Training and Inference Methods over Translation Forests

Zhifei Li
Center for Language and Speech Processing
Computer Science Department
Johns Hopkins University

Joint work with Sanjeev Khudanpur, Jason Eisner, Chris Callison-Burch, and many other team members in the Joshua project
Statistical Machine Translation Pipeline

- **Bilingual Data**
- **Monolingual English**
- **Translation Model**
- **Language Model**
- **MT Decoder**
- **Training**
- **Optimal Weights**
- **Unseen Sentences**
- **Translation Outputs**
垫子上的猫
dianzi shang de mao
a cat on the mat

zhongguo de shoudu
capital of China

wo de mao
my cat

zhifei de mao
zhifei's cat

X→⟨X₀ de X₁, X₁ on X₀⟩
X→⟨X₀ de X₁, X₁ of X₀⟩
X→⟨X₀ de X₁, X₀ X₁⟩
X→⟨X₀ de X₁, X₀ 's X₁⟩
Joshua
(chart parser)

S → ⟨ X₀, X₀ ⟩

X → ⟨ X₀ de X₁, X₀ 's X₁ ⟩

X → ⟨ mao, a cat ⟩

X → ⟨ dianzi shang, the mat ⟩
$S \rightarrow \langle X_0, X_0 \rangle$

$X \rightarrow \langle X_0 \text{ de } X_1, X_1 \text{ on } X_0 \rangle$

$X \rightarrow \langle \text{dianzi shang, the mat} \rangle$

$X \rightarrow \langle \text{mao, a cat} \rangle$

dianzi$_0$ shang$_1$ de$_2$ mao$_3$

**a cat on the mat**

$S \rightarrow \langle X_0, X_0 \rangle$

$X \rightarrow \langle X_0 \text{ de } X_1, X_0 \text{ of } X_0 \rangle$

$X \rightarrow \langle \text{dianzi shang, the mat} \rangle$

$X \rightarrow \langle \text{mao, a cat} \rangle$

dianzi$_0$ shang$_1$ de$_2$ mao$_3$

**a cat of the mat**

$S \rightarrow \langle X_0, X_0 \rangle$

$X \rightarrow \langle X_0 \text{ de } X_1, X_0 \text{'s } X_1 \rangle$

$X \rightarrow \langle \text{dianzi shang, the mat} \rangle$

$X \rightarrow \langle \text{mao, a cat} \rangle$

dianzi$_0$ shang$_1$ de$_2$ mao$_3$

**the mat 's a cat**

**Joshua**

*(chart parser)*

**dianzi shang de mao**
hypergraph

Joshua (chart parser)

dianzi shang de mao
A hypergraph is a compact data structure to encode \textbf{exponentially many trees}.
A hypergraph is a compact data structure to encode **exponentially many trees**.
A hypergraph is a compact data structure to encode exponentially many trees.
A hypergraph is a compact data structure to encode exponentially many trees.
A hypergraph is a compact data structure to encode \textit{exponentially many trees}. 
Why hypergraphs?

• Contains a much larger hypothesis space than an n-best list

• General compact data structure
  • special cases include lattice, finite state machine, and packed forest
  • can be used for speech, monolingual parsing, tree-based MT systems, and many more
Linear model:

$$p(d \mid x) = \theta \cdot \Phi(d, x)$$

- **weights**
- **features**

**Weighted Hypergraph**

- $S \rightarrow \langle X_0, X_0 \rangle$
- $X \rightarrow \langle X_0, X_0 \rangle$
- $X \rightarrow \langle X_0 \text{ de } X_1, X_0 \text{ on } X_0 \rangle$
- $X \rightarrow \langle X_0 \text{ de } X_1, X_0 \text{'s } X_1 \rangle$
- $X \rightarrow \langle \text{dianzi shang, the mat} \rangle$
- $X \rightarrow \langle \text{mao, a cat} \rangle$
- $p=1$
- $p=2$
- $p=3$

- **the mat 's a cat**
- **a cat of the mat**
- **a cat on the mat**
Log-linear model:

\[ p(d \mid x) = \frac{e^{\theta \cdot \Phi(d, x)}}{Z(x)} \]

\[ Z = 2 + 1 + 3 + 2 = 8 \]

\[ p = \frac{2}{8} \]

\[ p = \frac{3}{8} \]

\[ p = \frac{1}{8} \]

\[ p = \frac{2}{8} \]
The hypergraph defines a probability distribution over trees!

the distribution is parameterized by $\Theta$

---

**a cat on the mat**

**dianzi shang maoli de mao**

**dianzi shang de mao**

**S → ⟨X₀, X₀⟩**

**X → ⟨X₀ de X₁, X₀ of X₀⟩**

**X → ⟨dianzi shang, the mat⟩**

**X → ⟨mao, a cat⟩**

**p = 1/8**

**p = 2/8**

**p = 3/8**

**a cat of the mat**

**the mat's a cat**

**the mat a cat**
The hypergraph defines a probability distribution over trees! the distribution is parameterized by $\Theta$

What criterion do we use to set the parameters $\Theta$?

Training

Given a fixed $\Theta$ and a hypergraph,  
- how to compute the posterior of a hyperedge?
  
- how to choose a desired translation output?

Inference

Why are the problems difficult?

- brute-force will be too slow as there are exponentially many trees, so require sophisticated dynamic programs
  
- sometimes intractable, require approximations
Outline

• Hypergraph as hypothesis space

• **Training methods**  (Li and Eisner, 2009)

• Exact and approximate inference
  
  ‣ exact inference
    
    ★ semiring framework  (Li and Eisner, 2009)
  
  ‣ approximate inference
    
    ★ “principled” ones
      
      ✓ variational decoding  (Li et. al, 2009)
      
      ✓ sampling and message passing methods
    
    ★ heuristic-based ones
      
      ✓ e.g., cube-pruning  (Chiang, 2007)

• **Joshua project**  (Li et. al, 2009)
Motivation for Training

• Towards a training method that can
  • dealing with millions of features
  • incorporate linguistics or semantic information
  • incorporate non-local dependency
  • perform reliable training
Training Setup

• Each **training example** consists of
  • a hypergraph representing hypothesized translations
  • a reference translation

\[ \text{x: dianzi shang de mao} \]

\[ \text{y*: a cat on the mat} \]

• Training
  • adjust the parameters \( \Theta \) so that the reference translation is preferred by the model
Training objectives

- Minimum error rate training (MERT)
- Minimum risk training
- **Minimum risk with deterministic annealing**
- Conditional random field (CRF)
- Perceptron
- MIRA
Minimum Error Rate Training

