Joint Feature Selection in Distributed Stochastic Learning for Large-Scale Discriminative SMT

Patrick Simianer*, Stefan Riezler*, Chris Dyer†

* Department of Computational Linguistics, Heidelberg University, Germany
† Language Technologies Institute, Carnegie Mellon University, Pittsburgh, PA
Discriminative training in SMT

- Machine learning theory and practice suggests **benefits from tuning on large training samples**.
- Discriminative training in SMT has been content with tuning weights for **large feature sets on small development data**.
- Why is this?
  - Manually designed “error-correction features” (Chiang et al. NAACL’09) can be tuned well on small datasets.
  - “Syntactic constraint” features (Marton and Resnik ACL’08) don’t scale well to large data sets.
  - “Special” overfitting problem in stochastic learning: Weight updates may not generalize well beyond example considered in update.
Discriminative training in SMT

- Machine learning theory and practice suggests benefits from tuning on large training samples.
- Discriminative training in SMT has been content with tuning weights for large feature sets on small development data.
- Why is this?
  - Manually designed “error-correction features” (Chiang et al. NAACL’09) can be tuned well on small datasets.
  - “Syntactic constraint” features (Marton and Resnik ACL’08) don’t scale well to large data sets.
  - “Special” overfitting problem in stochastic learning: Weight updates may not generalize well beyond example considered in update.
Discriminative training in SMT

- Machine learning theory and practice suggests benefits from tuning on large training samples.
- Discriminative training in SMT has been content with tuning weights for large feature sets on small development data.
- Why is this?
  - Manually designed “error-correction features” (Chiang et al. NAACL’09) can be tuned well on small datasets.
  - “Syntactic constraint” features (Marton and Resnik ACL’08) don’t scale well to large data sets.
  - “Special” overfitting problem in stochastic learning: Weight updates may not generalize well beyond example considered in update.
Discriminative training in SMT

- Machine learning theory and practice suggests benefits from tuning on large training samples.
- Discriminative training in SMT has been content with tuning weights for large feature sets on small development data.
- Why is this?
  - Manually designed “error-correction features” (Chiang et al. NAACL’09) can be tuned well on small datasets.
  - “Syntactic constraint” features (Marton and Resnik ACL’08) don’t scale well to large data sets.
  - “Special” overfitting problem in stochastic learning: Weight updates may not generalize well beyond example considered in update.
Discriminative training in SMT

- Machine learning theory and practice suggests benefits from tuning on large training samples.
- Discriminative training in SMT has been content with tuning weights for large feature sets on small development data.
- Why is this?
  - Manually designed “error-correction features” (Chiang et al. NAACL’09) can be tuned well on small datasets.
  - “Syntactic constraint” features (Marton and Resnik ACL’08) don’t scale well to large data sets.
  - “Special” overfitting problem in stochastic learning: Weight updates may not generalize well beyond example considered in update.
Discriminative training in SMT

- Machine learning theory and practice suggests **benefits from tuning on large training samples.**
- Discriminative training in SMT has been content with tuning weights for **large feature sets** on **small development data.**
- Why is this?
  - Manually designed “error-correction features” (Chiang et al. NAACL’09) can be tuned well on small datasets.
  - “Syntactic constraint” features (Marton and Resnik ACL’08) don’t scale well to large data sets.
  - “Special” overfitting problem in stochastic learning: Weight updates may not generalize well beyond example considered in update.
Our goal: Tuning SMT on the training set

- Research question: Is it possible to benefit from scaling discriminative training for SMT to large training sets?

- Our approach:
  - Deploy generic local features that can be read off efficiently from rules at runtime.
  - Combine distributed stochastic learning with feature selection inspired by multi-task learning.

- Results:
  - Feature selection is key for efficiency and quality when tuning on the training set.
  - Significant improvements over tuning large features sets on small dev set and over tuning on training data without $l_1/l_2$-based feature selection.
Our goal: Tuning SMT on the training set

- Research question: Is it possible to benefit from scaling discriminative training for SMT to large training sets?
- Our approach:
  - Deploy **generic local features** that can be read off efficiently from rules at runtime.
  - Combine **distributed stochastic learning** with **feature selection** inspired by multi-task learning.
- Results:
  - **Feature selection is key** for efficiency and quality when tuning on the training set.
  - **Significant improvements** over tuning large features sets on small dev set and over tuning on training data without $l_1/l_2$-based feature selection.
Our goal: Tuning SMT on the training set

• Research question: Is it possible to benefit from scaling discriminative training for SMT to large training sets?

• Our approach:
  • Deploy **generic local features** that can be read off efficiently from rules at runtime.
  • Combine **distributed stochastic learning** with **feature selection inspired by multi-task learning**.

• Results:
  • **Feature selection is key** for efficiency and quality when tuning on the training set.
  • **Significant improvements** over tuning large features sets on small dev set and over tuning on training data without \( l_1/l_2 \)-based feature selection.
Our goal: Tuning SMT on the training set

- Research question: Is it possible to benefit from scaling discriminative training for SMT to large training sets?
- Our approach:
  - Deploy **generic local features** that can be read off efficiently from rules at runtime.
  - Combine **distributed stochastic learning** with **feature selection inspired by multi-task learning**.
- Results:
  - **Feature selection is key** for efficiency and quality when tuning on the training set.
  - **Significant improvements** over tuning large features sets on small dev set and over tuning on training data without $l_1/l_2$-based feature selection.
Our goal: Tuning SMT on the training set

- Research question: Is it possible to benefit from scaling discriminative training for SMT to large training sets?
- Our approach:
  - Deploy **generic local features** that can be read off efficiently from rules at runtime.
  - Combine **distributed stochastic learning** with **feature selection** inspired by multi-task learning.
- Results:
  - **Feature selection is key** for efficiency and quality when tuning on the training set.
  - **Significant improvements** over tuning large features sets on small dev set and over tuning on training data without $\ell_1/\ell_2$-based feature selection.
Related work

