genCNN:A Convolutional Architecture for word sequence prediction

刘荣

2015-03-25

A Convolutional Architecture for word sequence prediction

- Architecture

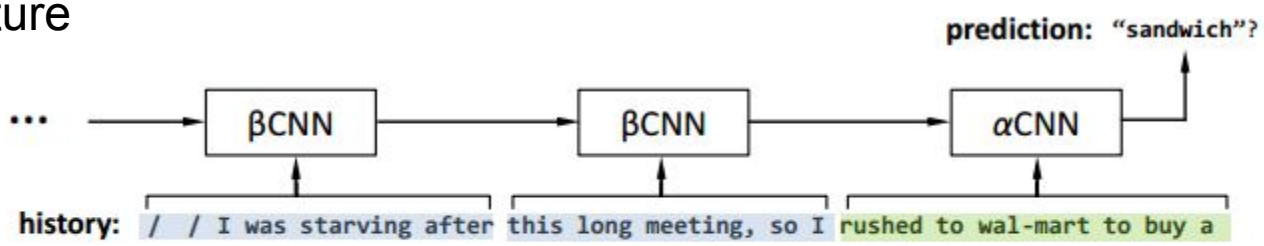


Figure 1: The overall diagram of a *gen*CNN. Here "/" stands for a zero padding. In this example, each CNN component covers 6 words, while in practice the coverage is 30-40 words.

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

- αCNN as the "front-end", dealing with the history that is closest to the prediction;
- βCNNs (which can repeat), in charge of more "ancient" history.

Together, *gen*CNN takes history $\mathbf{e}_{1:t}$ of arbitrary length to predict the next word $\mathbf{e}_{t+1}$ with probability

$$p(\mathbf{e}_{t+1} \,|\mathbf{e}_{1:t}; \bar{\Theta}), \tag{1}$$

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

$$p(\mathbf{e}_{t+1}|\mathbf{e}_{1:t}; \bar{\Theta}) \propto e^{\mu_{\mathbf{e}_{t+1}}^{\top} \phi(\mathbf{e}_{1:t}) + b_{\mathbf{e}_{t+1}}}. \tag{2}$$

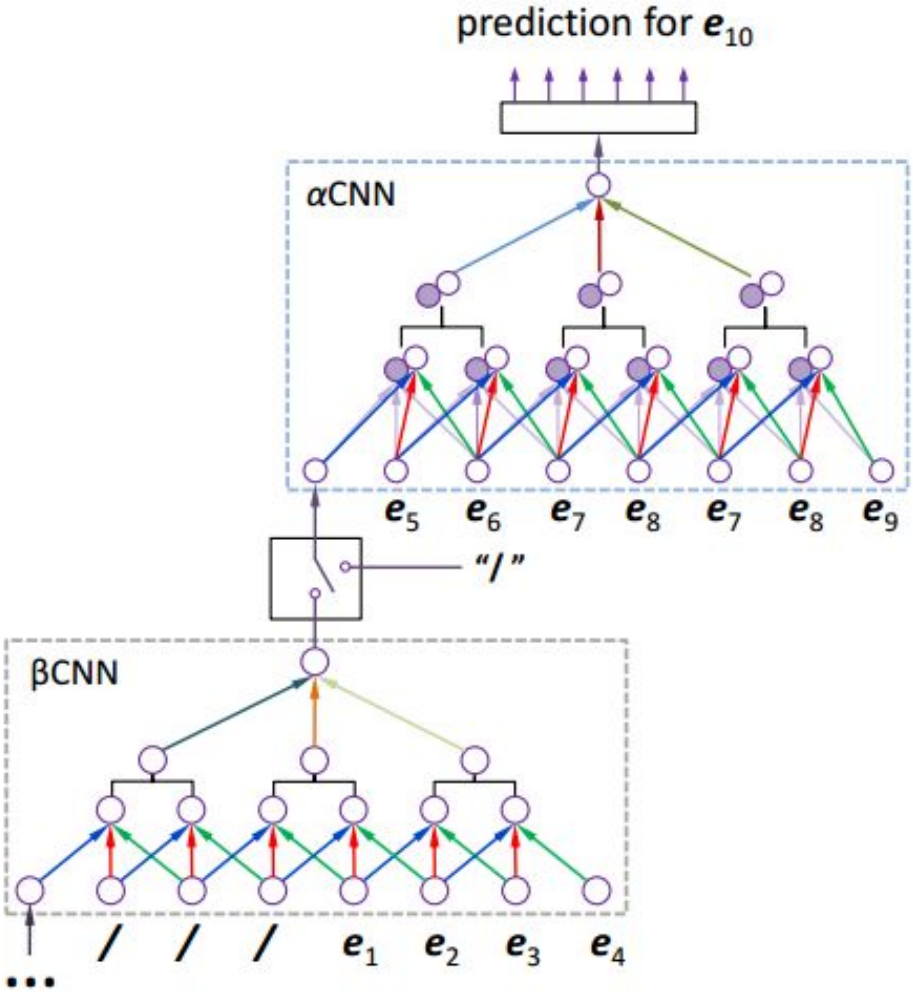
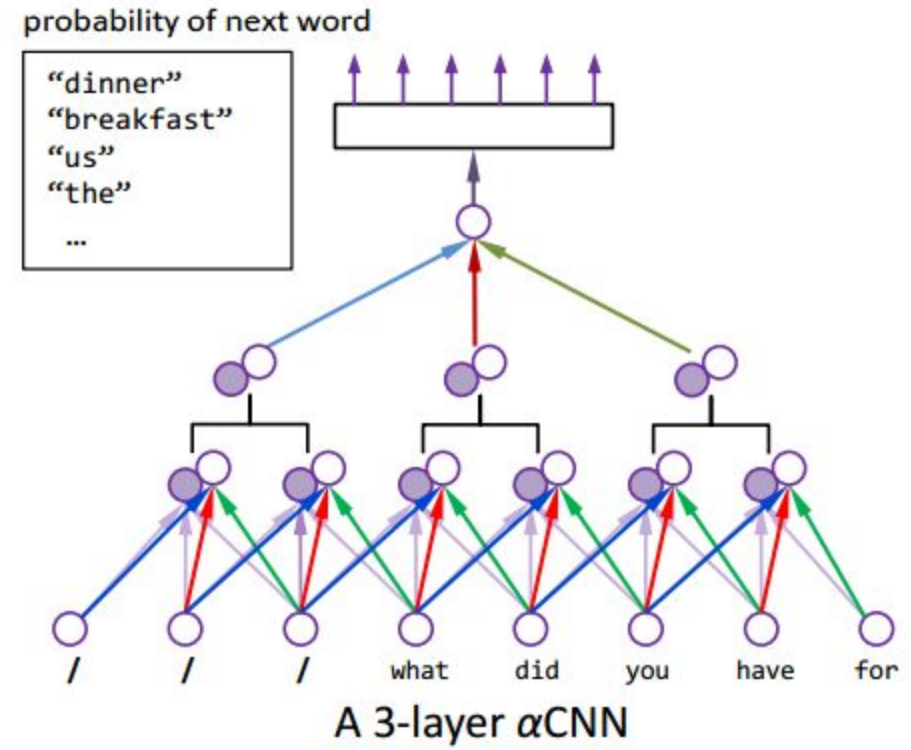


