Improvements to Training an RNN parser

Richard J. BILLINGSLEY *James R. CURRAN*
School of Information Technologies
University of Sydney
NSW 2006, Australia
{richbill,james}@it.usyd.edu.au

ABSTRACT
Many parsers learn sparse class distributions over trees to model natural language. Recursive Neural Networks (RNN) use much denser representations, yet can still achieve an F-score of 92.06% for right binarized sentences up to 15 words long. We examine an RNN model by comparing it with an abstract generative probabilistic model using a Deep Belief Network (DBN). The DBN provides both an upwards and downwards pointing conditional model, drawing a connection between RNN and Charniak type parsers, while analytically predicting average scoring parameters in the RNN. In addition, we apply the RNN to longer sentences and develop two methods which, while having negligible effect on short sentence parsing, are able to improve the parsing F-Score by 0.83% on longer sentences.

KEYWORDS: Parsing Recurrent Neural Network Restricted Boltzmann Machine.

1 Introduction

Fast and accurate constituent parsing is critical for many natural language processing systems (Curran et al., 2007), as it enables rich information to be extracted from text.

Charniak (2000) parsed text by learning to compute top down production probabilities. The Stanford (Klein and Manning, 2003) and Berkeley (Petrov and Klein, 2007) parsers represent a parse as a sub-class distribution on a tree, and maximize the entropy of predicting label probabilities for each node. Advanced parsers use techniques like training on unlabeled text (McClosky et al., 2006), and parse re-ranking (Charniak and Johnson, 2005) to improve their accuracy.

To achieve such accuracy requires conditional probabilities to be computed correctly far up the parse tree, and requires plenty of sub-class labels (typically over 4000) to produce a sufficiently fine grained analysis that accommodates subtle distinctions in text. The resulting transition matrices become large very quickly.

Titov and Henderson (2007) used vectors of real numbers to represent words in a history based model with a mean field representation. Such word vectors have the advantage of maintaining expressive power with much lower dimensionality, typically just 100 dimensions. Garg and Henderson (2011) extended this model to use a Restricted Boltzmann Machine (RBM, Hinton et al., 2006) representation.

Socher et al. (2010) constructed a parse tree using word vectors in a Recursive Neural Network (RNN, Bengio et al., 1994), and applied a similar unsupervised approach for sentiment analysis (Socher et al., 2011).

We focus on the RNN models of Socher, which perform well on short (up to 15 word) sentences, achieving a parse F-score of 92.06% on right binarized text, and we obtain a baseline of 83.94% by applying this to all sentences.

We develop an abstract fully generative model of parsing by considering a Deep Belief Network (DBN, Hinton et al., 2006), and by examining the mean field approximation, contrast this with the conditional RNN model to discover underlying properties of the scoring function.

Motivated by insights from the DBN model, we use an RBM to implement a better model for scoring production probabilities. While this has negligible affect on short sentences, it achieves a 0.83% gain in parsing performance on long sentences.

Noting that the RNN parser's CKY performance drops over large trees, we also develop a novel method of applying gradient methods during evaluation time that improves the parsing F-score by 0.38% for long sentences.

While demonstrating these methods over an RNN model, the gradient method can broadly be applied to a wide range of conditional tree-based models.

2 Background

The Charniak (2000) parser represents a parse probability as the product of the top down production probabilities of a parse. One intuition is that the head node represents the entire sentence, and lower nodes are the probabilities of expressing a span of the text in each possible way. This makes each parse a downwards pointing conditional graphical model, and as explained by Charniak (1997), parser performance increases as more conditional information is used in calculating these production probabilities.

The more recent Stanford (Klein and Manning, 2003) and Berkeley (Petrov and Klein, 2007) parsers use increasingly fine grained sub-class schemes to convey rich information to each parse node to increase performance.

2.1 Berkeley Parser

The Berkeley parser solves the graphical model using the inside-outside algorithm to calculate the distribution of latent subcategories from transition probabilities, β:

$$\beta(A_x \to B_y C_z) := \frac{\#\{A_x \to B_y C_z\}}{\sum_{y'z'}\{A_x \to B_{y'} C_{z'}\}} \tag{1}$$

The algorithm is an application of the Expectation Maximization algorithm (Dempster et al., 1977) for a tree based graphical model. Each tree node maintains a distribution of being in each class with probability A_x. The Berkeley parser uses about 4000 different classes, requiring the optimization algorithm to learn 16 million transition probabilities. This makes computation slow. Other parsers like the C&C parser (Clark and Curran, 2004) speed up this process by effectively limiting the transition space through operator rules that reduce the transition space.

2.2 Word Vectors

Instead of representing the sparse high dimensional distributions of word selection, classes and transitions directly, the word-vector approach attempts to encode such sparse distributions into a much shorter (say 100 dimensions) dense vectors of latent states.