- Many approaches to discriminative training in last ten years.
- Mostly “large scale” means feature sets of size $\leq 10K$, tuning on development data of size $2K$.
- Notable exceptions:
  - Liang et al. ACL’06: 1.5M features, 67K parallel sentences.
  - Tillmann and Zhang ACL’06: 35M features, 230K parallel sentences.
  - Blunsom et al. ACL’08: 7.8M features, 100K sentences.
- Inspiration for our work: Duh et al. WMT’10 use 500 100-best lists for multi-task learning of 2.4M features.
Related work

- Many approaches to discriminative training in last ten years.
- Mostly “large scale” means feature sets of size $\leq 10K$, tuning on development data of size $2K$.
- Notable exceptions:
  - Liang et al. ACL’06: 1.5M features, 67K parallel sentences.
  - Tillmann and Zhang ACL’06: 35M features, 230K parallel sentences.
  - Blunsom et al. ACL’08: 7.8M features, 100K sentences.
- Inspiration for our work: Duh et al. WMT’10 use 500 100-best lists for multi-task learning of 2.4M features.
Related work

• Many approaches to discriminative training in last ten years.
• Mostly “large scale” means feature sets of size $\leq 10K$, tuning on development data of size $2K$.
• Notable exceptions:
  • Liang et al. ACL’06: 1.5M features, 67K parallel sentences.
  • Tillmann and Zhang ACL’06: 35M features, 230K parallel sentences.
  • Blunsom et al. ACL’08: 7.8M features, 100K sentences.
• Inspiration for our work: Duh et al. WMT’10 use 500 100-best lists for multi-task learning of 2.4M features.
Related work

• Many approaches to discriminative training in last ten years.
• Mostly “large scale” means feature sets of size $\leq 10K$, tuning on development data of size $2K$.
• Notable exceptions:
  • Liang et al. ACL’06: 1.5M features, 67K parallel sentences.
  • Tillmann and Zhang ACL’06: 35M features, 230K parallel sentences.
  • Blunsom et al. ACL’08: 7.8M features, 100K sentences.
• Inspiration for our work: Duh et al. WMT’10 use 500 100-best lists for multi-task learning of 2.4M features.
Local features for SCFGs

(1) $X \rightarrow X_1 \text{ hat } X_2 \text{ versprochen}; X_1 \text{ promised } X_2$
(2) $X \rightarrow X_1 \text{ hat mir } X_2 \text{ versprochen; } X_1 \text{ promised me } X_2$
(3) $X \rightarrow X_1 \text{ versprach } X_2; X_1 \text{ promised } X_2$

- **Rule identifiers** for SCFG productions
  Examples: rule (1), (2) and (3)
- **Rule source n-gram** features
  Examples: “$X \text{ hat}”,” “hat $X”,” “$X \text{ versprochen}”
- **Rule shape** features
  Examples: (NT, term*, NT, term*; NT, term*, NT) for (1), (2); (NT, term*, NT; NT, term*, NT) for rule (3).
Local features for SCFGs

(1) \( X \rightarrow X_1 \text{ hat } X_2 \text{ versprochen}; X_1 \text{ promised } X_2 \)
(2) \( X \rightarrow X_1 \text{ hat mir } X_2 \text{ versprochen}; X_1 \text{ promised me } X_2 \)
(3) \( X \rightarrow X_1 \text{ versprach } X_2; X_1 \text{ promised } X_2 \)

- **Rule identifiers** for SCFG productions
  Examples: rule (1), (2) and (3)

- **Rule source n-gram** features
  Examples: “\( X \text{ hat} \)”, “\( \text{hat } X \)”, “\( X \text{ versprochen} \)"

- **Rule shape** features
  Examples: (\( \text{NT, term*}, \text{NT, term*}; \text{NT, term*}, \text{NT} \)) for (1), (2);
  (\( \text{NT, term*}, \text{NT}; \text{NT, term*}, \text{NT} \)) for rule (3).
Local features for SCFGs

(1) $X \rightarrow X_1 \text{ hat } X_2 \text{ versprochen}; X_1 \text{ promised } X_2$
(2) $X \rightarrow X_1 \text{ hat mir } X_2 \text{ versprochen; }$
    $X_1 \text{ promised me } X_2$
(3) $X \rightarrow X_1 \text{ versprach } X_2; X_1 \text{ promised } X_2$

- **Rule identifiers** for SCFG productions
  Examples: rule (1), (2) and (3)

- **Rule source n-gram** features
  Examples: “$X \text{ hat}”$, “$\text{hat } X”$, “$X \text{ versprochen}”$

- **Rule shape** features
  Examples: (NT, term*, NT, term*; NT, term*, NT) for (1), (2);
  (NT, term*, NT; NT, term*, NT) for rule (3).
Local features for SCFGs

(1) $X \rightarrow X_1 \text{ hat } X_2 \text{ versprochen}; \ X_1 \text{ promised } X_2$
(2) $X \rightarrow X_1 \text{ hat mir } X_2 \text{ versprochen};$
    $\quad X_1 \text{ promised me } X_2$
(3) $X \rightarrow X_1 \text{ versprach } X_2; \ X_1 \text{ promised } X_2$

- **Rule identifiers** for SCFG productions
  Examples: rule (1), (2) and (3)
- **Rule source n-gram** features
  Examples: “X hat”, “hat X”, “X versprochen”
- **Rule shape** features
  Examples: (NT, term*, NT, term*; NT, term*, NT) for (1), (2);
  (NT, term*, NT; NT, term*, NT) for rule (3).
Learning framework: Pairwise ranking using SGD

- Preference pairs $x_j = (x_j^{(1)}, x_j^{(2)})$ where $x_j^{(1)}$ is preferred over $x_j^{(2)}$, are defined by sorting translations $x \in \mathbb{R}^D$ by smoothed sentence-wise BLEU.