Figure 4: *gen*CNN with recursive structure.

A Convolutional Architecture for word sequence prediction

- Architecture



For sentence input $\mathbf{x}=\{\mathbf{x}_1, \cdots, \mathbf{x}_T\}$, the feature-map of type-f on Layer-ℓ is

if $f \in$ TIME-FLOW:

$$z_i^{(\ell,f)}(\mathbf{x}) = \sigma(\mathbf{w}_{\text{TF}}^{(\ell,f)} \hat{\mathbf{z}}_i^{(\ell-1)} + b_{\text{TF}}^{(\ell,f)}), \tag{3}$$

if $f \in$ TIME-ARROW:

$$z_i^{(\ell,f)}(\mathbf{x}) = \sigma(\mathbf{w}_{\text{TA}}^{(\ell,f,i)} \hat{\mathbf{z}}_i^{(\ell-1)} + b_{\text{TA}}^{(\ell,f,i)}), \tag{4}$$

where

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

$$\hat{\mathbf{z}}_i^{(0)} \stackrel{\text{def}}{=} [\mathbf{x}_i^\top, \ \mathbf{x}_{i+1}^\top, \ \cdots, \ \mathbf{x}_{i+k_1-1}^\top]^\top$$

concatenates the vectors for k_1 words from sentence input $\mathbf{x}$.

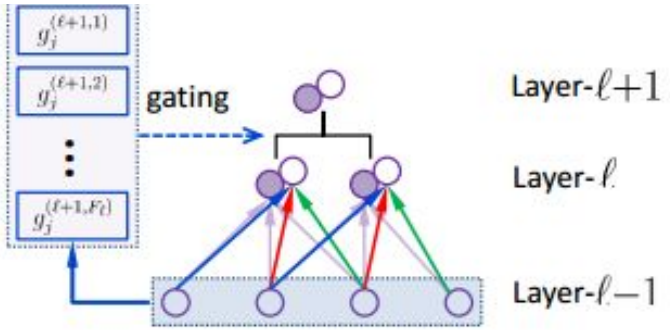


Figure 3: Illustration for gating network.

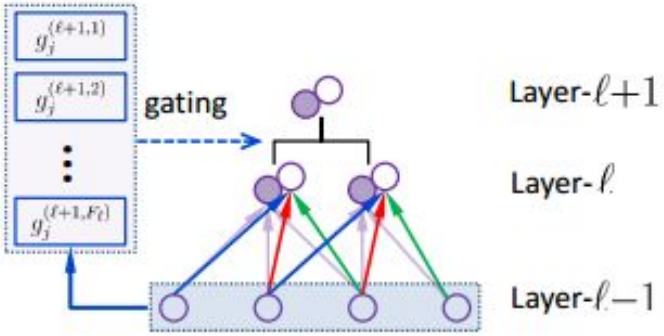


Figure 3: Illustration for gating network.

