Journal of Machine Learning Research 12 (2011) 2493-2537 Submitted 1/10; Revised 11/10; Published 8/11

Natural Language Processing (Almost) from Scratch

Ronan Collobert[*] RONAN@COLLOBERT.COM
Jason Weston[†] JWESTON@GOOGLE.COM
Léon Bottou[‡] LEON@BOTTOU.ORG
Michael Karlen MICHAEL.KARLEN@GMAIL.COM
Koray Kavukcuoglu[§] KORAY@CS.NYU.EDU
Pavel Kuksa[¶] PKUKSA@CS.RUTGERS.EDU
NEC Laboratories America
4 Independence Way
Princeton, NJ 08540

Editor: Michael Collins

Abstract

We propose a unified neural network architecture and learning algorithm that can be applied to various natural language processing tasks including part-of-speech tagging, chunking, named entity recognition, and semantic role labeling. This versatility is achieved by trying to avoid task-specific engineering and therefore disregarding a lot of prior knowledge. Instead of exploiting man-made input features carefully optimized for each task, our system learns internal representations on the basis of vast amounts of mostly unlabeled training data. This work is then used as a basis for building a freely available tagging system with good performance and minimal computational requirements.

Keywords: natural language processing, neural networks

1. Introduction

Will a computer program ever be able to convert a piece of English text into a programmer friendly data structure that describes the meaning of the natural language text? Unfortunately, no consensus has emerged about the form or the existence of such a data structure. Until such fundamental Articial Intelligence problems are resolved, computer scientists must settle for the reduced objective of extracting simpler representations that describe limited aspects of the textual information.

These simpler representations are often motivated by specific applications (for instance, bag-of-words variants for information retrieval), or by our belief that they capture something more general about natural language. They can describe syntactic information (e.g., part-of-speech tagging, chunking, and parsing) or semantic information (e.g., word-sense disambiguation, semantic role labeling, named entity extraction, and anaphora resolution). Text corpora have been manually annotated with such data structures in order to compare the performance of various systems. The availability of standard benchmarks has stimulated research in Natural Language Processing (NLP)

∗. Ronan Collobert is now with the Idiap Research Institute, Switzerland.
†. Jason Weston is now with Google, New York, NY.
‡. Léon Bottou is now with Microsoft, Redmond, WA.
§. Koray Kavukcuoglu is also with New York University, New York, NY.
¶. Pavel Kuksa is also with Rutgers University, New Brunswick, NJ.

and effective systems have been designed for all these tasks. Such systems are often viewed as software components for constructing real-world NLP solutions.

The overwhelming majority of these state-of-the-art systems address their single benchmark task by applying linear statistical models to ad-hoc features. In other words, the researchers themselves discover intermediate representations by engineering task-specific features. These features are often derived from the output of preexisting systems, leading to complex runtime dependencies. This approach is effective because researchers leverage a large body of linguistic knowledge. On the other hand, there is a great temptation to optimize the performance of a system for a specific benchmark. Although such performance improvements can be very useful in practice, they teach us little about the means to progress toward the broader goals of natural language understanding and the elusive goals of Artificial Intelligence.

In this contribution, we try to excel on *multiple benchmarks* while *avoiding task-specific engineering*. Instead we use a *single learning system* able to discover adequate internal representations. In fact we view the benchmarks as indirect measurements of the relevance of the internal representations discovered by the learning procedure, and we posit that these intermediate representations are more general than any of the benchmarks. Our desire to avoid task-specific engineered features prevented us from using a large body of linguistic knowledge. Instead we reach good performance levels in most of the tasks by transferring intermediate representations discovered on large unlabeled data sets. We call this approach "almost from scratch" to emphasize the reduced (but still important) reliance on a priori NLP knowledge.

The paper is organized as follows. Section 2 describes the benchmark tasks of interest. Section 3 describes the unified model and reports benchmark results obtained with supervised training. Section 4 leverages large unlabeled data sets ($\sim$ 852 million words) to train the model on a language modeling task. Performance improvements are then demonstrated by transferring the unsupervised internal representations into the supervised benchmark models. Section 5 investigates multitask supervised training. Section 6 then evaluates how much further improvement can be achieved by incorporating standard NLP task-specific engineering into our systems. Drifting away from our initial goals gives us the opportunity to construct an all-purpose tagger that is simultaneously accurate, practical, and fast. We then conclude with a short discussion section.

2. The Benchmark Tasks

In this section, we briefly introduce four standard NLP tasks on which we will benchmark our architectures within this paper: Part-Of-Speech tagging (POS), chunking (CHUNK), Named Entity Recognition (NER) and Semantic Role Labeling (SRL). For each of them, we consider a standard experimental setup and give an overview of state-of-the-art systems on this setup. The experimental setups are summarized in Table 1, while state-of-the-art systems are reported in Table 2.

2.1 Part-Of-Speech Tagging

POS aims at labeling each word with a unique tag that indicates its *syntactic role*, for example, plural noun, adverb, ... A standard benchmark setup is described in detail by Toutanova et al. (2003). Sections 0–18 of Wall Street Journal (WSJ) data are used for training, while sections 19–21 are for validation and sections 22–24 for testing.

The best POS classifiers are based on classifiers trained on windows of text, which are then fed to a bidirectional decoding algorithm during inference. Features include preceding and following

Task	Benchmark	Data set	Training set (#tokens)	Test set (#tokens)	(#tags)
POS	Toutanova et al. (2003)	WSJ	sections 0–18 (912,344)	sections 22–24 (129,654)	(45)
Chunking	CoNLL 2000	WSJ	sections 15–18 (211,727)	section 20 (47,377)	(42) (IOBES)
NER	CoNLL 2003	Reuters	"eng.train" (203,621)	"eng.testb" (46,435)	(17) (IOBES)
SRL	CoNLL 2005	WSJ	sections 2–21 (950,028)	section 23 + 3 Brown sections (63,843)	(186) (IOBES)

Table 1: Experimental setup: for each task, we report the standard benchmark we used, the data set it relates to, as well as training and test information.

System	Accuracy
Shen et al. (2007)	97.33%
Toutanova et al. (2003)	97.24%
Giménez and Màrquez (2004)	97.16%

(a) POS

System	F1
Shen and Sarkar (2005)	95.23%
Sha and Pereira (2003)	94.29%
Kudo and Matsumoto (2001)	93.91%

(b) CHUNK

System	F1
Ando and Zhang (2005)	89.31%
Florian et al. (2003)	88.76%
Kudo and Matsumoto (2001)	88.31%

(c) NER

System	F1
Koomen et al. (2005)	77.92%
Pradhan et al. (2005)	77.30%
Haghighi et al. (2005)	77.04%

(d) SRL

Table 2: State-of-the-art systems on four NLP tasks. Performance is reported in per-word accuracy for POS, and F1 score for CHUNK, NER and SRL. Systems in bold will be referred as *benchmark systems* in the rest of the paper (see Section 2.6).

tag context as well as multiple words (bigrams, trigrams...) context, and handcrafted features to deal with unknown words. Toutanova et al. (2003), who use maximum entropy classifiers and inference in a bidirectional dependency network (Heckerman et al., 2001), reach 97.24% per-word accuracy. Giménez and Màrquez (2004) proposed a SVM approach also trained on text windows, with bidirectional inference achieved with two Viterbi decoders (left-to-right and right-to-left). They obtained 97.16% per-word accuracy. More recently, Shen et al. (2007) pushed the state-of-the-art up to 97.33%, with a new learning algorithm they call *guided learning*, also for bidirectional sequence classification.

2.2 Chunking

Also called shallow parsing, chunking aims at labeling segments of a sentence with syntactic constituents such as noun or verb phrases (NP or VP). Each word is assigned only one unique tag, often encoded as a begin-chunk (e.g., B-NP) or inside-chunk tag (e.g., I-NP). Chunking is often evaluated using the CoNLL 2000 shared task.[1] Sections 15–18 of WSJ data are used for training and section 20 for testing. Validation is achieved by splitting the training set.

Kudoh and Matsumoto (2000) won the CoNLL 2000 challenge on chunking with a F1-score of 93.48%. Their system was based on Support Vector Machines (SVMs). Each SVM was trained in a pairwise classification manner, and fed with a window around the word of interest containing POS and words as features, as well as surrounding tags. They perform dynamic programming at test time. Later, they improved their results up to 93.91% (Kudo and Matsumoto, 2001) using an ensemble of classifiers trained with different tagging conventions (see Section 3.3.3).

Since then, a certain number of systems based on second-order random fields were reported (Sha and Pereira, 2003; McDonald et al., 2005; Sun et al., 2008), all reporting around 94.3% F1 score. These systems use features composed of words, POS tags, and tags.

More recently, Shen and Sarkar (2005) obtained 95.23% using a voting classifier scheme, where each classifier is trained on different tag representations[2] (IOB, IOE, ...). They use POS features coming from an external tagger, as well carefully hand-crafted *specialization* features which again change the data representation by concatenating some (carefully chosen) chunk tags or some words with their POS representation. They then build trigrams over these features, which are finally passed through a Viterbi decoder a test time.

2.3 Named Entity Recognition

NER labels atomic elements in the sentence into categories such as "PERSON" or "LOCATION". As in the chunking task, each word is assigned a tag prefixed by an indicator of the beginning or the inside of an entity. The CoNLL 2003 setup[3] is a NER benchmark data set based on Reuters data. The contest provides training, validation and testing sets.

Florian et al. (2003) presented the best system at the NER CoNLL 2003 challenge, with 88.76% F1 score. They used a combination of various machine-learning classifiers. Features they picked included words, POS tags, CHUNK tags, prefixes and suffixes, a large gazetteer (not provided by the challenge), as well as the output of two other NER classifiers trained on richer data sets. Chieu (2003), the second best performer of CoNLL 2003 (88.31% F1), also used an external gazetteer (their performance goes down to 86.84% with no gazetteer) and several hand-chosen features.

Later, Ando and Zhang (2005) reached 89.31% F1 with a semi-supervised approach. They trained jointly a linear model on NER with a linear model on two auxiliary unsupervised tasks. They also performed Viterbi decoding at test time. The unlabeled corpus was 27M words taken from Reuters. Features included words, POS tags, suffixes and prefixes or CHUNK tags, but overall were less specialized than CoNLL 2003 challengers.

1. See `http://www.cnts.ua.ac.be/conll2000/chunking`.
2. See Table 3 for tagging scheme details.
3. See `http://www.cnts.ua.ac.be/conll2003/ner`.

2.4 Semantic Role Labeling

SRL aims at giving a semantic role to a syntactic constituent of a sentence. In the PropBank (Palmer et al., 2005) formalism one assigns roles ARG0-5 to words that are arguments of a verb (or more technically, a *predicate*) in the sentence, for example, the following sentence might be tagged "[John]$_{ARG0}$ [ate]$_{REL}$ [the apple]$_{ARG1}$ ", where "ate" is the predicate. The precise arguments depend on a verb's *frame* and if there are multiple verbs in a sentence some words might have multiple tags. In addition to the ARG0-5 tags, there there are several modifier tags such as ARGM-LOC (locational) and ARGM-TMP (temporal) that operate in a similar way for all verbs. We picked CoNLL 2005[4] as our SRL benchmark. It takes sections 2–21 of WSJ data as training set, and section 24 as validation set. A test set composed of section 23 of WSJ concatenated with 3 sections from the Brown corpus is also provided by the challenge.

