What data do we learn from?

- Supervised settings:
  - Tagging: POS, NER
  - Parsing: constituency, dependency, semantic parsing
  - IE, MT, QA, ...
- Semi-supervised models
  - Word embeddings / word clusters (helpful for nearly all tasks)
  - Language models for machine translation
  - Learn linguistic structure from unlabeled data and use it?

This Lecture

- Discrete linguistic structure from generative models: unsupervised POS induction
  - Expectation maximization for learning HMMs
- Continuous structure with generative models: variational autoencoders
- Continuous structure with “discriminative” models: transfer learning

EM for HMMs
Recall: Hidden Markov Models

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

- Observation ($x$) depends only on current state ($y$)
- Multinomials: tag x tag transitions, tag x word emissions
- $P(x|y)$ is a distribution over all words in the vocabulary — not a distribution over features (but could be!)

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 $x = (x_1, \ldots, x_n)$  
  Output $y = (y_1, \ldots, y_n)$

- Assume we don’t have access to labeled examples — how can we learn a POS tagger?
- Key idea: optimize $P(x) = \sum_{y} P(y, x)$
  Generative model explains the data $x$; the right HMM makes it look likely

- Optimizing marginal log-likelihood with no labels $y$:
  $\mathcal{L}(x_1, \ldots, D) = \sum_{i=1}^{D} \log \sum_{y} P(y, x_i)$
  non-convex optimization problem

Part-of-Speech Induction

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

- Optimizing marginal log-likelihood with no labels $y$:
  $\mathcal{L}(x_1, \ldots, D) = \sum_{i=1}^{D} \log \sum_{y} P(y, x_i)$

- 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

\[
\begin{align*}
\log \sum_y P(x, y|\theta) & \quad \text{Condition on parameters } \theta \\
= \log \sum_y q(y) \frac{P(x, y|\theta)}{q(y)} & \quad \text{Variational approximation } q \text{ — this is a trick we’ll return to later!} \\
\geq \sum_y q(y) \log \frac{P(x, y|\theta)}{q(y)} & \quad \text{Jensen’s inequality (uses concavity of log)} \\
= \mathbb{E}_{q(y)} \log P(x, y|\theta) + \text{Entropy}[q(y)]
\end{align*}
\]

- Can optimize this lower-bound on log likelihood instead of log-likelihood

Adapted from Leon Gu

Expectation Maximization

\[
\begin{align*}
\log \sum_y P(x, y|\theta) & \geq \mathbb{E}_{q(y)} \log P(x, y|\theta) + \text{Entropy}[q(y)] \\
\text{If } q(y) = P(y|x, \theta), \text{ this bound ends up being tight} \\
\text{Expectation-maximization: alternating maximization of the lower bound over } q \text{ and } \theta \\
\text{Current timestep } = t, \text{ have parameters } \theta^{t-1} \\
\text{E-step: maximize w.r.t. } q; \text{ that is, } q^t = P(y|x, \theta^{t-1}) \\
\text{M-step: maximize w.r.t. } \theta; \text{ that is, } \theta^t = \arg\max_q \mathbb{E}_{q^t} \log P(x, y|\theta)
\end{align*}
\]

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, \( \theta^t = \arg\max_q \mathbb{E}_{q^t} \log P(x, y|\theta) \)

- 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: set parameters to optimize the crazy argmax term

M-Step

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

\[
\begin{align*}
\text{DT} & \quad \text{NN} \quad \text{the} \quad \text{dog} \\
\text{count(DT, the)} & = 1 \quad \text{count(DT, dog)} = 0 \quad \text{P(the | DT)} = 1 \quad \text{P(dog | DT)} = 0 \\
\text{count(NN, the)} & = 0 \quad \text{count(NN, dog)} = 1 \quad \text{P(the | NN)} = 0 \quad \text{P(dog | NN)} = 1
\end{align*}
\]

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

\[
\begin{align*}
\text{DT: 0.9} & \quad \text{DT: 0.3} \quad \text{NN: 0.1} \quad \text{NN: 0.7} \quad \text{the} \quad \text{dog} \\
\text{count(DT, the)} & = 0.9 \quad \text{count(DT, dog)} = 0.3 \quad \text{P(the | DT)} = 0.75 \quad \text{P(dog | DT)} = 0.25 \\
\text{count(NN, the)} & = 0.1 \quad \text{count(NN, dog)} = 0.7 \quad \text{P(the | NN)} = 0.125 \quad \text{P(dog | NN)} = 0.875
\end{align*}
\]
How does EM learn things?

