Methods and Theories for Large-scale Structured Prediction

Xu SUN
Peking University
xusun@pku.edu.cn

Yansong FENG
Peking University
fengyansong@pku.edu.cn
Models & Regularization

- Conventional Model
- Latent Model
- Neural Model

Structures & Applications

- Sequence Structure
- Tree/Graph Structure

Content And Lecturer

Xu SUN

Yansong FENG
Models & Regularization

- Conventional Model
- Latent Model
- Neural Model

Structures & Applications

- Sequence Structure
- Tree/Graph Structure

Xu SUN

Yansong FENG
Models and Regularization for Large-scale Structured Prediction

Xu SUN
Peking University
xusun@pku.edu.cn
What Is Structured Prediction/Classification?

- **Types of Classification**
  - **Binary Classification**
  - **Multiclass Classification**

Structured Classification

```
NN VBD DT NN IN DT NN .
```

John hit the ball with the bat .
 Structures are important in natural language processing

Linguists also attempt to understand the rules regarding language structures

Why Structured Prediction?
Why Structured Prediction?

- Challenges in NLP involve Understanding and Generation
- Understanding the structures in natural languages is an essential step towards the goal

D Jurafsky and JH Martin. Speech and language processing.
CD Manning and H Schütze. Foundations of Statistical Natural Language Processing.
However, most of the time, structure prediction is not straight-forward.

Why it is hard? --Natural languages encode all of the structures in a linear form.

N Chomsky. Language and mind.
N Chomsky. Reflections on language.
MAK Halliday. Language structure and language function.
Why Structured Prediction?

- **Structured prediction helps to recover the structures in natural languages**

![Diagram of structured prediction]

- **POS Tagging** (Collins, ACL 2002; Gimenez & Marquez, LREC 2004.; Shen et al., ACL 2007; Søegaard, ACL-HLT 2011; Sun, NIPS 2014; Collobert et al., JMLR 2011; Huang et al., 2015)

- **Chunking** (Kudo & Matsumoto, NAACL 2001; Collins, ACL 2002; McDonald et al., HLT-EMNLP 2005; Sun et al., COLING 2008; Collobert et al., JMLR 2011; Huang et al., 2015)

- **NER** (Florian et al., HLT-NAACL 2003; Chieu, CoNLL 2003; Ando & Zhang, JMLR 2005; Collobert et al., JMLR 2011; Passos et al., CoNLL 2014; Huang et al., 2015)


- **Word Segmentation, Summarization, Machine Translation...**
Outline

- Conditional Random Field
- Structure Perceptron
- MIRA
- Probabilistic Perceptron

- Latent Dynamic CRF
- Latent Variable Perceptron

- Recurrent Neural Network
- Long Short-term Memory
- LSTM-CRF

Reduce Annotation Engineering

Conventional Model

Latent Model

Neural Model

Reduce Feature Engineering
Conditional Random Fields (CRFs)

- **Proposed by** Lafferty et al. (2001)

- **Maximize a conditional probability**
  
  \[ p(y|x, \theta) = \frac{1}{z(x,\theta)} \exp(\sum_k \theta_k f_k(y,x)) \]

  \[ Z(x, \theta) = \sum_{y'} \exp(\sum_k \theta_k f_k(y',x)) \]

- **Global model**
  
  - Predict global structure, not local classifier
  - Training: globally normalized objective
  - Decode: Viterbi algorithm
Conditional Random Fields (CRFs)

- **Proposed by** Lafferty et al. (2001)

- **Maximize a conditional probability**
  
  \[ p(y|x, \theta) = \frac{1}{z(x, \theta)} \exp(\sum_k \theta_k f_k(y, x)) \]

  \[ Z(x, \theta) = \sum_{y'} \exp(\sum_k \theta_k f_k(y', x)) \]

- **Global model**
  
  - Predict global structure, not local classifier
  - Training: globally normalized objective
  - Decode: Viterbi algorithm

  **But the training speed is quite slow…**
Proposed by Collins (2002)

- Simple and fast
  - No gradient computation
  - Update only on error
  - Viterbi decode

Theoretical guarantee

- Converge if data is separable

1: **input**: Training Examples \( \{(x_i, y_i)\}_{i=1}^{n} \)
2: **initialize**: \( \alpha = 0 \)
3: **repeat**
4: Get a random example \( (x_i, y_i) \)
5: \( y^* = \arg\max_{z \in \text{GEN}(x)} \Phi(x, y) \cdot \alpha \)
6: if \( (y^* \neq y) \) then \( \alpha = \alpha + \Phi(x, y) - \Phi(x, y^*) \)
7: **until** Convergence
8: **return** parameter \( \alpha \)
Structured Perceptron

- **Proposed by** Collins (2002)

- **Simple and fast**
  - No gradient computation
  - Update only on error
  - Viterbi decode

- **Theoretical guarantee**
  - Converge if data is separable

1: **input**: Training Examples \( \{(x_i, y_i)\}_{i=1}^{n} \)
2: **initialize**: \( \alpha = 0 \)
3: **repeat**
4: Get a random example \( (x_i, y_i) \)
5: \( y^* = \arg\max_{x \in \text{GEN}(x)} \Phi(x, y) \cdot \alpha \)
6: if \( y^* \neq y \) then \( \alpha = \alpha + \Phi(x, y) - \Phi(x, y^*) \)
7: **until** Convergence
8: **return** parameter \( \alpha \)
Structured Perceptron

- **Inexact search**
  - Greedy search
  - Beam search

- **Parameter update**
  - Early update (Collins & Roark, ACL 2004)
  - Margin Infused Relaxed Algorithm (MIRA) (Crammer et al. 2006)
  - Max-violation (Huang et al., NAACL 2012)
Current structured prediction methods are not ideal

- A trade-off between accuracy and speed...

For Large-scale Structured Prediction

- CRFs
- k-best MIRA
- MIRA
- Structured Perceptron

Performance trade-off:

- Low performance, fast
- High performance, slow

Speed trade-off:

- Fast
- Speed
- Slow
A solution works well in practice for large-scale structured prediction problems

- Introducing probabilistic information into perceptrons
- This goes to Probabilistic Perceptron (SAPO) (Sun 2015)
Proposed by Sun (2015)

Same (or even higher) accuracy like CRF

Fast training speed like perceptron

Probabilistic Perceptron (SAPO)

- **Proposed by Sun (2015)**

  Training sample
  
  1-best sample
  
  n-best samples with probability

  Structured Perceptron Update

  Probabilistic Perceptron Update
Theoretical guarantee of convergence

Theorem 1 (Optimum, convergence, and rate) With the conditions (16), (17), (18), (19), let $\epsilon > 0$ be a target degree of convergence. Let $\tau$ be an approximation-based bound from $s(w)$ to $\nabla f(w)$ such that

$$\tau \leq \frac{c\epsilon}{2q}$$

where $w$ is a historical weight vector that updated during SAPO training, and $s(w)$ is expected $s_z(w)$ over $z$ such that $s(w) = \mathbb{E}_z[s_z(w)]$. Since $s(w)$ can be arbitrary-close to $\nabla f(w)$ by increasing $n$, SAPO can use the smallest $n$ as far as the following holds:

$$\tau \leq \frac{c\epsilon}{2q}$$

Let $\gamma$ be a learning rate as

$$\gamma = \frac{c\epsilon - 2\tau q}{\beta q \kappa^2}$$

where we can set $\beta$ as any value as far as $\beta \geq 1$. Let $t$ be the smallest integer satisfying

$$t \geq \frac{\beta q \kappa^2 \log(qa_0/\epsilon)}{c(c\epsilon - 2\tau q)}$$

where $a_0$ is the initial distance such that $a_0 = \|w_0 - w^*\|^2$. Then, after $t$ updates of $w$, SAPO converges towards the optimum such that

$$\mathbb{E}[f(w_t) - f(w^*)] \leq \epsilon$$
Probabilistic Perceptron (SAPO)

