that denominator in the softmax function is that it allows us to use softmax as

\[ \hat{y}_{ij} = \text{Estimated value of } P(\text{class } j \mid f_i) \]

OK. We need to create some notation to mean “the reference label for the \( i^{th} \) token.” Let’s call it \( j(i) \).

\[ w_{mk} = \arg\max_w \prod_{i=1}^n P(\text{class } j(i) \mid f) \]
Training: Maximize the probability of the training data

Wow, Cool!! So we can maximize the probability of the training data by just picking the softmax output corresponding to the correct class \( j(i) \), for each token, and then multiplying them all together:

\[
w_{mk} = \arg\max_w \prod_{i=1}^{n} \hat{y}_{i,j(i)}
\]

So, hey, let’s take the logarithm, to get rid of that nasty product operation.

\[
w_{mk} = \arg\max_w \sum_{i=1}^{n} \ln \hat{y}_{i,j(i)}
\]
Training: Minimizing the negative log probability

So, to maximize the probability of the training data given the model, we need:

\[ w_{mk} = \arg\max_w \sum_{i=1}^{n} \ln \hat{y}_{i,j(i)} \]

If we just multiply by \((-1)\), that will turn the max into a min. It’s kind of a stupid thing to do---who cares whether you’re minimizing \(L\) or maximizing \(-L\), same thing, right? But it’s standard, so what the heck.

\[ w_{mk} = \arg\min_w L \]

\[ L = \sum_{i=1}^{n} -\ln \hat{y}_{i,j(i)} \]
Training: Minimizing the negative log probability

Softmax neural networks are almost always trained in order to minimize the negative log probability of the training data:

$$w_{mk} = \arg\min_w L$$

$$L = \sum_{i=1}^{n} - \ln \hat{y}_{i,j(i)}$$

This loss function, defined above, is called the cross-entropy loss. The reasons for that name are very cool, and very far beyond the scope of this course. Take CS 446 (Machine Learning) and/or ECE 563 (Information Theory) to learn more.
Summary: Training Algorithms You Know

1. Naïve Bayes with Laplace Smoothing:
   \[
P(f_k = x | \text{class } j) = \frac{(\# \text{tokens of class } j \text{ with } f_k = x) + 1}{(\# \text{tokens of class } j) + (\# \text{possible values of } f_k)}
   \]

2. Multi-Class Perceptron: If token \( \vec{f}_i \) of class \( j \) is misclassified as class \( m \), then
   \[
   \vec{w}_j = \vec{w}_j + \eta \vec{f}_i
   \]
   \[
   \vec{w}_m = \vec{w}_m - \eta \vec{f}_i
   \]

3. Softmax Neural Net: for all weight vectors (correct or incorrect),
   \[
   \vec{w}_m = \vec{w}_m - \eta \nabla_{\vec{w}_m} L
   \]
   \[
   = \vec{w}_m - \eta (\hat{y}_{im} - y_{im}) \vec{f}_i
   \]
Summary: Perceptron versus Softmax

Softmax Neural Net: for all weight vectors (correct or incorrect),
\[ \vec{w}_m = \vec{w}_m - \eta (\hat{y}_{im} - y_{im}) \hat{f}_i \]

Notice that, if the network were adjusted so that
\[ \hat{y}_{im} = \begin{cases} 
1 & \text{network thinks the correct class is } m \\
0 & \text{otherwise} 
\end{cases} \]

Then we’d have
\[ (\hat{y}_{im} - y_{im}) = \begin{cases} 
-2 & \text{correct class is } m, \text{ but network is wrong} \\
2 & \text{network guesses } m, \text{ but it’s wrong} \\
0 & \text{otherwise} 
\end{cases} \]
Summary: Perceptron versus Softmax

Softmax Neural Net: for all weight vectors (correct or incorrect),

$$\vec{w}_m = \vec{w}_m - \eta (\hat{y}_{im} - y_{im}) \vec{f}_i$$

Notice that, if the network were adjusted so that

$$\hat{y}_{im} = \begin{cases} 
1 & \text{network thinks the correct class is } m \\
0 & \text{otherwise}
\end{cases}$$

