Neural CRF Parsing

Greg Durrett and **Dan Klein**
Computer Science Division
University of California, Berkeley
{gdurrett,klein}@cs.berkeley.edu

Abstract

This paper describes a parsing model that combines the exact dynamic programming of CRF parsing with the rich nonlinear featurization of neural net approaches. Our model is structurally a CRF that factors over anchored rule productions, but instead of linear potential functions based on sparse features, we use nonlinear potentials computed via a feedforward neural network. Because potentials are still local to anchored rules, structured inference (CKY) is unchanged from the sparse case. Computing gradients during learning involves backpropagating an error signal formed from standard CRF sufficient statistics (expected rule counts). Using only dense features, our neural CRF already exceeds a strong baseline CRF model (Hall et al., 2014). In combination with sparse features, our system[1] achieves 91.1 F_1 on section 23 of the Penn Treebank, and more generally outperforms the best prior single parser results on a range of languages.

1 Introduction

Neural network-based approaches to structured NLP tasks have both strengths and weaknesses when compared to more conventional models, such conditional random fields (CRFs). A key strength of neural approaches is their ability to learn nonlinear interactions between underlying features. In the case of unstructured output spaces, this capability has led to gains in problems ranging from syntax (Chen and Manning, 2014; Belinkov et al., 2014) to lexical semantics (Kalchbrenner et al., 2014; Kim, 2014). Neural methods are also powerful tools in the case of structured output spaces. Here, past work has often relied on recurrent architectures (Henderson, 2003; Socher et al., 2013; İrsoy and Cardie, 2014), which can propagate information through structure via real-valued hidden state, but as a result do not admit efficient dynamic programming (Socher et al., 2013; Le and Zuidema, 2014). However, there is a natural marriage of nonlinear induced features and efficient structured inference, as explored by Collobert et al. (2011) for the case of sequence modeling: feedforward neural networks can be used to score local decisions which are then "reconciled" in a discrete structured modeling framework, allowing inference via dynamic programming.

In this work, we present a CRF constituency parser based on these principles, where individual anchored rule productions are scored based on nonlinear features computed with a feedforward neural network. A separate, identically-parameterized replicate of the network exists for each possible span and split point. As input, it takes vector representations of words at the split point and span boundaries; it then outputs scores for anchored rules applied to that span and split point. These scores can be thought of as nonlinear potentials analogous to linear potentials in conventional CRFs. Crucially, while the network replicates are connected in a unified model, their computations factor along the same substructures as in standard CRFs.

Prior work on parsing using neural network models has often sidestepped the problem of structured inference by making sequential decisions (Henderson, 2003; Chen and Manning, 2014; Tsuboi, 2014) or by doing reranking (Socher et al., 2013; Le and Zuidema, 2014); by contrast, our framework permits exact inference via CKY, since the model's structured interactions are purely discrete and do not involve continuous hidden state. Therefore, we can exploit a neural net's capacity to learn nonlinear features without modifying

[1]System available at `http://nlp.cs.berkeley.edu`

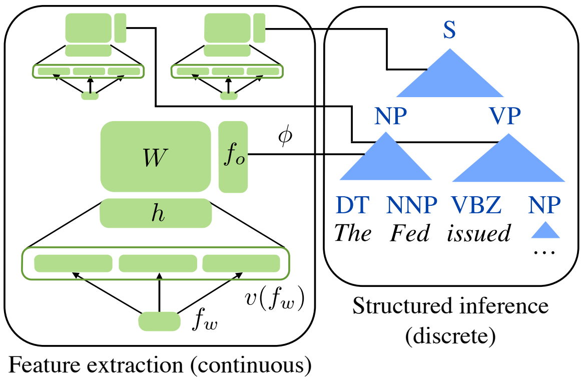
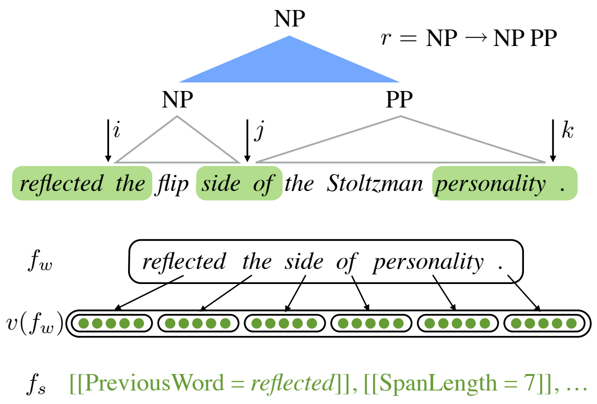


Figure 1: Neural CRF model. On the right, each anchored rule (r, s) in the tree is independently scored by a function ϕ, so we can perform inference with CKY to compute marginals or the Viterbi tree. On the left, we show the process for scoring an anchored rule with neural features: words in f_w (see Figure 2) are embedded, then fed through a neural network with one hidden layer to compute dense intermediate features, whose conjunctions with sparse rule indicator features f_o are scored according to parameters W.



