genCNN: A Convolutional Architecture for word sequence prediction

刘荣
2015-03-25
A Convolutional Architecture for word sequence prediction

- Architecture

Figure 1: The overall diagram of a genCNN. Here “/” stands for a zero padding. In this example, each CNN component covers 6 words, while in practice the coverage is 30-40 words.

As shown in Figure 1, genCNN is overall recursive, consisting of CNN-based processing units of two types:

- \( \alpha \text{CNN} \) as the “front-end”, dealing with the history that is closest to the prediction;
- \( \beta \text{CNNs} \) (which can repeat), in charge of more “ancient” history.

Together, genCNN takes history \( e_{1:t} \) of arbitrary length to predict the next word \( e_{t+1} \) with probability

\[
p(e_{t+1} | e_{1:t}; \Theta),
\]

based on a representation \( \phi(e_{1:t}; \Theta) \) produced by the CNN, and a \( |\mathcal{V}| \)-class soft-max:

\[
p(e_{t+1}|e_{1:t}; \Theta) \propto e^{\mu^T_{e_{t+1}} \phi(e_{1:t}) + b_{e_{t+1}}}.
\]
Figure 4: genCNN with recursive structure.
A Convolutional Architecture for word sequence prediction

- Architecture
For sentence input $x = \{x_1, \ldots, x_T\}$, the feature-map of type-$f$ on Layer-$\ell$ is

if $f \in \text{TIME-FLOW}$:

$$z_i^{(\ell,f)}(x) = \sigma(w_{TF}^{(\ell,f)} z_i^{(\ell-1)} + b_{TF}^{(\ell,f)})$$

(3)

if $f \in \text{TIME-ARROW}$:

$$z_i^{(\ell,f)}(x) = \sigma(w_{TA}^{(\ell,f,i)} \hat{z}_i^{(\ell-1)} + b_{TA}^{(\ell,f,i)})$$

(4)

where

- $z_i^{(\ell,f)}(x)$ gives the output of feature-map of type-$f$ for location $i$ in Layer-$\ell$;
- $\sigma(\cdot)$ is the activation function, e.g., Sigmoid or Relu (Dahl et al., 2013)
- $(w_{TF}^{(\ell,f)}, b_{TF}^{(\ell,f)})$ denotes the location-independent parameters for $f \in \text{TIME-FLOW}$ on Layer-$\ell$,
  while $(w_{TA}^{(\ell,f,i)}, b_{TA}^{(\ell,f,i)})$ stands for that for $f \in \text{TIME-ARROW}$ and location $i$ on Layer-$\ell$;
- $\hat{z}_i^{(\ell-1)}$ denotes the segment of Layer-$\ell-1$ for the convolution at location $i$, while

$$\hat{z}_i^{(0)} \overset{\text{def}}{=} [x_i^T, x_{i+1}^T, \cdots, x_{i+k_1-1}^T]^T$$

concatenates the vectors for $k_1$ words from sentence input $x$.

Figure 3: Illustration for gating network.
Figure 3: Illustration for gating network.

Suppose we have convolution feature-maps on Layer-$\ell$ and gating (with window size = 2) on Layer-$\ell+1$. For the $j^{th}$ gating window $(2j-1, 2j)$, we merge $\hat{z}_{2j-1}^{(\ell-1)}$ and $\hat{z}_{2j}^{(\ell-1)}$ as the input (denoted as $\tilde{z}_j^{(\ell)}$) for gating network, as illustrated in Figure 3. We use a separate gate for each feature-map, but follow a different parametrization strategy for TIME-FLOW and TIME-ARROW. With window size = 2, the gating is binary, we use a logistic regressor to determine the weights of two candidates. For $f \in \text{TIME-ARROW}$, with location-dependent $w_{\text{gate}}^{(\ell,f,j)}$, the normalized weight for left side is

$$g_j^{(\ell+1,f)} = \frac{1}{1 + e^{-w_{\text{gate}}^{(\ell,f,j)} \tilde{z}_j^{(\ell)}}},$$

while for $f \in \text{TIME-FLOW}$, the parameters for the corresponding gating network, denoted as $w^{(\ell,f)}_{\text{gate}}$, are shared. The gated feature map is then a weighted sum to feature-maps from the two windows:

$$z_j^{(\ell+1,f)} = g_j^{(\ell+1,f)} z_{2j-1}^{(\ell,f)} + (1 - g_j^{(\ell+1,f)}) z_{2j}^{(\ell,f)}. \tag{5}$$

We find that this gating strategy works significantly better than direct pooling over feature-maps, and also slightly better than a hard gate version of Equation (5).
The parameters of a genCNN $\tilde{\Theta}$ consists of the parameters for CNN $\Theta_{nn}$, word-embedding $\Theta_{embed}$, and the parameters for soft-max $\Theta_{softmax}$. All the parameters are jointly learned by maximizing the likelihood of observed sentences. Formally the log-likelihood of sentence $S_n \left( \equiv [e_1^{(n)}, e_2^{(n)}, \ldots, e_{T_n}^{(n)}] \right)$ is

$$\log p(S_n; \tilde{\Theta}) = \sum_{t=1}^{T_n} \log p(e_t^{(n)}|e_{1:t-1}^{(n)}; \tilde{\Theta}),$$

which can be trivially split into $T_n$ training instances during the optimization, in contrast to the training of RNN that requires unfolding through time due to the temporal-dependency of the hidden states.

