Enhanced Universal Dependency Parsing with Automated Concatenation of Embeddings

Xinyu Wang, Zixia Jia, Yong Jiang, Kewei Tu

School of Information Science and Technology, ShanghaiTech University
DAMO Academy, Alibaba Group
Our Parser

• A first-order graph-based dependency parser
• Equip the parser with Automated Concatenation of Embeddings (ACE) [1]
• Second-Place in the shared task

[1]: Wang, Xinyu and Jiang, Yong and Bach, Nguyen and Wang, Tao and Huang, Zhongqiang and Huang, Fei and Tu, Kewei. 2021. Automated Concatenation of Embeddings for Structured Prediction. In ACL-IJCNLP 2021
Preprocessing: Empty Nodes
Preprocessing: Repeated Edges
Preprocessing

• Tokenization: transkit (Nguyen et al., 2021)
• Multiple Treebanks: concatenate the datasets
• Splitting the development sets into halves as validation and test sets

Automated Concatenation of Embeddings (ACE)

- A controller samples a concatenation of embeddings according to its belief model.
- The concatenated word represents are fed as input of a task model and return the model accuracy after training.
- Use the accuracy as a reward signal and update the controller’s belief model.
- Optimization: policy gradient algorithm in reinforcement learning.
Task Model

- Graph-structured outputs
  - BiLSTM-Biaffine: $P_{\text{graph}}(y|x) = \text{BiLSTM-Biaffine}(V, y)$
- Word representation: $V = [v_1; \ldots; v_n]$
  - Embedding concatenation $v_i^l = \text{embed}_i^l(x)$; $v_i = [v_i^1; v_i^2; \ldots; v_i^L]$
Search Space Design

• Decide which embedding candidates are concatenated as word representation $v_i = \{v_i^1, \ldots, v_i^l, \ldots, v_i^L\}$
  • The resulting search space contains $2^L$ possible combinations
• We use a binary vector to mask out embeddings which are not selected
  $a = [a_1, \ldots, a_l, \ldots, a_L]$  $v_i = [v_i^1 a_1; \ldots; v_i^l a_l; \ldots; v_i^L a_L]$
Searching in the Space

• The parameter for the controller: \( \theta = [\theta_1; \theta_2; \ldots; \theta_L] \)

• The probability distribution of selecting a certain concatenation \( \mathbf{a} \):
  \[
P^{\text{ctrl}}(\mathbf{a}; \theta) = \prod_{l=1}^{L} P^{\text{ctrl}}(a_l; \theta_l)
  \]

• Each element \( a_l \) of \( \mathbf{a} \) is sampled independently from a Bernoulli distribution
Optimization

• Policy gradient with accuracy $R$: $J(\theta) = \mathbb{E}_{P_{\text{ctrl}}(a; \theta)}[R]$
• Approximate the gradient $J(\theta)$ by sampling only one selection:

$$\nabla_{\theta} J(\theta) \approx \sum_{l=1}^{L} \nabla_{\theta} \log P^{\text{ctrl}}_l(a_l; \theta_l)(R - b)$$
Optimization: Reward Function

- Reward function on how each embedding candidate contributes to accuracy change

\[ r^t = \sum_{i=1}^{t-1} (R_t - R_i) \gamma^{Hamm(a^t, a^i) - 1} |a^t - a^i| \]

A reward for each embedding
Accumulated accuracy change

When many embeddings are switched on/off, we are unsure which should get the credit, so we discount it

Only those responsible for the accuracy change get the credit
Training

1. Initialization: A dictionary $\mathbb{D}$ to store the searched concatenations and scores. Set time step $t = 0$.
2. Sample a concatenation $\mathbf{a}^t$ based on the probability distribution
3. Train the task model with $\mathbf{a}^t$ and evaluate the model on the development set to get the accuracy $R_t$.
4. Given the concatenation $\mathbf{a}^t$, accuracy $R_t$ and $\mathbb{D}$, compute the gradient of $J(\theta)$ and update the parameters of controller.
5. Add $\mathbf{a}^t$ and $R_t$ into $\mathbb{D}$, set $t = t + 1$.
6. Repeat 2~5 until $t$ is larger than a maximum iteration $T$. 
Post-processing

• MST to keep the connection of parser
• Back-conversion
Embeddings (for English)

- Flair: monolingual + multilingual
- BERT: monolingual + multilingual
- Roberta: monolingual
- XLM-Roberta: multilingual
- XLNet: monolingual
- GLoVe: English
- fastText: monolingual
- Character embeddings: train over the task

- The size of search space (for English): $2^{12} - 1 = 4095$
Embedding Fine-tuning

• Fine-tuning transformer-based embeddings is a usual approach
• It is difficult to fine-tune specific group of embeddings when multiple embeddings are concatenated
• Impractical due to complicated hyper-parameter settings and massive GPU memory consumption
• Our solution: First fine-tune each single embedding on the task, then concatenate fine-tuned embeddings together with other embeddings
## Results

