The Inside-Outside Recursive Neural Network model for Dependency Parsing

Phong Le and **Willem Zuidema**
Institute for Logic, Language, and Computation
University of Amsterdam, the Netherlands
{p.le,zuidema}@uva.nl

Abstract

We propose the first implementation of an infinite-order generative dependency model. The model is based on a new recursive neural network architecture, the Inside-Outside Recursive Neural Network. This architecture allows information to flow not only bottom-up, as in traditional recursive neural networks, but also top-down. This is achieved by computing content as well as context representations for any constituent, and letting these representations interact. Experimental results on the English section of the Universal Dependency Treebank show that the infinite-order model achieves a perplexity seven times lower than the traditional third-order model using counting, and tends to choose more accurate parses in k-best lists. In addition, reranking with this model achieves state-of-the-art unlabelled attachment scores and unlabelled exact match scores.

1 Introduction

Estimating probability distributions is the core issue in modern, data-driven natural language processing methods. Because of the traditional definition of discrete probability

$$Pr(A) \equiv \frac{\text{the number of times A occurs}}{\text{the size of event space}}$$

counting has become a standard method to tackle the problem. When data are sparse, smoothing techniques are needed to adjust counts for non-observed or rare events. However, successful use of those techniques has turned out to be an art. For instance, much skill and expertise is required to create reasonable reduction lists for back-off, and to avoid impractically large count tables, which store events and their counts.

An alternative to counting for estimating probability distributions is to use neural networks. Thanks to recent advances in deep learning, this approach has recently started to look very promising again, with state-of-the-art results in sentiment analysis (Socher et al., 2013), language modelling (Mikolov et al., 2010), and other tasks. The Mikolov et al. (2010) work, in particular, demonstrates the advantage of neural-network-based approaches over counting-based approaches in language modelling: it shows that recurrent neural networks are capable of capturing long histories efficiently and surpass standard n-gram techniques (e.g., Kneser-Ney smoothed 5-gram).

In this paper, keeping in mind the success of these models, we compare the two approaches. Complementing recent work that focused on such a comparison for the case of finding appropriate word vectors (Baroni et al., 2014), we focus here on models that involve more complex, hierarchical structures. Starting with existing generative models that use counting to estimate probability distributions over constituency and dependency parses (e.g., Eisner (1996b), Collins (2003)), we develop an alternative based on recursive neural networks. This is a non-trivial task because, to our knowledge, no existing neural network architecture can be used in this way. For instance, classic recurrent neural networks (Elman, 1990) unfold to left-branching trees, and are not able to process arbitrarily shaped parse trees that the counting approaches are applied to. Recursive neural networks (Socher et al., 2010) and extensions (Socher et al., 2012; Le et al., 2013), on the other hand, do work with trees of arbitrary shape, but process them in a bottom-up manner. The probabilities we need to estimate are, in contrast, defined by top-down generative models, or by models that require information flows in both directions (e.g., the probability of generating a node depends on the whole fragment rooted at its just-generated sis-

Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 729–739,
October 25-29, 2014, Doha, Qatar.

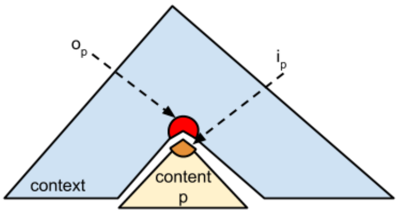

Figure 1: Inner ($\mathbf{i}_p$) and outer ($\mathbf{o}_p$) representations at the node that covers constituent p. They are vectorial representations of p's content and context, respectively.

ter).

To tackle this problem, we propose a new architecture: the Inside-Outside Recursive Neural Network (IORNN) in which information can flow not only bottom-up but also top-down, inward and outward. The crucial innovation in our architecture is that every node in a hierarchical structure is associated with two vectors: one vector, the *inner representation*, representing the content under that node, and another vector, the *outer representation*, representing its context (see Figure 1). Inner representations can be computed bottom-up; outer representations, in turn, can be computed top-down. This allows information to flow in any direction, depending on the application, and makes the IORNN a natural tool for estimating probabilities in tree-based generative models.

We demonstrate the use of the IORNN by applying it to an ∞-order generative dependency model which is impractical for counting due to the problem of data sparsity. Counting, instead, is used to estimate a third-order generative model as in Sangati et al. (2009) and Hayashi et al. (2011). Our experimental results show that our new model not only achieves a seven times lower perplexity than the third-order model, but also tends to choose more accurate candidates in k-best lists. In addition, reranking with this model achieves state-of-the-art scores on the task of supervised dependency parsing.

The outline of the paper is following. Firstly, we give an introduction to Eisner's generative model in Section 2. Then, we present the third-order model using counting in Section 3, and propose the IORNN in Section 4. Finally, in Section 5 we show our experimental results.

2 Eisner's Generative Model

Eisner (1996b) proposed a generative model for dependency parsing. The generation process is top-down: starting at the ROOT, it generates left dependents and then right dependents for the ROOT. After that, it generates left dependents and right dependents for each of ROOT's dependents. The process recursively continues until there is no further dependent to generate. The whole process is captured in the following formula

$$P(T(H)) = \prod_{l=1}^{L} P\left(H_l^L | \mathcal{C}_{H_l^L}\right) P\left(T(H_l^L)\right) \times \prod_{r=1}^{R} P\left(H_r^R | \mathcal{C}_{H_r^R}\right) P\left(T(H_r^R)\right) \quad (1)$$

where H is the current head, $T(N)$ is the fragment of the dependency parse rooted in N, and $\mathcal{C}_N$ is the context in which N is generated. H^L, H^R are respectively H's left dependents and right dependents, plus EOC (End-Of-Children), a special token to indicate that there are no more dependents to generate. Thus, $P(T(ROOT))$ is the probability of generating the entire dependency structure T. We refer to $\langle H_l^L, \mathcal{C}_{H_l^L}\rangle$, $\langle H_r^R, \mathcal{C}_{H_r^R}\rangle$ as "events", and $\langle \mathcal{C}_{H_l^L}\rangle$, $\langle \mathcal{C}_{H_r^R}\rangle$ as "conditioning contexts".

