CS395T: Structured Models for NLP
Lecture 6: Sequence Models III

Greg Durrett

Some slides adapted from Leon Gu (CMU), Taylor Berg-Kirkpatrick (CMU)
P1 has been updated, didn’t include adagrad_trainer.py
Recall: CRFs

\[ P(y|x) = \frac{1}{Z} \prod_{i=2}^{n} \exp(\phi_t(y_{i-1}, y_i)) \prod_{i=1}^{n} \exp(\phi_e(y_i, i, x)) \]

- Using standard feature-based potentials:

\[ P(y|x) \propto \exp w^\top \left[ \sum_{i=2}^{n} f_t(y_{i-1}, y_i) + \sum_{i=1}^{n} f_e(y_i, i, x) \right] \]

- Gradient: gold features - expected features under model

- Compute max path with Viterbi, compute feature expectations from tag probabilities with forward-backward
Structured SVM

\[ w^\top f(x, y) = \sum_{i=2}^{n} w^\top f_t(y_{i-1}, y_i) + \sum_{i=1}^{n} w^\top f_e(x_i, y_i) \]

Minimize \( \lambda \|w\|_2^2 + \sum_{j=1}^{m} \xi_j \)

s.t. \( \forall j \quad \xi_j \geq 0 \)

\( \forall j \forall y \in Y \quad w^\top f(x_j, y_j^*) \geq w^\top f(x_j, y) + \ell(y, y_j^*) - \xi_j \)

- Loss-augmented decode can be done with Viterbi
- Only need Viterbi for inference here...hmm...
Viterbi Time Complexity

Fed raises interest rates 0.5 percent

- n word sentence, s tags to consider — what is the time complexity?

- $O(ns^2)$ — s is ~40 for POS, n is ~20
Viterbi Time Complexity

- Many tags are totally implausible
- Can any of these be:
  - Determiners?
  - Prepositions?
  - Adjectives?
- Features quickly eliminate many outcomes from consideration — don’t need to consider these going forward

Fed raises interest rates 0.5 percent

```
VBD VBN VBZ VB VBP VBZ NNP NNS NN NNS CD NN
```

Beam Search

- Maintain a beam of $k$ plausible states at the current timestep
- Expand all states, only keep $k$ top hypotheses at new timestep

- Beam size of $k$, time complexity $O(nks \log(k))$

Maintain priority queue to efficiently add things

Fed raises Not expanded

Maintains priority queue to efficiently add things
How good is beam search?

- $k=1$: greedy search

Choosing beam size:
- 2 is usually better than 1
- Usually don’t use larger than 50
- Depends on problem structure
This Lecture

- Unsupervised POS tagging
- EM for learning HMMs
- Gradient-based unsupervised learning
- (briefly) Some writing tips
Can we induce linguistic structure? Thought experiment...

a b a c c c c
b a c c c

What’s a two-state HMM that could produce this?

What if I show you this sequence?

a a b c c a a

What did you do? Use current model parameters + data to refine your model. This is what EM will do
Part-of-Speech Induction

- Input $x = (x_1, ..., x_n)$  
  Output $y = (y_1, ..., y_n)$

- Assume we don’t have access to labeled examples — how can we learn a POS tagger?

- Key idea: optimize $P(x) = \sum_y P(y, x)$

- Optimizing marginal log-likelihood with no labels $y$:

$$\mathcal{L}(x_1, ..., D) = \sum_{i=1}^{D} \log \sum_y P(y, x_i)$$

- non-convex optimization problem
Part-of-Speech Induction

- Input $x = (x_1, ..., x_n)$ Output $y = (y_1, ..., y_n)$

- Optimizing marginal log-likelihood with no labels $y$:

  $$\mathcal{L}(x_1, ..., D) = \sum_{i=1}^{D} \log \sum_{y} P(y, x_i)$$

- Can’t use a discriminative model; $\sum_{y} P(y|x) = 1$, doesn’t model $x$

- What’s the point of this? Model has inductive bias and so should learn some useful latent structure $y$ (clustering effect)