- **Experiment results**
  - Similar or even **higher** accuracy compared with CRFs, perceptron and MIRA

It indicates the number of samples is not the larger the better, why?
Experiment results

- Much faster than CRFs
- Nearly as fast as perceptrons

Probabilistic Perceptron (SAPO)
No rising complexity of weights compared to MIRA or Perc.
Probabilistic Perceptron Sun (2015)
- Typical methods need large-scale annotations
- Problem in reality
  - Lack of annotations
  - Inaccurate annotations
- Latent Model
  - Reduce Annotation Engineering
Outline

- Conditional Random Field
- Structure Perceptron
- MIRA
- Probabilistic Perceptron

Conventional Model

Latent Model
- Latent Dynamic CRF
- Latent Variable Perceptron

Neural Model
- Recurrent Neural Network
- Long Short-term Memory
- LSTM-CRF

Reduce Annotation Engineering

Reduce Feature Engineering
Latent-structures (hidden info) are important in natural language processing (Matsuzaki et al., ACL 2005; Petrov & Klein, NIPS 2008)

Parsing: Learn refined grammars with latent info

```
S
  /\  
NP  VP
  /\  
PRP VBD NP .
    /\  
He heard DT NN .
        /\  
the voice
```
Latent-structures (hidden info) are important in natural language processing (Matsuzaki et al., ACL 2005; Petrov & Klein, NIPS 2008)

Parsing: Learn refined grammars with latent info

He heard the voice
Latent-structures (hidden info) are important in dependency parsing (Honnibal & Johnson, TACL 2014)

A transition system (arc-eager) extended with an EDIT transition
Latent-structures (hidden info) are important in dependency parsing (Honnibal & Johnson, TACL 2014)

Different transition sequences to the same gold-standard tree
Latent-structures (hidden info) are important in dependency parsing (Honnibal & Johnson, TACL 2014)

Motivation

dynamic oracle
map a configuration to a set of transitions
partially annotated
latent variable
Latent-structures (hidden info) are important in question answering (Fader et al., SIGKDD 2014)
Latent-structures (hidden info) are important in multi-intent speech recognition (Xu & Sarikaya, INTERSPEECH 2013)

Figure 1: Graphical model representation of the baseline log-linear classification model.

Figure 2: Graphical model representation of the hidden variable model.
Latent-structures (hidden info) are important in multi-intent speech recognition (Xu & Sarikaya, INTERSPEECH 2013)

Problem 1:
Annotating latent info requires much more tags and human time → Costly to annotate

Problem 2:
Different tasks have different latent info. → Hard to annotate

Figure 1: Graphical model representation of the baseline log-linear classification model.

Figure 2: Graphical model representation of the hidden variable model.
A solution without additional annotation

Latent-dynamic CRFs (LDCRF)  
[Morency et al., CVPR 2007; Sun et al., COLING 2008]  
* No need to annotate latent info

Figure from Xu & Sarikaya, INTERSPEECH 2013.
Latent-dynamic CRFs

Latent-dynamic CRFs (LDCRF) (Morency et al., CVPR 2007; Sun et al., COLING 2008)

We can think (informally) it as “CRF + unsup. learning on latent info”
Latent-dynamic CRFs

**Latent-dynamic CRFs (LDCRF)**
(Morency et al., CVPR 2007; Sun et al., COLING 2008)

\[
P(y | x, \theta) = \sum_{h: \forall h_j \in H_{y_j}} P(h | x, \theta) = \sum_{h: \forall h_j \in H_{y_j}} \frac{1}{Z(x, \theta)} \exp \left( \sum_k \theta_k F_k (h, x) \right)
\]
Latent-dynamic CRFs (LDCRF)

- Training is slow
  - May need week-level time

Solution

- Perceptron is much faster than CRF
- Latent CRF -> **Latent perceptron**


For fast training of latent variable models
(Sun et al., IJCAI 2009; Sun et al., TKDE 2013)

Latent Variable Perceptron

Restricted Viterbi output by gold annotation

Global Viterbi output

A related work on machine translation: Liang et al., 2006
For fast training of latent variable models
(Sun et al., IJCAI 2009; Sun et al., TKDE 2013)

Latent Variable Perceptron

LDCRF:

\[
y^* = \arg \max_{y} \sum_{h: \text{Proj}(h) = y} P(h | x, \theta)
\]

Normally, batch training
(do weight update after go over all samples)

Latent variable perceptron (Sun et al., 2009):

\[
h^* = \arg \max_{h} P'(h | x, \theta)
\]

Online training
(do weight update on each sample)
## LDCRF training method

<table>
<thead>
<tr>
<th></th>
<th>Seg-0</th>
<th>Seg-1</th>
<th>Seg-2</th>
<th>noSeg-0</th>
<th>noSeg-1</th>
<th>noSeg-2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>⬤</td>
<td>⬤</td>
<td>⬤</td>
<td>⬤</td>
<td>⬤</td>
<td>⬤</td>
</tr>
</tbody>
</table>

These are her flowers.
**Latent Variable Perceptron**

- **LDCRF training method**

\[
y^* = \arg\max_y \sum_{h: \text{Proj}(h)=y} P(h|x, \theta)
\]

These are her flowers.
### Latent perceptron training

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seg-0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Seg-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Seg-2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>noSeg-0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>noSeg-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>noSeg-2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

These are her flowers.
Latent Variable Perceptron

- **Latent perceptron training**

\[
\theta^{i+1} = \theta^i + f([\arg\max_{\mathbf{h}} F(\mathbf{h} | y_i^*, x_i, \theta^i), x_i]) - f([\arg\max_{\mathbf{h}} F(\mathbf{h} | x_i, \theta^i), x_i])
\]
Theoretical guarantee of convergence

As far as traditional perceptron is separable, latent structured perceptron is also separable.

Theorem 1. Given the latent feature mapping \( m = (m_1, \ldots, m_n) \), for any sequence of training examples \((x_i, y_i^*)\) which is separable with margin \( \delta \) by a vector \( \mathbf{U} \) represented by \((\alpha_1, \ldots, \alpha_n)\) with \( \sum_{i=1}^{n} \alpha_i^2 = 1 \), the examples then will also be latently separable with margin \( \bar{\delta} \), and \( \bar{\delta} \) is bounded below by

\[
\bar{\delta} \geq \frac{\delta}{T},
\]

where \( T = \left( \sum_{i=1}^{n} m_i \alpha_i^2 \right)^{1/2} \).
Theoretical guarantee of convergence

Latent perceptron still converges

**Theorem 2.** For any sequence of training examples \( (x_i, y_i^*) \) which is separable with margin \( \delta \), the number of mistakes of the latent perceptron algorithm in Figure 1 is bounded above by

\[
\text{number of mistakes} \leq 2T^2 M^2 / \delta^2
\]

**Theorem 3.** For any training sequence \( (x_i, y_i^*) \), the number of mistakes made by the latent perceptron training algorithm is bounded above by

\[
\text{number of mistakes} \leq \min_{\overline{U}, \overline{\delta}} (\sqrt{2M} + D_{\overline{U}, \overline{\delta}}) \overline{\delta}^2 / \delta^2
\]
Experiment on **synthetic data:** Much better accuracy than CRF & Perceptron

Significance of latent info

Latent Perc
LDCRF
CRF
Averaged perceptron
Good performance in question answering (Fader et al. SIGKDD2014)

- Use query with partial annotation as latent variable

Figure 5: Training OQA on questions from all question sets leads to greater precision and recall than training on domain questions only.

Figure 6: OQA has higher precision and recall than the Open QA system Paralex.

