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 ℓ_1/ℓ_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 hat X_2 versprochen; X_1 promised X_2
(2) $X \rightarrow$ X_1 hat mir X_2 versprochen;
X_1 promised me X_2
(3) $X \rightarrow$ X_1 versprach X_2; X_1 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).

Local features for SCFGs

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

Local features for SCFGs

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

Local features for SCFGs

(1) $X \to$ X_1 hat X_2 versprochen; X_1 promised X_2
(2) $X \to$ X_1 hat mir X_2 versprochen;
X_1 promised me X_2
(3) $X \to$ X_1 versprach X_2; X_1 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 $\mathbf{x}_j = (\mathbf{x}_j^{(1)}, \mathbf{x}_j^{(2)})$ 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

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

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

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

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

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

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

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

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

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

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

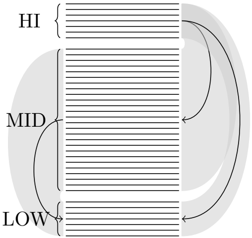
$$l_j(\mathbf{w}) = (-\langle \mathbf{w}, \bar{\mathbf{x}}_j \rangle)_+$$

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

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

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

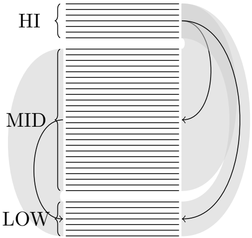
Multipartite ranking



- 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

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

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 $z \in \{1 \dots Z\}$: **parallel do**
    Initialize $\mathbf{w}_{z,0,0,0} \leftarrow \mathbf{0}$.
    for epochs $t \leftarrow 0 \dots T-1$: **do**
        for all $i \in \{0 \dots S-1\}$: **do**
            Decode i^{th} input with $\mathbf{w}_{z,t,i,0}$.
            for all pairs $x_j, j \in \{0 \dots P-1\}$: **do**
                $\mathbf{w}_{z,t,i,j+1} \leftarrow \mathbf{w}_{z,t,i,j} - \eta \nabla l_j(\mathbf{w}_{z,t,i,j})$
            end for
            $\mathbf{w}_{z,t,i+1,0} \leftarrow \mathbf{w}_{z,t,i,P}$
        end for
        $\mathbf{w}_{z,t+1,0,0} \leftarrow \mathbf{w}_{z,t,S,0}$
    end for
end for
Collect final weights from each machine,
return $\frac{1}{Z} \sum_{z=1}^{Z} \left(\frac{1}{T} \sum_{t=1}^{T} \mathbf{w}_{z,t,0,0} \right)$.

Algorithm 3

- $\approx$ **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 $\mathbf{v} \leftarrow \mathbf{0}$.
for epochs $t \leftarrow 0 \ldots T-1$: **do**
    for all shards $z \in \{1 \ldots Z\}$: **parallel do**
        $\mathbf{w}_{z,t,0,0} \leftarrow \mathbf{v}$
        for all $i \in \{0 \ldots S-1\}$: **do**
            Decode i^{th} input with $\mathbf{w}_{z,t,i,0}$.
            for all pairs $x_j, j \in \{0 \ldots P-1\}$: **do**
                $\mathbf{w}_{z,t,i,j+1} \leftarrow \mathbf{w}_{z,t,i,j} - \eta \nabla l_j(\mathbf{w}_{z,t,i,j})$
            end for
            $\mathbf{w}_{z,t,i+1,0} \leftarrow \mathbf{w}_{z,t,i,P}$
        end for
    end for
    Collect weights $\mathbf{v} \leftarrow \frac{1}{Z} \sum_{z=1}^{Z} \mathbf{w}_{z,t,S,0}$.
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 $\mathbf{v} \leftarrow \mathbf{0}$.
for epochs $t \leftarrow 0 \dots T-1$: **do**
  for all shards $z \in \{1 \dots Z\}$: **parallel do**
    $\mathbf{w}_{z,t,0,0} \leftarrow \mathbf{v}$
    for all $i \in \{0 \dots S-1\}$: **do**
      Decode i^{th} input with $\mathbf{w}_{z,t,i,0}$.
      for all pairs $x_j, j \in \{0 \dots P-1\}$: **do**
        $\mathbf{w}_{z,t,i,j+1} \leftarrow \mathbf{w}_{z,t,i,j} - \eta \nabla l_j(\mathbf{w}_{z,t,i,j})$
      end for
      $\mathbf{w}_{z,t,i+1,0} \leftarrow \mathbf{w}_{z,t,i,P}$
    end for
  end for
  Collect/stack weights $\mathbf{W} \leftarrow [\mathbf{w}_{1,t,S,0}| \dots |\mathbf{w}_{Z,t,S,0}]^T$
  Select top K feature columns of $\mathbf{W}$ by ℓ_2 norm and
  for $k \leftarrow 1 \dots K$ **do**
    $\mathbf{v}[k] = \frac{1}{Z} \sum_{z=1}^{Z} \mathbf{W}[z][k].$
  end for
end for
return v

