The Thirty-Second AAAI Conference
on Artificial Intelligence (AAAI-18)

Tree-Structured Neural Machine for Linguistics-Aware Sentence Generation

Ganbin Zhou,[1,2] **Ping Luo,**[1,2] **Rongyu Cao,**[1,2] **Yijun Xiao,**[3] **Fen Lin,**[4] **Bo Chen,**[4] **Qing He**[1,2]
[1]Key Lab of Intelligent Information Processing of Chinese Academy of Sciences (CAS),
Institute of Computing Technology, CAS, Beijing 100190, China.{zhouganbin, luop, heqing}@ict.ac.cn
[2]University of Chinese Academy of Sciences, Beijing 100049, China.
[3]Department of Computer Science, University of California Santa Barbara, Santa Barbara, CA 93106, USA.
[4]WeChat Search Application Department, Tencent, China.

Abstract

Different from other sequential data, sentences in natural language are structured by linguistic grammars. Previous generative conversational models with chain-structured decoder ignore this structure in human language and might generate plausible responses with less satisfactory relevance and fluency. In this study, we aim to incorporate the results from linguistic analysis into the process of sentence generation for high-quality conversation generation. Specifically, we use a dependency parser to transform each response sentence into a dependency tree and construct a training corpus of *sentence-tree* pairs. A tree-structured decoder is developed to learn the mapping from a sentence to its tree, where different types of hidden states are used to depict the local dependencies from an internal tree node to its children. For training acceleration, we propose a tree canonicalization method, which transforms trees into equivalent ternary trees. Then, with a proposed tree-structured search method, the model is able to generate the most probable responses in the form of dependency trees, which are finally flattened into sequences as the system output. Experimental results demonstrate that the proposed X2TREE framework outperforms baseline methods over 11.15% increase of acceptance ratio.

Introduction

Many natural language processing tasks can be formulated as sequence to sequence problems. Given a sequence of tokens, this task is to generate another sequence of tokens of equal or non-equal length. For example, machine translation models try to find a sequence of words in the target language, expressing the identical meaning to a source sentence; conversational models respond to a post utterance with a semantically coherent and grammatically correct sentence. Neural models were applied to these tasks and achieved state-of-the-art performances in recent years (Cho et al. 2014; Shang, Lu, and Li 2015; Vinyals and Le 2015; Sordoni et al. 2015; Serban et al. 2015).

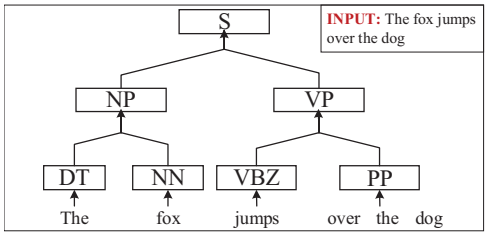
These neural models in essence use a chain-structured decoder to sequentially generate tokens given a context vector encoded from an input sequence. We notice that this decoding process is mostly linear, meaning that tokens are obtained in the order of their appearances. It basically considers the dependency between any word and all its preceding ones. RNN models, such as LSTM (Hochreiter and Schmidhuber 1997) and GRU (Cho et al. 2014), are developed in demand of capturing both the short and long-distance dependency over the chain structure.

Our work improves upon these studies by incorporating the results from linguistic analysis into the decoder. Specifically, we leverage a dependency parser to transform each response sentence into a dependency tree, containing more local dependency information. The proposed model learns to map a sentence into a canonicalized tree, which is then flattened as the final output. Consider the intermediate task for automatic conversation generation. Instead of generating the response to a given input post directly, we aim to generate the *dependency parse tree* of the corresponding response in top-down fashion. Additionally, a tree canonicalization method is proposed, aiming at transforming trees with different numbers of children nodes into their equivalent form, namely full ternary trees, in order to accelerate training and simplify model implementation on GPU. Then, a postprocessing step converts the dependency tree into a sequence as the final response. We also theoretically prove that ternary tree is the "best" choice for model complexity.

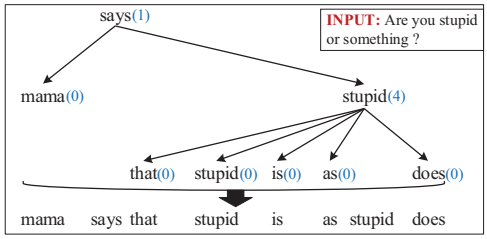
Some models also process trees in a *bottom-up* fashion. Socher et al. (2011) proposed a max-margin structure prediction architecture based on recursive neural networks, and demonstrated that it successfully parses sentences and understands scene images. Tai et al. (2015) and Zhu et al. (2015) extended the chain-structured LSTM to tree-structured LSTM, which is shown to be more effective in representing a tree structure as a latent vector. All these models process trees in a *bottom-up* fashion, where children nodes are recursively merged into parent nodes until the root is generated.

However, bottom-up models require all the leaf nodes in the predicted tree given in advance. For example, to generate the *constituency parse tree* for a sentence (shown in Fig. 1(a)), tokens appeared in the given sentence are used as leaf nodes in this tree. Similarly, to parse natural scene images (Socher et al. 2011), an image is first divided into segments, each of which corresponds to one leaf node in output tree. With these given leaves bottom-up process recursively processes the internal nodes until the root is built.

Here, we argue that the bottom-up generative models may not work well when the leaf nodes are not specified ahead of



(a) Constituency parser in bottom-up fashion.



(b) Dependency parser in top-down fashion.

