Fast Semantic Extraction Using a Novel Neural Network Architecture

Ronan Collobert
NEC Laboratories America, Inc.
4 Independence Way
Suite 200, Princeton, NJ 08540
`collober@nec-labs.com`

Jason Weston
NEC Laboratories America, Inc.
4 Independence Way
Suite 200, Princeton, NJ 08540
`jasonw@nec-labs.com`

Abstract

We describe a novel neural network architecture for the problem of *semantic role labeling*. Many current solutions are complicated, consist of several stages and hand-built features, and are too slow to be applied as part of real applications that require such semantic labels, partly because of their use of a syntactic parser (Pradhan et al., 2004; Gildea and Jurafsky, 2002). Our method instead learns a direct mapping from source sentence to semantic tags for a given predicate without the aid of a parser *or* a chunker. Our resulting system obtains accuracies comparable to the current state-of-the-art at a fraction of the computational cost.

1 Introduction

Semantic understanding plays an important role in many end-user applications involving text: for information extraction, web-crawling systems, question and answer based systems, as well as machine translation, summarization and search. Such applications typically have to be *computationally cheap* to deal with an enormous quantity of data, e.g. web-based systems process large numbers of documents, whilst interactive human-machine applications require almost instant response. Another issue is the cost of producing labeled training data required for statistical models, which is exacerbated when those models also depend on syntactic features which must themselves be learnt.

To achieve the goal of semantic understanding, the current consensus is to divide and conquer the problem. Researchers tackle several layers of processing tasks ranging from the syntactic, such as part-of-speech labeling and parsing, to the semantic: word-sense disambiguation, semantic role-labeling, named entity extraction, co-reference resolution and entailment. None of these tasks are end goals in themselves but can be seen as layers of feature extraction that can help in a language-based end application, such as the ones described above. Unfortunately, the state-of-the-art solutions of many of these tasks are simply too slow to be used in the applications previously described. For example, state-of-the-art syntactic parsers theoretically have cubic complexity in the sentence length (Younger, 1967)[1] and several semantic extraction algorithms use the parse tree as an initial feature.

[The company]$_{\text{ARG0}}$ [bought]$_{\text{REL}}$ [sugar]$_{\text{ARG1}}$ [on the world market]$_{\text{ARGM-LOC}}$ [to meet export commitments]$_{\text{ARGM-PNC}}$

Figure 1: Example of *Semantic Role Labeling* from the PropBank dataset (Palmer et al., 2005). ARG0 is typically an actor, REL an action, ARG1 an object, and ARGM describe various modifiers such as location (LOC) and purpose (PNC).

In this work, we describe a novel type of neural network architecture that could help to solve some of these issues. We focus our experimental study on the *semantic role labeling* problem (Palmer et al., 2005): being able to give a semantic role to a syn-

[1]Even though some parsers effectively exhibit linear behavior in sentence length (Ratnaparkhi, 1997), fast statistical parsers such as (Henderson, 2004) still take around 1.5 seconds for sentences of length 35 in tests that we made.

tactic constituent of a sentence, i.e. annotating the predicate argument structure in text (see for example Figure 1). Because of its nature, role labeling seems to require the syntactic analysis of a sentence before attributing semantic labels. Using this intuition, state-of-the-art systems first build a *parse tree*, and syntactic constituents are then labeled by feeding hand-built features extracted from the parse tree to a machine learning system, e.g. the ASSERT system (Pradhan et al., 2004). This is rather slow, taking a few seconds per sentence at test time, partly because of the parse tree component, and partly because of the use of Support Vector Machines (Boser et al., 1992), which have linear complexity in testing time with respect to the number of training examples. This makes it hard to apply this method to interesting end user applications.

Here, we propose a radically different approach that avoids the more complex task of building a full parse tree. From a machine learning point of view, a human does not need to be taught about parse trees to talk. It is possible, however, that our brains may implicitly learn features highly correlated with those extracted from a parse tree. We propose to develop an architecture that implements this kind of implicit learning, rather than using explicitly engineered features. In practice, our system also provides semantic tags at a fraction of the computational cost of other methods, taking on average 0.02 seconds to label a sentence from the Penn Treebank, with almost no loss in accuracy.

