Discriminative Training
MT Marathon lecture

Miloš Stanojević

ILLC, University of Amsterdam

September 11, 2015
Going from Generative to Discriminative models

Start with generative noisy channel model:

\[ t^* = \arg \max_{t \in T(s)} p(t|s) \]

Why would we want to do this?

▶ We can add more indicators (features) of good translation
▶ We can give different weight to different features
▶ And all this done in a way to directly optimize desired metric

Disadvantage? Losing probabilistic interpretation
Going from Generative to Discriminative models

Start with generative noisy channel model:

\[
t^* = \arg \max_{t \in T(s)} p(t \mid s) = \arg \max_{t \in T(s)} \frac{p(s \mid t)p(t)}{p(s)}
\]
Going from Generative to Discriminative models

Start with generative noisy channel model:

\[
t^* = \arg\max_{t \in T(s)} p(t|s) = \arg\max_{t \in T(s)} \frac{p(s|t)p(t)}{p(s)} = \arg\max_{t \in T(s)} p(s|t)p(t)
\]

Why would we want to do this?

- We can add more indicators (features)
- We can give different weight to different features
- And all this done in a way to directly optimize desired metric

Disadvantage? Losing probabilistic interpretation
Going from Generative to Discriminative models

Start with generative noisy channel model:

\[ t^* = \arg\max_{t \in T(s)} p(t | s) = \arg\max_{t \in T(s)} \frac{p(s | t) p(t)}{p(s)} = \arg\max_{t \in T(s)} p(s | t) p(t) \]

\[ = \arg\max_{t \in T(s)} \log p(s | t) + \log p(t) \]
Going from Generative to Discriminative models

Start with generative noisy channel model:

\[
\begin{align*}
t^* &= \arg\max_{t \in T(s)} p(t|s) = \arg\max_{t \in T(s)} \frac{p(s|t)p(t)}{p(s)} = \arg\max_{t \in T(s)} p(s|t)p(t) \\
&= \arg\max_{t \in T(s)} \log p(s|t) + \log p(t) \\
&= \arg\max_{t \in T(s)} \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} \log p(s|t) \\ \log p(t) \end{bmatrix}
\end{align*}
\]

Why would we want to do this?

▶ We can add more indicators (features)
▶ We can give different weight to different features
▶ And all this done in a way to directly optimize desired metric

Disadvantage? Losing probabilistic interpretation
Going from Generative to Discriminative models

Start with generative noisy channel model:

\[
\begin{align*}
t^* &= \arg\max_{t \in T(s)} p(t|s) = \arg\max_{t \in T(s)} \frac{p(s|t)p(t)}{p(s)} = \arg\max_{t \in T(s)} p(s|t)p(t) \\
&= \arg\max_{t \in T(s)} \log p(s|t) + \log p(t) \\
&= \arg\max_{t \in T(s)} \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} \log p(s|t) \\ \log p(t) \end{bmatrix} \\
&= \arg\max_{t \in T(s)} \lambda^T h(s, t)
\end{align*}
\]

end with linear discriminative model

Why would we want to do this?

▶ We can add more indicators (features) of good translation
▶ We can give different weight to different features
▶ And all this done in a way to directly optimize desired metric

Disadvantage? Losing probabilistic interpretation
Going from Generative to Discriminative models

Start with generative noisy channel model:

\[ t^* = \arg\max_{t \in T(s)} p(t|s) = \arg\max_{t \in T(s)} \frac{p(s|t)p(t)}{p(s)} = \arg\max_{t \in T(s)} p(s|t)p(t) \]

\[ = \arg\max_{t \in T(s)} \log p(s|t) + \log p(t) \]

\[ = \arg\max_{t \in T(s)} \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} \log p(s|t) \\ \log p(t) \end{bmatrix} \]

\[ = \arg\max_{t \in T(s)} \lambda^T h(s, t) \]

end with linear discriminative model

Why would we want to do this?

- We can add more indicators (features) of good translation
- We can give different weight to different features
- And all this done in a way to directly optimize desired metric

Disadvantage? Losing probabilistic interpretation
Going from Generative to Discriminative models

Start with generative noisy channel model:

\[
t^* = \arg \max_{t \in T(s)} p(t|s) = \arg \max_{t \in T(s)} \frac{p(s|t)p(t)}{p(s)} = \arg \max_{t \in T(s)} p(s|t)p(t)
\]

\[
= \arg \max_{t \in T(s)} \log p(s|t) + \log p(t)
\]

\[
= \arg \max_{t \in T(s)} \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} \log p(s|t) \\ \log p(t) \end{bmatrix}
\]

\[
= \arg \max_{t \in T(s)} \lambda^T h(s, t)
\]

Why would we want to do this?