State-of-the-art SRL systems consist of several stages: producing a parse tree, identifying which parse tree nodes represent the arguments of a given verb, and finally classifying these nodes to compute the corresponding SRL tags. This entails extracting numerous base features from the parse tree and feeding them into statistical models. Feature categories commonly used by these system include (Gildea and Jurafsky, 2002; Pradhan et al., 2004):

- 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;
- the voice of the sentence: active or passive;
- the node's head word; and
- the verb sub-categorization.

Pradhan et al. (2004) take these base features and define additional features, notably the part-of-speech tag of the head word, the predicted named entity class of the argument, features providing word sense disambiguation for the verb (they add 25 variants of 12 new feature types overall). This system is close to the state-of-the-art in performance. Pradhan et al. (2005) obtain 77.30% F1 with a system based on SVM classifiers and simultaneously using the two parse trees provided for the SRL task. In the same spirit, Haghighi et al. (2005) use log-linear models on each tree node, re-ranked globally with a dynamic algorithm. Their system reaches 77.04% using the five top Charniak parse trees.

Koomen et al. (2005) hold the state-of-the-art with Winnow-like (Littlestone, 1988) classifiers, followed by a decoding stage based on an integer program that enforces specific constraints on SRL tags. They reach 77.92% F1 on CoNLL 2005, thanks to the five top parse trees produced by the Charniak (2000) parser (only the first one was provided by the contest) as well as the Collins (1999) parse tree.

4. See `http://www.lsi.upc.edu/~srlconll`.

2.5 Evaluation

In our experiments, we strictly followed the standard evaluation procedure of each CoNLL challenges for NER, CHUNK and SRL. In particular, we chose the hyper-parameters of our model according to a simple validation procedure (see Remark 8 later in Section 3.5), performed over the validation set available for each task (see Section 2). All these three tasks are evaluated by computing the F1 scores over *chunks* produced by our models. The POS task is evaluated by computing the *per-word* accuracy, as it is the case for the standard benchmark we refer to (Toutanova et al., 2003). We used the `conlleval` script[5] for evaluating POS,[6] NER and CHUNK. For SRL, we used the `srl-eval.pl` script included in the `srlconll` package.[7]

2.6 Discussion

When participating in an (open) challenge, it is legitimate to increase generalization by all means. It is thus not surprising to see many top CoNLL systems using *external labeled data*, like additional NER classifiers for the NER architecture of Florian et al. (2003) or additional parse trees for SRL systems (Koomen et al., 2005). Combining multiple systems or tweaking carefully features is also a common approach, like in the chunking top system (Shen and Sarkar, 2005).

However, when *comparing* systems, we do not learn anything of the quality of each system if they were trained with *different* labeled data. For that reason, we will refer to *benchmark systems*, that is, top existing systems which avoid usage of external data and have been well-established in the NLP field: Toutanova et al. (2003) for POS and Sha and Pereira (2003) for chunking. For NER we consider Ando and Zhang (2005) as they were using additional *unlabeled* data only. We picked Koomen et al. (2005) for SRL, keeping in mind they use 4 additional parse trees not provided by the challenge. These benchmark systems will serve as baseline references in our experiments. We marked them in bold in Table 2.

We note that for the four tasks we are considering in this work, it can be seen that for the more complex tasks (with corresponding lower accuracies), the best systems proposed have more engineered features relative to the best systems on the simpler tasks. That is, the POS task is one of the simplest of our four tasks, and only has relatively few engineered features, whereas SRL is the most complex, and many kinds of features have been designed for it. This clearly has implications for as yet unsolved NLP tasks requiring more sophisticated semantic understanding than the ones considered here.

3. The Networks

All the NLP tasks above can be seen as tasks assigning labels to words. The traditional NLP approach is: extract from the sentence a rich set of hand-designed features which are then fed to a standard classification algorithm, for example, a Support Vector Machine (SVM), often with a linear kernel. The choice of features is a completely empirical process, mainly based first on linguistic intuition, and then trial and error, and the feature selection is task dependent, implying additional research for each new NLP task. Complex tasks like SRL then require a large number of possibly

5. Available at `http://www.cnts.ua.ac.be/conll2000/chunking/conlleval.txt`.
6. We used the "`-r`" option of the `conlleval` script to get the per-word accuracy, for POS only.
7. Available at `http://www.lsi.upc.es/~srlconll/srlconll-1.1.tgz`.

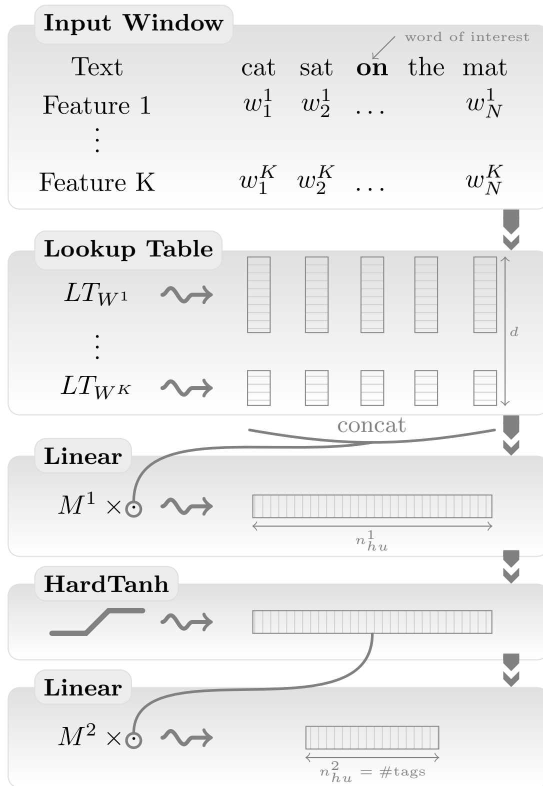


Figure 1: Window approach network.

complex features (e.g., extracted from a parse tree) which can impact the computational cost which might be important for large-scale applications or applications requiring real-time response.

Instead, we advocate a radically different approach: as input we will try to pre-process our features as little as possible and then use a multilayer neural network (NN) architecture, trained in an end-to-end fashion. The architecture takes the input sentence and learns several layers of feature extraction that process the inputs. The features computed by the deep layers of the network are automatically trained by backpropagation to be relevant to the task. We describe in this section a general multilayer architecture suitable for all our NLP tasks, which is generalizable to other NLP tasks as well.

Our architecture is summarized in Figure 1 and Figure 2. The first layer extracts features for each word. The second layer extracts features from a window of words or from the whole sentence, treating it as a *sequence* with local and global structure (i.e., it is not treated like a bag of words). The following layers are standard NN layers.

3.1 Notations

We consider a neural network $f_\theta(\cdot)$, with parameters θ. Any feed-forward neural network with L layers, can be seen as a composition of functions $f_\theta^l(\cdot)$, corresponding to each layer l:

$$f_\theta(\cdot) = f_\theta^L(f_\theta^{L-1}(\dots f_\theta^1(\cdot)\dots)) .$$

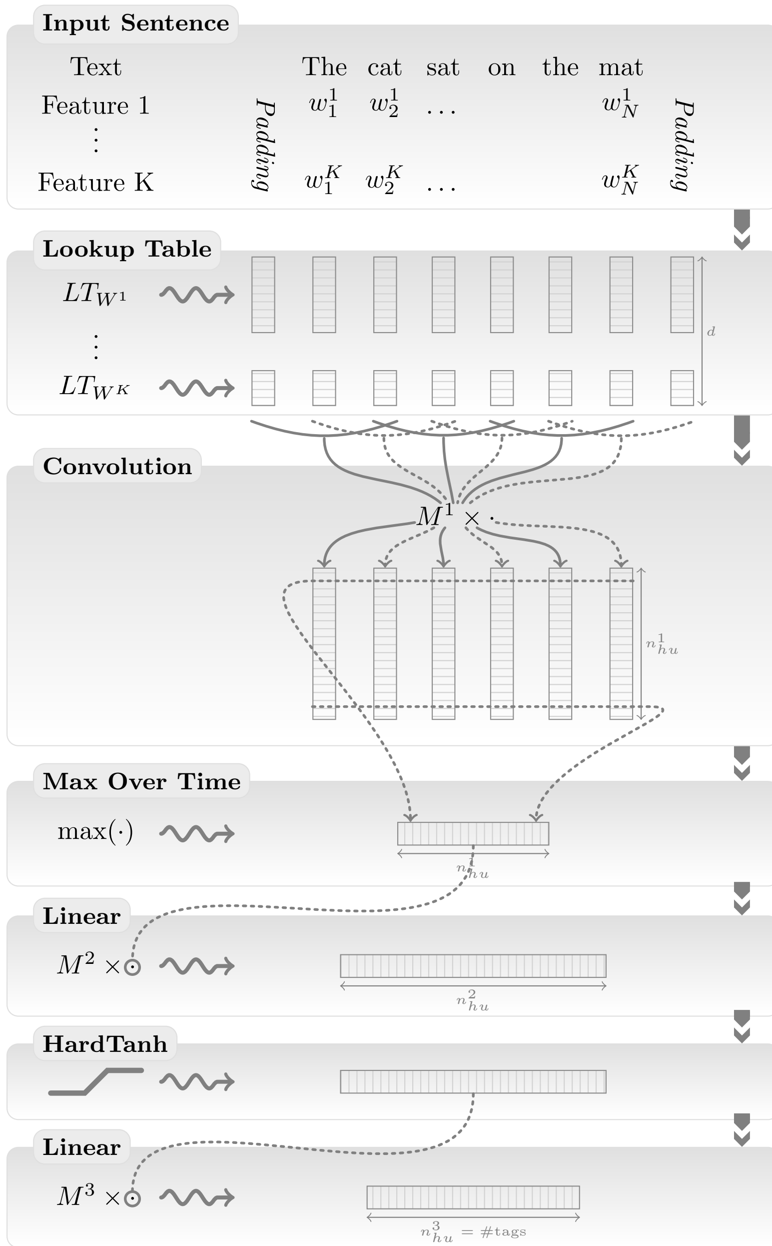


Figure 2: Sentence approach network.

In the following, we will describe each layer we use in our networks shown in Figure 1 and Figure 2. We adopt few notations. Given a matrix A we denote $[A]_{i,\,j}$ the coefficient at row i and column j in the matrix. We also denote $\langle A\rangle_i^{d_{win}}$ the vector obtained by concatenating the d_{win} column vectors around the i^{th} column vector of matrix $A \in \mathbb{R}^{d_1 \times d_2}$:

$$\left[\langle A\rangle_i^{d_{win}}\right]^{\mathrm{T}} = \Big([A]_{1,\,i-d_{win}/2} \dots [A]_{d_1,\,i-d_{win}/2}\,, \dots, [A]_{1,\,i+d_{win}/2} \dots [A]_{d_1,\,i+d_{win}/2}\Big)\,.$$

As a special case, $\langle A \rangle_i^1$ represents the i^{th} column of matrix A. For a vector v, we denote $[v]_i$ the scalar at index i in the vector. Finally, a sequence of element $\{x_1, x_2, \ldots, x_T\}$ is written $[x]_1^T$. The i^{th} element of the sequence is $[x]_i$.