Then we get the perceptron update rule back again (multiplied by 2, which doesn’t matter):

$$\vec{w}_m = \begin{cases} 
\vec{w}_m + 2\eta \vec{f}_i & \text{correct class is } m, \text{ but network is wrong} \\
\vec{w}_m - 2\eta \vec{f}_i & \text{network guesses } m, \text{ but it’s wrong} \\
\vec{w}_m & \text{otherwise}
\end{cases}$$
Summary: Perceptron versus Softmax

So the key difference between perceptron and softmax is that, for a perceptron,

\[ \hat{y}_{ij} = \begin{cases} 1 & \text{network thinks the correct class is } j \\ 0 & \text{otherwise} \end{cases} \]

Whereas, for a softmax,

\[ 0 \leq \hat{y}_{ij} \leq 1, \quad \sum_{j=1}^{c} \hat{y}_{ij} = 1 \]
Summary: Perceptron versus Softmax

...or, to put it another way, for a perceptron,

\[
\hat{y}_{ij} = \begin{cases} 
1 & \text{if } j = \arg\max_{1 \leq \ell \leq c} \vec{w}_\ell \cdot \vec{f}_i \\
0 & \text{otherwise}
\end{cases}
\]

Whereas, for a softmax network,

\[
\hat{y}_{ij} = \operatorname{softmax}_j(\vec{w}_\ell \cdot \vec{f}_i)
\]
CS 440/ECE448 Lecture 19: Bayes Net Inference

Including slides by Svetlana Lazebnik, 11/2016

| B | E | P(A|B,E) |
|---|---|---------|
| T | T |  .95    |
| T | F |  .94    |
| F | T |  .29    |
| F | F |  .001   |

| B | E | P(A|B,E) |
|---|---|---------|
| T | T |  .95    |
| T | F |  .94    |
| F | T |  .29    |
| F | F |  .001   |

| B | E | P(A|B,E) |
|---|---|---------|
| T | T |  .95    |
| T | F |  .94    |
| F | T |  .29    |
| F | F |  .001   |

| B | E | P(A|B,E) |
|---|---|---------|
| T | T |  .95    |
| T | F |  .94    |
| F | T |  .29    |
| F | F |  .001   |

| B | E | P(A|B,E) |
|---|---|---------|
| T | T |  .95    |
| T | F |  .94    |
| F | T |  .29    |
| F | F |  .001   |
Bayesian Inference with Hidden Variables

- **A general scenario:**
  - **Query variables:** $X$
  - **Evidence (observed) variables and their values:** $E = e$
  - **Hidden (unobserved) variables:** $Y$

- **Inference problem:** answer questions about the query variables given the evidence variables
  - This can be done using the posterior distribution $P(X \mid E = e)$
  - In turn, the posterior needs to be derived from the full joint $P(X, E, Y)$

$$P(X \mid E = e) = \frac{P(X, e)}{P(e)} \propto \sum_y P(X, e, y)$$

- Bayesian networks are a tool for representing joint probability distributions efficiently
Bayesian networks

- **Nodes**: random variables
- **Edges**: dependencies
  - An edge from one variable (parent) to another (child) indicates direct influence (conditional probabilities)
  - Edges must form a directed, *acyclic* graph
  - Each node is conditioned on its parents: \( P(X \mid \text{Parents}(X)) \)
    These conditional distributions are the parameters of the network
- **Each node is conditionally independent of its non-descendants given its parent**

We have four random variables
Weather is independent of cavity, toothache and catch
Toothache and catch both depend on cavity.
Conditional independence and the joint distribution

- **Key property:** each node is conditionally independent of its *non-descendants* given its *parents*

- Suppose the nodes $X_1, \ldots, X_n$ are sorted in topological order of the graph (i.e. if $X_i$ is a parent of $X_j$, $i < j$)

- To get the joint distribution $P(X_1, \ldots, X_n)$, use chain rule (step 1 below) and then take advantage of independencies (step 2)

\[
P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i \mid X_1, \ldots, X_{i-1})
\]

\[
= \prod_{i=1}^{n} P(X_i \mid \text{Parents}(X_i))
\]
The joint probability distribution

\[ P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i \mid Parents(X_i)) \]

\[ P(j, m, a, \neg b, \neg e) = P(\neg b) \cdot P(\neg e) \cdot P(a \mid \neg b, \neg e) \cdot P(j \mid a) \cdot P(m \mid a) \]
Example: N independent coin flips