In order to avoid the problem of data sparsity, the conditioning context in which a dependent D is generated should capture only part of the fragment generated so far. Based on the amount of information that contexts hold, we can define the *order* of a generative model (see Hayashi et al. (2011, Table 3) for examples)

- first-order: $\mathcal{C}_D^1$ contains the head H,
- second-order: $\mathcal{C}_D^2$ contains H and the just-generated sibling S,
- third-order: $\mathcal{C}_D^3$ contains H, S, the sibling S' before S (tri-sibling); or H, S and the grand-head G (the head of H) (grandsibling) (the fragment enclosed in the blue doted contour in Figure 2),
- ∞-order: $\mathcal{C}_D^\infty$ contains all of D's ancestors, theirs siblings, and its generated siblings (the fragment enclosed in the red dashed contour in Figure 2).

In the original models (Eisner, 1996a), each dependent D is a 4-tuple $\langle dist, w, c, t\rangle$

- $dist(H, D)$ the distance between D and its head H, represented as one of the four ranges 1, 2, 3-6, 7-∞.

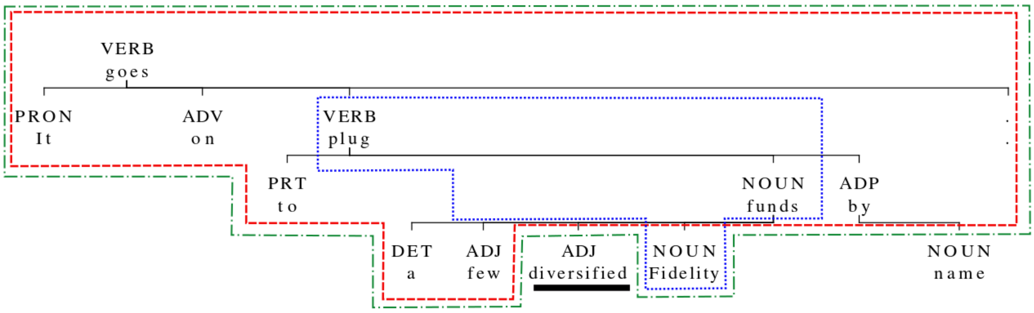

Figure 2: Example of different orders of context of "diversified". The blue dotted shape corresponds to the third-order outward context, while the red dashed shape corresponds to the ∞-order left-to-right context. The green dot-dashed shape corresponds to the context to compute the outer representation.

- $\underline{w}ord(D)$ the lowercase version of the word of D,
- $\underline{c}ap(D)$ the capitalisation feature of the word of D (all letters are lowercase, all letters are uppercase, the first letter is uppercase, the first letter is lowercase),
- $\underline{t}ag(D)$ the POS-tag of D,

Here, to make the dependency complete, $\underline{d}eprel(D)$, the dependency relation of D (e.g., SBJ, DEP), is also taken into account.

3 Third-order Model with Counting

The third-order model we suggest is similar to the grandsibling model proposed by Sangati et al. (2009) and Hayashi et al. (2011). It defines the probability of generating a dependent $D = \langle dist, d, w, c, t \rangle$ as the product of the distance-based probability and the probabilities of generating each of its components (d, t, w, c, denoting dependency relation, POS-tag, word and capitalisation feature, respectively). Each of these probabilities is smoothed using back-off according to the given reduction lists (as explained below).

$$
\begin{aligned}
& P(D|\mathcal{C}_D) \\
& = P(dist(H,D), dwct(D)|H,S,G,dir) \\
& = P(d(D)|H,S,G,dir)
\end{aligned}
$$

$$
\text{reduction list: } \begin{array}{|l|}\hline tw(H),tw(S),tw(G),dir \\ \hline tw(H),tw(S),t(G),dir \\ \hline \left\{ \begin{array}{l} tw(H),t(S),t(G),dir \\ t(H),tw(S),t(G),dir \end{array}\right. \\ \hline t(H),t(S),t(G),dir \\ \hline \end{array}
$$

$$
\times\, P(t(D)|d(D),H,S,G,dir)
$$

$$
\text{reduction list: } \begin{array}{|l|}\hline d(D),dtw(H),t(S),dir \\ \hline d(D),d(H),t(S),dir \\ \hline d(D),d(D),dir \\ \hline \end{array}
$$

$$
\times\, P(w(D)|dt(D),H,S,G,dir)
$$

$$
\text{reduction list: } \begin{array}{|l|}\hline dtw(H),t(S),dir \\ \hline dt(H),t(S),dir \\ \hline \end{array}
$$

$$
\times\, P(c(D)|dtw(D),H,S,G,dir)
$$

$$
\text{reduction list: } \begin{array}{|l|}\hline tw(D),d(H),dir \\ \hline tw(D),dir \\ \hline \end{array}
$$

$$
\times\, P(dist(H,D)|dtwc(D),H,S,G,dir) \tag{2}
$$

$$
\text{reduction list: } \begin{array}{|l|}\hline dtw(D),dt(H),t(S),dir \\ \hline dt(D),dt(H),t(S),dir \\ \hline \end{array}
$$

The reason for generating the dependency relation first is based on the similarity between relation/dependent and role/filler: we generate a role and then choose a filler for that role.

Back-off The back-off parameters are identical to Eisner (1996b). To estimate the probability $P(A|context)$ given a reduction list $L = (l_1, l_2, ..., l_n)$ of $context$, let

$$
p_i = \begin{cases} \frac{count(A,l_i)+0.005}{count(l_i)+0.5} & \text{if } i = n \\ \frac{count(A,l_i)+3p_{i+1}}{count(l_i)+3} & \text{otherwise} \end{cases}
$$

then $P(A|context) = p_1$.