3.2 Transforming Words into Feature Vectors

One of the key points of our architecture is its ability to perform well with the use of (almost[8]) *raw words*. The ability for our method to learn good word representations is thus crucial to our approach. For efficiency, words are fed to our architecture as indices taken from a finite dictionary $\mathcal{D}$. Obviously, a simple index does not carry much useful information about the word. However, the first layer of our network maps each of these word indices into a feature vector, by a lookup table operation. Given a task of interest, a relevant representation of each word is then given by the corresponding lookup table feature vector, which is *trained* by backpropagation, starting from a random initialization.[9] We will see in Section 4 that we can learn very good word representations from unlabeled corpora. Our architecture allow us to take advantage of better trained word representations, by simply initializing the word lookup table with these representations (instead of randomly).

More formally, for each word $w \in \mathcal{D}$, an internal d_{wrd}-dimensional feature vector representation is given by the *lookup table* layer $LT_W(\cdot)$:

$$LT_W(w) = \langle W \rangle_w^1,$$

where $W \in \mathbb{R}^{d_{wrd} \times |\mathcal{D}|}$ is a matrix of parameters to be learned, $\langle W \rangle_w^1 \in \mathbb{R}^{d_{wrd}}$ is the w^{th} column of W and d_{wrd} is the word vector size (a hyper-parameter to be chosen by the user). Given a sentence or any sequence of T words $[w]_1^T$ in $\mathcal{D}$, the lookup table layer applies the same operation for each word in the sequence, producing the following output matrix:

$$LT_W([w]_1^T) = \begin{pmatrix} \langle W \rangle_{[w]_1}^1 & \langle W \rangle_{[w]_2}^1 & \ldots & \langle W \rangle_{[w]_T}^1 \end{pmatrix}. \tag{1}$$

This matrix can then be fed to further neural network layers, as we will see below.

3.2.1 EXTENDING TO ANY DISCRETE FEATURES

One might want to provide features other than words if one suspects that these features are helpful for the task of interest. For example, for the NER task, one could provide a feature which says if a word is in a gazetteer or not. Another common practice is to introduce some basic pre-processing, such as word-stemming or dealing with upper and lower case. In this latter option, the word would be then represented by three discrete features: its lower case stemmed root, its lower case ending, and a capitalization feature.

Generally speaking, we can consider a word as represented by K discrete features $w \in \mathcal{D}^1 \times \cdots \times \mathcal{D}^K$, where $\mathcal{D}^k$ is the dictionary for the k^{th} feature. We associate to each feature a lookup table $LT_{W^k}(\cdot)$, with parameters $W^k \in \mathbb{R}^{d_{wrd}^k \times |\mathcal{D}^k|}$ where $d_{wrd}^k \in \mathbb{N}$ is a user-specified vector size. Given a

8. We did some pre-processing, namely lowercasing and encoding capitalization as another feature. With enough (unlabeled) training data, presumably we could learn a model without this processing. Ideally, an even more raw input would be to learn from letter sequences rather than words, however we felt that this was beyond the scope of this work.
9. As any other neural network layer.

word w, a feature vector of dimension $d_{wrd} = \sum_k d^k_{wrd}$ is then obtained by concatenating all lookup table outputs:

$$LT_{W^1,\ldots,W^K}(w) = \begin{pmatrix} LT_{W^1}(w_1) \\ \vdots \\ LT_{W^K}(w_K) \end{pmatrix} = \begin{pmatrix} \langle W^1 \rangle^1_{w_1} \\ \vdots \\ \langle W^K \rangle^1_{w_K} \end{pmatrix} .$$

The matrix output of the lookup table layer for a sequence of words $[w]_1^T$ is then similar to (1), but where extra rows have been added for each discrete feature:

$$LT_{W^1,\ldots,W^K}([w]_1^T) = \begin{pmatrix} \langle W^1 \rangle^1_{[w_1]_1} & \ldots & \langle W^1 \rangle^1_{[w_1]_T} \\ \vdots & & \vdots \\ \langle W^K \rangle^1_{[w_K]_1} & \ldots & \langle W^K \rangle^1_{[w_K]_T} \end{pmatrix} . \tag{2}$$

These vector features in the lookup table effectively learn features for words in the dictionary. Now, we want to use these trainable features as input to further layers of trainable feature extractors, that can represent groups of words and then finally sentences.

3.3 Extracting Higher Level Features from Word Feature Vectors

Feature vectors produced by the lookup table layer need to be combined in subsequent layers of the neural network to produce a tag decision for each word in the sentence. Producing tags for each element in variable length sequences (here, a sentence is a sequence of words) is a standard problem in machine-learning. We consider two common approaches which tag *one word at the time*: a window approach, and a (convolutional) sentence approach.

3.3.1 WINDOW APPROACH

A window approach assumes the tag of a word depends mainly on its neighboring words. Given a word to tag, we consider a fixed size k_{sz} (a hyper-parameter) window of words around this word. Each word in the window is first passed through the lookup table layer (1) or (2), producing a matrix of word features of fixed size $d_{wrd} \times k_{sz}$. This matrix can be viewed as a $d_{wrd}\, k_{sz}$-dimensional vector by concatenating each column vector, which can be fed to further neural network layers. More formally, the word feature window given by the first network layer can be written as:

$$f^1_\theta = \langle LT_W([w]_1^T) \rangle_t^{d_{win}} = \begin{pmatrix} \langle W \rangle^1_{[w]_{t-d_{win}/2}} \\ \vdots \\ \langle W \rangle^1_{[w]_t} \\ \vdots \\ \langle W \rangle^1_{[w]_{t+d_{win}/2}} \end{pmatrix} . \tag{3}$$

Linear Layer. The fixed size vector f^1_θ can be fed to one or several standard neural network layers which perform affine transformations over their inputs:

$$f^l_\theta = W^l\, f^{l-1}_\theta + b^l , \tag{4}$$

where $W^l \in \mathbb{R}^{n^l_{hu} \times n^{l-1}_{hu}}$ and $b^l \in \mathbb{R}^{n^l_{hu}}$ are the parameters to be *trained*. The hyper-parameter n^l_{hu} is usually called the *number of hidden units* of the l^{th} layer.

HardTanh Layer. Several linear layers are often stacked, interleaved with a non-linearity function, to extract highly non-linear features. If no non-linearity is introduced, our network would be a simple linear model. We chose a "hard" version of the hyperbolic tangent as non-linearity. It has the advantage of being slightly cheaper to compute (compared to the exact hyperbolic tangent), while leaving the generalization performance unchanged (Collobert, 2004). The corresponding layer l applies a HardTanh over its input vector:

$$\left[f_\theta^l\right]_i = \text{HardTanh}(\left[f_\theta^{l-1}\right]_i),$$

where

$$\text{HardTanh}(x) = \begin{cases} -1 & \text{if } x < -1 \\ x & \text{if } -1 <= x <= 1 \\ 1 & \text{if } x > 1 \end{cases} . \tag{5}$$

Scoring. Finally, the output size of the last layer L of our network is equal to the number of possible tags for the task of interest. Each output can be then interpreted as a *score* of the corresponding tag (given the input of the network), thanks to a carefully chosen cost function that we will describe later in this section.

Remark 1 (Border Effects) *The feature window (3) is not well defined for words near the beginning or the end of a sentence. To circumvent this problem, we augment the sentence with a special "PADDING" word replicated $d_{win}/2$ times at the beginning and the end. This is akin to the use of "start" and "stop" symbols in sequence models.*

3.3.2 SENTENCE APPROACH

We will see in the experimental section that a window approach performs well for most natural language processing tasks we are interested in. However this approach fails with SRL, where the tag of a word depends on a verb (or, more correctly, predicate) chosen beforehand in the sentence. If the verb falls outside the window, one cannot expect this word to be tagged correctly. In this particular case, tagging a word requires the consideration of the *whole* sentence. When using neural networks, the natural choice to tackle this problem becomes a convolutional approach, first introduced by Waibel et al. (1989) and also called Time Delay Neural Networks (TDNNs) in the literature.

We describe in detail our convolutional network below. It successively takes the complete sentence, passes it through the lookup table layer (1), produces local features around each word of the sentence thanks to convolutional layers, combines these feature into a global feature vector which can then be fed to standard affine layers (4). In the semantic role labeling case, this operation is performed for each word in the sentence, and for each verb in the sentence. It is thus necessary to encode in the network architecture which verb we are considering in the sentence, and which word we want to tag. For that purpose, each word at position i in the sentence is augmented with two features in the way described in Section 3.2.1. These features encode the relative distances $i - pos_v$ and $i - pos_w$ with respect to the chosen verb at position pos_v, and the word to tag at position pos_w respectively.

Convolutional Layer. A convolutional layer can be seen as a generalization of a window approach: given a sequence represented by columns in a matrix f_θ^{l-1} (in our lookup table matrix (1)), a matrix-vector operation as in (4) is applied to each window of successive windows in the sequence.

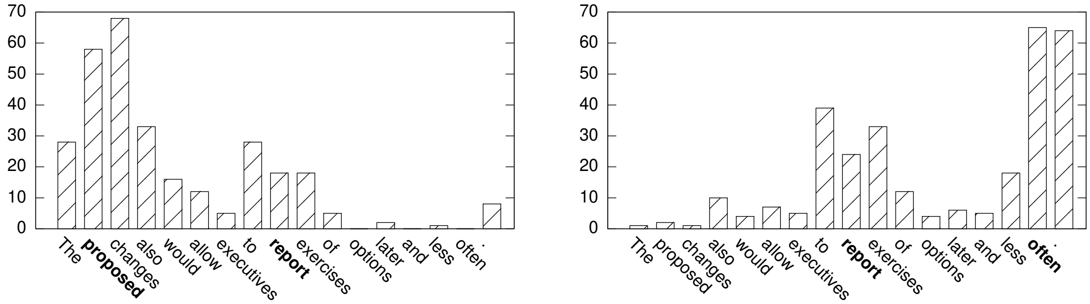


Figure 3: Number of features chosen at each word position by the Max layer. We consider a sentence approach network (Figure 2) trained for SRL. The number of "local" features output by the convolution layer is 300 *per word*. By applying a Max over the sentence, we obtain 300 features for the *whole sentence*. It is interesting to see that the network catches features mostly around the verb of interest (here "report") and word of interest ("proposed" (left) or "often" (right)).

Using previous notations, the t^{th} output column of the l^{th} layer can be computed as:

$$\langle f_\theta^l \rangle_t^1 = W^l \, \langle f_\theta^{l-1} \rangle_t^{d_{win}} + b^l \quad \forall t \,, \tag{6}$$

where the weight matrix W^l is the same across all windows t in the sequence. Convolutional layers extract local features around each window of the given sequence. As for standard affine layers (4), convolutional layers are often stacked to extract higher level features. In this case, each layer must be followed by a non-linearity (5) or the network would be equivalent to one convolutional layer.