The rest of the article is as follows. First, we describe the problem of shallow semantic parsing in more detail, as well as existing solutions to this problem. We then detail our algorithmic approach – the neural network architecture we employ – followed by experiments that evaluate our method. Finally, we conclude with a summary and discussion of future work.

2 Shallow Semantic Parsing

FrameNet (Baker et al., 1998) and the Proposition Bank (Palmer et al., 2005), or PropBank for short, are the two main systems currently developed for semantic role-labeling annotation. We focus here on PropBank. PropBank encodes role labels by semantically tagging the syntactic structures of hand annotated parses of sentences. The current version of the dataset gives semantic tags for the same sentences as in the Penn Treebank (Marcus et al., 1993), which are excerpts from the Wall Street Journal. The central idea is that each verb in a sentence is labeled with its propositional arguments, where the abstract numbered arguments are intended to fill typical roles. For example, ARG0 is typically the actor, and ARG1 is typically the thing acted upon. The precise usage of the numbering system is labeled for each particular verb as so-called frames. Additionally, semantic roles can also be labeled with one of 13 ARGM adjunct labels, such as ARGM-LOC or ARGM-TMP for additional locational or temporal information relative to some verb.

Shallow semantic parsing has immediate applications in tasks such as meta-data extraction (e.g. from web documents) and question and answer based systems (e.g. call center systems), amongst others.

3 Previous Work

Several authors have already attempted to build machine learning approaches for the semantic role-labeling problem. In (Gildea and Jurafsky, 2002) the authors presented a statistical approach to learning (for FrameNet), with some success. They proposed to take advantage of the syntactic tree structure that can be predicted by a parser, such as Charniak's parser (Charniak, 2000). Their aim is, given a node in the parse tree, to assign a semantic role label to the words that are the children of that node. They extract several key types of features from the parse tree to be used in a statistical model for prediction. These same features also proved crucial to subsequent approaches, e.g. (Pradhan et al., 2004). These features include:

- The parts of speech and syntactic labels of words and nodes in the tree.
- The node's position (left or right) in relation to the verb.
- The syntactic path to the verb in the parse tree.
- Whether a node in the parse tree is part of a noun or verb phrase (by looking at the parent nodes of that node).

- The voice of the sentence: active or passive (part of the PropBank gold annotation);

as well as several other features (predicate, head word, verb sub-categorization, ...).

The authors of (Pradhan et al., 2004) used a similar structure, but added more features, notably head word part-of-speech, the predicted named entity class of the argument, word sense disambiguation of the verb and verb clustering, and others (they add 25 variants of 12 new feature types overall.) Their system also uses a parser, as before, and then a polynomial Support Vector Machine (SVM) (Boser et al., 1992) is used in two further stages: to classify each node in the tree as being a semantic argument or not for a given verb; and then to classify each semantic argument into one of the classes (ARG1, ARG2, etc.). The first SVM solves a two-class problem, the second solves a multi-class problem using a one-vs-the-rest approach. The final system, called ASSERT, gives state-of-the-art performance and is also freely available at: `http://oak.colorado.edu/assert/`. We compare to this system in our experimental results in Section 5. Several other competing methods exist, e.g. the ones that participated in the CONLL 2004 and 2005 challenges (`http://www.lsi.upc.edu/~srlconll/st05/st05.html`). In this paper we focus on a comparison with ASSERT because software to re-run it is available online. This also gives us a timing result for comparison purposes.

The three-step procedure used in ASSERT (calculating a parse tree and then applying SVMs twice) leads to good classification performance, but has several drawbacks. First in speed: predicting a parse tree is extremely demanding in computing resources. Secondly, choosing the features necessary for SVM classification requires extensive research. Finally, the SVM classification algorithm used in existing approaches is rather slow: SVM training is at least quadratic in time with respect to the number of training examples. The number of support vectors involved in the SVM decision function also increases linearly with the number of training examples. This makes SVMs slow on large-scale problems, both during training and testing phases.

To alleviate the burden of parse tree computation, several attempts have been made to remove the full parse tree information from the semantic role labeling system, in fact the shared task of CONLL 2004 was devoted to this goal, but the results were not completely satisfactory. Previously, in (Gildea and Palmer, 2001), the authors tried to show that the parse tree is necessary for good generalization by showing that segments derived from a shallow syntactic parser *or* chunker do not perform as well for this goal. A further analysis of using chunkers, with improved results was also given in (Punyakanok et al., 2005), but still concluded the full parse tree is most useful.