OQA is based on latent variable perceptron

(Sun et al., IJCAI 2009; Sun et al., TKDE 2013)
Latent Variable Perceptron

- **Good performance in** speech sentence classification
  (Xu & Sarikaya, INTERSPEECH 2013)

- Split multi-intent using latent variable

- Based on latent variable perceptron  
  (Sun et al., IJCAI 2009; Sun et al., TKDE 2013)

![Graph showing intent detection accuracy (%) on SI test set with models trained on SI data plus various amounts of DI data.](image)

**Figure 4:** Intent detection accuracy (%) on SI test set with models trained on SI data plus various amounts of DI data.
Latent Variable Perceptron

- **Good performance in** semantic parsing (Zhou et al., IJCAI 2013)
- Hybrid tree as latent structure variable
- Based on latent variable perceptron (Sun et al., IJCAI 2009; Sun et al., TKDE 2013)

### Table 4: Performance comparison with other directly comparable systems over English corpus.

<table>
<thead>
<tr>
<th></th>
<th>10-fold cross-val</th>
<th>600 train/280 test</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Prec.</td>
<td>Rec.</td>
</tr>
<tr>
<td>WASP</td>
<td>87.2</td>
<td>74.8</td>
</tr>
<tr>
<td>LU</td>
<td>89.3</td>
<td>81.5</td>
</tr>
<tr>
<td>tsVB-hand</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>LVP+EXT</td>
<td><strong>90.9</strong></td>
<td><strong>90.9</strong></td>
</tr>
</tbody>
</table>

### Table 5: Performance comparison among models on the multilingual section of GeoQuery.

<table>
<thead>
<tr>
<th></th>
<th>German</th>
<th>Greek</th>
<th>Thai</th>
</tr>
</thead>
<tbody>
<tr>
<td>WASP</td>
<td><strong>87.1</strong></td>
<td>65.7</td>
<td>74.9</td>
</tr>
<tr>
<td>LU</td>
<td>76.4</td>
<td>62.1</td>
<td>68.5</td>
</tr>
<tr>
<td>tsVB-hand</td>
<td>74.6</td>
<td>74.6</td>
<td>74.6</td>
</tr>
<tr>
<td>LVP+EXT</td>
<td><strong>78.6</strong></td>
<td><strong>78.6</strong></td>
<td><strong>78.6</strong></td>
</tr>
</tbody>
</table>

Better Performance
Latent Variable Perceptron

- **Good performance in dependency parsing**
  (Honnibal & Johnson, TACL 2014)

- Dynamic oracle/transitions as latent variable
- Train with latent variable perceptron (Sun et al., IJCAI 2009; Sun et al., TKDE 2013)

<table>
<thead>
<tr>
<th>Model</th>
<th>P</th>
<th>R</th>
<th>F</th>
<th>UAS</th>
<th>LAS</th>
<th>w/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline joint</td>
<td>79.4</td>
<td>70.1</td>
<td>74.5</td>
<td>89.9</td>
<td>86.9</td>
<td>711</td>
</tr>
<tr>
<td>+Features</td>
<td>86.0</td>
<td>77.2</td>
<td>81.3</td>
<td>90.5</td>
<td>87.5</td>
<td>539</td>
</tr>
<tr>
<td>+Edit transition</td>
<td>92.2</td>
<td>80.2</td>
<td>85.8</td>
<td>90.9</td>
<td>87.9</td>
<td>555</td>
</tr>
<tr>
<td>Oracle pipeline</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>91.7</td>
<td>88.6</td>
<td>782</td>
</tr>
</tbody>
</table>

Table 1: Development results for the joint models. For the baseline model, disfluencies reduce parse accuracy by 1.7% Unlabelled Attachment Score (UAS). Our features and Edit transition reduce the gap to 0.7%, and improve disfluency detection by 11.3% F-measure.

<table>
<thead>
<tr>
<th>Model</th>
<th>Disfl. F</th>
<th>UAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Johnson et al pipeline</td>
<td>82.1</td>
<td>90.3</td>
</tr>
<tr>
<td>Qian and Liu pipeline</td>
<td>83.9</td>
<td>90.1</td>
</tr>
<tr>
<td>Baseline joint parser</td>
<td>73.9</td>
<td>89.4</td>
</tr>
<tr>
<td>Final joint parser</td>
<td>84.1</td>
<td><strong>90.5</strong></td>
</tr>
</tbody>
</table>

Table 2: Test-set parse and disfluency accuracies. The joint parser is improved by the features and Edit transition, and is better than pre-processing the text with state-of-the-art disfluency detectors.

Much better compared with baseline model
Good performance in coreference resolution

(Fernandes et al., CL 2012)

- Use structures of coreference trees as latent variable
- Based on latent variable perceptron (Sun et al., IJCAI 2009; Sun et al., TKDE 2013)

<table>
<thead>
<tr>
<th>Language</th>
<th>Parse / Mentions</th>
<th>MUC R</th>
<th>MUC P</th>
<th>MUC F1</th>
<th>B^4 R</th>
<th>B^4 P</th>
<th>B^4 F1</th>
<th>CEAF_e R</th>
<th>CEAF_e P</th>
<th>CEAF_e F1</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arabic</td>
<td>Auto / GB</td>
<td>45.18</td>
<td>47.39</td>
<td>46.26</td>
<td>64.56</td>
<td>69.44</td>
<td>66.91</td>
<td>49.73</td>
<td>47.39</td>
<td>48.53</td>
<td>53.90</td>
</tr>
<tr>
<td></td>
<td>Auto / GM</td>
<td>57.25</td>
<td>76.48</td>
<td>65.48</td>
<td>60.27</td>
<td>79.81</td>
<td>68.68</td>
<td>72.61</td>
<td>46.00</td>
<td>56.32</td>
<td>63.49</td>
</tr>
<tr>
<td></td>
<td>Golden / Auto</td>
<td>46.38</td>
<td>51.78</td>
<td>48.93</td>
<td>63.53</td>
<td>72.37</td>
<td>67.66</td>
<td>52.57</td>
<td>46.88</td>
<td>49.56</td>
<td>55.38</td>
</tr>
<tr>
<td></td>
<td>Golden / GB</td>
<td>46.38</td>
<td>51.78</td>
<td>48.93</td>
<td>63.53</td>
<td>72.37</td>
<td>67.66</td>
<td>52.57</td>
<td>46.88</td>
<td>49.56</td>
<td>55.38</td>
</tr>
<tr>
<td></td>
<td>Golden / GM</td>
<td>56.89</td>
<td>76.27</td>
<td>65.17</td>
<td>60.07</td>
<td>80.02</td>
<td>68.62</td>
<td>72.24</td>
<td>45.58</td>
<td>55.90</td>
<td>63.23</td>
</tr>
<tr>
<td>Chinese</td>
<td>Auto / GB</td>
<td>58.76</td>
<td>71.46</td>
<td>64.49</td>
<td>66.62</td>
<td>79.88</td>
<td>72.65</td>
<td>54.09</td>
<td>42.02</td>
<td>47.29</td>
<td>61.48</td>
</tr>
<tr>
<td></td>
<td>Auto / GM</td>
<td>61.64</td>
<td>90.81</td>
<td>73.43</td>
<td>63.55</td>
<td>89.43</td>
<td>74.30</td>
<td>72.78</td>
<td>39.68</td>
<td>51.36</td>
<td>66.36</td>
</tr>
<tr>
<td></td>
<td>Golden / Auto</td>
<td>59.35</td>
<td>74.49</td>
<td>66.07</td>
<td>66.31</td>
<td>81.43</td>
<td>73.10</td>
<td>55.97</td>
<td>41.50</td>
<td>47.66</td>
<td>62.28</td>
</tr>
<tr>
<td></td>
<td>Golden / GB</td>
<td>59.35</td>
<td>74.49</td>
<td>66.07</td>
<td>66.31</td>
<td>81.43</td>
<td>73.10</td>
<td>55.97</td>
<td>41.50</td>
<td>47.66</td>
<td>62.28</td>
</tr>
<tr>
<td></td>
<td>Golden / GM</td>
<td>61.70</td>
<td>91.45</td>
<td>73.69</td>
<td>63.57</td>
<td>89.76</td>
<td>74.43</td>
<td>72.84</td>
<td>39.49</td>
<td>51.21</td>
<td>66.44</td>
</tr>
<tr>
<td>English</td>
<td>Auto / GB</td>
<td>64.92</td>
<td>77.53</td>
<td>70.67</td>
<td>64.25</td>
<td>78.95</td>
<td>70.85</td>
<td>56.48</td>
<td>41.69</td>
<td>47.97</td>
<td>63.16</td>
</tr>
<tr>
<td></td>
<td>Auto / GM</td>
<td>70.69</td>
<td>91.21</td>
<td>79.65</td>
<td>65.46</td>
<td>85.61</td>
<td>74.19</td>
<td>74.71</td>
<td>42.55</td>
<td>54.22</td>
<td>69.35</td>
</tr>
<tr>
<td></td>
<td>Golden / Auto</td>
<td>67.73</td>
<td>77.25</td>
<td>72.18</td>
<td>66.42</td>
<td>78.01</td>
<td>71.75</td>
<td>56.16</td>
<td>44.51</td>
<td>49.66</td>
<td>64.53</td>
</tr>
<tr>
<td></td>
<td>Golden / GB</td>
<td>65.65</td>
<td>78.26</td>
<td>71.40</td>
<td>64.36</td>
<td>79.09</td>
<td>70.97</td>
<td>57.36</td>
<td>42.23</td>
<td>48.65</td>
<td>63.67</td>
</tr>
<tr>
<td></td>
<td>Golden / GM</td>
<td>71.18</td>
<td>91.24</td>
<td>79.97</td>
<td>65.81</td>
<td>85.51</td>
<td>74.38</td>
<td>74.93</td>
<td>43.09</td>
<td>54.72</td>
<td>69.69</td>
</tr>
</tbody>
</table>