- Complete independence: no interactions:
  \[ P(X_1) \cdot P(X_2) \cdot P(X_3) \]  

\[ X_1 \quad X_2 \quad \ldots \quad X_n \]
Conditional probability distributions

- To specify the full joint distribution, we need to specify a conditional distribution for each node given its parents:
  \[ P(X | \text{Parents}(X)) \]
Naïve Bayes document model

- Random variables:
  - $X$: document class
  - $W_1, ..., W_n$: words in the document

- Dependencies: $P(X) \ P(W_1 \mid X) \ldots \ P(W_n \mid X)$
Independence

• By saying that $X_i$ and $X_j$ are independent, we mean that
  \[ P(X_j, X_i) = P(X_i) P(X_j) \]

• $X_i$ and $X_j$ are independent if and only if they have no common ancestors

• Example: *independent coin flips*

  \[ \begin{array}{c}
    X_1 \\
    X_2 \\
    \vdots \\
    X_n
  \end{array} \]

• Another example: Weather is independent of all other variables in this model.
Conditional independence

• By saying that $W_i$ and $W_j$ are conditionally independent given $X$, we mean that

$$P(W_i, W_j | X) = P(W_i | X)P(W_j | X)$$

• $W_i$ and $W_j$ are conditionally independent given $X$ if and only if they have no common ancestors other than the ancestors of $X$.

• Example: naïve Bayes model:
**Conditional independence ≠ Independence**

**Common cause:** Conditionally Independent

- **Y**: Project due
- **X**: Newsgroup busy
- **Z**: Lab full

Are X and Z independent? **No**

\[
P(Z, X) = \sum_{Y} P(Z|Y)P(X|Y)P(Y)
\]

\[
P(Z)P(X) = \left(\sum_{Y} P(Z|Y)P(Y)\right) \left(\sum_{Y} P(X|Y)P(Y)\right)
\]

Are they conditionally independent given Y? **Yes**

\[
P(Z, X|Y) = P(Z|Y)P(X|Y)
\]

**Common effect:** Independent

- **X**: Raining
- **Z**: Ballgame
- **Y**: Traffic

Are X and Z independent? **Yes**

\[
P(X, Z) = P(X)P(Z)
\]

Are they conditionally independent given Y? **No**

\[
P(Z, X|Y) = \frac{P(Y|X, Z)P(X)P(Z)}{P(Y)} 
\neq P(Z|Y)P(X|Y)
\]
Constructing a Bayes Network: Two Methods

   1. Suppose you know the variables, but you don’t know which variables depend on which others. You can learn this from data.
   2. This is an exciting new area of research in statistics, where it goes by the name of “analysis of causality.”
   3. ... but it’s almost always harder than method #2. You should know how to do this in very simple examples (like the Los Angeles burglar alarm), but you don’t need to know how to do this in the general case.

2. “Hire an Expert:”
   1. Find somebody who knows how to solve the problem.
   2. Get her to tell you what are the important variables, and which variables depend on which others.
   3. THIS IS ALMOST ALWAYS THE BEST WAY.
Bayes Network Inference & Learning

Bayes net is a memory-efficient model of dependencies among a set of random variables.

**Inference problem:** answer questions about the query variables $X$ given the evidence variables and their values $E=e$ as well as some unobserved (hidden) variables $Y$.

- We want to know the posterior distribution $P(X \mid E = e)$
- The posterior can be derived from the full joint $P(X, E, Y)$
- How do we make this computationally efficient?

**Learning problem:** given some training examples, how do we estimate the parameters of the model?

- Parameters = $p(\text{variable} \mid \text{parents})$, for each variable in the net
Bayes Net Inference: The Hard Way

1. \[ P(B, E, A, J, M) = P(B) \cdot P(E) \cdot P(A|B,E) \cdot P(J|A) \cdot P(M|A) \]
2. \[ P(B, J) = \sum_E \sum_A \sum_M P(B, E, A, J, M) \]

Exponential complexity (#P-hard, actually): N variables, each of which has K possible values \( \Rightarrow O\{K^N\} \) time complexity
Is there an easier way?