- We can add more indicators (features) of good translation

end with linear discriminative model
Going from Generative to Discriminative models

Start with generative noisy channel model:

\[ t^* = \arg\max_{t \in T(s)} p(t|s) = \arg\max_{t \in T(s)} \frac{p(s|t)p(t)}{p(s)} = \arg\max_{t \in T(s)} p(s|t)p(t) \]

\[ = \arg\max_{t \in T(s)} \log p(s|t) + \log p(t) \]

\[ = \arg\max_{t \in T(s)} \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} \log p(s|t) \\ \log p(t) \end{bmatrix} \]

\[ = \arg\max_{t \in T(s)} \lambda^T h(s, t) \quad \text{end with linear discriminative model} \]

Why would we want to do this?

- We can add more indicators (features) of good translation
- We can give different weight to different features
Going from Generative to Discriminative models

Start with generative noisy channel model:

\[ t^* = \arg\max_{t \in T(s)} p(t|s) = \arg\max_{t \in T(s)} \frac{p(s|t)p(t)}{p(s)} = \arg\max_{t \in T(s)} p(s|t)p(t) \]

\[ = \arg\max_{t \in T(s)} \log p(s|t) + \log p(t) \]

\[ = \arg\max_{t \in T(s)} \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} \log p(s|t) \\ \log p(t) \end{bmatrix} \]

\[ = \arg\max_{t \in T(s)} \lambda^T h(s, t) \]

end with linear discriminative model

Why would we want to do this?

- We can add more indicators (features) of good translation
- We can give different weight to different features
- And all this done in a way to directly optimize desired metric
Going from Generative to Discriminative models

Start with generative noisy channel model:

\[ t^* = \arg\max_{t \in T(s)} p(t|s) = \arg\max_{t \in T(s)} \frac{p(s|t)p(t)}{p(s)} = \arg\max_{t \in T(s)} p(s|t)p(t) \]

\[ = \arg\max_{t \in T(s)} \log p(s|t) + \log p(t) \]

\[ = \arg\max_{t \in T(s)} \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} \log p(s|t) \\ \log p(t) \end{bmatrix} \]

\[ = \arg\max_{t \in T(s)} \lambda^T h(s, t) \quad \text{end with linear discriminative model} \]

Why would we want to do this?

▶ We can add more indicators (features) of good translation
▶ We can give different weight to different features
▶ And all this done in a way to directly optimize desired metric

Disadvantage? Losing probabilistic interpretation
Optimize for BLEU directly

Decoder

\[
\text{argmax } \lambda^t \ h(s,t)
\]
Optimize for BLEU directly

\[
\text{Decoder} \quad \text{argmax} \; \lambda^T \; h(s,t)
\]
Optimize for BLEU directly

Decoder

\[ \text{argmax} \ \lambda^T h(s, t) \]

MT Metric

BLEU

\[
\begin{array}{ccc}
\text{t1} & \text{S1} & \text{B1} \\
\text{t2} & \text{S2} & \text{B2} \\
\vdots & \vdots & \vdots \\
\text{tn} & \text{S3} & \text{B3} \\
\end{array}
\]
Optimize for BLEU directly
Optimize for BLEU directly

Decoder
\[ \text{argmax} \, \lambda^T \, h(s, t) \]

New \( \lambda \)

Learning algorithm
MERT or MIRA or PRO

MT Metric
BLEU

\( t_1 \)
\( t_2 \)
\( \ldots \)
\( t_n \)

\( S_1 \)
\( S_2 \)
\( \ldots \)
\( S_3 \)
MERT [Och, 2003]

- MERT is the most often used algorithm for this task
- Optimizes parameters one by one
- Directly optimizes objective
- Works well with systems with small number of features
MERT [Och, 2003]  

MERT optimizes only one parameter while keeping others fixed.

\[
\text{score}(s, t) = \lambda^T h(s, t) = \sum_i \lambda_i h_i(s, t)
\]
MERT [Och, 2003]