• Minimum Error Rate Training (MERT) (Och, 2003)

$$\theta^* = \arg \min_{\theta} \text{Loss}(\hat{y}, y^*)$$

$$- \text{BLEU}(y, y^*) \quad \hat{y} = \arg \max_y p_\theta(y | x)$$

(Papineni et al., 2001)

piecewise constant (not smoothed), not amenable to gradient descent, so cannot be scaled up to a large number of features

(Smith and Eisner, 2006)
Minimum Risk

- Minimum Error Rate Training (MERT) (Och, 2003)

$$\theta^* = \arg \min_{\theta} \text{Loss}(\hat{y}, y^*)$$

$$- \text{BLEU}(y, y^*)$$

$$\hat{y} = \arg \max_y p_{\theta}(y \mid x)$$

(Papineni et al., 2001)

- Minimum Risk Training

$$\theta^* = \arg \min_{\theta} \text{Risk}(\theta, y^*)$$

$$= \arg \min_{\theta} \sum_{y \in HG} p_{\theta}(y \mid x) \times \text{Loss}(y, y^*)$$

risk = expected loss = expected error rate
Minimum Risk with Deterministic Annealing

• Minimum risk objective

\[ \theta^* = \arg \min_{\theta} \text{Risk}(\theta, y^*) \]

suffer from local-minimum problem

• Minimum risk with deterministic annealing

\[ \theta^* = \arg \min_{\theta} \text{Risk}(\theta, y^*) - \text{Temperature} \times \text{Entropy}(p_\theta) \]

Smith and Eisner (2006) tried this on an n-best list
Conditional Random Field (CRF)

- Minimum risk objective
  \[ \theta^* = \arg\min_\theta \sum_{y \in HG} p_\theta(y|x) \times \text{Loss}(y, y^*) \]

- Conditional Random Field (CRF)
  - or maximum conditional likelihood (MLE)
    \[ \theta^* = \arg\max_\theta p_\theta(y^*|x) \]

CRF is minimum-risk training with a zero-one loss function!

\[ \text{Loss}(y, y^*) = \begin{cases} 0 & \text{if } y = y^* \\ 1 & \text{otherwise.} \end{cases} \]

\[ \text{Risk}(\theta, y^*) = \sum_{y \in HG \& y \neq y^*} p_\theta(y|x) = 1 - p_\theta(y^*|x) \]

no partial credit ☹
**Perceptron**

- Not explicitly optimize a certain objective function, rather it is a simple procedure

\[
\text{Perceptron}(x, \text{GEN}(x), y)
\]

1. \( \theta \leftarrow 0 \) \hspace{1cm} \triangleright \text{initialize as zero vector}  
2. \text{for } t \leftarrow 1 \text{ to } T  
3. \text{for } i \leftarrow 1 \text{ to } N  
4. \quad \hat{y} \leftarrow \arg \max_{y \in \text{GEN}(x_i)} \Phi(x_i, y) \cdot \theta  
5. \quad \text{if } (\hat{y} \neq y_i^*)  
6. \quad \quad \theta \leftarrow \theta + \Phi(x_i, y_i^*) - \Phi(x_i, \hat{y})  
7. \quad \text{return } \theta  

See Li and Khudanpur (2009) for its application in MT
Why Minimum Risk Training?

<table>
<thead>
<tr>
<th></th>
<th>Min-Risk</th>
<th>MERT</th>
<th>CRF</th>
<th>Perceptron</th>
<th>MIRA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Scalability</strong></td>
<td>☺</td>
<td>☞</td>
<td>☞</td>
<td>☞</td>
<td>☞</td>
</tr>
<tr>
<td><strong>BLEU</strong></td>
<td>☞</td>
<td>☞</td>
<td>☞</td>
<td>☞</td>
<td>☞</td>
</tr>
<tr>
<td><strong>Latent variable</strong></td>
<td>☞</td>
<td>☞</td>
<td>☞</td>
<td>☞</td>
<td>☞</td>
</tr>
<tr>
<td><strong>Oracle translation</strong></td>
<td>☞</td>
<td>☞</td>
<td>☞</td>
<td>☞</td>
<td>☞</td>
</tr>
<tr>
<td><strong>Model regularization</strong></td>
<td>☞</td>
<td>☞</td>
<td>☞</td>
<td>☞</td>
<td>☞</td>
</tr>
</tbody>
</table>

See Zens et al. (2007) for experimental comparisons.
Experimental Results

- Data set: IWSLT CN-EN 2005

<table>
<thead>
<tr>
<th>Training scheme</th>
<th>BLEU</th>
</tr>
</thead>
<tbody>
<tr>
<td>MERT (Nbest, small)</td>
<td>47.7</td>
</tr>
<tr>
<td>MR (Nbest, small)</td>
<td>47.7</td>
</tr>
<tr>
<td>MR (hypergraph, small)</td>
<td>48.4</td>
</tr>
<tr>
<td>MR (hypergraph, large)</td>
<td>48.7</td>
</tr>
</tbody>
</table>

Small: discriminatively tune 5 features

Large: also discriminatively tune the language model (21k additional features)

See Zens et al. (2007) for experimental comparisons on nbest for small number of features.
Outline

• Hypergraph as hypothesis space

• Training methods (Li and Eisner, 2009)

• Exact and approximate inference
  ‣ exact inference
    ★ semiring framework (Li and Eisner, 2009)
  ‣ approximate inference
    ★ “principled” ones
      ✓ variational decoding (Li et. al, 2009)
      ✓ sampling and message passing methods
    ★ heuristic-based ones
      ✓ cube-pruning (Chiang, 2007)

• Joshua project (Li et. al, 2009)
A semiring framework to compute all of these

- **First-order quantities:**
  - expectation
  - entropy
  - Bayes risk
  - cross-entropy
  - KL divergence
  - feature expectations
  - first-order gradient of $Z$

- **Second-order quantities:**
  - Expectation over product
  - interaction between features
  - Hessian matrix of $Z$
  - second-order gradient descent
  - gradient of expectation
  - gradient of entropy or Bayes risk

- **“decoding” quantities:**
  - Viterbi
  - K-best
  - Counting
  - ....
Compute Quantities on a Hypergraph: a Recipe

• Semiring-weighted inside algorithm
  • three steps:
    ▶ choose a semiring
      \[ \langle K, \oplus, \otimes \rangle \]
      a set with plus and times operations
      e.g., integer numbers with regular + and ×
    ▶ specify a weight for each hyperedge
      each weight is a semiring member
    ▶ run the inside algorithm
      complexity is O(hypergraph size)
Semirings

