Structured Training for Neural Network Transition-Based Parsing

David Weiss **Chris Alberti** **Michael Collins** **Slav Petrov**
Google Inc
New York, NY
{djweiss,chrisalberti,mjcollins,slav}@google.com

Abstract

We present structured perceptron training for neural network transition-based dependency parsing. We learn the neural network representation using a gold corpus augmented by a large number of automatically parsed sentences. Given this fixed network representation, we learn a final layer using the structured perceptron with beam-search decoding. On the Penn Treebank, our parser reaches 94.26% unlabeled and 92.41% labeled attachment accuracy, which to our knowledge is the best accuracy on Stanford Dependencies to date. We also provide in-depth ablative analysis to determine which aspects of our model provide the largest gains in accuracy.

1 Introduction

Syntactic analysis is a central problem in language understanding that has received a tremendous amount of attention. Lately, dependency parsing has emerged as a popular approach to this problem due to the availability of dependency treebanks in many languages (Buchholz and Marsi, 2006; Nivre et al., 2007; McDonald et al., 2013) and the efficiency of dependency parsers.

Transition-based parsers (Nivre, 2008) have been shown to provide a good balance between efficiency and accuracy. In transition-based parsing, sentences are processed in a linear left to right pass; at each position, the parser needs to choose from a set of possible actions defined by the transition strategy. In greedy models, a classifier is used to independently decide which transition to take based on local features of the current parse configuration. This classifier typically uses hand-engineered features and is trained on individual transitions extracted from the gold transition sequence. While extremely fast, these greedy models typically suffer from search errors due to the inability to recover from incorrect decisions. Zhang and Clark (2008) showed that a beam-search decoding algorithm utilizing the structured perceptron training algorithm can greatly improve accuracy. Nonetheless, significant manual feature engineering was required before transition-based systems provided competitive accuracy with graph-based parsers (Zhang and Nivre, 2011), and only by incorporating graph-based scoring functions were Bohnet and Kuhn (2012) able to exceed the accuracy of graph-based approaches.

In contrast to these carefully hand-tuned approaches, Chen and Manning (2014) recently presented a neural network version of a greedy transition-based parser. In their model, a feed-forward neural network with a hidden layer is used to make the transition decisions. The hidden layer has the power to learn arbitrary combinations of the atomic inputs, thereby eliminating the need for hand-engineered features. Furthermore, because the neural network uses a distributed representation, it is able to model lexical, part-of-speech (POS) tag, and arc label similarities in a continuous space. However, although their model outperforms its greedy hand-engineered counterparts, it is not competitive with state-of-the-art dependency parsers that are trained for structured search.

In this work, we combine the representational power of neural networks with the superior search enabled by structured training and inference, making our parser one of the most accurate dependency parsers to date. Training and testing on the Penn Treebank (Marcus et al., 1993), our transition-based parser achieves 93.99% unlabeled (UAS) / 92.05% labeled (LAS) attachment accuracy, outperforming the 93.22% UAS / 91.02% LAS of Zhang and McDonald (2014) and 93.27 UAS / 91.19 LAS of Bohnet and Kuhn (2012). In addition, by incorporating unlabeled data into training, we further improve the accuracy of our model to 94.26% UAS / 92.41% LAS (93.46%

Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing, pages 323–333, Beijing, China, July 26-31, 2015. ©2015 Association for Computational Linguistics

UAS / 91.49% LAS for our greedy model).

In our approach we start with the basic structure of Chen and Manning (2014), but with a deeper architecture and improvements to the optimization procedure. These modifications (Section 2) increase the performance of the greedy model by as much as 1%. As in prior work, we train the neural network to model the probability of individual parse actions. However, we do not use these probabilities directly for prediction. Instead, we use the activations from all layers of the neural network as the representation in a structured perceptron model that is trained with beam search and early updates (Section 3). On the Penn Treebank, this structured learning approach significantly improves parsing accuracy by 0.8%.

An additional contribution of this work is an effective way to leverage unlabeled data. Neural networks are known to perform very well in the presence of large amounts of training data; however, obtaining more expert-annotated parse trees is very expensive. To this end, we generate large quantities of high-confidence parse trees by parsing unlabeled data with two different parsers and selecting only the sentences for which the two parsers produced the same trees (Section 3.3). This approach is known as "tri-training" (Li et al., 2014) and we show that it benefits our neural network parser significantly more than other approaches. By adding 10 million automatically parsed tokens to the training data, we improve the accuracy of our parsers by almost ~1.0% on web domain data.

We provide an extensive exploration of our model in Section 5 through ablative analysis and other retrospective experiments. One of the goals of this work is to provide guidance for future refinements and improvements on the architecture and modeling choices we introduce in this paper.

Finally, we also note that neural network representations have a long history in syntactic parsing (Henderson, 2004; Titov and Henderson, 2007; Titov and Henderson, 2010); however, like Chen and Manning (2014), our network avoids any recurrent structure so as to keep inference fast and efficient and to allow the use of simple backpropagation to compute gradients. Our work is also not the first to apply structured training to neural networks (see e.g. Peng et al. (2009) and Do and Artires (2010) for Conditional Random Field (CRF) training of neural networks). Our paper extends this line of work to the setting of inexact search with beam decoding for dependency parsing; Zhou et al. (2015) concurrently explored a similar approach using a structured probabilistic ranking objective. Dyer et al. (2015) concurrently developed the Stack Long Short-Term Memory (S-LSTM) architecture, which does incorporate recurrent architecture and look-ahead, and which yields comparable accuracy on the Penn Treebank to our greedy model.

