A Tree-Structured Decoder for Image-to-Markup Generation

Jianshu Zhang [1 2] **Jun Du** [1] **Yongxin Yang** [3] **Yi-Zhe Song** [3] **Si Wei** [2] **Lirong Dai** [1]

Abstract

Recent encoder-decoder approaches typically employ string decoders to convert images into serialized strings for image-to-markup. However, for tree-structured representational markup, string representations can hardly cope with the structural complexity. In this work, we first show via a set of toy problems that string decoders struggle to decode tree structures, especially as structural complexity increases, we then propose a tree-structured decoder that specifically aims at generating a tree-structured markup. Our decoders works sequentially, where at each step a child node and its parent node are simultaneously generated to form a sub-tree. This sub-tree is consequently used to construct the final tree structure in a recurrent manner. Key to the success of our tree decoder is twofold, (i) it strictly respects the parent-child relationship of trees, and (ii) it explicitly outputs trees as oppose to a linear string. Evaluated on both math formula recognition and chemical formula recognition, the proposed tree decoder is shown to greatly outperform strong string decoder baselines.

1. Introduction

Attention-based encoder-decoder models have been proven to be effective in sequence-to-sequence learning tasks such as speech recognition (Bahdanau et al., 2016), machine translation (Bahdanau et al., 2015), and image-to-sequence learning tasks like scene text recognition (Cheng et al., 2017), image captioning (Xu et al., 2015). These encoder-decoder models typically employ an RNN-based string decoder to generate the target sequence using features extracted by an upstream encoder. This choice is intuitive since target sequences in these tasks exhibit no internal structure – they are essentially one-dimensional linear text.

[1]NEL-SLIP Lab, University of Science and Technology of China, Hefei, Anhui, China [2]iFLYTEK Research, Hefei, Anhui, China [3]SketchX Lab, University of Surrey, Guildford, Surrey, United Kingdom. Correspondence to: Jun Du <jundu@ustc.edu.cn>.

Proceedings of the 37th International Conference on Machine Learning, Vienna, Austria, PMLR 119, 2020. Copyright 2020 by the author(s).

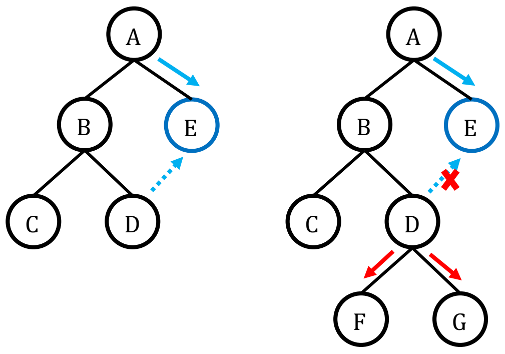


Figure 1. Full arrow indicates the decoding way of tree decoder and dotted arrow indicates the decoding way of string decoder for generating node "E". Left tree has string markup as "A (B (C D) E)" and sub-trees (child node - parent node) as "A-Root B-A C-B D-B E-A", right tree has string markup as "A (B (C D (F G)) E)" and sub-trees as "A-Root B-A C-B D-B F-D G-D E-A".

Nonetheless, many research problems naturally dictate the generation of both the text itself and the inherent structural relationships directly from images (Zanibbi et al., 2002; Harada et al., 2011; Xu et al., 2017; Sharma et al., 2018). These problems are typically referred to as image-to-markup generation (Deng et al., 2017): given an image x, the task is to generate a set of markups $\mathcal{M} = \{\mathcal{T}, \mathcal{S}\}$, where $\mathcal{T}$ is the text description and $\mathcal{S}$ is the associated structure. The underlying structural complexity of image markups is application dependent, yet trees have been commonly identified as a flexible representation – trees with branching factor of 1 is sufficient to tackle one-dimensional text sequence learning (Shi et al., 2018), whereas trees with larger branching factor and depth are typically required for problems such as math formula and chemical formula recognition (Staker et al., 2019; Wu et al., 2020).

Although string decoders can be used to tackle different structures by generating serialized sequences of target trees, by default they do not respect the structural integrity of trees. This is because string decoders typically work by serializing the structure by traversing in a depth-first order, yet with no specific design considerations to ensure the inherent parent-child relationships during traversal. This is best explained in

Figure 1. Assuming the left tree is a training sample, and the right being a testing sample. String decoders would learn to place node "E" after "D", despite "A" being its parent node. It follows that when testing on a complex tree on the right, string decoder will tend to generate node "E" right after it generates node "D", hence violating the structural integrity.

Furthermore, such a violation of structural integrity would have a direct impact on the generalization ability of string decoders, especially in terms of generalizing towards more complex structures. This is intuitive since as structural complexity arises, nodes that would otherwise follow a parent-child relationship, would be set further apart by the string encoder. This effect can also be observed in Figure 1, where newly introduced nodes in the right tree ("F" and "G") are inserted in front of "E" for the string representation. In fact, as we later discover via a set toy problems (Section 2), string decoders would completely collapse when decoding structures whose complexity was unobserved during training.

In this paper, we propose a tree-structured decoder that *treats trees as trees*: (i) it is specifically designed to respect the parent-child relationship of trees during decoding, and (ii) it explicitly produces a tree as final output. More specifically, in order to enable recurrent decoding, the target tree is decomposed into a sequence of sub-trees, where each sub-tree is composed of a parent node and a child node. Our tree decoder generates sub-trees in a sequential manner. At each time step, it first employs an attention model to *locate* the parent node, then the child node is *generated* based on the recognition result of its parent. Furthermore, we show this tree decoder can be jointly optimized with an encoder in an end-to-end fashion. Note that, the commonplace one-dimensional markup language can be seen as a special case for tree decoder, since the current parent is always a previous child of a super parent.

The contribution of this paper is threefold: (i) We propose a novel tree-structured decoder that respects tree structures and directly outputs trees, and can also be jointly optimized within an attention based encoder-decoder framework. (ii) We design a set of toy problems to spell out the reasons behind the failure of string decoders especially in terms of its generalization ability towards more complex structures, and in turn demonstrate why respecting tree structures during decoding can lead to superior generalization ability. (iii) We demonstrate the effectiveness of the proposed tree decoder on two practical problems of math formula recognition and chemical formula recognition, where structural complexity of markups is significant.