Figure 1: Examples of two tree-structured prediction tasks in language understanding.

prediction. Consider the task in Fig. 1(b), which is an intermediate task for automatic conversation generation. Instead of generating the response to a given input post directly, we aim to generate the *dependency parse tree* of the corresponding response. Then, a postprocessing step converts the dependency tree into a sequence as the final response. [1] Compared to the SEQ2SEQ solution to conversation generation, we argue that this tree-structured modeling method is more effective due to a shorter average decoding length and the extra structure information provided from the parse tree. In this task, it is clearly seen that: since all the tokens in response are not explicitly given by the input post, it may not be appropriate to generate the dependency from bottom to top.

Previous works on tree-structured LSTM (Tai, Socher, and Manning 2015; Zhu, Sobhani, and Guo 2015; Zhang, Lu, and Lapata 2016) show that incorporating syntactic structures into the encoder or decoder results in sentence embedding with improved performances on tasks like sentiment analysis and semantic relatedness. In this paper, we propose to inject tree structures into the decoding process with the following motivations:

1) Dependency tree parsing extracts short-distance dependencies in the local area of a sentence. Utilizing these linguistic results reduces the difficulty in sequential learning, thus helps decoders to generate grammatically and semantically correct utterances. Let $\boldsymbol{y}$ be the response sentence to an input $\boldsymbol{x}$, and $\mathcal{T}_{\boldsymbol{y}}$ be the dependency tree for $\boldsymbol{y}$. Then, the average length between any node in $\mathcal{T}_{\boldsymbol{y}}$ to its root is $O(\sqrt{|\boldsymbol{y}|})$ (Flajolet and Odlyzko 1982), much smaller than the sentence length $|\boldsymbol{y}|$. Thus, this tree transformation may alleviate the long-distance gap in sequence generation. 2) Words in higher levels of the dependency tree usually are more influential for the sentence. By generating more "important" words in earlier stages of the decoding process, we essentially free the decoder from the burden to store important semantic information for many time steps. 3) We also believe that the process of tree-structured sentence generation is more consistent with how human construct sentences. Although people speak a sentence in a sequential order, they may keep some keywords, such as verbs and nouns, in mind before filling in more descriptive adjectives and adverbs to generate a full sentence.

In this paper, we develop a tree-structured decoder in the framework of "X to tree" (X2TREE) learning, where X represents any structure (e.g. chain, tree) encoding the post as a latent vector. Since all the tokens in the response are not explicitly given by the input post, it is appropriate to generate the dependency from top to bottom. To this end, we need to address the following challenges:

1) We need to carefully model the different dependencies between a tree node and its children. Children at different positions may have different meanings, and the generation of a child node depends on not only its parent and ancestors but also its siblings. Thus, we need to fully consider the memory inherited from both its ancestors and siblings (detailed in Section *Generative Model for K-ary Full Tree*).

2) A tree node could obtain any number of children. It is non-trivial to automatically determine the number of children. Furthermore, GPU-based parallel computing is difficult when the children number is different for each node. We therefore need a tree canonicalization process, which outputs an *equivalent* standard tree, where each internal node has a fixed number of child nodes (detailed in Section *Tree Canonicalization*).

3) In model inference, it is required to develop a algorithm searching for the most probable trees instead of sequences. Since the beam search utilized by previous studies only handles chain structures, a more general search algorithm for tree structures needs to be developed (detailed in Section *Tree Generation*).

With all these challenges addressed, our main contributions are twofold: 1) We propose a generative neural machine for tree structures, and apply it to conversational model. Specifically, we introduce a tree canonicalization method to standardize the generative process and a greedy search method for tree structure inference. 2) We empirically demonstrate that the proposed method successfully predict the dependency trees of conversational responses to an input post. Specifically, for the task of automatic conversation the proposed X2TREE framework achieves 11.15% increase of acceptance ratio.

It is also worth mentioning that we do not need a perfect dependency parser. In our task, the sequential sentence is the final output, while the dependency tree is only the immediate result. If the parsed tree contains errors in similar patterns, the model can learn these patterns. After we convert the generated tree into a sequence, the sequence may be still correct, which is also demonstrated by the experiments.

[1] The motivation of this solution is detailed in Section *Tree Generation*.

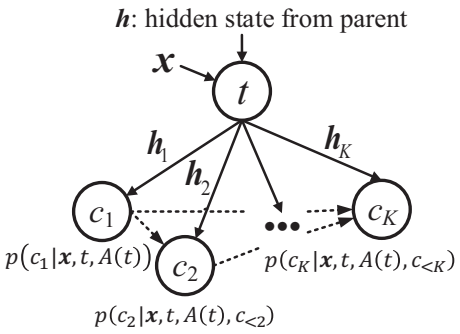


Figure 2: Parent-children dependency.

X2Tree Neural Network

In this section, we introduce the X2TREE learning framework. The training dataset is given as: $D = \{(\boldsymbol{x}, \mathcal{T}_{\boldsymbol{y}})\}$ where $\boldsymbol{y}$ is the response of the post $\boldsymbol{x}$ and $\mathcal{T}_{\boldsymbol{y}}$ is the corresponding tree of $\boldsymbol{y}$, e.g. dependency tree. Our task is to learn the mapping from $\boldsymbol{x}$ to a tree structure $\mathcal{T}_{\boldsymbol{y}}$. Specifically, it adopts the encoder-decoder framework. We assume $\boldsymbol{x}$ has already been encoded as a latent vector $\big($see e.g. (Sutskever, Vinyals, and Le 2014; Zhu, Sobhani, and Guo 2015)$\big)$, and mostly focus on the tree-structured decoder for the generation of $\mathcal{T}_{\boldsymbol{y}}$.