- Hinge loss-type objective

$$l_j(w) = (- \langle w, \tilde{x}_j \rangle)_+$$

where $\tilde{x}_j = x_j^{(1)} - x_j^{(2)}$, $(a)_+ = \max(0, a)$, $w \in \mathbb{R}^D$ is a weight vector, and $\langle \cdot, \cdot \rangle$ denotes the standard vector dot product.

- Ranking perceptron by stochastic subgradient descent:

$$\nabla l_j(w) = \begin{cases} -\tilde{x}_j & \text{if } \langle w, \tilde{x}_j \rangle \leq 0, \\ 0 & \text{else}. \end{cases}$$
Learning framework: Pairwise ranking using SGD

- Preference pairs $\mathbf{x}_j = (\mathbf{x}_j^{(1)}, \mathbf{x}_j^{(2)})$ where $\mathbf{x}_j^{(1)}$ is preferred over $\mathbf{x}_j^{(2)}$, are defined by sorting translations $\mathbf{x} \in \mathbb{R}^D$ by smoothed sentence-wise BLEU.

- Hinge loss-type objective

$$l_j(\mathbf{w}) = (−\langle \mathbf{w}, \tilde{\mathbf{x}}_j \rangle)_+$$

where $\tilde{\mathbf{x}}_j = \mathbf{x}_j^{(1)} − \mathbf{x}_j^{(2)}$, $(a)_+ = \max(0, a)$, $\mathbf{w} \in \mathbb{R}^D$ is a weight vector, and $\langle \cdot, \cdot \rangle$ denotes the standard vector dot product.

- Ranking perceptron by stochastic subgradient descent:

$$\nabla l_j(\mathbf{w}) = \begin{cases} -\tilde{\mathbf{x}}_j & \text{if } \langle \mathbf{w}, \tilde{\mathbf{x}}_j \rangle \leq 0, \\ 0 & \text{else.} \end{cases}$$
Learning framework: Pairwise ranking using SGD

• Preference pairs $\mathbf{x}_j = (\mathbf{x}_j^{(1)}, \mathbf{x}_j^{(2)})$ where $\mathbf{x}_j^{(1)}$ is preferred over $\mathbf{x}_j^{(2)}$, are defined by sorting translations $\mathbf{x} \in \mathbb{R}^D$ by smoothed sentence-wise BLEU.

• Hinge loss-type objective

$$l_j(\mathbf{w}) = (- \langle \mathbf{w}, \tilde{\mathbf{x}}_j \rangle)_+$$

where $\tilde{\mathbf{x}}_j = \mathbf{x}_j^{(1)} - \mathbf{x}_j^{(2)}$, $(a)_+ = \max(0, a)$, $\mathbf{w} \in \mathbb{R}^D$ is a weight vector, and $\langle \cdot, \cdot \rangle$ denotes the standard vector dot product.

• **Ranking perceptron** by stochastic subgradient descent:

$$\nabla l_j(\mathbf{w}) = \begin{cases} -\tilde{\mathbf{x}}_j & \text{if } \langle \mathbf{w}, \tilde{\mathbf{x}}_j \rangle \leq 0, \\ 0 & \text{else.} \end{cases}$$
Multipartite ranking

- Instead of training on all pairs, only compare good translations with bad ones without teasing apart small differences.
- Build pairs from levels HI-MID, HI-LOW, and MID-LOW, but not from translations inside sets on the same level.\(^1\)

\(^1\) Here: HI = LOW = 10% of 100-best list.
Multipartite ranking

- Instead of training on all pairs, only compare good translations with bad ones without teasing apart small differences.
- Build pairs from levels HI-MID, HI-LOW, and MID-LOW, but not from translations inside sets on the same level.\(^1\)

\(^1\) Here: HI = LOW = 10% of 100-best list.
Algorithm 1

- Baseline, **not distributed**, used for **tuning on dev set**.
- **Averages** final weight updates of each epoch.

**Algorithm 1 SGD**

```markdown
Initialize \( w_{0,0,0} \leftarrow 0 \).
for epochs \( t \leftarrow 0 \ldots T - 1 \): do
    for all \( i \in \{0 \ldots I - 1\} \): do
        Decode \( i \)th input with \( w_{t,i,0} \).
        for all pairs \( x_j, j \in \{0 \ldots P - 1\} \): do
            \( w_{t,i,j+1} \leftarrow w_{t,i,j} - \eta \nabla l_j(w_{t,i,j}) \)
        end for
        \( w_{t,i+1,0} \leftarrow w_{t,i,P} \)
    end for
    \( w_{t+1,0,0} \leftarrow w_{t,I,0} \)
end for
return \( \frac{1}{T} \sum_{t=1}^{T} w_{t,0,0} \)
```
Algorithm 2

- \( \approx \) Distributed SGD w/ MapReduce (Zinkevich et al. NIPS’10).
- Mixing of final parameters from each shard.

Algorithm 2 MixSGD

Partition data into \( Z \) shards, each of size \( S \leftarrow I/Z \); distribute to machines.

for all shards \( z \in \{1 \ldots Z\} \): parallel do

Initialize \( w_{z,0,0,0} \leftarrow 0 \).

for epochs \( t \leftarrow 0 \ldots T - 1 \): do

for all \( i \in \{0 \ldots S - 1\} \): do

Decode \( i^{\text{th}} \) input with \( w_{z,t,i,0} \).

for all pairs \( x_j, j \in \{0 \ldots P - 1\} \): do

\( w_{z,t,i,j+1} \leftarrow w_{z,t,i,j} - \eta \nabla l_j(w_{z,t,i,j}) \)

end for

\( w_{z,t,i+1,0} \leftarrow w_{z,t,i,P} \)

end for

\( w_{z,t+1,0,0} \leftarrow w_{z,t,S,0} \)

end for

Collect final weights from each machine,

return \( \frac{1}{Z} \sum_{z=1}^{Z} \left( \frac{1}{T} \sum_{t=1}^{T} w_{z,t,0,0} \right) \).
Algorithm 3

- \( \approx \textbf{Iterative Mixing} \) w/ MapReduce (McDonald et al. HLT’10).
- Mixing of weights from each shard after each epoch.