2. Motivation

In this section, we design a set of image-to-markup problems to spell out the differences between string and tree

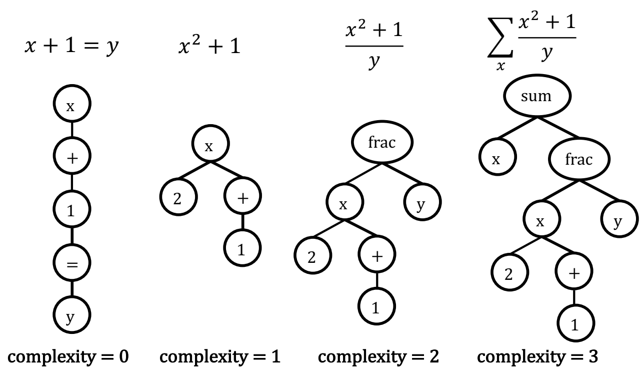


Figure 2. Illustration of math formulas with increased structural complexity.

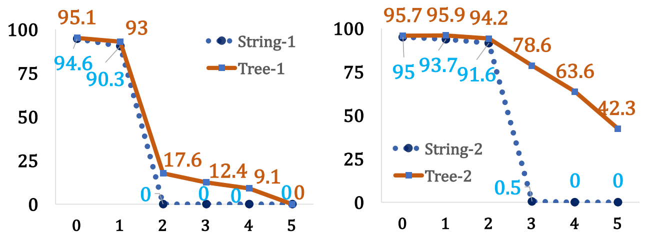


Figure 3. Expression recognition rate (ExpRate in %) of string decoders and tree decoders along test sets with different structural complexities $(0, 1, 2, \ldots, 5)$. "String-1" and "Tree-1" are trained on complexity $\{0, 1\}$, "String-2" and "Tree-2" are trained on complexity $\{0, 1, 2\}$.

decoders, especially in terms of their generalization ability. This first resulted in a key yet somewhat surprising discovery – string decoders would completely collapse when tested on complex tree structures that were unseen during training. This discovery largely motivated the introduction of our tree decoder which generalizes significantly better as structural complexity increases.

We define the structural complexity of a tree as the maximum number of non-terminal nodes containing more than one child node among all searching paths of the tree. Rendered math formulas are chosen to be the basis of our toy problems, for (i) difficulty of recognition is mostly with structural complexity other than symbol recognition, and (ii) the flexibility in generating structures of arbitrary complexities. How to parse a tree is problem dependent, and we follow the convention in the literature. As for math formulas, we use Label Graph (LG) (Zanibbi et al., 2013) to construct a tree. Figure 2 illustrates input images and target math trees of different structural complexities. For example, the one-dimensional formula like "$x + 1 = y$" has a branching factor of 1, which means it has no structural complexity as per our definition. Yet as more math structures like "superscript", "fraction" and "sum" are included, their structural complexity increases.

More specifically, we design six test sets at increasing structural complexity levels $(0, 1, 2, \ldots, 5)$. Each test set has 2,000 samples. We then introduce two train sets covering different range of complexities, where one train set includes complexity $\{0, 1\}$ ("String-1", "Tree-1"), and another train set includes complexity $\{0, 1, 2\}$ ("String-2", "Tree-2"). This is so that we can observe performance change on unseen complexities. Each train set contains $40,000$ samples per complexity level, so the training data is sufficiently large. The state-of-the-art DenseWAP string decoder model (Zhang et al., 2018) is used for comparison.

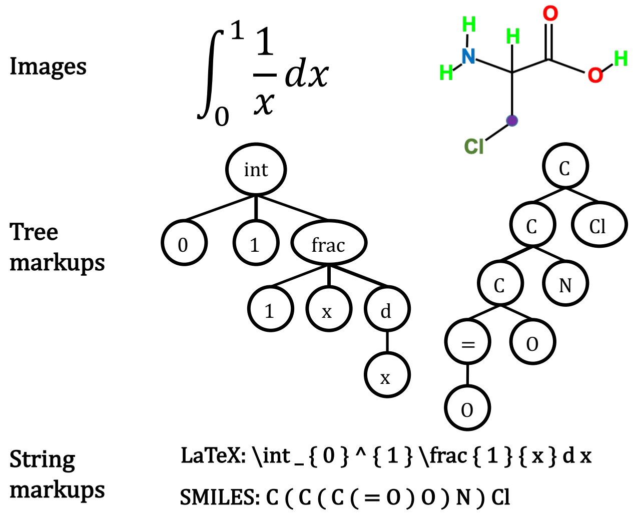
Results are shown in Figure 3. In general, we can conclude that string decoder has no generalization ability on unseen structural complexity, yet a situation tree decoder can handle very well. For example, when the train set only includes math formulas of complexity $\{0, 1\}$, string decoder drops to 0% recognition rate on testing math formulas of complexity 2 or larger. Tree decoder on the other hand shows much better generalization ability on the math formulas of these unseen structural complexity. The same trend can be observed when the train set only includes math formulas of complexity $\{0, 1, 2\}$, which further confirms our conclusion.

3. Related Work

3.1. Neural networks that deal with trees

There are some studies in the line of using neural network models to encode/decode a tree. Among them, research on encoding a tree has been well studied (Tai et al., 2015; Zhu et al., 2015; Socher et al., 2011; Eriguchi et al., 2016), which usually employs the recursive neural network to be a natural tree-structured encoder.

As for research on decoding a tree, some models (Rabinovich et al., 2017; Parisotto et al., 2017; Yin & Neubig, 2017) rely on specific grammar knowledge (e.g., grammar of a specific language for code generation) to decode the tree structure, which is hard to train and more importantly non-trivial to be adapted to other tasks. Some studies (Dong & Lapata, 2016; Vinyals et al., 2015; Aharoni & Goldberg, 2017) choose to employ string decoders to generate strings with structure tokens to implement a general tree decoder, but as introduced before, such string decoders are not natural for dealing with trees. General tree decoders for decoding tree structures has also been investigated in several studies (Chen et al., 2018; Chakraborty et al., 2018; Harer et al., 2019; Jaakkola, 2017; Dyer et al., 2016), these studies as well as our work both utilize the structural knowledge (e.g., relation between parent and child node) to enhance RNN based decoder. In summary, key differences between our work and those proposed general tree decoders: (i) We do not need to pre-process the tree into a specific format (e.g., convert it to binary tree). (ii) Our method can output the parent node and child node of each sub-tree explicitly, other than implicitly. (iii) Our method can deal with edge attributes.



LaTeX: \int _ { 0 } ^ { 1 } \frac { 1 } { x } d x

SMILES: C (C (C (= O) O) N) Cl

Figure 4. A math formula example with its related tree structure and serialized LaTeX string. A chemical formula example with its related tree structure and serialized SMILES string.