Max Layer. The size of the output (6) depends on the number of words in the sentence fed to the network. Local feature vectors extracted by the convolutional layers have to be combined to obtain a global feature vector, with a fixed size independent of the sentence length, in order to apply subsequent standard affine layers. Traditional convolutional networks often apply an average (possibly weighted) or a max operation over the "time" t of the sequence (6). (Here, "time" just means the position in the sentence, this term stems from the use of convolutional layers in, for example, speech data where the sequence occurs over time.) The average operation does not make much sense in our case, as in general most words in the sentence do not have any influence on the semantic role of a given word to tag. Instead, we used a max approach, which forces the network to capture the most useful local features produced by the convolutional layers (see Figure 3), for the task at hand. Given a *matrix* f_θ^{l-1} output by a convolutional layer $l-1$, the Max layer l outputs a *vector* f_θ^l:

$$\left[f_\theta^l \right]_i = \max_t \left[f_\theta^{l-1} \right]_{i,t} \qquad 1 \le i \le n_{hu}^{l-1} \,. \tag{7}$$

This fixed sized global feature vector can be then fed to standard affine network layers (4). As in the window approach, we then finally produce one score per possible tag for the given task.

Remark 2 *The same border effects arise in the convolution operation (6) as in the window approach (3). We again work around this problem by padding the sentences with a special word.*

Scheme	Begin	Inside	End	Single	Other
IOB	B-X	I-X	I-X	B-X	O
IOE	I-X	I-X	E-X	E-X	O
IOBES	B-X	I-X	E-X	S-X	O

Table 3: Various tagging schemes. Each word in a segment labeled "X" is tagged with a prefixed label, depending of the word position in the segment (begin, inside, end). Single word segment labeling is also output. Words not in a labeled segment are labeled "O". Variants of the IOB (and IOE) scheme exist, where the prefix B (or E) is replaced by I for all segments not contiguous with another segment having the same label "X".

3.3.3 TAGGING SCHEMES

As explained earlier, the network output layers compute scores for all the possible tags for the task of interest. In the window approach, these tags apply to the word located in the center of the window. In the (convolutional) sentence approach, these tags apply to the word designated by additional markers in the network input.

The POS task indeed consists of marking the syntactic role of each word. However, the remaining three tasks associate labels with segments of a sentence. This is usually achieved by using special tagging schemes to identify the segment boundaries, as shown in Table 3. Several such schemes have been defined (IOB, IOE, IOBES, ...) without clear conclusion as to which scheme is better in general. State-of-the-art performance is sometimes obtained by combining classifiers trained with different tagging schemes (e.g., Kudo and Matsumoto, 2001).

The ground truth for the NER, CHUNK, and SRL tasks is provided using two different tagging schemes. In order to eliminate this additional source of variations, we have decided to use the most expressive IOBES tagging scheme for all tasks. For instance, in the CHUNK task, we describe noun phrases using four different tags. Tag "S-NP" is used to mark a noun phrase containing a single word. Otherwise tags "B-NP", "I-NP", and "E-NP" are used to mark the first, intermediate and last words of the noun phrase. An additional tag "O" marks words that are not members of a chunk. During testing, these tags are then converted to the original IOB tagging scheme and fed to the standard performance evaluation scripts mentioned in Section 2.5.

3.4 Training

All our neural networks are trained by maximizing a likelihood over the training data, using stochastic gradient ascent. If we denote θ to be all the trainable parameters of the network, which are trained using a training set $\mathcal{T}$ we want to maximize the following log-likelihood with respect to θ:

$$\theta \mapsto \sum_{(x,y)\in\mathcal{T}} \log p(y\,|\,x,\,\theta)\,, \tag{8}$$

where x corresponds to either a training word window or a sentence and its associated features, and y represents the corresponding tag. The probability $p(\cdot)$ is computed from the outputs of the neural network. We will see in this section two ways of interpreting neural network outputs as probabilities.

3.4.1 WORD-LEVEL LOG-LIKELIHOOD

In this approach, each word in a sentence is considered independently. Given an input example x, the network with parameters θ outputs a score $\left[f_\theta(x)\right]_i$, for the i^{th} tag with respect to the task of interest. To simplify the notation, we drop x from now, and we write instead $\left[f_\theta\right]_i$. This score can be interpreted as a conditional tag probability $p(i\,|\,x,\,\theta)$ by applying a softmax (Bridle, 1990) operation over all the tags:

$$p(i\,|\,x,\theta) = \frac{e^{\left[f_\theta\right]_i}}{\sum_j e^{\left[f_\theta\right]_j}}\,. \tag{9}$$

Defining the log-add operation as

$$\operatorname*{logadd}_i z_i = \log\left(\sum_i e^{z_i}\right), \tag{10}$$

we can express the log-likelihood for one training example (x,y) as follows:

$$\log p(y\,|\,x,\,\theta) = [f_\theta]_y - \operatorname*{logadd}_j\,[f_\theta]_j\,. \tag{11}$$

While this training criterion, often referred as *cross-entropy* is widely used for classification problems, it might not be ideal in our case, where there is often a correlation between the tag of a word in a sentence and its neighboring tags. We now describe another common approach for neural networks which enforces dependencies between the predicted tags in a sentence.

3.4.2 SENTENCE-LEVEL LOG-LIKELIHOOD

In tasks like chunking, NER or SRL we know that there are dependencies between word tags in a sentence: not only are tags organized in chunks, but some tags cannot follow other tags. Training using a word-level approach discards this kind of labeling information. We consider a training scheme which takes into account the sentence structure: given the predictions of *all tags* by our network for *all words* in a sentence, and given a score for going from one tag to another tag, we want to encourage valid paths of tags during training, while discouraging all other paths.

We consider the *matrix* of scores $f_\theta([x]_1^T)$ output by the network. As before, we drop the input $[x]_1^T$ for notation simplification. The element $\left[f_\theta\right]_{i,t}$ of the matrix is the score output by the network with parameters θ, for the sentence $[x]_1^T$ and for the i^{th} tag, at the t^{th} word. We introduce a transition score $[A]_{i,\,j}$ for jumping from i to j tags in successive words, and an initial score $[A]_{i,0}$ for starting from the i^{th} tag. As the transition scores are going to be trained (as are all network parameters θ), we define $\tilde{\theta} = \theta \cup \{[A]_{i,\,j}\ \forall i, j\}$. The score of a sentence $[x]_1^T$ along a path of tags $[i]_1^T$ is then given by the sum of transition scores and network scores:

$$s([x]_1^T,\,[i]_1^T,\,\tilde{\theta}) = \sum_{t=1}^{T} \left([A]_{[i]_{t-1},\,[i]_t} + [f_\theta]_{[i]_t,\,t} \right)\,. \tag{12}$$

Exactly as for the word-level likelihood (11), where we were normalizing with respect to all *tags* using a softmax (9), we normalize this score over all possible *tag paths* $[j]_1^T$ using a softmax, and we interpret the resulting ratio as a conditional *tag path* probability. Taking the log, the conditional probability of the true path $[y]_1^T$ is therefore given by:

$$\log p([y]_1^T\,|\,[x]_1^T,\,\tilde{\theta}) = s([x]_1^T,\,[y]_1^T,\,\tilde{\theta}) - \operatorname*{logadd}_{\forall [j]_1^T} s([x]_1^T,\,[j]_1^T,\,\tilde{\theta})\,. \tag{13}$$

While the number of terms in the logadd operation (11) was equal to the number of tags, it grows exponentially with the length of the sentence in (13). Fortunately, one can compute it in linear time with the following standard recursion over t (see Rabiner, 1989), taking advantage of the associativity and distributivity on the semi-ring[10] $(\mathbb{R} \cup \{-\infty\}, \text{logadd}, +)$:

$$\begin{aligned}
\delta_t(k) &\stackrel{\Delta}{=} \underset{\{[j]_1^t \cap [j]_t = k\}}{\text{logadd}} s([x]_1^t, [j]_1^t, \tilde{\theta}) \\
&= \underset{i}{\text{logadd}} \underset{\{[j]_1^t \cap [j]_{t-1} = i \cap [j]_t = k\}}{\text{logadd}} s([x]_1^t, [j]_1^{t-1}, \tilde{\theta}) + [A]_{[j]_{t-1}, k} + [f_\theta]_{k,t} \\
&= \underset{i}{\text{logadd}}\, \delta_{t-1}(i) + [A]_{i,k} + [f_\theta]_{k,t} \\
&= [f_\theta]_{k,t} + \underset{i}{\text{logadd}} \left(\delta_{t-1}(i) + [A]_{i,k} \right) \quad \forall k\,,
\end{aligned} \tag{14}$$

followed by the termination

$$\underset{\forall [j]_1^T}{\text{logadd}}\, s([x]_1^T, [j]_1^T, \tilde{\theta}) = \underset{i}{\text{logadd}}\, \delta_T(i)\,. \tag{15}$$

We can now maximize in (8) the log-likelihood (13) over all the training pairs $([x]_1^T, [y]_1^T)$.

At inference time, given a sentence $[x]_1^T$ to tag, we have to find the best tag path which minimizes the sentence score (12). In other words, we must find

$$\underset{[j]_1^T}{\text{argmax}}\, s([x]_1^T, [j]_1^T, \tilde{\theta})\,.$$

The Viterbi algorithm is the natural choice for this inference. It corresponds to performing the recursion (14) and (15), but where the logadd is replaced by a max, and then tracking back the optimal path through each max.

Remark 3 (Graph Transformer Networks) *Our approach is a particular case of the discriminative forward training for graph transformer networks (GTNs) (Bottou et al., 1997; Le Cun et al., 1998). The log-likelihood (13) can be viewed as the difference between the forward score constrained over the valid paths (in our case there is only the labeled path) and the unconstrained forward score (15).*

Remark 4 (Conditional Random Fields) *An important feature of equation (12) is the absence of normalization. Summing the exponentials $e^{[f_\theta]_{i,t}}$ over all possible tags does not necessarily yield the unity. If this was the case, the scores could be viewed as the logarithms of conditional transition probabilities, and our model would be subject to the label-bias problem that motivates Conditional Random Fields (CRFs) (Lafferty et al., 2001). The denormalized scores should instead be likened to the potential functions of a CRF. In fact, a CRF maximizes the same likelihood (13) using a linear model instead of a nonlinear neural network. CRFs have been widely used in the NLP world, such as for POS tagging (Lafferty et al., 2001), chunking (Sha and Pereira, 2003), NER (McCallum and Li, 2003) or SRL (Cohn and Blunsom, 2005). Compared to such CRFs, we take advantage of the nonlinear network to learn appropriate features for each task of interest.*

10. In other words, read logadd as $\oplus$ and $+$ as $\otimes$.

3.4.3 STOCHASTIC GRADIENT

Maximizing (8) with stochastic gradient (Bottou, 1991) is achieved by iteratively selecting a random example (x, y) and making a gradient step:

$$\theta \longleftarrow \theta + \lambda \frac{\partial \log p(y\,|\,x, \theta)}{\partial \theta}, \tag{16}$$

where λ is a chosen learning rate. Our neural networks described in Figure 1 and Figure 2 are a succession of layers that correspond to successive composition of functions. The neural network is finally composed with the word-level log-likelihood (11), or successively composed in the recursion (14) if using the sentence-level log-likelihood (13). Thus, an *analytical* formulation of the derivative (16) can be computed, by applying the differentiation chain rule through the network, and through the word-level log-likelihood (11) or through the recurrence (14).

