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
- $P(x \mid y)$ is a distribution over all words in the vocabulary — not a distribution over features (but could be!)

\[
P(y, x) = P(y_1) \prod_{i=2}^{n} P(y_i \mid y_{i-1}) \prod_{i=1}^{n} P(x_i \mid y_i)
\]

- Initial distribution
- Transition probabilities
- Emission probabilities
Unsupervised Learning

- Can we induce linguistic structure? Thought experiment...

  a b a c c c c
  b a c 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 $P(x) = \sum_y P(y, x)$ Generator model explains the data $x$; the right HMM makes it look likely

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

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

  $\triangleright$ non-convex optimization problem
Part-of-Speech Induction

- **Input** $\mathbf{x} = (x_1, ..., x_n)$  
  **Output** $\mathbf{y} = (y_1, ..., y_n)$

- Optimizing marginal log-likelihood with no labels $\mathbf{y}$:

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

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

- What’s the point of this? Model has inductive bias and so should learn some useful latent structure $\mathbf{y}$ (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

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

- If \( q(y) = P(y | x, \theta) \), this bound ends up being tight

- 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, \( \theta^t = \arg\max_{\theta} \mathbb{E}_{q^t} \log P(x, y | \theta) \)
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: set parameters to optimize the crazy argmax term
M-Step

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

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

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

  \[
  \begin{align*}
  q & & q \\
  \text{DT: 0.9} & & \text{DT: 0.3} \\
  \text{NN: 0.1} & & \text{NN: 0.7} \\
  \text{the} & & \text{dog} \\
  \text{count}(\text{DT, the}) &= 0.9 & P(\text{the} | \text{DT}) &= 0.75 \\
  \text{count}(\text{DT, dog}) &= 0.3 & P(\text{dog} | \text{DT}) &= 0.25 \\
  \text{count}(\text{NN, the}) &= 0.1 & P(\text{the} | \text{NN}) &= 0.125 \\
  \text{count}(\text{NN, dog}) &= 0.7 & P(\text{dog} | \text{NN}) &= 0.875
  \end{align*}
  \]
M-Step

- Same for transition probabilities
  - $q$
    - DT—NN: 0.6
    - DT—DT: 0.1
    - NN—DT: 0.2
    - NN—NN: 0.1
  - $P(DT|DT) = 1/7$
  - $P(NN|DT) = 6/7$
  - $P(DT|NN) = 2/3$
  - $P(NN|NN) = 1/3$
How does EM learn things?

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

  - Transition probabilities: uniform

- E-step 1: (all values are approximate)
  
  - Emissions
    
    - $\text{DT}: 0.95$
    - $\text{NN}: 0.05$
    - $\text{the}$
    - $\text{dog}$
    - $\text{DT}: 0.95$
    - $\text{NN}: 0.95$
    - $\text{the}$
    - $\text{marsupial}$
    - $\text{DT}: 0.05$
    - $\text{NN}: 0.5$
    - 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:**
  - \textbf{DT: 0.95} DT: 0.05 \quad \textbf{DT: 0.95} DT: 0.30
  - \textbf{NN: 0.95} NN: 0.05 \quad \textbf{NN: 0.70} NN: 0.05
  - \textit{the} \quad \textit{dog} \quad \textit{the} \quad \textit{marsupial}

- **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

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

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

Initialize probabilities \( \theta \)

repeat

- Compute expected counts \( \mathbf{e} \)
- Fit parameters \( \theta \)

until convergence
EM’s Lower Bound

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

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

**Initialize probabilities \( \theta \)**

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

**until** convergence

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

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

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

[Diagram]

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

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

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

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

- 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

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

- 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

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

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

slide credit: Taylor Berg-Kirkpatrick
Mericaldo (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: Number of tagged sentences used for the initial model

<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></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)
Two Hours of Annotation

- 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)

<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>KIN tokens A</td>
<td>72 90 58</td>
<td>55 82 32</td>
<td>71 86 58</td>
</tr>
<tr>
<td>KIN types A</td>
<td></td>
<td>63 77 32</td>
<td>78 83 69</td>
</tr>
<tr>
<td>MLG tokens B</td>
<td>74 89 49</td>
<td>68 87 39</td>
<td>74 89 49</td>
</tr>
<tr>
<td>MLG types B</td>
<td></td>
<td>71 87 46</td>
<td>72 81 57</td>
</tr>
<tr>
<td>ENG tokens A</td>
<td>63 83 38</td>
<td>62 83 37</td>
<td>72 85 55</td>
</tr>
<tr>
<td>ENG types A</td>
<td></td>
<td>66 76 37</td>
<td>75 81 56</td>
</tr>
<tr>
<td>ENG tokens B</td>
<td>70 87 44</td>
<td>70 87 43</td>
<td>78 90 60</td>
</tr>
<tr>
<td>ENG types B</td>
<td></td>
<td>69 83 38</td>
<td>75 82 61</td>
</tr>
</tbody>
</table>
Variational Autoencoders
Continuous Latent Variables

- For discrete latent variables $y$, we optimized: $P(x) = \sum_y P(y, x)$

- 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

\[ P(z, x) = P(z)P(x|z) \]

- \( z \) is a latent variable which should control the generation of the sentence, maybe capture something about its topic.

the movie was good [STOP]
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

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

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

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

- KL divergence: distance metric over distributions (more dissimilar \(\Rightarrow\) higher KL)
Variational Autoencoders