3.2. Image-to-Markup

Among image-to-markup problems, there commonly exists special markup languages which are naturally represented as tree structures other than flat strings, like math formula (Zanibbi et al., 2002) and chemical formula (Staker et al., 2019). To tackle these tree-structured languages, prior research (Zhang et al., 2019; Deng et al., 2017) commonly adapted string decoders with the hope that serialized strings can be used as an equivalent representation for tree-structured markups (e.g., LaTeX (Lamport, 1994) markup and SMILES (Weininger, 1988) markup in Figure 4). However, as already seen in the previous toy example (Section 2), and later shown in experiments (Section 5.3), such a string presentation can not cope well in the presence of complex structures. A tree-specific decoder like the one proposed is needed to fully capture the inherent structure properties.

4. Methodology

4.1. Tree-structured Image-to-Markup

Common approaches for tree-structured image-to-markup employ a string decoder. As shown in Figure 1, the markup (e.g., math formula) is serialized by traversing the tree following a depth-first order, and the structural layout information is reflected by the special tokens like "()". The target of string decoder consists of a sequence of tokens $y_1, y_2, \ldots, y_{T'}$, each y is a token in the serialized string, and T' is the length of the sequence.

The proposed tree decoder works in a different way. Its objective is to produce a sequence of sub-trees which directly form the ground-truth tree structure. The target consists of a sequence of sub-trees as $(o_1^{\mathrm{c}}, o_1^{\mathrm{p}}), (o_2^{\mathrm{c}}, o_2^{\mathrm{p}}), \ldots, (o_T^{\mathrm{c}}, o_T^{\mathrm{p}})$, each sub-tree includes a child node o^{c} and a parent node o^{p}, T is the length of sequence. We have $T \leq T'$ as there are no special tokens such as "()". The order of the sub-tree sequence is also determined by traversing the tree following a depth-first order. We can tell that the objective of string decoder and tree decoder is equivalent:

$$\max_{y} \prod_{t'} p(y_{t'}) \Longleftrightarrow \max_{o^{\mathrm{c}}, o^{\mathrm{p}}} \prod_{t} p(o_t^{\mathrm{c}}, o_t^{\mathrm{p}}) \tag{1}$$

At each step the tree decoder first outputs the parent node then outputs its child node:

$$\prod_t p(o_t^{\mathrm{c}}, o_t^{\mathrm{p}}) = \prod_t p(o_t^{\mathrm{c}} | o_t^{\mathrm{p}}) p(o_t^{\mathrm{p}}) \tag{2}$$

For training, the ground-truth parent is fed for predicting its child node (similar to teacher forcing in seq2seq models). For testing, the tree decoder first generates N most likely parent nodes, then the child nodes are predicted given these nodes:

$$\prod_t p(o_t^{\mathrm{c}} | o_t^{\mathrm{p}}) p(o_t^{\mathrm{p}}) \approx \sum_{i \in \mathrm{TopN}(p(o_t^{\mathrm{p}}))} \prod_t p(o_t^{\mathrm{c}} | o_t^{\mathrm{p}_i}) p(o_t^{\mathrm{p}_i}) \tag{3}$$

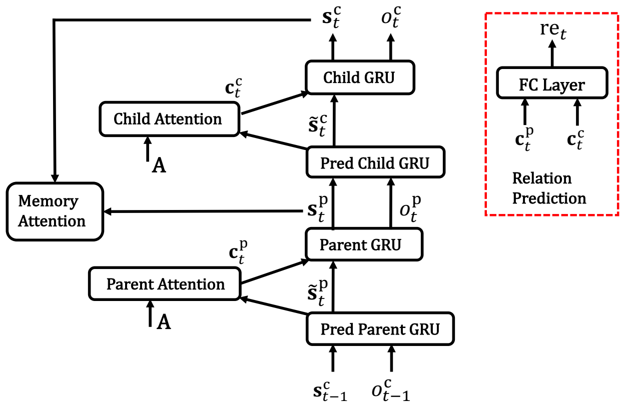
4.2. CNN Encoder

The CNN encoder used in this work is configured as densely connected layers (DenseNet) (Huang et al., 2017). We use the output of the last convolution layer as the feature, represented by $\mathbf{A}$. It can be seen as a grid, and each element in the grid is a feature vector $\mathbf{a}_i$ corresponding to a local region of the image: $\mathbf{A} = \{\mathbf{a}_i\}$.

4.3. Tree Decoder

The tree decoder is an RNN that produces a sequence of sub-tree structures $(o_1^{\mathrm{c}}, o_1^{\mathrm{p}}), (o_2^{\mathrm{c}}, o_2^{\mathrm{p}}), \ldots, (o_T^{\mathrm{c}}, o_T^{\mathrm{p}})$. Here we choose Gated Recurrent Units (GRU) (Chung et al., 2014), but other RNN alternatives should also work.

To guarantee that the predicted sub-tree sequence can reconstruct the full tree, we enforce two rules: (i) every child node must have a parent node, therefore there would not be any isolated nodes; (ii) the parent node must be one of the existing nodes.



$\mathbf{s}_t^{\mathrm{c}}$
o_t^{c}
Child GRU
$\mathbf{c}_t^{\mathrm{c}}$
Child Attention
$\tilde{\mathbf{s}}_t^{\mathrm{c}}$
A
Pred Child GRU
Memory Attention
$\mathbf{s}_t^{\mathrm{p}}$
o_t^{p}
$\mathbf{c}_t^{\mathrm{p}}$
Parent GRU
Parent Attention
$\tilde{\mathbf{s}}_t^{\mathrm{p}}$
A
Pred Parent GRU
$\mathbf{s}_{t-1}^{\mathrm{c}}$
o_{t-1}^{c}
re_t
FC Layer
$\mathbf{c}_t^{\mathrm{p}}$
$\mathbf{c}_t^{\mathrm{c}}$
Relation Prediction

Figure 5. Illustration of tree decoder, including parent decoder part, child decoder part, memory attention part and an optional relation prediction part. "Pred" is short for "prediction".