This method was used in the adjacent field of language modeling (Chen and Goodman, 1996), which aims to predict the smoothed frequency of n-gram distributions. Using Neural Networks in a method similar to Principle Component Analysis (Pearson, 1901), Mnih and Hinton (2007) show a log-bilinear model having a low perplexity in predicting the last word of an n-gram. This approach effectively encodes words into same-sized word vectors which can be combined to maximally represent n-gram distributional information through a neural network.

2.3 Neural Networks

Neural Networks are the natural extension of logistic regression that can be used to transform word-vectors into sub-class distributions.

Given a dataset of inputs X and their corresponding outputs Q, a simple logistic regression predicts an output $q_x \in Q$ for each input $x \in X$ by learning a matrix W such that

$$p_x = P(q_x|x;W) = \sigma(Wx) \tag{2}$$

where σ is the sigmoid or hyperbolic tangent function. Learning consists of training W to minimize the loss error function $E = \sum_{x\in X} E_x$, commonly using the square or cross entropy loss

$$E_x = (q_x - p_x)^2 \text{ or } E_x = q_x \log(p_x) + (1-q_x)\log(1-p_x) \tag{3}$$

which can be solved through Stochastic Gradient Descent (SGD, Bottou, 2010) by subtracting from W the gradient of E_x for each training example, multiplied by a learning rate.

Stacking logistic regression layers into neural networks (Rumelhart et al., 1986) is possible by passing back the gradient of the error using the derivative chain rule, but learning may become slow if both the outputs and inputs to a logistic regression are latent. This can be overcome by pre-training with Restricted Boltzmann Machines (RBM, Hinton et al., 2006).

2.4 Neural Network Parsing

Costa et al. (2003) previously used a Neural Network for parsing. Titov and Henderson (2007) achieved successful results, with an F-score of 89.3% on sentences of up to 15 words of the WSJ dataset Marcus et al. (1993) by implementing a recurring neural network, predicting the constituents and label decisions at each step based on text features and previous decisions.

Garg and Henderson (2011) used RBM in a similar approach to dependency parsing. Here, while the prediction step stays the same, the learning method is adjusted. Instead of trying to learn $p_x = P(q|x;W)$ over the dataset $\{X,Q\}$, the goal is to learn the generative probability of $P(x,q;W)$. Garg and Henderson implements a recurrent model, with weight biases derived from previous parse decisions, achieving 89% dependency parser score on short sentences.

3 Recurrent Neural Network Parsing

Socher et al. (2010) used a Recurrent Neural Network (RNN) that represented a parse tree consisting of real-valued node vectors from which the various sub-class distributions and parse decisions were computed through logistic regression classifiers. The conditionally independent nature of the elements of each node vector allows the normally sparse sub-class distributions to be compressed into a length of just 100 dimensions.

3.1 Leaf Layer

At the lexical leaf node layer, each tokenized word in the text is represented by a word vector t_i that is supplied from a pre-generated word-to-vector table L. This table L (the Lexicon) is generated through a distributional similarity process (Collobert and Weston, 2008) using back-propagation through a series of transformations of word and feature distributions.

The text is padded with a pair of start-of-sentence and a pair end-of-sentence tokens, each pair with their own vector. A direct 300→100 feature word-to-leaf logistic regression combines the 100 dimensional word-vectors for the current word t_i, previous word t_{i-1} and next word t_{i+1}, to generate the 100 dimensional real-valued vector v_i that acts as a leaf-vectors for the parse tree, as shown on the left of Figure 1. The word-to-leaf logistic regression learns a 300x100 element matrix Y, which is composed of three 100x100 element matrices Y_1, $Y_2$and Y_3, so that the leaf vectors v_i for each word are calculated by:

$$v_i = \sigma(Y_1 t_{i-1} + Y_2 t_i + Y_3 t_{i+1}) \tag{4}$$

For example, in Figure 1, the leaf node a above the word "John" would be calculated as:

$$v_a = \sigma(Y_1 t_{[start]} + Y_2 t_{[John]} + Y_3 t_{[eats]}) \tag{5}$$

As well as predicting parse information, the leaf-vectors predict Part of Speech (POS) tags which, when given a gold standard POS tag during training, provides an additional error gradient to help learn the word-to-leaf logistic regression layer.