- E-step 1:
  - **DT**: 0.95, **NN**: 0.05
  - **DT**: 0.95, **NN**: 0.05
  - **the**: 0.05, **dog**: 0.95
  - **the**: 0.05, **marsupial**: 0.95

- 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, **NN**: 0.05
  - **DT**: 0.95, **NN**: 0.05
  - **the**: 0.05, **dog**: 0.95
  - **the**: 0.05, **marsupial**: 0.70

- M-step 1:
  - Emissions aren’t so different
  - Transition probabilities (approx): P(NN|DT) = 3/4, P(DT|DT) = 1/4

---

M-Step

- Same for transition probabilities
- Emissions:
  - P(DT|DT) = 1/7
  - P(NN|DT) = 6/7
  - P(DT|NN) = 2/3
  - P(NN|NN) = 1/3

- Transition probabilities: uniform
- E-step 1 (all values are approximate)
How does EM learn things?

- E-step 2:
  - DT: 0.95 DT: 0.05
  - NN: 0.05 **NN: 0.95**

- M-step 2:
  - DT: 0.05 **NN: 0.70**
  - DT: 0.30

- 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

- 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

---

**EM’s Lower Bound**

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

- **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**

---

**EM’s Lower Bound**

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

- **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: compute $q$ which gives this lower bound**

---

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

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

$L(x_1,\ldots,D; \theta)$

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

M-step: find maximum of lower bound

E-step 2: re-estimate $q$

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

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

**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...

<table>
<thead>
<tr>
<th>Number of tagged sentences used for the initial model</th>
<th>Correct tags (% words) after ML on 1M words</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iter</td>
<td>0</td>
</tr>
<tr>
<td>------</td>
<td>---</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>

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

**Merialdo (1994) Slide credit:** Taylor Berg-Kirkpatrick
### Two Hours of Annotation

<table>
<thead>
<tr>
<th>Human Annotations</th>
<th>0. No EM</th>
<th>1. EM only</th>
<th>2. With LP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>T</td>
<td>K</td>
<td>U</td>
</tr>
<tr>
<td>Initial data</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KIN tokens A</td>
<td>72</td>
<td>90</td>
<td>58</td>
</tr>
<tr>
<td>KIN types A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MLG tokens B</td>
<td>74</td>
<td>89</td>
<td>49</td>
</tr>
<tr>
<td>MLG types B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENG tokens A</td>
<td>63</td>
<td>83</td>
<td>38</td>
</tr>
<tr>
<td>ENG types A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENG tokens B</td>
<td>70</td>
<td>87</td>
<td>44</td>
</tr>
<tr>
<td>ENG types B</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Kinyarwanda and Malagasy (two actual low-resource languages)
- Label propagation (technique for using dictionary labels) helps a lot, with data that was collected in two hours

Garrette and Baldridge (2013)

### Variational Autoencoders

- What if we want to use continuous latent variables?

$$P(z, x) = P(z)P(x|z)$$

$$P(x) = \int P(z)P(x|z)dz$$

- Can use EM here when $P(z)$ and $P(x|z)$ are Gaussians
- What if we want $P(x|z)$ to be something more complicated, like an LSTM with $z$ as the initial state?

### Deep Generative Models

- $z$ is a latent variable which should control the generation of the sentence, maybe capture something about its topic
Deep Generative Models

\[
\log \int_z P(x, z|\theta) = \log \int_z q(z) \frac{P(x, z|\theta)}{q(z)} \geq \int_z q(z) \log \frac{P(x, z|\theta)}{q(z)}
\]

Jensen

\[
= \mathbb{E}_{q(z|x)}[-\log q(z|x) + \log P(x, z|\theta)]
\]

\[
= \mathbb{E}_{q(z|x)}[\log P(x|z, \theta)] - KL(q(z|x)\|P(z))
\]

“make the data likely under q” “make q close to the prior” (discriminative)

KL divergence: distance metric over distributions (more dissimilar <=> higher KL)

Variational Autoencoders

\[
\mathbb{E}_{q(z|x)}[\log P(x|z, \theta)] - KL(q(z|x)\|P(z))
\]