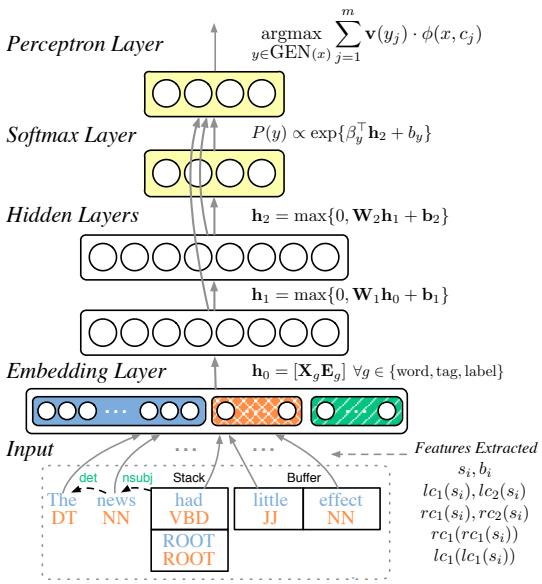

Figure 1: Schematic overview of our neural network model. Atomic features are extracted from the i'th elements on the stack (s_i) and the buffer (b_i); lc_i indicates the i'th leftmost child and rc_i the i'th rightmost child. We use the top two elements on the stack for the arc features and the top four tokens on stack and buffer for words, tags and arc labels.

2 Neural Network Model

In this section, we describe the architecture of our model, which is summarized in Figure 1. Note that we separate the embedding processing to a distinct "embedding layer" for clarity of presentation. Our model is based upon that of Chen and Manning (2014) and we discuss the differences between our model and theirs in detail at the end of this section. We use the *arc-standard* (Nivre, 2004) transition system.

2.1 Input layer

Given a parse configuration c (consisting of a stack s and a buffer b), we extract a rich set of discrete features which we feed into the neural network. Following Chen and Manning (2014), we group these features by their input source: words, POS tags, and arc labels. The features extracted

for each group are represented as a sparse $F \times V$ matrix $\mathbf{X}$, where V is the size of the vocabulary of the feature group and F is the number of features. The value of element X_{fv} is 1 if the f'th feature takes on value v. We produce three input matrices: $\mathbf{X}_{\text{word}}$ for words features, $\mathbf{X}_{\text{tag}}$ for POS tag features, and $\mathbf{X}_{\text{label}}$ for arc labels, with $F_{\text{word}} = F_{\text{tag}} = 20$ and $F_{\text{label}} = 12$ (Figure 1).

For all feature groups, we add additional special values for "ROOT" (indicating the POS or word of the root token), "NULL" (indicating no valid feature value could be computed) or "UNK" (indicating an out-of-vocabulary item).

2.2 Embedding layer

The first learned layer h_0 in the network transforms the sparse, discrete features $\mathbf{X}$ into a dense, continuous embedded representation. For each feature group $\mathbf{X}_g$, we learn a $V_g \times D_g$ embedding matrix $\mathbf{E}_g$ that applies the conversion:

$$\mathbf{h}_0 = [\mathbf{X}_g \mathbf{E}_g \mid g \in \{\text{word}, \text{tag}, \text{label}\}], \quad (1)$$

where we apply the computation separately for each group g and concatenate the results. Thus, the embedding layer has $E = \sum_g F_g D_g$ outputs, which we reshape to a vector $\mathbf{h}_0$. We can choose the embedding dimensionality D for each group freely. Since POS tags and arc labels have much smaller vocabularies, we show in our experiments (Section 5.1) that we can use smaller D_{tag} and D_{label}, without a loss in accuracy.

2.3 Hidden layers

We experimented with one and two hidden layers composed of M rectified linear (Relu) units (Nair and Hinton, 2010). Each unit in the hidden layers is fully connected to the previous layer:

$$\mathbf{h}_i = \max\{0, \mathbf{W}_i \mathbf{h}_{i-1} + \mathbf{b}_i\}, \quad (2)$$

where $\mathbf{W}_1$ is a $M_1 \times E$ weight matrix for the first hidden layer and $\mathbf{W}_i$ are $M_i \times M_{i-1}$ matrices for all subsequent layers. The weights $\mathbf{b}_i$ are bias terms. Relu layers have been well studied in the neural network literature and have been shown to work well for a wide domain of problems (Krizhevsky et al., 2012; Zeiler et al., 2013). Through most of development, we kept $M_i = 200$, but we found that significantly increasing the number of hidden units improved our results for the final comparison.

2.4 Relationship to Chen and Manning (2014)

Our model is clearly inspired by and based on the work of Chen and Manning (2014). There are a few structural differences: (1) we allow for much smaller embeddings of POS tags and labels, (2) we use Relu units in our hidden layers, and (3) we use a deeper model with two hidden layers. Somewhat to our surprise, we found these changes combined with an SGD training scheme (Section 3.1) during the "pre-training" phase of the model to lead to an almost 1% accuracy gain over Chen and Manning (2014). This trend held despite carefully tuning hyperparameters for each method of training and structure combination.

Our main contribution from an algorithmic perspective is our training procedure: as described in the next section, we use the structured perceptron for learning the final layer of our model. We thus present a novel way to leverage a neural network representation in a structured prediction setting.

3 Semi-Supervised Structured Learning

In this work, we investigate a semi-supervised structured learning scheme that yields substantial improvements in accuracy over the baseline neural network model. There are two complementary contributions of our approach: (1) incorporating structured learning of the model and (2) utilizing unlabeled data. In both cases, we use the neural network to model the probability of each parsing action y as a soft-max function taking the final hidden layer as its input:

$$P(y) \propto \exp\{\beta_y^\top \mathbf{h}_i + b_y\}, \quad (3)$$

where β_y is a M_i dimensional vector of weights for class y and i is the index of the final hidden layer of the network. At a high level our approach can be summarized as follows:

- First, we pre-train the network's hidden representations by learning probabilities of parsing actions. Fixing the hidden representations, we learn an additional final output layer using the structured perceptron that uses the output of the network's hidden layers. In practice this improves accuracy by ~0.6% absolute.