- EM is just one procedure for optimizing this kind of objective
Expectation Maximization

\[ \log \sum_{y} P(x, y|\theta) \]  
\[ = \log \sum_{y} q(y) \frac{P(x, y|\theta)}{q(y)} \]  
\[ \geq \sum_{y} q(y) \log \frac{P(x, y|\theta)}{q(y)} \]  
\[ = \mathbb{E}_{q(y)} \log P(x, y|\theta) + \text{Entropy}[q(y)] \]

- Condition on parameters \( \theta \)
- Variational approximation \( q \) — this is a trick we’ll return to later!
- Jensen’s inequality (uses concavity of log)
- Can optimize this lower-bound on log likelihood instead of log-likelihood

Adapted from Leon Gu
Expectation Maximization

\[
\log \sum_y P(x, y|\theta) \geq \mathbb{E}_{q(y)} \log P(x, y|\theta) + \text{Entropy}[q(y)]
\]

- Exact equality:

\[
\log \sum_y P(x, y|\theta) = \mathbb{E}_{q(y)} \log P(x, y|\theta) + \text{Entropy}[q(y)] + KL(q(y)||P(y|x, \theta))
\]

- KL divergence: asymmetric measure of difference between two distributions

\[
KL(q(y)||p(y)) = \sum_y q(y) \log \frac{q(y)}{p(y)}
\]

- Related to cross-entropy (= KL + entropy of q)

- If \(q(y) = P(y|x, \theta)\), KL term is 0 so equality is achieved

Adapted from Leon Gu
Expectation Maximization

\[ \log \sum_y P(x, y | \theta) \geq \mathbb{E}_{q(y)} \log P(x, y | \theta) + \text{Entropy}[q(y)] \]

- If \( q(y) = P(y | x, \theta) \), KL term is 0 so equality is achieved

- Expectation-maximization: alternating maximization of the lower bound over \( q \) and \( \theta \)
  - Current timestep = \( t \), have parameters \( \theta^{t-1} \)
  - E-step: maximize w.r.t. \( q \); that is, \( q^t = P(y | x, \theta^{t-1}) \)
  - M-step: maximize w.r.t. \( \theta \); that is, \( \theta^t = \arg\max_\theta \mathbb{E}_{q^t} \log P(x, y | \theta) \)

Adapted from Leon Gu
EM for HMMs

- Expectation-maximization: alternating maximization
  - E-step: maximize w.r.t. $q$; that is, $q^t = P(y|x, \theta^{t-1})$
  - M-step: maximize w.r.t. $\theta$; that is, $
  \theta^t = \arg\max_{\theta} \mathbb{E}_{q^t} \log P(x, y|\theta)$

- E-step: for an HMM: run forward-backward with the given parameters

- Compute $P(y_i = s|x, \theta^{t-1})$, $P(y_i = s_1, y_{i+1} = s_2|x, \theta^{t-1})$
  
  tag marginals at each position
  tag pair marginals at each position

- M-step: need to find parameters to optimize the crazy argmax term
EM for HMMs

- Recall how we maximized log $P(x, y)$: read counts off data

<table>
<thead>
<tr>
<th>DT</th>
<th>NN</th>
</tr>
</thead>
<tbody>
<tr>
<td>the</td>
<td>dog</td>
</tr>
</tbody>
</table>

  - count(DT, the) = 1  \quad P(\text{the}|\text{DT}) = 1
  - count(DT, dog) = 0  \quad P(\text{dog}|\text{DT}) = 0
  - count(NN, the) = 0  \quad P(\text{the}|\text{NN}) = 0
  - count(NN, dog) = 1  \quad P(\text{dog}|\text{NN}) = 1

- Same procedure, but maximizing $P(x, y)$ in expectation under $q$ means that $q$ specifies *fractional counts*

<table>
<thead>
<tr>
<th>DT</th>
<th>NN</th>
</tr>
</thead>
<tbody>
<tr>
<td>the</td>
<td>dog</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$q$</th>
<th>DT: 0.9</th>
<th>DT: 0.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>NN: 0.1</td>
<td>NN: 0.7</td>
<td></td>
</tr>
</tbody>
</table>