- “Decoding” time semirings (Goodman, 1999)
  - counting, Viterbi, K-best, etc.
- “Training” time semirings
  - first-order expectation semirings (Eisner, 2002)
  - second-order expectation semirings (new)
- Applications of the Semirings (new)
  - entropy, risk, gradient of them, and many more
How many trees?

four 😊

compute it
use a semiring?
Compute the Number of Derivation Trees

Three steps:

- choose a semiring
  - counting semiring: ordinary integers with regular + and x

- specify a weight for each hyperedge

- run the inside algorithm
Bottom-up process in computing the number of trees

\[ k(v_1) = k(e_1) \]
Bottom-up process in computing the number of trees
Compute $k(v_3)$: the weight at node $v_3$

Hyperedge $e_3$: $k(e_3) \otimes k(v_1) \otimes k(v_2) = 1 \otimes 1 \otimes 1 = 1$

Bottom-up process in computing the number of trees
Compute $k(v_3)$: the weight at node $v_3$

Hyperedge $e_4$: $k(e_4) \otimes k(v_1) \otimes k(v_2) = 1 \otimes 1 \otimes 1 = 1$

Bottom-up process in computing the number of trees
Compute $k(v_3)$: the weight at node $v_3$

Hyperedge $e_3$: $k(e_3) \otimes k(v_1) \otimes k(v_2) = 1$

Hyperedge $e_4$: $k(e_4) \otimes k(v_1) \otimes k(v_2) = 1$

Bottom-up process in computing the number of trees
Compute $k(v_3)$: the weight at node $v_3$

$$k(v_3) = k(e_3) \otimes k(v_1) \otimes k(v_2) \oplus k(e_4) \otimes k(v_1) \otimes k(v_2)$$

$$1 \oplus 1 = 2$$

**Bottom-up** process in computing the number of trees
Bottom-up process in computing the number of trees
Bottom-up process in computing the number of trees

\[
k(v_1) = k(e_1) \quad k(v_2) = k(e_2)
\]
\[
k(v_3) = k(e_3) \times k(v_1) \times k(v_2) \oplus k(e_4) \times k(v_1) \times k(v_2)
\]
\[
k(v_4) = k(e_5) \times k(v_1) \times k(v_2) \oplus k(e_6) \times k(v_1) \times k(v_2)
\]
\[
k(v_5) = k(e_7) \times k(v_3) \oplus k(e_8) \times k(v_4)
\]

\[
2 \oplus 2 = 4
\]

\[
k(v_5) = 4
\]

\[
k(v_4) = 2
\]

\[
k(v_3) = 2
\]

\[
k(v_1) = 1
\]

\[
k(v_2) = 1
\]

\[
dianzi_0 \quad shang_1 \quad de_2 \quad mao_3
\]
$k(v_1) = k(e_1)$

$k(v_2) = k(e_2)$

$k(v_3) = k(e_3) \times k(v_1) \times k(v_2) \oplus k(e_4) \times k(v_1) \times k(v_2)$

$k(v_4) = k(e_5) \times k(v_1) \times k(v_2) \oplus k(e_6) \times k(v_1) \times k(v_2)$

$k(v_5) = k(e_7) \times k(v_3) \oplus k(e_8) \times k(v_4)$

**Summary:**

- **input:** a weight at each edge
- **output:** a weight at each node
- $\oplus$ is used at nodes
- $\times$ is used at edges
\[
\begin{align*}
\text{expected translation length?} & \quad 2/8 \times 4 + 6/8 \times 5 = 4.75 \\
\text{variance?} & \quad 2/8 \times (4-4.75)^2 + 6/8 \times (5-4.75)^2 \approx 0.19
\end{align*}
\]
First- and Second-order Expectation Semirings

First-order:  

- each member is a 2-tuple:  \( \langle p, r \rangle \)

<table>
<thead>
<tr>
<th>( \langle p_1, r_1 \rangle \otimes \langle p_2, r_2 \rangle )</th>
<th>( \langle p_1 p_2, p_1 r_2 + p_2 r_1 \rangle )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \langle p_1, r_1 \rangle \oplus \langle p_2, r_2 \rangle )</td>
<td>( \langle p_1 + p_2, r_1 + r_2 \rangle )</td>
</tr>
</tbody>
</table>

Second-order:

- each member is a 4-tuple:  \( \langle p, r, s, t \rangle \)

<table>
<thead>
<tr>
<th>( \langle p_1, r_1, s_1, t_1 \rangle \otimes \langle p_2, r_2, s_2, t_2 \rangle )</th>
<th>( \langle p_1 p_2, p_1 r_2 + p_2 r_1, p_1 s_2 + p_2 s_1, p_1 t_2 + p_2 t_1 + r_1 s_2 + r_2 s_1 \rangle )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \langle p_1, r_1, s_1, t_1 \rangle \oplus \langle p_2, r_2, s_2, t_2 \rangle )</td>
<td>( \langle p_1 + p_2, r_1 + r_2, s_1 + s_2, t_1 + t_2 \rangle )</td>
</tr>
</tbody>
</table>
Compute Expected Translation Length

- choose a semiring
  - first-order expectation semiring
- specify a weight for each hyperedge

\[ k_e \overset{\text{def}}{=} \langle p_e, p_e r_e \rangle \]

- \( p_e \): transition probability or log-linear score at edge \( e \)
- \( r_e \): number of English words generated at edge \( e \)

- run the inside algorithm
Compute Variance in Translation Length

- choose a semiring
  second-order expectation semiring
- specify a weight for each edge
  \[ k_e \overset{\text{def}}{=} \langle p_e, p_e r_e, p_e s_e, p_e r_e s_e \rangle \]
  - \( p_e \): transition probability or log-linear score at edge \( e \)
  - \( r_e = s_e \): number of English words generated at edge \( e \)
- run the inside algorithm
First-order:
each semiring member is a 2-tuple
Second-order: each semiring member is a 4-tuple
Expectations on Hypergraphs

• Expectation over a hypergraph

\[ \bar{r} \overset{\text{def}}{=} \mathbb{E}_p[r] = \sum_{d \in \text{HG}} p(d)r(d) \]

• \( r(d) \) is a function over a derivation \( d \)
e.g., the length of the translation yielded by \( d \)

• \( r(d) \) is additively decomposed
\[ r(d) \overset{\text{def}}{=} \sum_{e \in d} r_e \]
e.g., translation length is additively decomposed!
Second-order Expectations on Hypergraphs

- **Expectation of products** over a hypergraph
  \[
  \bar{t} \overset{\text{def}}{=} \mathbb{E}_p[r \cdot s] = \sum_{d \in \text{HG}} p(d) r(d) s(d)
  \]

- \(r\) and \(s\) are additively decomposed
  \[
  r(d) \overset{\text{def}}{=} \sum_{e \in d} r_e
  \]
  \[
  s(d) \overset{\text{def}}{=} \sum_{e \in d} s_e
  \]

\(r\) and \(s\) can be identical or different functions.
Compute expectation using expectation semiring:

\[ k_e \overset{\text{def}}{=} \langle p_e, p_e r_e \rangle \]

\( p_e \): transition probability or log-linear score at edge \( e \).