4 Neural Network Architecture

Ideally, we want an end-to-end *fast* learning system to output semantic roles for syntactic constituents without using a time consuming parse tree.

Also, as explained before, we are interesting in exploring whether machine learning approaches can learn structure implicitly. Hence, even if there is a deep relationship between syntax and semantics, we prefer to avoid hand-engineered features that exploit this, and see if we can develop a model that can learn these features instead. We are thus not interested in chunker-based techniques, even though they are faster than parser-based techniques.

We propose here a *neural network based architecture* which achieves these two goals.

4.1 Basic Architecture

The type of neural network that we employ is a Multi Layer Perceptron (MLP). MLPs have been used for many years in the machine learning field and slowly abandoned for several reasons: partly because of the difficulty of solving the non-convex optimization problems associated with learning (LeCun et al., 1998), and partly because of the difficulty of their theoretical analysis compared to alternative convex approaches.

An MLP works by successively projecting the data to be classified into different spaces. These projections are done in what is called *hidden layers*. Given an input vector $\boldsymbol{z}$, a hidden layer applies a linear transformation (matrix M) followed by a squashing function h:

$$\boldsymbol{z} \mapsto M\boldsymbol{z} \mapsto h(M\boldsymbol{z})\,. \qquad (1)$$

A typical squashing function is the hyperbolic tangent $h(\cdot) = \tanh(\cdot)$. The last layer (the output layer) linearly separates the classes. The composition of the projections in the hidden layers could be viewed as the work done by the kernel in SVMs. However there is a very important difference: the kernel in SVM is fixed and arbitrarily chosen, while the hidden layers in an MLP are *trained* and *adapted* to the classification task. This allows us to create much more flexible classification architectures.

Our method for semantic role labeling classifies each word of a sentence separately. We do not use any semantic constituent information: if the model is powerful enough, words in the same semantic constituent should have the same class label. This means we also do not separate the problem into an identification and classification phase, but rather solve in a single step.

4.1.1 Notation

We represent words as indices. We consider a finite dictionary of words $\mathcal{D} \subset \mathbb{N}$. Let us represent a sentence of n_w words to be analyzed as a function $s(\cdot)$. The i^{th} word in the sentence is given by the index $s(i)$:

$$1 \leq i \leq n_w \quad s(i) \in \mathcal{D} \,.$$

We are interested in predicting the semantic role label of the word at position pos_w, given a verb at position pos_v ($1 \leq pos_w, pos_v \leq n_w$). A mathematical description of our network architecture schematically shown in Figure 2 follows.

4.1.2 Transforming words into feature vectors

Our first concern in semantic role labeling is that we have to deal with words, and that a simple index $i \in \mathcal{D}$ does not carry any information specific to a word: for each word we need a set of features relevant for the task. As described earlier, previous methods construct a parse tree, and then compute hand-built features which are then fed to a classification algorithm. In order to bypass the use of a parse tree, we convert each word $i \in \mathcal{D}$ into a particular vector $\boldsymbol{w}_i \in \mathbb{R}^d$ *which is going to be learnt* for the task we are interested in. This approach has already been used with great success in the domain of language models (Bengio and Ducharme, 2001; Schwenk and Gauvain, 2002).