4 The Inside-Outside Recursive Neural Network

In this section, we first describe the Recursive Neural Network architecture of Socher et al. (2010) and then propose an extension we call the Inside-Outside Recursive Neural Network (IORNN). The IORNN is a general architecture for trees, which works with tree-based generative models including those employed by Eisner (1996b) and Collins (2003). We then explain how to apply the IORNN to the ∞-order model. Note that for the present paper we are only concerned with the problem of computing the probability of

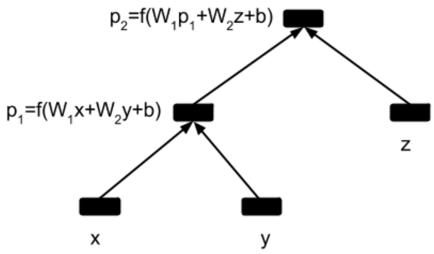

Figure 3: Recursive Neural Network (RNN).

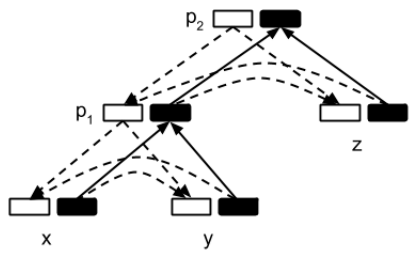

Figure 4: Inside-Outside Recursive Neural Network (IORNN). Black rectangles correspond to inner representations, white rectangles correspond to outer representations.

a tree; we assume an independently given parser is available to assign a syntactic structure, or multiple candidate structures, to an input string.

4.1 Recursive Neural Network

The architecture we propose can best be understood as an extension of the Recursive Neural Networks (RNNs) proposed by Socher et al. (2010), that we mentioned above. In order to see how an RNN works, consider the following example. Assume that there is a constituent with parse tree $(p_2\ (p_1\ x\ y)\ z)$ (Figure 3), and that $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbb{R}^n$ are the (inner) representations of the three words x, y and z, respectively. We use a neural network which consists of a weight matrix $\mathbf{W}_1 \in \mathbb{R}^{n \times n}$ for left children and a weight matrix $\mathbf{W}_2 \in \mathbb{R}^{n \times n}$ for right children to compute the vector for a parent node in a bottom up manner. Thus, we compute p_1 as follows

$$\mathbf{p}_1 = f(\mathbf{W}_1\mathbf{x} + \mathbf{W}_2\mathbf{y} + \mathbf{b})$$

where $\mathbf{b}$ is a bias vector and f is an activation function (e.g., *tanh* or *logistic*). Having computed p_1, we can then move one level up in the hierarchy and compute p_2:

$$\mathbf{p}_2 = f(\mathbf{W}_1\mathbf{p}_1 + \mathbf{W}_2\mathbf{z} + \mathbf{b})$$

This process is continued until we reach the root node. The RNN thus computes a single vector for each node p in the tree, representing the *content* under that node. It has in common with logical semantics that representations for compounds (here *xyz*) are computed by recursively applying a composition function to meaning representations of the parts. It is difficult to characterise the expressivity of the resulting system in logical terms, but recent work suggests it is surprisingly powerful (e.g., Kanerva (2009)).

4.2 IORNN

We extend the RNN-architecture by adding a second vector to each node, representing the *context* of the node, shown as white rectangles in figure 4. The job of this second vector, the outer representation, is to summarize all information about the context of node p so that we can either predict its content (i.e., predict an inner representation), or pass on this information to the daughters of p (i.e., compute outer representations of these daughters). Outer representations thus allow information to flow top-down.

We explain the operation of the resulting *Inside-Outside Recursive Neural Network* in terms of the same example parse tree $(p_2\ (p_1\ x\ y)\ z)$ (see Figure 4). Each node u in the syntactic tree carries two vectors $\mathbf{o}_u$ and $\mathbf{i}_u$, the outer representation and inner representation of the constituent that is covered by the node.

Computing inner representations Inner representations are computed from the bottom up. We assume for every word w an inner representation $\mathbf{i}_w \in \mathbb{R}^n$. The inner representation of a non-terminal node, say p_1, is given by

$$\mathbf{i}_{p_1} = f(\mathbf{W}_1^i\mathbf{i}_x + \mathbf{W}_2^i\mathbf{i}_y + \mathbf{b}^i)$$

where $\mathbf{W}_1^i, \mathbf{W}_2^i$ are $n \times n$ real matrices, $\mathbf{b}^i$ is a bias vector, and f is an activation function, e.g. $tanh$. (This is the same as the computation of non-terminal vectors in the RNNs.) The inner representation of a parent node is thus a function of the inner representations of its children.

Computing outer representations Outer representations are computed from the top down. For a node which is not the root, say p_1, the outer representation is given by

$$\mathbf{o}_{p_1} = g(\mathbf{W}_1^o \mathbf{o}_{p_2} + \mathbf{W}_2^o \mathbf{i}_z + \mathbf{b}^o)$$

where $\mathbf{W}_1^o, \mathbf{W}_2^o$ are $n \times n$ real matrices, $\mathbf{b}^o$ is a bias vector, and g is an activation function. The outer representation of a node is thus a function of the outer representation of its parent and the inner representation of its sisters.

If there is information about the external context of the utterance that is being processed, this information determines the outer representation of the root node $\mathbf{o}_{root}$. In our first experiments reported here, no such information was assumed to be available. In this case, a random value $\mathbf{o}_\emptyset$ is chosen at initialisation and assigned to the root nodes of all utterances; this value is then adjusted by the learning process discussed below.