  - count(DT, the) = 0.9  \quad P(\text{the}|\text{DT}) = 0.75
  - count(DT, dog) = 0.3  \quad P(\text{dog}|\text{DT}) = 0.25
  - count(NN, the) = 0.1  \quad P(\text{the}|\text{NN}) = 0.125
  - count(NN, dog) = 0.7  \quad P(\text{dog}|\text{NN}) = 0.875
EM for HMMs

- Same for transition probabilities

\[
\begin{align*}
q & \quad \text{DT—NN: 0.6} & \quad \text{P(DT|DT) = 1/7} \\
& \quad \text{DT—DT: 0.1} & \quad \text{P(NN|DT) = 6/7} \\
& \quad \text{NN—DT: 0.2} & \quad \text{P(DT|NN) = 2/3} \\
& \quad \text{NN—NN: 0.1} & \quad \text{P(NN|NN) = 1/3}
\end{align*}
\]

\text{the dog}
How does EM learn things?

- Initialize (M-step 0):
  - Emissions
    
    $P(\text{the} | \text{DT}) = 0.9$  \hspace{1cm} $P(\text{the} | \text{NN}) = 0.05$
    
    $P(\text{dog} | \text{DT}) = 0.05$  \hspace{1cm} $P(\text{dog} | \text{NN}) = 0.9$
    
    $P(\text{marsupial} | \text{DT}) = 0.05$  \hspace{1cm} $P(\text{marsupial} | \text{NN}) = 0.05$

  - Transition probabilities: uniform

- E-step 1: (all values are approximate)

  \textbf{DT}: 0.95  \hspace{1cm} \textbf{DT}: 0.05  \hspace{1cm} \textbf{DT}: 0.95  \hspace{1cm} \textbf{DT}: 0.5

  \textbf{NN}: 0.05  \hspace{1cm} \textbf{NN}: 0.95  \hspace{1cm} \textbf{NN}: 0.05  \hspace{1cm} \textbf{NN}: 0.5

  \textit{the}  \hspace{1cm} \textit{dog}  \hspace{1cm} \textit{the}  \hspace{1cm} \textit{marsupial}  \hspace{1cm} \textit{uniform}
How does EM learn things?

E-step 1:

- DT: 0.95  DT: 0.05  DT: 0.95  DT: 0.5
- NN: 0.05  NN: 0.95  NN: 0.05  NN: 0.5

  the   dog   the   marsupial

M-step 1:

- Emissions aren’t so different
- Transition probabilities (approx): P(NN|DT) = 3/4, P(DT|DT) = 1/4
How does EM learn things?

- **E-step 2:**
  
  - **DT:** 0.95  **DT:** 0.05  **DT:** 0.95  **DT:** 0.30
  - **NN:** 0.05  **NN:** 0.95  **NN:** 0.05  **NN:** 0.70
  - the dog  the marsupial

- **M-step 1:**
  
  - Emissions aren’t so different
  - Transition probabilities (approx): $P(\text{NN}|\text{DT}) = 3/4$, $P(\text{DT}|\text{DT}) = 1/4$
How does EM learn things?

- **E-step 2:**
  
  \[
  \begin{array}{ccc}
  \text{DT: } 0.95 & \text{DT: } 0.05 & \text{DT: } 0.95 \\
  \text{NN: } 0.05 & \textbf{NN: } 0.95 & \text{NN: } 0.70 \\
  \text{the} & \text{dog} & \text{the} \\
  \text{marsupial} & & \\
  \end{array}
  \]

- **M-step 2:**
  
  - Emission $P(\text{marsupial} | \text{NN}) > P(\text{marsupial} | \text{DT})$
  - Remember to tag marsupial as NN in the future!
  - Context constrained what we learned! That’s how data helped us
How does EM learn things?