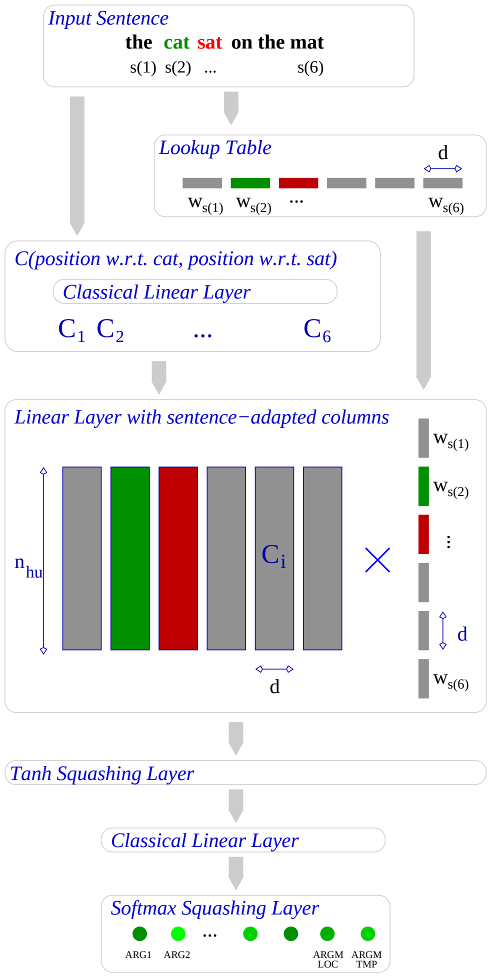

Figure 2: MLP architecture for shallow semantic parsing. The input sequence is at the top. The output class probabilities for the word of interest ("cat") given the verb of interest ("sat") are given at the bottom.

The first layer of our MLP is thus a lookup table which replaces the word indices into a concatenation of vectors:

$$\begin{aligned} &\{s(1), \ldots, s(n_w)\} \\ &\qquad \mapsto (\boldsymbol{w}_{s(1)} \ldots \boldsymbol{w}_{s(n_w)}) \in \mathbb{R}^{n_w\, d}\,. \end{aligned} \tag{2}$$

The weights $\{\boldsymbol{w}_i \,|\, i \in \mathcal{D}\}$ for this layer are considered during the backpropagation phase of the MLP, and thus adapted automatically for the task we are interested in.

4.1.3 Integrating the verb position

Feeding word vectors alone to a linear classification layer as in (Bengio and Ducharme, 2001) leads to very poor accuracy because the semantic classification of a given word also depends on the verb in question. We need to provide the MLP with information about the verb position within the sentence. For that purpose we use a kind of linear layer which is *adapted* to the sentence considered. It takes the form:

$$(\boldsymbol{w}_{s(1)} \, \dots \, \boldsymbol{w}_{s(n_w)}) \mapsto M \begin{pmatrix} \boldsymbol{w}^{\mathrm{T}}_{s(1)} \\ \vdots \\ \boldsymbol{w}^{\mathrm{T}}_{s(n_w)} \end{pmatrix} ,$$

where $M \in \mathbb{R}^{n_{hu} \times n_w \, d}$, and n_{hu} is the number of hidden units. The specific nature of this layer is that the matrix M has a special block-column form which depends on the sentence:

$$M = (C_1 | \dots | C_{n_w}) \ ,$$

where each column $C_i \in \mathbb{R}^{n_{hu} \times d}$ depends on the position of the i^{th} word in $s(\cdot)$, with respect to the position pos_w of the word of interest, and with respect to the position pos_v of the verb of interest:

$$C_i = \mathcal{C}(i - pos_w, \, i - pos_v) \, ,$$

where $\mathcal{C}(\cdot, \cdot)$ is a function to be chosen.

In our experiments $\mathcal{C}(\cdot, \cdot)$ was a linear layer with discretized inputs $(i - pos_w, \, i - pos_v)$ which were transformed into two binary vectors of size wsz, where a bit is set to 1 if it corresponds to the position to encode, and 0 otherwise. These two binary vectors are then concatenated and fed to the linear layer. We chose the "window size" $wsz = 11$. If a position lies outside the window, then we still set the leftmost or rightmost bit to 1. The parameters involved in this function are also considered during the backpropagation. With such an architecture we allow our MLP to automatically adapt the importance of a word in the sentence given its distance to the word we want to classify, and to the verb we are interested in.

This idea is the major novelty in this work, and is crucial for the success of the entire architecture, as we will see in the experiments.

4.1.4 Learning class probabilities

The last layer in our MLP is a classical linear layer as described in (1), with a *softmax* squashing function (Bridle, 1990). Considering (1) and given $\tilde{\boldsymbol{z}} = M\boldsymbol{z}$, we have

$$h_i(\tilde{\boldsymbol{z}}) = \frac{\exp \tilde{z}_i}{\sum_j \exp \tilde{z}_j} \, .$$

This allows us to interpret outputs as probabilities for each semantic role label. The training of the whole system is achieved using a normal stochastic gradient descent.

4.2 Word representation

As we have seen, in our model we are *learning* one d dimensional vector to represent each word. If the dataset were large enough, this would be an elegant solution. In practice many words occur infrequently within PropBank, so (independent of the size of d) we can still only learn a very poor representation for words that only appear a few times. Hence, to control the capacity of our model we take the original word and replace it with its part-of-speech if it is a verb, noun, adjective, adverb or number as determined by a part-of-speech classifier, and keep the words for all other parts of speech. This classifier is itself a neural network. This way we keep linking words which are important for this task. We do not do this replacement for the predicate itself.