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

Generative model (test):

\[ z \sim P(z) \]

Autoencoder (training):

\[ x \quad \text{Input} \]
\[ \text{"inference network"} \]
\[ q(z|x) \quad \text{distribution over } z \]
\[ \text{generative model} \]
\[ x \quad \text{Maximize } P(x|z, \theta) \]

Miao et al. (2015)
Training VAEs

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

- Choose \( q \) to be Gaussian with parameters that are computed from \( x \)
  \[ q = \mathcal{N}(\mu(x), \text{diag}(\sigma^2(x))) \]
  - \( \mu \) and \( \sigma \) are computed from an LSTM over \( x \), call their parameters \( \phi \)

- How to handle the expectation? Sampling

Autoencoder (training):

```
\[
\begin{array}{c}
\text{x} \\
\phi \\
\theta \\
\text{q(z|x)} \\
\text{generative model} \\
\text{x}
\end{array}
\]
```

Miao et al. (2015)
Training VAEs

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

Autoencoder (training):

For each example $x$, compute $q(z|x)$:

```
φ → "inference network"

θ → generative model
```

$x$
Autoencoders

- 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

Inference network (q) is the encoder and generator is the decoder

Gaussian noise

the movie was great + <s> the movie was good [STOP]
What does gradient encourage latent space to do?

\[ \mathbb{E}_{q(z|x)} \left[ \log P(x|z, \theta) \right] + \text{KL}(q(z|x) \| P(z)) \]
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>
<th>i went to the kitchen</th>
<th>how are you doing ?</th>
<th>i went to the kitchen</th>
<th>what are you doing ?</th>
<th>“are you sure ?”</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'll see you in the early morning .</td>
<td>i looked up at the blue sky .</td>
<td>it was down on the dance floor .</td>
<td>i went to the kitchen</td>
<td>how are you doing ?</td>
<td>i went to my apartment .</td>
<td>what are you doing ?</td>
<td>“are you sure ?”</td>
</tr>
</tbody>
</table>

- Style transfer: also condition on sentiment, change sentiment

- ...or use the latent representations for semi-supervised learning

Positive

⇒ ARAE great indoor mall .
⇒ Cross-AE no smoking mall .
⇒ Terrible outdoor urine .

Positive

⇒ ARAE it has a great atmosphere , with wonderful service .
⇒ Cross-AE it has no taste , with a complete jerk .
⇒ Cross-AE it has a great horrible food and run out service .

Bowman et al. (2016), Zhao et al. (2017)
Self-Supervision / Transfer Learning
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

\[ P(x_i | x_1, \ldots, x_{i-1}) = \text{LSTM}(x_1, \ldots, x_{i-1}) \]

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

learn a linear classifier on top of this vector to get a POS tagger with 97.3% accuracy (~SOTA)
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 \textit{store} and bought \textit{a} gallon of \textit{milk}. My \textit{favorite} kind is 2%.

Transformer (12-24 layers)

I went to the \textit{MASK} and bought \textit{MASK} gallon of \textit{dog}. My \textit{MASK} kind is 2%.

- Also generate “fake” sentence pairs and try to predict real from fake

I went to the \textit{MASK} and bought \textit{MASK} gallon of \textit{dog}. \textit{I love karaoke}!
## Results

<table>
<thead>
<tr>
<th>System</th>
<th>MNLI-(m/mm)</th>
<th>QQP</th>
<th>QNLI</th>
<th>SST-2</th>
<th>CoLA</th>
<th>STS-B</th>
<th>MRPC</th>
<th>RTE</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>392k</td>
<td>363k</td>
<td>108k</td>
<td>67k</td>
<td>8.5k</td>
<td>5.7k</td>
<td>3.5k</td>
<td>2.5k</td>
<td></td>
</tr>
<tr>
<td>Pre-OpenAI SOTA</td>
<td>80.6/80.1</td>
<td>66.1</td>
<td>82.3</td>
<td>93.2</td>
<td>35.0</td>
<td>81.0</td>
<td>86.0</td>
<td>61.7</td>
<td>74.0</td>
</tr>
<tr>
<td>BiLSTM+ELMo+Attn</td>
<td>76.4/76.1</td>
<td>64.8</td>
<td>79.9</td>
<td>90.4</td>
<td>36.0</td>
<td>73.3</td>
<td>84.9</td>
<td>56.8</td>
<td>71.0</td>
</tr>
<tr>
<td>OpenAI GPT</td>
<td>82.1/81.4</td>
<td>70.3</td>
<td>88.1</td>
<td>91.3</td>
<td>45.4</td>
<td>80.0</td>
<td>82.3</td>
<td>56.0</td>
<td>75.2</td>
</tr>
<tr>
<td>BERT\text{BASE}</td>
<td>84.6/83.4</td>
<td>71.2</td>
<td>90.1</td>
<td>93.5</td>
<td>52.1</td>
<td>85.8</td>
<td>88.9</td>
<td>66.4</td>
<td>79.6</td>
</tr>
<tr>
<td>BERT\text{LARGE}</td>
<td>86.7/85.9</td>
<td>72.1</td>
<td>91.1</td>
<td>94.9</td>
<td>60.5</td>
<td>86.5</td>
<td>89.3</td>
<td>70.1</td>
<td>81.9</td>
</tr>
</tbody>
</table>

- 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...
Unsupervised Learning

- 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