NP
r = NP → NP PP
NP
PP
i
j
k
reflected the flip side of the Stoltzman personality .
f_w
reflected the side of personality .
$v(f_w)$
f_s [[PreviousWord = reflected]], [[SpanLength = 7]], …

Figure 2: Example of an anchored rule production for the rule NP → NP PP. From the anchoring $s = (i, j, k)$, we extract either sparse surface features f_s or a sequence of word indicators f_w which are embedded to form a vector representation $v(f_w)$ of the anchoring's lexical properties.

our core inference mechanism, allowing us to use tricks like coarse pruning that make inference efficient in the purely sparse model. Our model can be trained by gradient descent exactly as in a conventional CRF, with the gradient of the network parameters naturally computed by backpropagating a difference of expected anchored rule counts through the network for each span and split point.

Using dense learned features alone, the neural CRF model obtains high performance, outperforming the CRF parser of Hall et al. (2014). When sparse indicators are used in addition, the resulting model gets 91.1 F_1 on section 23 of the Penn Treebank, outperforming the parser of Socher et al. (2013) as well as the Berkeley Parser (Petrov and Klein, 2007) and matching the discriminative parser of Carreras et al. (2008). The model also obtains the best single parser results on nine other languages, again outperforming the system of Hall et al. (2014).

2 Model

Figure 1 shows our neural CRF model. The model decomposes over anchored rules, and it scores each of these with a potential function; in a standard CRF, these potentials are typically linear functions of sparse indicator features, whereas in our approach they are nonlinear functions of word embeddings.[2] Section 2.1 describes our notation for anchored rules, and Section 2.2 talks about how they are scored. We then discuss specific choices of our featurization (Section 2.3) and the backbone grammar used for structured inference (Section 2.4).

2.1 Anchored Rules

The fundamental units that our parsing models consider are *anchored rules*. As shown in Figure 2, we define an anchored rule as a tuple (r, s), where r is an indicator of the rule's identity and $s = (i, j, k)$ indicates the span (i, k) and split point j of the rule.[3] A tree T is simply a collection of anchored rules subject to the constraint that those rules form a tree. All of our parsing models are CRFs that decompose over anchored rule productions and place a probability distribution over trees conditioned on a sentence $\mathbf{w}$ as follows:

$$P(T|\mathbf{w}) \propto \exp\left(\sum_{(r,s)\in T} \phi(\mathbf{w}, r, s)\right)$$

[2]Throughout this work, we will primarily consider two potential functions: linear functions of sparse indicators and nonlinear neural networks over dense, continuous features. Although other modeling choices are possible, these two points in the design space reflect common choices in NLP, and past work has suggested that nonlinear functions of indicators or linear functions of dense features may perform less well (Wang and Manning, 2013).

[3]For simplicity of exposition, we ignore unary rules; however, they are easily supported in this framework by simply specifying a null value for the split point.

where ϕ is a scoring function that considers the input sentence and the anchored rule in question. Figure 1 shows this scoring process schematically. As we will see, the module on the left can be be a neural net, a linear function of surface features, or a combination of the two, as long as it provides anchored rule scores, and the structured inference component is the same regardless (CKY).

A PCFG estimated with maximum likelihood has $\phi(\mathbf{w}, r, s) = \log P(r|\text{parent}(r))$, which is independent of the anchoring s and the words $\mathbf{w}$ except for preterminal productions; a basic discriminative parser might let this be a learned parameter but still disregard the surface information. However, surface features can capture useful syntactic cues (Finkel et al., 2008; Hall et al., 2014). Consider the example in Figure 2: the proposed parent NP is preceded by the word *reflected* and followed by a period, which is a surface context characteristic of NPs or PPs in object position. Beginning with *the* and ending with *personality* are typical properties of NPs as well, and the choice of the particular rule NP $\rightarrow$ NP PP is supported by the fact that the proposed child PP begins with *of*. This information can be captured with sparse features (f_s in Figure 2) or, as we describe below, with a neural network taking lexical context as input.

2.2 Scoring Anchored Rules

Following Hall et al. (2014), our baseline sparse scoring function takes the following bilinear form:

$$\phi_{\text{sparse}}(\mathbf{w}, r, s; W) = f_s(\mathbf{w}, s)^\top W f_o(r)$$

where $f_o(r) \in \{0,1\}^{n_o}$ is a sparse vector of features expressing properties of r (such as the rule's identity or its parent label) and $f_s(\mathbf{w}, s) \in \{0,1\}^{n_s}$ is a sparse vector of surface features associated with the words in the sentence and the anchoring, as shown in Figure 2. W is a $n_s \times n_o$ matrix of weights.[4] The scoring of a particular anchored rule is depicted in Figure 3a; note that surface features and rule indicators are conjoined in a systematic way.

The role of f_s can be equally well played by a vector of dense features learned via a neural net-

[4]A more conventional expression of the scoring function for a CRF is $\phi(\mathbf{w}, r, s) = \theta^\top f(\mathbf{w}, r, s)$, with a vector θ for the parameters and a single feature extractor f that jointly inspects the surface and the rule. However, when the feature representation conjoins each rule r with surface properties of the sentence in a systematic way (an assumption that holds in our case as well as for standard CRF models for POS tagging and NER), this is equivalent to our formalism.