MERT optimizes only one parameter while keeping others fixed.

\[
\text{score}(s, t) = \lambda^T h(s, t) \\
= \sum_i \lambda_i h_i(s, t) \\
= \lambda_c h_c(s, t) + \sum_{i \neq c} \lambda_i h_i(s, t)
\]
MERT [Och, 2003]

MERT optimizes only one parameter while keeping others fixed.

\[ score(s, t) = \lambda^T h(s, t) = \sum_i \lambda_i h_i(s, t) = \lambda_c h_c(s, t) + \sum_{i \neq c} \lambda_i h_i(s, t) = \lambda_c h_c(s, t) + u_c(s, t) \]
MERT [Och, 2003]

MERT optimizes only one parameter while keeping others fixed.

\[
\text{score}(s, t) = \lambda^T h(s, t) \\
= \sum_{i} \lambda_i h_i(s, t) \\
= \lambda_c h_c(s, t) + \sum_{i \neq c} \lambda_i h_i(s, t) \\
= \lambda_c h_c(s, t) + u_c(s, t)
\]
MERT [Och, 2003]

- Extract all *threshold points* where argmax changes
MERT [Och, 2003]

- Extract all **threshold points** where argmax changes
- Evaluate each set of threshold points with BLEU score
MERT [Och, 2003]

- Extract all **threshold points** where argmax changes
- Evaluate each set of threshold points with BLEU score
- Take the best one and then go again through the decoding loop
\[ \text{score}(s, t_1) = \text{score}(s, t_2) \]
MERT\[Och, 2003\]

\[
\begin{align*}
\text{score}(s, t_1) &= \text{score}(s, t_2) \\
\lambda_c h_c(s, t_1) + u_c(s, t_1) &= \lambda_c h_c(s, t_2) + u_c(s, t_2)
\end{align*}
\]
MERT [Och, 2003]

\[ \text{score}(s, t_1) = \text{score}(s, t_2) \]

\[ \lambda_c h_c(s, t_1) + u_c(s, t_1) = \lambda_c h_c(s, t_2) + u_c(s, t_2) \]

\[ \lambda_c = \frac{u_c(s, t_1) - u_c(s, t_2)}{h_c(s, t_2) - h_c(s, t_1)} \]
MERT[Och, 2003]

Few more tricks:

- We can speed up this by looking for top threshold points:
  - start with the steepest line (smallest $h_c(s, t_1)$)
  - $score(x) = \lambda_c \cdot h_c(s, t_1) + u_c(s, t_1)$
  - and find the most negative threshold point for that line
MERT [Och, 2003]

Few more tricks:

- We can speed up this by looking for top threshold points start with the steepest line (smallest $h_c(s, t_1)$)
  
  $\text{score}(x) = \lambda_c h_c(s, t_1) + u_c(s, t_1)$

  and find the most negative threshold point for that line

- Accumulate n-best lists over different decoder runs
Few more tricks:

- We can speed up this by looking for top threshold points
  start with the steepest line (smallest $h_c(s, t_1)$)

\[
score(x) = \lambda_c h_c(s, t_1) + u_c(s, t_1)
\]

and find the most negative threshold point for that line

- Accumulate n-best lists over different decoder runs

- Average the weights of 3 MERT runs
MERT – good and bad sides

Good sides:
- Optimizes corpus level metrics directly.

Bad sides:
- Gets stuck in local minima
  - example of finding the highest point in San Francisco
    - [Koehn, 2010]
- Instable: BLEU varies a lot
  - requires at least 3 runs to make it significant
    - [Clark et al., 2011]
- Cannot handle more than a dozen of features
PRO [Hopkins and May, 2011]

PRO is a simple alternative that can allow training lots of features.
PRO[Hopkins and May, 2011]

PRO is a simple alternative that can allow training lots of features. First sample from n-best list many hypotheses pairs $(t_{\text{better}}, t_{\text{worse}})$ where $\text{eval}(t_{\text{better}}, r) > \text{eval}(t_{\text{worse}}, r)$
PRO [Hopkins and May, 2011]

PRO is a simple alternative that can allow training lots of features. First sample from n-best list many hypotheses pairs $(t_{\text{better}}, t_{\text{worse}})$ where $\text{eval}(t_{\text{better}}, r) > \text{eval}(t_{\text{worse}}, r)$

