Discriminative Training

MT Marathon lecture

Miloš Stanojević

ILLC, University of Amsterdam

May 2, 2017

Going from Generative to Discriminative models

Start with generative noisy channel model:

$$t^* = \operatorname*{argmax}_{t \in T(s)} p(t|s)$$

Going from Generative to Discriminative models

Start with generative noisy channel model:

$$t^* = \operatorname*{argmax}_{t \in T(s)} p(t|s) = \operatorname*{argmax}_{t \in T(s)} \frac{p(s|t)p(t)}{p(s)}$$

Going from Generative to Discriminative models

Start with generative noisy channel model:

$$t^* = \operatorname*{argmax}_{t \in T(s)} p(t|s) = \operatorname*{argmax}_{t \in T(s)} \frac{p(s|t)p(t)}{p(s)} = \operatorname*{argmax}_{t \in T(s)} p(s|t)p(t)$$

Going from Generative to Discriminative models

Start with generative noisy channel model:

$$t^* = \underset{t \in T(s)}{\operatorname{argmax}}\ p(t|s) = \underset{t \in T(s)}{\operatorname{argmax}}\ \frac{p(s|t)p(t)}{p(s)} = \underset{t \in T(s)}{\operatorname{argmax}}\ p(s|t)p(t)$$

$$= \underset{t \in T(s)}{\operatorname{argmax}}\ \log p(s|t) + \log p(t)$$

Going from Generative to Discriminative models

Start with generative noisy channel model:

$$t^* = \operatorname*{argmax}_{t \in T(s)} p(t|s) = \operatorname*{argmax}_{t \in T(s)} \frac{p(s|t)p(t)}{p(s)} = \operatorname*{argmax}_{t \in T(s)} p(s|t)p(t)$$

$$= \operatorname*{argmax}_{t \in T(s)} \log p(s|t) + \log p(t)$$

$$= \operatorname*{argmax}_{t \in T(s)} \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} \log p(s|t) \\ \log p(t) \end{bmatrix}$$

Going from Generative to Discriminative models

Start with generative noisy channel model:

$$
\begin{aligned}
t^* &= \operatorname*{argmax}_{t \in T(s)} p(t|s) = \operatorname*{argmax}_{t \in T(s)} \frac{p(s|t)p(t)}{p(s)} = \operatorname*{argmax}_{t \in T(s)} p(s|t)p(t) \\
&= \operatorname*{argmax}_{t \in T(s)} \log p(s|t) + \log p(t) \\
&= \operatorname*{argmax}_{t \in T(s)} \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} \log p(s|t) \\ \log p(t) \end{bmatrix} \\
&= \boxed{\operatorname*{argmax}_{t \in T(s)} \lambda^T \mathbf{h}(s,t)}
\end{aligned}
$$

end with linear discriminative model

Going from Generative to Discriminative models

Start with generative noisy channel model:

$$
\begin{aligned}
t^* &= \underset{t \in T(s)}{\operatorname{argmax}}\; p(t|s) = \underset{t \in T(s)}{\operatorname{argmax}}\; \frac{p(s|t)p(t)}{p(s)} = \underset{t \in T(s)}{\operatorname{argmax}}\; p(s|t)p(t) \\
&= \underset{t \in T(s)}{\operatorname{argmax}}\; \log p(s|t) + \log p(t) \\
&= \underset{t \in T(s)}{\operatorname{argmax}}\; \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} \log p(s|t) \\ \log p(t) \end{bmatrix} \\
&= \boxed{\underset{t \in T(s)}{\operatorname{argmax}}\; \lambda^T \mathbf{h}(s, t)} \qquad \text{end with linear discriminative model}
\end{aligned}
$$

Why would we want to do this?