- Can think of $q$ as a kind of “fractional annotation”
- E-step: compute annotations (posterior under current model)
- M-step: supervised learning with those fractional annotations
- Initialize with some reasonable weights, alternate E and M until convergence
EM’s Lower Bound

\[ \mathcal{L}(x_1, \ldots, D) = \sum_{i=1}^{D} \log \sum_{y} P(y, x_i) \]

- Initialize probabilities \( \theta \)
- repeat
  - Compute expected counts \( e \)
  - Fit parameters \( \theta \)
- until convergence

slide credit: Taylor Berg-Kirkpatrick
EM’s Lower Bound

\[ \mathcal{L}(x_1, \ldots, D) = \sum_{i=1}^{D} \log \sum_y P(y, x_i) \]

\[ \mathcal{L}(x_1, \ldots, D; \theta) \]

- E-step: compute \( q \) which gives this lower bound
- Initial theta
- Repeat
  - Compute expected counts \( e \)
  - Fit parameters \( \theta \)
- Until convergence

Initialize probabilities \( \theta \)
EM’s Lower Bound

\[
\mathcal{L}(x_1, \ldots, D) = \sum_{i=1}^{D} \log \sum_y P(y, x_i)
\]

- **M-step:** find maximum of lower bound

\[
\mathcal{L}(x_1, \ldots, D; \theta)
\]

Initialize probabilities \( \theta \)

**repeat**

- Compute expected counts \( e \)
- Fit parameters \( \theta \)

**until** convergence

---

slide credit: Taylor Berg-Kirkpatrick
EM’s Lower Bound

\[ \mathcal{L}(x_1, \ldots, D) = \sum_{i=1}^{D} \log \sum_y P(y, x_i) \]

- Initialize probabilities \( \theta \)
- Repeat
  - Compute expected counts \( e \)
  - Fit parameters \( \theta \)
- Until convergence

E-step 2: re-estimate \( q \)
EM’s Lower Bound

\[ \mathcal{L}(x_1, \ldots, D) = \sum_{i=1}^{D} \log \sum_y P(y, x_i) \]

\[ \mathcal{L}(x_1, \ldots, D; \theta) \]

- Initialize probabilities \( \theta \)
- repeat
  - Compute expected counts \( e \)
  - Fit parameters \( \theta \)
- until convergence

- E-step 2: re-estimate \( q \)

Slide credit: Taylor Berg-Kirkpatrick
EM’s Lower Bound

\[ \mathcal{L}(x_1, \ldots, D) = \sum_{i=1}^{D} \log \sum_{y} P(y, x_i) \]

\[ \mathcal{L}(x_1, \ldots, D; \theta) \]

Initialize probabilities \( \theta \)

repeat

- Compute expected counts \( e \)
- Fit parameters \( \theta \)

until convergence

slide credit: Taylor Berg-Kirkpatrick
EM’s Lower Bound

\[ \mathcal{L}(x_1, \ldots, D) = \sum_{i=1}^{D} \log \sum_y P(y, x_i) \]

\[ \mathcal{L}(x_1, \ldots, D; \theta) \]

- Initialize probabilities \( \theta \)
- \textbf{repeat}
  - Compute expected counts \( e \)
  - Fit parameters \( \theta \)
- \textbf{until} convergence

slide credit: Taylor Berg-Kirkpatrick
EM’s Lower Bound

\[ \mathcal{L}(x_1,\ldots,D) = \sum_{i=1}^{D} \log \sum_{y} P(y,x_i) \]

Initialize probabilities \( \theta \)

repeat

- Compute expected counts \( e \)
- Fit parameters \( \theta \)

until convergence

slide credit: Taylor Berg-Kirkpatrick
Part-of-speech Induction

- Merialdo (1994): you have a whitelist of tags for each word
- Learn parameters on $k$ examples to start, use those to initialize EM, run on 1 million words of unlabeled data
- Tag dictionary + data should get us started in the right direction...
Part-of-speech Induction

- Small amounts of data > large amounts of unlabeled data
- Running EM *hurts* performance once you have labeled data

<table>
<thead>
<tr>
<th>Iter</th>
<th>Correct tags (% words) after ML on 1M words</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>77.0</td>
</tr>
<tr>
<td>1</td>
<td>80.5</td>
</tr>
<tr>
<td>2</td>
<td>81.8</td>
</tr>
<tr>
<td>3</td>
<td>83.0</td>
</tr>
<tr>
<td>4</td>
<td>84.0</td>
</tr>
<tr>
<td>5</td>
<td>84.8</td>
</tr>
<tr>
<td>6</td>
<td>85.3</td>
</tr>
<tr>
<td>7</td>
<td>85.8</td>
</tr>
<tr>
<td>8</td>
<td>86.1</td>
</tr>
<tr>
<td>9</td>
<td>86.3</td>
</tr>
<tr>
<td>10</td>
<td>86.6</td>
</tr>
</tbody>
</table>

Merialdo (1994)
Does unsupervised learning help?