For each pair

\[
\text{score}(s, t_{\text{better}}) > \text{score}(s, t_{\text{worse}})
\]
PRO [Hopkins and May, 2011]

PRO is a simple alternative that can allow training lots of features. First sample from n-best list many hypotheses pairs 
\((t_{\text{better}}, t_{\text{worse}})\) where \(\text{eval}(t_{\text{better}}, r) > \text{eval}(t_{\text{worse}}, r)\)

For each pair

\[
\begin{align*}
\text{score}(s, t_{\text{better}}) & > \text{score}(s, t_{\text{worse}}) \\
\lambda^T h(s, t_{\text{better}}) & > \lambda^T h(s, t_{\text{worse}})
\end{align*}
\]

PRO [Hopkins and May, 2011]

PRO is a simple alternative that can allow training lots of features. First sample from n-best list many hypotheses pairs

\((t_{\text{better}}, t_{\text{worse}})\) where \(\text{eval}(t_{\text{better}}, r) > \text{eval}(t_{\text{worse}}, r)\)

For each pair

\[
\text{score}(s, t_{\text{better}}) > \text{score}(s, t_{\text{worse}})
\]

\[
\lambda^T h(s, t_{\text{better}}) > \lambda^T h(s, t_{\text{worse}})
\]

\[
\lambda^T (h(s, t_{\text{better}}) - h(s, t_{\text{worse}})) > 0
\]
PRO [Hopkins and May, 2011]

PRO is a simple alternative that can allow training lots of features. First sample from n-best list many hypotheses pairs $(t_{\text{better}}, t_{\text{worse}})$ where $\text{eval}(t_{\text{better}}, r) > \text{eval}(t_{\text{worse}}, r)$

For each pair

$$\text{score}(s, t_{\text{better}}) > \text{score}(s, t_{\text{worse}})$$

$$\lambda^T \mathbf{h}(s, t_{\text{better}}) > \lambda^T \mathbf{h}(s, t_{\text{worse}})$$

$$\lambda^T (\mathbf{h}(s, t_{\text{better}}) - \mathbf{h}(s, t_{\text{worse}})) > 0$$

$$\lambda^T (\mathbf{h}(s, t_{\text{worse}}) - \mathbf{h}(s, t_{\text{better}})) < 0$$
PRO is a simple alternative that can allow training lots of features. First sample from n-best list many hypotheses pairs \((t_{\text{better}}, t_{\text{worse}})\) where \(\text{eval}(t_{\text{better}}, r) > \text{eval}(t_{\text{worse}}, r)\)

For each pair

\[
\text{score}(s, t_{\text{better}}) > \text{score}(s, t_{\text{worse}})
\]

\[
\lambda^T \mathbf{h}(s, t_{\text{better}}) > \lambda^T \mathbf{h}(s, t_{\text{worse}})
\]

\[
\lambda^T (\mathbf{h}(s, t_{\text{better}}) - \mathbf{h}(s, t_{\text{worse}})) > 0
\]

\[
\lambda^T (\mathbf{h}(s, t_{\text{worse}}) - \mathbf{h}(s, t_{\text{better}})) < 0
\]

Train linear classifier with these as positive and negative training instance.
PRO is a simple alternative that can allow training lots of features. First sample from n-best list many hypotheses pairs \((t_{\text{better}}, t_{\text{worse}})\) where \(\text{eval}(t_{\text{better}}, r) > \text{eval}(t_{\text{worse}}, r)\)

For each pair

\[
\text{score}(s, t_{\text{better}}) > \text{score}(s, t_{\text{worse}})
\]
\[
\lambda^T h(s, t_{\text{better}}) > \lambda^T h(s, t_{\text{worse}})
\]
\[
\lambda^T (h(s, t_{\text{better}}) - h(s, t_{\text{worse}})) > 0
\]
\[
\lambda^T (h(s, t_{\text{worse}}) - h(s, t_{\text{better}})) < 0
\]

Train linear classifier with these as positive and negative training instance.
Repeat this many times until convergence in n-best list
PRO [Hopkins and May, 2011]

PRO is a simple alternative that can allow training lots of features. First sample from n-best list many hypotheses pairs \((t_{\text{better}}, t_{\text{worse}})\) where \(\text{eval}(t_{\text{better}}, r) > \text{eval}(t_{\text{worse}}, r)\)

For each pair

\[
\begin{align*}
\text{score}(s, t_{\text{better}}) &> \text{score}(s, t_{\text{worse}}) \\
\lambda^T h(s, t_{\text{better}}) &> \lambda^T h(s, t_{\text{worse}}) \\
\lambda^T (h(s, t_{\text{better}}) - h(s, t_{\text{worse}})) &> 0 \\
\lambda^T (h(s, t_{\text{worse}}) - h(s, t_{\text{better}})) &< 0
\end{align*}
\]

Train linear classifier with these as positive and negative training instance.