\( r_e \)?

**Entropy:**

\[ r_e \overset{\text{def}}{=} \log p_e \]

**Why?**

Entropy is an **expectation**

\[ H(p) = \mathbb{E}_p[-\log p] = - \sum_{d \in \mathbb{H}_G} p(d) \log p(d) \]

\( \log p(d) \) is additively decomposed!
Compute expectation using expectation semiring:

\[
\begin{align*}
ke & \overset{\text{def}}{=} \langle pe, pe re \rangle \\
pe & \text{: transition probability or log-linear score at edge } e \\
re & \text{?}
\end{align*}
\]

**Entropy:**

\[
\begin{align*}
re & \overset{\text{def}}{=} \log pe \\

\end{align*}
\]

**Cross-entropy:**

\[
\begin{align*}
re & \overset{\text{def}}{=} \log qe \\

\end{align*}
\]

**Why?**

cross-entropy is an **expectation**

\[
H(p, q) = \mathbb{E}_p (− \log q) = − \sum_{d \in HG} p(d) \log q(d)
\]

log q(d) is additively decomposed!
Compute expectation using expectation semiring:

\[ k_e \overset{\text{def}}{=} \langle p_e, p_e r_e \rangle \]

\( p_e \): transition probability or log-linear score at edge \( e \)

\( r_e \)?

**Entropy:**

\[ r_e \overset{\text{def}}{=} \log p_e \]

**Cross-entropy:**

\[ r_e \overset{\text{def}}{=} \log q_e \]

**Bayes risk:**

\[ r_e \overset{\text{def}}{=} \text{loss at edge } e \]

**Why?**

Bayes risk is an **expectation**

\[ \text{Risk} = \mathbb{E}_p(L) = - \sum_{d \in HG} p(d) \cdot L(Y(d)) \]

\( L(Y(d)) \) is additively decomposed! (Tromble et al. 2008)
Applications of expectation semirings: a summary

First-order:

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Weight for edge $e$</th>
<th>Value at root</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expectation</td>
<td>$\langle p_e, pe r_e \rangle$</td>
<td>$\langle Z, \bar{r} \rangle$</td>
</tr>
<tr>
<td>First-order gradient</td>
<td>$\langle p_e, \nabla p_e \rangle$</td>
<td>$\langle Z, \nabla Z \rangle$</td>
</tr>
</tbody>
</table>

Second-order:

| Covariance matrix       | $\langle p_e, pe r_e, pe s_e, pe r_e s_e \rangle$             | $\langle Z, \bar{r}, \bar{s}, \bar{t} \rangle$ |
| Hessian matrix          | $\langle p_e, \nabla p_e, \nabla p_e, \nabla^2 p_e \rangle$   | $\langle Z, \nabla Z, \nabla Z, \nabla^2 Z \rangle$ |
| Gradient of expectation | $\langle p_e, pe r_e, \nabla p_e, (\nabla p_e) r_e + p_e (\nabla r_e) \rangle$ | $\langle Z, \bar{r}, \nabla Z, \nabla \bar{r} \rangle$ |

\[ p_e = \exp(\Phi_e \cdot \theta) \quad \nabla p_e = p_e \Phi_e \]

- choose a semiring
- define a weight for each edge
- run inside algorithm
**Applications of expectation semirings: a summary**

### First-order:

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Weight for edge $e$</th>
<th>Value at root</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expectation</td>
<td>$\langle p_e, p_e r_e \rangle$</td>
<td>$\langle Z, r \rangle$</td>
</tr>
<tr>
<td>First-order gradient</td>
<td>$\langle p_e, \nabla p_e \rangle$</td>
<td>$\langle Z, \nabla Z \rangle$</td>
</tr>
</tbody>
</table>

### Second-order:

<table>
<thead>
<tr>
<th></th>
<th>$\langle p_e, p_e r_e, p_e s_e, p_e r_e s_e \rangle$</th>
<th>$\langle Z, \bar{r}, \bar{s}, \bar{t} \rangle$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Covariance matrix</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hessian matrix</td>
<td>$\langle p_e, \nabla p_e, \nabla p_e, \nabla^2 p_e \rangle$</td>
<td>$\langle Z, \nabla Z, \nabla Z, \nabla^2 Z \rangle$</td>
</tr>
<tr>
<td>Gradient of expectation</td>
<td>$\langle p_e, p_e r_e, \nabla p_e, (\nabla p_e) r_e + p_e(\nabla r_e) \rangle$</td>
<td>$\langle Z, \bar{r}, \nabla Z, \nabla \bar{r} \rangle$</td>
</tr>
</tbody>
</table>

Entropy and Bayes risk are expectations.

- choose a semiring
- define a weight for each edge
- run inside algorithm
A semiring framework to compute all of these

- “decoding” quantities:
  - Viterbi
  - K-best
  - Counting
  - ......

• First-order quantities:
  - expectation
  - entropy
  - Bayes risk
  - cross-entropy
  - KL divergence
  - feature expectations
  - first-order gradient of $Z$

• Second-order quantities:
  - Expectation over product
  - interaction between features
  - Hessian matrix of $Z$
  - second-order gradient descent
  - gradient of expectation
  - gradient of entropy or Bayes risk
This work provides a unified, elegant, and efficient framework to compute all of these!