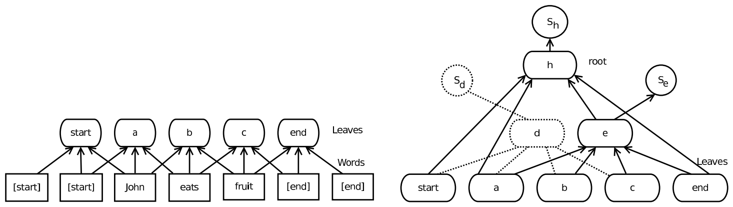

Figure 1: (left) Word to leaf network for text "John eats fruit" and (right) parse tree for parse $T_1 = a,(b,c)$ and (dotted) $T_2 = (a,b),c$ showing scoring nodes s_d, s_e and s_h. For clarity the predictions for the POS and phrase classes are not shown, however these are logistic regression classifications directly from the tree leaf nodes, and tree non-leaf nodes respectively.

3.2 Non Leaf Layer

The non-leaf node (i.e., parent node) vectors, p, of the tree are predicted using a 400→100 feature child-to-parent logistic regression layer. This logistic regression takes four lower tree node vectors as inputs. Two of these input vectors c_1 and c_2 are the vectors of p's left and right children, which may themselves be either parent or leaf node vectors. The other two input vectors are the leaf-node vectors v_1 and v_2 of the two words that directly surround the span of text represented by p. The child-to-parent logistic regression uses a 400x100 element matrix W, which is composed of four 100x100 element matrices W_1, W_2, $W_3$and W_4, so that the parent vectors p are calculated by:

$$p = \sigma(W_1 c_1 + W_2 c_2 + W_3 v_1 + W_4 v_2) \tag{6}$$

In Figure 1, the parent node h for the parse "John (eats fruit)" would be calculated as:

$$p_h = \sigma(W_1 v_a + W_2 p_e + W_3 v_{start} + W_4 v_{end}) \tag{7}$$

As well as being used to predict parent vectors higher in the parse tree, the parent vectors are used to predict node class labels (like VP or NP), and a real-valued score feature s_p which is used to predict whether a particular parent node is part of the gold-parse tree. The score is calculated using the output of a 100→1 logistic regression classifier, with parameter R:

$$s = \sigma(r) \text{ where } r = Rp \tag{8}$$

3.3 Parse Generation

Parse trees are built from the leaves up. The leaf nodes vectors are first calculated, and then parent vectors are calculated for each pair of adjacent leaf nodes. These parent vectors are then scored using the node-to-score classifier.

For subsequent layers up the parse tree, each new parent vector is computed using each combination of potential children vectors that might constitute the new parent vector. These new parent vectors are each scored using the node-to-score classifier. The computed score is added to the scores for sub-trees below, and the CKY algorithm (Kasami, 1965) is used to store and select the highest scoring overall tree.

The RNN's continuous vector model does not have a direct class representation, so there are no equivalence classes as needed for a packed chart. To overcome this, the RNN uses a beam to store a selection of high-scoring parses and their node vectors. A beam size of one was used in most experiments, as larger beam sizes gave little improvements. This is consistent with Socher et al. (2010)'s experiments and observations.

3.4 Updating the Weights

The parameters for each regression layer are updated using gradient descent, back propagating the error gradient through both the gold and the generated tree. The error function used is a hinge loss error gradient for the predicted scores s_i.

$$\frac{dE}{dr} = \begin{cases} -1 & \text{predicted, not gold} \\ 0 & \text{predicted+gold} \\ 1 & \text{gold, not predicted} \end{cases} \tag{9}$$

For gold trees, the gradient is 0 if the computed parse includes the gold node, and 1 if it does not. This is added to the gradients for the logistic regression POS tagger and node category tagger. For computed trees, the error for the predicted score s is 0 if the node is included in the gold parse, and -1 if it is not.

The errors for each tree are propagated down through each tree to the leaves, to update the word-to-leaf classifiers and eventually update the word vectors in the lexicon L. Note the gradient is calculated on r, not s (i.e. before the sigmoid function is applied), which has the effect of making it an entropy based gradient.

4 Generative Model

The RNN model is an upwards pointing conditional model in which the parents are conditioned on the children. This ignores aspects of a generative nature. By contrast, the intuition of Charniak shows a parse can be generated top down through a conditional generative process from a root node. To bridge these two models, we develop a novel expressive generative model (using a Deep Belief Network) that is conditional both up and down a tree, and examine the similarity of the equations with those of an RNN.

We are also motivated towards using a generative top-down conditional model because of the intuition that the speaking and writing process starts with ideas, which are reformulated into phrases and expressed in words. This process supports the notion that words are conditionally chosen based on ideas which would be found higher in the tree. A counter argument might exist that for the listener, the ideas heard are selected conditionally on the words of the text, however, the words and ideas were chosen by the speaker, so this is a weaker argument.

This bidirectional approach does not require these arguments to be mutually exclusive, as learning to speak and learning to listen might be shared aspects of one bidirectionally conditional model, transforming ideas into words, and back again.

4.1 Hypothetical Model

First we consider a (completely impractical) hypothetical model of parsing that follows the top-down approach, so we can later constrain it and compare it with the RNN model.