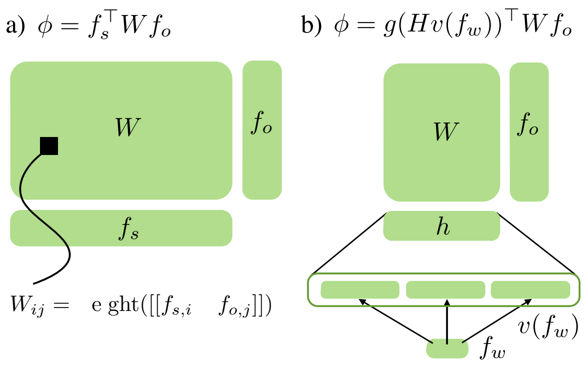

a) $\phi = f_s^\top W f_o$
b) $\phi = g(Hv(f_w))^\top W f_o$
W
f_o
f_s
W_{ij} = e ght([[$f_{s,i}$ $f_{o,j}$]])
W
f_o
h
$v(f_w)$
f_w

Figure 3: Our sparse (left) and neural (right) scoring functions for CRF parsing. f_s and f_w are raw surface feature vectors for the sparse and neural models (respectively) extracted over anchored spans with split points. (a) In the sparse case, we multiply f_s by a weight matrix W and then a sparse output vector f_o to score the rule production. (b) In the neural case, we first embed f_w and then transform it with a one-layer neural network in order to produce an intermediate feature representation h before combining with W and f_o.

work. We will now describe how to compute these features, which represent a transformation of surface lexical indicators f_w. Define $f_w(\mathbf{w}, s) \in \mathbb{N}^{n_w}$ to be a function that produces a fixed-length sequence of word indicators based on the input sentence and the anchoring. This vector of word identities is then passed to an embedding function $v : \mathbb{N} \rightarrow \mathbb{R}^{n_e}$ and the dense representations of the words are subsequently concatenated to form a vector we denote by $v(f_w)$.[5] Finally, we multiply this by a matrix $H \in \mathbb{R}^{n_h \times (n_w n_e)}$ of real-valued parameters and pass it through an elementwise nonlinearity $g(\cdot)$. We use rectified linear units $g(x) = \max(x, 0)$ and discuss this choice more in Section 6.

Replacing f_s with the end result of this computation $h(\mathbf{w}, s; H) = g(Hv(f_w(\mathbf{w}, s)))$, our scoring function becomes

$$\phi_{\text{neural}}(\mathbf{w}, r, s; H, W) = h(\mathbf{w}, s; H)^\top W f_o(r)$$

as shown in Figure 3b. For a fixed H, this model can be viewed as a basic CRF with dense input features. By learning H, we learn intermediate feature representations that provide the model with

[5]Embedding words allows us to use standard pre-trained vectors more easily and tying embeddings across word positions substantially reduces the number of model parameters. However, embedding features rather than words has also been shown to be effective (Chen et al., 2014).

more discriminating power. Also note that it is possible to use deeper networks or more sophisticated architectures here; we will return to this in Section 6.

Our two models can be easily combined:

$$\phi(\mathbf{w}, r, s; W_1, H, W_2) = \phi_{\text{sparse}}(\mathbf{w}, r, s; W_1) + \phi_{\text{neural}}(\mathbf{w}, r, s; H, W_2)$$

Weights for each component of the scoring function can be learned fully jointly and inference proceeds as before.

2.3 Features

We take f_s to be the set of features described in Hall et al. (2014). At the preterminal layer, the model considers prefixes and suffixes up to length 5 of the current word and neighboring words, as well as the words' identities. For nonterminal productions, we fire indicators on the words[6] before and after the start, end, and split point of the anchored rule (as shown in Figure 2) as well as on two other span properties, span length and span shape (an indicator of where capitalized words, numbers, and punctuation occur in the span).

For our neural model, we take f_w for all productions (preterminal and nonterminal) to be the words surrounding the beginning and end of a span and the split point, as shown in Figure 2; in particular, we look two words in either direction around each point of interest, meaning the neural net takes 12 words as input.[7] For our word embeddings v, we use pre-trained word vectors from Bansal et al. (2014). We compare with other sources of word vectors in Section 5. Contrary to standard practice, we do not update these vectors during training; we found that doing so did not provide an accuracy benefit and slowed down training considerably.

2.4 Grammar Refinements

A recurring issue in discriminative constituency parsing is the granularity of annotation in the base grammar (Finkel et al., 2008; Petrov and Klein, 2008; Hall et al., 2014). Using finer-grained symbols in our rules r gives the model greater capacity, but also introduces more parameters into W and increases the ability to overfit. Following Hall et al. (2014), we use grammars with very little annotation: we use no horizontal Markovization for any of experiments, and all of our English experiments with the neural CRF use no vertical Markovization ($V = 0$). This also has the benefit of making the system much faster, due to the smaller state space for dynamic programming. We do find that using parent annotation ($V = 1$) is useful on other languages (see Section 7.2), but this is the only grammar refinement we consider.

3 Learning

To learn weights for our neural model, we maximize the conditional log likelihood of our D training trees T^*:

$$\mathcal{L}(H, W) = \sum_{i=1}^{D} \log P(T_i^* | \mathbf{w}_i; H, W)$$

Because we are using rectified linear units as our nonlinearity, our objective is not everywhere differentiable. The interaction of the parameters and the nonlinearity also makes the objective nonconvex. However, in spite of this, we can still follow subgradients to optimize this objective, as is standard practice.