Repeat this many times until convergence in n-best list

Repeat this with the loop trough the decoder
MIRA

- MIRA is a **large-margin** online learning algorithm similar to perceptron [Watanabe et al., 2007].
- Large margin is enforced between between hope and fear translations [Chiang et al., 2008]

\[
t_{\text{hope}} = \arg\max_t \text{score}(s, t) + \text{eval}(t, r)
\]

\[
t_{\text{fear}} = \arg\max_t \text{score}(s, t) - \text{eval}(t, r)
\]

- Batch version [Cherry and Foster, 2012] present in Moses.
MIRA

\[
\begin{align*}
    t_{\text{hope}} & = \arg\max_t \, \text{score}(s, t) + \text{eval}(t, r) \\
    t_{\text{fear}} & = \arg\max_t \, \text{score}(s, t) - \text{eval}(t, r)
\end{align*}
\]
MIRA

\[ t_{\text{hope}} = \operatorname{argmax}_t \text{score}(s, t) + \text{eval}(t, r) \]

\[ t_{\text{fear}} = \operatorname{argmax}_t \text{score}(s, t) - \text{eval}(t, r) \]

\[ \text{margin} = \text{score}(s, t_{\text{fear}}) - \text{score}(s, t_{\text{hope}}) \]
\[
\begin{align*}
t_{\text{hope}} &= \arg\max_t \text{score}(s, t) + \text{eval}(t, r) \\
t_{\text{fear}} &= \arg\max_t \text{score}(s, t) - \text{eval}(t, r) \\
\text{margin} &= \text{score}(s, t_{\text{fear}}) - \text{score}(s, t_{\text{hope}}) \\
\text{cost} &= \text{BLEU}(t_{\text{hope}}, r) - \text{BLEU}(t_{\text{fear}}, r)
\end{align*}
\]
\begin{aligned}
  t_{\text{hope}} &= \arg\max_t \text{score}(s, t) + \text{eval}(t, r) \\
  t_{\text{fear}} &= \arg\max_t \text{score}(s, t) - \text{eval}(t, r) \\
  \text{margin} &= \text{score}(s, t_{\text{fear}}) - \text{score}(s, t_{\text{hope}}) \\
  \text{cost} &= \text{BLEU}(t_{\text{hope}}, r) - \text{BLEU}(t_{\text{fear}}, r) \\
  \lambda &\leftarrow \lambda + \delta(h(s, t_{\text{hope}}) - h(s, t_{\text{fear}}))
\end{aligned}
\[ t_{\text{hope}} = \arg\max_t \text{score}(s, t) + \text{eval}(t, r) \]
\[ t_{\text{fear}} = \arg\max_t \text{score}(s, t) - \text{eval}(t, r) \]
\[ \text{margin} = \text{score}(s, t_{\text{fear}}) - \text{score}(s, t_{\text{hope}}) \]
\[ \text{cost} = \text{BLEU}(t_{\text{hope}}, r) - \text{BLEU}(t_{\text{fear}}, r) \]
\[ \lambda \leftarrow \lambda + \delta (h(s, t_{\text{hope}}) - h(s, t_{\text{fear}})) \]
\[ \delta = \min \left( C, \frac{\text{margin} + \text{cost}}{\|h(s, t_{\text{hope}}) - h(s, t_{\text{fear}})\|^2} \right) \]