Going from Generative to Discriminative models

Start with generative noisy channel model:

$$
\begin{aligned}
t^* &= \operatorname*{argmax}_{t \in T(s)} p(t|s) = \operatorname*{argmax}_{t \in T(s)} \frac{p(s|t)p(t)}{p(s)} = \operatorname*{argmax}_{t \in T(s)} p(s|t)p(t) \\
&= \operatorname*{argmax}_{t \in T(s)} \log p(s|t) + \log p(t) \\
&= \operatorname*{argmax}_{t \in T(s)} \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} \log p(s|t) \\ \log p(t) \end{bmatrix} \\
&= \boxed{\operatorname*{argmax}_{t \in T(s)} \lambda^T \mathbf{h}(s,t)} \qquad \text{end with linear discriminative model}
\end{aligned}
$$

Why would we want to do this?

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

Going from Generative to Discriminative models

Start with generative noisy channel model:

$$
\begin{aligned}
t^* &= \operatorname*{argmax}_{t \in T(s)} p(t|s) = \operatorname*{argmax}_{t \in T(s)} \frac{p(s|t)p(t)}{p(s)} = \operatorname*{argmax}_{t \in T(s)} p(s|t)p(t) \\
&= \operatorname*{argmax}_{t \in T(s)} \log p(s|t) + \log p(t) \\
&= \operatorname*{argmax}_{t \in T(s)} \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} \log p(s|t) \\ \log p(t) \end{bmatrix} \\
&= \boxed{\operatorname*{argmax}_{t \in T(s)} \lambda^T \mathbf{h}(s,t)} \qquad \text{end with linear discriminative model}
\end{aligned}
$$

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:

$$
\begin{aligned}
t^* &= \underset{t \in T(s)}{\operatorname{argmax}}\; p(t|s) = \underset{t \in T(s)}{\operatorname{argmax}}\; \frac{p(s|t)p(t)}{p(s)} = \underset{t \in T(s)}{\operatorname{argmax}}\; p(s|t)p(t) \\
&= \underset{t \in T(s)}{\operatorname{argmax}}\; \log p(s|t) + \log p(t) \\
&= \underset{t \in T(s)}{\operatorname{argmax}}\; \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} \log p(s|t) \\ \log p(t) \end{bmatrix} \\
&= \boxed{\underset{t \in T(s)}{\operatorname{argmax}}\; \lambda^T \mathbf{h}(s,t)} \qquad \text{end with linear discriminative model}
\end{aligned}
$$

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:

$$t^* = \operatorname*{argmax}_{t \in T(s)} p(t|s) = \operatorname*{argmax}_{t \in T(s)} \frac{p(s|t)p(t)}{p(s)} = \operatorname*{argmax}_{t \in T(s)} p(s|t)p(t)$$

$$= \operatorname*{argmax}_{t \in T(s)} \log p(s|t) + \log p(t)$$

$$= \operatorname*{argmax}_{t \in T(s)} \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} \log p(s|t) \\ \log p(t) \end{bmatrix}$$

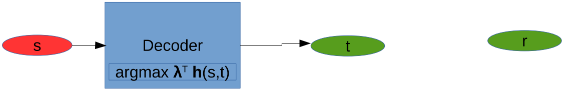
$$= \boxed{\operatorname*{argmax}_{t \in T(s)} \lambda^T \mathbf{h}(s,t)}$$ end with linear discriminative model

Why would we want to do this?

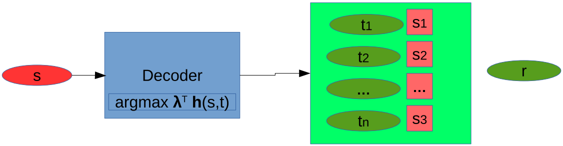
- We can add more indicators (features) of good translation
- We can give different weight to different features
- And all this done in a way to directly optimize desired metric