Recall that $h(\mathbf{w}, s; H)$ are the hidden layer activations. The gradient of W takes the standard form of log-linear models:

$$\frac{\partial \mathcal{L}}{\partial W} = \left(\sum_{(r,s) \in T^*} h(\mathbf{w}, s; H) f_o(r)^\top \right) - \left(\sum_{T} P(T|\mathbf{w}; H, W) \sum_{(r,s) \in T} h(\mathbf{w}, s; H) f_o(r)^\top \right)$$

Note that the outer products give matrices of feature counts isomorphic to W. The second expression can be simplified to be in terms of expected feature counts. To update H, we use standard backpropagation by first computing:

$$\frac{\partial \mathcal{L}}{\partial h} = \left(\sum_{(r,s) \in T^*} W f_o(r) \right) - \left(\sum_{T} P(T|\mathbf{w}; H, W) \sum_{(r,s) \in T} W f_o(r) \right)$$

Since h is the output of the neural network, we can then apply the chain rule to compute gradients for H and any other parameters in the neural network.

[6]The model actually uses the longest suffix of each word occurring at least 100 times in the training set, up to the entire word. Removing this abstraction of rare words harms performance.

[7]The sparse model did not benefit from using this larger neighborhood, so improvements from the neural net are not simply due to considering more lexical context.

Learning uses Adadelta (Zeiler, 2012), which has been employed in past work (Kim, 2014). We found that Adagrad (Duchi et al., 2011) performed equally well with tuned regularization and step size parameters, but Adadelta worked better out of the box. We set the momentum term $\rho = 0.95$ (as suggested by Zeiler (2012)) and did not regularize the weights at all. We used a minibatch size of 200 trees, although the system was not particularly sensitive to this. For each treebank, we trained for either 10 passes through the treebank or 1000 minibatches, whichever is shorter.

We initialized the output weight matrix W to zero. To break symmetry, the lower level neural network parameters H were initialized with each entry being independently sampled from a Gaussian with mean 0 and variance 0.01; Gaussian performed better than uniform initialization, but the variance was not important.

4 Inference

Our baseline and neural model both score anchored rule productions. We can use CKY in the standard fashion to compute either expected anchored rule counts $\mathbb{E}_{P(T|\mathbf{w})}[(r, s)]$ or the Viterbi tree $\arg\max_T P(T|\mathbf{w})$.

We speed up inference by using a coarse pruning pass. We follow Hall et al. (2014) and prune according to an X-bar grammar with head-outward binarization, ruling out any constituent whose max marginal probability is less than e^{-9}. With this pruning, the number of spans and split points to be considered is greatly reduced; however, we still need to compute the neural network activations for each remaining span and split point, of which there may be thousands for a given sentence.[8] We can improve efficiency further by noting that the same word will appear in the same position in a large number of span/split point combinations, and cache the contribution to the hidden layer caused by that word (Chen and Manning, 2014). Computing the hidden layer then simply requires adding n_w vectors together and applying the nonlinearity, instead of a more costly matrix multiply.

Because the number of rule indicators n_o is fairly large (approximately 4000 in the Penn Treebank), the multiplication by W in the model is also expensive. However, because only a small number of rules can apply to a given span and split point, f_o is sparse and we can selectively compute the terms necessary for the final bilinear product.

Our combined sparse and neural model trains on the Penn Treebank in 24 hours on a single machine with a parallelized CPU implementation. For reference, the purely sparse model with a parent-annotated grammar (necessary for the best results) takes around 15 hours on the same machine.

5 System Ablations

Table 1 shows results on section 22 (the development set) of the English Penn Treebank (Marcus et al., 1993), computed using `evalb`. Full test results and comparisons to other systems are shown in Table 4. We compare variants of our system along two axes: whether they use standard linear sparse features, nonlinear dense features from the neural net, or both, and whether any word representations (vectors or clusters) are used.

Sparse vs. neural The neural CRF (line (d) in Table 1) on its own outperforms the sparse CRF (a, b) even when the sparse CRF has a more heavily annotated grammar. This is a surprising result: the features in the sparse CRF have been carefully engineered to capture a range of linguistic phenomena (Hall et al., 2014), and there is no guarantee that word vectors will capture the same. For example, at the POS tagging layer, the sparse model looks at prefixes and suffixes of words, which give the model access to morphology for predicting tags of unknown words, which typically have regular inflection patterns. By contrast, the neural model must rely on the geometry of the vector space exposing useful regularities. At the same time, the strong performance of the combination of the two systems (g) indicates that not only are both featurization approaches high-performing on their own, but that they have complementary strengths.

Unlabeled data Much attention has been paid to the choice of word vectors for various NLP tasks, notably whether they capture more syntactic or semantic phenomena (Bansal et al., 2014; Levy and Goldberg, 2014). We primarily use vectors from Bansal et al. (2014), who train the skip-gram model of Mikolov et al. (2013) using contexts from dependency links; a similar approach was also suggested by Levy and Goldberg (2014).

[8]One reason we did not choose to include the rule identity f_o as an input to the network is that it requires computing an even larger number of network activations, since we cannot reuse them across rules over the same span and split point.