Training Training the IORNN is to minimise an objective function $J(\theta)$ which depends on the purpose of usage where θ is the set of parameters. To do so, we compute the gradient $\partial J / \partial \theta$ and apply the gradient descent method. The gradient is effectively computed thanks to back-propagation through structure (Goller and Küchler, 1996). Following Socher et al. (2013), we use AdaGrad (Duchi et al., 2011) to update the parameters.

4.3 The ∞-order Model with IORNN

The RNN and IORNN are defined for context-free trees. To apply the IORNN architecture to dependency parses we need to adapt the definitions somewhat. In particular, in the generative dependency model, every step in the generative story involves the decision to generate a specific word while the span of the subtree that this word will dominate only becomes clear when all dependents are generated. We therefore introduce *partial outer representation* as a representation of the current context of a word in the generative process, and compute the final outer representation only when all its siblings have been generated.

Consider an example of head h and its dependents x, y (we ignore directions for simplicity) in Figure 5. Assume that we are in the state in the generative process where the generation of h is complete, i.e. we know its inner and outer representations $\mathbf{i}_h$ and $\mathbf{o}_h$. Now, when generating h's first dependent x (see Figure 5-a), we first compute x's *partial* outer representation (representing its context at this stage in the process), which is a function of the outer representation of the head (representing the head's context) and the inner representation of the head (representing the content of the head word):

$$\bar{\mathbf{o}}_1 = f(\mathbf{W}_{hi}\mathbf{i}_h + \mathbf{W}_{ho}\mathbf{o}_h + \mathbf{b}_o)$$

where $\mathbf{W}_{hi}, \mathbf{W}_{ho}$ are $n \times n$ real matrices, $\mathbf{b}_o$ is a bias vector, f is an activation function.

With the context of the first dependent determined, we can proceed and generate its content. For this purpose, we assume a separate weight matrix $\mathbf{W}$, trained (as explained below) to predict a specific word given a (partial) outer representation. To compute a proper probability for word x, we use the softmax function:

$$softmax(x, \bar{\mathbf{o}}_1) = \frac{e^{u(x,\bar{\mathbf{o}}_1)}}{\sum_{w \in V} e^{u(w,\bar{\mathbf{o}}_1)}}$$

where $\left[u(w_1, \bar{\mathbf{o}}_1), ..., u(w_{|V|}, \bar{\mathbf{o}}_1)\right]^T = \mathbf{W}\bar{\mathbf{o}}_1 + \mathbf{b}$ and V is the set of all possible dependents.

Note that since $\mathbf{o}_h$, the outer representation of h, represents the entire dependency structure generated up to that point, $\bar{\mathbf{o}}_1$ is a vectorial representation of the ∞-order context generating the first dependent (like the fragment enclosed in the red dashed contour in Figure 2). The softmax function thus estimates the probability $P(D = x | \mathcal{C}_D^\infty)$.

The next step, now that x is generated, is to compute the partial outer representation for the second dependent (see Figure 5-b)

$$\bar{\mathbf{o}}_2 = f(\mathbf{W}_{hi}\mathbf{i}_h + \mathbf{W}_{ho}\mathbf{o}_h + \mathbf{W}_{dr(x)}\mathbf{i}_x + \mathbf{b}_o)$$

where $\mathbf{W}_{dr(x)}$ is a $n \times n$ real matrix specific for the dependency relation of x with h.

Next y is generated (using the softmax function above), and the partial outer representation for the third dependent (see Figure 5-c) is computed:

$$\bar{\mathbf{o}}_3 = f(\mathbf{W}_{hi}\mathbf{i}_h + \mathbf{W}_{ho}\mathbf{o}_h + \frac{1}{2}\left(\mathbf{W}_{dr(x)}\mathbf{i}_x + \mathbf{W}_{dr(y)}\mathbf{i}_y\right) + \mathbf{b}_o)$$

Since the third dependent is the End-of-Children symbol (EOC), the process of generating dependents for h stops. We can then return to x and y to replace the partial outer representations with complete outer representations[1] (see

[1]According to the IORNN architecture, to compute the outer representation of a node, the inner representations of the whole fragments rooting at its sisters must be taken into account. Here, we replace the inner representation of a fragment by the inner representation of its root since the meaning of a phrase is often dominated by the meaning of its head.

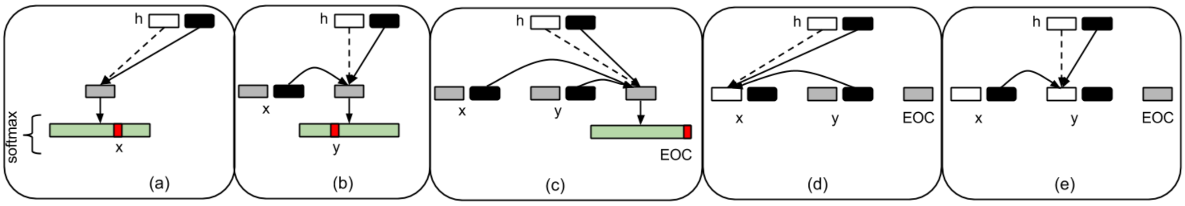

Figure 5: Example of applying IORNN to dependency parsing. Black, grey, white boxes are respectively inner, partial outer, and outer representations. For simplicity, only links related to the current computation are drawn (see text).