Disadvantage? Losing probabilistic interpretation

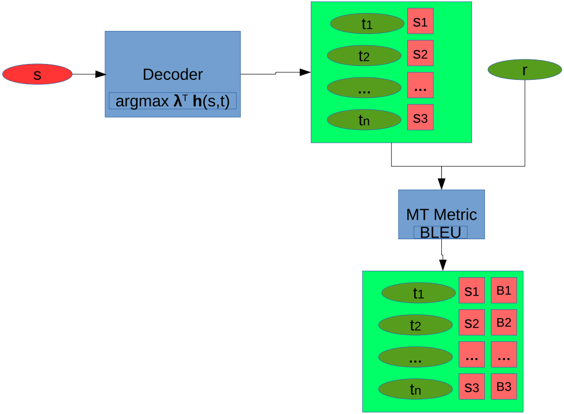
Optimize for BLEU directly



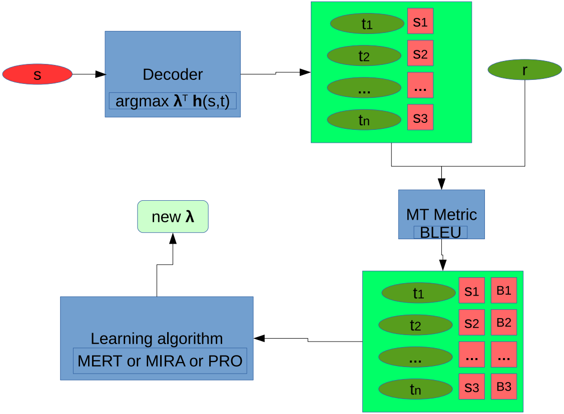
Optimize for BLEU directly



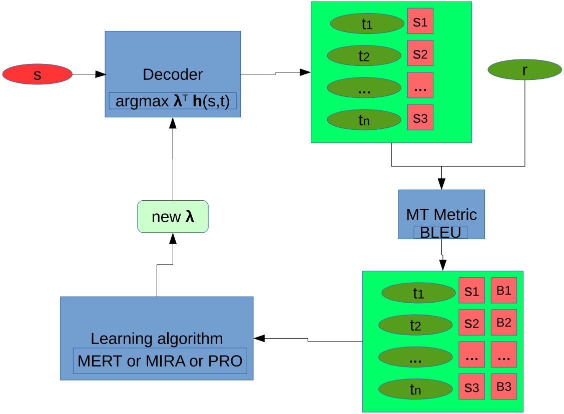
Optimize for BLEU directly



Optimize for BLEU directly



Optimize for BLEU directly



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.

$$\begin{aligned} score(s,t) &= \lambda^T \mathbf{h}(s,t) \\ &= \sum_i \lambda_i h_i(s,t) \end{aligned}$$

MERT[Och, 2003]

MERT optimizes only one parameter while keeping others fixed.

$$
\begin{aligned}
score(s,t) &= \lambda^T \mathbf{h}(s,t) \\
&= \sum_i \lambda_i h_i(s,t) \\
&= \lambda_c h_c(s,t) + \sum_{i \neq c} \lambda_i h_i(s,t)
\end{aligned}
$$

MERT[Och, 2003]

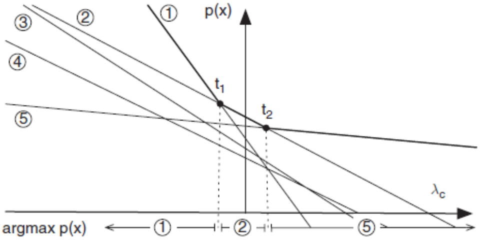
MERT optimizes only one parameter while keeping others fixed.

$$
\begin{aligned}
score(s,t) &= \lambda^T \mathbf{h}(s,t) \\
&= \sum_i \lambda_i h_i(s,t) \\
&= \lambda_c h_c(s,t) + \sum_{i \neq c} \lambda_i h_i(s,t) \\
&= \lambda_c h_c(s,t) + u_c(s,t)
\end{aligned}
$$