- Next, we show that we can supplement the gold data with a large corpus of high quality

automatic parses. We show that incorporating unlabeled data in this way improves accuracy by as much as 1% absolute.

3.1 Backpropagation Pretraining

To learn the hidden representations, we use mini-batched averaged stochastic gradient descent (ASGD) (Bottou, 2010) with momentum (Hinton, 2012) to learn the parameters Θ of the network, where $\Theta = \{\mathbf{E}_g, \mathbf{W}_i, \mathbf{b}_i, \beta_y \mid \forall g, i, y\}$. We use backpropagation to minimize the multinomial logistic loss:

$$L(\Theta) = -\sum_j \log P(y_j \mid c_j, \Theta) + \lambda \sum_i \|\mathbf{W}_i\|_2^2, \quad (4)$$

where λ is a regularization hyper-parameter over the hidden layer parameters (we use $\lambda = 10^{-4}$ in all experiments) and j sums over all decisions and configurations $\{y_j, c_j\}$ extracted from gold parse trees in the dataset.

The specific update rule we apply at iteration t is as follows:

$$g_t = \mu g_{t-1} - \Delta L(\Theta_t), \quad (5)$$

$$\Theta_{t+1} = \Theta_t + \eta_t g_t, \quad (6)$$

where the descent direction g_t is computed by a weighted combination of the previous direction $g_{t-1}$and the current gradient $\Delta L(\Theta_t)$. The parameter $\mu \in [0, 1)$ is the momentum parameter while η_t is the traditional learning rate. In addition, since we did not tune the regularization parameter λ, we apply a simple exponential step-wise decay to η_t; for every γ rounds of updates, we multiply $\eta_t = 0.96\eta_{t-1}$.

The final component of the update is parameter averaging: we maintain averaged parameters $\bar{\Theta}_t = \alpha_t \bar{\Theta}_{t-1} + (1 - \alpha_t)\Theta_t$, where α_t is an averaging weight that increases from 0.1 to 0.9999 with $1/t$. Combined with averaging, careful tuning of the three hyperparameters μ, η_0, and γ using held-out data was crucial in our experiments.

3.2 Structured Perceptron Training

Given the hidden representations, we now describe how the perceptron can be trained to utilize these representations. The perceptron algorithm with early updates (Collins and Roark, 2004) requires a feature-vector definition ϕ that maps a sentence x together with a configuration c to a feature vector $\phi(x, c) \in \mathbb{R}^d$. There is a one-to-one mapping between configurations c and decision sequences $y_1 \ldots y_{j-1}$ for any integer $j \geq 1$: we will use c and $y_1 \ldots y_{j-1}$ interchangeably.

For a sentence x, define GEN(x) to be the set of parse trees for x. Each $y \in$ GEN(x) is a sequence of decisions $y_1 \ldots y_m$ for some integer m. We use $\mathcal{Y}$ to denote the set of possible decisions in the parsing model. For each decision $y \in \mathcal{Y}$ we assume a parameter vector $\mathbf{v}(y) \in \mathbb{R}^d$. These parameters will be trained using the perceptron.

In decoding with the perceptron-trained model, we will use beam search to attempt to find:

$$\operatorname*{argmax}_{y \in \text{GEN}(x)} \sum_{j=1}^{m} \mathbf{v}(y_j) \cdot \phi(x, y_1 \ldots y_{j-1}).$$

Thus each decision y_j receives a score:

$$\mathbf{v}(y_j) \cdot \phi(x, y_1 \ldots y_{j-1}).$$

In the perceptron with early updates, the parameters $\mathbf{v}(y)$ are trained as follows. On each training example, we run beam search until the gold-standard parse tree falls out of the beam.[1] Define j to be the length of the beam at this point. A structured perceptron update is performed using the gold-standard decisions $y_1 \ldots y_j$ as the target, and the highest scoring (incorrect) member of the beam as the negative example.

A key idea in this paper is to use the neural network to define the representation $\phi(x, c)$. Given the sentence x and the configuration c, assuming two hidden layers, the neural network defines values for $\mathbf{h}_1$, $\mathbf{h}_2$, and $P(y)$ for each decision y. We experimented with various definitions of ϕ (Section 5.2) and found that $\phi(x, c) = [\mathbf{h}_1\ \mathbf{h}_2\ P(y)]$ (the concatenation of the outputs from both hidden layers, as well as the probabilities for all decisions y possible in the current configuration) had the best accuracy on development data.

Note that it is possible to continue to use backpropagation to learn the representation $\phi(x, c)$ during perceptron training; however, we found using ASGD to pre-train the representation always led to faster, more accurate results in preliminary experiments, and we left further investigation for future work.

3.3 Incorporating Unlabeled Data

Given the high capacity, non-linear nature of the deep network we hypothesize that our model can

[1] If the gold parse tree stays within the beam until the end of the sentence, conventional perceptron updates are used.

be significantly improved by incorporating more data. One way to use unlabeled data is through unsupervised methods such as word clusters (Koo et al., 2008); we follow Chen and Manning (2014) and use pretrained word embeddings to initialize our model. The word embeddings capture similar distributional information as word clusters and give consistent improvements by providing a good initialization and information about words not seen in the treebank data.

However, obtaining more training data is even more important than a good initialization. One potential way to obtain additional training data is by parsing unlabeled data with previously trained models. McClosky et al. (2006) and Huang and Harper (2009) showed that iteratively re-training a single model ("self-training") can be used to improve parsers in certain settings; Petrov et al. (2010) built on this work and showed that a slow and accurate parser can be used to "up-train" a faster but less accurate parser.