• **First-order quantities:**
  - expectation
  - entropy
  - Bayes risk
  - cross-entropy
  - KL divergence
  - feature expectations
  - first-order gradient of $Z$

• **Second-order quantities:**
  - Expectation over product
  - interaction between features
  - Hessian matrix of $Z$
  - second-order gradient descent
  - gradient of expectation
  - gradient of entropy or Bayes risk

**Improved BLEU score!**
Future: machine learning for MT

- feature interaction
- second-order gradient descent
- minimum risk
- deterministic annealing
- active learning
- semi-supervised learning

semirings for parameter estimation
Outline

• Hypergraph as hypothesis space
  
• Training methods  
  (Li and Eisner, 2009)

• Exact and approximate inference
  
  ▸ exact inference
    ★ semiring framework  
    (Li and Eisner, 2009)
  
  ▸ approximate inference
    ★ “principled” ones
      ✓ variational decoding  
      (Li et. al, 2009)
      ✓ sampling and message passing methods
    ★ heuristic-based ones
      ✓ cube-pruning  
      (Chiang, 2007)

• Joshua project  
  (Li et. al, 2009)
Decoding over a hypergraph

Given a hypergraph of possible translations
(generated for a given foreign sentence by already-trained model)

Pick a single translation to output
(why not just pick the tree with the highest weight?)
Spurious Ambiguity

- Statistical models in MT exhibit **spurious ambiguity**
  
  - Many **different derivations** (e.g., trees or segmentations) generate the **same translation string**

- Tree-based MT systems
  
  - **derivation tree ambiguity**

- Regular phrase-based MT systems
  
  - **phrase segmentation ambiguity**
Spurious Ambiguity in Derivation Trees

Same output: “machine translation software”

Three different derivation trees

Another translation: machine transfer software
**Maximum A Posterior (MAP) Decoding**

<table>
<thead>
<tr>
<th>Translation String</th>
<th>MAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>red translation</td>
<td></td>
</tr>
<tr>
<td>blue translation</td>
<td></td>
</tr>
<tr>
<td>green translation</td>
<td></td>
</tr>
</tbody>
</table>

- **Exact MAP decoding**

\[
y^* = \arg\max_{y \in \text{Trans}(x)} p(y|x)
\]

\[
= \arg\max_{y \in \text{Trans}(x)} \sum_{d \in D(x,y)} p(y, d|x)
\]

- **x**: Foreign sentence
- **y**: English sentence
- **d**: derivation
Maximum A Posterior (MAP) Decoding

- **Exact MAP decoding**

\[ y^* = \arg \max_{y \in \text{Trans}(x)} p(y|x) \]
\[ = \arg \max_{y \in \text{Trans}(x)} \sum_{d \in \mathcal{D}(x,y)} p(y,d|x) \]

**Translation String**  | **MAP**  | **Derivation** | **Probability**
--- | --- | --- | ---
red translation  | 0.28  |  | 0.16
blue translation |  |  | 0.14
green translation |  |  | 0.14

- **x**: Foreign sentence
- **y**: English sentence
- **d**: derivation
Maximum A Posterior (MAP) Decoding

- **translation string**
  - red translation: 0.28
  - blue translation: 0.28
  - green translation

- **Exact MAP decoding**
  \[
  y^* = \arg \max_{y \in \text{Trans}(x)} p(y|x) = \arg \max_{y \in \text{Trans}(x)} \sum_{d \in \text{D}(x,y)} p(y, d|x)
  \]

- **derivation and probability**
  - red translation: derivation, probability = 0.16
  - blue translation: derivation, probability = 0.14
  - green translation: derivation, probability = 0.13

- **Notes**
  - **x**: Foreign sentence
  - **y**: English sentence
  - **d**: derivation
### Maximum A Posterior (MAP) Decoding

<table>
<thead>
<tr>
<th>Translation String</th>
<th>MAP</th>
<th>Derivation</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>red translation</td>
<td>0.28</td>
<td></td>
<td>0.16</td>
</tr>
<tr>
<td>blue translation</td>
<td>0.28</td>
<td></td>
<td>0.14</td>
</tr>
<tr>
<td>green translation</td>
<td>0.44</td>
<td></td>
<td>0.13</td>
</tr>
</tbody>
</table>

- **Exact MAP decoding**

\[
y^* = \arg \max_{y \in \text{Trans}(x)} p(y|x)
\]

\[
= \arg \max_{y \in \text{Trans}(x)} \sum_{d \in D(x,y)} p(y, d|x)
\]

- **x**: Foreign sentence
- **y**: English sentence
- **d**: derivation
Maximum A Posterior (MAP) Decoding

<table>
<thead>
<tr>
<th>translation string</th>
<th>MAP</th>
<th>derivation</th>
<th>probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>red translation</td>
<td>0.28</td>
<td></td>
<td>0.16</td>
</tr>
<tr>
<td>blue translation</td>
<td>0.28</td>
<td></td>
<td>0.14</td>
</tr>
<tr>
<td>green translation</td>
<td>0.44</td>
<td></td>
<td>0.14</td>
</tr>
</tbody>
</table>

• Exact MAP decoding

\[
y^* = \arg \max_{y \in \text{Trans}(x)} p(y|x)
\]

\[
= \arg \max_{y \in \text{Trans}(x)} \sum_{d \in D(x,y)} p(y, d|x)
\]

exponential size NP-hard (Sima’an 1996)

• x: Foreign sentence
• y: English sentence
• d: derivation
Viterbi Approximation

<table>
<thead>
<tr>
<th>translation string</th>
<th>MAP</th>
<th>Viterbi</th>
</tr>
</thead>
<tbody>
<tr>
<td>red translation</td>
<td>0.28</td>
<td></td>
</tr>
<tr>
<td>blue translation</td>
<td>0.28</td>
<td></td>
</tr>
<tr>
<td>green translation</td>
<td>0.44</td>
<td></td>
</tr>
</tbody>
</table>

- **Viterbi approximation**

\[ y^* = \arg \max_{y \in \text{Trans}(x)} \max_{d \in \mathcal{D}(x, y)} p(y, d | x) \]

\[ = \ Y(\arg \max_{d \in \mathcal{D}(x)} p(y, d | x)) \]
### Viterbi Approximation

<table>
<thead>
<tr>
<th>Translation String</th>
<th>MAP</th>
<th>Viterbi</th>
<th>Derivation</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red translation</td>
<td>0.28</td>
<td>0.16</td>
<td></td>
<td>0.16</td>
</tr>
<tr>
<td>Blue translation</td>
<td>0.28</td>
<td>0.14</td>
<td></td>
<td>0.14</td>
</tr>
<tr>
<td>Green translation</td>
<td>0.44</td>
<td>0.13</td>
<td></td>
<td>0.13</td>
</tr>
</tbody>
</table>