**Algorithm 3** IterMixSGD

Partition data into \( Z \) shards, each of size \( S \leftarrow I/Z \); distribute to machines. Initialize \( v \leftarrow 0 \).

for epochs \( t \leftarrow 0 \ldots T - 1 \): do

  for all shards \( z \in \{1 \ldots Z\} \): parallel do
    \( w_{z,t,0,0} \leftarrow v \)
    for all \( i \in \{0 \ldots S - 1\} \): do
      Decode \( i^{th} \) input with \( w_{z,t,i,0} \).
      for all pairs \( x_j, j \in \{0 \ldots P - 1\} \): do
        \( w_{z,t,i,j+1} \leftarrow w_{z,t,i,j} - \eta \nabla l_j (w_{z,t,i,j}) \)
      end for
      \( w_{z,t,i+1,0} \leftarrow w_{z,t,i,P} \)
    end for
  end for

Collect weights \( v \leftarrow \frac{1}{Z} \sum_{z=1}^{Z} w_{z,t,S,0} \).

end for

return \( v \)
Algorithm 4

- **Feature selection** on shards after each epoch,
- combined with **iterative mixing of reduced weight vectors**.

**Algorithm 4 IterSelSGD**

Partition data into $Z$ shards, each of size $S = I/Z$; distribute to machines.

Initialize $v \leftarrow 0$.

for epochs $t \leftarrow 0 \ldots T - 1$:

for all shards $z \in \{1 \ldots Z\}$: parallel do

$w_{z,t,0,0} \leftarrow v$

for all $i \in \{0 \ldots S - 1\}$: do

Decode $i^{th}$ input with $w_{z,t,i,0}$.

for all pairs $x_j, j \in \{0 \ldots P - 1\}$: do

$w_{z,t,i,j+1} \leftarrow w_{z,t,i,j} - \eta \nabla l_j(w_{z,t,i,j})$

end for

$w_{z,t,i+1,0} \leftarrow w_{z,t,i,P}$

end for

end for

Collect/stack weights $W \leftarrow [w_{1,t,s,0} | \ldots | w_{Z,t,s,0}]^T$

Select top $K$ feature columns of $W$ by $\ell_2$ norm and

for $k \leftarrow 1 \ldots K$ do

$v[k] = \frac{1}{Z} \sum_{z=1}^{Z} W[z][k]$.

end for

end for

return $v$
Algorithm 4 as feature selection procedure

- Represent weights in a $Z$-by-$D$ matrix
  
  $W = [w_{z1} | \ldots | w_{zz}]^T$

  of stacked $D$-dimensional weight vectors across $Z$ shards.

- Select top $K$ feature columns that have highest $\ell_2$ norm over shards (or equivalently, by setting a threshold $\lambda$).

- Average weights of selected features $k \leftarrow 1 \ldots K$ over shards
  
  $v[k] = \frac{1}{Z} \sum_{z=1}^{Z} W[z][k]$

- Resend reduced weight vector $v$ to shards for new epoch.
Algorithm 4 as feature selection procedure

- Represent weights in a $Z$-by-$D$ matrix
  \[ W = \begin{bmatrix} w_{z1} & \cdots & w_{zz} \end{bmatrix}^T \]
  of stacked $D$-dimensional weight vectors across $Z$ shards.

- Select top $K$ feature columns that have highest $\ell_2$ norm over shards (or equivalently, by setting a threshold $\lambda$).

- Average weights of selected features $k \leftarrow 1 \ldots K$ over shards
  \[ v[k] = \frac{1}{Z} \sum_{z=1}^{Z} W[z][k] \]

- Resend reduced weight vector $v$ to shards for new epoch.
Algorithm 4 as feature selection procedure

• Represent weights in a $Z$-by-$D$ matrix

$$W = [w_{z1} | \ldots | w_{zz}]^T$$

of stacked $D$-dimensional weight vectors across $Z$ shards.

• **Select top $K$ feature columns that have highest $\ell_2$ norm over shards** (or equivalently, by setting a threshold $\lambda$).

• **Average weights of selected features** $k \leftarrow 1 \ldots K$ over shards

$$v[k] = \frac{1}{Z} \sum_{z=1}^{Z} W[z][k]$$

• Resend reduced weight vector $v$ to shards for new epoch.
Algorithm 4 as feature selection procedure

- Represent weights in a $Z$-by-$D$ matrix

\[ W = [w_{z1} | \ldots | w_{zz}]^T \]

of stacked $D$-dimensional weight vectors across $Z$ shards.

- Select top $K$ feature columns that have highest $\ell_2$ norm over shards (or equivalently, by setting a threshold $\lambda$).

- Average weights of selected features $k \leftarrow 1 \ldots K$ over shards

\[ v[k] = \frac{1}{Z} \sum_{z=1}^{Z} W[z][k] \]

- Resend reduced weight vector $v$ to shards for new epoch.
Algorithm 4 as $\ell_1/\ell_2$ regularization

- Let $w_d$ be the $d$th column vector of $W$, representing the weights for the $d$th feature across shards.
- Weighted $\ell_1/\ell_2$ norm:

$$\lambda \|W\|_{1,2} = \lambda \sum_{d=1}^{D} \|w_d\|_2.$$

- Each $\ell_2$ norm of a weight column represents the relevance of the corresponding feature across shards.
- The $\ell_1$ sum of the $\ell_2$ norms enforces a selection among features based on these norms.
Algorithm 4 as $\ell_1/\ell_2$ regularization