In this hypothetical model, the structure and meaning of any sentence can be represented by a 100 dimensional root vector. This root vector generates two child vectors, one representing a span of text on the left, and one representing a span on the right, and each of these child vectors continue generating their own two children vectors until some stopping process makes leaf vectors that generate words.

In this way, the process implements a top down generative conditional probability model (Charniak, 2000) in which the children are conditionally dependent on their parents.

To parse of a piece of text, samples from every possible root node vector generate every possible tree and their resulting texts. These trees and latent variables are then selected only where the generated text equals the supplied text. From this sample, the most likely parse is determined to be the most likely tree that generated the text.

The parameters of this hypothetical model would be tuned to create an optimum distribution at the root node, and an optimum conditional child generation distribution and stopping process to best model the parses of the text.

4.1.1 Constrained Model

We take this hypothetical model, and explore whether applying the constraints found in a Deep Belief Network (DBN, Hinton et al., 2006) would severely limit the model's expressive power.

A DBN consists of an RBM which generates a stationary distribution vector of binary values that acts as the complementary prior (CP, Hinton et al., 2006) for a downwards pointing directed acyclic graph (DAG) of binary vectors, each conditional on the parent. The complementary prior is defined by Hinton et al. to be the prior distribution that contains exactly the correct correlations necessary to enable the posterior parent distribution to be factorial (independent of one-another) given the child vector.

To implement a parse tree using a DBN, we firstly require that elements of each latent node-vector p (including the root vector) only take values 0 and 1. These vectors could potentially predict any other latent multi-class variable from p through soft-max classifiers, so a binary constraint should not be too onerous.

Secondly, we require that the root node be able to generate a stationary distribution that is optimum to generate the text. We note that RBMS can flexibly generate any distribution of child vectors, limited only by the number of hidden units (the dimension of p) (Le Roux and Bengio, 2008). For modeling text, this should be sufficient, since the text is independent of the trees when given the leaf vectors' distributions.

Thirdly, we require that the DBN generates a tree with downwards pointing conditional distributions. The DBN will naturally support the conditional distribution with function F:

$$P(c_1, c_2|p) = F(p, c_1, c_2) \tag{10}$$

and the probability of generating a tree T given its root will be:

$$P(T) = \prod_{nodes\, p \in T} F(p, c_1, c_2) \tag{11}$$

Finally, however, using a DBN within a tree structure presents the problem of maintaining a complementary prior at every parent node throughout the tree. Typically, the DAG used with RBMS only recover the original complementary prior at every alternate layer. This can be overcome by initializing W with a symmetric weight matrix with $W = W^T$.

A second issue is that as we travel down the tree, the total vector count grows, so an issue of double-counting the prior arises. The prior must be a prior of both its children together, and not each child taken separately. While constructing a Deep Boltzmann Machine (DBM), Salakhutdinov and Hinton (2009) discuss how to manage the asymmetric application of DBN so the downward posterior has twice the dimensionality of the upwards posterior.

4.1.2 Parent Probability

The main feature of the DBN is that inference can be performed quickly to calculate the parent vector given the children vectors. Given a tree T, this allows us to calculate up the tree, even though it is through a downwards pointing conditional model, so the ith element of p can be predicted by:

$$P(p^{(i)} = 1 | c_1, c_2) = \sigma(c_1 W_1 + c_2 W_2)^{(i)} \tag{12}$$

It is important to note that here c_1, c_2 and p are binary sampled values, not their probabilities or averages, whereas the resulting probabilities are distributions. However, used as a mean-field approximation, Equation (12) is essentially the same as that of the RNN. The DBN overcomes this limitation by sampling at each step within the tree.

4.1.3 Tree Probability

For each word sequence, there are multiple tree structures that could generate the same text, each with varying probabilities. Occasional examples exist like 'I saw that gas can explode' having two different parses of similar probability. To select the most likely parse, we must compare the generative probability of the two trees.

When two parses differ only at some parent node h, and the trees beneath h are of the form $T_1 = a, (b, c)$ and $T_2 = (a, b), c$ (see Figure 1) and d is the parent of (a, b) and e is the parent of (b, c), then we approximate the relative probability each tree was generated as follows:

Given the supplied text under nodes a, b, c being respectively x, y, z, the conditional vector distributions can be calculated through sampling going up each tree. In both T_1 and $T_{2,}$, when we arrive at nodes a, b and c, they will each have the same sampled distributions, as below a, b, and c, the two trees are identical. The mean field approximations for the most probable combinations of each vector a, b, c will be near their expected values $\bar{a} = E(a|x)$, $\bar{b} = E(b|y)$ and $\bar{c} = E(c|z)$.

