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:

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

- Gradient: gold features - expected features under model

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

Structured SVM

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

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

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

\( \forall j \forall y \in \mathcal{Y} \) \( w^T f(x_j, y_j^*) \geq w^T f(x_j, y) + \ell(y, y_j^*) - \xi_j \)

- Loss-augmented decode can be done with Viterbi

- Only need Viterbi for inference here...hmm...
### Viterbi Time Complexity

- **VBD**
- **VBZ**
- **NNP**
- **NNS**

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

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

### Beam Search

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

### 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

Beam size of k, time complexity **O(nks log(k))**
This Lecture

- Unsupervised POS tagging
- EM for learning HMMs
- Gradient-based unsupervised learning
- (briefly) Some writing tips

Unsupervised Learning

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

- 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

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

- Exact equality:
- KL divergence: asymmetric measure of difference between two distributions
- Related to cross-entropy (= KL + entropy of \( q \))
- If \( q(y) = P(y|x, \theta) \), KL term is 0 so equality is achieved

**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**

- 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

Adapted from Leon Gu
EM for HMMs

- Recall how we maximized $\log P(x, y)$: read counts off data
  
  - $P(\text{the} | \text{DT}) = 1$
  - $P(\text{dog} | \text{DT}) = 0$
  - $P(\text{the} | \text{NN}) = 0$
  - $P(\text{dog} | \text{NN}) = 1$

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

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

- E-step 1: (all values are approximate)
  - $P(\text{the} | \text{DT}) = 0.95$
  - $P(\text{dog} | \text{DT}) = 0.05$
  - $P(\text{the} | \text{NN}) = 0.95$
  - $P(\text{dog} | \text{NN}) = 0.05$
  - Transition probabilities (approx): $P(\text{NN} | \text{DT}) = 3/4$, $P(\text{DT} | \text{NN}) = 1/4$

How does EM learn things?

- Same for transition probabilities

- E-step 1:
  - Emissions aren’t so different
  - Transition probabilities (approx): $P(\text{NN} | \text{DT}) = 3/4$, $P(\text{DT} | \text{NN}) = 1/4$
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(NN|DT) = 3/4, P(DT|DT) = 1/4

- M-step 2:
  - Emission P(marsupial|NN) > P(marsupial|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

$$L(x_1,...,D) = \sum_{i=1}^{D} \log \sum_{y} P(y, x_i)$$
EM’s Lower Bound

\[ \mathcal{L}(\mathbf{x}_1, \ldots, \mathbf{D}; \theta) = \sum_{i=1}^{D} \log \sum_{y} P(y, x_i) \]

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

- E-step: compute \( q \) which gives this lower bound

- M-step: find maximum of lower bound

Slide credit: Taylor Berg-Kirkpatrick
\[
\mathcal{L}(x_1, \ldots, D; \theta) = \sum_{i=1}^{D} \log \sum_{y} P(y, x_i)
\]

Initialize probabilities \( \theta \)

repeat

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

---

**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

<table>
<thead>
<tr>
<th>Iter</th>
<th>Correct tags (% words) after ML on 1M words</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>77.0 90.0 95.4 96.2 96.6 96.9 97.0</td>
</tr>
<tr>
<td>1</td>
<td>80.5 92.6 95.8 96.3 96.6 96.7 96.8</td>
</tr>
<tr>
<td>2</td>
<td>81.8 93.0 95.7 96.1 96.3 96.4 96.4</td>
</tr>
<tr>
<td>3</td>
<td>83.0 93.1 95.4 95.8 96.1 96.2 96.2</td>
</tr>
<tr>
<td>4</td>
<td>84.0 93.0 95.2 95.5 95.8 96.0 96.0</td>
</tr>
<tr>
<td>5</td>
<td>84.8 92.9 95.1 95.4 95.6 95.8 95.8</td>
</tr>
<tr>
<td>6</td>
<td>85.3 92.8 94.9 95.2 95.5 95.6 95.7</td>
</tr>
<tr>
<td>7</td>
<td>85.8 92.8 94.7 95.1 95.3 95.5 95.5</td>
</tr>
<tr>
<td>8</td>
<td>86.1 92.7 94.6 95.0 95.2 95.4 95.4</td>
</tr>
<tr>
<td>9</td>
<td>86.3 92.6 94.5 94.9 95.1 95.3 95.3</td>
</tr>
<tr>
<td>10</td>
<td>86.6 92.6 94.4 94.8 95.0 95.2 95.2</td>
</tr>
</tbody>
</table>

Meritaldo (1994)

Does unsupervised learning help?

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

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

$$P(x_i|y_i) = \frac{\exp(w^T f(x_i, y_i))}{\sum_x \exp(w^T f(x, y_i))}$$

- Features can only look at current word and tag!
- 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

EM with Features

- Key distribution: $P(x|\text{NNP})$
- Idea: still a generative model, but instead of distributions being multinomials, have them be log-linear models

$$P(x_i|y_i) = \frac{\exp(w^T f(x_i, y_i))}{\sum_x \exp(w^T f(x, y_i))}$$

- Features can only look at current word and tag!
- 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

CRFs don’t have this local parameterization

- 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

$$P(x_i|y_i) = \frac{\exp(w^T f(x_i, y_i))}{\sum_x \exp(w^T f(x, y_i))}$$

- Features can only look at current word and tag!
- 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

EM with Features

- Key distribution: $P(x|\text{NNP})$
- Idea: still a generative model, but instead of distributions being multinomials, have them be log-linear models

$$P(x_i|y_i) = \frac{\exp(w^T f(x_i, y_i))}{\sum_x \exp(w^T f(x, y_i))}$$

- Features can only look at current word and tag!
- 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

CRFs don’t have this local parameterization
**EM with Features**

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

- **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)

---

**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**
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

- 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

Berg-Kirkpatrick et al. (2010)

EM with Features

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

Berg-Kirkpatrick et al. (2010)

Evalutaing Direct Gradient

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

#### Many-to-1 Accuracy

<table>
<thead>
<tr>
<th></th>
<th>HMM EM</th>
<th>HMM Features EM</th>
<th>HMM Features Gradient</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic</td>
<td>63.1</td>
<td>68.1</td>
<td>75.5</td>
</tr>
<tr>
<td>Contains-Digit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Contains-Hyphen</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Initial-Capital</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Suffix</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- 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)