- **Viterbi approximation**

\[
y^* = \arg \max_{y \in \text{Trans}(x)} \max_{d \in D(x,y)} p(y, d|x)
\]

\[
y^* = Y(\arg \max_{d \in D(x)} p(y, d|x))
\]
\begin{itemize}
\item N-best approximation (\textit{crunching}) (May and Knight, 2006)
\end{itemize}

\[
y^* = \arg \max_{y \in \text{Trans}(x)} \sum_{d \in D(x, y) \cap \text{ND}(x)} p(y, d | x)
\]
### N-best Approximation

<table>
<thead>
<tr>
<th>Translation String</th>
<th>MAP</th>
<th>Viterbi</th>
<th>4-best crunching</th>
<th>Derivation</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>red translation</td>
<td>0.28</td>
<td>0.16</td>
<td>0.16</td>
<td></td>
<td>0.16</td>
</tr>
<tr>
<td>blue translation</td>
<td>0.28</td>
<td>0.14</td>
<td>0.28</td>
<td></td>
<td>0.14</td>
</tr>
<tr>
<td>green translation</td>
<td>0.44</td>
<td>0.13</td>
<td>0.13</td>
<td></td>
<td>0.13</td>
</tr>
</tbody>
</table>

- **N-best approximation (crunching)** ([May and Knight, 2006](#))

\[
y^* = \arg \max_{y \in \text{Trans}(x)} \sum_{d \in \text{D}(x,y) \cap \text{ND}(x)} p(y, d | x)
\]
### MAP vs. Approximations

<table>
<thead>
<tr>
<th>Translation String</th>
<th>MAP</th>
<th>Viterbi</th>
<th>4-Best Crunching</th>
<th>Derivation</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>red translation</td>
<td>0.28</td>
<td>0.16</td>
<td>0.16</td>
<td></td>
<td>0.16</td>
</tr>
<tr>
<td>blue translation</td>
<td>0.28</td>
<td>0.14</td>
<td>0.28</td>
<td></td>
<td>0.14</td>
</tr>
<tr>
<td>green translation</td>
<td>0.44</td>
<td>0.13</td>
<td>0.13</td>
<td></td>
<td>0.13</td>
</tr>
</tbody>
</table>

- Exact MAP decoding under spurious ambiguity is **intractable** on HG
- Viterbi and crunching are efficient, but ignore most derivations
- **Our goal:** develop an **approximation** that considers all the derivations but still allows **tractable** decoding
Variational Decoding

Decoding using Variational approximation

Decoding using a sentence-specific approximate distribution
Sentence-specific decoding

Three steps:

1. Generate a hypergraph for the foreign sentence

Variational Decoding for MT: an Overview

MAP decoding under P is intractable
Generate a hypergraph

Estimate a model from the hypergraph

\[ q^*(y \mid x) \approx \sum_{d \in D(x,y)} p(y,d \mid x) \]

Decode using \( q^* \) on the hypergraph
Variational Inference

- We want to do inference under $p$, but it is intractable

$$y^* = \arg \max_y p(y | x)$$

- Instead, we derive a simpler distribution $q^*$

$$q^* = \arg \min_{q \in Q} \text{KL}(p || q)$$

- Then, we will use $q^*$ as a surrogate for $p$ in inference

$$y^* = \arg \max_y q^*(y | x)$$
Variational Approximation

• \( q^* \): an approximation having minimum distance to \( p \)

\[
q^* = \arg \min_{q \in Q} KL(p \| q) = \arg \min_{q \in Q} \sum_{y \in \text{Trans}(x)} p \log \frac{p}{q} = \arg \min_{q \in Q} \sum_{y \in \text{Trans}(x)} (p \log p - p \log q) = \arg \max_{q \in Q} \sum_{y \in \text{Trans}(x)} p \log q
\]

• Three questions
  • how to parameterize \( q \)?
  • how to estimate \( q^* \)?
  • how to use \( q^* \) for decoding?

A family of distributions

A constant
Parameterization of $q \in Q$

- Naturally, we parameterize $q$ as an $n$-gram model.
- The probability of a string is a product of the probabilities of those $n$-grams appearing in that string.

**3-gram model**

$y: a \ b \ c \ d \ e \ f$

$$q(y) = q(a) \cdot q(b|a) \cdot q(c|ab) \cdot q(d|bc) \cdot q(e|cd) \cdot q(f|de)$$

Other ways of parameterizations are possible!
Parameterization of $q \in \mathcal{Q}$

- Naturally, we parameterize $q$ as an n-gram model.
- The probability of a string is a product of the probabilities of those n-grams appearing in that string.

3-gram model

$y: a \ b \ c \ d \ e \ f$

$$q(y) = q(a) \cdot q(b|a) \cdot q(c|ab) \cdot q(d|bc) \cdot q(e|cd) \cdot q(f|de)$$

how to estimate these n-gram probabilities?
Estimation of $q^* \in Q$

- Variational approximation
  \[
  q^* = \arg\max_{q \in Q} \sum_{y \in \text{Trans}(x)} p \log q
  \]

- $q^*$ is a maximum likelihood estimate (MLE) where $p$ is the empirical distribution

But in our case, $p$ is defined **not** by a corpus, but by a **hypergraph** for a given test sentence!

---

**estimate**

- bi-gram model
  - brute force
  - dynamic programming
Estimating $q^*$ from a hypergraph: brute force

Bi-gram estimation:

- unpack the hypergraph
Estimating \( q^* \) from a hypergraph: brute force

Bi-gram estimation:

- unpack the hypergraph

\[
\begin{align*}
S &\rightarrow \langle X_0, X_0 \rangle \\
X &\rightarrow \langle X_0 \text{ de } X_1, X_1 \text{ on } X_0 \rangle \\
X &\rightarrow \langle X_0 \text{ de } X_1, X_0 \text{'s } X_1 \rangle \\
S &\rightarrow \langle X_0, X_0 \rangle
\end{align*}
\]
Estimating $q^*$ from a hypergraph: brute force

- **Unpack the hypergraph**
- **Accumulate the soft-count of each bigram**
- **Normalize the counts**

---

**Bi-gram estimation:**

- $p = 1/8$ for "cat on the mat"
- $p = 3/8$ for "the mat's a cat"
- $p = 2/8$ for both "the mat a cat" and "a cat of the mat"

**Example calculations:**

- $q*(on | cat) = 1/8$
- $q*(</s> | cat) = 5/8$
- $q*(of | cat) = 2/8$
Estimating $q^*$ from a hypergraph: dynamic programming

**Bi-gram estimation:**

- run inside-outside on the hypergraph expectation semirings!
- accumulate the **soft-count** of each bigram at each hyperedge
- normalize the counts
Decoding using $q^* \in Q$

- Rescore the hypergraph $HG(x)$

\[ y^* = \arg \max_{y \in HG(x)} q^*(y|x) \]

$q^*$ is an n-gram model.