As aforementioned, the developed decoder adopts a top-down generative process. The atom step is generating the children for a given node. This atom step is performed on each node until it cannot generate any valid nodes. Thus, the key to the decoder is modeling the parent-children dependency. Note also that the model parameters for parent-children dependency are shared for all the atom steps in tree generation.

We first assume the tree is K-ary full tree where every internal node has exactly K children, and model this type of tree in Section *Generative Model for K-ary Full Tree*. Then, we propose a canonicalization method that transforms any tree into a K-ary full tree and discuss the K for different applications in Section *Tree Canonicalization*. Finally, we introduce an algorithm for tree inference in Section *Tree Generation*.

Generative Model for K-ary Full Tree

Here, we propose a generative model for K-ary full tree. For simplicity, $\boldsymbol{x}$ also represents the latent vector encoded from the input post. Within the probabilistic learning framework, our main task is to express the conditional probability $p(\mathcal{T}|\boldsymbol{x})$ for a pair $(\boldsymbol{x}, \mathcal{T}) \in D$. We can first reformulate $p(\mathcal{T}|\boldsymbol{x})$ as:

$$p(\mathcal{T}|\boldsymbol{x}) = p(t_{\mathrm{r}}|\boldsymbol{x}) \cdot p(\mathcal{T}_{\neg \mathrm{r}} \mid t_{\mathrm{r}}, \boldsymbol{x}) \tag{1}$$

where t_{r} and $\boldsymbol{\mathcal{T}}_{\neg \mathrm{r}}$ denotes the root and the set of non-root nodes respectively. The first term $p(t_{\mathrm{r}}|\boldsymbol{x})$ in Equ. (1) is modeled as follows $p(t_{\mathrm{r}}|\boldsymbol{x}) = \frac{\exp g_{\mathrm{r}}(t_{\mathrm{r}}, \boldsymbol{x})}{\sum_{v \in V} \exp g_{\mathrm{r}}(v, \boldsymbol{x})}$ where g_{r} is a nonlinear and potentially multi-layered function, and V is the vocabulary containing all possible values for the discrete random variables.

To model $p(\mathcal{T}_{\neg \mathrm{r}} \mid t_{\mathrm{r}}, \boldsymbol{x})$, we make the following conditional independence assumption:

Assumption 1. *The children of different nodes are conditionally independent given their ancestors.*

With assumption 1, $p(\mathcal{T}_{\neg \mathrm{r}}|t_{\mathrm{r}}, \boldsymbol{x})$ is decomposed as:

$$p(\mathcal{T}_{\neg \mathrm{r}}|t_{\mathrm{r}}, \boldsymbol{x}) = \prod_{t \in \mathcal{T}} p\big(C(t) \mid \boldsymbol{x}, t, A(t)\big) \tag{2}$$

where $C(t) = (c_1, c_2, \cdots, c_K)$ denotes the set of t's children, and $A(t)$ denotes all t's ancestors.

We then move to model the conditional probability $p\big(C(t)|\boldsymbol{x}, t, A(t)\big)$. Concretely, since the child nodes to a parent usually correlate with each other, it is inappropriate to assume conditional independence among them. Thus, the probability $p\big(C(t)|\boldsymbol{x}, t, A(t)\big)$ is then decomposed into the following ordered conditional probabilities:

$$p\big(C(t)|\boldsymbol{x}, t, A(t)\big) = \prod_{c_k \in C(t)} p\big(c_k|\boldsymbol{x}, t, A(t), c_{<k}\big) \tag{3}$$

Furthermore, we argue that children at different positions obtain different underlying meanings. Hence, K different types of hidden states are designed for the K children of node t:

$$\boldsymbol{h}_k = f_k(t, \boldsymbol{h}, \boldsymbol{x}) \,, k = 1, 2, \ldots, K \tag{4}$$

where $\{f_k\}_{k=1}^{K}$ are activation functions which can be LSTM or other RNN cells. $\boldsymbol{h}$ denotes the hidden state fed to node t, containing the memory from t's ancestors , and $\boldsymbol{h}_r = \boldsymbol{0}$ for the root node. With $\boldsymbol{h}_k$, we define $p\big(c_k \mid \boldsymbol{x}, t, A(t), c_{<k}\big)$ as follows:

$$\begin{aligned} p\big(c_k \mid \boldsymbol{x}, t, A(t), c_{<k}\big) &= \frac{\exp g_k\big(c_k, \boldsymbol{x}, t, \boldsymbol{h}_k, \tilde{\boldsymbol{c}}_{k-1}\big)}{\sum_{v \in V} \exp g_k\big(v, \boldsymbol{x}, t, \boldsymbol{h}_k, \tilde{\boldsymbol{c}}_{k-1}\big)} \\ \tilde{\boldsymbol{c}}_{k-1} &= [c_1; c_2; \cdots; c_{k-1}] \end{aligned} \tag{5}$$

where $\tilde{\boldsymbol{c}}_{k-1}$ is the concatenation of $\{c_i\}_{i=1}^{k-1}$.

Modeling of parent-children dependency is summarized in Fig.2. With all these modelings, we train the X2TREE model by maximizing the data likelihood, namely

$$\begin{aligned} &\prod_{(\boldsymbol{x}, \mathcal{T}) \in D} p(\mathcal{T}|\boldsymbol{x}) \\ &= \prod_{(\boldsymbol{x}, \mathcal{T}) \in D} p(t_{\mathrm{r}}|\boldsymbol{x}) \prod_{t \in \mathcal{T}} \prod_{c_k \in C(t)} p\big(c_k \mid \boldsymbol{x}, t, A(t), c_{<k}\big) \end{aligned} \tag{6}$$

It is worth mentioning that in order to explicitly notify the end of tree generation we need to add the special token "EOB" (short for "End Of Branch") to the leaf nodes as their children. Hence, all the leaf nodes of the tree in the training dataset are EOB nodes.