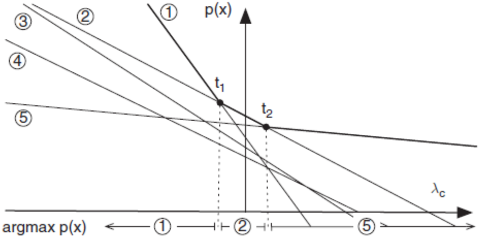
MERT[Och, 2003]

MERT optimizes only one parameter while keeping others fixed.

$$
\begin{aligned}
score(s,t) &= \lambda^T \mathbf{h}(s,t) \\
&= \sum_i \lambda_i h_i(s,t) \\
&= \lambda_c h_c(s,t) + \sum_{i \neq c} \lambda_i h_i(s,t) \\
&= \lambda_c h_c(s,t) + u_c(s,t)
\end{aligned}
$$

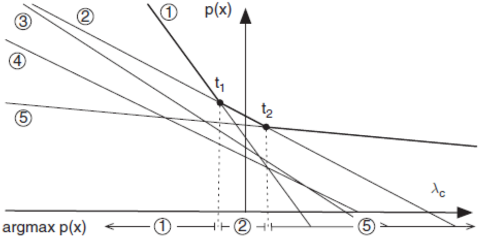


MERT[Och, 2003]



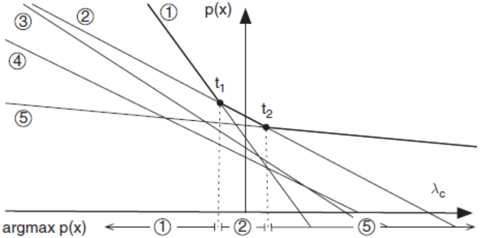
- Extract all **threshold points** where argmax changes

MERT[Och, 2003]



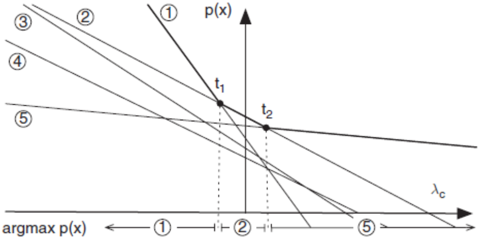
- Extract all **threshold points** where argmax changes
- Evaluate each set of threshold points with BLEU score

MERT[Och, 2003]



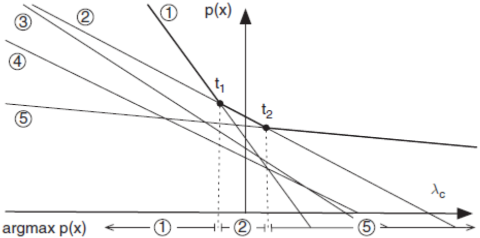
- Extract all **threshold points** where argmax changes
- Evaluate each set of threshold points with BLEU score
- Take the best one and then go again trough the decoding loop

MERT[Och, 2003]



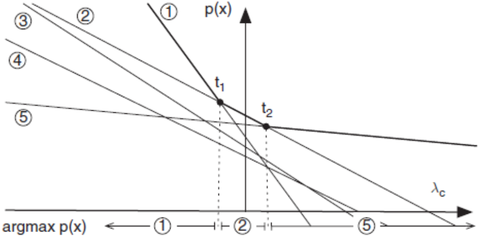
$$score(s, t_1) = score(s, t_2)$$

MERT[Och, 2003]



$$score(s, t_1) = score(s, t_2)$$
$$\lambda_c h_c(s, t_1) + u_c(s, t_1) = \lambda_c h_c(s, t_2) + u_c(s, t_2)$$

MERT[Och, 2003]