Remark 5 (Differentiability) *Our cost functions are differentiable almost everywhere. Non-differentiable points arise because we use a "hard" transfer function (5) and because we use a "max" layer (7) in the sentence approach network. Fortunately, stochastic gradient still converges to a meaningful local minimum despite such minor differentiability problems (Bottou, 1991, 1998). Stochastic gradient iterations that hit a non-differentiability are simply skipped.*

Remark 6 (Modular Approach) *The well known "back-propagation" algorithm (LeCun, 1985; Rumelhart et al., 1986) computes gradients using the chain rule. The chain rule can also be used in a modular implementation.*[11] *Our modules correspond to the boxes in Figure 1 and Figure 2. Given derivatives with respect to its outputs, each module can independently compute derivatives with respect to its inputs and with respect to its trainable parameters, as proposed by Bottou and Gallinari (1991). This allows us to easily build variants of our networks. For details about gradient computations, see Appendix A.*

Remark 7 (Tricks) *Many tricks have been reported for training neural networks (LeCun et al., 1998). Which ones to choose is often confusing. We employed only two of them: the initialization and update of the parameters of each network layer were done according to the "fan-in" of the layer, that is the number of inputs used to compute each output of this layer (Plaut and Hinton, 1987). The fan-in for the lookup table (1), the l^{th} linear layer (4) and the convolution layer (6) are respectively* 1, n_{hu}^{l-1} *and* $d_{win} \times n_{hu}^{l-1}$. *The initial parameters of the network were drawn from a centered uniform distribution, with a variance equal to the inverse of the square-root of the fan-in. The learning rate in (16) was divided by the fan-in, but stays fixed during the training.*

3.5 Supervised Benchmark Results

For POS, chunking and NER tasks, we report results with the window architecture[12] described in Section 3.3.1. The SRL task was trained using the sentence approach (Section 3.3.2). Results are reported in Table 4, in per-word accuracy (PWA) for POS, and F1 score for all the other tasks. We performed experiments both with the word-level log-likelihood (WLL) and with the sentence-level log-likelihood (SLL). The hyper-parameters of our networks are reported in Table 5. All our

11. See `http://torch5.sf.net`.
12. We found that training these tasks with the more complex sentence approach was computationally expensive and offered little performance benefits. Results discussed in Section 5 provide more insight about this decision.

Approach	**POS** (PWA)	**Chunking** (F1)	**NER** (F1)	**SRL** (F1)
Benchmark Systems	97.24	94.29	89.31	77.92
NN+WLL	96.31	89.13	79.53	55.40
NN+SLL	96.37	90.33	81.47	70.99

Table 4: Comparison in generalization performance of benchmark NLP systems with a vanilla neural network (NN) approach, on POS, chunking, NER and SRL tasks. We report results with both the word-level log-likelihood (WLL) and the sentence-level log-likelihood (SLL). Generalization performance is reported in per-word accuracy rate (PWA) for POS and F1 score for other tasks. The NN results are behind the benchmark results, in Section 4 we show how to improve these models using unlabeled data.

Task	Window/Conv. size	Word dim.	Caps dim.	Hidden units	Learning rate
POS	$d_{win} = 5$	$d^0 = 50$	$d^1 = 5$	$n^1_{hu} = 300$	$\lambda = 0.01$
CHUNK	”	”	”	”	”
NER	”	”	”	”	”
SRL	”	”	”	$n^1_{hu} = 300$ $n^2_{hu} = 500$	”

Table 5: Hyper-parameters of our networks. They were chosen by a minimal validation (see Remark 8), preferring identical parameters for most tasks. We report for each task the window size (or convolution size), word feature dimension, capital feature dimension, number of hidden units and learning rate.

networks were fed with two raw text features: lower case words, and a capital letter feature. We chose to consider lower case words to limit the number of words in the dictionary. However, to keep some upper case information lost by this transformation, we added a “caps” feature which tells if each word was in lowercase, was all uppercase, had first letter capital, or had at least one non-initial capital letter. Additionally, all occurrences of sequences of numbers within a word are replaced with the string “NUMBER”, so for example both the words “PS1” and “PS2” would map to the single word “psNUMBER”. We used a dictionary containing the 100,000 most common words in WSJ (case insensitive). Words outside this dictionary were replaced by a single special “RARE” word.

Results show that neural networks “out-of-the-box” are behind baseline benchmark systems. Although the initial performance of our networks falls short from the performance of the CoNLL challenge winners, it compares honorably with the performance of most competitors. The training criterion which takes into account the sentence structure (SLL) seems to boost the performance for the Chunking, NER and SRL tasks, with little advantage for POS. This result is in line with existing NLP studies comparing sentence-level and word-level likelihoods (Liang et al., 2008). The capacity of our network architectures lies mainly in the word lookup table, which contains $50 \times 100,000$ parameters to train. In the WSJ data, 15% of the most common words appear about 90% of the time. Many words appear only a few times. It is thus very difficult to train properly their corresponding

FRANCE 454	JESUS 1973	XBOX 6909	REDDISH 11724	SCRATCHED 29869	MEGABITS 87025
PERSUADE	THICKETS	DECADENT	WIDESCREEN	ODD	PPA
FAW	SAVARY	DIVO	ANTICA	ANCHIETA	UDDIN
BLACKSTOCK	SYMPATHETIC	VERUS	SHABBY	EMIGRATION	BIOLOGICALLY
GIORGI	JFK	OXIDE	AWE	MARKING	KAYAK
SHAHEED	KHWARAZM	URBINA	THUD	HEUER	MCLARENS
RUMELIA	STATIONERY	EPOS	OCCUPANT	SAMBHAJI	GLADWIN
PLANUM	ILIAS	EGLINTON	REVISED	WORSHIPPERS	CENTRALLY
GOA'ULD	GSNUMBER	EDGING	LEAVENED	RITSUKO	INDONESIA
COLLATION	OPERATOR	FRG	PANDIONIDAE	LIFELESS	MONEO
BACHA	W.J.	NAMSOS	SHIRT	MAHAN	NILGIRIS

Table 6: Word embeddings in the word lookup table of a SRL neural network trained from scratch, with a dictionary of size $100,000$. For each column the queried word is followed by its index in the dictionary (higher means more rare) and its 10 nearest neighbors (arbitrarily using the Euclidean metric).

50 dimensional feature vectors in the lookup table. Ideally, we would like semantically similar words to be close in the embedding space represented by the word lookup table: by continuity of the neural network function, tags produced on semantically similar sentences would be similar. We show in Table 6 that it is not the case: neighboring words in the embedding space do not seem to be semantically related.

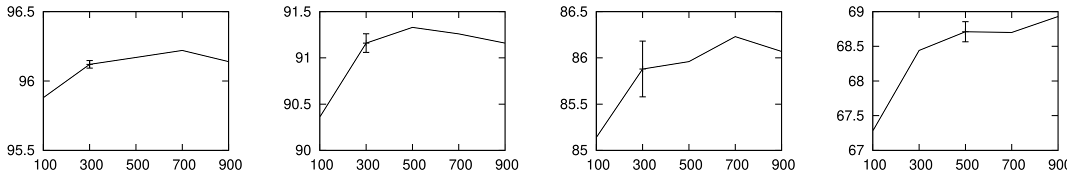
We will focus in the next section on improving these word embeddings by leveraging unlabeled data. We will see our approach results in a performance boost for all tasks.

Remark 8 (Architectures) *In all our experiments in this paper, we tuned the hyper-parameters by trying only a few different architectures by validation. In practice, the choice of hyperparameters such as the number of hidden units, provided they are large enough, has a limited impact on the generalization performance. In Figure 4, we report the F1 score for each task on the validation set, with respect to the number of hidden units. Considering the variance related to the network initialization, we chose the smallest network achieving "reasonable" performance, rather than picking the network achieving the top performance obtained on a single run.*

Remark 9 (Training Time) *Training our network is quite computationally expensive. Chunking and NER take about one hour to train, POS takes few hours, and SRL takes about three days. Training could be faster with a larger learning rate, but we preferred to stick to a small one which works, rather than finding the optimal one for speed. Second order methods (LeCun et al., 1998) could be another speedup technique.*

4. Lots of Unlabeled Data

We would like to obtain word embeddings carrying more syntactic and semantic information than shown in Table 6. Since most of the trainable parameters of our system are associated with the word embeddings, these poor results suggest that we should use considerably more training data.



(a) POS (b) CHUNK (c) NER (d) SRL

Figure 4: F1 score on the *validation* set (y-axis) versus number of hidden units (x-axis) for different tasks trained with the sentence-level likelihood (SLL), as in Table 4. For SRL, we vary in this graph only the number of hidden units in the second layer. The scale is adapted for each task. We show the standard deviation (obtained over 5 runs with different random initialization), for the architecture we picked (300 hidden units for POS, CHUNK and NER, 500 for SRL).

Following our NLP *from scratch* philosophy, we now describe how to dramatically improve these embeddings using large unlabeled data sets. We then use these improved embeddings to initialize the word lookup tables of the networks described in Section 3.5.

4.1 Data Sets

Our first English corpus is the entire English Wikipedia.[13] We have removed all paragraphs containing non-roman characters and all MediaWiki markups. The resulting text was tokenized using the Penn Treebank tokenizer script.[14] The resulting data set contains about 631 million words. As in our previous experiments, we use a dictionary containing the 100,000 most common words in WSJ, with the same processing of capitals and numbers. Again, words outside the dictionary were replaced by the special "RARE" word.

Our second English corpus is composed by adding an extra 221 million words extracted from the Reuters RCV1 (Lewis et al., 2004) data set.[15] We also extended the dictionary to $130,000$ words by adding the $30,000$ most common words in Reuters. This is useful in order to determine whether improvements can be achieved by further increasing the unlabeled data set size.

4.2 Ranking Criterion versus Entropy Criterion

We used these unlabeled data sets to train *language models* that compute *scores* describing the acceptability of a piece of text. These language models are again large neural networks using the window approach described in Section 3.3.1 and in Figure 1. As in the previous section, most of the trainable parameters are located in the lookup tables.

Similar language models were already proposed by Bengio and Ducharme (2001) and Schwenk and Gauvain (2002). Their goal was to estimate the *probability* of a word given the previous words in a sentence. Estimating conditional probabilities suggests a cross-entropy criterion similar to those described in Section 3.4.1. Because the dictionary size is large, computing the normalization term

13. Available at `http://download.wikimedia.org`. We took the November 2007 version.
14. Available at `http://www.cis.upenn.edu/~treebank/tokenization.html`.
15. Now available at `http://trec.nist.gov/data/reuters/reuters.html`.

can be extremely demanding, and sophisticated approximations are required. More importantly for us, neither work leads to significant word embeddings being reported.