4.3.1. Parent Decoder

For the parent node decoding, we compute the prediction of current parent hidden state $\tilde{\mathbf{s}}_t^{\mathrm{p}}$ from the previous child node o_{t-1}^{c} and its hidden state $\mathbf{s}_{t-1}^{\mathrm{c}}$:

$$\tilde{\mathbf{s}}_t^{\mathrm{p}} = \mathrm{GRU}(o_{t-1}^{\mathrm{c}}, \mathbf{s}_{t-1}^{\mathrm{c}}) \tag{4}$$

Then we employ an attention mechanism with $\tilde{\mathbf{s}}_t^{\mathrm{p}}$ as the query and encoder features $\mathbf{A}$ as both the key and the value:

$$\boldsymbol{\alpha}_t^{\mathrm{p}} = f_{\mathrm{att}}^{\mathrm{p}}(\mathbf{A}, \tilde{\mathbf{s}}_t^{\mathrm{p}}) \tag{5}$$

$$\mathbf{c}_t^{\mathrm{p}} = \sum\nolimits_i \alpha_{ti}^{\mathrm{p}} \mathbf{a}_i \tag{6}$$

where $\boldsymbol{\alpha}_t^{\mathrm{p}}$ is the parent attention probabilities, $\mathbf{c}_t^{\mathrm{p}}$ is the parent context vector, $\mathbf{a}_i$ is the i-th element of $\mathbf{A}$. We design $f_{\mathrm{att}}^{\mathrm{p}}$ function as follows:

$$\mathbf{F}^{\mathrm{p}} = \mathbf{Q}^{\mathrm{p}} * \sum\nolimits_{l=1}^{t-1} \boldsymbol{\alpha}_l^{\mathrm{p}} \tag{7}$$

$$e_{ti}^{\mathrm{p}} = \boldsymbol{\nu}_{\mathrm{p}}^{\mathrm{T}} \tanh(\mathbf{W}_{\mathrm{att}}^{\mathrm{p}} \tilde{\mathbf{s}}_t^{\mathrm{p}} + \mathbf{U}_{\mathrm{att}}^{\mathrm{p}} \mathbf{a}_i + \hat{\mathbf{U}}_{\mathrm{F}}^{\mathrm{p}} \mathbf{f}_i^{\mathrm{p}}) \tag{8}$$

$$\alpha_{ti}^{\mathrm{p}} = \frac{\exp(e_{ti}^{\mathrm{p}})}{\sum_k \exp(e_{tk}^{\mathrm{p}})} \tag{9}$$

where α_{ti}^{p} denotes the parent attention probability of i-th element at decoding step t, $*$ denotes a convolution layer, $\sum_{l=1}^{t-1} \boldsymbol{\alpha}_l^{\mathrm{p}}$ denotes the sum of past parent attention probabilities, e_{ti}^{p} denotes the output energy, $\mathbf{f}_i^{\mathrm{p}}$ denotes the elements of $\mathbf{F}^{\mathrm{p}}$, which is used to help append the history information into standard attention mechanism.

With the parent context vector $\mathbf{c}_t^{\mathrm{p}}$, we compute the parent decoder state:

$$\mathbf{s}_t^{\mathrm{p}} = \mathrm{GRU}(\mathbf{c}_t^{\mathrm{p}}, \tilde{\mathbf{s}}_t^{\mathrm{p}}) \tag{10}$$

4.3.2. Child Decoder

Based on parent node o_t^{p} and its hidden state $\mathbf{s}_t^{\mathrm{p}}$, we compute the prediction of child node's hidden state $\tilde{\mathbf{s}}_t^{\mathrm{p}}$ at the same step:

$$\tilde{\mathbf{s}}_t^{\mathrm{c}} = \mathrm{GRU}(o_t^{\mathrm{p}}, \mathbf{s}_t^{\mathrm{p}}) \tag{11}$$

Then we use the same attention mechanism to compute the child attention probabilities $\boldsymbol{\alpha}_t^{\mathrm{c}}$ and the child context vector $\mathbf{c}_t^{\mathrm{c}}$:

$$\boldsymbol{\alpha}_t^{\mathrm{c}} = f_{\mathrm{att}}^{\mathrm{c}}(\mathbf{A}, \tilde{\mathbf{s}}_t^{\mathrm{c}}) \tag{12}$$

$$\mathbf{c}_t^{\mathrm{c}} = \sum\nolimits_i \alpha_{ti}^{\mathrm{c}} \mathbf{a}_i \tag{13}$$

$f_{\mathrm{att}}^{\mathrm{c}}$ has the same architecture of $f_{\mathrm{att}}^{\mathrm{p}}$ but with different parameters. The child node hidden state is then computed:

$$\mathbf{s}_t^{\mathrm{c}} = \mathrm{GRU}(\mathbf{c}_t^{\mathrm{c}}, \tilde{\mathbf{s}}_t^{\mathrm{c}}) \tag{14}$$

Finally, the probability of each predicted child node o_t^{c} is computed from the concatenation of parent node o_t^{p}, child node hidden state $\mathbf{s}_t^{\mathrm{c}}$ and child context vector $\mathbf{c}_t^{\mathrm{c}}$:

$$p(o_t^{\mathrm{c}}) = \mathrm{softmax}\left(\mathbf{W}_{\mathrm{out}}^{\mathrm{c}}(o_t^{\mathrm{p}}, \mathbf{s}_t^{\mathrm{c}}, \mathbf{c}_t^{\mathrm{c}})\right) \tag{15}$$

The classification loss of child decoder part is:

$$\mathcal{L}_{\mathrm{c}} = -\sum\nolimits_t \log p(w_t^{\mathrm{c}}) \tag{16}$$

where w_t^{c} represents the ground-truth child node at time step t.

4.3.3. Memory based Attention

In the sub-tree sequence, the child nodes always follow the depth-first order, but there is no explicit order for parent nodes, which causes some difficulties for recognizing them. For example, a math formula "$x + x^2$" has the sub-tree sequence "$(x, \mathrm{root}), (+, x), (x, +), (2, x)$". In this sequence, there are two child nodes denoting math symbol "x". Because child nodes follow a certain order, two "x" can be distinguished according to different decoding steps. For parent nodes, this is no longer reliable. Multiple occurrences of one math symbol in one math formula will bring ambiguities, e.g., we can not determine whether the parent node "x" of symbol "2" in the last sub-tree corresponds to the first "x" or the second "x" of formula "$x + x^2$".

To deal with this problem, we propose to use child nodes' positions in the ordered sub-tree sequence as an intermediate variable to denote the parent node. Hence, we get the intermediate sub-tree sequence as "$(x, 0), (+, 1), (x, 2), (2, 3)$", then we know the parent node of math symbol "2" is the child node of third sub-tree, which is the second "x" of math formula "$x + x^2$". In other words, we treat predicting the parent node as finding which previous child node should be the current parent node.

Here, we introduce a memory based attention module to help generate the intermediate parent node sequence and compute the objective function for training the parent decoder.

Following Section 4.3.1 and Section 4.3.2, we get the parent decoder state $\mathbf{s}_t^{\mathrm{p}}$, child decoder state $\mathbf{s}_t^{\mathrm{c}}$ and child node output o_t^{c}. During decoding, we append the child decoder state $\mathbf{s}_t^{\mathrm{c}}$ into the key memory and append the child node output o_t^{c} into the value memory, respectively. Therefore, the $\mathbf{s}_t^{\mathrm{c}}$ in the key memory and the o_t^{c} in the value memory both follow the order of child node sequence.