Tree Canonicalization

As aforementioned, the proposed X2TREE model requires that the tree is K-ary full tree. Whereas for dialogue generation task a response sentence can be parsed into a dependency tree with any number of child nodes at each level. During training and generating, it is difficult to determine the

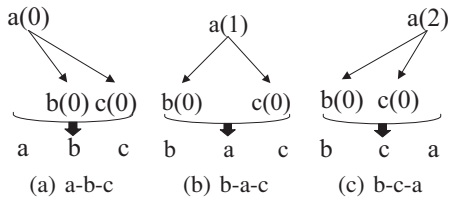


Figure 3: Three different sequence-preserved trees.

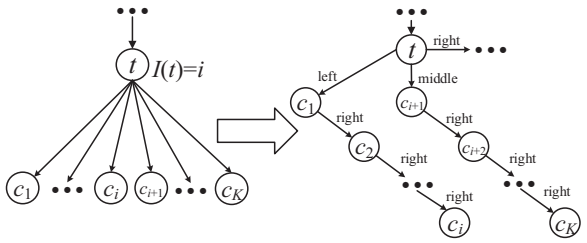


Figure 4: The canonicalization for node t.

child-node number of a word. Additionally, variable-length data is tricky for GPU acceleration. Hence, the original dependency tree is canonicalized into K-ary full tree before training.

Basically, the transformed K-ary full tree should be equivalent to the original one. In other words, there must exist an algorithm to support the bi-directional transformation between a tree and its K-ary full counterpart. Considering the number of K is linear to the number of model parameters, to reduce model complexity, we usually hope K to be as small as possible. For a given tree, a simple method to transform it into a full tree is to fill all the empty positions with EOB nodes. With this method, every tree node obtains K children where K is the maximal number of immediate children over all tree nodes. However, when K is large and the tree nodes are sparse, the redundant EOB nodes significantly increase the learning complexity. Hence, ideally, before the EOB filling step we want to transform the tree into a binary or ternary tree.

Here, we mainly consider two scenarios. For an ordered tree, where ordering is specified for the children of each node, we transform it to a left-child right-sibling (LCRS) binary tree (Cormen 2009). This transformation is reversible with a one-to-one mapping between the ordered tree and its LCRS counterpart. Furthermore, for the conversational generation tasks, we need to flatten the predicted tree into a sequence. Therefore we need to store position information in the dependency tree. For this purpose we first give the following definition of *sequence-preserved tree* (SP tree for short).

Definition 1. *An SP tree is an ordered tree where each node t is tagged an integer $I(t) \in \{0, 1, \cdots, K\}$, and K is the number of children to node t.*

The in-order traversal of an SP tree corresponds to a node sequence. Node t's children are divided into two parts. The left part contains the first $I(t)$ nodes (child nodes are ordered from left to right), while the right part contains the remaining nodes. In the in-order traversal we first visit the nodes in the left part, then the current node, finally the right part. Fig. 3 shows three SP trees with their corresponding sequences. Obviously, the dependency tree of a sentence is an SP tree, where the number attached on each node can be obtained by checking the position relationship of the node and its children in the original sentence, as shown in Figure 1(b). For example, the node "says" obtains a number "1" which means one child of this node are on its left part in the original sequence. As discussed earlier, a tree canonicalization step is needed to transform the original dependency tree into a K-ary full tree. To preserve sequence order, we transform the dependency into a ternary tree. We now present the algorithm and discuss why ternary tree is the "best" choice. Alg. 1 details this canonicalization process, and an illustration is shown in Fig.4.

In a ternary tree, each node has three children, namely left, middle and right nodes. For node t with attached number $I(t)$, Alg. 1 first determines its left and middle child in the ternary tree. Specifically, its left child is set to c_1, the first child in the original tree; and its middle child is set to $c_{I(t)+1}$. Any other child $c_j\big(j \neq 1$ and $j \neq I(t)+1\big)$ is set as the right child of c_{j-1} recursively. With this ternary tree a simple in-order traversal in the order of left child, parent, middle child and right child can restore it into a sequence.

Algorithm 1 CANONICALIZE

Input: A node of SP tree, t
Output: Ternary tree node corresponding to t, t'
Let $c_1, c_2, \cdots, c_n$ denote t's children;
Create an new node $t' = t$;
for $j \leftarrow 1$ **to** n **do**
  currentNode $\leftarrow$ CANONICALIZE(c_j);
  if $j = 1$ **and** $j \leqslant I(t)$ **then**
    t'.leftChild $\leftarrow$currentNode;
  else if $j = I(t) + 1$ **then**
    t'.middleChild $\leftarrow$currentNode;
  else
    lastNode.rightChild $\leftarrow$currentNode;
  end if
  lastNode $\leftarrow$ currentNode;
end for
return t';

Next, we prove that the resulting ternary tree is equivalent to the original SP tree in the sense that they can be transformed into each other.

Theorem 1. *Given any SP tree $\mathcal{T}$, it can be transformed into a ternary tree $\mathcal{T}'$, and $\mathcal{T}'$ can be transformed back into the original tree $\mathcal{T}$.*

Proof. Using the Alg.2, we can transform $\mathcal{T}$ into a ternary tree $\mathcal{T}'$.

We now show how to transform $\mathcal{T}'$ back into $\mathcal{T}$. For each node $t \in \mathcal{T}'$, if t is not a right child, let r_1 denote the right child of t, r_2 denote the right child of r_1, r_n denote the right child of r_{n-1} until r_n obtains no right child.