Shannon (1951) has estimated the entropy of the English language between 0.6 and 1.3 bits per character by asking human subjects to guess upcoming characters. Cover and King (1978) give a lower bound of 1.25 bits per character using a subtle gambling approach. Meanwhile, using a simple word trigram model, Brown et al. (1992b) reach 1.75 bits per character. Teahan and Cleary (1996) obtain entropies as low as 1.46 bits per character using variable length character *n*-grams. The human subjects rely of course on all their knowledge of the language and of the world. Can we learn the grammatical structure of the English language and the nature of the world by leveraging the 0.2 bits per character that separate human subjects from simple n-gram models? Since such tasks certainly require high capacity models, obtaining sufficiently small confidence intervals on the test set entropy may require prohibitively large training sets.[16] The entropy criterion lacks dynamical range because its numerical value is largely determined by the most frequent phrases. In order to learn syntax, rare but legal phrases are no less significant than common phrases.

It is therefore desirable to define alternative training criteria. We propose here to use a *pairwise ranking* approach (Cohen et al., 1998). We seek a network that computes a higher score when given a legal phrase than when given an incorrect phrase. Because the ranking literature often deals with information retrieval applications, many authors define complex ranking criteria that give more weight to the ordering of the best ranking instances (see Burges et al., 2007; Clémençon and Vayatis, 2007). However, in our case, we do not want to emphasize the most common phrase over the rare but legal phrases. Therefore we use a simple pairwise criterion.

We consider a *window* approach network, as described in Section 3.3.1 and Figure 1, with parameters θ which outputs a score $f_\theta(x)$ given a window of text $x = [w]_1^{d_{win}}$. We minimize the ranking criterion with respect to θ:

$$\theta \mapsto \sum_{x \in \mathcal{X}} \sum_{w \in \mathcal{D}} \max\left\{ 0\,,\, 1 - f_\theta(x) + f_\theta(x^{(w)}) \right\}, \tag{17}$$

where $\mathcal{X}$ is the set of all possible text windows with d_{win} words coming from our training corpus, $\mathcal{D}$ is the dictionary of words, and $x^{(w)}$ denotes the text window obtained by replacing the central word of text window $[w]_1^{d_{win}}$ by the word w.

Okanohara and Tsujii (2007) use a related approach to avoiding the entropy criteria using a binary classification approach (correct/incorrect phrase). Their work focuses on using a kernel classifier, and not on learning word embeddings as we do here. Smith and Eisner (2005) also propose a contrastive criterion which estimates the likelihood of the data conditioned to a "negative" neighborhood. They consider various data neighborhoods, including sentences of length d_{win} drawn from $\mathcal{D}^{d_{win}}$. Their goal was however to perform well on some tagging task on fully unsupervised data, rather than obtaining generic word embeddings useful for other tasks.

4.3 Training Language Models

The language model network was trained by stochastic gradient minimization of the ranking criterion (17), sampling a sentence-word pair (s, w) at each iteration.

16. However, Klein and Manning (2002) describe a rare example of realistic unsupervised grammar induction using a cross-entropy approach on binary-branching parsing trees, that is, by forcing the system to generate a hierarchical representation.

Since training times for such large scale systems are counted in weeks, it is not feasible to try many combinations of hyperparameters. It also makes sense to speed up the training time by initializing new networks with the embeddings computed by earlier networks. In particular, we found it expedient to train a succession of networks using increasingly large dictionaries, each network being initialized with the embeddings of the previous network. Successive dictionary sizes and switching times are chosen arbitrarily. Bengio et al. (2009) provides a more detailed discussion of this, the (as yet, poorly understood) "curriculum" process.

For the purposes of model selection we use the process of "breeding". The idea of breeding is instead of trying a full grid search of possible values (which we did not have enough computing power for) to search for the parameters in analogy to breeding biological cell lines. Within each line, child networks are initialized with the embeddings of their parents and trained on increasingly rich data sets with sometimes different parameters. That is, suppose we have k processors, which is much less than the possible set of parameters one would like to try. One chooses k initial parameter choices from the large set, and trains these on the k processors. In our case, possible parameters to adjust are: the learning rate λ, the word embedding dimensions d, number of hidden units n_{hu}^1 and input window size d_{win}. One then trains each of these models in an online fashion for a certain amount of time (i.e., a few days), and then selects the best ones using the validation set error rate. That is, breeding decisions were made on the basis of the value of the ranking criterion (17) estimated on a validation set composed of one million words held out from the Wikipedia corpus. In the next breeding iteration, one then chooses another set of k parameters from the possible grid of values that permute slightly the most successful candidates from the previous round. As many of these parameter choices can share weights, we can effectively continue online training retaining some of the learning from the previous iterations.

Very long training times make such strategies necessary for the foreseeable future: if we had been given computers ten times faster, we probably would have found uses for data sets ten times bigger. However, we should say we believe that although we ended up with a particular choice of parameters, many other choices are almost equally as good, although perhaps there are others that are better as we could not do a full grid search.

In the following subsections, we report results obtained with two trained language models. The results achieved by these two models are representative of those achieved by networks trained on the full corpora.

- Language model LM1 has a window size $d_{win} = 11$ and a hidden layer with $n_{hu}^1 = 100$ units. The embedding layers were dimensioned like those of the supervised networks (Table 5). Model LM1 was trained on our first English corpus (Wikipedia) using successive dictionaries composed of the 5000, $10,000$, $30,000$, $50,000$ and finally $100,000$ most common WSJ words. The total training time was about four weeks.

- Language model LM2 has the same dimensions. It was initialized with the embeddings of LM1, and trained for an additional three weeks on our second English corpus (Wikipedia+Reuters) using a dictionary size of 130,000 words.

4.4 Embeddings

Both networks produce much more appealing word embeddings than in Section 3.5. Table 7 shows the ten nearest neighbors of a few randomly chosen query words for the LM1 model. The syntactic

FRANCE 454	JESUS 1973	XBOX 6909	REDDISH 11724	SCRATCHED 29869	MEGABITS 87025
AUSTRIA	GOD	AMIGA	GREENISH	NAILED	OCTETS
BELGIUM	SATI	PLAYSTATION	BLUISH	SMASHED	MB/S
GERMANY	CHRIST	MSX	PINKISH	PUNCHED	BIT/S
ITALY	SATAN	IPOD	PURPLISH	POPPED	BAUD
GREECE	KALI	SEGA	BROWNISH	CRIMPED	CARATS
SWEDEN	INDRA	PSNUMBER	GREYISH	SCRAPED	KBIT/S
NORWAY	VISHNU	HD	GRAYISH	SCREWED	MEGAHERTZ
EUROPE	ANANDA	DREAMCAST	WHITISH	SECTIONED	MEGAPIXELS
HUNGARY	PARVATI	GEFORCE	SILVERY	SLASHED	GBIT/S
SWITZERLAND	GRACE	CAPCOM	YELLOWISH	RIPPED	AMPERES

Table 7: Word embeddings in the word lookup table of the language model neural network LM1 trained with a dictionary of size $100,000$. For each column the queried word is followed by its index in the dictionary (higher means more rare) and its 10 nearest neighbors (using the Euclidean metric, which was chosen arbitrarily).

and semantic properties of the neighbors are clearly related to those of the query word. These results are far more satisfactory than those reported in Table 7 for embeddings obtained using purely supervised training of the benchmark NLP tasks.

4.5 Semi-supervised Benchmark Results

Semi-supervised learning has been the object of much attention during the last few years (see Chapelle et al., 2006). Previous semi-supervised approaches for NLP can be roughly categorized as follows:

- Ad-hoc approaches such as Rosenfeld and Feldman (2007) for relation extraction.
- Self-training approaches, such as Ueffing et al. (2007) for machine translation, and McClosky et al. (2006) for parsing. These methods augment the labeled training set with examples from the unlabeled data set using the labels predicted by the model itself. Transductive approaches, such as Joachims (1999) for text classification can be viewed as a refined form of self-training.
- Parameter sharing approaches such as Ando and Zhang (2005); Suzuki and Isozaki (2008). Ando and Zhang propose a multi-task approach where they jointly train models sharing certain parameters. They train POS and NER models together with a language model (trained on 15 million words) consisting of predicting words given the surrounding tokens. Suzuki and Isozaki embed a generative model (Hidden Markov Model) inside a CRF for POS, Chunking and NER. The generative model is trained on one billion words. These approaches should be seen as a linear counterpart of our work. Using multilayer models vastly expands the parameter sharing opportunities (see Section 5).

Our approach simply consists of initializing the word lookup tables of the supervised networks with the embeddings computed by the language models. Supervised training is then performed as in Section 3.5. In particular the supervised training stage is free to modify the lookup tables. This sequential approach is computationally convenient because it separates the lengthy training of the

Approach	**POS** (PWA)	**CHUNK** (F1)	**NER** (F1)	**SRL** (F1)
Benchmark Systems	97.24	94.29	89.31	77.92
NN+WLL	96.31	89.13	79.53	55.40
NN+SLL	96.37	90.33	81.47	70.99
NN+WLL+LM1	97.05	91.91	85.68	58.18
NN+SLL+LM1	97.10	93.65	87.58	73.84
NN+WLL+LM2	97.14	92.04	86.96	58.34
NN+SLL+LM2	97.20	93.63	88.67	74.15

Table 8: Comparison in generalization performance of benchmark NLP systems with our (NN) approach on POS, chunking, NER and SRL tasks. We report results with both the word-level log-likelihood (WLL) and the sentence-level log-likelihood (SLL). We report with (LMn) performance of the networks trained from the language model embeddings (Table 7). Generalization performance is reported in per-word accuracy (PWA) for POS and F1 score for other tasks.

language models from the relatively fast training of the supervised networks. Once the language models are trained, we can perform multiple experiments on the supervised networks in a relatively short time. Note that our procedure is clearly linked to the (semi-supervised) deep learning procedures of Hinton et al. (2006), Bengio et al. (2007) and Weston et al. (2008).

Table 8 clearly shows that this simple initialization significantly boosts the generalization performance of the supervised networks for each task. It is worth mentioning the larger language model led to even better performance. This suggests that we could still take advantage of even bigger unlabeled data sets.

4.6 Ranking and Language

There is a large agreement in the NLP community that syntax is a necessary prerequisite for semantic role labeling (Gildea and Palmer, 2002). This is why state-of-the-art semantic role labeling systems thoroughly exploit multiple parse trees. The parsers themselves (Charniak, 2000; Collins, 1999) contain considerable prior information about syntax (one can think of this as a kind of informed pre-processing).

Our system does not use such parse trees because we attempt to learn this information from the unlabeled data set. It is therefore legitimate to question whether our ranking criterion (17) has the conceptual capability to capture such a rich hierarchical information. At first glance, the ranking task appears unrelated to the induction of probabilistic grammars that underly standard parsing algorithms. The lack of hierarchical representation seems a fatal flaw (Chomsky, 1956).

However, ranking is closely related to an alternative description of the language structure: *operator grammars* (Harris, 1968). Instead of directly studying the structure of a sentence, Harris defines an algebraic structure on the space of all sentences. Starting from a couple of elementary sentence forms, sentences are described by the successive application of sentence transformation operators. The sentence structure is revealed as a side effect of the successive transformations. Sentence transformations can also have a semantic interpretation.