As we try to find which previous child node should be the current parent node, we use current parent decoder state $\mathbf{s}_t^{\mathrm{p}}$ as the query vector and employ child decoder states stored in key memory $\mathbf{s}_j^{mem}$ as the key vectors to compute the probabilities:

$$\mathbf{d}_{tj}^{mem} = \tanh(\mathbf{W}_{\mathrm{mem}}\mathbf{s}_t^{\mathrm{p}} + \mathbf{U}_{\mathrm{mem}}\mathbf{s}_j^{mem}) \tag{17}$$

$$G_{tj}^{mem} = \sigma(\boldsymbol{\nu}_{\mathrm{mem}}^{\mathrm{T}}\mathbf{d}_{tj}^{mem}) \tag{18}$$

The training loss of parent decoder part is then defined as a binary classification loss:

$$\mathcal{L}_{\mathrm{p}} = -\sum\nolimits_t \sum\nolimits_j [\bar{G}_{tj}^{\mathrm{mem}} \log(G_{tj}^{\mathrm{mem}}) + (1 - \bar{G}_{tj}^{\mathrm{mem}}) \log(1 - G_{tj}^{\mathrm{mem}})] \tag{19}$$

where $\bar{G}_{tj}^{\mathrm{mem}}$ denotes the ground-truth of parent node. $\bar{G}_{tj}^{\mathrm{mem}}$ is 1 if j-th child node stored in memory is the parent node of time step t, otherwise 0.

In testing stage, we choose $o_{\hat{j}}^{\mathrm{c}}, \hat{j} = \mathrm{argmax}(\mathbf{G}_{tj}^{\mathrm{mem}})$ in the value memory as parent node.

4.3.4. Relation Prediction

For some tasks, e.g., math formula recognition, the edge between parent and child has its own attribute, denoted as relation. For example, to fully represent math formula "$x + x^2$", the sub-tree structure should be "$(x, \mathrm{root}, \mathrm{start}), (+, x, \mathrm{right}), (x, +, \mathrm{right}), (2, x, \mathrm{sup})$", where "sup" means "superscript" relationship. As the parent context vector and child context vector both contain the spatial information, as well as the content information of parent node and child node, they are sufficient to generate the relationship between parent and child node:

$$p^{\mathrm{re}}(o_t^{\mathrm{re}}) = \mathrm{softmax}\left(\mathbf{W}_{\mathrm{out}}^{\mathrm{re}}(\mathbf{c}_t^{\mathrm{p}}, \mathbf{c}_t^{\mathrm{c}})\right) \tag{20}$$

The training loss of relation generation is computed as:

$$\mathcal{L}_{\mathrm{re}} = -\sum\nolimits_t \log p^{\mathrm{re}}(v_t) \tag{21}$$

where v_t represents the ground-truth relation at time step t.

4.4. Attention Self-Regularization

As the parent node at time step t must be the child node at a previous time step, there is correlation between parent attention probabilities and child attention probabilities. Take

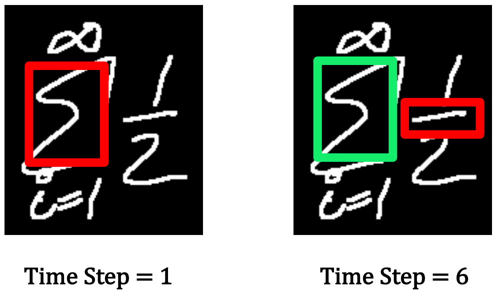


Figure 6. Illustration of what child attention and parent attention focuses at time step 1 and at time step 6. Red box denotes the region that child attention focuses on, green box denotes the region that parent attention focuses on.

the math formula in Figure 6 as an example, at time step 1, the child node is "$\sum$", the parent node is "root", while at time step 6, the child node is "fraction", the parent node is "$\sum$". We can see the parent node at time step 6 is the child node at time step 1, therefore we can infer that the parent attention probabilities at time step 6 should be similar to the child attention probabilities at time step 1.

To implement this regularization between parent attention and child attention, we first rearrange the order of child attention probabilities such that they match the target of parent attention probabilities (e.g., in Figure 6, the target of parent attention $\hat{\boldsymbol{\alpha}}_6^{\mathrm{p}} = \boldsymbol{\alpha}_1^{\mathrm{c}}$). Then we can get a sequence of target parent attention probabilities as $\hat{\boldsymbol{\alpha}}_1^{\mathrm{p}}, \hat{\boldsymbol{\alpha}}_2^{\mathrm{p}}, \ldots, \hat{\boldsymbol{\alpha}}_T^{\mathrm{p}}$. The regularization function is the Kullback-Leibler (KL) divergence between $\hat{\boldsymbol{\alpha}}_t^{\mathrm{p}}$ and $\boldsymbol{\alpha}_t^{\mathrm{p}}$:

$$\mathcal{L}_{\mathrm{reg}} = -\sum\nolimits_t \hat{\boldsymbol{\alpha}}_t^{\mathrm{p}} \log \frac{\hat{\boldsymbol{\alpha}}_t^{\mathrm{p}}}{\boldsymbol{\alpha}_t^{\mathrm{p}}} \tag{22}$$

5. Experiments

5.1. Implementation Details

5.1.1. Training

The overall recognition model is trained end-to-end. The training objective of our model is to minimize the child node loss (Eq. (16)), parent node loss (Eq. (19)) and attention self-regularization loss Eq. (22). For markup languages that need to consider relation between child node and parent node, our model also minimizes the relation loss (Eq. (21)). The objective function for optimization is shown as follows:

$$O = \lambda_1 \mathcal{L}_{\mathrm{c}} + \lambda_2 \mathcal{L}_{\mathrm{p}} + \lambda_3 \mathcal{L}_{\mathrm{re}} + \lambda_4 \mathcal{L}_{\mathrm{reg}} \tag{23}$$

In our experiments, we set $\lambda_1 = \lambda_2 = 1$ to reflect the fact that the child node and parent node are equally important. For math formula recognition, $\lambda_3 = 1$, for chemical formula recognition, $\lambda_3 = 0$ as we do not need to specify relationship. We set $\lambda_4 = 0.1$ for regularization loss.