Table 4: Supplementary results on the test sets alternating parse quality and mention candidates. Parse quality can be automatic or golden; and mention candidates can be automatically identified (Auto), golden mention boundaries (GB) or golden mentions (GM).
Latent Variable Perceptron

Proposed by Sun (2009, 2013)

Fast and accurate

- Accuracy equal or even better than CRF
- Almost as fast as perceptron
- Suitable for large-scale structured prediction problems


Latent Model can reduce annotation engineering

Furthermore, how to reduce the cost of feature engineering?

Neural Model

Automatically extract features
Outline

- Conditional Random Field
- Structured Perceptron
- MIRA
- Probabilistic Perceptron

Conventional Model

Reduce Annotation Engineering

- Latent Dynamic CRF
- Latent Variable Perceptron

Latent Model

- Recurrent Neural Network
- Long Short-term Memory
- LSTM-CRF

Neural Model

Reduce Feature Engineering
Motivation

- **Problem in feature engineering**
  - Require **linguistics** knowledge
  - A lot of feature templates
  - Ad-hoc, some features are not very reasonable
  - Task-sensitive

- **Neural networks**
  - Automatically learn features in hidden layers
Many kinds

- **Feed Forward NN**
  - logistic regression

- **Convolutional NN**
  - image processing

- **Recursive/Recurrent NN**
  - structured prediction
Recursive neural network (Socher et al., ICML 2011)

- Model hierarchical structures
- Condition on each sub-structure independently
Recurrent Neural Network (RNN)

- **Recurrent neural network** (Elman, Cognitive Science 1990)
  - Model time series
  - Predict linear-chain structures
  - Conditioned on all previous input

\[
h_t = f(Uh_{t-1} + WX_t)
\]

\[
\hat{y}_t = \text{softmax}(W^{(s)}h_t)
\]
Problems

- Gradient Exploding/Vanishing (Pascanu et al., ICML 2013)
- Hard to capture long-term dependencies
Long Short-term Memory (LSTM)

- **Long short-term memory** (Hochreiter and Schmidhuber 1997)
  - A lasting linear memory
    - Capture long distance dependency
  - Three gates: input, forget and output gates
    - Control modification to the memory

![LSTM Diagram](image-url)
Introduction of Gates

- **Gate**
  - Sigmoid-activated layer
    - Output value from 0 to 1
  - Member-wise multiplication
    - How much to **flow through**

- **Gates in LSTM**
  - Forget gate
    - How much **old memory** needs to be **remembered**

Some Picture from Christopher Olah
Introduction of Gates

- **Gate**
  - Sigmoid-activated layer
    - Output value from 0 to 1
  - Member-wise multiplication
    - How much to flow through

- **Gates in LSTM**
  - Forget gate
    - How much *old memory* needs to be remembered
  - Input gate
Introduction of Gates

- **Gate**
  - Sigmoid-activated layer
    - Output value from 0 to 1
  - Member-wise multiplication
    - How much to flow through

- **Gates in LSTM**
  - Forget gate
    - How much old memory needs to be remembered
  - Input gate
  - Output gate

Some Picture from Christopher Olah
Introduction of Gates

- **RNN**
  - Vanishing influence

- **LSTM**
  - Situation-aware
Problem of LSTM

- Can be slow for large models (hard to train)
  - Structure complexity is higher
  - Structure redundancy?
  - Too much parameters

Picture from Christopher Olah
Refinement of LSTM

- Peepholes
  - Introduced by Gers (2000)
  - Include current memory when compute gates
- **Fully connected gates**
  - Included in Hochreiter and Schmidhuber (1997)
  - Gates are also recurrent

![Diagram of LSTM](Picture from Christopher Olah)
Gated recurrent unit (GRU)

- Introduced by Chung et al. (2014)
- Coupled forget and input gates with structure simplification
More recent development

- **Sequence to sequence neural network** (Sutskever et al., NIPS 2014)
  - Encoder & Decoder
  - The encoder information is stored in a fixed-length vector

![Diagram of sequence to sequence neural network](image-url)
More recent development

- **Sequence to sequence neural network** (Sutskever et al., NIPS 2014)
  - Encoder & Decoder
  - The encoder information is stored in a fixed-length vector

![Diagram of sequence to sequence neural network]

- Learn to align
- Neural machine translation
- **Attention-based neural network** (Bahdanau et al. 2014; Luong et al. 2015)

- Each hidden state has an **unique** weight/attention/importance
Training large-scale neural models is costly
- Numerous parameters
- Very slow
- A NMT model may take weeks (even months) to train

How to accelerate training speed?
- Parallel training
- Especially, asynchronous (lock-free) parallel training
Asynchronous Parallel Learning

- **Motivation**

  - Asynchronous parallel learning is very popular for traditional **sparse** feature models
    - E.g., *HogWild!* (Niu et al. NIPS 2011)

  - However, previous asynchronous parallel learning methods do not suit **neural networks**
    - Because NN is **dense** feature model
    - Previous parallel learning for dense feature models is mostly synchronous, e.g., *mini-batch parallel learning, GPU parallel learning*
Motivation

Asynchronous parallel learning is very popular for traditional sparse feature models

- E.g., *HogWild!* (Niu et al. NIPS 2011)

1. Simple Case  →  No problem

- **Reading** parameters from shared memory
- **Computing** gradients
- **Writing** parameters to shared memory

(a) Simple case
Asynchronous Parallel Learning

**Motivation**

Asynchronous parallel learning is very popular for traditional **sparse** feature models.

- E.g., *HogWild!* (Niu et al. NIPS 2011)

2. This case is called Gradient Delay case

→ More complicated, but problem solved for sparse feature models (Niu et al. NIPS 2011)
Motivation

Asynchronous parallel learning is very popular for traditional sparse feature models.

- E.g., *HogWild!* (Niu et al. NIPS 2011)
3. Even more difficult case: **Gradient Error Case**

- Happens for dense feature models, like neural networks
  - Actions (R, G & W) are time-consuming
- Read-overwrite and write-overwrite problems

→ Not well studied before, how to deal with this problem?
Gradient error is **inevitable** in asynchronous training of neural networks in real-world tasks.
Gradient error is inevitable in asynchronous training of neural networks in real-world tasks.