In the spirit of structural linguistics, Harris describes procedures to discover sentence transformation operators by leveraging the statistical regularities of the language. Such procedures are obviously useful for machine learning approaches. In particular, he proposes a test to decide whether two sentences forms are semantically related by a transformation operator. He first defines a ranking criterion (Harris, 1968, Section 4.1):

> "Starting for convenience with very short sentence forms, say *ABC*, we choose a particular word choice for all the classes, say B_qC_q, except one, in this case A; for every pair of members A_i, A_j of that word class we ask how the sentence formed with one of the members, that is, $A_iB_qC_q$ compares as to acceptability with the sentence formed with the other member, that is, $A_jB_qC_q$."

These *gradings* are then used to compare sentence forms:

> "It now turns out that, given the graded n-tuples of words for a particular sentence form, we can find other sentences forms of the same word classes in which the same n-tuples of words produce the same grading of sentences."

This is an indication that these two sentence forms exploit common words with the same syntactic function and possibly the same meaning. This observation forms the empirical basis for the construction of operator grammars that describe real-world natural languages such as English.

Therefore there are solid reasons to believe that the ranking criterion (17) has the conceptual potential to capture strong syntactic and semantic information. On the other hand, the structure of our language models is probably too restrictive for such goals, and our current approach only exploits the word embeddings discovered during training.

5. Multi-Task Learning

It is generally accepted that features *trained* for one task can be useful for *related tasks*. This idea was already exploited in the previous section when certain language model features, namely the word embeddings, were used to initialize the supervised networks.

Multi-task learning (MTL) leverages this idea in a more systematic way. Models for all tasks of interests are *jointly trained* with an additional linkage between their trainable parameters in the hope of improving the generalization error. This linkage can take the form of a regularization term in the joint cost function that biases the models towards common representations. A much simpler approach consists in having the models *share certain parameters* defined a priori. Multi-task learning has a long history in machine learning and neural networks. Caruana (1997) gives a good overview of these past efforts.

5.1 Joint Decoding versus Joint Training

Multitask approaches do not necessarily involve joint training. For instance, modern speech recognition systems use Bayes rule to combine the outputs of an acoustic model trained on speech data and a language model trained on phonetic or textual corpora (Jelinek, 1976). This joint decoding approach has been successfully applied to structurally more complex NLP tasks. Sutton and McCallum (2005b) obtain improved results by combining the predictions of independently trained CRF models using a joint decoding process at test time that requires more sophisticated probabilistic inference techniques. On the other hand, Sutton and McCallum (2005a) obtain results somewhat below the state-of-the-art using joint decoding for SRL and syntactic parsing. Musillo and Merlo (2006) also describe a negative result at the same joint task.

Joint decoding invariably works by considering additional probabilistic dependency paths between the models. Therefore it defines an implicit supermodel that describes all the tasks in the same probabilistic framework. Separately training a submodel only makes sense when the training data blocks these additional dependency paths (in the sense of d-separation, Pearl, 1988). This implies that, without joint training, the additional dependency paths cannot directly involve unobserved variables. Therefore, the natural idea of discovering common internal representations across tasks requires joint training.

Joint training is relatively straightforward when the training sets for the individual tasks contain the same patterns with different labels. It is then sufficient to train a model that computes multiple outputs for each pattern (Suddarth and Holden, 1991). Using this scheme, Sutton et al. (2007) demonstrate improvements on POS tagging and noun-phrase chunking using jointly trained CRFs. However the joint labeling requirement is a limitation because such data is not often available. Miller et al. (2000) achieves performance improvements by jointly training NER, parsing, and relation extraction in a statistical parsing model. The joint labeling requirement problem was weakened using a predictor to fill in the missing annotations.

Ando and Zhang (2005) propose a setup that works around the joint labeling requirements. They define linear models of the form $f_i(x) = w_i^\top \Phi(x) + v_i^\top \Theta\Psi(x)$ where f_i is the classifier for the i-th task with parameters w_i and v_i. Notations $\Phi(x)$ and $\Psi(x)$ represent engineered features for the pattern x. Matrix Θ maps the $\Psi(x)$ features into a low dimensional subspace common across all tasks. Each task is trained using its own examples without a joint labeling requirement. The learning procedure alternates the optimization of w_i and v_i for each task, and the optimization of Θ to minimize the average loss for all examples in all tasks. The authors also consider auxiliary unsupervised tasks for predicting substructures. They report excellent results on several tasks, including POS and NER.

5.2 Multi-Task Benchmark Results

Table 9 reports results obtained by jointly trained models for the POS, CHUNK, NER and SRL tasks using the same setup as Section 4.5. We trained jointly POS, CHUNK and NER using the window approach network. As we mentioned earlier, SRL can be trained only with the sentence approach network, due to long-range dependencies related to the verb predicate. We thus performed additional experiments, where all four tasks were trained using the sentence approach network. In both cases, all models share the lookup table parameters (2). The parameters of the first linear layers (4) were shared in the window approach case (see Figure 5), and the first the convolution layer parameters (6) were shared in the sentence approach networks.

For the window approach, best results were obtained by enlarging the first hidden layer size to $n_{hu}^1 = 500$ (chosen by validation) in order to account for its shared responsibilities. We used the same architecture as SRL for the sentence approach network. The word embedding dimension was kept constant $d^0 = 50$ in order to reuse the language models of Section 4.5.

Training was achieved by minimizing the loss averaged across all tasks. This is easily achieved with stochastic gradient by alternatively picking examples for each task and applying (16) to all the parameters of the corresponding model, including the shared parameters. Note that this gives each task equal weight. Since each task uses the training sets described in Table 1, it is worth noticing

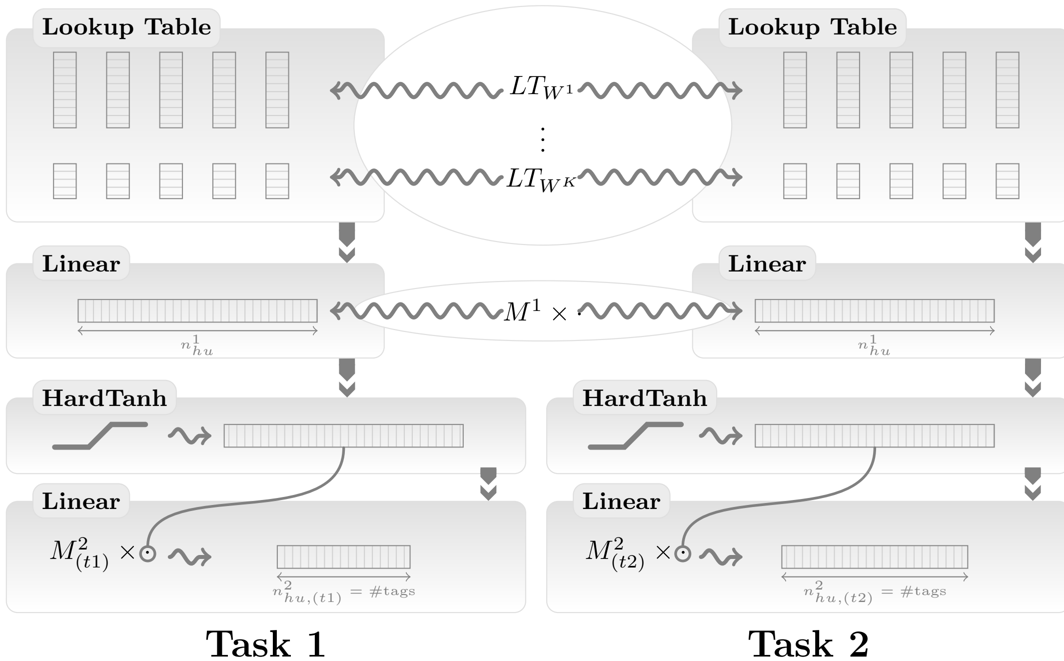


Figure 5: Example of multitasking with NN. Task 1 and Task 2 are two tasks trained with the window approach architecture presented in Figure 1. Lookup tables as well as the first hidden layer are shared. The last layer is task specific. The principle is the same with more than two tasks.

that examples can come from quite different data sets. The generalization performance for each task was measured using the traditional testing data specified in Table 1. Fortunately, none of the training and test sets overlap across tasks.

It is worth mentioning that MTL can produce a single *unified network* that performs well for all these tasks using the sentence approach. However this unified network only leads to marginal improvements over using a separate network for each task: the most important MTL task appears to be the unsupervised learning of the word embeddings. As explained before, simple computational considerations led us to train the POS, Chunking, and NER tasks using the window approach. The baseline results in Table 9 also show that using the sentence approach for the POS, Chunking, and NER tasks yields no performance improvement (or degradation) over the window approach. The next section shows we can leverage known correlations between tasks in more direct manner.

6. The Temptation

Results so far have been obtained by staying (almost[17]) true to our *from scratch* philosophy. We have so far avoided specializing our architecture for any task, disregarding a lot of useful *a priori*

17. We did some basic preprocessing of the raw input words as described in Section 3.5, hence the "almost" in the title of this article. A completely from scratch approach would presumably not know anything about words at all and would work from letters only (or, taken to a further extreme, from speech or optical character recognition, as humans do).

Approach	**POS** (PWA)	**CHUNK** (F1)	**NER** (F1)	**SRL** (F1)
Benchmark Systems	97.24	94.29	89.31	77.92
		Window Approach		
NN+SLL+LM2	97.20	93.63	88.67	–
NN+SLL+LM2+MTL	97.22	94.10	88.62	–
		Sentence Approach		
NN+SLL+LM2	97.12	93.37	88.78	74.15
NN+SLL+LM2+MTL	97.22	93.75	88.27	74.29

Table 9: Effect of multi-tasking on our neural architectures. We trained POS, CHUNK NER in a MTL way, both for the window and sentence network approaches. SRL was only included in the sentence approach joint training. As a baseline, we show previous results of our window approach system, as well as additional results for our sentence approach system, when trained separately on each task. Benchmark system performance is also given for comparison.

Approach	**POS** (PWA)	**CHUNK** (F1)	**NER** (F1)	**SRL**
Benchmark Systems	97.24	94.29	89.31	77.92
NN+SLL+LM2	97.20	93.63	88.67	74.15
NN+SLL+LM2+Suffix2	97.29	–	–	–
NN+SLL+LM2+Gazetteer	–	–	89.59	–
NN+SLL+LM2+POS	–	94.32	88.67	–
NN+SLL+LM2+CHUNK	–	–	–	74.72

Table 10: Comparison in generalization performance of benchmark NLP systems with our neural networks (NNs) using increasing task-specific engineering. We report results obtained with a network trained without the extra task-specific features (Section 5) and with the extra task-specific features described in Section 6. The POS network was trained with two character word suffixes; the NER network was trained using the small CoNLL 2003 gazetteer; the CHUNK and NER networks were trained with additional POS features; and finally, the SRL network was trained with additional CHUNK features.

NLP knowledge. We have shown that, thanks to large unlabeled data sets, our generic neural networks can still achieve close to state-of-the-art performance by discovering useful features. This section explores what happens when we increase the level of task-specific engineering in our systems by incorporating some common techniques from the NLP literature. We often obtain further improvements. These figures are useful to quantify how far we went by leveraging large data sets instead of relying on a priori knowledge.