Given no information, the prior probability of h will be the complementary prior, so taking a mean field approximation, the probability T_1 generates the text will be:

$$P(x, y, z | T_1) = \sum_h \sum_d P(h, \bar{a}, d) P(\bar{b}, \bar{c} | d) P(x|\bar{a}) P(y|\bar{b}) P(z|\bar{c}) \tag{13}$$

Ignoring the common terms $P(x|a)$ etc, we get:

$$P(x, y, z | T_1) \propto \sum_{hd} P(h, \bar{a}, d) P(\bar{b}, \bar{c} | d) \tag{14}$$

If again we make the mean field approximation that $P(\bar{b},\bar{c}|d)$ only has probability mass at the point $E(\bar{b},\bar{c})$, and apply Bayes rule ignoring common terms we get:

$$P(T_1|x,y,z) \propto P(\bar{a},d) \tag{15}$$

4.1.4 Scoring the Tree Probability

The relative log probabilities for T_1 and T_2 is (where d and e are as shown in Figure 1):

$$\log(P(a,d)) : \log(P(e,c)) \tag{16}$$

So for children u, v of an arbitrary parent node h, where the parent node h has the distribution of the complementary prior, we can model the log probability $\log(P(u,v))$ by summing each one of the $2^{|h|}$ combinations of h using:

$$\log(\sum_h P(h,u,v)) = \log(\sum_h \exp(hW_1u + hW_2v + \gamma h + \alpha u + \beta v)) - \log(Z) \tag{17}$$

where α, β and γ are the biases to be learned with W_1 and W_2. We can ignore $\log(Z)$ as it is found in both T_1 and T_2. A common rearrangement is to multiply out the contributions made for each element $h^{(i)}$ of h being a 0 and a 1 (writing $\sum_i g^{(i)}$ as the sum of g's elements):

$$\begin{aligned} &= \alpha u + \beta v + \sum_i \log(1 + \exp(\gamma + W_1u + W_2v))^{(i)} && (18) \\ &= \alpha u + \beta v - \sum_i \log(1 - \sigma(\gamma + W_1u + W_2v)))^{(i)} && (19) \\ \log(P(u,v)) &= \alpha u + \beta v - \sum_i \log(1 - h_i) && (20) \end{aligned}$$

where $h_i = P(h^{(i)} = 1)$. Since u and v are factorial in h, the resulting log probability is also largely factorial in h making it suitable for a regression layer. As the RNN operates with mean-field values, it can learn $\log(P(u,v))$ through regression based on h using:

$$s(h) = \sigma(Rh) \simeq \log(P(u,v)) \propto \log(P(T_1)) \tag{21}$$

In this way, the approximate conditional probability of each parent is calculated upwards using the RNN model, and the log probability of each parse can be estimated by summing the expected values of s, and learned through back-propagation.

4.1.5 Estimating the Scoring Function

We can estimate the value of s by taking the Taylor expansion of the log probability of $\log(P(u,v))$ in terms of h_i:

$$log(P(u,v)) \simeq C + \sum_i - \log(1-h_i) \simeq C + \sum_i h_i + O(h_i^2) \tag{22}$$

The Taylor expansion of the logistic function used by the classifier by contrast is:

$$\sigma(Rh) = C + \sum_i \frac{1}{4} R^{(i)} h_i + O(h_i^2) \tag{23}$$

Comparing the linear h_i terms in Equations (22) and (23) would suggest an expected value of $R^{(i)} = 4$

It is interesting to note that in our experiments with the RNN model, the average value of $R^{(i)}$ was 1.77. However, the implementation of the RNN used the tanh function, instead of the sigmoid function σ found in the abstract DBN model for the node to node matrices W_1 and W_2 which generate the input to the scoring function. Since $\tanh(x)$ outputs values in twice the range as $\sigma(x)$, this gave the RNN model an effective average value for $R^{(i)}$ of 3.54, within 13% of the predicted value.

4.1.6 Reducing Divergence

In practice, the mean-field distribution diverges from the true combinatoric binary distribution. This becomes most noticeable when a node's distribution is sharply constrained by its own parent node, and is the result of each node being conditioned only on its children, and not on its parents. To reduce this effect, conditioning each node's probability on leaf-node vectors adjacent to the span (as in Equation 6) helps bring in some of the conditional information of the parent's node vector, since the parent node-vector is itself conditioned on its two children, one being the original node, the other being an ancestor of one of the adjacent leaf node.

4.2 RBM to Improve Modeling S

The RNN can be thought of as a mean-field approximation of the DBN model that learns to model the log probability of $P(T_1)$ through Equation (21). The R term implies that s is computed through a logistic regression.