<table>
<thead>
<tr>
<th>Team Name</th>
<th>TGIF</th>
<th>Ours</th>
<th>ROBERTNLP</th>
<th>COMBO</th>
<th>UNIPI</th>
<th>DCU EPFL</th>
<th>GREW</th>
<th>FASTPARSE</th>
<th>NUIG</th>
</tr>
</thead>
<tbody>
<tr>
<td>ar</td>
<td>81.23</td>
<td>82.26</td>
<td>81.58</td>
<td>76.39</td>
<td>77.17</td>
<td>71.01</td>
<td>71.13</td>
<td>53.74</td>
<td>0.00</td>
</tr>
<tr>
<td>bg</td>
<td>93.63</td>
<td>92.52</td>
<td>93.16</td>
<td>86.67</td>
<td>90.84</td>
<td>92.44</td>
<td>88.83</td>
<td>78.73</td>
<td>78.45</td>
</tr>
<tr>
<td>cs</td>
<td>92.24</td>
<td>91.78</td>
<td>90.21</td>
<td>89.08</td>
<td>88.73</td>
<td>89.93</td>
<td>87.66</td>
<td>72.85</td>
<td>0.00</td>
</tr>
<tr>
<td>nl</td>
<td>91.78</td>
<td>88.64</td>
<td>88.37</td>
<td>87.07</td>
<td>84.14</td>
<td>81.89</td>
<td>84.09</td>
<td>68.89</td>
<td>0.00</td>
</tr>
<tr>
<td>en</td>
<td>88.19</td>
<td>87.27</td>
<td>87.88</td>
<td>84.09</td>
<td>87.11</td>
<td>85.70</td>
<td>85.49</td>
<td>73.00</td>
<td>65.40</td>
</tr>
<tr>
<td>et</td>
<td>88.38</td>
<td>86.66</td>
<td>86.55</td>
<td>84.02</td>
<td>81.27</td>
<td>84.35</td>
<td>78.19</td>
<td>60.05</td>
<td>54.03</td>
</tr>
<tr>
<td>fi</td>
<td>91.75</td>
<td>90.81</td>
<td>91.01</td>
<td>87.28</td>
<td>89.62</td>
<td>89.02</td>
<td>85.20</td>
<td>57.71</td>
<td>0.00</td>
</tr>
<tr>
<td>fr</td>
<td>91.63</td>
<td>88.40</td>
<td>88.51</td>
<td>87.32</td>
<td>87.43</td>
<td>86.68</td>
<td>83.33</td>
<td>73.18</td>
<td>0.00</td>
</tr>
<tr>
<td>it</td>
<td>93.31</td>
<td>92.88</td>
<td>93.28</td>
<td>90.40</td>
<td>91.81</td>
<td>92.41</td>
<td>90.98</td>
<td>78.32</td>
<td>0.00</td>
</tr>
<tr>
<td>lv</td>
<td>90.23</td>
<td>89.17</td>
<td>88.82</td>
<td>84.57</td>
<td>83.01</td>
<td>86.96</td>
<td>77.45</td>
<td>66.43</td>
<td>56.67</td>
</tr>
<tr>
<td>lt</td>
<td>86.06</td>
<td>80.87</td>
<td>80.76</td>
<td>79.75</td>
<td>71.31</td>
<td>78.04</td>
<td>74.62</td>
<td>48.27</td>
<td>59.13</td>
</tr>
<tr>
<td>pl</td>
<td>91.46</td>
<td>90.66</td>
<td>89.78</td>
<td>87.65</td>
<td>88.31</td>
<td>89.17</td>
<td>78.20</td>
<td>71.52</td>
<td>0.00</td>
</tr>
<tr>
<td>ru</td>
<td>94.01</td>
<td>93.59</td>
<td>92.64</td>
<td>90.73</td>
<td>90.90</td>
<td>92.83</td>
<td>90.56</td>
<td>78.56</td>
<td>66.33</td>
</tr>
<tr>
<td>sl</td>
<td>94.96</td>
<td>90.25</td>
<td>89.66</td>
<td>87.04</td>
<td>86.05</td>
<td>89.59</td>
<td>86.92</td>
<td>64.28</td>
<td>67.45</td>
</tr>
<tr>
<td>sv</td>
<td>89.90</td>
<td>86.62</td>
<td>88.03</td>
<td>83.20</td>
<td>84.91</td>
<td>85.20</td>
<td>81.54</td>
<td>67.26</td>
<td>63.12</td>
</tr>
<tr>
<td>ta</td>
<td>65.58</td>
<td>58.94</td>
<td>59.33</td>
<td>52.27</td>
<td>51.73</td>
<td>39.32</td>
<td>58.69</td>
<td>42.53</td>
<td>0.00</td>
</tr>
<tr>
<td>uk</td>
<td>92.78</td>
<td>88.94</td>
<td>88.86</td>
<td>86.92</td>
<td>87.51</td>
<td>86.09</td>
<td>83.90</td>
<td>63.42</td>
<td>0.00</td>
</tr>
<tr>
<td>Avg.</td>
<td>89.24</td>
<td>87.07</td>
<td>86.97</td>
<td>83.79</td>
<td>83.64</td>
<td>83.57</td>
<td>81.58</td>
<td>65.81</td>
<td>30.03</td>
</tr>
</tbody>
</table>
Conclusion

• A parser with automated embeddings concatenation and biaffine encoder
• Our system ranks 2nd over 9 teams according to the official ELAS
Thanks!