So asynchronous parallel learning is doomed for neural networks?
→ No, still this problem can be solved.
A Recent Solution

- An asynchronous parallel learning solution for fast training of neural networks Proposed by Sun (COLING 2016)
  - Asynchronous Parallel Learning with Gradient Error (AsynGrad)
- Algorithm

**Algorithm 1 AsynGrad: Asynchronous Parallel Learning with Gradient Error**

**Input:** model weights $w$, training set $S$ of $m$ samples  
Run $k$ threads in parallel with share memory, and procedure of each thread is as follows:

**repeat**
- Get a sample $z$ uniformly at random from $S$
- Get the update term $s_z(w)$, which is computed as $\nabla f_z(w)$ but usually contains error
- Update $w$ such that $w \leftarrow w - \gamma s_z(w)$
**until** Convergence

**return** $w$

Can AsynGrad still converge with gradient errors?

Even though there are gradient errors, AsynGrad does not diverge... it still converges near the optimum with a small distance, when the errors are bounded.

→ The assumptions usually hold in the final convergence region

→ Confirmed by real-world experiments

\[ t = \frac{\beta q k^2 \log (qa_0/\epsilon)}{c(\epsilon \epsilon - 2\tau q)} \]

where \( \hat{\tau} \) means ceil-rounding of a real value to an integer, and \( a_0 \) is the initial distance such that \( a_0 = ||w_0 - w^*||^2 \). Then, after \( t \) updates of \( w \), AsynGrad converges towards the optimum such that \( \mathbb{E}[f(w_t) - f(w^*)] \leq \epsilon \), as far as the gradient errors are bounded such that

\[ \tau \leq \frac{c \epsilon}{2q} \]
Experiments show that AsynGrad still converge even with a high gradient error rate.
Experiments on LSTM

- No loss on accuracy/F-score
- With substantially faster training speed

AsynGrad
Gradient errors are common and inevitable in asynchronous training of dense feature models.

AsynGrad tolerates gradient errors

- For dense feature models, such as neural networks and dense-CRF
- With faster speed and no loss on accuracy

An alternative learning approach for large-scale structured predictions using neural networks.
Thanks!

Any questions until now?
Outline

Introduction

Model

- Conventional Model
- Latent Model
- Neural Model

Regularization

How-tos
Models for large-scale structured prediction often suffer from **overfitting**

**Overfitting**
- Low error rate in training set
- High error rate in test set

**Why overfitting?**
- Complex model
- Too many parameters, too little data

**How to deal with?**
- Penalty
- Reduce complexity
Weight Regularization

- **Penalty parameters in loss function**
  \[
  \min_w \text{loss}(x, y, w) + \lambda \text{regularizer}(w)
  \]

- **L1 regularizer**
  \[
  \text{regularizer} = \lambda \|w\|
  \]
  \[
  \frac{d}{dw_j} \text{regularizer} = \lambda \text{sign}(w_j)
  \]

- **L2 regularizer**
  \[
  \text{regularizer} = \frac{\lambda}{2} \|w\|^2
  \]
  \[
  \frac{d}{dw_j} \text{regularizer} = \lambda w_j
  \]
Motivation

- Reduce complexity of model structure

Structure regularization (Sun. NIPS 2014)

- Complex structure \(\rightarrow\) Simple structure
- Faster
- Easy to implement
- Theoretical guarantee
- **Complex structures** (high complexity)
  - Illustration

- **Simple structures** (low complexity)
  - Illustration
Structure Regularization

- Structure regularization (SR) can find good complexity
  - Simply split the structures!
  - Can (almost) be seen as a preprocessing step of the training data
Will the split causes feature loss? – loss of long distance features?

No loss of any (long distance) features
→ We can first extract features, then split the structures
→ Or, by simply copying observations to mini-samples, i.e., the split is only on tag-structures, like this:
Is structure regularization also required for test data?

No, no use of SR for testing data (in current implementation & experiments)

→ Like other regularization methods, SR is only for the training

→ i.e., No SR on the test stage (no decomposition of test samples)!
Structure & weight regularization

\[ R_{\alpha, \lambda}(G_S) \triangleq R_{\alpha}(G_S) + N_{\lambda}(G_S) \]

Algorithm 1 Training with structure regularization

1: Input: model weights \( w \), training set \( S \), structure regularization strength \( \alpha \)
2: repeat
3: \( S' \leftarrow \emptyset \)
4: for \( i = 1 \rightarrow m \) do
5: Randomly decompose \( z_i \in S \) into mini-samples \( N_{\alpha}(z_i) = \{z_{i,1}, \ldots, z_{i,\alpha}\} \)
6: \( S' \leftarrow S' \cup N_{\alpha}(z_i) \)
7: end for
8: for \( i = 1 \rightarrow |S'| \) do
9: Sample \( z' \) uniformly at random from \( S' \), with gradient \( \nabla g_{z'}(w) \)
10: \( w \leftarrow w - \eta \nabla g_{z'}(w) \)
11: end for
12: until Convergence
13: return \( w \)

The implementation is very simple
Theoretical Analysis: Overfitting Risk

**Theorem 4 (Generalization vs. structure regularization)** Let the structured prediction objective function of \( G \) be penalized by structure regularization with factor \( \alpha \in [1, n] \) and \( L_2 \) weight regularization with factor \( \lambda \), and the penalized function has a minimizer \( f \):

\[
f = \arg \min_{g \in \mathcal{F}} R_{\alpha, \lambda}(g) = \arg \min_{g \in \mathcal{F}} \left( \frac{1}{mn} \sum_{j=1}^{m_\alpha} \mathcal{L}_\tau(g, z'_j) + \frac{\lambda}{2} \|g\|_2^2 \right)
\]  

(8)

Assume the point-wise loss \( \ell_\tau \) is convex and differentiable, and is bounded by \( \ell_\tau(f, z, k) \leq \gamma \). Assume \( f(x, k) \) is \( \rho \)-admissible. Let a local feature value be bounded by \( v \) such that \( x_{(k, q)} \leq v \) for \( q \in \{1, \ldots, d\} \). Then, for any \( \delta \in (0, 1) \), with probability at least \( 1 - \delta \) over the random draw of the training set \( S \), the generalization risk \( R(f) \) is bounded by

\[
R(f) \leq R_e(f) + \frac{2d\tau^2 \rho^2 v^2 n^2}{m\lambda\alpha} + \left( \frac{(4m - 2)d\tau^2 \rho^2 v^2 n^2}{m\lambda\alpha} + \gamma \right) \sqrt{\frac{\ln \delta^{-1}}{2m}}
\]  

(9)

**Expected risk**
(risk on test data)

**Empirical risk**
(risk on training data)

**Overfitting risk**
(risk of overfitting from training data to test data)
Theoretical Analysis: Overfitting Risk

**Theorem 4 (Generalization vs. structure regularization)** Let the structured prediction objective function of $G$ be penalized by structure regularization with factor $\alpha \in [1, n]$ and $L_2$ weight regularization with factor $\lambda$, and the penalized function has a minimizer $f$:

$$f = \arg\min_{g \in \mathcal{F}} R_{\alpha,\lambda}(g) = \arg\min_{g \in \mathcal{F}} \left( \frac{1}{mn} \sum_{j=1}^{m\alpha} \mathcal{L}_\tau(g, z'_j) + \frac{\lambda}{2} \|g\|_2^2 \right)$$  \hspace{1cm} (8)