Although the DBN abstract model indicates logistic regression on h should provide a good solution of $P(T_1)$, the mean-field approximation diverges sufficiently enough that incorporating contextual features makes a significant improvement to the results (Socher et al., 2010). This suggests that a superior scoring classifier might also further improve the results. Recalling that RBMs are better at modeling general distributions, and that when given the complementary prior, the probability of $P(T_1)$ takes the form of Equation (17), this motivates that $s(p)$ would be better modeled by a Restricted Boltzmann Machines (RBM).

RBMs have been used before in parsing (Garg and Henderson, 2011). They aim to model a generative probability $P(x,h) \propto \exp(xWh)$ and are often trained through Gibbs sampling, one layer at a time. Layer-wise training is harder for a recursive model, however, they can be used to model the distribution of the scoring function $s(p)$ in the approximate RNN model:

$$s(p) \propto \sum_h \exp(hUp + ap + bh) \tag{24}$$

4.2.1 Configuration

RBMS can be used in several configurations for modeling the probability of $s(p)$.

One method involves using two RBMS, one with energy functions $E_1 = h_1U_1p$ to model the probability that the parent node is part of the correct parse, and one with $E_2 = h_2U_2p$ to model the probability that it is not. In this case, the probability the parent node is part of the parse is given by $\exp(E_1)/(\exp(E_1) + \exp(E_2))$.

Another configuration uses a single RBM, $E = hMp$ and regression $s = \sigma(Vh)$. This model most naturally extends to a three layer neural network.

The discriminative RBM (Schmah et al., 2009) which we use goes one step further, by assuming that s is itself a latent feature of the RBM which has energy $E = hMp + hVs$. This means that $P(s = 1|p) = \sigma(Vh)$, while $P(h_i = 1|p) = \sigma(Mp + Vs)$. By including the Vs term in the calculation of $P(h_i = 1)$, it enhances the expressiveness of the model.

4.2.2 Training

Instead of using Contrastive Divergence (CD, Hinton et al., 2006) to learn the parameters of M and V, we perform discriminative RBM training, since the dimensions of s are so small. This is done by calculating the RBM twice, once for $s = 0$ and once for $s = 1$. These results can be combined to give a value of $P(s = 1|p)$, and the gradient can be used to update M and V.

We train this model for s as a post-process to training the RNN, while holding fixed the parameters of the underlying RNN model. The motivation was to effectively implement the top layer-wise training seen in deep RBM training models. We do not allow the gradient to back-propagate to update W or other parameters, as it did not improve our overall results.

4.3 Improvement to CKY through Leaf Vector Nudging

Different parse decisions made at lower levels of the tree result in different parent vectors. Even when the structure of two trees differs only slightly, as T_1 and T_2 did in Section 4.1.3, the changes to the parent vectors are not local but propagate up all the way to the root node.

This affects the ability of the CKY algorithm to find the parse with the highest log likelihood score. For longer sentences, the scoring model may give the gold parse the highest parse score, but using the CKY algorithm with a small beam may fail to find this highest scoring parse.

In this situation, there must be a node where the CKY algorithm made an incorrect decision. For the parent of this node, the gold parse's local sub-tree score was lower than several incorrect parse's sub-tree scores, enough to incorrectly fill the CKY beam with the wrong parent node vector. In this case, there will be some group of higher nodes that would have scored higher had the gold node's children vectors been used, rather than the incorrect parse node's children vectors found in the CKY beam (and by a more significant margin), but the CKY algorithm would have never calculated them.

With this motivation, we examine how to temporarily nudge some parameters of the model to encourage the CKY algorithm to find the best overall parse and, in particular, how the gradient of higher node scores can be back-propagated at evaluation time to lower nodes to encourage the CKY algorithm to favour parse decisions at a lower level that will do better at higher levels.

4.3.1 Learning Leaf Vectors

Another motivation for this approach comes from considering the process of updating the word-vectors during learning. The back-propagation for the RNN passes through the tree and leaves into the lexicon L. One can imagine that words with multiple senses will be given a blended vector representation, sometimes being tugged towards one sense's vector, and sometimes tugged towards another. In this instance, even during testing, where a parse is mostly correct it is likely that the vector would have been tugged mostly in the correct direction.

With this in mind, we found it possible to back-propagate the parse score gradient down through the tree and into the leaf vectors temporarily, just for the current parse. The goal of this is to temporarily give the leaf vector a value closer to the sense that is currently being used, thereby giving a chance of fixing any errors in the parse.

During testing, however, we do not know the correct parse, so cannot use the correct gradient for back-propagation. Instead, we use the parse scores s as a measure of confidence on each parse node, and back-propagate the direction of the gradient $\frac{ds}{dp}$ from the most confident nodes.