In this work, we adopt the "tri-training" approach of Li et al. (2014): Two parsers are used to process the unlabeled corpus and only sentences for which both parsers produced the same parse tree are added to the training data. The intuition behind this idea is that the chance of the parse being correct is much higher when the two parsers agree: there is only one way to be correct, while there are many possible incorrect parses. Of course, this reasoning holds only as long as the parsers suffer from different biases.

We show that tri-training is far more effective than vanilla up-training for our neural network model. We use same setup as Li et al. (2014), intersecting the output of the BerkeleyParser (Petrov et al., 2006), and a reimplementation of ZPar (Zhang and Nivre, 2011) as our baseline parsers. The two parsers agree only 36% of the time on the tune set, but their accuracy on those sentences is 97.26% UAS, approaching the inter annotator agreement rate. These sentences are of course easier to parse, having an average length of 15 words, compared to 24 words for the tune set overall. However, because we only use these sentences to extract individual transition decisions, the shorter length does not seem to hurt their utility. We generate 10^7 tokens worth of new parses and use this data in the backpropagation stage of training.

4 Experiments

In this section we present our experimental setup and the main results of our work.

4.1 Experimental Setup

We conduct our experiments on two English language benchmarks: (1) the standard Wall Street Journal (WSJ) part of the Penn Treebank (Marcus et al., 1993) and (2) a more comprehensive union of publicly available treebanks spanning multiple domains. For the WSJ experiments, we follow standard practice and use sections 2-21 for training, section 22 for development and section 23 as the final test set. Since there are many hyperparameters in our models, we additionally use section 24 for tuning. We convert the constituency trees to Stanford style dependencies (De Marneffe et al., 2006) using version 3.3.0 of the converter. We use a CRF-based POS tagger to generate 5-fold jack-knifed POS tags on the training set and predicted tags on the dev, test and tune sets; our tagger gets comparable accuracy to the Stanford POS tagger (Toutanova et al., 2003) with 97.44% on the test set. We report unlabeled attachment score (UAS) and labeled attachment score (LAS) excluding punctuation on predicted POS tags, as is standard for English.

For the second set of experiments, we follow the same procedure as above, but with a more diverse dataset for training and evaluation. Following Vinyals et al. (2015), we use (in addition to the WSJ), the OntoNotes corpus version 5 (Hovy et al., 2006), the English Web Treebank (Petrov and McDonald, 2012), and the updated and corrected Question Treebank (Judge et al., 2006). We train on the union of each corpora's training set and test on each domain separately. We refer to this setup as the "Treebank Union" setup.

In our semi-supervised experiments, we use the corpus from Chelba et al. (2013) as our source of unlabeled data. We process it with the BerkeleyParser (Petrov et al., 2006), a latent variable constituency parser, and a reimplementation of ZPar (Zhang and Nivre, 2011), a transition-based parser with beam search. Both parsers are included as baselines in our evaluation. We select the first 10^7 tokens for which the two parsers agree as additional training data. For our tri-training experiments, we re-train the POS tagger using the POS tags assigned on the unlabeled data from the Berkeley constituency parser. This increases POS

Method	UAS	LAS	Beam
Graph-based			
Bohnet (2010)	92.88	90.71	n/a
Martins et al. (2013)	92.89	90.55	n/a
Zhang and McDonald (2014)	93.22	91.02	n/a
Transition-based			
⋆Zhang and Nivre (2011)	93.00	90.95	32
Bohnet and Kuhn (2012)	93.27	91.19	40
Chen and Manning (2014)	91.80	89.60	1
S-LSTM (Dyer et al., 2015)	93.20	90.90	1
Our Greedy	93.19	91.18	1
Our Perceptron	**93.99**	**92.05**	8
Tri-training			
⋆Zhang and Nivre (2011)	92.92	90.88	32
Our Greedy	93.46	91.49	1
Our Perceptron	**94.26**	**92.41**	8

Table 1: Final WSJ test set results. We compare our system to state-of-the-art graph-based and transition-based dependency parsers. ⋆ denotes our own re-implementation of the system so we could compare tri-training on a competitive baseline. All methods except Chen and Manning (2014) and Dyer et al. (2015) were run using predicted tags from our POS tagger. For reference, the accuracy of the Berkeley constituency parser (after conversion) is 93.61% UAS / 91.51% LAS.

Method	News	Web	QTB
Graph-based			
Bohnet (2010)	91.38	85.22	91.49
Martins et al. (2013)	91.13	85.04	91.54
Zhang and McDonald (2014)	91.48	85.59	90.69
Transition-based			
⋆Zhang and Nivre (2011)	91.15	85.24	**92.46**
Bohnet and Kuhn (2012)	91.69	85.33	92.21
Our Greedy	91.21	85.41	90.61
Our Perceptron (B=16)	**92.25**	**86.44**	92.06
Tri-training			
⋆Zhang and Nivre (2011)	91.46	85.51	91.36
Our Greedy	91.82	86.37	90.58
Our Perceptron (B=16)	**92.62**	**87.00**	**93.05**

Table 2: Final Treebank Union test set results. We report LAS only for brevity; see Appendix for full results. For these tri-training results, we sampled sentences to ensure the distribution of sentence lengths matched the distribution in the training set, which we found marginally improved the ZPar tri-training performance. For reference, the accuracy of the Berkeley constituency parser (after conversion) is 91.66% WSJ, 85.93% Web, and 93.45% QTB.

accuracy slightly to 97.57% on the WSJ.