- **Tree-structured Bayes nets: the sum-product algorithm**
  - Quadratic complexity, $O\{NK^3\}$
- **Polytrees: the junction tree algorithm**
  - Pseudo-polynomial complexity, $O\{NK^M\}$, for $M<N$
- **Arbitrary Bayes nets: #P complete, $O\{K^N\}$**
  - The SAT problem is a Bayes net!
The Sum-Product Algorithm (Belief Propagation)

- Find the **only undirected path** from the evidence variable to the query variable (E-D-B-F-G-I-H)
- Find the **directed root** of this path $P(F)$
- Find the **joint probabilities of root and evidence**: $P(F=0, E=1)$ and $P(F=1, E=1)$
- Find the **joint probabilities of query and evidence**: $P(H=0, E=1)$ and $P(H=1, E=1)$
- Find the **conditional probability** $P(H=1 | E=1)$
The Sum-Product Algorithm (Belief Propagation)

Starting with the root $P(F)$, we find $P(F,E)$ by alternating product steps and sum steps:

1. **Product:**
   \[ P(B,D,F) = P(F)P(B|F)P(D|B) \]

2. **Sum:**
   \[ P(D,F) = \sum_{B=0}^{1} P(B,D,F) \]

3. **Product:**
   \[ P(D,E,F) = P(D,F)P(E|D) \]

4. **Sum:**
   \[ P(E,F) = \sum_{D=0}^{1} P(D,E,F) \]
The Sum-Product Algorithm (Belief Propagation)

Starting with the root $P(E,F)$, we find $P(E,H)$ by alternating product steps and sum steps:

1. Product: $P(E,F,G) = P(E,F)P(G|F)$
2. Sum: $P(E,G) = \sum_{F=0}^{1} P(E,F,G)$
4. Sum: $P(E,I) = \sum_{G=0}^{1} P(E,G,I)$
6. Sum: $P(E,H) = \sum_{I=0}^{1} P(E,H,I)$
Time Complexity of Belief Propagation

- Each **product** step generates a table with 3 variables.
- Each **sum** step reduces that to a table with 2 variables.
- If each variable has K values, and if there are $O(N)$ variables on the path from evidence to query, then time complexity is $O(NK^3)$. 
2. The Junction Tree Algorithm

a. **Moralize** the graph (identify each variable’s Markov blanket)

b. **Triangulate** the graph (eliminate undirected cycles)

c. Create the **junction tree** (form cliques)

d. **Run the sum-product algorithm** on the junction tree
2.a. Markov Blanket

The Markov Blanket of variable F includes only its immediate family members:

- Its parent, D
- Its child, G
- The other parent of its child, E

Because \( P(F|A,B,C,D,E,G,H) = P(F|D,E,G) \)
2.a. Moralization

“Moralization” =

1. If two variables have a child together, force them to get married.
2. Get rid of the arrows (not necessary any more).

Result: **Markov blanket = the set of variables to which a variable is connected.**
2.b. Triangulation

**Triangulation** = draw edges so that there is no unbroken cycle of length $> 3$.

There are usually many different ways to do this. For example, here’s one:
2.c. Form Cliques

Clique = a group of variables, all of whom are members of each other’s immediate family.

Junction Tree = a tree in which
- Each node is a clique from the original graph,
- Each edge is an “intersection set,” naming the variables that overlap between the two cliques.
2.d. Sum-Product

Suppose we need $P(B,G)$:

1. Product: $P(B,C,D,F) = P(B)P(C|B)P(D|B)P(F|D)$
2. Sum: $P(B,C,F) = \sum_D P(B,C,D,F)$
4. Sum: $P(B,E,F) = \sum_C P(B,C,E,F)$
5. Product: $P(B,E,F,G) = P(B,E,F)P(G|E,F)$
6. Sum: $P(B,G) = \sum_E \sum_F P(B,E,F,G)$

Complexity: $O\{NK^M\}$, where $N = \#$ cliques,
$K = \#$ values for each variable,
$M = 1 + \#$ variables in the largest clique
Consider the burglar alarm example.

a. Moralize this graph

b. Is it already triangulated? If not, triangulate it.

c. Draw the junction tree
Solution

B
E
A
J
M
Solution

a. Moralize this graph
b. Is it already triangulated?