Generative model (test): Autoencoder (training):

\[ z \sim P(z) \]

\[ \text{"inference network"} \]

\[ x \]

\[ q(z|x) \]

distribution over z

\[ \text{generative model} \]

\[ x \]

Maximize \( P(x|z, \theta) \)

Miao et al. (2015)

Training VAEs

\[
\mathbb{E}_{q(z|x)}[\log P(x|z, \theta)] - KL(q(z|x)\|P(z))
\]

Choose \( q \) to be Gaussian with parameters that are computed from \( x \)

\[ q = N(\mu(x), \text{diag}(\sigma^2(x))) \]

mu and sigma are computed from an LSTM over \( x \), call their parameters \( \phi \)

Autoencoder (training):

For each example \( x \)

Compute \( q \) (run forward pass to compute mu and sigma)

For some number of samples

Sample \( z \sim q \)

Compute \( P(x|z) \) and compute loss

Backpropagate to update \( \phi, \theta \)

Miao et al. (2015)
Autoencoders

The movie was good [STOP]

‣ Another interpretation: train an autoencoder and add Gaussian noise
‣ Same computation graph as VAE, add KL divergence term to make the objective the same
‣ Inference network (q) is the encoder and generator is the decoder

What do VAEs do?

‣ Let us encode a sentence and generate similar sentences:

<table>
<thead>
<tr>
<th>INPUT</th>
<th>MEAN</th>
<th>SAMP. 1</th>
<th>SAMP. 2</th>
<th>SAMP. 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>we looked out at the setting sun.</td>
<td>they were laughing at the same time.</td>
<td>i went to the kitchen.</td>
<td>i went to my apartment.</td>
<td>i looked up at the blue sky.</td>
</tr>
<tr>
<td>i went to the kitchen.</td>
<td>i went to my apartment.</td>
<td>what are you doing?</td>
<td>what are you doing?</td>
<td>i looked around the room.</td>
</tr>
<tr>
<td>how are you doing?</td>
<td>what are you doing?</td>
<td>* are you sure?</td>
<td>* are you sure?</td>
<td>i turned back to the table.</td>
</tr>
<tr>
<td>it was down on the dance floor.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

‣ Style transfer: also condition on sentiment, change sentiment
‣ ...or use the latent representations for semi-supervised learning

Self-Supervision / Transfer Learning

Visualizaion

\[ \mathbb{E}_{q(z|x)} \left[ \log P(x|z, \theta) \right] + \text{KL}(q(z|x) \| P(z)) \]

‣ What does gradient encourage latent space to do?

Prior direction of better likelihood for x

Bowman et al. (2016), Zhao et al. (2017)
Goals of Unsupervised Learning

- We want to use unlabeled data, but EM “requires” generative models. Are models like this really necessary?
- word2vec: predict nearby word given context. This wasn’t generative, but the supervision is free...
- Language modeling is a “more contextualized” form of word2vec

ELMo

- Generative model of the data!
- Train one model in each direction on 1B words, use the LSTM hidden states as context-aware token representations

BERT

- Text “infilling” task: replace 15% of tokens with something else and try to predict the original
  - 80% of the time: MASK; 10%: random word; 10%: keep same
- I went to the store and bought a gallon of milk. My favorite kind is 2%.

Results

- Dramatic gains on a range of sentence pair / single sentence tasks: paraphrase identification, entailment, sentiment, textual similarity, ...
- Not a generative model! But learns really effective representations...
<table>
<thead>
<tr>
<th><strong>Unsupervised Learning</strong></th>
<th><strong>Takeaways</strong></th>
</tr>
</thead>
</table>
| ‣ Discrete linguistic structure with generative models: unsupervised POS induction  
  ‣ These models are hard to learn in an unsupervised way and too impoverished to really be all that useful  
| ‣ Continuous structure with generative models: variational autoencoders  
  ‣ Useful, but also hard to learn in practice  
| ‣ Continuous structure with “discriminative” models  
  ‣ ELMo / BERT seem extremely useful  
| **Takeaways** |
| ‣ EM sort of works for POS induction  
| ‣ VAE can learn sentence representations  
| ‣ Language modeling or text infilling as pretraining seems best — arguably not “unsupervised” but the annotation is free  
| ‣ Using unlabeled data effectively seems like one of the most important directions in NLP right now  
| ‣ Next time: Jessy Li guest lecture on discourse  |