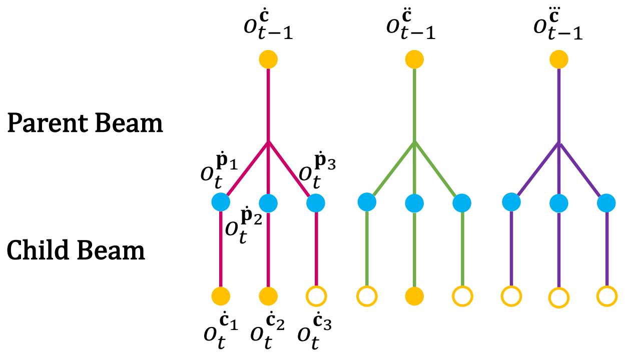

$o_{t-1}^{\dot{\mathrm{c}}}$
$o_{t-1}^{\ddot{\mathrm{c}}}$
$o_{t-1}^{\dddot{\mathrm{c}}}$
Parent Beam
$o_t^{\dot{\mathrm{p}}_1}$
$o_t^{\dot{\mathrm{p}}_3}$
$o_t^{\dot{\mathrm{p}}_2}$
Child Beam
$o_t^{\dot{\mathrm{c}}_1}$
$o_t^{\dot{\mathrm{c}}_2}$
$o_t^{\dot{\mathrm{c}}_3}$

Figure 7. Illustration of expanded beam search. As for the last 9 beams, only the 3 partial hypotheses with full yellow child node are kept.

To ensure a fair comparison with state-of-the-art LaTeX string decoder based methods, we use the same DenseNet encoder employed in the DenseWAP model (Zhang et al., 2018). We employ three dense blocks in the main branch. We set the growth rate to $k = 24$, the depth (number of convolution layers) of each block to $D = 32$. A Batch Normalization (Ioffe & Szegedy, 2015) layer and a ReLU activation (Nair & Hinton, 2010) layer are placed after each convolution layer.

Both the child and parent decoders adopt 2 unidirectional GRU layers, each layer has 256 forward GRU units. The child attention dimension, parent attention dimension and memory attention dimension are set to 512. The embedding dimensions for both child node and parent node are set to 256.

We employ the ADADELTA algorithm (Zeiler, 2012) for optimization, with the following hyper parameters: $\rho = 0.95$, and $\varepsilon = 10^{-6}$. The whole framework was implemented using PyTorch. Experiments were conducted on a single Nvidia Tesla V100 with 16GB RAM.

5.1.2. Testing

During testing, as we do not have ground-truth child and parent node, we resort to a hierarchical version of the traditional beam search algorithm (Reddy et al., 1977). More specifically, as illustrated in Figure 7, a beam size of 3 is set for both parent and child beam in our experiments. In the parent beam, 3 most likely previous child nodes $\{o_{t-1}^{\mathrm{c}}\}$ are kept to compute the current parent nodes $\{o_t^{\mathrm{p}}\}$. Then in the corresponding child beam, each partial hypothesis is expanded with 3 most likely current parent nodes, which are used to compute current child nodes $\{o_t^{\mathrm{c}}\}$. Therefore, at each step, we have $3 \times 3 = 9$ beams out in total. We then choose 3 beams from these 9 hypotheses based on the combined score of parent beam and child beam (i.e., combined likelihood of $\{o_t^{\mathrm{p}}\}$ and $\{o_t^{\mathrm{c}}\}$) for next decoding step.

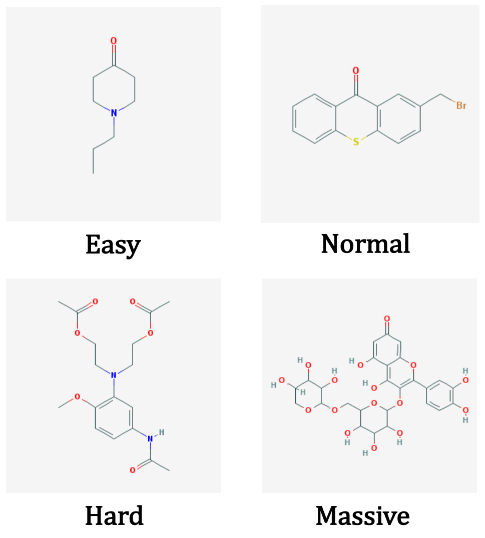


Figure 8. Split the SMILES test set into four sub-sets ("Easy", "Normal", "Hard", "Massive") based on the length of testing SMILES strings.

5.2. Datasets

5.2.1. Handwritten Math Formula

For math formula recognition, we evaluate our model on CROHME benchmark (Mouchère et al., 2016b;a; Mahdavi et al., 2019), which is currently the largest dataset for online handwritten math formula recognition. We convert the online handwritten math formula from trajectory sequence format into image format for implementing image-to-markup.

The CROHME training set contains 8,836 handwritten math formulas. There are 6 math relations between parent node and child node ("above", "below", "right", "inside", "superscript", "subscript"). We evaluate our model on three CROHME test sets. CROHME 2014/2016/2019 contain 986/1,147/1,199 handwritten math formulas, respectively.

5.2.2. Chemical Formula

We also evaluate our model on chemical formula recognition as chemical formulas usually contain much more complicated structures than math formulas. The chemical formula recognition experiments are conducted based on SMILES dataset. SMILES (Jin et al., 2018) dataset provides a large mount of printed chemical formula images and corresponding SMILES strings. As the input is printed images, the effect of performance mostly depends on structure analysis. Therefore, the performance on SMILES chemical formula dataset can act as a good indicator towards the difference of structure understanding between string decoder and tree decoder.

We choose 100,000 chemical formulas from SMILES dataset. We use 90,000 formulas as train set, 3,000 formulas as validation set and the other 7,000 as test set. The structures of formulas in test set have never be seen in train set. We split the test test into four sub-sets based on the length of testing SMILES strings, and as shown in Figure 8. The length of SMILES strings can reflect the structural complexity and the depth of tree structures. The four subsets are named as "Easy", "Normal", "Hard", "Massive", and their percentages in the whole test set are 20%, 40%, 25% and 15% respectively. "Easy" set consists of chemical formulas with SMILES length in range of $[1, 20]$, "Normal" consists of length $[20, 40]$, "Hard" consists of length $[40, 80]$, "Massive" consists of length more than 80 and usually approaching 200.

5.3. Experimental Analysis

5.3.1. Results on CROHME

Results on CROHME can be seen in Table 1. As baselines, we first offer comparisons with several promising traditional methods that use tree grammars for parsing math formula. For fairness, we show the results of methods using only official 8,836 training samples, without any other training data or any additional language models. "UPV", "UNATES" and "TUAT" are the top 3 systems using only official training samples in the CROHME 2014 competition, and "TOKYO" is the best performing system using official training samples in the CROHME 2016 competition. As for string decoders, we offer comparative results with all five previous state-of-the-art image-to-LaTeX markup methods: "WYGIWYS", "PAL", "Transformer", "WAP" and "DenseWAP". The results of string decoder and tree decoder all comes from single model. Since the reported results of "DenseWAP" is of ensemble model, to be comparable with other results of single model, we train a single "DenseWAP" model using official published codes.