Answer: yes. There is no unbroken cycle of length > 3.
Solution

c. Draw the junction tree
Time Complexity of Bayes Net Inference

• Tree-structured Bayes nets: the sum-product algorithm
  • Quadratic complexity, $O\{NK^3\}$
• Polytrees: the junction tree algorithm
  • Pseudo-polynomial complexity, $O\{NK^M\}$, for M<N
• Arbitrary Bayes nets: #P complete, $O\{K^N\}$
  • The SAT problem is a Bayes net!
Parameter learning

- **Inference problem**: given values of evidence variables $E = e$, answer questions about query variables $X$ using the posterior $P(X | E = e)$

- **Learning problem**: estimate the parameters of the probabilistic model $P(X | E)$ given a *training sample* $\{(x_1,e_1), \ldots, (x_n,e_n)\}$
Parameter learning

• Suppose we know the network structure (but not the parameters), and have a training set of complete observations

• Example:

\[ P(S = T|C = T) = \frac{\# \text{ samples with } S = T, C = T}{\# \text{ samples with } C = T} = \frac{1}{4} \]

<table>
<thead>
<tr>
<th>Sample</th>
<th>C</th>
<th>S</th>
<th>R</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>2</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>3</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>4</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>5</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>6</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Suppose we know the network structure (but not the parameters), and have a training set, but the training set is *missing some observations.*
Missing data: the EM algorithm

• The EM algorithm starts ("Expectation Maximization") starts with an initial guess for each parameter value.

• We try to improve the initial guess, using the algorithm on the next two slides:
  • E-step
  • M-step

The EM algorithm starts ("Expectation Maximization")
starts with an initial guess for each parameter value.

We try to improve the initial guess, using the algorithm on the next two slides:

- E-step
- M-step

Training set

<table>
<thead>
<tr>
<th>Sample</th>
<th>C</th>
<th>S</th>
<th>R</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>?</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>2</td>
<td>?</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>3</td>
<td>?</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>4</td>
<td>?</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>5</td>
<td>?</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>6</td>
<td>?</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>
Missing data: the EM algorithm

- E-Step (Expectation): Given the model parameters, replace each of the missing numbers with a probability (a number between 0 and 1) using

\[ P(C = 1|S, R, W) = \frac{P(C = 1, S, R, W)}{P(C = 1, S, R, W) + P(C = 0, S, R, W)} \]

Training set

<table>
<thead>
<tr>
<th>Sample</th>
<th>C</th>
<th>S</th>
<th>R</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.5?</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>2</td>
<td>0.5?</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>3</td>
<td>0.5?</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>4</td>
<td>0.5?</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>5</td>
<td>0.5?</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>6</td>
<td>0.5?</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>
Missing data: the EM algorithm

- M-Step (Maximization): Given the missing data estimates, replace each of the missing model parameters using

\[
P(\text{Variable} = T|\text{Parents} = \text{value}) = \frac{E[\# \text{ times Variable} = T, \text{Parents} = \text{value}]}{E[\# \text{times Parents} = \text{value}]}\]

Training set

<table>
<thead>
<tr>
<th>Sample</th>
<th>C</th>
<th>S</th>
<th>R</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.5?</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>2</td>
<td>0.5?</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>3</td>
<td>0.5?</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>4</td>
<td>0.5?</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>5</td>
<td>0.5?</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>6</td>
<td>0.5?</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Hidden Markov Model = [ i ↦ j ↦ k ↦ ... ]

State Sequence $Q = [ i \ i \ i \ j \ j \ k \ k \ k \ ... ]$

Observations $O = [ o_1 \ o_2 \ o_3 \ o_4 \ o_5 \ o_6 \ o_7 \ o_8 \ o_9 ]$
Hidden Markov Models

- At each time slice $t$, the state of the world is described by an unobservable (hidden) variable $X_t$ and an observable evidence variable $E_t$