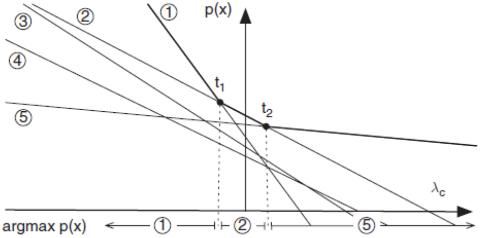


$$score(s, t_1) = score(s, t_2)$$

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

$$\lambda_c = \frac{u_c(s, t_1) - u_c(s, t_2)}{h_c(s, t_2) - h_c(s, t_1)}$$

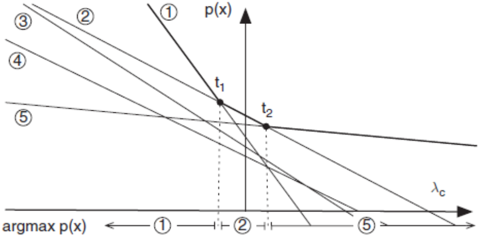
MERT[Och, 2003]



Few more tricks:

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

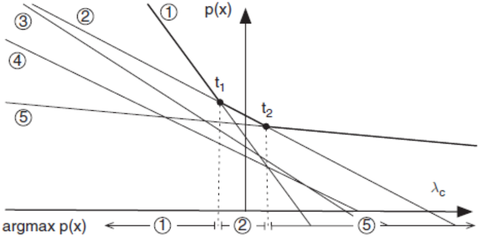
MERT[Och, 2003]



Few more tricks:

- We can speed up this by looking for top threshold points start with the steepest line (smallest $h_c(s, t_1)$)
  $score(x) = \lambda_c \, h_c(s, t_1) + u_c(s, t_1)$
  and find the most negative threshold point for that line
- Accumulate n-best lists over different decoder runs

MERT[Och, 2003]



Few more tricks:

- We can speed up this by looking for top threshold points start with the steepest line (smallest $h_c(s, t_1)$)
  $score(x) = \lambda_c\, h_c(s, t_1) + u_c(s, t_1)$
  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_{better}, t_{worse}) where $eval(t_{better}, r) > eval(t_{worse}, r)$

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_{better}, t_{worse}) where $eval(t_{better}, r) > eval(t_{worse}, r)$
For each pair

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

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_{better}, t_{worse}) where $eval(t_{better}, r) > eval(t_{worse}, r)$
For each pair

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

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_{better}, t_{worse}) where $eval(t_{better}, r) > eval(t_{worse}, r)$
For each pair

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

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_{better}, t_{worse}) where $eval(t_{better}, r) > eval(t_{worse}, r)$
For each pair

$$
\begin{aligned}
score(s, t_{better}) &> score(s, t_{worse}) \\
\lambda^T \mathbf{h}(s, t_{better}) &> \lambda^T \mathbf{h}(s, t_{worse}) \\
\lambda^T (\mathbf{h}(s, t_{better}) - \mathbf{h}(s, t_{worse})) &> 0 \\
\lambda^T (\mathbf{h}(s, t_{worse}) - \mathbf{h}(s, t_{better})) &< 0
\end{aligned}
$$

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_{better}, t_{worse}) where $eval(t_{better}, r) > eval(t_{worse}, r)$
For each pair

$$
\begin{aligned}
score(s, t_{better}) &> score(s, t_{worse}) \\
\lambda^T \mathbf{h}(s, t_{better}) &> \lambda^T \mathbf{h}(s, t_{worse}) \\
\lambda^T (\boxed{\mathbf{h}(s, t_{better}) - \mathbf{h}(s, t_{worse})}) &> 0 \\
\lambda^T (\boxed{\mathbf{h}(s, t_{worse}) - \mathbf{h}(s, t_{better})}) &< 0
\end{aligned}
$$

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

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_{better}, t_{worse}) where $eval(t_{better}, r) > eval(t_{worse}, r)$
For each pair

$$
\begin{aligned}
score(s, t_{better}) &> score(s, t_{worse}) \\
\lambda^T \mathbf{h}(s, t_{better}) &> \lambda^T \mathbf{h}(s, t_{worse}) \\
\lambda^T (\boxed{\mathbf{h}(s, t_{better}) - \mathbf{h}(s, t_{worse})}) &> 0 \\
\lambda^T (\boxed{\mathbf{h}(s, t_{worse}) - \mathbf{h}(s, t_{better})}) &< 0
\end{aligned}
$$