In the original tree $\mathcal{T}$, t and $\{r_j\}_{j=1}^{n}$ must be siblings. For simplicity, let t' denote their parent.

1) If t is a left child in $\mathcal{T}'$, $(t\,, r_1\,, \cdots, r_n)$ are first, second, $\ldots$, $(n+1)$-th child of t' in the original SP tree $\mathcal{T}$.

2) If t is a middle child in $\mathcal{T}'$, $(t\,, r_1\,, \cdots, r_n)$ are $\big(I(t')+1\big)$-th, $\big(I(t')+2\big)$-th, $\ldots$, $\big(I(t')+n+1\big)$-th child of t' in the original SP tree $\mathcal{T}$.

In this way, for each node in $\mathcal{T}'$, we can find its original position in $\mathcal{T}$, and then re-converts $\mathcal{T}'$ to $\mathcal{T}$.

□

Additionally, we prove that ternary tree is the "best" choice for model complexity. Theoretically, a dependency tree is equivalent to a K-ary tree when $K \geqslant 3$. Since the number of K is linear to parameter size in the X2TREE model, we prefer simpler models with smaller values of K. Theorem 2 formally shows that SP trees are *not* equivalent to binary trees. Therefore, the ternary tree is the "best" choice. Thus before training, we perform a preprocessing step which converts each response into its corresponding dependency tree (instance of SP tree), and canonicalize them into ternary trees. A visualization of this canonicalization process is provided in the slides in the supplemental files.

Theorem 2. *Given any SP tree $\mathcal{T}$, no algorithm exists which transforms $\mathcal{T}$ into an LCRS tree $\mathcal{T}'$ and re-converts $\mathcal{T}'$ to $\mathcal{T}$.*

Proof. Let $\mathcal{S}^n$, $\mathcal{O}^n$ and $\mathcal{L}^n$ respectively denote the set of sequence-preserved trees, ordered trees and LCRS trees with n nodes. Since the ordered trees and LCRS trees obtain one-to-one correspondence (Cormen 2009), it can be inferred that the element number $|\mathcal{O}^n| = |\mathcal{L}^n|$.

For a node t in ordered tree, if t and its children obtain specified ordering, namely $I(t)$ is defined, it converts to a SP tree. Furthermore, for different $I(t)$, the SP trees are different. Thus, $|\mathcal{S}^n| > |\mathcal{O}^n| = |\mathcal{L}^n|$. Moreover, suppose that an algorithm exists that transforms $\mathcal{T}$ into an LCRS tree $\mathcal{T}'$, and re-converts $\mathcal{T}'$ to $\mathcal{T}$. This infers that $|\mathcal{S}^n| \leqslant |\mathcal{L}^n|$. It is contradictory to $|\mathcal{S}^n| > |\mathcal{L}^n|$.

□

Note that the generated tree is a full tree (the leaf nodes can be in the lower depth) but not a perfect tree. For a sentence with n words, the transformed k-ary full tree contains exactly $(kn+1)$ nodes, and the extra $(kn+1-n)$ nodes are the EOB tokens. Thus, only the $(kn+1-n)$ nodes induce the computing waste. To minimize this waste we expect k as small as possible. Theorems 1 and 2 tell that k=3 is the minimal number we can use so that the transformed tree is equivalent to the original dependency tree.

Tree Generation

With the trained model we can infer the most probable trees for a given input $\boldsymbol{x}$. In this section we develop a greedy search algorithm for this inference task.

The beam search is traditionally adopted for sequence structure generation At each step, it keeps G (called *global* beam size) best candidates with the maximal probabilities so far. Then, only those candidates are expanded next. For each candidate on the beam it grows a new node at the current end of the sequence. This process repeats recursively until all candidates end with EOB nodes.

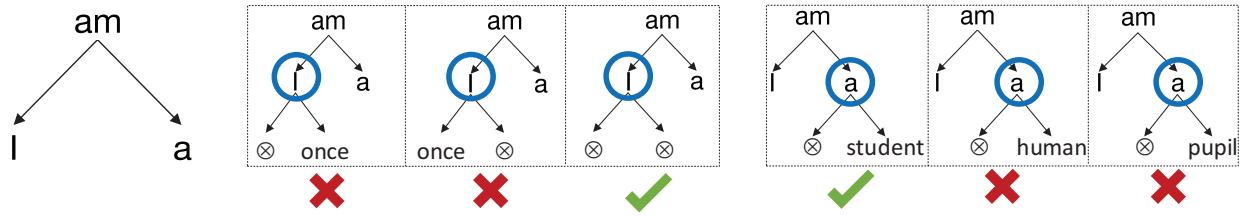
Algorithm 2 GENERALIZEDBEAMSEARCH

Input:
latent vector, $\boldsymbol{x}$
global beam size, G
local beam size, L
child number of each node, K
Output: A set of trees, $\mathcal{R}$

$\mathcal{S} \leftarrow \{G$ roots with highest $p(t_{\text{r}}|\boldsymbol{x})\}$; $\mathcal{R} \leftarrow \phi$;
while $|\mathcal{R}| < G$ **do**
  for each $\mathcal{T} \in \mathcal{S}$ **do**
    for each leaf $t \in \mathcal{T}$ **do**
      if $t =$ EOB **then**
        continue;
      end if
      Via chain beam search find L groups of $C(t)$ by maximizing $p\big(C(t)|\boldsymbol{x}, t, A(t)\big)$;
      for each $C(t)$ **do**
        Connect $C(t)$ to $\mathcal{T}$ as a new tree $\mathcal{T}'$;
        Add $\mathcal{T}'$ to $\mathcal{S}$;
      end for
    end for
    Delete $\mathcal{T}$ from $\mathcal{S}$;
  end for
  $\mathcal{S} \leftarrow \{G$ trees with highest $p(\mathcal{T}|\boldsymbol{x})$ in $\mathcal{S}\}$;
  for each $\mathcal{T} \in \mathcal{S}$ **do**
    if $\mathcal{T}$'s leaves are all EOBs **then**
      Add $\mathcal{T}$ to $\mathcal{R}$;
    end if
  end for
end while
$\mathcal{R} \leftarrow \{G$ trees with highest $p(\mathcal{T}|\boldsymbol{x})$ in $\mathcal{R}\}$;
return $\mathcal{R}$;