4.3.2 Method of Gradient Improvement

First, a parse of the sentence is generated using the RNN model and scores s_i for each parse decision calculated. The average score $\hat{s}$ is computed for the entire tree, and a reliability variable $g_i = \max(0, s_i - \hat{s})$ calculated to determine the most confident parse decisions. These values are then used as the error model and the gradient $k g_i \frac{ds_i}{dp}$ is back-propagated to the leaf-node level. This gradient is temporarily added to the leaf vectors and the sentence is re-parsed and the highest scoring parse selected. Finally the new parse is re-scored using the original leaf vectors and the new parse is only selected if the total score exceeds the sentence's original total parse score. Since we expect half the nodes to contribute gradients, we set $k = 2/(10 + \text{len(sentence)})$ which empirically worked well with the development set.

5 Results

We used the WSJ corpus of the Penn Treebank (Marcus et al., 1993), using sections 2-21 for training, 22 as a development set and 23 as the test set.

We require binary branching parse trees so, before commencing we made the following adjustments to the WSJ data that was kept for both training, development and final evaluation: all traces were removed; all unary rules (nodes with a single child) were collapsed (the resulting label was the POS tag of the leaf word); all nodes with more than 2 children were right branched (except for the most right node if it was punctuation) and the resulting node labels of the newly generated parents was made the same as for the original multi-branched parent; for short sentence experiments only sentences containing 15 or fewer tokens (including punctuation) were used.

The networks were trained with 500 passes over the training corpus with an initial learning rate of 0.005 that decreased by 2% of its value after pass. The parameters were stored whenever evaluation against the development set revealed a higher result, which was tested every 1000 sentences. Final testing was performed against section 23.

The word-vector table was initialized from pre-computed values in the lexicon (Collobert and Weston, 2008) and the word vectors took the values $L_{u(S,i)}$ according to each word $u(S,i)$ of the

Method	F1	significance
Socher $\leqslant$ 15 words	92.06%	
RNN $\leqslant$ 15 words	92.41%	
RNN all sentences	83.94%	
Gradient	84.32%	+0.38% (p=0.0001)
RBM	84.77%	+0.83% (p=0.0008)

Table 1: F1 Test scores for section 23 of the WSJ

text. For tokens found in WSJ but not in the table, we added a fresh token with a value given by either the lowercase version of the token (if found) or with the value of the *UNKNOWN* token. All numeric characters were replaced with the digit '2'.

5.1 De-Binarization

The RNN and DBN model both require binarized trees. The standard WSJ corpus is not binarized, but (especially at the leaf layer) contains many nodes consisting of more than two children. For example, tokens of multi-word nouns like "`New York Stock Exchange`" will all be children of the same node, while binarized trees might nest these as `(New (York (Stock Exchange)))`. During evaluation, the computed F-scores for binarized trees will appear inflated compared to those of un-binarized trees, since it is easier to learn and reproduce the additional right bracketings.

For comparison with other systems, we consider how the binarization process could be reversed so that generated trees may be compared with the original unbinary WSJ corpus. This could be done through coded compounding rules based on predicted phrase categories labels, or by learning a de-binarize feature. The method we used was as follows.

There are 28 non-terminal labels that are applied to the gold-standard text (e.g. `S, VP, NP`). During the process of right binarizing the text, we label the newly created nodes with one of 28 newly created categories (e.g. `!S, !VP, !NP`) generated by adding '`!`' to the label of the new node's parent node. In this manner we identify which nodes to delete when debinarizing generated parses.

Unless otherwise noted, the results we show are for binarized trees, as binarized trees were used in the original experiments by Socher (2012).

5.2 Evaluation

Testing was performed using evalb. The F1 score (precision and accuracy are are identical for binarized trees) was taken as the overall reported performance.

A result of 92.41% was achieved by training on all the text, and testing on just the sentences up to 15 tokens long. A result of 83.94% used the same model, testing on the entire test set, both short and long sentences (refer to Table 1). For comparison with other parsers, we obtained a debinarized F-Score of 89.47% on short sentences.

Significance was calculated using Dan Bikel's Randomized Parsing Evaluation Comparator. [1]

[1] http://www.cis.upenn.edu/~dbikel/download/compare.pl

token	$error^2$	token	$error^2$
.	0.128	if	0.032
?	0.125	It	0.029
:	0.077	get	0.028
.	0.054	what	0.026
–	0.046	That	0.025

Table 2: Tokens and square errors of modifications during testing

5.3 RBM

The discriminative RBM model was used as a post-process to learn an improved model of the scoring function. It was trained in about 24 hours, holding the other parameters of the model fixed. It out-performed the original model by 0.83% (with a p-value of 0.0008). The training was entirely discriminatively performed with no back propagation through the structure.

5.4 Gradient Update

The gradient update method was used entirely during evaluation time. It improved the F-score by just 0.38% (with a p-value of 0.0001). The significance was higher because more of the mistakes it made were also made in the baseline model.