	Sparse	Neural	V	Word Reps	F_1 len ≤ 40	F_1 all
	Hall et al. (2014), $V = 1$				90.5	
a	✓		0		89.89	89.22
b	✓		1		90.82	90.13
c	✓		1	Brown	90.80	90.17
d		✓	0	Bansal	90.97	90.44
e		✓	0	Collobert	90.25	89.63
f		✓	0	PTB	89.34	88.99
g	✓	✓	0	Bansal	**92.04**	**91.34**
h	✓	✓	0	PTB	91.39	90.91

Table 1: Results of our sparse CRF, neural CRF, and combined parsing models on section 22 of the Penn Treebank. Systems are broken down by whether local potentials come from sparse features and/or the neural network (the primary contribution of this work), their level of vertical Markovization, and what kind of word representations they use. The neural CRF (d) outperforms the sparse CRF (a, b) even when a more heavily annotated grammar is used, and the combined approach (g) is substantially better than either individual model. The contribution of the neural architecture cannot be replaced by Brown clusters (c), and even word representations learned just on the Penn Treebank are surprisingly effective (f, h).

		F_1 len ≤ 40	Δ
Neural CRF		90.97	—
Nonlinearity	ReLU	90.97	—
	Tanh	90.74	-0.23
	Cube	89.94	-1.03
Depth	0 HL	90.54	-0.43
	1 HL	90.97	—
	2 HL	90.58	-0.39
Embed output		88.81	-2.16

Table 2: Exploration of other implementation choices in the feedforward neural network on sentences of length ≤ 40 from section 22 of the Penn Treebank. Rectified linear units perform better than tanh or cubic units, a network with one hidden layer performs best, and embedding the output feature vector gives worse performance.

However, as these embeddings are trained on a relatively small corpus (BLLIP minus the Penn Treebank), it is natural to wonder whether less-syntactic embeddings trained on a larger corpus might be more useful. This is not the case: line (e) in Table 1 shows the performance of the neural CRF using the Wikipedia-trained word embeddings of Collobert et al. (2011), which do not perform better than the vectors of Bansal et al. (2014).

To isolate the contribution of continuous word representations themselves, we also experimented with vectors trained on just the text from the training set of the Penn Treebank using the skip-gram model with a window size of 1. While these vectors are somewhat lower performing on their own (f), they still provide a surprising and noticeable gain when stacked on top of sparse features (h), again suggesting that dense and sparse representations have complementary strengths. This result also reinforces the notion that the utility of word vectors does *not* come primarily from importing information about out-of-vocabulary words (Andreas and Klein, 2014).

Since the neural features incorporate information from unlabeled data, we should provide the sparse model with similar information for a true apples-to-apples comparison. Brown clusters have been shown to be effective vehicles in the past (Koo et al., 2008; Turian et al., 2010; Bansal et al., 2014). We can incorporate Brown clusters into the baseline CRF model in an analogous way to how embedding features are used in the dense model: surface features are fired on Brown cluster identities (we use prefixes of length 4 and 10) of key words. We use the Brown clusters from Koo et al. (2008), which are trained on the same data as the vectors of Bansal et al. (2014). However, Table 1 shows that these features provide no benefit to the baseline model, which suggests either that it is difficult to learn reliable weights for these as sparse features or that different regularities are being captured by the word embeddings.

6 Design Choices

The neural net design space is large, so we wish to analyze the particular design choices we made for this system by examining the performance of several variants of the neural net architecture used in our system. Table 2 shows development results from potential alternate architectural choices, which we now discuss.

Choice of nonlinearity The choice of nonlinearity g has been frequently discussed in the neural network literature. Our choice $g(x) = \max(x, 0)$, a rectified linear unit, is increasingly popular in computer vision (Krizhevsky et al., 2012). $g(x) = \tanh(x)$ is a traditional nonlinearity widely used throughout the history of neural nets (Bengio et al., 2003). $g(x) = x^3$ (cube) was found to be most successful by Chen and Manning (2014).

Table 2 compares the performance of these three nonlinearities. We see that rectified linear units perform the best, followed by tanh units, followed by cubic units.[9] One drawback of tanh as an activation function is that it is easily "saturated" if the input to the unit is too far away from zero, causing the backpropagation of derivatives through that unit to essentially cease; this is known to cause problems for training, requiring special purpose machinery for use in deep networks (Ioffe and Szegedy, 2015).

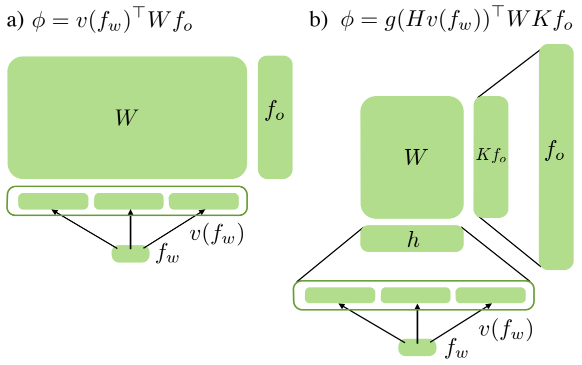
Depth Given that we are using rectified linear units, it bears asking whether or not our implementation is improving substantially over linear features of the continuous input. We can use the embedding vector of an anchored span $v(f_w)$ directly as input to a basic linear CRF, as shown in Figure 4a. Table 1 shows that the purely linear architecture (0 HL) performs surprisingly well, but is still less effective than the network with one hidden layer. This agrees with the results of Wang and Manning (2013), who noted that dense features typically benefit from nonlinear modeling. We also compare against a two-layer neural network, but find that this also performs worse than the one-layer architecture.