Since sequence is a special case of trees, searching tree generation has more challenges to address. First, an arbitrary tree has multiple leaves which could potentially generate new children. Second, when growing new children for a leaf node we need to generate all children as a whole since they correlate with each other (as mentioned in Section *Generative Model for K-ary Full Tree*). Multiple groups of such K children need to be generated as the best candidates.

We use the example in Fig. 5 to describe this tree generation method. The original tree has two leaves, nodes "*i*" and "*a*". For each of these leaves, we can generate new children. Specifically, for node "*i*" it generates L groups of K children, as shown in Fig. 5(b) ($L = 3$ and $K = 2$ in this example). Since these new children are ordered, this local step of children generation is actually a task of sequence generation, thus the conventional beam search can be used. Here, L (called *local* beam size) is to specify the number of candidate sequences generated for each leaf. After the child generation for all the leaves, we compare all these candidate trees and only retain top-G ($G = 2$ in this example) trees for the next round of generation. This process recursively continues until all the leaves in the tree are EOB nodes. Note that the proposed method is a generalized beam search. Beam search for sequence generation is a special case with $K = 1$, since sequence is equivalent to 1-ary tree. The method is detailed in Algorithm 2. A visualization of this search process is provided in the slides in the supplementary files.



(a) The original tree "I-am-a". (b) Extending the leave node "I". (c) Extending the leave node "a".

Figure 5: Examples of one step of generalized beam search. The Fig.(a) shows the original tree. The Fig. (b) and (c) show the searching results. Note that words in double quotes are to be expanded. Here, "⊗" denotes special token "EOB".

Experiment Settings

Dataset Details

Our experiments focus on dialogue generation task. 14 million post-response pairs were obtained from Tencent Weibo[2]. After removing spams and advertisements, $815,852$ pairs were left, among which $775,852$ are for training, and $40,000$ for model validation.

Benchmark Methods

We implemented the following four popular neural-based dialogue models for comparison:

1. SEQ2SEQ(Sutskever, Vinyals, and Le 2014): A RNN model that utilizes the last hidden state of the encoder as the initial hidden state of the decoder;
2. ENCDEC(Cho et al. 2014): A RNN model that feeds the last hidden state of the encoder to every cell and softmax unit of the decoder;
3. ATT(Bahdanau, Cho, and Bengio 2015): A RNN model based on ENCDEC with attention signal;
4. NRM(Shang, Lu, and Li 2015): Neural Responding Machine with both global and local schemes.

All these models map sequences to sequences directly, and only differ in how to summarize the encoder hidden states into a latent vector. Thus, the proposed tree decoder can be applied to any of these models, and potentially improve the response quality from a different perspective. Here, we stress that this tree-decoder can be easily applied to the model (Serban et al. 2015), which summarizes multiple rounds of dialogues into a latent vector. In the future, tree decoder for multi-round dialog will be evaluated.

Implementation Details

All sentences in the experiments are segmented by LTP. A vocabulary of 28,000 most frequent Chinese words in the corpus is used for training, which contains 97% words. Out-of-vocabulary words are replaced with "UNK". Our implementations are based on the Theano library (Bastien et al. 2012) over NVIDIA K80 GPU. We applied one-layer GRU (Cho et al. 2014) with 1,024-dimensional hidden states to $\{f_k\}_{k=1}^{K}$ and all baseline models. As suggested in (Shang, Lu, and Li 2015), the word embeddings for the encoders and decoders are learned separately, whose dimensions are set to 128 for all models. All the parameters were initialized using a uniform distribution between -0.01 and 0.01. In training, the mini-batch size is 128. We used ADADELTA (Zeiler 2012) for optimization. The training stops if the perplexity on the validation set increases for 4 consecutive epochs. Models with best perplexities are selected for further evaluation. When generating responses, for X2TREE we use generalized beam search with global beam size $G = 6$, local beam size $L = 6$. For other X2SEQ baseline models, conventional beam search with beam size 200 is used.

[2]http://t.qq.com/?lang=en_US

Evaluation Methods

Due to the high diversity nature of dialogs, it is practically impossible to construct a data set which adequately covers all responses for each given post. Hence, we apply human judgment to our experiments. In detail, 3 labelers were invited to evaluate the quality of responses to 300 randomly sampled posts. For each post, each model generated top-5 different responses (for a total of 25). For fair comparison, we create a single file in which each post is followed by its 25 responses which are shuffled to avoid labelers knowing which model each response is generated by.

For each response the labelers determine the quality to be one of the following three levels:

- **Level 1**: The response is ungrammatical.
- **Level 2**: The response is basically grammatical but irrelevant to the input post.
- **Level 3**: The response is grammatical and relevant to the input post. The response on this level is acceptable for dialog system.

From labeling results, average percentages of responses in different levels are calculated. Additionally, labeling agreement is evaluated by Fleiss' kappa (Fleiss 1971) which is a measure of inter-rater consistency. Furthermore, we also report BLEU-4 (Papineni et al. 2002) scores for these 300 posts. Since some researchers indicate BLEU may not be a good measure for dialog evaluation(Liu et al. 2016), we consider human judgment as a major measure in experiments.