4.2 Model Initialization & Hyperparameters

In all cases, we initialized $\mathbf{W}_i$ and β randomly using a Gaussian distribution with variance 10^{-4}. We used fixed initialization with $b_i = 0.2$, to ensure that most Relu units are activated during the initial rounds of training. We did not systematically compare this random scheme to others, but we found that it was sufficient for our purposes.

For the word embedding matrix $\mathbf{E}_{\text{word}}$, we initialized the parameters using pretrained word embeddings. We used the publicly available `word2vec`[2] tool (Mikolov et al., 2013) to learn CBOW embeddings following the sample configuration provided with the tool. For words not appearing in the unsupervised data and the special "NULL" etc. tokens, we used random initialization. In preliminary experiments we found no difference between training the word embeddings on 1 billion or 10 billion tokens. We therefore trained the word embeddings on the same corpus we used for tri-training (Chelba et al., 2013).

We set $D_{\text{word}} = 64$ and $D_{\text{tag}} = D_{\text{label}} = 32$ for embedding dimensions and $M_1 = M_2 = 2048$ hidden units in our final experiments. For the perceptron layer, we used $\phi(x, c) = [\mathbf{h}_1\ \mathbf{h}_2\ P(y)]$ (concatenation of all intermediate layers). All hyperparameters (including structure) were tuned using Section 24 of the WSJ only. When not tri-training, we used hyperparameters of $\gamma = 0.2$, $\eta_0 = 0.05$, $\mu = 0.9$, early stopping after roughly 16 hours of training time. With the tri-training data, we decreased $\eta_0 = 0.05$, increased $\gamma = 0.5$, and decreased the size of the network to $M_1 = 1024$, $M_2 = 256$ for run-time efficiency, and trained the network for approximately 4 days. For the Treebank Union setup, we set $M_1 = M_2 = 1024$ for the standard training set and for the tri-training setup.

4.3 Results

Table 1 shows our final results on the WSJ test set, and Table 2 shows the cross-domain results from the Treebank Union. We compare to the best dependency parsers in the literature. For (Chen and Manning, 2014) and (Dyer et al., 2015), we use reported results; the other baselines were run by Bernd Bohnet using version 3.3.0 of the Stanford dependencies and our predicted POS tags for all datasets to make comparisons as fair as possible. On the WSJ and Web tasks, our parser outperforms all dependency parsers in our comparison by a substantial margin. The Question (QTB) dataset is more sensitive to the smaller beam size we use in order to train the models in a reasonable time; if we increase to $B = 32$ at inference

[2] http://code.google.com/p/word2vec/

time only, our perceptron performance goes up to 92.29% LAS.

Since many of the baselines could not be directly compared to our semi-supervised approach, we re-implemented Zhang and Nivre (2011) and trained on the tri-training corpus. Although tri-training did help the baseline on the dev set (Figure 4), test set performance did not improve significantly. In contrast, it is quite exciting to see that after tri-training, even our greedy parser is more accurate than any of the baseline dependency parsers and competitive with the Berkeley-Parser used to generate the tri-training data. As expected, tri-training helps most dramatically to increase accuracy on the Treebank Union setup with diverse domains, yielding 0.4-1.0% absolute LAS improvement gains for our most accurate model.

Unfortunately we are not able to compare to several semi-supervised dependency parsers that achieve some of the highest reported accuracies on the WSJ, in particular Suzuki et al. (2009), Suzuki et al. (2011) and Chen et al. (2013). These parsers use the Yamada and Matsumoto (2003) dependency conversion and the accuracies are therefore not directly comparable. The highest of these is Suzuki et al. (2011), with a reported accuracy of 94.22% UAS. Even though the UAS is not directly comparable, it is typically similar, and this suggests that our model is competitive with some of the highest reported accuries for dependencies on WSJ.

5 Discussion

In this section, we investigate the contribution of the various components of our approach through ablation studies and other systematic experiments. We tune on Section 24, and use Section 22 for comparisons in order to not pollute the official test set (Section 23). We focus on UAS as we found the LAS scores to be strongly correlated. Unless otherwise specified, we use 200 hidden units in each layer to be able to run more ablative experiments in a reasonable amount of time.

5.1 Impact of Network Structure

In addition to initialization and hyperparameter tuning, there are several additional choices about model structure and size a practitioner faces when implementing a neural network model. We explore these questions and justify the particular choices we use in the following. Note that we do not use a beam for this analysis and therefore do not train the final perceptron layer. This is done in order to reduce training times and because the trends persist across settings.

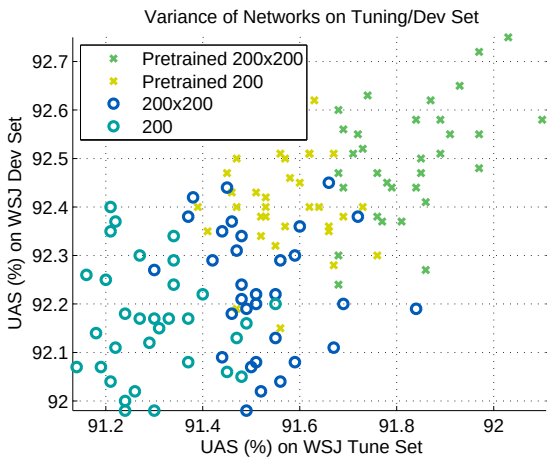

Figure 2: Effect of hidden layers and pre-training on variance of random restarts. Initialization was either completely random or initialized with `word2vec` embeddings ("Pre-trained"), and either one or two hidden layers of size 200 were used ("200" vs "200x200"). Each point represents maximization over a small hyperparameter grid with early stopping based on WSJ tune set UAS score. $D_{\text{word}} = 64$, $D_{\text{tag}}, D_{\text{label}} = 16$.