Suppose we have convolution feature-maps on Layer-ℓ and gating (with window size = 2) on Layer-ℓ+1. For the j^{th} gating window $(2j-1, 2j)$, we merge $\hat{\mathbf{z}}_{2j-1}^{(\ell-1)}$ and $\hat{\mathbf{z}}_{2j}^{(\ell-1)}$ as the input (denoted as $\bar{\mathbf{z}}_j^{(\ell)}$) 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 $f \in$TIME-ARROW, with location-dependent $\mathbf{w}_{\text{gate}}^{(\ell,f,j)}$, the normalized weight for *left* side is

$$g_j^{(\ell+1,f)} = 1/(1 + e^{-\mathbf{w}_{\text{gate}}^{(\ell,f,j)} \bar{\mathbf{z}}_j^{(\ell)}}),$$

while for For $f \in$TIME-FLOW, the parameters for the corresponding gating network, denoted as $\mathbf{w}_{\text{gate}}^{(\ell,f)}$, 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).

- train

The parameters of a genCNN $\bar{\Theta}$ consists of the parameters for CNN Θ_{nn}, word-embedding Θ_{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 $\mathcal{S}_n$ ($\stackrel{\text{def}}{=} [\mathbf{e}_1^{(n)}, \mathbf{e}_2^{(n)}, \cdots, \mathbf{e}_{T_n}^{(n)}]$) is

$$\log p(\mathcal{S}_n; \bar{\Theta}) = \sum_{t=1}^{T_n} \log p(\mathbf{e}_t^{(n)} | \mathbf{e}_{1:t-1}^{(n)}; \bar{\Theta}),$$

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 $p(\mathbf{e}_t | \mathbf{e}_{1:t-1}; \bar{\Theta})$ into the probability of its cluster and the probability of $\mathbf{e}_t$ given its cluster

$$p(c_m | \mathbf{e}_{1:t-1}; \bar{\Theta})\, p(\mathbf{e}_t | c_m; \Theta_{softmax}).$$

We found that this simple heuristic can speed-up the optimization by 5 times with only slight loss of accuracy.

- train

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

`` we are in the building of china 's social development and the businessmen audience , '' he said .
clinton was born in DDDD , and was educated at the university of edinburgh.
bush 's first album , `` the man '' , was released on DD november DDDD .
it is one of the first section of the act in which one is covered in real place that recorded in norway .
this objective is brought to us the welfare of our country
russian president putin delivered a speech to the sponsored by the 15th asia pacific economic cooperation (apec) meeting in an historical arena on oct .
light and snow came in kuwait and became operational , but was rarely placed in houston .
johnson became a drama company in the DDDDs , a television broadcasting company owned by the broadcasting program .
((the two ⋆ sides) ⋆ should (⋆ assume (a strong ⋆ target))) .)
(it ⋆ is time (⋆ in (every ⋆ country) ⋆ signed (the ⋆ speech)) .)
((initial ⋆ investigations) ⋆ showed (⋆ that (spot ⋆ could (⋆ be (further ⋆ improved significantly)) .)
((a ⋆ book (to ⋆ northern (the 21 st ⋆ century))) .)

Table 1: Examples of sentences generated by *gen*CNN. 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 *gen*CNN trained with parsed examples.

In this experiment, we randomly generate sentences by recurrently sampling

$$\mathbf{e}^{\star}_{t+1} \sim p(\mathbf{e}_{t+1}|\mathbf{e}_{1:t}; \bar{\Theta}),$$

- Experiment: Language Modeling

Competitor Models we compare *gen*CNN 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.

Model	Perplexity	Dynamic
5-gram, KN5	141.2	–
FFNN-LM	140.2	–
RNN	124.7	123.2
LSTM	126	117
*gen*CNN	**116.4**	**106.3**

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

Model	Perplexity
5-gram, KN5	278.6
FFN-LM(5-gram)	248.3
FFN-LM(20-gram)	228.2
RNN	223.4
LSTM	206.9
*gen*CNN	**181.2**
TIME-ARROW only	192
TIME-FLOW only	203
αCNN only	184.4

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

- Experiment: Language Modeling

Models	MT06	MT08	Ave.
Baseline	38.63	31.11	34.87
RNN rerank	39.03	31.50	35.26
LSTM rerank	39.20	31.90	35.55
FFN-LM rerank	38.93	31.41	35.14
*gen*CNN rerank	**39.90**	**32.50**	**36.20**
Base+FFN-LM	39.08	31.60	35.34
*gen*CNN rerank	**40.4**	**32.85**	**36.63**

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.