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_{better}, t_{worse}) where $eval(t_{better}, r) > eval(t_{worse}, r)$
For each pair

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

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]

$$t_{hope} = \operatorname*{argmax}_t \; score(s, t) + eval(t, r)$$
$$t_{fear} = \operatorname*{argmax}_t \; score(s, t) - eval(t, r)$$

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

MIRA

$$t_{hope} = \operatorname*{argmax}_{t} \, score(s,t) + eval(t,r)$$
$$t_{fear} = \operatorname*{argmax}_{t} \, score(s,t) - eval(t,r)$$

MIRA

$$
\begin{aligned}
t_{hope} &= \operatorname*{argmax}_t \; score(s,t) + eval(t,r) \\
t_{fear} &= \operatorname*{argmax}_t \; score(s,t) - eval(t,r) \\
margin &= score(s, t_{fear}) - score(s, t_{hope})
\end{aligned}
$$

MIRA

$$
\begin{aligned}
t_{hope} &= \operatorname*{argmax}_t \; score(s,t) + eval(t,r) \\
t_{fear} &= \operatorname*{argmax}_t \; score(s,t) - eval(t,r) \\
margin &= score(s, t_{fear}) - score(s, t_{hope}) \\
cost &= BLEU(t_{hope}, r) - BLEU(t_{fear}, r)
\end{aligned}
$$

MIRA

$$\begin{aligned}
t_{hope} &= \operatorname*{argmax}_t \, score(s,t) + eval(t,r) \\
t_{fear} &= \operatorname*{argmax}_t \, score(s,t) - eval(t,r) \\
margin &= score(s, t_{fear}) - score(s, t_{hope}) \\
cost &= BLEU(t_{hope}, r) - BLEU(t_{fear}, r) \\
\lambda &\leftarrow \lambda + \delta(\mathbf{h}(s, t_{hope}) - \mathbf{h}(s, t_{fear}))
\end{aligned}$$

MIRA

$$
\begin{aligned}
t_{hope} &= \operatorname*{argmax}_{t}\ score(s,t) + eval(t,r) \\
t_{fear} &= \operatorname*{argmax}_{t}\ score(s,t) - eval(t,r) \\
margin &= score(s,t_{fear}) - score(s,t_{hope}) \\
cost &= BLEU(t_{hope},r) - BLEU(t_{fear},r) \\
\lambda &\leftarrow \lambda + \delta(\mathbf{h}(s,t_{hope}) - \mathbf{h}(s,t_{fear})) \\
\delta &= min\left(C, \frac{margin + cost}{||h(s,t_{hope}) - h(s,t_{fear})||^2}\right)
\end{aligned}
$$

δ changes (unlike in Perceptron) to increase the margin

MIRA

$$t_{hope} = \operatorname*{argmax}_t \; score(s,t) + eval(t,r)$$
$$t_{fear} = \operatorname*{argmax}_t \; score(s,t) - eval(t,r)$$
$$margin = score(s, t_{fear}) - score(s, t_{hope})$$
$$cost = BLEU(t_{hope}, r) - BLEU(t_{fear}, r)$$
$$\lambda \leftarrow \lambda + \delta(\mathbf{h}(s, t_{hope}) - \mathbf{h}(s, t_{fear}))$$
$$\delta = min\left(C, \frac{margin + cost}{||h(s, t_{hope}) - h(s, t_{fear})||^2}\right)$$

δ changes (unlike in Perceptron) to increase the margin
Repeat this many times until convergence in n-best list