5 Experiments

We used Sections 02-21 of the PropBank dataset version 1 for training and validation and Section 23 for testing as standard in all our experiments. We first describe the part-of-speech tagger we employ, and then describe our semantic role labeling experiments. Software for our method, SENNA (Semantic Extraction using a Neural Network Architecture), more details on its implementation, an online applet and test set predictions of our system in comparison to ASSERT can be found at `http://ml.nec-labs.com/software/senna`.

Part-Of-Speech Tagger The part-of-speech classifier we employ is a neural network architecture of the same type as in Section 4, where the function $C_i = C(i - pos_w)$ depends now only on the word position, and not on a verb. More precisely:

$$C_i = \begin{cases} 0 & \text{if } 2\,|i - pos_w| > wsz - 1 \\ W_{i - pos_w} & \text{otherwise} \, , \end{cases}$$

where $W_k \in \mathbb{R}^{n_{hu} \times d}$ and wsz is a window size. We chose $wsz = 5$ in our experiments. The d-dimensional vectors learnt take into account the capitalization of a word, and the prefix and suffix calculated using Porter-Stemmer. See `http://ml.nec-labs.com/software/senna` for more details. We trained on the training set of PropBank supplemented with the Brown corpus, resulting in a test accuracy on the test set of PropBank of 96.85% which compares to 96.66% using the Brill tagger (Brill, 1992).

Semantic Role Labeling In our experiments we considered a 23-class problem of NULL (no label), the core arguments ARG0-5, REL, ARGA, and ARGM- along with the 13 secondary modifier labels such as ARGM-LOC and ARGM-TMP. We simplified R-ARGn and C-ARGn to be written as ARGn, and post-processed ASSERT to do this as well.

We compared our system to the freely available ASSERT system (Pradhan et al., 2004). Both systems are fed only the input sentence during testing, with traces removed, so they cannot make use of many PropBank features such as frameset identitifier, person, tense, aspect, voice, and form of the verb. As our algorithm outputs a semantic tag for each word of a sentence, we directly compare this per-word accuracy with ASSERT. Because ASSERT uses a parser, and because PropBank was built by labeling the nodes of a hand-annotated parse tree, per-node accuracy is usually reported in papers such as (Pradhan et al., 2004). Unfortunately our approach is based on a completely different premise: we tag words, not syntactic constituents coming from the parser. We discuss this further in Section 5.2.

The per-word accuracy comparison results can be seen in Table 5. Before labeling the semantic roles of each predicate, one must first identify the predicates themselves. If a predicate is not identified, NULL tags are assigned to each word for that predicate. The first line of results in the table takes into account this identification process. For the neural network, we used our part-of-speech tagger to perform this as a verb-detection task.

We noticed ASSERT failed to identify relatively many predicates. In particular, it seems predicates such as "is" are sometimes labeled as AUX by the part-of-speech tagger, and subsequently ignored. We informed the authors of this, but we did not receive a response. To deal with this, we considered the additional accuracy (second row in the table) measured over only those sentences where the predicate was identified by ASSERT.

Timing results The per-sentence compute time is also given in Table 5, averaged over all sentences in the test set. Our method is around 250 times faster than ASSERT. It is not really feasible to run ASSERT for most applications.

Measurement	NN	ASSERT
Per-word accuracy (all verbs)	83.64%	83.46%
Per-word accuracy (ASSERT verbs)	84.09%	86.06%
Per-sentence compute time (secs)	0.02 secs	5.08 secs

Table 1: Experimental comparison with ASSERT

5.1 Analysis of our MLP

While we gave an intuitive justification of the architecture choices of our model in Section 4, we now give a systematic empirical study of those choices. First of all, providing the position of the word *and* the predicate in function $C(\cdot, \cdot)$ is essential: the best model we obtained with a window around the word only gave 51.3%, assuming correct identification of all predicates. Our best model achieves 83.95% in this setting.