Figure 5-d,e):

$$\mathbf{o}_x = f(\mathbf{W}_{hi}\mathbf{i}_h + \mathbf{W}_{ho}\mathbf{o}_h + \mathbf{W}_{dr(y)}\mathbf{i}_y + \mathbf{b}_o)$$
$$\mathbf{o}_y = f(\mathbf{W}_{hi}\mathbf{i}_h + \mathbf{W}_{ho}\mathbf{o}_h + \mathbf{W}_{dr(x)}\mathbf{i}_x + \mathbf{b}_o)$$

In general, if u is the first dependent of h then

$$\bar{\mathbf{o}}_u = f(\mathbf{W}_{hi}\mathbf{i}_h + \mathbf{W}_{ho}\mathbf{o}_h + \mathbf{b}_o)$$

otherwise

$$\bar{\mathbf{o}}_u = f(\mathbf{W}_{hi}\mathbf{i}_h + \mathbf{W}_{ho}\mathbf{o}_h + \mathbf{b}_o + \frac{1}{|\bar{\mathcal{S}}(u)|}\sum_{v \in \bar{\mathcal{S}}(u)} \mathbf{W}_{dr(v)}\mathbf{i}_v)$$

where $\bar{\mathcal{S}}(u)$ is the set of u's sisters generated before it. And, if u is the only dependent of h (ignoring EOC) then

$$\mathbf{o}_u = f(\mathbf{W}_{hi}\mathbf{i}_h + \mathbf{W}_{ho}\mathbf{o}_h + \mathbf{b}_o)$$

otherwise

$$\mathbf{o}_u = f(\mathbf{W}_{hi}\mathbf{i}_h + \mathbf{W}_{ho}\mathbf{o}_h + \mathbf{b}_o + \frac{1}{|\mathcal{S}(u)|}\sum_{v \in \mathcal{S}(u)} \mathbf{W}_{dr(v)}\mathbf{i}_v)$$

where $\mathcal{S}(u)$ is the set of u's sisters.

We then continue this process to generate dependents for x and y until the process stops.

Inner Representations In the calculation of the probability of generating a word, described above, we assumed inner representations of all possible words to be given. These are, in fact, themselves a function of vector representations for the words (in our case, the word vectors are initially borrowed from Collobert et al. (2011)), the POS-tags and capitalisation features. That is, the inner representation at a node h is given by:

$$\mathbf{i}_h = f\left(\mathbf{W}_w\mathbf{w}_h + \mathbf{W}_p\mathbf{p}_h + \mathbf{W}_c\mathbf{c}_h\right)$$

where $\mathbf{W}_w \in R^{n \times d_w}$, $\mathbf{W}_p \in R^{n \times d_p}$, $\mathbf{W}_c \in R^{n \times d_c}$, $\mathbf{w}_h$ is the word vector of h, and $\mathbf{p}_h, \mathbf{c}_h$ are respectively binary vectors representing the POS-tag and capitalisation feature of h.

Training Training this IORNN is to minimise the following objective function which is the regularised cross-entropy

$$J(\theta) = -\frac{1}{m}\sum_{T \in \mathcal{D}}\sum_{w \in T}\log(P(w|\bar{\mathbf{o}}_w)) + \frac{1}{2}\left(\lambda_W\|\theta_W\|^2 + \lambda_L\|\theta_L\|^2\right)$$

where $\mathcal{D}$ is the set of training dependency parses, m is the number of dependents; θ_W, θ_L are the weight matrix set and the word embeddings ($\theta = (\theta_W, \theta_L)$); λ_W, λ_L are regularisation hyper-parameters.

Implementation We decompose a dependent D into four features: dependency relation, POS-tag, lowercase version of word, capitalisation feature of word. We then factorise $P(D|\mathcal{C}^{\infty}_D)$ similarly to Section 3, where each component is estimated by a softmax function.

5 Experiments

In our experiments, we convert the Penn Treebank to dependencies using the Universal dependency annotation (McDonald et al., 2013)[2]; this yields a dependency tree corpus we label PTB-U. In order to compare with other systems, we also experiment with an alternative conversion using the head rules of Yamada and Matsumoto (2003)[3]; this yields a dependency tree corpus we label PTB-YM. Sections 2-21 are used for training, section 22 for development, and section 23 for testing. For the PTB-U, the gold POS-tags are used. For the PTB-YM, the development and test sets are tagged by the Stanford POS-tagger[4] trained on the whole

[2] https://code.google.com/p/uni-dep-tb/

[3] http://stp.lingfil.uu.se/~nivre/research/Penn2Malt.html

[4] http://nlp.stanford.edu/software/tagger.shtml

	Perplexity
3rd-order model	1736.73
∞-order model	**236.58**

Table 1: Perplexities of the two models on PTB-U-22.

training data, whereas 10-way jackknifing is used to generate tags for the training set.

The vocabulary for both models, the third-order model and the ∞-order model, is taken as a list of words occurring more than two times in the training data. All other words are labelled 'UNKNOWN' and every digit is replaced by '0'. For the IORNN used by the ∞-order model, we set $n = 200$, and define f as the $tanh$ activation function. We initialise it with the 50-dim word embeddings from Collobert et al. (2011) and train it with the learning rate 0.1, $\lambda_W = 10^{-4}, \lambda_L = 10^{-10}$.

5.1 Perplexity

We firstly evaluate the two models on PTB-U-22 using the perplexity-per-word metric

$$ppl(P) = 2^{-\frac{1}{N}\sum_{T \in \mathcal{D}} \log_2 P(T)}$$

where $\mathcal{D}$ is a set of dependency parses, N is the total number of words. It is worth noting that, the better P estimates the true distribution P^* of $\mathcal{D}$, the lower its perplexity is. Because Eisner's model with the $dist(H, D)$ feature (Equation 2) is leaky (the model allocates some probability to events that can never legally arise), this feature is discarded (only in this experiment).

Table 1 shows results. The perplexity of the third-order model is more than seven times higher than the ∞-order model. This reflects the fact that data sparsity is more problematic for counting than for the IORNN.