MIRA

$$t_{hope} = \operatorname*{argmax}_t \; score(s,t) + eval(t,r)$$

$$t_{fear} = \operatorname*{argmax}_t \; score(s,t) - eval(t,r)$$

$$margin = score(s, t_{fear}) - score(s, t_{hope})$$

$$cost = BLEU(t_{hope}, r) - BLEU(t_{fear}, r)$$

$$\lambda \leftarrow \lambda + \delta(\mathbf{h}(s, t_{hope}) - \mathbf{h}(s, t_{fear}))$$

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

δ changes (unlike in Perceptron) to increase the margin
Repeat this many times until convergence in n-best list
Repeat this with the loop trough 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)

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 *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

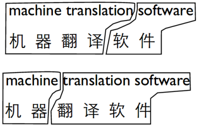
- Evaluation metrics related:
  - MIRA, PRO and Perceptron require sentence level metric (BLEU doesn't work well)
  - use good metrics
  - but good metrics oftend are not good for tuning
- Representation of space of translations:
  - n-best list is too small (compared to exponential space)
  - lattice 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 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?

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...)

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

System Name	TrueSkill Score Tuning-Only	TrueSkill Score All	BLEU
BLEU-MIRA-DENSE	0.153	-0.182	12.28
ILLC-UvA	0.108	-0.189	12.05
BLEU-MERT-DENSE	0.087	-0.196	12.11
AFRL	0.070	-0.210	12.20
USAAR-Tuna	0.011	-0.220	12.16
DCU	-0.027	-0.263	11.44
METEOR-CMU	-0.101	-0.297	10.88
BLEU-MIRA-SPARSE	-0.150	-0.320	10.84
HKUST	-0.150	-0.320	10.99
HKUST-LATE	—	—	12.20

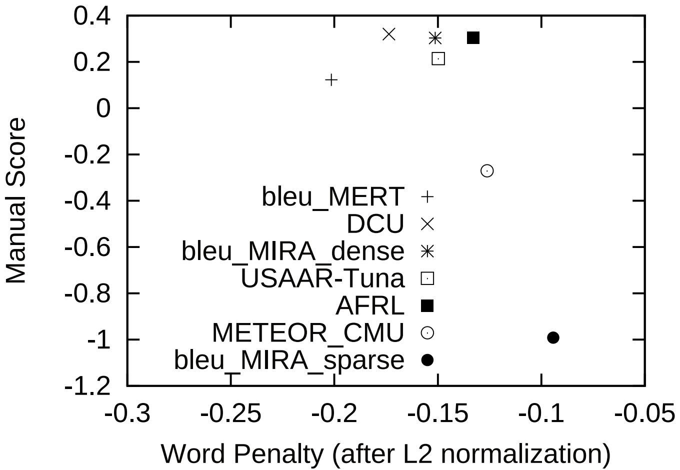
Table : Results on Czech-English tuning

English-Czech results

System Name	TrueSkill Score		BLEU
	Tuning-Only	All	
DCU	0.320	-0.342	4.96
BLEU-MIRA-DENSE	0.303	-0.346	5.31
AFRL	0.303	-0.342	5.34
USAAR-TUNA	0.214	-0.373	5.26
BLEU-MERT-DENSE	0.123	-0.406	5.24
METEOR-CMU	-0.271	-0.563	4.37
BLEU-MIRA-SPARSE	-0.992	-0.808	3.79
USAAR-BASELINE-MIRA	—	—	5.31
USAAR-BASELINE-MERT	—	—	5.25

Table : Results on English-Czech tuning

Word Penalty weights for English-Czech



- Difficult to analyse individual weights but if we have to...
- All non-sparse systems find similar weights for WP