- Sometimes can produce good representations: Stallard et al. (2012) shows that unsupervised morphological segmentation can work as well as supervised segmentation for Arabic machine translation.

- Later in the course: word embeddings produced from “naturally supervised” data.
EM with Features

- Can use more sophisticated forms of $P(x,y)$

- Idea: still a generative model, but instead of distributions being multinomials, have them be log-linear models

\[
P(x_i | y_i) = \frac{\exp(w^\top f(x_i, y_i))}{\sum_x \exp(w^\top f(x, y_i))}
\]

  - normalized over all words

- Still a generative model but local arcs are parameterized in a log-linear way

- CRFs don’t have this local parameterization

- Features can only look at current word and tag!

Berg-Kirkpatrick et al. (2010)
**EM with Features**

Key distribution: \( P(x | \text{NNP}) \)

| \( \theta_x | \text{NNP} \) | \( x \) | \( f \) | \( e^{w^T f} \) |
|---|---|---|---|
| 0.1 | John | +Cap | 0.3 |
| 0.0 | Mary | +Cap | 0.3 |
| 0.2 | running | +ing | 0.1 |
| 0.0 | jumping | +ing | 0.1 |

\( w: \)
- +Cap: +1.2
- +ing: -0.3

Berg-Kirkpatrick et al. (2010)
**EM with Features**

\[ P(x_i|y_i) = \frac{\exp(w^\top f(x_i, y_i))}{\sum_x \exp(w^\top f(x, y_i))} \]

- **Learning:**
  - E-step is the same
  - M-step now requires gradients (slightly different than CRF gradients due to local normalization)
- One approach: can run gradient to completion each M-step (i.e., fully fit the fractional annotations we have)
EM with Features

Initialize weights $w$

repeat

- Compute expected counts $e$

  repeat

  - Compute $\ell(w, e)$
  - Compute $\nabla \ell(w, e)$
  - $w \leftarrow \text{climb}(w, \ell(w, e), \nabla \ell(w, e))$

until convergence

- Transform $w$ to $\theta$

until convergence

Berg-Kirkpatrick et al. (2010)
EM with Features

- Initialize weights \( w \)
- \textbf{repeat}
  - Compute expected counts \( e \)
    - \textbf{repeat}
      - Compute \( \ell(w, e) \)
      - Compute \( \nabla \ell(w, e) \)
      - \( w \leftarrow \text{climb}(w, \ell(w, e), \nabla \ell(w, e)) \)
    - \textbf{until} convergence
  - Transform \( w \) to \( \theta \)
- \textbf{until} convergence

Berg-Kirkpatrick et al. (2010)
EM with Features

Initialize weights $w$

repeat
  Compute expected counts $e$
  repeat
    Compute $\ell(w, e)$
    Compute $\nabla \ell(w, e)$
    $w \leftarrow \text{climb}(w, \ell(w, e), \nabla \ell(w, e))$
  until convergence
  Transform $w$ to $\theta$
until convergence

Berg-Kirkpatrick et al. (2010)
EM with Features

Initialize weights $w$

repeat
  Compute expected counts $e$
  repeat
    Compute $\ell(w, e)$
    Compute $\nabla\ell(w, e)$
    $w \leftarrow \text{climb}(w, \ell(w, e), \nabla\ell(w, e))$
  until convergence
until convergence

Transform $w$ to $\theta$
until convergence

Berg-Kirkpatrick et al. (2010)
EM with Features

Initialize weights $w$

repeat

- Compute expected counts $e$

repeat

- Compute $\ell(w, e)$
- Compute $\nabla \ell(w, e)$
- $w \leftarrow \text{climb}(w, \ell(w, e), \nabla \ell(w, e))$

until convergence

- Transform $w$ to $\theta$

until convergence

Berg-Kirkpatrick et al. (2010)
EM with Features

Initialize weights $w$

repeat
  Compute expected counts $e$
  repeat
    Compute $\ell(w, e)$
    Compute $\nabla\ell(w, e)$
    $w \leftarrow$ climb($w$, $\ell(w, e)$, $\nabla\ell(w, e)$)
  until convergence
until convergence