Experimental Results and Analysis

The experimental results are summarized in Table 1. For SEQ2SEQ, NRM and X2TREE, the agreement value is in

a range from 0.6 to 0.8 which is interpreted as "substantial agreement". Meanwhile, ENCDEC and ATT obtain a relatively higher kappa value between 0.8 to 1.0 which is "almost perfect agreement". Hence, we believe the labeling standard is considered clear which leads to high agreement among labelers.

Table 1: The results from human judgment.

Models	Level-1%	Level-2%	Level-3%	Agreement	BLEU
ENCDEC	0.44	58.89	40.67	0.8114	8.78
SEQ2SEQ	1.58	50.73	47.69	0.7834	12.45
ATT	2.31	45.31	52.38	0.8269	13.89
NRM	0.64	44.98	54.38	0.7809	13.73
X2TREE	0.44	34.02	65.53	0.7733	15.87

For the Level-3 (acceptable ratio), X2TREE visibly outperforms other models. The best baseline method NRM achieves 54.38% Level-3 ratio, while X2TREE reaches 65.53% with an increase percentage of 11.15%. This improvement is mainly due to less irrelevant (Level-2) responses being generated (34.02% v.s. 44.98%), indicating X2TREE outputs more acceptable responses.

We further notice from Table 1 that the percentage of ungrammatical (Level-1) responses from X2TREE is less than other baselines (equal to ENCDEC) and the BLEU score is greater than other baselines in the experiments. It shows that responses generated by the tree-structured decoder are more grammatical than those from the chain-structured decoders and demonstrate the X2TREE's robustness to parser errors. Additionally, X2TREE and ENCDEC achieve best grammatical ratio (99.56%), but ENCDEC fails in generating relevant responses. Hence, Tree Decoder can improve the response relevance in experiments. We conjecture the reason is that X2TREE firstly generate the core verb of the responses. The first generated may be more relevant to the post and makes the whole response more relevant to the post.

In summary, the experiments demonstrate that X2TREE is able to generate more grammatical and relevant responses, and also show X2TREE obtains the ability to generate correct trees.

Easiness to Learning

From Table 1, we discover that the percentage of grammatical responses from X2TREE visibly surpasses other models in the experiments. We conjecture that the tree-structured decoder is easier to learn because its hidden states need to store less information than their counterparts in a chain-structured decoder.

In detail, given a response utterance with length T, the hidden state at position t in a chain-structured decoder needs to store the information of all previous words $\boldsymbol{y}_{<t}$, the average size of $\boldsymbol{y}_{<t}$ is $\frac{T+1}{2}$ (with an extra EOS token). In contrast, in a tree-structured decoder, $\boldsymbol{h}_t$ only needs to store the information of its ancestors y_t. After transforming the response into a triple dependency tree structure, the average depth of nodes is $O(\sqrt{T})$ (Flajolet and Odlyzko 1982). In the worst case, the depth of a triple dependency tree is T, and the average number of ancestors of nodes is $\frac{T+1}{2}$, which is the same to the average size of $\boldsymbol{y}_{<t}$. Fig. 6 shows the average number of steps hidden states need to remember at different sequence lengths for our data set.

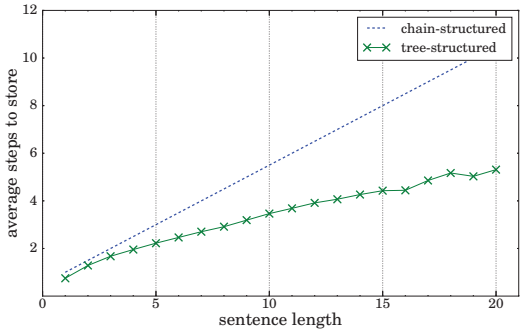


Figure 6: Average number of steps need to be stored for hidden states in both structures.

Overall, hidden states of a tree-structured decoder need store less information than chain-structured decoder's. This makes X2TREE potentially capable to handle more complex semantic structures in the response utterances.

Related Work

Statistical Machine Translation. The neural-based encoder-decoder framework for generative conversation models follows the line of statistical machine translation. Sutskever et al. (2014) used multi-layered LSTM as the encoder and the decoder for machine translation. Later, Cho et al. (2014) proposed the encoder-decoder framework, where the context vector is fed to every unit in the decoder. Bahdana et al. (2015) extended the encoder-decoder framework with the attention mechanism to model the alignment between source and target sequences.

Conversation models. Inspired by neural SMT, recent studies showed that these models can also be successfully applied to dialogue systems. Specifically, for short conversation, Shang et al. (2015) proposed the Neural Responding Machine which further extended the attention mechanism with both global and local schemes. Zhou et al. (2017) proposed MARM to generate diverse responses upon multiple mechanisms. Most recently, some researchers focused on multi-round conversation. Serban et al. (2015) built an end-to-end dialogue system using hierarchical neural network. Sordoni et al. (2015) proposed a related model with a hierarchical recurrent encoder-decoder framework for query suggestion. Our proposed model can also be applied to these multi-round conversation models and potentially improve the performances.

Tree-Structured Neural Network. Recently, some studies use tree-structured neural network instead of the conventional chain-structured neural network to improve the quality of semantic representation. Socher et al. (2013) proposed the Recursive Neural Tensor Network. Each phrase is represented by word vectors and its parse tree. Vectors of higher level nodes are computed using their child phrase vectors.