The tokens with the greatest average modification were mostly punctuation and function words that shaped the structure of the sentence (see Table 2). These words control the sentence structure but have little distributional information. Other highly modified words were those with low frequency counts in the corpus.

Conclusion

We have demonstrated the performance of a Recursive Neural Network parser to binarized forms of all sentences in the WSJ corpus, obtaining a baseline of 83.94%. We have presented a method to improve parser performance by using a discriminative Restricted Boltzmann Machine for scoring productions increasing F-score by 0.83%. We have also presented a gradient method that can be used during evaluation that augments the CKY algorithm and improves accuracy by a separate 0.38%. We have provided a framework to draw a connection between top-down generative conditional parsers with bottom-up conditional RNN parsers, and using this framework have analytically calculated estimates of the learned RNN parameters.

In future work, we hope to implement the generative model described in this paper using Contrastive Divergence starting with just the leaf layer and increasing one parse layer at a time.

The methods we presented are easily applied to other recursive tasks and network structures and provide a link between generative and conditional parsing. These improvements can be used in future down-stream tasks.

Acknowledgments

We would like to thank the reviewers and Schwa-Lab for their comments. This work was supported by Australian Research Council Discovery grant DP1097291, the Capital Markets CRC and an Australian Postgraduate Award. We are grateful to Richard Socher for his guidance reproducing his results in our system.

References

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *Neural Networks, IEEE Transactions on*, 5(2):157–166.

Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. In *Compstat*, volume 2010, pages 177–186.

Charniak, E. (1997). Statistical techniques for natural language parsing. *AI magazine*, 18(4):33.

Charniak, E. (2000). A maximum-entropy-inspired parser. In *North American chapter of the Association for Computational Linguistics*, pages 132–139.

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

Chen, S. F. and Goodman, J. (1996). An empirical study of smoothing techniques for language modeling. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, pages 310–318.

Clark, S. and Curran, J. R. (2004). Parsing the wsj using ccg and log-linear models. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, ACL '04.

Collobert, R. and Weston, J. (2008). A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167.

Costa, F., Frasconi, P., Lombardo, V., and Soda, G. (2003). Towards incremental parsing of natural language using recursive neural networks. *Applied Intelligence*, 19(1):9–25.

Curran, J. R., Clark, S., and Bos, J. (2007). Linguistically motivated large-scale nlp with c&c and boxer. In *Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions*, pages 33–36.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 1–38.

Garg, N. and Henderson, J. (2011). Temporal restricted boltzmann machines for dependency parsing. In *Proceedings of the 11th International Conference on Parsing Technologies*, pages 11–17.

Hinton, G. E., Osindero, S., and Teh, Y. W. (2006). A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554.

Kasami, T. (1965). An efficient recognition and syntax analysis algorithm for context-free languages. Technical report.

Klein, D. and Manning, C. D. (2003). Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*, pages 423–430.

Le Roux, N. and Bengio, Y. (2008). Representational power of restricted boltzmann machines and deep belief networks. *Neural Computation*, 20(6):1631–1649.

Marcus, M. P., Marcinkiewicz, M. A., and Santorini, B. (1993). Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330.

McClosky, D., Charniak, E., and Johnson, M. (2006). Effective self-training for parsing. In *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, pages 152–159.

Mnih, A. and Hinton, G. (2007). Three new graphical models for statistical language modelling. *Proceedings of the 24th international conference on Machine learning - ICML '07*, pages 641–648.

Pearson, K. (1901). On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572.

Petrov, S. and Klein, D. (2007). Improved inference for unlexicalized parsing. In *Proceedings of North American Association for Computational Linguistics HLT 2007*, pages 404–411.

Rumelhart, D. E., Hintont, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088):533–536.

Salakhutdinov, R. and Hinton, G. E. (2009). Deep boltzmann machines. In *Proceedings of the international conference on artificial intelligence and statistics*, volume 5, pages 448–455.

Schmah, T., Hinton, G. E., Zemel, R., Small, S. L., and Strother, S. (2009). Generative versus discriminative training of rbms for classification of fmri images. In *Proceedings of Neural Information Processing Systems*, volume 2009.

Socher, R. (2012). Personal communication.

Socher, R., Manning, C. D., and Ng, A. Y. (2010). Learning continuous phrase representations and syntactic parsing with recursive neural networks. In *Deep Learning and Unsupervised Feature Learning Workshop*, pages 1–9.

Socher, R., Pennington, J., Huang, E. H., Ng, A. Y., and Manning, C. D. (2011). Semi-supervised recursive autoencoders for predicting sentiment distributions. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Titov, I. and Henderson, J. (2007). Constituent parsing with incremental sigmoid belief networks. In *Annual Meeting of the Association for Computational Linguistics*, volume 45, page 632.