Variance reduction with pre-trained embeddings. Since the learning problem is non-convex, different initializations of the parameters yield different solutions to the learning problem. Thus, for any given experiment, we ran multiple random restarts for every setting of our hyperparameters and picked the model that performed best using the held-out tune set. We found it important to allow the model to stop training early if tune set accuracy decreased.

We visualize the performance of 32 random restarts with one or two hidden layers and with and without pretrained word embeddings in Figure 2, and a summary of the figure in Table 3. While adding a second hidden layer results in a large gain on the tune set, there is no gain on the dev set if pre-trained embeddings are not used. In fact, while the overall UAS scores of the tune set and dev set are strongly correlated ($\rho = 0.64$, $p < 10^{-10}$), they are *not* significantly correlated if pre-trained embeddings are not used ($\rho = 0.12$, $p > 0.3$). This suggests that an additional benefit of pre-trained embeddings, aside from allowing learning to reach a more accurate solution, is to push learning towards a solution that generalizes to more data.

Pre	Hidden	WSJ §24 (Max)	WSJ §22
Y	200×200	92.10 ± 0.11	**92.58** ± 0.12
Y	200	91.76 ± 0.09	92.30 ± 0.10
N	200×200	91.84 ± 0.11	92.19 ± 0.13
N	200	91.55 ± 0.10	92.20 ± 0.12

Table 3: Impact of network architecture on UAS for greedy inference. We select the best model from 32 random restarts based on the tune set and show the resulting dev set accuracy. We also show the standard deviation across the 32 restarts.

# Hidden	64	128	256	512	1024	2048
1 Layer	91.73	92.27	92.48	92.73	92.74	**92.83**
2 Layers	91.89	92.40	92.71	92.70	92.96	**93.13**

Table 4: Increasing hidden layer size increases WSJ Dev UAS. Shown is the average WSJ Dev UAS across hyperparameter tuning and early stopping with 3 random restarts with a greedy model.

Diminishing returns with increasing embedding dimensions. For these experiments, we fixed one embedding type to a high value and reduced the dimensionality of all others to very small values. The results are plotted in Figure 3, suggesting larger embeddings do not significantly improve results. We also ran tri-training on a very compact model with $D_{\text{word}} = 8$ and $D_{\text{tag}} = D_{\text{label}} = 2$ (8× fewer parameters than our full model) which resulted in 92.33% UAS accuracy on the dev set. This is comparable to the full model without tri-training, suggesting that more training data can compensate for fewer parameters.

Increasing hidden units yields large gains. For these experiments, we fixed the embedding sizes $D_{\text{word}} = 64$, $D_{\text{tag}} = D_{\text{label}} = 32$ and tried increasing and decreasing the dimensionality of the hidden layers on a logarthmic scale. Improvements in accuracy did not appear to saturate even with increasing the number of hidden units by an order of magnitude, though the network became too slow to train effectively past $M = 2048$. These results suggest that there are still gains to be made by increasing the efficiency of larger networks, even for greedy shift-reduce parsers.

5.2 Impact of Structured Perceptron

We now turn our attention to the importance of structured perceptron training as well as the impact of different latent representations.

Bias reduction through structured training. To evaluate the impact of structured training, we compare using the estimates $P(y)$ from the neural network directly for beam search to using the activations from all layers as features in the structured perceptron. Using the probability estimates directly is very similar to Ratnaparkhi (1997), where a maximum-entropy model was used to model the distribution over possible actions at each parser state, and beam search was used to search for the highest probability parse. A known problem with beam search in this setting is the label-bias problem. Table 5 shows the impact of using structured perceptron training over using the softmax function during beam search as a function of the beam size used. For reference, our reimplementation of Zhang and Nivre (2011) is trained equivalently for each setting. We also show the impact on beam size when tri-training is used. Although the beam does marginally improve accuracy for the softmax model, much greater gains are achieved when perceptron training is used.

Beam	1	2	4	8	16	32
WSJ Only						
ZN'11	90.55	91.36	92.54	92.62	92.88	**93.09**
Softmax	92.74	93.07	93.16	**93.25**	93.24	93.24
Perceptron	92.73	93.06	93.40	93.47	93.50	**93.58**
Tri-training						
ZN'11	91.65	92.37	**93.37**	93.24	93.21	93.18
Softmax	93.71	93.82	93.86	**93.87**	93.87	93.87
Perceptron	93.69	94.00	94.23	**94.33**	94.31	94.32

Table 5: Beam search always yields significant gains but using perceptron training provides even larger benefits, especially for the tri-trained neural network model. The best result for each model is highlighted in bold.

$\phi(x, c)$	WSJ Only	Tri-training
$[\mathbf{h}_2]$	93.16	93.93
$[P(y)]$	93.26	93.80
$[\mathbf{h}_1\ \mathbf{h}_2]$	93.33	93.95
$[\mathbf{h}_1\ \mathbf{h}_2\ P(y)]$	**93.47**	**94.33**

Table 6: Utilizing all intermediate representations improves performance on the WSJ dev set. All results are with $B = 8$.