- have efficient dynamic programming algorithms
- score the hypergraph using an n-gram model
KL divergences under different variational models

\[ q^* = \arg \min_{q \in Q} \text{KL}(p || q) = H(p, q) - H(p) \]

| Measure | \( \overline{H}(p) \) | \( \overline{\text{KL}}(p || \cdot) \) |
|---------|----------------|-----------------|
| bits/word |                | \( q_1^* \) | \( q_2^* \) | \( q_3^* \) | \( q_4^* \) |
| MT’04   | 1.36           | 0.97            | 0.32            | 0.21            | 0.17            |
| MT’05   | 1.37           | 0.94            | 0.32            | 0.21            | 0.17            |

- Larger \( n \) ==> better approximation \( q_n \) ==> smaller KL divergence from \( p \)
- The reduction of KL divergence happens mostly when switching from unigram to bigram
KL divergences under different variational models

\[ q^* = \arg \min_{q \in Q} \text{KL}(p \| q) = H(p, q) - H(p) \]

<table>
<thead>
<tr>
<th>Measure</th>
<th>( \bar{H}(p) )</th>
<th>( q_1^* )</th>
<th>( q_2^* )</th>
<th>( q_3^* )</th>
<th>( q_4^* )</th>
</tr>
</thead>
<tbody>
<tr>
<td>bits/word</td>
<td>1.36</td>
<td>0.97</td>
<td>0.32</td>
<td>0.21</td>
<td>0.17</td>
</tr>
<tr>
<td>MT’04</td>
<td>1.37</td>
<td>0.94</td>
<td>0.32</td>
<td>0.21</td>
<td>0.17</td>
</tr>
<tr>
<td>MT’05</td>
<td>1.37</td>
<td>0.94</td>
<td>0.32</td>
<td>0.21</td>
<td>0.17</td>
</tr>
</tbody>
</table>

How to compute KL on a hypergraph?

use expectation semiring, again ☺
### BLEU scores when using a single variational n-gram model

<table>
<thead>
<tr>
<th>Decoding scheme</th>
<th>MT’04</th>
<th>MT’05</th>
</tr>
</thead>
<tbody>
<tr>
<td>Viterbi</td>
<td>35.4</td>
<td>32.6</td>
</tr>
<tr>
<td>1gram</td>
<td>25.9</td>
<td>24.5</td>
</tr>
<tr>
<td>2gram</td>
<td><strong>36.1</strong></td>
<td><strong>33.4</strong></td>
</tr>
<tr>
<td>3gram</td>
<td>36.0</td>
<td>33.1</td>
</tr>
<tr>
<td>4gram</td>
<td>35.8</td>
<td>32.9</td>
</tr>
</tbody>
</table>

- unigram performs very badly
- bigram achieves best BLEU scores

modeling error in \( p \)
BLEU cares about both low- and high-order \( n \)-gram matches

- Interpolate variational \( n \)-gram approximations for different \( n \)

\[
y^* = \arg \max_{y \in \text{HG}(x)} \sum_n \theta_n \cdot \log q^*_n(y \mid x)
\]

- Viterbi is yet another approximation of \( p \), so throw it in too

\[
y^* = \arg \max_{y \in \text{HG}(x)} \left( \sum_n \theta_n \cdot \log q^*_n(y \mid x) + \theta_v \cdot \log p_{\text{Viterbi}}(y \mid x) \right)
\]
Minimum Bayes Risk (MBR) decoding?

(Kumar and Byrne, 2004)
(Tromble et al. 2008)
(Denero et al. 2009)
(Li et al. 2009)
Minimum Risk Decoding

• Maximum A Posterior (MAP) decoding
  • find the most probable translation string
    \[ y^* = \arg \max_{y \in HG(x)} p(y|x) \]

• Minimum risk decoding
  • find the consensus translation string
    \[ y^* = \arg \min_{y \in HG(x)} \text{Risk}(y) \]

risk = expected loss = expected error rate
Variational Decoding (VD) vs. MBR (Tromble et al. 2008)

Both BLEU metric and our variational distributions happen to use n-gram dependencies.
• Variational decoding with interpolation

\[ y^* = \arg \max_{y \in \text{HG}(x)} \sum_n \theta_n \cdot \log q_n^*(y \mid x) \]

\[ q_n(y \mid x) = \prod_{w \in W_n} q(r(w) \mid h(w), x)c_w(y) \]

\[ q(r(w) \mid h(w), x) = \frac{\sum_{y'} c_w(y')p(y' \mid x)}{\sum_{y'} c_h(w)(y')p(y' \mid x)} \]

• Minimum risk decoding (Tromble et al. 2008)

\[ y^* = \arg \max_{y \in \text{HG}(x)} \sum_n \theta_n \cdot g_n(y \mid x) \]

\[ g_n(y \mid x) = \sum_{w \in W_n} g(w \mid x)c_w(y) \]

\[ g(w \mid x) = \sum_{y'} \delta_w(y')p(y' \mid x) \]
• Variational decoding with interpolation

\[ y^* = \arg \max_{y \in \mathbb{HG}(x)} \sum_n \theta_n \cdot \log q_n^*(y \mid x) \]

\[ q_n(y \mid x) = \prod_{w \in W_n} q(r(w) \mid h(w), x)^{c_w}(y) \]

\[ q(r(w) \mid h(w), x) = \frac{\sum_{y'} c_w(y') p(y' \mid x)}{\sum_{y'} c_h(w)(y') p(y' \mid x)} \]

• Minimum risk decoding (Tromble et al. 2008)

\[ y^* = \arg \max_{y \in \mathbb{HG}(x)} \sum_n \theta_n \cdot g_n(y \mid x) \]

\[ g_n(y \mid x) = \sum_{w \in W_n} g(w \mid x)^{c_w}(y) \]

\[ g(w \mid x) = \sum_{y'} \delta_w(y') p(y' \mid x) \]

non-probabilistic

very expensive to compute
## BLEU Results on Chinese-English NIST MT 2004 Tasks

<table>
<thead>
<tr>
<th>Decoding scheme</th>
<th>BLEU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Viterbi</td>
<td>35.4</td>
</tr>
<tr>
<td>MBR ((K=1000))</td>
<td>35.8</td>
</tr>
<tr>
<td>Crunching ((N=10000))</td>
<td>35.7</td>
</tr>
<tr>
<td>Crunching+MBR ((N=10000))</td>
<td>35.8</td>
</tr>
<tr>
<td>Variational ((1to4gram+wp+vt))</td>
<td><strong>36.6</strong></td>
</tr>
</tbody>
</table>