Algorithm 4 as feature selection procedure

- Represent weights in a Z-by-D matrix

$$\mathbf{W} = [\mathbf{w}_{z_1}| \dots |\mathbf{w}_{z_Z}]^T$$

of stacked D-dimensional weight vectors across Z shards.

- **Select top K feature columns that have highest ℓ_2 norm over shards** (or equivalently, by setting a threshold λ).
- **Average weights of selected features** $k \leftarrow 1 \dots K$ over shards

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

- Resend reduced weight vector $\mathbf{v}$ to shards for new epoch.

Algorithm 4 as feature selection procedure

- Represent weights in a Z-by-D matrix

$$\mathbf{W} = [\mathbf{w}_{z_1}|\dots|\mathbf{w}_{z_Z}]^T$$

  of stacked D-dimensional weight vectors across Z shards.
- **Select top K feature columns that have highest ℓ_2 norm over shards** (or equivalently, by setting a threshold λ).
- **Average weights of selected features** $k \leftarrow 1 \dots K$ over shards

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

- Resend reduced weight vector $\mathbf{v}$ to shards for new epoch.

Algorithm 4 as feature selection procedure

- Represent weights in a Z-by-D matrix

$$\mathbf{W} = [\mathbf{w}_{z_1}| \dots |\mathbf{w}_{z_Z}]^T$$

of stacked D-dimensional weight vectors across Z shards.

- **Select top K feature columns that have highest ℓ_2 norm over shards** (or equivalently, by setting a threshold λ).
- **Average weights of selected features** $k \leftarrow 1 \dots K$ over shards

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

- Resend reduced weight vector $\mathbf{v}$ to shards for new epoch.

Algorithm 4 as feature selection procedure

- Represent weights in a Z-by-D matrix

$$\mathbf{W} = [\mathbf{w}_{z_1} | \dots | \mathbf{w}_{z_Z}]^T$$

  of stacked D-dimensional weight vectors across Z shards.
- **Select top K feature columns that have highest ℓ_2 norm over shards** (or equivalently, by setting a threshold λ).
- **Average weights of selected features** $k \leftarrow 1 \dots K$ over shards

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

- Resend reduced weight vector $\mathbf{v}$ to shards for new epoch.

Algorithm 4 as ℓ_1/ℓ_2 regularization

- Let w_d be the dth column vector of **W**, representing the weights for the dth feature across shards.
- **Weighted ℓ_1/ℓ_2 norm**:

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

- Each ℓ_2 **norm** of a weight column represents the **relevance of the corresponding feature across shards**.
- The ℓ_1 **sum** of the ℓ_2 norms **enforces a selection among features** based on these norms.

Algorithm 4 as ℓ_1/ℓ_2 regularization

- Let w_d be the dth column vector of $\mathbf{W}$, representing the weights for the dth feature across shards.
- **Weighted ℓ_1/ℓ_2 norm**:

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

- Each ℓ_2 **norm** of a weight column represents the **relevance of the corresponding feature across shards**.
- The ℓ_1 **sum** of the ℓ_2 norms **enforces a selection among features** based on these norms.

Algorithm 4 as ℓ_1/ℓ_2 regularization

- Let w_d be the dth column vector of **W**, representing the weights for the dth feature across shards.
- **Weighted ℓ_1/ℓ_2 norm**:

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

- Each ℓ_2 **norm** of a weight column represents the **relevance of the corresponding feature across shards**.
- The ℓ_1 **sum** of the ℓ_2 norms **enforces a selection among features** based on these norms.