Densifying output features Overall, it appears beneficial to use dense representations of surface features; a natural question that one might ask is whether the same technique can be applied to the sparse output feature vector f_o. We can apply the approach of Srikumar and Manning (2014) and multiply the sparse output vector by a dense matrix K, giving the following scoring function (shown in Figure 4b):

$$\phi(\mathbf{w}, r, s; H, W, K) = g(Hv(f_w(\mathbf{w}, s)))^\top W K f_o(r)$$

where W is now $n_h \times n_{oe}$ and K is $n_{oe} \times n_o$. WK can be seen a low-rank approximation of the original W at the output layer, similar to low-rank factorizations of parameter matrices used in past

[9]The performance of cube decreased substantially late in learning; it peaked at around 90.52. Dropout may be useful for alleviating this type of overfitting, but in our experiments we did not find dropout to be beneficial overall.



a) $\phi = v(f_w)^\top W f_o$
b) $\phi = g(Hv(f_w))^\top W K f_o$
W
f_o
$v(f_w)$
f_w
W
Kf_o
f_o
h
$v(f_w)$
f_w

Figure 4: Two additional forms of the scoring function. a) Linear version of the dense model, equivalent to a CRF with continuous-valued input features. b) Version of the dense model where outputs are also embedded according to a learned matrix K.

work (Lei et al., 2014). This approach saves us from having to learn a separate row of W for every rule in the grammar; if rules are given similar embeddings, then they will behave similarly according to the model.

We experimented with $n_{oe} = 20$ and show the results in Table 2. Unfortunately, this approach does not seem to work well for parsing. Learning the output representation was empirically very unstable, and it also required careful initialization. We tried Gaussian initialization (as in the rest of our model) and initializing the model by clustering rules either randomly or according to their parent symbol. The latter is what is shown in the table, and gave substantially better performance. We hypothesize that blurring distinctions between output classes may harm the model's ability to differentiate between closely-related symbols, which is required for good parsing performance. Using pretrained rule embeddings at this layer might also improve performance of this method.

7 Test Results

We evaluate our system under two conditions: first, on the English Penn Treebank, and second, on the nine languages used in the SPMRL 2013 and 2014 shared tasks.

7.1 Penn Treebank

Table 4 reports results on section 23 of the Penn Treebank (PTB). We focus our comparison on single parser systems as opposed to rerankers, ensembles, or self-trained methods (though these are also mentioned for context). First, we compare against

	Arabic	Basque	French	German	Hebrew	Hungarian	Korean	Polish	Swedish	Avg
Dev, all lengths										
Hall et al. (2014)	78.89	83.74	79.40	83.28	88.06	87.44	81.85	91.10	75.95	83.30
This work*	**80.68**	**84.37**	**80.65**	**85.25**	**89.37**	**89.46**	**82.35**	**92.10**	**77.93**	**84.68**
Test, all lengths										
Berkeley	79.19	70.50	80.38	78.30	86.96	81.62	71.42	79.23	79.18	78.53
Berkeley-Tags	78.66	74.74	79.76	78.28	85.42	85.22	78.56	86.75	80.64	80.89
Crabbé and Seddah (2014)	77.66	85.35	79.68	77.15	86.19	87.51	79.35	91.60	82.72	83.02
Hall et al. (2014)	78.75	83.39	79.70	78.43	87.18	88.25	80.18	90.66	82.00	83.17
This work*	**80.24**	**85.41**	**81.25**	**80.95**	**88.61**	**90.66**	**82.23**	**92.97**	**83.45**	**85.08**
Reranked ensemble										
2014 Best	**81.32**	**88.24**	**82.53**	**81.66**	**89.80**	**91.72**	**83.81**	**90.50**	**85.50**	**86.12**

Table 3: Results for the nine treebanks in the SPMRL 2013/2014 Shared Tasks; all values are F-scores for sentences of all lengths using the version of `evalb` distributed with the shared task. Our parser substantially outperforms the strongest single parser results on this dataset (Hall et al., 2014; Crabbé and Seddah, 2014). Berkeley-Tags is an improved version of the Berkeley parser designed for the shared task (Seddah et al., 2013). 2014 Best is a reranked ensemble of modified Berkeley parsers and constitutes the best published numbers on this dataset (Björkelund et al., 2013; Björkelund et al., 2014).

	F_1 all
Single model, PTB only	
Hall et al. (2014)	89.2
Berkeley	90.1
Carreras et al. (2008)	91.1
Shindo et al. (2012) single	91.1
Single model, PTB + vectors/clusters	
Zhu et al. (2013)	91.3
This work*	91.1
Extended conditions	
Charniak and Johnson (2005)	91.5
Socher et al. (2013)	90.4
Vinyals et al. (2014) single	90.5
Vinyals et al. (2014) ensemble	91.6
Shindo et al. (2012) ensemble	92.4

Table 4: Test results on section 23 of the Penn Treebank. We compare to several categories of parsers from the literatures. We outperform strong baselines such as the Berkeley Parser (Petrov and Klein, 2007) and the CVG Stanford parser (Socher et al., 2013) and we match the performance of sophisticated generative (Shindo et al., 2012) and discriminative (Carreras et al., 2008) parsers.

four parsers trained only on the PTB with no auxiliary data: the CRF parser of Hall et al. (2014), the Berkeley parser (Petrov and Klein, 2007), the discriminative parser of Carreras et al. (2008), and the single TSG parser of Shindo et al. (2012). To our knowledge, the latter two systems are the highest performing in this PTB-only, single parser data condition; we match their performance at 91.1 F_1, though we also use word vectors computed from unlabeled data. We further compare to the shift-reduce parser of Zhu et al. (2013), which uses unlabeled data in the form of Brown clusters. Our method achieves performance close to that of their parser.