English-Czech PCA

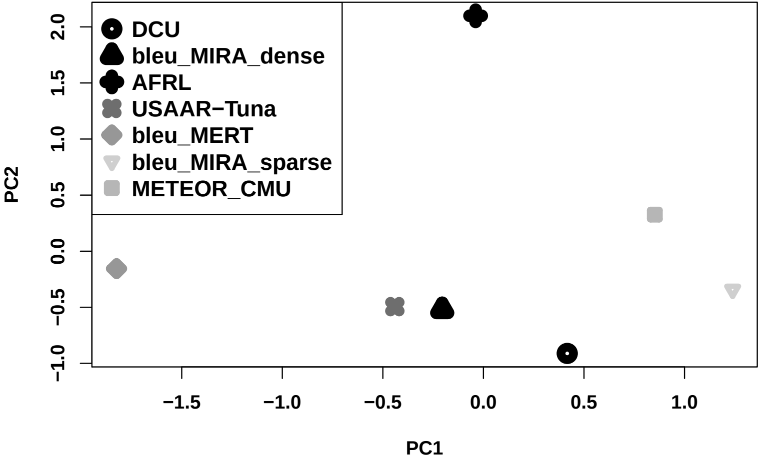
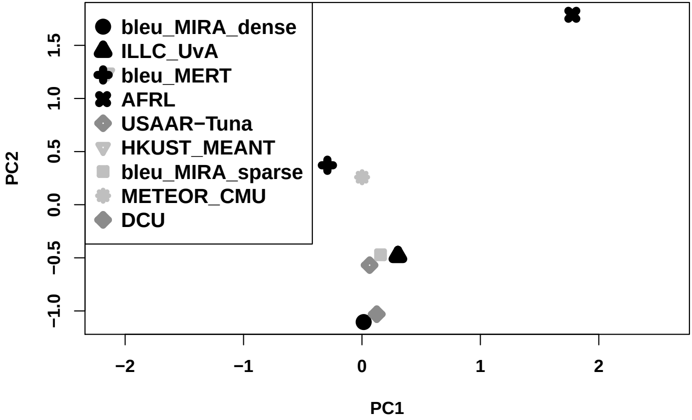


Table of contents

	PC1	PC2
LM0	**-0.69**	0.44
PhrasePenalty0	0.15	**-0.63**
TranslationModel0_0	**-0.91**	-0.13
TranslationModel0_1	**0.91**	-0.03
TranslationModel0_2	-0.55	**0.72**
TranslationModel0_3	0.36	**0.75**
TranslationModel1	0.42	**0.84**
WordPenalty0	**0.84**	0.27

Table : Loadings (correlations) of each component with each feature function for English-Czech

Czech-English PCA



- No obvious pattern
- Very similar systems perform complitely 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.
In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, NAACL HLT '12, pages 427–436, Stroudsburg, PA, USA. Association for Computational Linguistics.

Chiang, D., Marton, Y., and Resnik, P. (2008).
Online large-margin training of syntactic and structural translation features.
In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 224–233. Association for Computational Linguistics.

Clark, J. H., Dyer, C., Lavie, A., and Smith, N. A. (2011).
Better Hypothesis Testing for Statistical Machine Translation: Controlling for Optimizer Instability.
In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: Short Papers - Volume 2*, HLT '11, pages 176–181, Stroudsburg, PA, USA. Association for Computational Linguistics.

Hopkins, M. and May, J. (2011).
Tuning As Ranking.
In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '11, pages 1352–1362, Stroudsburg, PA, USA. Association for Computational Linguistics.

Koehn, P. (2010).
Statistical Machine Translation.
Cambridge University Press, New York, NY, USA, 1st edition.

Liu, L. and Huang, L. (2014).
Search-aware tuning for machine translation.
In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1942–1952, Doha, Qatar. Association for Computational Linguistics.

Bibliography II

Och, F. J. (2003).
Minimum error rate training in statistical machine translation.
In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 1*, ACL '03, pages 160–167, Stroudsburg, PA, USA. Association for Computational Linguistics.

Watanabe, T., Suzuki, J., Tsukada, H., and Isozaki, H. (2007).
Online large-margin training for statistical machine translation.
In *In Proc. of EMNLP*. Citeseer.