Transform $w$ to $\theta$

until convergence

Berg-Kirkpatrick et al. (2010)
EM with Features

- Faster approach: after the E-step, just take one gradient step

- “Direct gradient” on the marginal log likelihood $\log \sum_y P(x, y|\theta)$
EM with Features

- Initialize weights $w$
- repeat
  - Compute expected counts $e$
  - Compute $L(w)$
  - Compute $\nabla \ell(w, e)$
  - $w \leftarrow$ climb($w, L(w), \nabla \ell(w, e)$)
- Transform $w$ to $\theta$
- until convergence

Berg-Kirkpatrick et al. (2010)
EM with Features

Initialize weights $w$

repeat

- Compute expected counts $e$
- Compute $L(w)$
- Compute $\nabla \ell(w, e)$
- $w \leftarrow \text{climb}(w, L(w), \nabla \ell(w, e))$
- Transform $w$ to $\theta$

until convergence

Berg-Kirkpatrick et al. (2010)
EM with Features

Initialize weights $w$

repeat

- Compute expected counts $e$
- Compute $L(w)$
- Compute $\nabla \ell(w, e)$
- $w \leftarrow \text{climb}(w, L(w), \nabla \ell(w, e))$
- Transform $w$ to $\theta$

until convergence

Berg-Kirkpatrick et al. (2010)
EM with Features

Initialize weights $w$

repeat

- Compute expected counts $e$
- Compute $L(w)$
- Compute $\nabla \ell(w, e)$
- $w \leftarrow \text{climb}(w, L(w), \nabla \ell(w, e))$
- Transform $w$ to $\theta$

until convergence

Berg-Kirkpatrick et al. (2010)
EM with Features

Initialize weights $w$
repeat
  \begin{itemize}
  \item Compute expected counts $e$
  \item Compute $L(w)$
  \item Compute $\nabla \ell(w, e)$
  \item $w \leftarrow \text{climb}(w, L(w), \nabla \ell(w, e))$
  \item Transform $w$ to $\theta$
  \end{itemize}
until convergence

Berg-Kirkpatrick et al. (2010)
EM with Features

Initialize weights $w$

repeat

- Compute expected counts $e$
- Compute $L(w)$
- Compute $\nabla \ell(w, e)$
- $w \leftarrow \text{climb}(w, L(w), \nabla \ell(w, e))$
- Transform $w$ to $\theta$

until convergence

Berg-Kirkpatrick et al. (2010)
EM with Features

- Faster approach: after the E-step, just take *one* gradient step

- “Direct gradient” on the marginal log likelihood: \( \log \sum_y P(x, y|\theta) \)

- Looks a lot like CRF training: compute marginals, estimate gradient based on them
Evaluating Direct Gradient

- Different setup: don’t assume any initialization, you just have 45 tags that you need to learn

- “Many-to-one” accuracy: map each of your learned tags to its closest gold tag, evaluate how many words are tagged correctly
Evaluating Direct Gradient

Many-to-1 Accuracy

- HMM EM: 63.1
- HMM Features EM: 68.1
- HMM Features Gradient: 75.5

Features:

- Basic: John \( \wedge \) NNP
- Contains-Digit: +Digit \( \wedge \) NNP
- Contains-Hyphen: +Hyph \( \wedge \) NNP
- Initial-Capital: +Cap \( \wedge \) NNP
- Suffix: +ing \( \wedge \) NNP

Also strong results on grammar induction, word alignment, and morphological segmentation.
Takeaways

- EM sort of works for POS induction
- A supervised system on a little bit of labeled data gives better POS accuracy, but unsupervised learning can still learn useful representations for downstream tasks (like machine translation)
- EM isn’t restricted to multinomial distributions or ones with closed-form M-step updates: the M-step can be gradient ascent
- “Direct gradient” can work really well!
Next Time

- Constituency parsing
- In two lectures: dependency parsing (project 2)