We also compare to the compositional vector grammar (CVG) parser of Socher et al. (2013) as well as the LSTM-based parser of Vinyals et al. (2014). The conditions these parsers are operating under are slightly different: the former is a reranker on top of the Stanford Parser (Klein and Manning, 2003) and the latter trains on much larger amounts of data parsed by a product of Berkeley parsers (Petrov, 2010). Regardless, we outperform the CVG parser as well as the single parser results from Vinyals et al. (2014).

7.2 SPMRL

We also examine the performance of our parser on other languages, specifically the nine morphologically-rich languages used in the SPMRL 2013/2014 shared tasks (Seddah et al., 2013; Seddah et al., 2014). We train word vectors on the monolingual data distributed with the SPMRL 2014 shared task (typically 100M-200M tokens per language) using the skip-gram approach of `word2vec` with a window size of 1

(Mikolov et al., 2013).[10] Here we use $V = 1$ in the backbone grammar, which we found to be beneficial overall. Table 3 shows that our system improves upon the performance of the parser from Hall et al. (2014) as well as the top single parser from the shared task (Crabbé and Seddah, 2014), with robust improvements on all languages.

8 Conclusion

In this work, we presented a CRF parser that scores anchored rule productions using dense input features computed from a feedforward neural net. Because the neural component is modularized, we can easily integrate it into a preexisting learning and inference framework based around dynamic programming of a discrete parse chart. Our combined neural and sparse model gives strong performance both on English and on other languages.

Our system is publicly available at `http://nlp.cs.berkeley.edu`.

Acknowledgments

This work was partially supported by BBN under DARPA contract HR0011-12-C-0014, by a Facebook fellowship for the first author, and by a Google Faculty Research Award to the second author. Thanks to David Hall for assistance with the Epic parsing framework and for a preliminary implementation of the neural architecture, to Kush Rastogi for training word vectors on the SPMRL data, to Dan Jurafsky for helpful discussions, and to the anonymous reviewers for their insightful comments.

References

Jacob Andreas and Dan Klein. 2014. How much do word embeddings encode about syntax? In *Proceedings of the Association for Computational Linguistics*.

Mohit Bansal, Kevin Gimpel, and Karen Livescu. 2014. Tailoring Continuous Word Representations for Dependency Parsing. In *Proceedings of the Association for Computational Linguistics*.

Yonatan Belinkov, Tao Lei, Regina Barzilay, and Amir Globerson. 2014. Exploring Compositional Architectures and Word Vector Representations for Prepositional Phrase Attachment. *Transactions of the Association for Computational Linguistics*, 2:561–572.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. 2003. A Neural Probabilistic Language Model. *Journal of Machine Learning Research*, 3:1137–1155, March.

Anders Björkelund, Ozlem Cetinoglu, Richárd Farkas, Thomas Mueller, and Wolfgang Seeker. 2013. (Re)ranking Meets Morphosyntax: State-of-the-art Results from the SPMRL 2013 Shared Task. In *Proceedings of the Fourth Workshop on Statistical Parsing of Morphologically-Rich Languages*.

Anders Björkelund, Özlem Çetinoğlu, Agnieszka Faleńska, Richárd Farkas, Thomas Mueller, Wolfgang Seeker, and Zsolt Szántó. 2014. Introducing the IMS-Wrocław-Szeged-CIS entry at the SPMRL 2014 Shared Task: Reranking and Morpho-syntax meet Unlabeled Data. In *Proceedings of the First Joint Workshop on Statistical Parsing of Morphologically Rich Languages and Syntactic Analysis of Non-Canonical Languages*.

Xavier Carreras, Michael Collins, and Terry Koo. 2008. TAG, Dynamic Programming, and the Perceptron for Efficient, Feature-rich Parsing. In *Proceedings of the Conference on Computational Natural Language Learning*.

Eugene Charniak and Mark Johnson. 2005. Coarse-to-Fine n-Best Parsing and MaxEnt Discriminative Reranking. In *Proceedings of the Association for Computational Linguistics*.

Danqi Chen and Christopher D Manning. 2014. A Fast and Accurate Dependency Parser using Neural Networks. In *Proceedings of Empirical Methods in Natural Language Processing*.

Wenliang Chen, Yue Zhang, and Min Zhang. 2014. Feature Embedding for Dependency Parsing. In *Proceedings of the International Conference on Computational Linguistics*.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural Language Processing (Almost) from Scratch. *Journal of Machine Learning Research*, 12:2493–2537.