- Let $w_d$ be the $d$th column vector of $W$, representing the weights for the $d$th feature across shards.
- **Weighted $\ell_1/\ell_2$ norm:**

$$\lambda \| W \|_{1,2} = \lambda \sum_{d=1}^{D} \| w_d \|_2.$$ 

- Each $\ell_2$ norm of a weight column represents the relevance of the corresponding feature across shards.
- The $\ell_1$ sum of the $\ell_2$ norms enforces a selection among features based on these norms.
Algorithm 4 as $\ell_1/\ell_2$ regularization

- Let $w_d$ be the $d$th column vector of $W$, representing the weights for the $d$th feature across shards.
- **Weighted $\ell_1/\ell_2$ norm:**

$$\lambda \|W\|_{1,2} = \lambda \sum_{d=1}^{D} \|w_d\|_2.$$ 

- Each $\ell_2$ norm of a weight column represents the **relevance** of the corresponding feature across shards.
- The $\ell_1$ sum of the $\ell_2$ norms enforces a selection among features based on these norms.
Algorithm 4 as $\ell_1/\ell_2$ regularization

- Let $w_d$ be the $d$th column vector of $W$, representing the weights for the $d$th feature across shards.
- **Weighted $\ell_1/\ell_2$ norm:**

\[ \lambda \| W \|_{1,2} = \lambda \sum_{d=1}^{D} \| w_d \|_2. \]

- Each $\ell_2$ norm of a weight column represents the relevance of the corresponding feature across shards.
- The $\ell_1$ sum of the $\ell_2$ norms enforces a selection among features based on these norms.
\( l_1 / l_2 \) regularization and multi-task learning

- **Multi-task learning** aims to find **common set of features** that are **relevant simultaneously to different tasks**.

- Minimizing \( l_1 / l_2 \) norm promotes **feature sharing** and enforces **similar sparsity patterns across tasks**.

- Example: 2 matrices for 5 features and 3 tasks/shards.

  \[
  \begin{array}{c}
  w_{z_1} \ \ \begin{bmatrix} w_1 \ w_2 \ w_3 \ w_4 \ w_5 \end{bmatrix} \\
  w_{z_2} \ \ \begin{bmatrix} 0 \ 0 \ 3 \ 0 \ 0 \end{bmatrix} \\
  w_{z_3} \ \ \begin{bmatrix} 0 \ 0 \ 0 \ 2 \ 3 \end{bmatrix}
  \end{array}
  \begin{array}{c}
  \text{column } l_2 \text{ norm:} \\
  \text{\( l_1 \text{ sum:} \)}
  \end{array}
  \begin{array}{c}
  6 \ 4 \ 0 \ 0 \ 0 \\
  0 \ 0 \ 3 \ 0 \ 0 \\
  0 \ 0 \ 0 \ 2 \ 3
  \end{array}
  \begin{array}{c}
  \Rightarrow 18
  \end{array}
  \begin{array}{c}
  6 \ 4 \ 0 \ 0 \ 0 \\
  3 \ 0 \ 0 \ 0 \ 0 \\
  2 \ 3 \ 0 \ 0 \ 0
  \end{array}
  \begin{array}{c}
  \Rightarrow 12
  \end{array}
  \]

- Right-hand side has smaller \( l_1 / l_2 \) norm (12 instead of 18).

- Algorithm 4 enforces this choice by weight-based recursive feature elimination (Lal et al. 2006).\(^2\)

\(^2\)Alternative is incremental forward selection (Obozinski et al. 2010)
**ℓ₁/ℓ₂ regularization and multi-task learning**

- **Multi-task learning** aims to find **common set of features** that are **relevant simultaneously to different tasks**.
- Minimizing ℓ₁/ℓ₂ norm promotes **feature sharing** and enforces **similar sparsity patterns across tasks**.
- Example: 2 matrices for 5 features and 3 tasks/shards.

<table>
<thead>
<tr>
<th></th>
<th>w₁</th>
<th>w₂</th>
<th>w₃</th>
<th>w₄</th>
<th>w₅</th>
<th>w₁</th>
<th>w₂</th>
<th>w₃</th>
<th>w₄</th>
<th>w₅</th>
</tr>
</thead>
<tbody>
<tr>
<td>w₂₁</td>
<td>6</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>w₂₂</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>w₂₃</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

| column ℓ₂ norm: | 6 | 4 | 3 | 2 | 3 | 7 | 5 | 0 | 0 | 0 |
| ℓ₁ sum:        | 18 | 12 |

- Right-hand side has smaller ℓ₁/ℓ₂ norm (12 instead of 18).
- Algorithm 4 enforces this choice by weight-based recursive feature elimination (Lal et al. 2006).²

²Alternative is incremental forward selection (Obozinski et al. 2010)
\( \ell_1/\ell_2 \) regularization and multi-task learning

- **Multi-task learning** aims to find **common set of features** that are **relevant simultaneously to different tasks**.
- Minimizing \( \ell_1/\ell_2 \) norm promotes **feature sharing** and enforces **similar sparsity patterns across tasks**.
- Example: 2 matrices for 5 features and 3 tasks/shards.

\[
\begin{bmatrix}
  w_{z_1} & 6 & 4 & 0 & 0 & 0 \\
  w_{z_2} & 0 & 0 & 3 & 0 & 0 \\
  w_{z_3} & 0 & 0 & 0 & 2 & 3 \\
\end{bmatrix}
\quad \begin{bmatrix}
  w_{z_1} & 6 & 4 & 0 & 0 & 0 \\
  w_{z_2} & 3 & 0 & 0 & 0 & 0 \\
  w_{z_3} & 2 & 3 & 0 & 0 & 0 \\
\end{bmatrix}
\]

- Right-hand side has smaller \( \ell_1/\ell_2 \) norm (12 instead of 18).
- Algorithm 4 enforces this choice by weight-based recursive feature elimination (Lal et al. 2006).\(^2\)