- variational decoding improves over Viterbi, MBR, and crunching

---

*(Kumar and Byrne, 2004)*

*(May and Knight, 2006)*
Variational Inference

- We want to do inference under $p$, but it is intractable

\[ y^* = \arg \max_y p(y|x) \]

- Instead, we derive a simpler distribution $q^*$

\[ q^* = \arg \min_{q \in Q} \text{KL}(p||q) \]

- Then, we will use $q^*$ as a surrogate for $p$ in inference

\[ y^* = \arg \max_y q^*(y | x) \]
Summary of Variational Decoding

• Exact MAP decoding with spurious ambiguity is intractable

• Viterbi or N-best approximations are efficient, but ignore most derivations

• We developed a variational approximation, which considers all derivations but still allows tractable decoding

• Our variational decoding improves a state of the art baseline
Future directions

• Decoding with spurious ambiguity is a common problem in many other NLP applications
  • Models with latent variables
  • Data oriented parsing (DOP)
  • Hidden Markov Models (HMM)
  • ......
Outline

- Hypergraph as hypothesis space

- Training methods  
  (Li and Eisner, 2009)

- Exact and approximate inference
  
  - exact inference
    
    - semiring framework  
      (Li and Eisner, 2009)
  
  - approximate inference
    
    - “principled” ones
      
      - variational decoding  
        (Li et. al, 2009)
      
      - sampling and message passing methods

    - heuristic-based ones
      
      - cube-pruning  
        (Chiang, 2007)

- Joshua project  
  (Li et. al, 2009)
Graphical Model for Part of Speech Tagging

Variable: pos tags as values

Factor: encode transition or observation probabilities
Belief Propagation

- $\alpha$ sends evidence from left part of graph
- $\beta$ sends evidence from right part of graph
Non-local dependency

Variable: pos tags as values

Factor: encode transition or observation probabilities

Loopy BP!
Belief Propagation for NLP

• Dependency parsing with non-local information  (Smith and Eisner, 2008)

• Word alignment  (Cromieres and Kurohashi, 2009)

• String to string transducers  (Dreyer and Eisner, 2009)
Sampling

- **General idea**
  - Generate a randomized n-best list
  - Compute posterior or max over the randomized n-best list

- **Sampling used for MT**
  - Gibbs sampler for Bayesian inference (Blunsom et. al, 2009)
  - Gibbs sampling for MAP decoding with spurious ambiguity (Arun et. al, 2009)
  - Gibbs sampling for word alignment (Denero et. al, 2008)
Heuristic-based Approximate Inference

- Beam pruning
- Cube pruning (Chiang, 2007)
- Cube growing (Huang and Chiang, 2007)
- Forest re-ranking (Huang, 2008)
- Oracle extraction (Li and Khudanpur, 2009)
Outline

• Hypergraph as hypothesis space

• Training methods (Li and Eisner, 2009)

• Exact and approximate inference
  ▸ exact inference
    ★ semiring framework (Li and Eisner, 2009)
  ▸ approximate inference
    ★ “principled” ones
      ✓ variational decoding (Li et. al, 2009)
      ✓ sampling and message passing methods
    ★ heuristic-based ones
      ✓ cube-pruning (Chiang, 2007)

• Joshua project (Li et. al, 2009)
Conclusions

• A hypergraph is a general way to represent a hypothesis space
• Minimum risk is a promising training objective
• Semiring framework is a general way to do exact inference
• Approximate inference has great applications (e.g., variational decoding) in MT
Future: machine learning for MT

Current paradigm:  Approximate model + Exact inference

- rely on a specific grammars (e.g., FST, ITG, Hiero, or GHKM)
- everything is **local**, e.g., we can’t even enforce *translation consistence* (similar words should get similar translations)
- rely on dynamic programs for exact inference

New paradigm:  “Exact” model + Approximate inference

- use a flexible search space, each grammar is treated as a feature function
- include non-local dependency
- use principled approximate inference techniques, e.g., variational inference, Gibbs sampling, or belief propagation
- approximate inference calls dynamic programs for sub-steps
Joshua project
Motivation

• Towards a general purpose tree-based MT toolkit
  • string to string with latent tree structures (Chiang, 2005)
  • tree to string (Quirk et. al, 2006); (Liu et. al, 2006)
  • string to tree (Galley et. al, 2006)
  • tree to tree (Eisner, 2003)

• Successor to Moses
Joshua project

• An open-source parsing-based MT toolkit (Li et al. 2009)
  • support Hiero (Chiang, 2007) and SAMT (Venugopal et al., 2007)

• Team members
  • Zhifei Li, Chris Callison-Burch, Chris Dyer, Sanjeev Khudanpur, Wren Thornton, Jonathan Weese, Juri Ganitkevitch, Lane Schwartz, and Omar Zaidan

Only rely on Giza++ and SRI LM!
All the methods presented have been implemented in Joshua!
Highlights

• Everything is written in Java

• easy to run under different platforms including Windows

• Easy to extend

• organize the code into packages

• define interfaces across packages

• Scalable

• decode a sentence with one second
Chart-Parsing Decoder

- Input sentences are parsed using the CKY algorithm
- Feature functions are evaluated during parsing
- High-cost constituents are pruned during parsing
- Parsing results are stored in a hypergraph
- K-best extraction is performed on the hypergraph
Suffix-Array Grammar Extraction  (Lopez, 2007)

- Grammars are extracted using suffix arrays
- Grammar rules can be extracted per sentence while decoding
- Allows for very large parallel training corpora
  - eliminates traditional requirement to explicitly extract, sort, and calculate probabilities for all possible rules.
  - only extracts rules actually needed to translate the sentence
Other functions

- Minimum error rate training (MERT)
- Parallel decoding
- Distributed language models
- Variational decoding
- Semiring parsing
- Minimum risk training
- Tree and hypergraph visualization
Quick start

- Check out the software
  - `svn co https://joshua.svn.sf.net/svnroot/joshua/trunk` joshua
- Prepare monolingual and bilingual training data
- Train a language model using the SRI LM toolkit
- Train a translation model
  - sub-sample bilingual data (optional)
  - create word alignments using GIZA++
  - run suffix-array grammar extraction
- Perform minimum error rate training
- Decode test sets
Thank you!

XieXie!

谢谢!