We can see from Table 1 that "DenseWAP" offers best result as a string decoder based image-to-markup model. Hence we implement our overall model ("DenseWAP-TD") by replacing the string decoder in "DenseWAP" with the proposed tree decoder. That is the encoder used in "DenseWAP-TD" has the same architecture as the encoder in "DenseWAP". Also, the optimization algorithm and basic hyper parameters of "DenseWAP-TD" and "DenseWAP" are all the same. By comparing the performance between "DenseWAP-TD" and "DenseWAP", it is clear to see that the proposed tree decoder outperforms the top performing string decoder by a significant margin for all test sets.

We can further analyze the difference on structure understanding between string decoder and tree decoder via attention visualization. In LaTeX strings, the token "{" and "}" denote the start and the end of a structure respectively. As shown in Figure 9, when string decoder meets the "subscript"

Table 1. Evaluation of math formula recognition systems on CROHME 2014, CROHME 2016 and CROHME 2019 test sets (in %). "ExpRate", "$\leq$ 1 s.error" and "$\leq$ 1 s.error" means expression recognition rate when 0 to 2 symbol or structural level errors can be tolerated, "StruRate" means structure recognition rate.

Dataset	Model	Decoder	ExpRate	$\leq$ 1 s.error	$\leq$ 2 s.error	StruRate
CROHME14	UPV	tree grammar	37.2	44.2	47.3	-
	UNATES	tree grammar	26.1	33.9	38.5	-
	TUAT	tree grammar	25.7	33.2	35.9	-
	WYGIWYS	string decoder	36.4	-	-	-
	PAL	string decoder	39.7	-	-	-
	WAP	string decoder	40.4	56.1	59.9	-
	DenseWAP	string decoder	43.0±1.0	57.8±1.4	61.9±1.8	63.2±1.7
	DenseWAP-TD	**tree decoder**	**49.1±0.9**	**64.2±0.9**	**67.8±1.0**	**68.6±1.6**
CROHME16	TOKYO	tree grammar	43.9	50.9	53.7	61.6
	WAP	string decoder	37.1	-	-	-
	DenseWAP	string decoder	40.1±0.8	54.3±1.0	57.8±0.9	59.2±0.8
	DenseWAP-TD	**tree decoder**	**48.5±0.9**	**62.3±0.9**	**65.3±0.7**	**65.9±0.6**
CROHME19	Transformer	string decoder	41.5	54.1	58.9	60.0
	DenseWAP	string decoder	41.7±0.9	55.5±0.9	59.3±0.5	60.7±0.6
	DenseWAP-TD	**tree decoder**	**51.4±1.3**	**66.1±1.4**	**69.1±1.2**	**69.8±1.1**

Table 2. Ablation study on CROHME 2014, CROHME 2016 and CROHME 2019 test sets (in %).

Dataset	Att Self-Reg	Ensemble	ExpRate
CROHME14	✗	✗	44.5±1.1
	✓	✗	49.1±0.9
	✓	✓	54.0±0.4
CROHME16	✗	✗	41.8±1.0
	✓	✗	48.5±0.9
	✓	✓	52.1±0.3
CROHME19	✗	✗	45.4±1.2
	✓	✗	51.4±1.3
	✓	✓	54.6±0.4

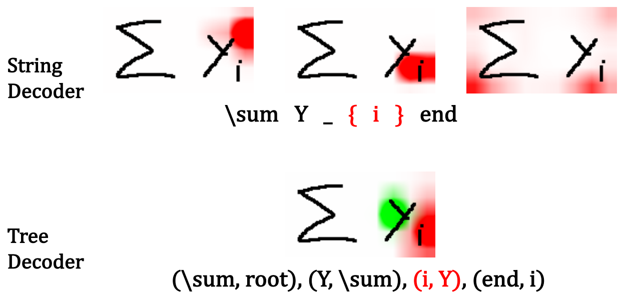


Figure 9. Comparison of attention visualization on analyzing "subscript" structure. For tree decoder, red color denotes the attention probabilities of child node, green color denotes the attention probabilities of parent node.

structure and tries to generate the pair of "{" and "}", the attention probabilities can be unseasonable. Therefore, we can infer that string decoder possibly relies on the implicit language model to generate "{" and "}", and it has no understanding of tree structures. This echoes well with our earlier discovery (Section 2) that the string decoder struggles to deal with unseen complicated tree structures. Conversely, the tree decoder can find the true parent node of subscript math symbol "i", indicating a good understanding of tree structure.

Ablation study on all three CROHME datasets can be found in Table 2. We first evaluate the effectiveness of the proposed attention self-regularization (Section 4.4). It can be observed that without it, only slight performance gains over string decoder can be observed. Whereas with it in play, recognition rate improved from 44.5% to 49.1% on CROHME 2014, from 41.8% to 48.5% on CROHME 2016, and from 45.4% to 51.4% on CROHME 2019. We then report results on the use of ensemble methods, and show that it can further improve the overall recognition performance. Following (Dietterich, 2000; Zhang et al., 2019), we implement the ensemble system by combing 5 models at each decoding step.

5.3.2. Results on SMILES

As introduced in Section 5.2.2, we split the SMILES test set into four sub-sets based on different range of SMILES

Table 3. Recognition rate comparison (in %) between string decoder and tree decoder on SMILES dataset. "Easy", "Normal", "Hard", "Massive" denote the four sub-sets of test set with different length of SMILES string, "All" means the overall recognition rate on the whole test set (in %). "SD" and "TD" refer to string decoder and tree decoder based approaches, respectively.

System	Easy	Normal	Hard	Massive	All
SD	88.9	82.8	60.7	28.2	72.7
TD	89.5	85.8	65.6	34.4	76.9

length. We can see from Table 3, tree decoder significantly outperforms string decoder. More specifically, in "Easy" set, where chemical formula SMILES string has length less than 20, the tree decoder offers relatively slight improvements. However, results for "Normal", "Hard" and "Massive", show that as the length of SMILES string increases, the performance gain of tree decoder over string decoder becomes much more significant.

6. Conclusions

In this paper, we propose a tree-structured decoder for improving the generalization ability of encoder-decoder models on markups with complicated structures. Our proposed tree decoder shows significant improvement compared with state-of-the-art string decoders on tree-structured image-to-markup problems. Despite evaluated on math and chemical formula recognition only, our tree decoder can also be used for common one-dimensional image-to-markup and sequence-to-sequence problems without much tweaking. Source code and the toy datasets will be publicly released to facilitate future research.