To investigate why the perplexity of the third-order model is so high, we compute the percentages of events extracted from the development set appearing more than twice in the training set. Events are grouped according to the reduction lists in Equation 2 (see Table 2). We can see that reductions at level 0 (the finest) for dependency relations and words seriously suffer from data sparsity: more than half of the events occur less than three times, or not at all, in the training data. We thus conclude that counting-based models heavily rely on carefully designed reduction lists for back-off.

back-off level	d	t	w	c
0	47.4	61.6	43.7	87.7
1	69.8	98.4	77.8	97.3
2	76.0, 89.5	99.7		
3	97.9			
total	76.1	86.6	60.7	92.5

Table 2: Percentages of events extracted from PTB-U-22 appearing more than twice in the training set. Events are grouped according to the reduction lists in Equation 2. d, t, w, c stand for dependency relation, POS-tag, word, and capitalisation feature.

5.2 Reranking

In the second experiment, we evaluate the two models in the reranking framework proposed by Sangati et al. (2009) on PTB-U. We used the MSTParser (with the 2nd-order feature mode) (McDonald et al., 2005) to generate k-best lists. Two evaluation metrics are labelled attachment score (LAS) and unlabelled attachment score (UAS), *including* punctuation.

Rerankers Given $\mathcal{D}(S)$, a k-best list of parses of a sentence S, we define the *generative* reranker

$$T^* = \arg\max_{T \in \mathcal{D}(S)} P(T(ROOT))$$

which is identical to Sangati et al. (2009). Moreover, as in many mixture-model-based approaches, we define the *mixture* reranker as a combination of the generative model and the MST discriminative model (Hayashi et al., 2011)

$$T^* = \arg\max_{T \in \mathcal{D}(S)} \alpha \log P(T(ROOT)) + (1-\alpha)s(S,T)$$

where $s(S, T)$ is the score given by the MSTParser, and $\alpha \in [0, 1]$.

Results Figure 6 shows UASs of the generative reranker on the development set. The MSTParser achieves 92.32% and the Oracle achieve 96.23% when $k = 10$. With the third-order model, the generative reranker performs better than the MSTParser when $k < 6$ and the maximum improvement is 0.17%. Meanwhile, with the ∞-order model, the generative reranker always gains higher UASs than the MSTParser, and with $k = 6$, the difference reaches 0.7%. Figure 7 shows UASs of the mixture reranker on the same set. α is optimised by searching with the step-size 0.005. Unsurprisingly, we observe improvements over the

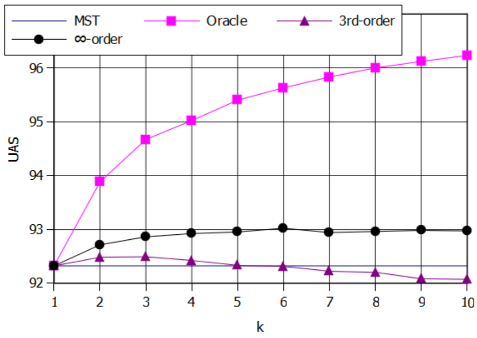

Figure 6: Performance of the generative reranker on PTB-U-22.

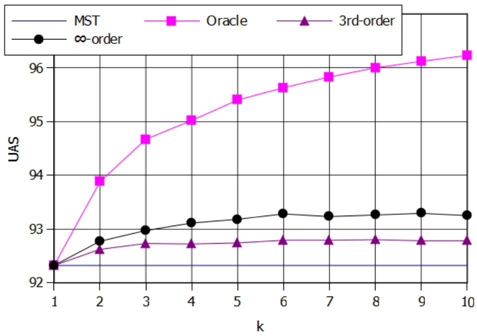

Figure 7: Performance of the mixture reranker on PTB-U-22. For each k, α was optimized with the step-size 0.005.

	LAS	UAS
MSTParser	89.97	91.99
Oracle ($k = 10$)	93.73	96.24
Generative reranker with		
3rd-order ($k = 3$)	90.27 (+0.30)	92.27 (+0.28)
∞-order ($k = 6$)	90.76 (+0.79)	92.83 (+0.84)
Mixture reranker with		
3rd-order ($k = 6$)	90.62 (+0.65)	92.62 (+0.63)
∞-order ($k = 9$)	**91.02 (+1.05)**	**93.08 (+1.09)**

Table 3: Comparison based on reranking on PTB-U-23. The numbers in the brackets are improvements over the MSTParser.

generative reranker as the mixture reranker can combine the advantages of the two models.

Table 3 shows scores of the two rerankers on the test set with the parameters tuned on the development set. Both the rerankers, either using third-order or ∞-order models, outperform the MSTParser. The fact that both gain higher improvements with the ∞-order model suggests that the IORNN surpasses counting.

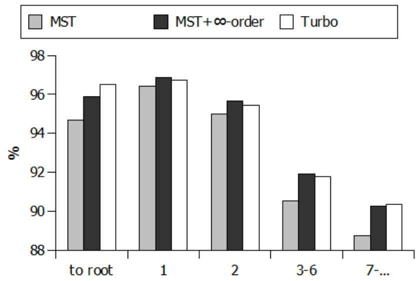

Figure 9: F1-scores of binned HEAD distance (PTB-U-23).

5.3 Comparison with other systems

We first compare the mixture reranker using the ∞-order model against the state-of-the-art dependency parser TurboParser (with the full mode) (Martins et al., 2013) on PTB-U-23. Table 4 shows LASs and UASs. When taking labels into account, the TurboParser outperforms the reranker. But without counting labels, the two systems perform comparably, and when ignoring punctuation the reranker even outperforms the TurboParser. This pattern is also observed when the exact match metrics are used (see Table 4). This is due to the fact that the TurboParser performs significantly better than the MSTParser, which generates k-best lists for the reranker, in labelling: the former achieves 96.03% label accuracy score whereas the latter achieves 94.92%.

One remarkable point is that reranking with the ∞-order model helps to improve the exact match scores 4% - 6.4% (see Table 4). Because the exact match scores correlate with the ability to handle global structures, we conclude that the IORNN is able to capture ∞-order contexts. Figure 8 shows distributions of correct-head accuracy over CPOS-tags and Figure 9 shows F1-scores of binned HEAD distance. Reranking with the ∞-order model is clearly helpful for all CPOS-tags and dependent-to-head distances, except a minor decrease on PRT.