\(^2\)Alternative is incremental forward selection (Obozinski et al. 2010)
\( \ell_1 / \ell_2 \) regularization and multi-task learning

- **Multi-task learning** aims to find **common set of features** that are **relevant simultaneously to different tasks**.
- Minimizing \( \ell_1 / \ell_2 \) norm promotes **feature sharing** and enforces **similar sparsity patterns across tasks**.
- Example: 2 matrices for 5 features and 3 tasks/shards.

\[
\begin{align*}
\mathbf{w}_{z_1} & = \begin{bmatrix} 6 & 4 & 0 & 0 & 0 \end{bmatrix} \quad \Rightarrow 18 \\
\mathbf{w}_{z_2} & = \begin{bmatrix} 0 & 0 & 3 & 0 & 0 \end{bmatrix} \quad \Rightarrow 12 \\
\mathbf{w}_{z_3} & = \begin{bmatrix} 0 & 0 & 0 & 2 & 3 \end{bmatrix}
\end{align*}
\]

- Right-hand side has smaller \( \ell_1 / \ell_2 \) norm (12 instead of 18).
- Algorithm 4 enforces this choice by weight-based recursive feature elimination (Lal et al. 2006).\(^2\)

\(^2\)Alternative is incremental forward selection (Obozinski et al. 2010)
\( \ell_1/\ell_2 \) regularization and multi-task learning

- **Multi-task learning** aims to find common set of features that are relevant simultaneously to different tasks.
- Minimizing \( \ell_1/\ell_2 \) norm promotes feature sharing and enforces similar sparsity patterns across tasks.
- Example: 2 matrices for 5 features and 3 tasks/shards.

\[
\begin{align*}
[w_{z1}] & = \begin{bmatrix} 6 & 4 & 0 & 0 & 0 \end{bmatrix} & \Rightarrow 18 \\
[w_{z2}] & = \begin{bmatrix} 0 & 0 & 3 & 0 & 0 \end{bmatrix} & \Rightarrow 12 \\
[w_{z3}] & = \begin{bmatrix} 0 & 0 & 0 & 2 & 3 \end{bmatrix}
\end{align*}
\]

- Right-hand side has smaller \( \ell_1/\ell_2 \) norm (12 instead of 18).
- Algorithm 4 enforces this choice by weight-based recursive feature elimination (Lal et al. 2006).\(^2\)

\(^2\)Alternative is incremental forward selection (Obozinski et al. 2010)
Experiments: SMT setup

- German-to-English hierarchical phrase-based translation (Chiang CL’07).
- cdec (Dyer et al. ACL’10) framework for decoding, induction of SCFGs, compound splitting, etc.
- 3-gram and 5-gram language models using SRILM (Stolcke ICSLP’02) and binarized for efficient querying using kenlm (Heafield WMT’11).
- SCFG per-sentence grammars are stored on disk instead of in memory (Lopez EMNLP’07), extracted by leave-one-out (Zollmann and Sima’an JACL’05) for training-set tuning.
Experiments: SMT setup

- German-to-English hierarchical phrase-based translation (Chiang CL’07).
- cdec (Dyer et al. ACL’10) framework for decoding, induction of SCFGs, compound splitting, etc.
- 3-gram and 5-gram language models using SRILM (Stolcke ICSLP’02) and binarized for efficient querying using kenlm (Heafield WMT’11).
- SCFG per-sentence grammars are stored on disk instead of in memory (Lopez EMNLP’07), extracted by leave-one-out (Zollmann and Sima’an JACL’05) for training-set tuning.
Experiments: SMT setup

- German-to-English hierarchical phrase-based translation (Chiang CL’07).
- **cdec** (Dyer et al. ACL’10) framework for decoding, induction of SCFGs, compound splitting, etc.
- 3-gram and 5-gram language models using SRILM (Stolcke ICSLP’02) and binarized for efficient querying using kenlm (Heafield WMT’11).
- SCFG per-sentence grammars are stored on disk instead of in memory (Lopez EMNLP’07), extracted by leave-one-out (Zollmann and Sima’an JACL’05) for training-set tuning.
Experiments: SMT setup

- German-to-English hierarchical phrase-based translation (Chiang CL’07).
- **cdec** (Dyer et al. ACL’10) framework for decoding, induction of SCFGs, compound splitting, etc.
- 3-gram and 5-gram language models using SRILM (Stolcke ICSLP’02) and binarized for efficient querying using kenlm (Heafield WMT’11).
- SCFG per-sentence grammars are stored on disk instead of in memory (Lopez EMNLP’07), extracted by leave-one-out (Zollmann and Sima’an JACL’05) for training-set tuning.
Distributed processing

- MapReduce cluster able to handle 300 jobs at once.
- Data are split into shards holding about 1,000 sentences, corresponding to dev set size.
- Training and decoding fit MapReduce framework very naturally:
  - Storing grammars on disk instead of memory deploys DFS with minimal overhead of loading grammars immediately prior to decoding.
  - Algorithm 4 uses data shards for distribution with minimal extra network communication.
Distributed processing

- MapReduce cluster able to handle 300 jobs at once.
- Data are split into shards holding about 1,000 sentences, corresponding to dev set size.
- Training and decoding fit MapReduce framework very naturally:
  - Storing grammars on disk instead of memory deploys DFS with minimal overhead of loading grammars immediately prior to decoding.
  - Algorithm 4 uses data shards for distribution with minimal extra network communication.
Distributed processing

- MapReduce cluster able to handle 300 jobs at once.
- Data are split into shards holding about 1,000 sentences, corresponding to dev set size.
- Training and decoding fit MapReduce framework very naturally:
  - Storing grammars on disk instead of memory deploys DFS with minimal overhead of loading grammars immediately prior to decoding.
  - Algorithm 4 uses data shards for distribution with minimal extra network communication.
Distributed processing

- MapReduce cluster able to handle 300 jobs at once.
- Data are split into shards holding about 1,000 sentences, corresponding to dev set size.
- Training and decoding fit MapReduce framework very naturally:
  - Storing grammars on disk instead of memory deploys DFS with minimal overhead of loading grammars immediately prior to decoding.
  - Algorithm 4 uses data shards for distribution with minimal extra network communication.
Distributed processing

- MapReduce cluster able to handle 300 jobs at once.
- Data are split into shards holding about 1,000 sentences, corresponding to dev set size.
- Training and decoding fit MapReduce framework very naturally:
  - Storing grammars on disk instead of memory deploys DFS with minimal overhead of loading grammars immediately prior to decoding.
  - Algorithm 4 uses data shards for distribution with minimal extra network communication.
Learning setup

- Perceptron is deterministic when started from 0 vector while MIRA and PRO results fluctuate due to hypergraph sampling.