\( \delta \) changes (unlike in Perceptron) to increase the margin
\[ t_{\text{hope}} = \arg\max_t \text{score}(s, t) + \text{eval}(t, r) \]

\[ t_{\text{fear}} = \arg\max_t \text{score}(s, t) - \text{eval}(t, r) \]

\[ \text{margin} = \text{score}(s, t_{\text{fear}}) - \text{score}(s, t_{\text{hope}}) \]

\[ \text{cost} = \text{BLEU}(t_{\text{hope}}, r) - \text{BLEU}(t_{\text{fear}}, r) \]

\[ \lambda \leftarrow \lambda + \delta (h(s, t_{\text{hope}}) - h(s, t_{\text{fear}})) \]

\[ \delta = \min \left( C, \frac{\text{margin} + \text{cost}}{||h(s, t_{\text{hope}}) - h(s, t_{\text{fear}})||^2} \right) \]

\[ \delta \text{ changes (unlike in Perceptron) to increase the margin} \]

Repeat this many times until convergence in n-best list.
MIRA

\[
\begin{align*}
t_{\text{hope}} &= \arg\max_t \text{score}(s, t) + \text{eval}(t, r) \\
t_{\text{fear}} &= \arg\max_t \text{score}(s, t) - \text{eval}(t, r) \\
\text{margin} &= \text{score}(s, t_{\text{fear}}) - \text{score}(s, t_{\text{hope}}) \\
\text{cost} &= \text{BLEU}(t_{\text{hope}}, r) - \text{BLEU}(t_{\text{fear}}, r) \\
\lambda &\leftarrow \lambda + \delta (h(s, t_{\text{hope}}) - h(s, t_{\text{fear}})) \\
\delta &= \min \left( C, \frac{\text{margin} + \text{cost}}{\| h(s, t_{\text{hope}}) - h(s, t_{\text{fear}}) \|^2} \right)
\end{align*}
\]

\(\delta\) changes (unlike in Perceptron) to increase the margin
Repeat this many times until convergence in n-best list
Repeat this with the loop through the decoder
Lots of open problems

▶ Evaluation metrics related:
Lots of open problems

- Evaluation metrics related:
  - MIRA, PRO and Perceptron require sentence level metric (BLEU doesn’t work well)

- Representation of space of translations:
  - n-best list is too small (compared to exponential space)
  - lattice and hyper-graph are better options but too complicated to use because metrics don’t decompose to (hyper-)arcs
  - n-best is not really n-best because of pruning which breaks convergence guarantees [Liu and Huang, 2014]

- Optimization itself:
  - increase margin? minimize risk?
  - latent variables (towards which derivation to optimize?)
Lots of open problems

- Evaluation metrics related:
  - MIRA, PRO and Perceptron require sentence level metric (BLEU doesn't work well)
  - use good metrics
Lots of open problems

- Evaluation metrics related:
  - MIRA, PRO and Perceptron require sentence level metric (BLEU doesn’t work well)
  - use good metrics
  - but good metrics oftend are not good for tuning
Lots of open problems

- Evaluation metrics related:
  - MIRA, PRO and Perceptron require sentence level metric (BLEU doesn't work well)
  - use good metrics
  - but good metrics oftend are not good for tuning

- Representation of space of translations:
Lots of open problems

- Evaluation metrics related:
  - MIRA, PRO and Perceptron require sentence level metric (BLEU doesn’t work well)
  - use good metrics
  - but good metrics oftend are not good for tuning

- Representation of space of translations:
  - n-best list is too small (compared to exponential space)
Lots of open problems

- Evaluation metrics related:
  - MIRA, PRO and Perceptron require sentence level metric (BLEU doesn’t work well)
  - use good metrics
  - but good metrics oftend are not good for tuning

- Representation of space of translations:
  - n-best list is too small (compared to exponential space)
  - lattice and hyper-graph are better options but too complicated to use because metrics don’t decompose to (hyper-)arcs
Lots of open problems

- Evaluation metrics related:
  - MIRA, PRO and Perceptron require sentence level metric (BLEU doesn’t work well)
  - use good metrics
  - but good metrics oftend are not good for tuning

- Representation of space of translations:
  - n-best list is too small (compared to exponential space)
  - lattice and hyper-graph are better options but too complicated to use because metrics don’t decompose to (hyper-)arcs
  - n-best is not \textit{really} n-best because of pruning which breaks convergence guarantees [Liu and Huang, 2014]
Lots of open problems

- Evaluation metrics related:
  - MIRA, PRO and Perceptron require sentence level metric (BLEU doesn't work well)
  - use good metrics
  - but good metrics oftend are not good for tuning