If we do not cluster the words according to their part-of-speech, we also lose some performance, obtaining 78.6% at best. On the other hand, clustering *all* words (such as CC, DT, IN part-of-speech tags) also gives weaker results (81.1% accuracy at best). We believe that including all words would give very good performance if the dataset was large enough, but training only on PropBank leads to overfitting, many words being infrequent. Clustering is a way to fight against overfitting, by grouping infrequent words: for example, words with the label NNP, JJ, RB (which we cluster) appear on average 23, 22 and 72 times respectively in the training set, while CC, DT, IN (which we do not cluster) appear 2420, 5659 and 1712 times respectively.

Even though some verbs are infrequent, one cannot cluster all verbs into a single group, as each verb dictates the types of semantic roles in the sentence, depending on its frame. Clustering all words into their part-of-speech, including the predicate, gives a poor 73.8% compared with 81.1%, where everything is clustered apart from the predicate.

Figure 3 gives some anecdotal examples of test set predictions of our final model compared to ASSERT.

TRUTH: He camped out at a high-tech nerve center on the floor of [the Big Board, where]$_{\text{ARGM-LOC}}$ [he]$_{\text{ARG0}}$ [could]$_{\text{ARGM-MOD}}$ [watch]$_{\text{REL}}$ [updates on prices and pending stock orders]$_{\text{ARG1}}$.
ASSERT (68.7%): He camped out at a high-tech nerve center on the floor of the Big Board, [where]$_{\text{ARGM-LOC}}$ [he]$_{\text{ARG0}}$ [could]$_{\text{ARGM-MOD}}$ [watch]$_{\text{REL}}$ [updates]$_{\text{ARG1}}$ on prices and pending stock orders.
NN (100%): He camped out at a high-tech nerve center on the floor of [the Big Board, where]$_{\text{ARGM-LOC}}$ [he]$_{\text{ARG0}}$ [could]$_{\text{ARGM-MOD}}$ [watch]$_{\text{REL}}$ [updates on prices and pending stock orders]$_{\text{ARG1}}$.

TRUTH: [United Auto Workers Local 1069, which]$_{\text{ARG0}}$ [represents]$_{\text{REL}}$ [3,000 workers at Boeing's helicopter unit in Delaware County, Pa.]$_{\text{ARG1}}$, said it agreed to extend its contract on a day-by-day basis, with a 10-day notification to cancel, while it continues bargaining.
ASSERT (100%): [United Auto Workers Local 1069, which]$_{\text{ARG0}}$ [represents]$_{\text{REL}}$ [3,000 workers at Boeing's helicopter unit in Delaware County, Pa.]$_{\text{ARG1}}$, said it agreed to extend its contract on a day-by-day basis, with a 10-day notification to cancel, while it continues bargaining.
NN (89.1%): [United Auto Workers Local 1069, which]$_{\text{ARG0}}$ [represents]$_{\text{REL}}$ [3,000 workers at Boeing's helicopter unit]$_{\text{ARG1}}$ [in Delaware County]$_{\text{ARGM-LOC}}$, Pa. , said it agreed to extend its contract on a day-by-day basis, with a 10-day notification to cancel, while it continues bargaining.

Figure 3: Two examples from the PropBank test set, showing Neural Net and ASSERT and gold standard labelings, with per-word accuracy in brackets. Note that even though our labeling does not match the hand-annotated one in the second sentence it still seems to make some sense as "in Delaware County" is labeled as a location modifier. The complete set of predictions on the test set can be found at `http://ml.nec-labs.com/software/senna`.

5.2 Argument Classification Accuracy

So far we have not used the same accuracy measures as in previous work (Gildea and Jurafsky, 2002; Pradhan et al., 2004). Currently our architecture is designed to label on a per-word basis, while existing systems perform a segmentation process, and then label segments. While we do not optimize our model for the same criteria, it is still possible to measure the accuracy using the same metrics. We measured the argument classification accuracy of our network, assuming the correct segmentation is given to our system, as in (Pradhan et al., 2004), by post-processing our per-word tags to form a majority vote over each segment. This gives 83.18% accuracy for our network when we suppose the predicate must also be identified, and 80.53% for the ASSERT software. Measuring only on predicates identified by ASSERT we instead obtain 84.32% accuracy for our network, and 87.02% for ASSERT.

6 Discussion

We have introduced a neural network architecture that can provide computationally efficient semantic role tagging. It is also a general architecture that could be applied to other problems as well. Because our network currently outputs labels on a per-word basis it is difficult to assess existing accuracy measures. However, it should be possible to combine our approach with a shallow parser to enhance performance, and make comparisons more direct.