Acknowledgements

We thank the reviewers for their constructive comments. This work was supported in part by the National Key R&D Program of China under Grant 2017YFB1002202, the National Natural Science Foundation of China under Grants 61671422 and U1613211, the Fundamental Research Funds for the Central Universities, the MOE-Microsoft Key Laboratory of USTC, and the iFLYTEK-Surrey Joint Research Centre on Artificial Intelligence.

References

Aharoni, R. and Goldberg, Y. Towards string-to-tree neural machine translation. In *ACL*, 2017.

Bahdanau, D., Cho, K., and Bengio, Y. Neural machine translation by jointly learning to align and translate. In *ICLR*, 2015.

Bahdanau, D., Chorowski, J., Serdyuk, D., Brakel, P., and Bengio, Y. End-to-end attention-based large vocabulary speech recognition. In *ICASSP*, 2016.

Chakraborty, S., Allamanis, M., and Ray, B. Tree2tree neural translation model for learning source code changes. *arXiv*, abs/1810.00314, 2018.

Chen, X., Liu, C., and Song, D. Tree-to-tree neural networks for program translation. In *NeurIPS*, 2018.

Cheng, Z., Bai, F., Xu, Y., Zheng, G., Pu, S., and Zhou, S. Focusing attention: Towards accurate text recognition in natural images. In *ICCV*, 2017.

Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv*, abs/1412.3555, 2014.

Deng, Y., Kanervisto, A., Ling, J., and Rush, A. M. Image-to-markup generation with coarse-to-fine attention. In *ICML*, 2017.

Dietterich, T. G. Ensemble methods in machine learning. In *IWMCS*, pp. 1–15, 2000.

Dong, L. and Lapata, M. Language to logical form with neural attention. In *ACL*, 2016.

Dyer, C., Kuncoro, A., Ballesteros, M., and Smith, N. A. Recurrent neural network grammars. *arXiv*, abs/1602.07776, 2016.

Eriguchi, A., Hashimoto, K., and Tsuruoka, Y. Tree-to-sequence attentional neural machine translation. In *ACL*, 2016.

Harada, A., Kobayashi, R., Takashima, Y., Hashidzume, A., and Yamaguchi, H. Macroscopic self-assembly through molecular recognition. *Nature chemistry*, 2011.

Harer, J., Reale, C., and Chin, P. Tree-transformer: A transformer-based method for correction of tree-structured data. *arXiv*, abs/1908.00449, 2019.

Huang, G., Liu, Z., van der Maaten, L., and Weinberger, K. Q. Densely connected convolutional networks. In *CVPR*, 2017.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv*, abs/1502.03167, 2015.

Jaakkola, D. A.-M. . T. S. Tree-structured decoding with doubly-recurrent neural networks. In *ICLR*, 2017.

Jin, W., Barzilay, R., and Jaakkola, T. Junction tree variational autoencoder for molecular graph generation. In *ICML*, 2018.

Lamport, L. *LATEX: a document preparation system: user's guide and reference manual*. Addison-wesley, 1994.

Mahdavi, M., Zanibbi, R., Mouchère, H., Viard-Gaudin, C., and Garain, U. Icdar 2019 crohme+ tfd: Competition on recognition of handwritten mathematical expressions and typeset formula detection. 2019.

Mouchère, H., Viard-Gaudin, C., Zanibbi, R., and Garain, U. Icfhr2016 crohme: Competition on recognition of online handwritten mathematical expressions. In *ICFHR*, 2016a.

Mouchère, H., Zanibbi, R., Garain, U., and Viard-Gaudin, C. Advancing the state of the art for handwritten math recognition: the crohme competitions, 2011–2014. *IJDAR*, 2016b.

Nair, V. and Hinton, G. E. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

Parisotto, E., Mohamed, A.-r., Singh, R., Li, L., Zhou, D., and Kohli, P. Neuro-symbolic program synthesis. In *ICLR*, 2017.

Rabinovich, M., Stern, M., and Klein, D. Abstract syntax networks for code generation and semantic parsing. In *ACL*, 2017.

Reddy, D. R. et al. *Speech understanding systems: A summary of results of the five-year research effort*. Department of Computer Science. Camegie-Mell University, 1977.

Sharma, G., Goyal, R., Liu, D., Kalogerakis, E., and Maji, S. Csgnet: Neural shape parser for constructive solid geometry. In *CVPR*, pp. 5515–5523, 2018.

Shi, B., Yang, M., Wang, X., Lyu, P., Yao, C., and Bai, X. Aster: An attentional scene text recognizer with flexible rectification. *IEEE TPAMI*, 2018.

Socher, R., Lin, C. C.-Y., Ng, A. Y., and Manning, C. D. Parsing natural scenes and natural language with recursive neural networks. In *ICML*, 2011.

Staker, J., Marshall, K., Abel, R., and McQuaw, C. M. Molecular structure extraction from documents using deep learning. *JCIM*, 2019.

Tai, K. S., Socher, R., and Manning, C. D. Improved semantic representations from tree-structured long short-term memory networks. In *ACL*, 2015.

Vinyals, O., Kaiser, L., Koo, T., Petrov, S., Sutskever, I., and Hinton, G. Grammar as a foreign language. In *NeurIPS*, 2015.

Weininger, D. Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules. *JCICS*, 1988.

Wu, J.-W., Yin, F., Zhang, Y.-M., Zhang, X.-Y., and Liu, C.-L. Handwritten mathematical expression recognition via paired adversarial learning. *IJCV*, 2020.

Xu, D., Zhu, Y., Choy, C. B., and Fei-Fei, L. Scene graph generation by iterative message passing. In *CVPR*, 2017.

Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., Zemel, R., and Bengio, Y. Show, attend and tell: Neural image caption generation with visual attention. In *ICML*, 2015.

Yin, P. and Neubig, G. A syntactic neural model for general-purpose code generation. In *ACL*, 2017.

Zanibbi, R., Blostein, D., and Cordy, J. R. Recognizing mathematical expressions using tree transformation. *IEEE TPAMI*, 2002.

Zanibbi, R., Mouchere, H., and Viard-Gaudin, C. Evaluating structural pattern recognition for handwritten math via primitive label graphs. In *DRR*, volume 8658, pp. 865817, 2013.

Zeiler, M. D. Adadelta: an adaptive learning rate method. *arXiv*, abs/1212.5701, 2012.

Zhang, J., Du, J., and Dai, L. Multi-scale attention with dense encoder for handwritten mathematical expression recognition. In *ICPR*, 2018.

Zhang, J., Du, J., and Dai, L. Track, attend and parse (tap): An end-to-end framework for online handwritten mathematical expression recognition. *TMM*, 2019.

Zhu, X., Sobhani, P., and Guo, H. Long short-term memory over tree structures. In *ICML*, 2015.