Assume the point-wise loss $\ell_\tau$ is convex and differentiable, and is bounded by $\ell_\tau(f, z, k) \leq \gamma$. Assume $f(x, k)$ is $\rho$-admissible. Let a local feature value be bounded by $v$ such that $x_{(k,q)} \leq v$ for $q \in \{1, \ldots, d\}$. Then, for any $\delta \in (0, 1)$, with probability at least $1 - \delta$ over the random draw of the training set $S$, the generalization risk $R(f)$ is bounded by

$$R(f) \leq R_e(f) + \frac{2d^2 \rho^2 v^2 n^2}{m\lambda\alpha} + \left( \frac{(4m - 2)d^2 \rho^2 v^2 n^2}{m\lambda\alpha} + \gamma \right) \sqrt{\frac{\ln \delta^{-1}}{2m}}$$  \hspace{1cm} (9)

**Complexity of structure (nodes of a training sample with structured dependencies)**

$\rightarrow$ Complex structure leads to higher overfitting risk
Theoretical Analysis: Overfitting Risk

Theorem 4 (Generalization vs. structure regularization) Let the structured prediction objective function of $G$ be penalized by structure regularization with factor $\alpha \in [1, n]$ and $L_2$ weight regularization with factor $\lambda$, and the penalized function has a minimizer $f$:

$$f = \arg\min_{g \in \mathcal{F}} R_{\alpha, \lambda}(g) = \arg\min_{g \in \mathcal{F}} \left( \frac{1}{mn} \sum_{j=1}^{m_{\alpha}} \mathcal{L}_\tau(g, z'_j) + \frac{\lambda}{2} ||g||_2^2 \right) \quad (8)$$

Assume the point-wise loss $\ell_\tau$ is convex and differentiable, and is bounded by $\ell_\tau(f, z, k) \leq \gamma$. Assume $f(x, k)$ is $\rho$-admissible. Let a local feature value be bounded by $\nu$ such that $x_{(k, q)} \leq \nu$ for $q \in \{1, \ldots, d\}$. Then, for any $\delta \in (0, 1)$, with probability at least $1 - \delta$ over the random draw of the training set $S$, the generalization risk $R(f)$ is bounded by

$$R(f) \leq R_e(f) + \frac{2d^2 \rho^2 \nu^2 n^2}{m\lambda\alpha} + \left( \frac{(4m - 2)d^2 \rho^2 \nu^2 n^2}{m\lambda\alpha} + \gamma \right) \sqrt{\frac{\ln \delta^{-1}}{2m}} \quad (9)$$

Strength of structure regularization (strength of decomposition) 
→ Stronger SR leads to reduction of overfitting risk
Theoretical Analysis: Overfitting Risk

Theorem 4 (Generalization vs. structure regularization) Let the structured prediction objective function of $G$ be penalized by structure regularization with factor $\alpha \in [1, n]$ and $L_2$ weight regularization with factor $\lambda$, and the penalized function has a minimizer $f$:

$$
  f = \arg\min_{g \in F} R_{\alpha, \lambda}(g) = \arg\min_{g \in F} \left( \frac{1}{mn} \sum_{j=1}^{mn} L_{\tau}(g, z'_j) + \frac{\lambda}{2} ||g||_2^2 \right)
$$

Assume the point-wise loss $l_{\tau}$ is convex and differentiable, and is bounded by $l_{\tau}(f, z, k) \leq \gamma$. Assume $f(x, k)$ is $\rho$-admissible. Let a local feature value be bounded by $v$ such that $x_{(k, q)} \leq v$ for $q \in \{1, \ldots, d\}$. Then, for any $\delta \in (0, 1)$, with probability at least $1 - \delta$ over the random draw of the training set $S$, the generalization risk $R(f)$ is bounded by

$$
  R(f) \leq R_e(f) + \frac{2d\tau^2 \rho^2 v^2 n^2}{m \lambda \alpha} + \left( \frac{(4m - 2)d\tau^2 \rho^2 v^2 n^2}{m \lambda \alpha} + \gamma \right) \sqrt{\frac{\ln \delta^{-1}}{2m}}
$$

Number of training samples

→ More training samples leads to reduction of overfitting risk
Conclusions from our analysis:

1. Complex structure $\rightarrow$ low empirical risk & high overfitting risk
2. Simple structure $\rightarrow$ high empirical risk & low overfitting risk
3. Need a balanced complexity of structures
Theoretical Analysis: Overfitting Risk

**Theorem 4 (Generalization vs. structure regularization)** Let the structured prediction objective function of $G$ be penalized by structure regularization with factor $\alpha \in [1, n]$ and $L_2$ weight regularization with factor $\lambda$, and the penalized function has a minimizer $f$:

$$f = \arg\min_{g \in \mathcal{F}} R_{\alpha, \lambda}(g) = \arg\min_{g \in \mathcal{F}} \left( \frac{1}{mn} \sum_{j=1}^{m\alpha} \mathcal{L}_\tau(g, z'_j) + \frac{\lambda}{2} ||g||_2^2 \right) \tag{8}$$

Assume the point-wise loss $\ell_\tau$ is convex and differentiable, and is bounded by $\ell_\tau(f, z, k) \leq \gamma$. Assume $f(\mathbf{x}, k)$ is $\rho$-admissible. Let a local feature value be bounded by $v$ such that $\mathbf{x}_{(k,q)} \leq v$ for $q \in \{1, \ldots, d\}$. Then, for any $\delta \in (0, 1)$, with probability at least $1 - \delta$ over the random draw of the training set $S$, the generalization risk $R(f)$ is bounded by

$$R(f) \leq R_c(f) + \frac{2d\tau^2 \rho^2 v^2 n^2}{m\lambda \alpha} + \left( \frac{(4m - 2)d\tau^2 \rho^2 v^2 n^2}{m\lambda \alpha} + \gamma \right) \sqrt{\frac{\ln \delta^{-1}}{2m}} \tag{9}$$

- **In other words, more intuitively:**
  1. Too complex structure $\rightarrow$ high accuracy on training + very easy to overfit $\rightarrow$ low accuracy on testing
  2. Too simple structure $\rightarrow$ very low accuracy on training + not easy to overfit $\rightarrow$ low accuracy on testing

Proper structure $\rightarrow$ good accuracy on training + not easy to overfit $\rightarrow$ high accuracy on testing
1. Simple structure $\rightarrow$ low overfitting risk & high empirical risk
2. Complex structure $\rightarrow$ high overfitting risk & low empirical risk
3. Need a balanced complexity of structures

Some intuition in the proof (as in the full version paper):
1) The decomposition can improve stability
2) Better stability leads to better generalization (less overfitting)
Proosition 5 (Convergence rates vs. structure regularization) With the aforementioned assumptions, let the SGD training have a learning rate defined as \( \eta = \frac{c \epsilon \beta \alpha^2}{q \kappa^2 n^2} \), where \( \epsilon > 0 \) is a convergence tolerance value and \( \beta \in (0, 1] \). Let \( t \) be an integer satisfying

\[
    t \geq \frac{q \kappa^2 n^2 \log (q a_0 / \epsilon)}{\epsilon \beta c^2 \alpha^2}
\]  

(15)

where \( n \) and \( \alpha \in [1, n] \) is like before, and \( a_0 \) is the initial distance which depends on the initialization of the weights \( w_0 \) and the minimizer \( w^* \), i.e., \( a_0 = ||w_0 - w^*||^2 \). Then, after \( t \) updates of \( w \) it converges to \( \mathbb{E}[g(w_t) - g(w^*)] \leq \epsilon \).

- SR also with faster speed

(a by-product of simpler structures)

- using structure regularization can quadratically accelerate the convergence rate
Some Advantages

- If the original obj. function is convex, can still keep the convexity of the objective function
- No conflict with the weight regularization
  - E.g, L2, and/or L1 regularization
- General purpose and model-independent (because act like a preprocessing step)
  - E.g., can be used for different types of models, including CRFs, perceptrons, & neural networks
Experiments-1: Accuracy

State-of-the-art scores on competitive tasks
Experiments-2: Learning Speed

- POS-Tagging: CRF
- Bio-NER: CRF
- Word-Seg: CRF
- Act-Recog: CRF

- POS-Tagging: Perc
- Bio-NER: Perc
- Word-Seg: Perc
- Act-Recog: Perc

- Also with faster speed
- (a by-product of simpler structures)
Question: Is structure complexity matters in structured prediction?