Using all hidden layers crucial for structured perceptron. We also investigated the impact of connecting the final perceptron layer to all prior hidden layers (Table 6). Our results suggest that all intermediate layers of the network are indeed discriminative. Nonetheless, aggregating all of their activations proved to be the most effective representation for the structured perceptron. This suggests that the representations learned by the network collectively contain the information re-

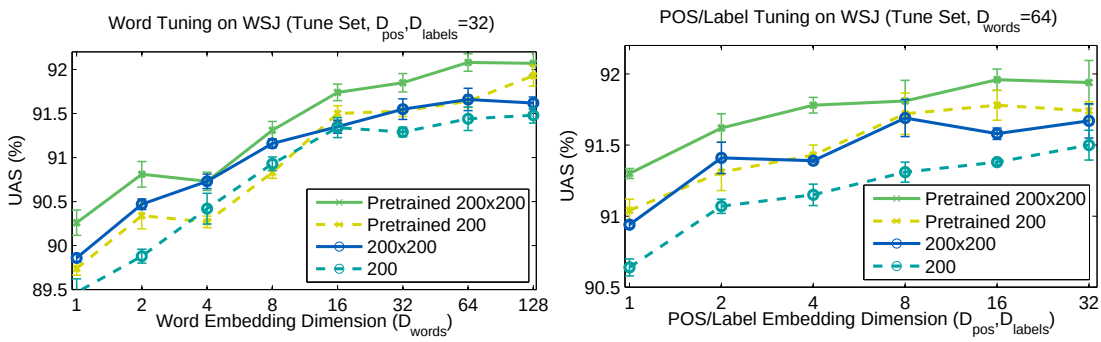

Figure 3: Effect of embedding dimensions on the WSJ tune set.

quired to reduce the bias of the model, but not when filtered through the softmax layer. Finally, we also experimented with connecting both hidden layers to the softmax layer during backpropagation training, but we found this did not significantly affect the performance of the greedy model.

5.3 Impact of Tri-Training

To evaluate the impact of the tri-training approach, we compared to up-training with the Berkely-Parser (Petrov et al., 2006) alone. The results are summarized in Figure 4 for the greedy and perceptron neural net models as well as our reimplementated Zhang and Nivre (2011) baseline.

For our neural network model, training on the output of the BerkeleyParser yields only modest gains, while training on the data where the two parsers agree produces significantly better results. This was especially pronounced for the greedy models: after tri-training, the greedy neural network model surpasses the BerkeleyParser in accuracy. It is also interesting to note that up-training improved results far more than tri-training for the baseline. We speculate that this is due to the a lack of diversity in the tri-training data for this model, since the same baseline model was intersected with the BerkeleyParser to generate the tri-training data.

5.4 Error Analysis

Regardless of tri-training, using the structured perceptron improved error rates on some of the common and difficult labels: `ROOT`, `ccomp`, `cc`, `conj`, and `nsubj` all improved by >1%. We inspected the learned perceptron weights $\mathbf{v}$ for the softmax probabilities $P(y)$ (see Appendix) and found that the perceptron reweights the softmax probabilities based on common confusions; e.g. a strong negative weight for the action RIGHT(`ccomp`) given the softmax model outputs RIGHT(`conj`). Note

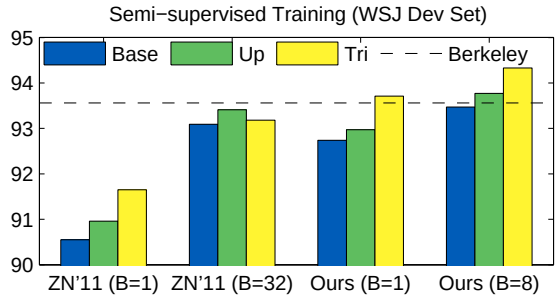

Figure 4: Semi-supervised training with 10^7 additional tokens, showing that tri-training gives significant improvements over up-training for our neural net model.

that this trend did not hold when $\phi(x, c) = [P(y)]$; without the hidden layer, the perceptron was not able to reweight the softmax probabilities to account for the greedy model's biases.

6 Conclusion

We presented a new state of the art in dependency parsing: a transition-based neural network parser trained with the structured perceptron and ASGD. We then combined this approach with unlabeled data and tri-training to further push state-of-the-art in semi-supervised dependency parsing. Nonetheless, our ablative analysis suggests that further gains are possible simply by scaling up our system to even larger representations. In future work, we will apply our method to other languages, explore end-to-end training of the system using structured learning, and scale up the method to larger datasets and network structures.

Acknowledgements

We would like to thank Bernd Bohnet for training his parsers and TurboParser on our setup. This paper benefitted tremendously from discussions with Ryan McDonald, Greg Coppola, Emily Pitler and Fernando Pereira. Finally, we are grateful to all members of the Google Parsing Team.

References

Bernd Bohnet and Jonas Kuhn. 2012. The best of both worlds: a graph-based completion model for transition-based parsers. In *Proc. EACL*, pages 77–87.

Bernd Bohnet. 2010. Top accuracy and fast dependency parsing is not a contradiction. In *Proc. 23rd International Conference on Computational Linguistics (Coling 2010)*, pages 89–97.

Léon Bottou. 2010. Large-scale machine learning with stochastic gradient descent. In *Proc. COMPSTAT*, pages 177–186.

Sabine Buchholz and Erwin Marsi. 2006. Conll-x shared task on multilingual dependency parsing. In *Proc. CoNLL*, pages 149–164.

Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, and Phillipp Koehn. 2013. One billion word benchmark for measuring progress in statistical language modeling. *CoRR*.

Danqi Chen and Christopher D. Manning. 2014. A fast and accurate dependency parser using neural networks. In *Proc. EMNLP*, pages 740–750.

Wenliang Chen, Min Zhang, and Yue Zhang. 2013. Semi-supervised feature transformation for dependency parsing. In *Proc. 2013 EMNLP*, pages 1303–1313.

Michael Collins and Brian Roark. 2004. Incremental parsing with the perceptron algorithm. In *Proc. ACL, Main Volume*, pages 111–118, Barcelona, Spain.

Marie-Catherine De Marneffe, Bill MacCartney, and Christopher D. Manning. 2006. Generating typed dependency parses from phrase structure parses. In *Proc. LREC*, pages 449–454.