We consider this work as a starting point for different research directions, including the following areas:

- *Incorporating hand-built features* Currently, the only prior knowledge our system encodes comes from part-of-speech tags, in stark contrast to other methods. Of course, performance would improve with more hand-built features. For example, simply adding whether each word is part of a noun or verb phrase using the hand-annotated parse tree (the so-called "GOV" feature from (Gildea and Jurafsky, 2002)) improves the performance of our system from 83.95% to 85.8%. One must trade the generality of the model with its specificity, and also take into account how long the features take to compute.

- *Incorporating segment information* Our system has no prior knowledge about segmentation in text. This could be encoded in many ways: most obviously by using a chunker, but also by designing a different network architecture, e.g. by encoding contiguity constraints. To show the latter is useful, using hand-annotated segments to force contiguity by majority vote leads to an improvement from 83.95% to 85.6%.

- *Incorporating known invariances via virtual training data.* In image recognition problems it is common to create artificial training data by taking into account invariances in the images, e.g. via rotation and scale. Such data improves generalization substantially. It may be possible to achieve similar results for text, by "warping" training data to create new sentences, or by constructing sentences from scratch using a hand-built grammar.
- *Unlabeled data.* Our representation of words is as d dimensional vectors. We could try to improve this representation by learning a language model from unlabeled data (Bengio and Ducharme, 2001). As many words in PropBank only appear a few times, the representation might improve, even though the learning is unsupervised. This may also make the system generalize better to types of data other than the Wall Street Journal.
- *Transductive Inference.* Finally, one can also use unlabeled data as part of the supervised training process, which is called transduction or semi-supervised learning.

In particular, we find the possibility of using unlabeled data, invariances and the use of transduction exciting. These possibilities naturally fit into our framework, whereas scalability issues will limit their application in competing methods.

References

C.F. Baker, C.J. Fillmore, and J.B. Lowe. 1998. The Berkeley FrameNet project. *Proceedings of COLING-ACL*, 98.

Y. Bengio and R. Ducharme. 2001. A neural probabilistic language model. In *Advances in Neural Information Processing Systems, NIPS 13*.

B.E. Boser, I.M. Guyon, and V.N. Vapnik. 1992. A training algorithm for optimal margin classifiers. *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152.

J.S. Bridle. 1990. Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In F. Fogelman Soulié and J. Hérault, editors, *Neurocomputing: Algorithms, Architectures and Applications*, pages 227–236. NATO ASI Series.

E. Brill. 1992. A simple rule-based part of speech tagger. *Proceedings of the Third Conference on Applied Natural Language Processing*, pages 152–155.

E. Charniak. 2000. A maximum-entropy-inspired parser. *Proceedings of the first conference on North American chapter of the Association for Computational Linguistics*, pages 132–139.

D. Gildea and D. Jurafsky. 2002. Automatic labeling of semantic roles. *Computational Linguistics*, 28(3):245–288.

D. Gildea and M. Palmer. 2001. The necessity of parsing for predicate argument recognition. *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 239–246.

J. Henderson. 2004. Discriminative training of a neural network statistical parser. In *Proceedings of the 42nd Meeting of Association for Computational Linguistics.*

Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller. 1998. Efficient backprop. In G.B. Orr and K.-R. Müller, editors, *Neural Networks: Tricks of the Trade*, pages 9–50. Springer.

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

M. Palmer, D. Gildea, and P. Kingsbury. 2005. The proposition bank: An annotated corpus of semantic roles. *Comput. Linguist.*, 31(1):71–106.

S. Pradhan, W. Ward, K. Hacioglu, J. Martin, and D. Jurafsky. 2004. Shallow semantic parsing using support vector machines. *Proceedings of HLT/NAACL-2004*.

V. Punyakanok, D. Roth, and W. Yih. 2005. The necessity of syntactic parsing for semantic role labeling. *Proceedings of IJCAI'05*, pages 1117–1123.

A. Ratnaparkhi. 1997. A linear observed time statistical parser based on maximum entropy models. *Proceedings of EMNLP*.

H. Schwenk and J.L. Gauvain. 2002. Connectionist language modeling for large vocabulary continuousspeech recognition. *Proceedings of ICASSP'02*.

D. H. Younger. 1967. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10.