- Representation of space of translations:
  - n-best list is too small (compared to exponential space)
  - lattice and hyper-graph are better options but too complicated to use because metrics don’t decompose to (hyper-)arcs
  - n-best is not really n-best because of pruning which breaks convergence guarantees [Liu and Huang, 2014]

- Optimization itself:
Lots of open problems

- Evaluation metrics related:
  - MIRA, PRO and Perceptron require sentence level metric (BLEU doesn’t work well)
  - use good metrics
  - but good metrics oftend are not good for tuning

- Representation of space of translations:
  - n-best list is too small (compared to exponential space)
  - lattice and hyper-graph are better options but too complicated to use because metrics don’t decompose to (hyper-)arcs
  - n-best is not really n-best because of pruning which breaks convergence guarantees [Liu and Huang, 2014]

- Optimization itself:
Lots of open problems

▶ Evaluation metrics related:
  ▶ MIRA, PRO and Perceptron require sentence level metric (BLEU doesn’t work well)
  ▶ use good metrics
  ▶ but good metrics often are not good for tuning

▶ Representation of space of translations:
  ▶ n-best list is too small (compared to exponential space)
  ▶ lattice and hyper-graph are better options but too complicated to use because metrics don’t decompose to (hyper-)arcs
  ▶ n-best is not really n-best because of pruning which breaks convergence guarantees [Liu and Huang, 2014]

▶ Optimization itself:
  ▶ increase margin? minimize risk?
Lots of open problems

▶ Evaluation metrics related:
  ▶ MIRA, PRO and Perceptron require sentence level metric (BLEU doesn’t work well)
  ▶ use good metrics
  ▶ but good metrics oftend are not good for tuning

▶ Representation of space of translations:
  ▶ n-best list is too small (compared to exponential space)
  ▶ lattice and hyper-graph are better options but too complicated to use because metrics don’t decompose to (hyper-)arcs
  ▶ n-best is not really n-best because of pruning which breaks convergence guarantees [Liu and Huang, 2014]

▶ Optimization itself:
  ▶ increase margin? minimize risk?
  ▶ latent variables (towards which derivation to optimize?)
Tuning task

- So many things to choose in tuning (metric, algorithm, data, features...)

Final performance usually measured by BLEU and not humans

Organised Tuning Task on WMT15 to explore these options in proper way
Tuning task

- So many things to choose in tuning (metric, algorithm, data, features...)
- Final performance usually measured by BLEU and not humans
Tuning task

- So many things to choose in tuning (metric, algorithm, data, features...)
- Final performance usually measured by BLEU and not humans
- Organised Tuning Task on WMT15 to explore these options in proper way
Tuning task - system for tuning

- Hiero Moses trained both for English-Czech and Czech-English on small dataset
Tuning task - system for tuning

- Hiero Moses trained both for English-Czech and Czech-English on small dataset
- constrained version allowed 2000 sentence pairs for tuning
Tuning task - system for tuning

- Hiero Moses trained both for English-Czech and Czech-English on small dataset
- constrained version allowed 2000 sentence pairs for tuning
- constrained version allowed only dense features
Tuning task - system for tuning

- Hiero Moses trained both for English-Czech and Czech-English on small dataset
- constrained version allowed 2000 sentence pairs for tuning
- constrained version allowed only dense features
- any tuning algorithm or metric tuning was allowed (even manually setting weights)
Czech-English results

<table>
<thead>
<tr>
<th>System Name</th>
<th>TrueSkill Score</th>
<th>BLEU</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Tuning-Only</td>
<td>All</td>
</tr>
<tr>
<td>bleu-MIRA-dense</td>
<td>0.153</td>
<td>-0.182</td>
</tr>
<tr>
<td>ILLC-UvA</td>
<td>0.108</td>
<td>-0.189</td>
</tr>
<tr>
<td>bleu-MERT-dense</td>
<td>0.087</td>
<td>-0.196</td>
</tr>
<tr>
<td>AFRL</td>
<td>0.070</td>
<td>-0.210</td>
</tr>
<tr>
<td>USAAR-Tuna</td>
<td>0.011</td>
<td>-0.220</td>
</tr>
<tr>
<td>DCU</td>
<td>-0.027</td>
<td>-0.263</td>
</tr>
<tr>
<td>METEOR-CMU</td>
<td>-0.101</td>
<td>-0.297</td>
</tr>
<tr>
<td>bleu-MIRA-sparse</td>
<td>-0.150</td>
<td>-0.320</td>
</tr>
<tr>
<td>HKUST</td>
<td>-0.150</td>
<td>-0.320</td>
</tr>
<tr>
<td>HKUST-LATE</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