Algorithm 4 as ℓ_1/ℓ_2 regularization

- Let w_d be the dth column vector of $\mathbf{W}$, representing the weights for the dth feature across shards.
- **Weighted ℓ_1/ℓ_2 norm**:

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

- Each ℓ_2 **norm** of a weight column represents the **relevance of the corresponding feature across shards**.
- The ℓ_1 **sum** of the ℓ_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.

	w_1	w_2	w_3	w_4	w_5		w_1	w_2	w_3	w_4	w_5	
$\mathbf{w}_{z_1}$	6	4	0	0	0		6	4	0	0	0	
$\mathbf{w}_{z_2}$	0	0	3	0	0		3	0	0	0	0	
$\mathbf{w}_{z_3}$	0	0	0	2	3		2	3	0	0	0	
column ℓ_2 norm:	6	4	3	2	3		7	5	0	0	0	
ℓ_1 sum:					$\Rightarrow$	18					$\Rightarrow$	12

- 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)

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.

	w_1	w_2	w_3	w_4	w_5		w_1	w_2	w_3	w_4	w_5	
$\mathbf{w}_{z_1}$	6	4	0	0	0		6	4	0	0	0	
$\mathbf{w}_{z_2}$	0	0	3	0	0		3	0	0	0	0	
$\mathbf{w}_{z_3}$	0	0	0	2	3		2	3	0	0	0	
column ℓ_2 norm:	6	4	3	2	3		7	5	0	0	0	
ℓ_1 sum:					$\Rightarrow$	18					$\Rightarrow$	12

- 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)

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.

$$
\begin{array}{r|c|c}
 & \begin{array}{ccccccc} & w_1 & w_2 & w_3 & w_4 & w_5 & \end{array} & \begin{array}{ccccccc} & w_1 & w_2 & w_3 & w_4 & w_5 & \end{array} \\
\mathbf{w}_{z_1} & \begin{array}{ccccccc} [& 6 & 4 & 0 & 0 & 0 &] \end{array} & \begin{array}{ccccccc} [& 6 & 4 & 0 & 0 & 0 &] \end{array} \\
\mathbf{w}_{z_2} & \begin{array}{ccccccc} [& 0 & 0 & 3 & 0 & 0 &] \end{array} & \begin{array}{ccccccc} [& 3 & 0 & 0 & 0 & 0 &] \end{array} \\
\mathbf{w}_{z_3} & \begin{array}{ccccccc} [& 0 & 0 & 0 & 2 & 3 &] \end{array} & \begin{array}{ccccccc} [& 2 & 3 & 0 & 0 & 0 &] \end{array} \\
\text{column } \ell_2 \text{ norm:} & \begin{array}{ccccccc} & 6 & 4 & 3 & 2 & 3 & \end{array} & \begin{array}{ccccccc} & 7 & 5 & 0 & 0 & 0 & \end{array} \\
\ell_1 \text{ sum:} & \Rightarrow \quad 18 & \Rightarrow \quad 12
\end{array}
$$

- 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)

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.

$$
\begin{array}{r|c}
\begin{array}{rccccccc}
 & & w_1 & w_2 & w_3 & w_4 & w_5 & \\
\mathbf{w}_{z_1} & [& 6 & 4 & 0 & 0 & 0 &] \\
\mathbf{w}_{z_2} & [& 0 & 0 & 3 & 0 & 0 &] \\
\mathbf{w}_{z_3} & [& 0 & 0 & 0 & 2 & 3 &] \\
\text{column } \ell_2 \text{ norm:} & & 6 & 4 & 3 & 2 & 3 & \\
\ell_1 \text{ sum:} & & & & & & \Rightarrow & 18
\end{array}
&
\begin{array}{ccccccc}
 & w_1 & w_2 & w_3 & w_4 & w_5 & \\
[& 6 & 4 & 0 & 0 & 0 &] \\
[& 3 & 0 & 0 & 0 & 0 &] \\
[& 2 & 3 & 0 & 0 & 0 &] \\
 & 7 & 5 & 0 & 0 & 0 & \\
 & & & & & \Rightarrow & 12
\end{array}
\end{array}
$$

- 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)

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.

$$
\begin{array}{r|r}
\begin{array}{rcccccc}
 & w_1 & w_2 & w_3 & w_4 & w_5 & \\
\mathbf{w}_{z_1} & [\ 6 & 4 & 0 & 0 & 0\ ] & \\
\mathbf{w}_{z_2} & [\ 0 & 0 & 3 & 0 & 0\ ] & \\
\mathbf{w}_{z_3} & [\ 0 & 0 & 0 & 2 & 3\ ] & \\
\text{column } \ell_2 \text{ norm:} & 6 & 4 & 3 & 2 & 3 & \\
\ell_1 \text{ sum:} & & & & & \Rightarrow & 18
\end{array}
&
\begin{array}{cccccc}
w_1 & w_2 & w_3 & w_4 & w_5 & \\
[\ 6 & 4 & 0 & 0 & 0\ ] & \\
[\ 3 & 0 & 0 & 0 & 0\ ] & \\
[\ 2 & 3 & 0 & 0 & 0\ ] & \\
7 & 5 & 0 & 0 & 0 & \\
 & & & & \Rightarrow & 12
\end{array}
\end{array}
$$