**Soft-max:** Calculating a full soft-max is expensive since it has to enumerate all the words in vocabulary (in our case 40K words) in the denominator. Here we take a simple hierarchical approximation of it, following (Bahdanau et al., 2014). Basically we group the words into 200 clusters (indexed by $c_m$), and factorize (in an approximate sense) the conditional probability of a word $p(e_t|e_{1:t-1}; \tilde{\Theta})$ into the probability of its cluster and the probability of $e_t$ given its cluster

$$p(c_m|e_{1:t-1}; \tilde{\Theta}) p(e_t|c_m; \Theta_{softmax}).$$

We found that this simple heuristic can speed-up the optimization by 5 times with only slight loss of accuracy.
Optimization: We use stochastic gradient descent with mini-batch (size 500) for optimization, aided further by AdaGrad (Duchi et al., 2011). For initialization, we use Word2Vec (Mikolov et al., 2013) for the starting state of the word-embeddings (trained on the same dataset as the main task), and set all the other parameters by randomly sampling from uniform distribution in $[-0.1, 0.1]$. The optimization is done mainly on a Tesla K40 GPU, which takes about 2 days for the training on a dataset containing 1M sentences.
• Experiment: sentence generation

Table 1: Examples of sentences generated by genCNN. In the upper block (row 1-4) the underline words are given by the human; In the middle block (row 5-8), all the sentences are generated without any hint. The bottom block (row 9-12) shows the sentences with dependency tag generated by genCNN trained with parsed examples.

In this experiment, we randomly generate sentences by recurrently sampling

\[ e_{t+1}^* \sim p(e_{t+1}^*|e_{1:t}; \Theta) , \]
• **Experiment: Language Modeling**

**Competitor Models** we compare genCNN to the following competitor models

- 5-gram: We use SRI Language Modeling Toolkit (Stolcke and others, 2002) to train a 5-gram language model with modified Kneser-Ney smoothing;

- FFN-LM: The neural language model based on feedfoward network (Vaswani et al., 2013). We vary the input window-size from 5 to 20, while the performance stops improving after window size 20;

- RNN: we use the implementation\(^1\) of RNN-based language model with hidden size 600 for optimal performance of it;

- LSTM: we use the code in Groundhog\(^2\), but vary the hyper-parameters, including the depth and word-embedding dimension, for best performance. LSTM (Hochreiter and Schmidhuber, 1997) is widely considered to be the state-of-the-art for sequence modeling.

<table>
<thead>
<tr>
<th>Model</th>
<th>Perplexity</th>
<th>Dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-gram, KN5</td>
<td>141.2</td>
<td>–</td>
</tr>
<tr>
<td>FFNN-LM</td>
<td>140.2</td>
<td>–</td>
</tr>
<tr>
<td>RNN</td>
<td>124.7</td>
<td>123.2</td>
</tr>
<tr>
<td>LSTM</td>
<td>126</td>
<td>117</td>
</tr>
<tr>
<td>genCNN</td>
<td><strong>116.4</strong></td>
<td><strong>106.3</strong></td>
</tr>
</tbody>
</table>

Table 2: **PENN TREEBANK** results, where the 3rd column are the perplexity in dynamic evaluation, while the numbers for RNN and LSTM are taken as reported in the paper cited above. The numbers in boldface indicate that the result is significantly better than all competitors in the same setting.
- **Experiment: Language Modeling**

<table>
<thead>
<tr>
<th>Model</th>
<th>Perplexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-gram, KN5</td>
<td>278.6</td>
</tr>
<tr>
<td>FFN-LM(5-gram)</td>
<td>248.3</td>
</tr>
<tr>
<td>FFN-LM(20-gram)</td>
<td>228.2</td>
</tr>
<tr>
<td>RNN</td>
<td>223.4</td>
</tr>
<tr>
<td>LSTM</td>
<td>206.9</td>
</tr>
<tr>
<td>genCNN</td>
<td><strong>181.2</strong></td>
</tr>
<tr>
<td>TIME-ARROW only</td>
<td>192</td>
</tr>
<tr>
<td>TIME-FLOW only</td>
<td>203</td>
</tr>
<tr>
<td>αCNN only</td>
<td>184.4</td>
</tr>
</tbody>
</table>

**Table 3:** FBIS results. The upper block (row 1-6) compares genCNN and the competitor models, and the bottom block (row 7-9) compares different variants of genCNN.
- **Experiment: Language Modeling**

<table>
<thead>
<tr>
<th>Models</th>
<th>MT06</th>
<th>MT08</th>
<th>Ave.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>38.63</td>
<td>31.11</td>
<td>34.87</td>
</tr>
<tr>
<td>RNN rerank</td>
<td>39.03</td>
<td>31.50</td>
<td>35.26</td>
</tr>
<tr>
<td>LSTM rerank</td>
<td>39.20</td>
<td>31.90</td>
<td>35.55</td>
</tr>
<tr>
<td>FFN-LM rerank</td>
<td>38.93</td>
<td>31.41</td>
<td>35.14</td>
</tr>
<tr>
<td>genCNN rerank</td>
<td><strong>39.90</strong></td>
<td><strong>32.50</strong></td>
<td><strong>36.20</strong></td>
</tr>
<tr>
<td>Base+FFN-LM</td>
<td>39.08</td>
<td>31.60</td>
<td>35.34</td>
</tr>
<tr>
<td>genCNN rerank</td>
<td><strong>40.4</strong></td>
<td><strong>32.85</strong></td>
<td><strong>36.63</strong></td>
</tr>
</tbody>
</table>

Table 4: The results for re-ranking the 1000-best of Moses. Note that the two bottom rows are on a baseline with enhanced LM.