![Diagram showing BLEU scores]

- Interest in relative gains by scaling up features and/or data, thus choice for perceptron as base learner.
- Statistical significance assessed by Approximate Randomization (Noreen’89).
Learning setup

- Perceptron is deterministic when started from 0 vector while MIRA and PRO results fluctuate due to hypergraph sampling.

- Interest in relative gains by scaling up features and/or data, thus choice for perceptron as base learner.
  - Statistical significance assessed by Approximate Randomization (Noreen’89).
Learning setup

- Perceptron is deterministic when started from $0$ vector while MIRA and PRO results fluctuate due to hypergraph sampling.

- Interest in relative gains by scaling up features and/or data, thus choice for perceptron as base learner.


- Statistical significance assessed by Approximate Randomization (Noreen’89).
Learning setup

- Perceptron is deterministic when started from 0 vector while MIRA and PRO results fluctuate due to hypergraph sampling.

- Interest in relative gains by scaling up features and/or data, thus choice for perceptron as base learner.
- Statistical significance assessed by Approximate Randomization (Noreen’89).
## Data

### News Commentary ($nc$)

<table>
<thead>
<tr>
<th></th>
<th>train-$nc$</th>
<th>lm-train-$nc$</th>
<th>dev-$nc$</th>
<th>devtest-$nc$</th>
<th>test-$nc$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sentences</td>
<td>132,753</td>
<td>180,657</td>
<td>1057</td>
<td>1064</td>
<td>2007</td>
</tr>
<tr>
<td>Tokens $de$</td>
<td>3,530,907</td>
<td>–</td>
<td>27,782</td>
<td>28,415</td>
<td>53,989</td>
</tr>
<tr>
<td>Tokens $en$</td>
<td>3,293,363</td>
<td>4,394,428</td>
<td>26,098</td>
<td>26,219</td>
<td>50,443</td>
</tr>
<tr>
<td>Rule Count</td>
<td>14,350,552 (1G)</td>
<td>–</td>
<td>2,322,912</td>
<td>2,320,264</td>
<td>3,274,771</td>
</tr>
</tbody>
</table>

### Europarl ($ep$)

<table>
<thead>
<tr>
<th></th>
<th>train-$ep$</th>
<th>lm-train-$ep$</th>
<th>dev-$ep$</th>
<th>devtest-$ep$</th>
<th>test-$ep$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sentences</td>
<td>1,655,238</td>
<td>2,015,440</td>
<td>2000</td>
<td>2000</td>
<td>2000</td>
</tr>
<tr>
<td>Tokens $de$</td>
<td>45,293,925</td>
<td>–</td>
<td>57,723</td>
<td>56,783</td>
<td>59,297</td>
</tr>
<tr>
<td>Tokens $en$</td>
<td>45,374,649</td>
<td>54,728,786</td>
<td>58,825</td>
<td>58,100</td>
<td>60,240</td>
</tr>
<tr>
<td>Rule Count</td>
<td>203,552,525 (31.5G)</td>
<td>–</td>
<td>17,738,763</td>
<td>17,682,176</td>
<td>18,273,078</td>
</tr>
</tbody>
</table>

### News Crawl ($crawl$)

<table>
<thead>
<tr>
<th></th>
<th>dev-$crawl$</th>
<th>test-$crawl10$</th>
<th>test-$crawl11$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sentences</td>
<td>2051</td>
<td>2489</td>
<td>3003</td>
</tr>
<tr>
<td>Tokens $de$</td>
<td>49,848</td>
<td>64,301</td>
<td>76,193</td>
</tr>
<tr>
<td>Tokens $en$</td>
<td>49,767</td>
<td>61,925</td>
<td>74,753</td>
</tr>
<tr>
<td>Rule Count</td>
<td>9,404,339</td>
<td>11,307,304</td>
<td>12,561,636</td>
</tr>
</tbody>
</table>
## Results on News Commentary (nc) data

<table>
<thead>
<tr>
<th>Alg.</th>
<th>Tuning set</th>
<th>Features</th>
<th>#Features</th>
<th>test-nc</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>dev-nc</td>
<td>default</td>
<td>12</td>
<td>28.0</td>
</tr>
<tr>
<td></td>
<td>dev-nc</td>
<td>+id,ng,shape</td>
<td>180k</td>
<td>28.15(^{34})</td>
</tr>
<tr>
<td>2</td>
<td>train-nc</td>
<td>default</td>
<td>12</td>
<td>27.86</td>
</tr>
<tr>
<td></td>
<td>train-nc</td>
<td>+id,ng,shape</td>
<td>4.7M</td>
<td>27.86(^{34})</td>
</tr>
<tr>
<td>3</td>
<td>train-nc</td>
<td>default</td>
<td>12</td>
<td>27.94(\dagger)</td>
</tr>
<tr>
<td></td>
<td>train-nc</td>
<td>+id,ng,shape</td>
<td>4.7M</td>
<td>28.55(^{124})</td>
</tr>
<tr>
<td>4</td>
<td>train-nc</td>
<td>+id,ng,shape</td>
<td>100k</td>
<td>28.81(^{123})</td>
</tr>
</tbody>
</table>