- 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)

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

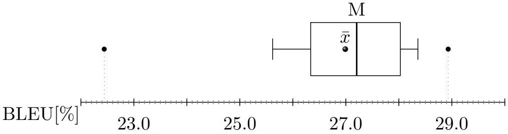
- MapReduce cluster able to handle 300 jobs at once.
- Data are split into shards holding about 1,000 sentences, corresponding to dev set size.
- Training and decoding fit MapReduce framework very naturally:
    - Storing grammars on disk instead of memory deploys DFS with minimal overhead of loading 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

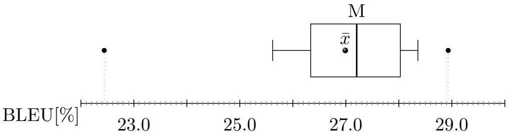
- Perceptron is deterministic when started from **0** vector while MIRA and PRO results fluctuate due to hypergraph sampling.



- Interest in relative gains by scaling up features and/or data, thus choice for perceptron as base learner.
- Evaluation using lowercased BLEU-4 (`mteval-v11b.pl`).
- Statistical significance assessed by Approximate Randomization (Noreen'89).

Learning setup

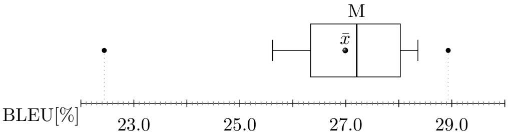
- Perceptron is deterministic when started from **0** vector while MIRA and PRO results fluctuate due to hypergraph sampling.



- Interest in relative gains by scaling up features and/or data, thus choice for perceptron as base learner.
- Evaluation using lowercased BLEU-4 (`mteval-v11b.pl`).
- Statistical significance assessed by Approximate Randomization (Noreen'89).

Learning setup

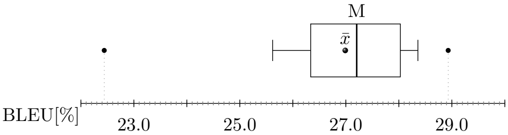
- Perceptron is deterministic when started from **0** vector while MIRA and PRO results fluctuate due to hypergraph sampling.



- Interest in relative gains by scaling up features and/or data, thus choice for perceptron as base learner.
- Evaluation using lowercased BLEU-4 (`mteval-v11b.pl`).
- 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.
- Evaluation using lowercased BLEU-4 (`mteval-v11b.pl`).
- Statistical significance assessed by Approximate Randomization (Noreen'89).

Data

News Commentary(*nc*)

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

Europarl(*ep*)

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

News Crawl(*crawl*)

		dev-*crawl*	**test-*crawl10***	**test-*crawl11***
Sentences		2051	2489	3003
Tokens *de*		49,848	64,301	76,193
Tokens *en*		49,767	61,925	74,753
Rule Count		9,404,339	11,307,304	12,561,636

Results on News Commentary (*nc*) data

Alg.	Tuning set	Features	#Features	test-*nc*
1	dev-*nc*	default	12	28.0
	dev-*nc*	+id,ng,shape	180k	**28.15**[34]
2	train-*nc*	default	12	27.86
	train-*nc*	+id,ng,shape	4.7M	27.86[34]
3	train-*nc*	default	12	27.94[†]
	train-*nc*	+id,ng,shape	4.7M	**28.55**[124]
4	train-*nc*	+id,ng,shape	100k	**28.81**[123]

- 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

Alg.	Tuning set	Features	#Features	test-*ep*
1	dev-*ep*	default	12	26.42[†]
	dev-*ep*	+id,ng,shape	300k	**28.37**
4	train-*ep*	+id,ng,shape	100k	**28.62**

Alg.	Tuning set	Features	#Feats	test-*crawl*10	test-*crawl*11
1	dev-*crawl*	default	12	15.39[†]	14.43[†]
	dev-*crawl*	+id,ng,shape	300k	**17.8**[4]	**16.83**[4]
4	train-*ep*	+id,ng,shape	100k	**19.12**[1]	**17.33**[1]

- **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.

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.

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!