Theoretical analysis to the question

1) Yes it matters
2) High complexity of structures $\rightarrow$ high overfitting risk
3) Low complexity $\rightarrow$ high empirical risk
4) We need to find an optimal complexity of structures

Proposed a solution

- Split the original structure to find the optimal complexity
- Better accuracies in real tasks, & faster (a by-product)

This work is published at NIPS 2014:
Drop Out

- **Proposed by** Srivastava et al. (2014)
- **Part of neurons do not participate in forward pass and backpropagation**
- **Only use in training**

**Advantage**
- Fewer parameters per training sample
- Reduce training time
Experiment on MINST (Srivastava, et al. JMLR 2014)
- **Experiment on Penn Tree Bank** *(Zaremba, et al. 2015)*

- Language modeling measured by perplexity

<table>
<thead>
<tr>
<th>Model</th>
<th>Validation set</th>
<th>Test set</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A single model</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pascanu et al. (2013)</td>
<td>107.5</td>
<td></td>
</tr>
<tr>
<td>Cheng et al.</td>
<td>100.0</td>
<td></td>
</tr>
<tr>
<td>non-regularized LSTM</td>
<td>120.7</td>
<td>114.5</td>
</tr>
<tr>
<td><strong>Medium regularized LSTM</strong></td>
<td><strong>86.2</strong></td>
<td><strong>82.7</strong></td>
</tr>
<tr>
<td><strong>Large regularized LSTM</strong></td>
<td><strong>82.2</strong></td>
<td><strong>78.4</strong></td>
</tr>
</tbody>
</table>

| Model averaging                            |                |          |
|--------------------------------------------|                |          |
| Mikolov (2012)                             | 83.5           |          |
| Cheng et al.                               | 80.6           |          |
| 2 non-regularized LSTMs                    | 100.4          | 96.1     |
| 5 non-regularized LSTMs                    | 87.9           | 84.1     |
| 10 non-regularized LSTMs                   | 83.5           | 80.0     |
| **2 medium regularized LSTMs**             | **80.6**       | **77.0** |
| **5 medium regularized LSTMs**             | **76.7**       | **73.3** |
| **10 medium regularized LSTMs**            | **75.2**       | **72.0** |
| 2 large regularized LSTMs                  | 76.9           | 73.6     |
| 10 large regularized LSTMs                 | 72.8           | 69.5     |
| 38 large regularized LSTMs                 | 71.9           | **68.7** |

<table>
<thead>
<tr>
<th>Model averaging with dynamic RNNs and n-gram models</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Mikolov &amp; Zweig (2012)</td>
<td>72.9</td>
</tr>
</tbody>
</table>
Discussion of Techniques

- **StrutReg**
  - decomposition of structure
  - randomly

- **LSTM**
  - forget useless information
  - learned

- **Drop Out**
  - deactivate neurons
  - randomly

- **ReLU**
  - sparse activation
  - forced

Picture credit: Christopher Olah
Discussion of Techniques

Relations?

- Struct Reg
- Drop Out
- ReLU
- LSTM

reduce complexity
How to choose a model

- Conventional models can achieve the state-of-the-arts results
  - stable for production setup

- Latent models are good for tasks short of annotations

- Neural models are promising
  - especially for high-level tasks
    - sentiment analysis
    - summarization, composition, translation
  - many nice frameworks available
    - theano, torch, tensorflow
    - caffe, cntk
How to Choose an Optimizer

- For models other than neural models
  - Perceptron
    - online, fast convergence

- For neural models
  - Mini-batch SGD
    - parameter-scaling: AdaGrad, AdaDelta, Adam, RMSProp
      - less tuning, fast
    - momentum-based: momentum, Nesterov Accelerated Gradient (NAG)
      - better results, more tuning

- Using GPUs can lead to significant speed-ups, compared to use CPUs only
Visualization of Optimization Algorithms

Long Valley

Picture from Alec Radford
Saddle Point

Picture from Alec Radford
Beale's Function
Noisy Data
Thanks!
Any Question?


References And Further Reading


76. X. Sun. Asynchronous Parallel Learning for Neural Networks and Structured Models with Dense Features. COLING 2016.

78. X. Sun, T. Matsuzaki, and W. Li. Latent structured perceptrons for large-scale learning with hidden information. TKDE 2013.


Models & Regularization

- Conventional Model
- Latent Model
- Neural Model

Structures & Applications

- Sequence Structure
- Tree/Graph Structure

Xu SUN

Yansong FENG
Type of Structures in Application

Yansong FENG
Peking University
fengyansong@pku.edu.cn
Outline

Sequence Tagging

- Chinese Word Segmentation
- POS Tagging/Named Entity Recognition

Tree/Graph Structure

Take-home Msg
Sequence Tagging

- **Sequence**
  - the most basic/simplest structures in NLP

- **Tag each item in the sequence using a given label set**

- **Common to see**
  - World segmentation
  - Part-of-Speech tagging
  - Named entity recognition
  - Chunking
  - Event trigger identification
  - ...
Conventional models

- CRF, Structured perceptron, ...

Work well, but

- Feature engineering
- Local context / Global information
- ...

Neural models

- CNN, RNN, LSTM, LSTM-CRF,...

Performances

- Comparable to state of the arts
- Combinations may give top performances
The Task

(The ground is covered with thick snow)

This area is really not small

Challenges

- Feature engineering
- Long distance dependencies

Neural Network Practice

- CNN/RNN to capture local information, instead of fixed windows
- RNN to capture long distance dependencies, or sentence-level/global information
Model Local Information

- **NN to model local features**
  - TNN to capture tag/word features and their combinations
  - CNN, RNN, LSTM, BLSTM to capture local features, beyond fixed window size

- **Choice of character/word level**
  - Word level features are still important, but not easy to incorporate in models
  - Explore word level information in a beam-search framework [Zhang et al., 2016, Cai and Zhao, 2016]
  - Word level features give 0.5%

- **Combinations**
  - Combining neural and discrete features gives top performances
Global View

- **Long Distance Dependencies**
  - Gated Recursive NN [Chen et al., 2015, Xu and Sun, 2016]
  - LSTM [Chen et al., 2015b, Zhang et al., 2016, Cai and Zhao, 2016]

- **Search Strategy**
  - CRF framework
    - Viterbi
  - Beam-search style
    - Fully explore word level information
    - Transition based [Zhang et al., 2016]
    - Beam search [Cai and Zhao, 2016]
Gated Recursive Neural Networks

Dependency based Gated Recursive Neural Networks

Window Context

“This area is really not small.”

块(C_i-2)  地(C_i-1)  面(C_i)  积(C_i+1)  还(C_i+2)

Layer 1

Layer 2

Layer 3

Output Layer

Rainy
Models

- LSTM

- LSTM + Gated Combination Neural Networks

```
\begin{align*}
&\text{Scoring} \\
&\text{Predicting} \\
&\text{LSTM Unit} \\
&\text{GCNN Unit} \\
&\text{Lookup Table} \\
\end{align*}
```

[Chen et al., 2015b][Cai and Zhao, 2016]
Conventional models are still strong

Neural models can be promising, and sometimes complementary to conventional models

- Long distance dependencies
- Word level features
- BiLSTM works to capture local context information
- Various NN models to sentence level/long-distance information
- CRF is still attractive
On PKU and MSR

- Sun et al., 2012
- Zhang et al., 2013
- Pei et al., 2014
- Chen et al., 2015a
- Chen et al., 2015b
- Xu and Sun, 2016
- Zhang et al., 2016
- Cai and Zhao, 2016

Graph showing performance metrics for PKU and MSR.
Typical Sequence Labeling tasks