We compare the reranker against other systems on PTB-YM-23 using the UAS metric ignoring punctuation (as the standard evaluation for English) (see Table 5). Our system performs slightly better than many state-of-the-art systems such as Martins et al. (2013) (a.k.a. TurboParser), Zhang and McDonald (2012), Koo and Collins (2010). It outperforms Hayashi et al. (2011) which is a reranker using a combination of third-order generative models with a variational model learnt

	LAS (w/o punc)	UAS (w/o punc)	LEM (w/o punc)	UEM (w/o punc)
MSTParser	89.97 (90.54)	91.99 (92.82)	32.37 (34.19)	42.80 (45.24)
w. ∞-order ($k = 9$)	91.02 (91.51)	**93.08** (**93.84**)	37.58 (39.16)	**49.17** (**51.53**)
TurboParser	**91.56** (**92.02**)	93.05 (93.70)	**40.65** (**41.72**)	48.05 (49.83)

Table 4: Comparison with the TurboParser on PTB-U-23. LEM and UEM are respectively the labelled exact match score and unlabelled exact match score metrics. The numbers in brackets are scores computed excluding punctuation.

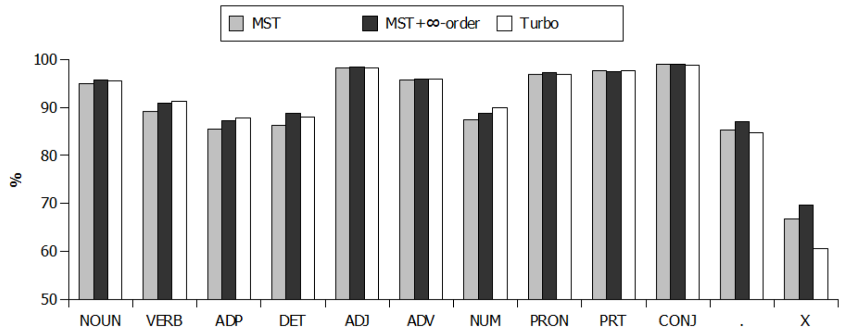

Figure 8: Distributions of correct-head accuracy over CPOS-tags (PTB-U-23).

System	UAS
Huang and Sagae (2010)	92.1
Koo and Collins (2010)	93.04
Zhang and McDonald (2012)	93.06
Martins et al. (2013)	93.07
Bohnet and Kuhn (2012)	93.39
Reranking	
Hayashi et al. (2011)	92.89
Hayashi et al. (2013)	93.12
MST+ ∞-order ($k = 12$)	**93.12**

Table 5: Comparison with other systems on PTB-YM-23 (excluding punctuation).

on the fly; performs equally with Hayashi et al. (2013) which is a discriminative reranker using the stacked technique; and slightly worse than Bohnet and Kuhn (2012), who develop a hybrid transition-based and graphical-based approach.

6 Related Work

Using neural networks to process trees was first proposed by Pollack (1990) in the Recursive Autoassociative Memory model which was used for unsupervised learning. Socher et al. (2010) later introduced the Recursive Neural Network architecture for supervised learning tasks such as syntactic parsing and sentiment analysis (Socher et al., 2013). Our IORNN is an extension of the RNN: the former can process trees not only bottom-up like the latter but also top-down.

Elman (1990) invented the simple recurrent neural network (SRNN) architecture which is capable of capturing very long histories. Mikolov et al. (2010) then applied it to language modelling and gained state-of-the-art results, outperforming the the standard n-gram techniques such as Kneser-Ney smoothed 5-gram. Our IORNN architecture for dependency parsing bears a resemblance to the SRNN in the sense that it can also capture long 'histories' in context representations (i.e., outer representations in our terminology). Moreover, the IORNN can be seen as a generalization of the SRNN since a left-branching tree is equivalent to a chain and vice versa.

The idea of letting parsing decisions depend on arbitrarily long derivation histories is also explored in Borensztajn and Zuidema (2011) and is related to parsing frameworks that allow arbitrarily large elementary trees (e.g., Scha (1990), O'Donnell et al. (2009), Sangati and Zuidema (2011), and van Cranenburgh and Bod (2013)).

Titov and Henderson (2007) were the first proposing to use deep networks for dependency parsing. They introduced a transition-based generative dependency model using incremental sigmoid belief networks and applied beam pruning for searching best trees. Differing from them, our work uses the IORNN architecture to rescore k-best candidates generated by an independent graph-based parser, namely the MSTParser.

Reranking k-best lists was introduced by Collins and Koo (2005) and Charniak and Johnson (2005). Their rerankers are discriminative and for constituent parsing. Sangati et al. (2009) proposed to use a third-order generative model for reranking k-best lists of dependency parses. Hayashi et al. (2011) then followed this idea but combined generative models with a variational model learnt on the fly to rerank forests. In this paper, we also followed Sangati et al. (2009)'s idea but used an ∞-order generative model, which has never been used before.

7 Conclusion

In this paper, we proposed a new neural network architecture, the Inside-Outside Recursive Neural Network, that can process trees both bottom-up and top-down. The key idea is to extend the RNN such that every node in the tree has two vectors associated with it: an inner representation for its content, and an outer representation for its context. Inner and outer representations of any constituent can be computed simultaneously and interact with each other. This way, information can flow top-down, bottom-up, inward and outward. Thanks to this property, by applying the IORNN to dependency parses, we have shown that using an ∞-order generative model for dependency parsing, which has never been done before, is practical.

Our experimental results on the English section of the Universal Dependency Treebanks show that the ∞-order generative model approximates the true dependency distribution better than the traditional third-order model using counting, and tends to choose more accurate parses in k-best lists. In addition, reranking with this model even outperforms the state-of-the-art TurboParser on unlabelled score metrics.