Trinh Minh Tri Do and Thierry Artires. 2010. Neural conditional random fields. In *AISTATS*, volume 9, pages 177–184.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. 2015. Transition-based dependency parsing with stack long short-term memory. In *Proc. ACL*.

James Henderson. 2004. Discriminative training of a neural network statistical parser. In *Proc. ACL, Main Volume*, pages 95–102.

Geoffrey E. Hinton. 2012. A practical guide to training restricted boltzmann machines. In *Neural Networks: Tricks of the Trade (2nd ed.)*, Lecture Notes in Computer Science, pages 599–619. Springer.

Eduard Hovy, Mitchell Marcus, Martha Palmer, Lance Ramshaw, and Ralph Weischedel. 2006. Ontonotes: The 90% solution. In *Proc. HLT-NAACL*, pages 57–60.

Zhongqiang Huang and Mary Harper. 2009. Self-training PCFG grammars with latent annotations across languages. In *Proc. 2009 EMNLP*, pages 832–841, Singapore.

John Judge, Aoife Cahill, and Josef van Genabith. 2006. Questionbank: Creating a corpus of parse-annotated questions. In *Proc. ACL*, pages 497–504.

Terry Koo, Xavier Carreras, and Michael Collins. 2008. Simple semi-supervised dependency parsing. In *Proc. ACL-HLT*, pages 595–603.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Proc. NIPS*, pages 1097–1105.

Zhenghua Li, Min Zhang, and Wenliang Chen. 2014. Ambiguity-aware ensemble training for semi-supervised dependency parsing. In *Proc. ACL*, pages 457–467.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Andre Martins, Miguel Almeida, and Noah A. Smith. 2013. Turning on the turbo: Fast third-order non-projective turbo parsers. In *Proc. ACL*, pages 617–622.

David McClosky, Eugene Charniak, and Mark Johnson. 2006. Effective self-training for parsing. In *Proc. HLT-NAACL*, pages 152–159.

Ryan McDonald, Joakim Nivre, Yvonne Quirmbach-Brundage, Yoav Goldberg, Dipanjan Das, Kuzman Ganchev, Keith Hall, Slav Petrov, Hao Zhang, Oscar Täckström, Claudia Bedini, Núria Bertomeu Castelló, and Jungmee Lee. 2013. Universal dependency annotation for multilingual parsing. In *Proc. ACL*, pages 92–97.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781.

Vinod Nair and Geoffrey E. Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *Proc. 27th ICML*, pages 807–814.

Joakim Nivre, Johan Hall, Sandra Kübler, Ryan McDonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret. 2007. The CoNLL 2007 shared task on dependency parsing. In *Proc. CoNLL*, pages 915–932.

Joakim Nivre. 2004. Incrementality in deterministic dependency parsing. In *Proc. ACL Workshop on Incremental Parsing*, pages 50–57.

Joakim Nivre. 2008. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34(4):513–553.

Jian Peng, Liefeng Bo, and Jinbo Xu. 2009. Conditional neural fields. In *Proc. NIPS*, pages 1419–1427.

Slav Petrov and Ryan McDonald. 2012. Overview of the 2012 shared task on parsing the web. Notes of the First Workshop on Syntactic Analysis of Non-Canonical Language (SANCL).

Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. 2006. Learning accurate, compact, and interpretable tree annotation. In *Proc. ACL*, pages 433–440.

Slav Petrov, Pi-Chuan Chang, Michael Ringgaard, and Hiyan Alshawi. 2010. Uptraining for accurate deterministic question parsing. In *Proc. EMNLP*, pages 705–713.

Adwait Ratnaparkhi. 1997. A linear observed time statistical parser based on maximum entropy models. In *Proc. EMNLP*, pages 1–10.

Jun Suzuki, Hideki Isozaki, Xavier Carreras, and Michael Collins. 2009. An empirical study of semi-supervised structured conditional models for dependency parsing. In *Proc. 2009 EMNLP*, pages 551–560.

Jun Suzuki, Hideki Isozaki, and Masaaki Nagata. 2011. Learning condensed feature representations from large unsupervised data sets for supervised learning. In *Proc. ACL-HLT*, pages 636–641.

Ivan Titov and James Henderson. 2007. Fast and robust multilingual dependency parsing with a generative latent variable model. In *Proc. EMNLP*, pages 947–951.

Ivan Titov and James Henderson. 2010. A latent variable model for generative dependency parsing. In *Trends in Parsing Technology*, pages 35–55. Springer.

Kristina Toutanova, Dan Klein, Christopher D. Manning, and Yoram Singer. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *NAACL*.

Oriol Vinyals, Lukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey E. Hinton. 2015. Grammar as a foreign language. *arXiv:1412.7449*.

Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proc. IWPT*, pages 195–206.

Matthew D. Zeiler, Marc'Aurelio Ranzato, Rajat Monga, Mark Z. Mao, K. Yang, Quoc Viet Le, Patrick Nguyen, Andrew W. Senior, Vincent Vanhoucke, Jeffrey Dean, and Geoffrey E. Hinton. 2013. On rectified linear units for speech processing. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 3517–3521.

Yue Zhang and Stephen Clark. 2008. A tale of two parsers: investigating and combining graph-based and transition-based dependency parsing using beam-search. In *Proc. EMNLP*, pages 562–571.

Hao Zhang and Ryan McDonald. 2014. Enforcing structural diversity in cube-pruned dependency parsing. In *Proc. ACL*, pages 656–661.

Yue Zhang and Joakim Nivre. 2011. Transition-based dependency parsing with rich non-local features. In *Proc. ACL-HLT*, pages 188–193.

Hao Zhou, Yue Zhang, and Jiajun Chen. 2015. A neural probabilistic structured-prediction model for transition-based dependency parsing. In *Proc. ACL*.