- Conventional models have achieved over ~90%

Again

- Feature engineering
- Language issues
- Local/Global
- Label bias

Neural Network Practice

- BLSTM/CNN to capture local context, both forward and backward
- CRF with Viterbi to find the best sequence
NN models seem to be capable of handling language issues to some extent

- CNN
- BLSTM → Dominating!
- Character level modeling

BLSTM works better than LSTM

- Look at both past and future

Traditional lexicon features are still there

- Extra resources, like dictionary, gazetteers, or Wiki, are always welcome
- BLSTM-CRF with feature concatenation
- work nice for POS, NER, Chunking

[Huang et al., 2015]
Typical BLSTM-CRF for POS / NER

Word / Character Level

[1] Lample et al., 2016
[2] Ling et al., 2015
**BLSTM + CNN for NER**

- **Word Embedding**
- **Additional Word Features**
- **CNN-extracted Char Features**

<table>
<thead>
<tr>
<th>Forward LSTM</th>
<th>We</th>
<th>saw</th>
<th>paintings</th>
<th>of</th>
<th>Picasso</th>
</tr>
</thead>
<tbody>
<tr>
<td>Backward LSTM</td>
<td>LSTM</td>
<td>LSTM</td>
<td>LSTM</td>
<td>LSTM</td>
<td>LSTM</td>
</tr>
<tr>
<td>Output Layers</td>
<td>Out</td>
<td>Out</td>
<td>Out</td>
<td>Out</td>
<td>Out</td>
</tr>
<tr>
<td>Tag Scores</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Best Tag Sequence</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>S-PER</td>
</tr>
</tbody>
</table>

- **Character Embedding**
- **Additional Char Features**
- **Convolution**
- **Max**
- **CNN-extracted Char features**

[Chiu and Nicols., 2016]
Language Issues

- Handling with different languages
- Handling with OOV
  - using CNN
  - character-level modeling

[dos Santos and Guimaraes, 2015] [Ling et al., 2015]
POS Tagging On WSJ

Collobert et al., 2011
Wang et al., 2015
Huang et al., 2015
Ling et al., 2015
Andor et al., 2016
NER On CoNLL

CoNLL03

Collobert et al., 2011
Wang et al., 2015
Huang et al., 2015
Passos et al., 2014
Chiu and Nichols, 2016
Lample et al., 2016

CoNLL03
- Event Trigger Identification
  
  犯罪嫌疑人都落入法网
  
  The suspects were arrested

- NN models work to help
  
  - Feature extraction
  
  - Both local and global features (CNN, BLSTM)
  
  - Language issues: character level modeling
**Models**

- BLSTM + CNN for event trigger identification
- CNN to capture local context

![Diagram of BLSTM + CNN model](image-url)

[Zeng et al., 2016]
One of most common/important structures in NLP

Both the tree/graph structures and their tags are latent

Building bricks

- Syntactic parsing
- Event extraction
- Semantic role labeling
- ...
The Task

Conventional Models
- Transition based
- Graph based

However,
- Feature engineering
- Label bias $\xrightarrow{\text{---->}}$ locally/globally normalized

Neural Network Practice
- Transition/Graph based
- NN models to extract various features
- choose from greedy search, beam search or approximate global normalization
Most DL based works in dependency parsing follows the transition based framework.

- lower complexity
- higher efficiency
- more choices of features

Follow normal transition styles

- but, most are based on greedy search

Various NN models used to produce dense features

- normal NNs
- LSTM
- BLSTM, both word level and character level

[Chen and Manning, 2014], [Weiss et al., 2015], [Dyer et al., 2015], [Ballesteros et al., 2015], [Zhou et al., 2015], [Kiperwasser and Goldberg, 2016], [Andor et al., 2016]
Features

- Key features, such as words, POS and dep labels, as well as their combinations can be transformed into dense format through a neural network.

**Softmax layer:**
\[ p = \text{softmax}(W_2h) \]

**Hidden layer:**
\[ h = (W^w_1x^w + W^t_1x^t + W^l_1x^l + b_1)^3 \]

**Input layer:** \([x^w, x^t, x^l]\)

**Configuration**

[Chen and Manning, 2014]
Or, features could be captured through LSTM units.

[21x453]

[Dyer et al., 2015] [Ballesteros et al., 2015] [Kiperwasser and Goldberg, 2016]
Greedy Search

- No global considerations about future decisions
  - look ahead with limited context
  - label bias
  - May be error propagations
- Training a global normalized model may not be trivial
  - Complexity
  - Space
- And, it is found that, in many parsers, beam search has minimal impact in the results. [Dyer et al., 2015]
- However, sometimes, better than beam-search based methods [Kiperwasser and Goldberg, 2016]
One solution is to use the output layers of the NN model to learn a structured perceptron with a beam search.

This gives 0.8% accuracy.

[Weiss et al., 2015]
Local normalized models may suffer from local optimal (greedy search), label bias, etc.

Perform global normalization exactly

- maximize the whole action sequence

It is not trivial!

- too many possible action sequences
- expensive (impossible) to enumerate and compute

Or, do it approximately?
**Contrastive Learning** [Hinton, 2002; LeCun and Huang, 2005; Liang and Jordan, 2008; Vickrey et al., 2010]

- reward observed data
- penalize noisy data

In the beam search case: [Zhou et al., 2014]

- Give gold sequence with higher probability
- Give incorrect sequences *in the beam* with lower probabilities
- Early update may be helpful
- gives more than **1.5%**
The importance of looking ahead

Approximate Global Normalization

[Andor et al., 2016]
Approximate Global Normalization

- **Global Training**
  - Backward propagation with beam search
  - with early update
  - slow

- **Works for multiple tasks:**
  - Dependency parsing
  - POS tagging
  - Sentence Compression

- **Insight:**
  - Global models are more expressive

[Andor et al., 2016]
On WSJ

- Chen and Manning, 2014
- Ballesteros et al., 2015
- [Greedy] Weiss et al., 2015
- Dyer et al., 2015
- Zhou et al., 2015
- Weiss et al., 2015
- Kipervasser & Goldberg, 2016
- Andor et al., 2016

UAS
LAS
The Task

\[ \text{He}_{A_0} \text{ had trouble raising } \text{funds}_{A_1}. \]  

Conventional Models

- Pipeline
- predicate identification, argument identification, argument classification

However,

- Feature engineering, local features, global features, ...
- Pipeline

Neural Practice

- NN models to extract various features, or as local classifiers
- In a sequence labeling style, an End-to-End system
Pipeline System

- **Step by step**: predicate identification, argument identification, argument classification

- **Each step**: local classifiers with feature extraction, using CNN, or LSTM

```
[Fortland and Martin, 2015] [Roth and Lapata, 2016]
```
Multi-Layered LSTM with CRF to **Sequence Tagging**.

Target: B-A1  I-A1  ...  O  B-V  ...

[Zhou and Xu, 2016]
Handle long distance dependencies

**CNN**
- good at capturing local features, or work on dependency path
- not so good at End-to-End systems, or extracting features from plain word sequence for SRL

**LSTM**
- good at capturing both local features, and global information, either for local decisions, or sentence level re-ranking
- more powerful in capturing features of various levels
- deep LSTM can also be used to build End-to-End SRL systems
Outline

Sequence Tagging

Tree/Graph Structure

Take-home Messages
Standard settings for Sequence Tagging

- BLSTM + CRF
- character modeling or CNN for fine modeling
- fused with successful traditional features
- perhaps, beam search for various features
- But Use as many useful resources as you can!

More complex structures

- BLSTM + Transition base systems
- choose from greedy search / beam search
- But
  Choose globally normalized models if you can!
Thanks
Reference


Jason P.C. Chiu and Eric Nichols, Named Entity Recognition with Bidirectional LSTM-CNNs,


Reference


Wenzhe Pei, Tao Ge, Baobao Chang, An Effective Neural Network Model for Graph-based Dependency Parsing, Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing, pages 313–322, Beijing, China, July 26-31, 2015.