  - **Transition model**: distribution over the current state given the whole past history:
    \[ P(X_t \mid X_0, ..., X_{t-1}) = P(X_t \mid X_{0:t-1}) \]

  - **Observation model**: $P(E_t \mid X_{0:t}, E_{1:t-1})$
Hidden Markov Models

- **Markov assumption** (first order)
  - The current state is conditionally independent of all the other states given the state in the previous time step
  - What does $P(X_t | X_{0:t-1})$ simplify to?
    $$P(X_t | X_{0:t-1}) = P(X_t | X_{t-1})$$

- **Markov assumption for observations**
  - The evidence at time $t$ depends only on the state at time $t$
  - What does $P(E_t | X_{0:t}, E_{1:t-1})$ simplify to?
    $$P(E_t | X_{0:t}, E_{1:t-1}) = P(E_t | X_t)$$
The Joint Distribution

• Transition model: $P(X_t \mid X_{0:t-1}) = P(X_t \mid X_{t-1})$
• Observation model: $P(E_t \mid X_{0:t}, E_{1:t-1}) = P(E_t \mid X_t)$
• How do we compute the full joint $P(X_{0:t}, E_{1:t})$?

$$P(X_{0:t}, E_{1:t}) = P(X_0) \prod_{i=1}^{t} P(X_i \mid X_{i-1}) P(E_i \mid X_i)$$

\[
\begin{align*}
X_0 & \rightarrow X_1 & \rightarrow X_2 & \rightarrow \cdots & \rightarrow X_{t-1} & \rightarrow X_t \\
E_1 & \quad \downarrow & \quad \downarrow & \quad \downarrow & \quad \downarrow & \quad \downarrow \\
E_2 & \quad \downarrow & \quad \downarrow & \quad \downarrow & \quad \downarrow & \quad \downarrow \\
E_{t-1} & \quad \downarrow & \quad \downarrow & \quad \downarrow & \quad \downarrow & \quad \downarrow \\
E_t & \quad \downarrow & \quad \downarrow & \quad \downarrow & \quad \downarrow & \quad \downarrow \\
\end{align*}
\]
HMM inference tasks

- **Filtering**: what is the distribution over the current state $X_t$ given all the evidence so far, $e_{1:t}$?
  - The forward algorithm = sum-product algorithm for $X_t$ given $e_{1:t}$
HMM inference tasks

- **Filtering:** what is the distribution over the current state $X_t$ given all the evidence so far, $e_{1:t}$?
- **Smoothing:** what is the distribution of some state $X_k$ given the entire observation sequence $e_{1:t}$?
  - The forward-backward algorithm = sum-product algorithm for $X_k$ given $e_{1:t}$, when $1 < k < t$
  - $X_k$ = query variable, unknown, need to consider all its possible values
  - $E_{1:t}$ = evidence variables, known, only need to consider the given values
HMM inference tasks

- **Filtering**: what is the distribution over the current state $X_t$ given all the evidence so far, $e_{1:t}$?
- **Smoothing**: what is the distribution of some state $X_k$ given the entire observation sequence $e_{1:t}$?
- **Evaluation**: compute the probability of a given observation sequence $e_{1:t}$
HMM inference tasks

- **Filtering:** what is the distribution over the current state $X_t$ given all the evidence so far, $e_{1:t}$
- **Smoothing:** what is the distribution of some state $X_k$ given the entire observation sequence $e_{1:t}$?
- **Evaluation:** compute the probability of a given observation sequence $e_{1:t}$
- **Decoding:** what is the most likely state sequence $X_{0:t}$ given the observation sequence $e_{1:t}$?
  - The Viterbi algorithm
HMM Learning and Inference

• Inference tasks
  • **Filtering**: what is the distribution over the current state $X_t$ given all the evidence so far, $e_{1:t}$
  • **Smoothing**: what is the distribution of some state $X_k$ given the entire observation sequence $e_{1:t}$?
  • **Evaluation**: compute the probability of a given observation sequence $e_{1:t}$
  • **Decoding**: what is the most likely state sequence $X_{0:t}$ given the observation sequence $e_{1:t}$?

• Learning
  • Given a training sample of sequences, learn the model parameters (transition and emission probabilities)
    • EM algorithm