Tai et al. (2015) and Zhu et al. (2015) extended the chain-structured LSTM to tree structures. All above models use tree structures to summarize a sentence into a context vector, while we propose to decode from a context vector to generate sentences in a root-to-leaf direction. Additionally, Zhang et al.(2016) proposed Tree LSTM activation function in *top-down* fashion. Here, two important points differentiate our work with theirs. First, Zhang et al. mainly estimate generation probability of dependency tree and apply their model to sentence completion and dependency parsing reranking tasks, while X2TREE handles dialogue modeling in encoder-decoder framework. Second, due to the canonicalization method, X2TREE model process fixed number ($K = 3$) of children at each step for GPU acceleration, while Zhang et al. need to process the children sequentially. Thus, the proposed tree canonicalization method helps to reduce the training time. To this end, some works also aim at generating different structure types. Rabinovich et al. (2017) proposed the abstract syntax networks to transform card image of the game HearthStone into well-formed and executable outputs. Cheng et al. (2017) utilized predicate-argument structures to store natural language utterances as intermediate and domain-general representations.

Conclusion and Future Work

In this study, we proposed a tree-structured decoder to improve the response quality in dialogue systems. By incorporating linguistic knowledge into the modeling process, the proposed X2TREE framework outperforms baseline methods over 11.15% increase of acceptance ratio in response generation. Future study on incorporating a tree-structured encoder is promising to further enhance the sentence generation quality.

Acknowledgments

This work was supported by the National Key Research and Development Program of China under Grant No. 2017YFB1002104, the National Natural Science Foundation of China (No.61473274, 61573335).

This work was also supported by WeChat Tencent. We thank Leyu Lin, Lixin Zhang, Cheng Niu and Xiaohu Cheng for their constructive advices. We also thank the anonymous AAAI reviewers for their helpful feedback.

References

Bahdanau, D.; Cho, K.; and Bengio, Y. 2015. Neural Machine Translation by Jointly Learning to Align and Translate. *ICLR*.

Bastien, F.; Lamblin, P.; Pascanu, R.; Bergstra, J.; Goodfellow, I. J.; Bergeron, A.; Bouchard, N.; and Bengio, Y. 2012. Theano: New Features and Speed Improvements. In *NIPS Workshop*.

Cheng, J.; Reddy, S.; Saraswat, V.; and Lapata, M. 2017. Learning Structured Natural Language Representations for Semantic Parsing. In *ACL*.

Cho, K.; van Merrienboer, B.; Gulcehre, C.; Bahdanau, D.; Bougares, F.; Schwenk, H.; and Bengio, Y. 2014. Learning Phrase Representations using RNN Encoder-decoder for Statistical Machine Translation. In *EMNLP*.

Cormen, T. H. 2009. *Introduction to algorithms*. MIT press.

Flajolet, P., and Odlyzko, A. 1982. The Average Height of Binary Trees and Other Simple Trees. *JCSS*.

Fleiss, J. L. 1971. Measuring Nominal Scale Agreement Among Many Raters. *Psychological Bulletin*.

Hochreiter, S., and Schmidhuber, J. 1997. Long Short-Term Memory. *Neural Computation*.

Liu, C.-W.; Lowe, R.; Serban, I. V.; Noseworthy, M.; Charlin, L.; and Pineau, J. 2016. How NOT To Evaluate Your Dialogue System: An Empirical Study of Unsupervised Evaluation Metrics for Dialogue Response Generation. In *ACL*.

Papineni, K.; Roukos, S.; Ward, T.; and Zhu, W. 2002. BLEU: a Method for Automatic Evaluation of Machine Translation. In *ACL*.

Rabinovich, M.; Stern, M.; and Klein, D. 2017. Abstract Syntax Networks for Code Generation and Semantic Parsing. In *ACL*.

Serban, I. V.; Sordoni, A.; Bengio, Y.; Courville, A.; and Pineau, J. 2015. Building End-To-End Dialogue Systems Using Generative Hierarchical Neural Network Models. In *AAAI*.

Shang, L.; Lu, Z.; and Li, H. 2015. Neural Responding Machine for Short-Text Conversation. In *ACL*.

Socher, R.; Lin, C. C.; Ng, A. Y.; and Manning, C. D. 2011. Parsing Natural Scenes and Natural Language with Recursive Neural Networks. In *ICML*.

Socher, R.; Perelygin, A.; Wu, J. Y.; Chuang, J.; Manning, C. D.; Ng, A. Y.; and Potts, C. 2013. Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. In *ACL*.

Sordoni, A.; Bengio, Y.; Vahabi, H.; Lioma, C.; Simonsen, J. G.; and Nie, J.-Y. 2015. A Hierarchical Recurrent Encoder-Decoder For Generative Context-Aware Query Suggestion. In *CIKM*.

Sutskever, I.; Vinyals, O.; and Le, Q. V. 2014. Sequence to Sequence Learning with Neural networks. In *NIPS*.

Tai, K. S.; Socher, R.; and Manning, C. D. 2015. Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks. In *ACL*.

Vinyals, O., and Le, Q. V. 2015. A Neural Conversational Model. *arXiv*.

Zeiler, M. D. 2012. ADADELTA: An Adaptive Learning Rate Method. *arXiv*.

Zhang, X.; Lu, L.; and Lapata, M. 2016. Top-down Tree Long Short-Term Memory Networks. In *NAACL*.

Zhou, G.; Luo, P.; Cao, R.; Lin, F.; Chen, B.; and He, Q. 2017. Mechanism-aware neural machine for dialogue response generation. In *AAAI*.

Zhu, X.; Sobhani, P.; and Guo, H. 2015. Long Short-Term Memory Over Tree Structures. In *ICML*.