Our source code is available at: `github.com/lephong/iornn-depparse`.

Acknowledgments

We thank Remko Scha and three anonymous reviewers for helpful comments.

References

Marco Baroni, Georgiana Dinu, and Germán Kruszewski. 2014. Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, volume 1.

Bernd Bohnet and Jonas Kuhn. 2012. The best of both worlds: a graph-based completion model for transition-based parsers. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 77–87. Association for Computational Linguistics.

Gideon Borensztajn and Willem Zuidema. 2011. Episodic grammar: a computational model of the interaction between episodic and semantic memory in language processing. In *Proceedings of the 33d Annual Conference of the Cognitive Science Society (CogSci'11)*, pages 507–512. Lawrence Erlbaum Associates.

Eugene Charniak and Mark Johnson. 2005. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 173–180.

Michael Collins and Terry Koo. 2005. Discriminative reranking for natural language parsing. *Computational Linguistics*, 31(1):25–66.

Michael Collins. 2003. Head-driven statistical models for natural language parsing. *Computational linguistics*, 29(4):589–637.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *The Journal of Machine Learning Research*, 12:2493–2537.

John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, pages 2121–2159.

Jason M. Eisner. 1996a. An empirical comparison of probability models for dependency grammar. Technical report, University of Pennsylvania Institute for Research in Cognitive Science.

Jason M Eisner. 1996b. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th conference on Computational linguistics-Volume 1*, pages 340–345. Association for Computational Linguistics.

Jeffrey L. Elman. 1990. Finding structure in time. *Cognitive science*, 14(2):179–211.

Christoph Goller and Andreas Küchler. 1996. Learning task-dependent distributed representations by backpropagation through structure. In *International Conference on Neural Networks*. IEEE.

Katsuhiko Hayashi, Taro Watanabe, Masayuki Asahara, and Yuji Matsumoto. 2011. Third-order variational reranking on packed-shared dependency forests. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1479–1488. Association for Computational Linguistics.

Katsuhiko Hayashi, Shuhei Kondo, and Yuji Matsumoto. 2013. Efficient stacked dependency parsing by forest reranking. *Transactions of the Association for Computational Linguistics*, 1(1):139–150.

Liang Huang and Kenji Sagae. 2010. Dynamic programming for linear-time incremental parsing. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1077–1086. Association for Computational Linguistics.

Pentti Kanerva. 2009. Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. *Cognitive Computation*, 1(2):139–159.

Terry Koo and Michael Collins. 2010. Efficient third-order dependency parsers. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1–11. Association for Computational Linguistics.

Phong Le, Willem Zuidema, and Remko Scha. 2013. Learning from errors: Using vector-based compositional semantics for parse reranking. In *Proceedings Workshop on Continuous Vector Space Models and their Compositionality (at ACL 2013)*. Association for Computational Linguistics.

Andre Martins, Miguel Almeida, and Noah A. Smith. 2013. Turning on the turbo: Fast third-order non-projective turbo parsers. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 617–622, Sofia, Bulgaria, August. Association for Computational Linguistics.

Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005. Online large-margin training of dependency parsers. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 91–98. Association for Computational Linguistics.

Ryan McDonald, Joakim Nivre, Yvonne Quirmbach-Brundage, Yoav Goldberg, Dipanjan Das, Kuzman Ganchev, Keith Hall, Slav Petrov, Hao Zhang, Oscar Täckström, et al. 2013. Universal dependency annotation for multilingual parsing. *Proceedings of ACL, Sofia, Bulgaria*.

Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *INTERSPEECH*, pages 1045–1048.

Timothy J O'Donnell, Noah D Goodman, and Joshua B Tenenbaum. 2009. Fragment grammar: Exploring reuse in hierarchical generative processes. Technical report, Technical Report MIT-CSAIL-TR-2009-013, MIT.

Jordan B. Pollack. 1990. Recursive distributed representations. *Artificial Intelligence*, 46(1):77105.

Federico Sangati and Willem Zuidema. 2011. Accurate parsing with compact tree-substitution grammars: Double-DOP. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMLNP'11)*, pages 84–95. Association for Computational Linguistics.

Federico Sangati, Willem Zuidema, and Rens Bod. 2009. A generative re-ranking model for dependency parsing. In *Proceedings of the 11th International Conference on Parsing Technologies*, pages 238–241.

Remko Scha. 1990. Taaltheorie en taaltechnologie; competence en performance. In R. de Kort and G.L.J. Leerdam, editors, *Computertoepassingen in de Neerlandistiek*, pages 7–22. LVVN, Almere, the Netherlands. English translation at http://iaaa.nl/rs/LeerdamE.html.

Richard Socher, Christopher D. Manning, and Andrew Y. Ng. 2010. Learning continuous phrase representations and syntactic parsing with recursive neural networks. In *Proceedings of the NIPS-2010 Deep Learning and Unsupervised Feature Learning Workshop*.

Richard Socher, Brody Huval, Christopher D. Manning, and Andrew Y. Ng. 2012. Semantic compositionality through recursive matrix-vector spaces. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, Seattle, Washington, USA, October.

Ivan Titov and James Henderson. 2007. A latent variable model for generative dependency parsing. In *Proceedings of the 10th International Conference on Parsing Technologies*, pages 144–155.

Andreas van Cranenburgh and Rens Bod. 2013. Discontinuous parsing with an efficient and accurate DOP model. In *Proceedings of the International Conference on Parsing Technologies (IWPT'13)*.

Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proceedings of International Conference on Parsing Technologies (IWPT)*, pages 195–206.

Hao Zhang and Ryan McDonald. 2012. Generalized higher-order dependency parsing with cube pruning. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 320–331. Association for Computational Linguistics.