- Scaling from 12 to 180K features on dev set does not help.
- **Scaling to full feature- and training-set does help** for Alg.3 (+0.4 BLEU) and Alg. 4 (+0.8 BLEU).
- **Alg.4 gives best BLEU** and is **most efficient on large data**.
### Results on Europarl (ep) and News Crawl (crawl) data

<table>
<thead>
<tr>
<th>Alg.</th>
<th>Tuning set</th>
<th>Features</th>
<th>#Features</th>
<th>test-ep</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>dev-ep</td>
<td>default</td>
<td>12</td>
<td>26.42†</td>
</tr>
<tr>
<td></td>
<td>dev-ep</td>
<td>+id,ng,shape</td>
<td>300k</td>
<td>28.37</td>
</tr>
<tr>
<td>4</td>
<td>train-ep</td>
<td>+id,ng,shape</td>
<td>100k</td>
<td>28.62</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Alg.</th>
<th>Tuning set</th>
<th>Features</th>
<th>#Feats</th>
<th>test-crawl/10</th>
<th>test-crawl/11</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>dev-crawl</td>
<td>default</td>
<td>12</td>
<td>15.39†</td>
<td>14.43†</td>
</tr>
<tr>
<td></td>
<td>dev-crawl</td>
<td>+id,ng,shape</td>
<td>300k</td>
<td>17.84</td>
<td>16.834</td>
</tr>
<tr>
<td>4</td>
<td>train-ep</td>
<td>+id,ng,shape</td>
<td>100k</td>
<td>19.12†</td>
<td>17.33†</td>
</tr>
</tbody>
</table>

- **On large scale, only Alg.4 is feasible** (1.7M parallel data!)
- Scaling up feature sets helps even for dev-set tuning.
- **Additional gains of 0.5 to 1.3 BLEU by scaling to large tuning set** on out-of-domain news crawl test data.
• SMT inference on large data sets is expensive, thus good parallelization is key.

• Our algorithm makes large-scale tuning in SMT feasible by
  • MapReduce-friendliness in decoding and learning,
  • Combination of parallel SGD and feature selection,
  • Efficiently computable features.

• And: It works!

• Future work:
  • Tricks-of-the-trade (larger lm, etc.) for general competitiveness.
  • More and better features and more sophisticated learners.
  • Application to multi-task patent translation.
Conclusion

- SMT inference on large data sets is expensive, thus **good parallelization is key**.
- Our algorithm makes large-scale tuning in SMT feasible by
  - **MapReduce-friendliness** in decoding and learning,
  - Combination of parallel SGD and feature selection,
  - Efficiently computable features.
- And: **It works**!
- Future work:
  - Tricks-of-the-trade (larger lm, etc.) for general competitiveness.
  - More and better features and more sophisticated learners.
  - Application to multi-task patent translation.
Conclusion

- SMT inference on large data sets is expensive, thus **good parallelization is key**.
- Our algorithm makes large-scale tuning in SMT feasible by
  - **MapReduce-friendliness** in decoding and learning,
  - **Combination of parallel SGD and feature selection**,
  - Efficiently computable features.
- And: **It works**!
- Future work:
  - Tricks-of-the-trade (larger lm, etc.) for general competitiveness.
  - More and better features and more sophisticated learners.
  - Application to multi-task patent translation.
Conclusion

- SMT inference on large data sets is expensive, thus **good parallelization is key**.
- Our algorithm makes large-scale tuning in SMT feasible by
  - **MapReduce-friendliness** in decoding and learning,
  - **Combination of parallel SGD and feature selection**,  
  - **Efficiently computable features**.
- And: **It works!**
- Future work:
  - Tricks-of-the-trade (larger lm, etc.) for general competitiveness. 
  - More and better features and more sophisticated learners. 
  - Application to multi-task patent translation.
Conclusion

- SMT inference on large data sets is expensive, thus **good parallelization is key**.
- Our algorithm makes large-scale tuning in SMT feasible by
  - **MapReduce-friendliness** in decoding and learning,
  - **Combination of parallel SGD and feature selection**,
  - **Efficiently computable features**.
- And: **It works!**
- Future work:
  - Tricks-of-the-trade (larger lm, etc.) for general competitiveness.
  - More and better features and more sophisticated learners.
  - Application to multi-task patent translation.
Conclusion

- SMT inference on large data sets is expensive, thus **good parallelization is key**.
- Our algorithm makes large-scale tuning in SMT feasible by
  - **MapReduce-friendliness** in decoding and learning,
  - **Combination of parallel SGD and feature selection**,  
  - **Efficiently computable features**.
- And: **It works!**
- Future work:
  - Tricks-of-the-trade (larger lm, etc.) for general competitiveness.
  - More and better features and more sophisticated learners.
  - Application to multi-task patent translation.
Conclusion

- SMT inference on large data sets is expensive, thus **good parallelization is key**.
- Our algorithm makes large-scale tuning in SMT feasible by
  - **MapReduce-friendliness** in decoding and learning,
  - **Combination of parallel SGD and feature selection,**
  - **Efficiently computable features.**

- And: **It works!**
- Future work:
  - Tricks-of-the-trade (larger lm, etc.) for general competitiveness.
  - More and better features and more sophisticated learners.
  - Application to multi-task patent translation.
Conclusion

• SMT inference on large data sets is expensive, thus **good parallelization is key**.
• Our algorithm makes large-scale tuning in SMT feasible by
  • **MapReduce-friendliness** in decoding and learning,
  • **Combination of parallel SGD and feature selection**,  
  • Efficiently computable features.
• And: **It works**!
• Future work:
  • Tricks-of-the-trade (larger lm, etc.) for general competitiveness.
  • More and better features and more sophisticated learners.
  • Application to multi-task patent translation.
Code

- dtrain code is part of cdec:
  https://github.com/redpony/cdec.
Thanks for your attention!