Table: Results on Czech-English tuning
## English-Czech results

<table>
<thead>
<tr>
<th>System Name</th>
<th>TrueSkill Score</th>
<th>BLEU</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Tuning-Only</td>
<td>All</td>
</tr>
<tr>
<td>DCU</td>
<td>0.320 -0.342</td>
<td>4.96</td>
</tr>
<tr>
<td>bleu-MIRA-dense</td>
<td>0.303 -0.346</td>
<td>5.31</td>
</tr>
<tr>
<td>AFRL</td>
<td>0.303 -0.342</td>
<td>5.34</td>
</tr>
<tr>
<td>USAAR-Tuna</td>
<td>0.214 -0.373</td>
<td>5.26</td>
</tr>
<tr>
<td>bleu-MERT-dense</td>
<td>0.123 -0.406</td>
<td>5.24</td>
</tr>
<tr>
<td>METEOR-CMU</td>
<td>-0.271 -0.563</td>
<td>4.37</td>
</tr>
<tr>
<td>bleu-MIRA-sparse</td>
<td>-0.992 -0.808</td>
<td>3.79</td>
</tr>
<tr>
<td>USAAR-baseline-mira</td>
<td>— —</td>
<td>5.31</td>
</tr>
<tr>
<td>USAAR-baseline-mert</td>
<td>— —</td>
<td>5.25</td>
</tr>
</tbody>
</table>

Table: Results on English-Czech tuning
Word Penalty weights for English-Czech

-1.2 -1 -0.8 -0.6 -0.4 -0.2 0 0.2 0.4
-0.3 -0.25 -0.2 -0.15 -0.1 -0.05

Difficult to analyse individual weights but if we have to...

All non-sparse systems find similar weights for WP
English-Czech PCA

- DCU
- bleu_MIRA_dense
- AFRL
- USAAR-Tuna
- bleu_MERT
- bleu_MIRA_sparse
- METEOR_CMU
Table: Loadings (correlations) of each component with each feature function for English-Czech

<table>
<thead>
<tr>
<th>Component</th>
<th>PC1</th>
<th>PC2</th>
</tr>
</thead>
<tbody>
<tr>
<td>LM0</td>
<td>-0.69</td>
<td>0.44</td>
</tr>
<tr>
<td>PhrasePenalty0</td>
<td>0.15</td>
<td>-0.63</td>
</tr>
<tr>
<td>TranslationModel0_0</td>
<td>-0.91</td>
<td>-0.13</td>
</tr>
<tr>
<td>TranslationModel0_1</td>
<td>0.91</td>
<td>-0.03</td>
</tr>
<tr>
<td>TranslationModel0_2</td>
<td>-0.55</td>
<td>0.72</td>
</tr>
<tr>
<td>TranslationModel0_3</td>
<td>0.36</td>
<td>0.75</td>
</tr>
<tr>
<td>TranslationModel1</td>
<td>0.42</td>
<td>0.84</td>
</tr>
<tr>
<td>WordPenalty0</td>
<td>0.84</td>
<td>0.27</td>
</tr>
</tbody>
</table>
Czech-English PCA

- No obvious pattern
- Very similar systems perform completely differently
- Very different systems perform similarly
Conclusion

- Tuning is a **standard procedure** of most modern MT systems
Conclusion

- Tuning is a **standard procedure** of most modern MT systems
- But still **difficult** in many respects
Conclusion

- Tuning is a **standard procedure** of most modern MT systems
- But still **difficult** in many respects
- **Tuning Task** will happen on again WMT16
Conclusion

- Tuning is a **standard procedure** of most modern MT systems
- But still **difficult** in many respects
- **Tuning Task** will happen on again WMT16
- Questions?
Bibliography I

Cherry, C. and Foster, G. (2012).
Batch tuning strategies for statistical machine translation.

Online large-margin training of syntactic and structural translation features.


Tuning As Ranking.

Statistical Machine Translation.

Liu, L. and Huang, L. (2014).
Search-aware tuning for machine translation.
Minimum error rate training in statistical machine translation.

Online large-margin training for statistical machine translation.