Benoit Crabbé and Djamé Seddah. 2014. Multilingual Discriminative Shift-Reduce Phrase Structure Parsing for the SPMRL 2014 Shared Task. In *Proceedings of the First Joint Workshop on Statistical Parsing of Morphologically Rich Languages and Syntactic Analysis of Non-Canonical Languages*.

John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, 12:2121–2159, July.

Jenny Rose Finkel, Alex Kleeman, and Christopher D. Manning. 2008. Efficient, Feature-based, Conditional Random Field Parsing. In *Proceedings of the Association for Computational Linguistics*.

[10]Training vectors with the SKIP$_{\text{DEP}}$ method of Bansal et al. (2014) did not substantially improve performance here.

David Hall, Greg Durrett, and Dan Klein. 2014. Less Grammar, More Features. In *Proceedings of the Association for Computational Linguistics*.

James Henderson. 2003. Inducing History Representations for Broad Coverage Statistical Parsing. In *Proceedings of the North American Chapter of the Association for Computational Linguistics*.

Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv preprint*, arXiv:1502.03167.

Ozan İrsoy and Claire Cardie. 2014. Opinion Mining with Deep Recurrent Neural Networks. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. 2014. A Convolutional Neural Network for Modelling Sentences. In *Proceedings of the Association for Computational Linguistics*.

Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Dan Klein and Christopher D. Manning. 2003. Accurate Unlexicalized Parsing. In *Proceedings of the Association for Computational Linguistics*.

Terry Koo, Xavier Carreras, and Michael Collins. 2008. Simple Semi-supervised Dependency Parsing. In *Proceedings of the Association for Computational Linguistics*.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*.

Phong Le and Willem Zuidema. 2014. The inside-outside recursive neural network model for dependency parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Tao Lei, Yu Xin, Yuan Zhang, Regina Barzilay, and Tommi Jaakkola. 2014. Low-Rank Tensors for Scoring Dependency Structures. In *Proceedings of the Association for Computational Linguistics*.

Omer Levy and Yoav Goldberg. 2014. Dependency-Based Word Embeddings. In *Proceedings of the Association for Computational Linguistics*.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2).

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. In *Proceedings of the International Conference on Learning Representations*.

Slav Petrov and Dan Klein. 2007. Improved Inference for Unlexicalized Parsing. In *Proceedings of the North American Chapter of the Association for Computational Linguistics*.

Slav Petrov and Dan Klein. 2008. Sparse Multi-Scale Grammars for Discriminative Latent Variable Parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Slav Petrov. 2010. Products of Random Latent Variable Grammars. In *Proceedings of the North American Chapter of the Association for Computational Linguistics*.

Djamé Seddah, Reut Tsarfaty, Sandra Kübler, Marie Candito, Jinho D. Choi, Richárd Farkas, Jennifer Foster, Iakes Goenaga, Koldo Gojenola Galletebeitia, Yoav Goldberg, Spence Green, Nizar Habash, Marco Kuhlmann, Wolfgang Maier, Joakim Nivre, Adam Przepiórkowski, Ryan Roth, Wolfgang Seeker, Yannick Versley, Veronika Vincze, Marcin Woliński, and Alina Wróblewska. 2013. Overview of the SPMRL 2013 Shared Task: A Cross-Framework Evaluation of Parsing Morphologically Rich Languages. In *Proceedings of the Fourth Workshop on Statistical Parsing of Morphologically-Rich Languages*.

Djamé Seddah, Sandra Kübler, and Reut Tsarfaty. 2014. Introducing the SPMRL 2014 Shared Task on Parsing Morphologically-rich Languages. In *Proceedings of the First Joint Workshop on Statistical Parsing of Morphologically Rich Languages and Syntactic Analysis of Non-Canonical Languages*.

Hiroyuki Shindo, Yusuke Miyao, Akinori Fujino, and Masaaki Nagata. 2012. Bayesian Symbol-refined Tree Substitution Grammars for Syntactic Parsing. In *Proceedings of the Association for Computational Linguistics*.

Richard Socher, John Bauer, Christopher D. Manning, and Andrew Y. Ng. 2013. Parsing With Compositional Vector Grammars. In *Proceedings of the Association for Computational Linguistics*.

Vivek Srikumar and Christopher D Manning. 2014. Learning Distributed Representations for Structured Output Prediction. In *Advances in Neural Information Processing Systems*.

Yuta Tsuboi. 2014. Neural Networks Leverage Corpus-wide Information for Part-of-speech Tagging. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Joseph Turian, Lev Ratinov, and Yoshua Bengio. 2010. Word Representations: A Simple and General Method for Semi-supervised Learning. In *Proceedings of the Association for Computational Linguistics*.

Oriol Vinyals, Lukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey E. Hinton. 2014. Grammar as a Foreign Language. *CoRR*, abs/1412.7449.

Mengqiu Wang and Christopher D. Manning. 2013. Effect of Non-linear Deep Architecture in Sequence Labeling. In *Proceedings of the International Joint Conference on Natural Language Processing*.

Matthew D. Zeiler. 2012. ADADELTA: An Adaptive Learning Rate Method. *CoRR*, abs/1212.5701.

Muhua Zhu, Yue Zhang, Wenliang Chen, Min Zhang, and Jingbo Zhu. 2013. Fast and Accurate Shift-Reduce Constituent Parsing. In *Proceedings of the Association for Computational Linguistics*.