6.1 Suffix Features

Word suffixes in many western languages are strong predictors of the syntactic function of the word and therefore can benefit the POS system. For instance, Ratnaparkhi (1996) uses inputs representing word suffixes and prefixes up to four characters. We achieve this in the POS task by adding discrete word features (Section 3.2.1) representing the last two characters of every word. The size of the suffix dictionary was 455. This led to a small improvement of the POS performance (Table 10, row NN+SLL+LM2+Suffix2). We also tried suffixes obtained with the Porter (1980) stemmer and obtained the same performance as when using two character suffixes.

6.2 Gazetteers

State-of-the-art NER systems often use a large dictionary containing well known named entities (e.g., Florian et al., 2003). We restricted ourselves to the gazetteer provided by the CoNLL challenge, containing $8,000$ locations, person names, organizations, and miscellaneous entities. We trained a NER network with 4 additional word features indicating (feature "on" or "off") whether the word is found in the gazetteer under one of these four categories. The gazetteer includes not only words, but also chunks of words. If a sentence chunk is found in the gazetteer, then all words in the chunk have their corresponding gazetteer feature turned to "on". The resulting system displays a clear performance improvement (Table 10, row NN+SLL+LM2+Gazetteer), slightly outperforming the baseline. A plausible explanation of this large boost over the network using only the language model is that gazetteers include word chunks, while we use only the word representation of our language model. For example, "united" and "bicycle" seen separately are likely to be non-entities, while "united bicycle" might be an entity, but catching it would require higher level representations of our language model.

6.3 Cascading

When one considers related tasks, it is reasonable to assume that tags obtained for one task can be useful for taking decisions in the other tasks. Conventional NLP systems often use features obtained from the output of other preexisting NLP systems. For instance, Shen and Sarkar (2005) describe a chunking system that uses POS tags as input; Florian et al. (2003) describes a NER system whose inputs include POS and CHUNK tags, as well as the output of two other NER classifiers. State-of-the-art SRL systems exploit parse trees (Gildea and Palmer, 2002; Punyakanok et al., 2005), related to CHUNK tags, and built using POS tags (Charniak, 2000; Collins, 1999).

Table 10 reports results obtained for the CHUNK and NER tasks by adding discrete word features (Section 3.2.1) representing the POS tags. In order to facilitate comparisons, instead of using the more accurate tags from our POS network, we use for each task the POS tags provided by the corresponding CoNLL challenge. We also report results obtained for the SRL task by adding word features representing the CHUNK tags (also provided by the CoNLL challenge). We consistently obtain moderate improvements.

6.4 Ensembles

Constructing ensembles of classifiers is a proven way to trade computational efficiency for generalization performance (Bell et al., 2007). Therefore it is not surprising that many NLP systems achieve state-of-the-art performance by combining the outputs of multiple classifiers. For instance, Kudo

Approach		**POS** (PWA)	**CHUNK** (F1)	**NER** (F1)
Benchmark Systems		97.24	94.29	89.31
NN+SLL+LM2+POS	worst	97.29	93.99	89.35
NN+SLL+LM2+POS	mean	97.31	94.17	89.65
NN+SLL+LM2+POS	best	97.35	94.32	89.86
NN+SLL+LM2+POS	voting ensemble	97.37	94.34	89.70
NN+SLL+LM2+POS	joined ensemble	97.30	94.35	89.67

Table 11: Comparison in generalization performance for POS, CHUNK and NER tasks of the networks obtained using by combining ten training runs with different initialization.

and Matsumoto (2001) use an ensemble of classifiers trained with different tagging conventions (see Section 3.3.3). Winning a challenge is of course a legitimate objective. Yet it is often difficult to figure out which ideas are most responsible for the state-of-the-art performance of a large ensemble.

Because neural networks are nonconvex, training runs with different initial parameters usually give different solutions. Table 11 reports results obtained for the CHUNK and NER task after ten training runs with random initial parameters. Voting the ten network outputs on a per tag basis ("voting ensemble") leads to a small improvement over the average network performance. We have also tried a more sophisticated ensemble approach: the ten network output scores (before sentence-level likelihood) were combined with an additional linear layer (4) and then fed to a new sentence-level likelihood (13). The parameters of the combining layers were then trained on the existing training set, while keeping the ten networks fixed ("joined ensemble"). This approach did not improve on simple voting.

These ensembles come of course at the expense of a ten fold increase of the running time. On the other hand, multiple training times could be improved using smart sampling strategies (Neal, 1996).

We can also observe that the performance variability among the ten networks is not very large. The local minima found by the training algorithm are usually good local minima, thanks to the oversized parameter space and to the noise induced by the stochastic gradient procedure (LeCun et al., 1998). In order to reduce the variance in our experimental results, we always use the same initial parameters for networks trained on the same task (except of course for the results reported in Table 11.)

6.5 Parsing

Gildea and Palmer (2002) and Punyakanok et al. (2005) offer several arguments suggesting that syntactic parsing is a necessary prerequisite for the SRL task. The CoNLL 2005 SRL benchmark task provides parse trees computed using *both* the Charniak (2000) and Collins (1999) parsers. State-of-the-art systems often exploit additional parse trees such as the k top ranking parse trees (Koomen et al., 2005; Haghighi et al., 2005).

In contrast our SRL networks so far do not use parse trees at all. They rely instead on internal representations transferred from a language model trained with an objective function that captures

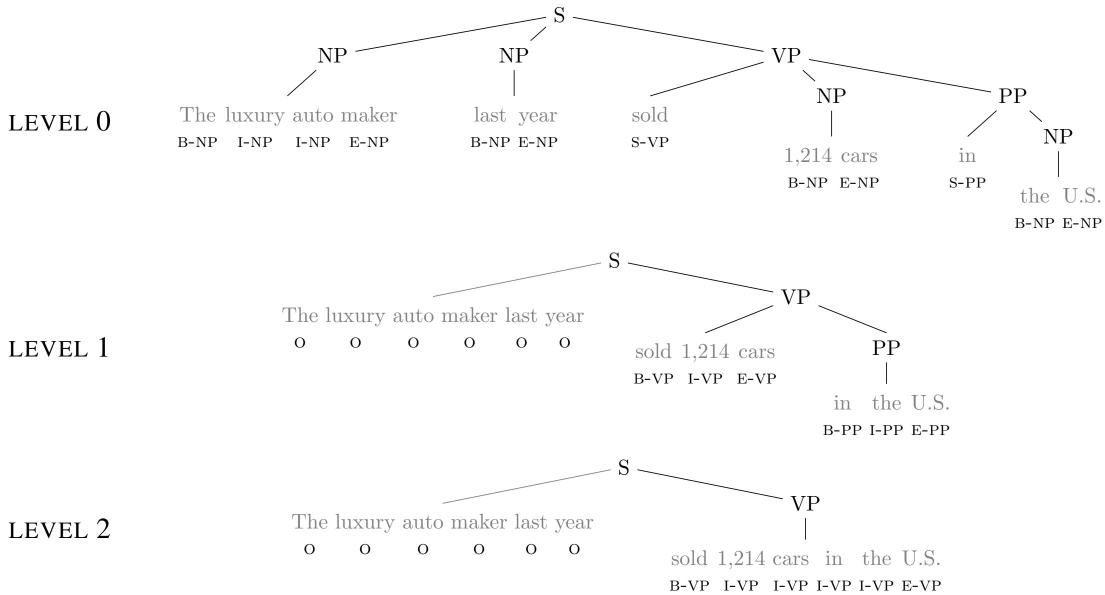


Figure 6: Charniak parse tree for the sentence *"The luxury auto maker last year sold 1,214 cars in the U.S."*. Level 0 is the original tree. Levels 1 to 4 are obtained by successively collapsing terminal tree branches. For each level, words receive tags describing the segment associated with the corresponding leaf. All words receive tag "O" at level 3 in this example.

a lot of syntactic information (see Section 4.6). It is therefore legitimate to question whether this approach is an acceptable lightweight replacement for parse trees.

We answer this question by providing parse tree information as additional input features to our system.[18] We have limited ourselves to the Charniak parse tree provided with the CoNLL 2005 data. Considering that a node in a syntactic parse tree assigns a label to a segment of the parsed sentence, we propose a way to feed (partially) this labeled segmentation to our network, through additional lookup tables. Each of these lookup tables encode *labeled* segments of each parse tree level (up to a certain depth). The labeled segments are fed to the network following a IOBES tagging scheme (see Sections 3.3.3 and 3.2.1). As there are 40 different phrase labels in WSJ, each additional tree-related lookup tables has 161 entries ($40 \times 4 + 1$) corresponding to the IBES segment tags, plus the extra O tag.

We call level 0 the information associated with the leaves of the original Charniak parse tree. The lookup table for level 0 encodes the corresponding IOBES phrase tags for each words. We obtain levels 1 to 4 by repeatedly trimming the leaves as shown in Figure 6. We labeled "O" words belonging to the root node "S", or all words of the sentence if the root itself has been trimmed.

Experiments were performed using the LM2 language model using the same network architectures (see Table 5) and using additional lookup tables of dimension 5 for each parse tree level. Table 12 reports the performance improvements obtained by providing increasing levels of parse

18. In a more recent work (Collobert, 2011), we propose an extension of this approach for the generation of full syntactic parse trees, using a recurrent version of our architecture.

Approach	SRL	
	(valid)	(test)
Benchmark System (six parse trees)	77.35	77.92
Benchmark System (top Charniak parse tree only)	74.76	–
NN+SLL+LM2	72.29	74.15
NN+SLL+LM2+Charniak (level 0 only)	74.44	75.65
NN+SLL+LM2+Charniak (levels 0 & 1)	74.50	75.81
NN+SLL+LM2+Charniak (levels 0 to 2)	75.09	76.05
NN+SLL+LM2+Charniak (levels 0 to 3)	75.12	75.89
NN+SLL+LM2+Charniak (levels 0 to 4)	75.42	76.06
NN+SLL+LM2+CHUNK	–	74.72
NN+SLL+LM2+PT0	–	75.49

Table 12: Generalization performance on the SRL task of our NN architecture compared with the benchmark system. We show performance of our system fed with different levels of depth of the Charniak parse tree. We report previous results of our architecture with no parse tree as a baseline. Koomen et al. (2005) report test and validation performance using six parse trees, as well as validation performance using only the top Charniak parse tree. For comparison purposes, we hence also report validation performance. Finally, we report our performance with the CHUNK feature, and compare it against a level 0 feature PT0 obtained by our network.

tree information. Level 0 alone increases the F1 score by almost 1.5%. Additional levels yield diminishing returns. The top performance reaches 76.06% F1 score. This is not too far from the state-of-the-art system which we note uses six parse trees instead of one. Koomen et al. (2005) also report a 74.76% F1 score on the validation set using only the Charniak parse tree. Using the first three parse tree levels, we reach this performance on the validation set. These results corroborate findings in the NLP literature (Gildea and Palmer, 2002; Punyakanok et al., 2005) showing that parsing is important for the SRL task.

We also reported in Table 12 our previous performance obtained with the CHUNK feature (see Table 10). It is surprising to observe that adding chunking features into the semantic role labeling network performs significantly worse than adding features describing the level 0 of the Charniak parse tree (Table 12). Indeed, if we ignore the label prefixes "BIES" defining the segmentation, the parse tree leaves (at level 0) and the chunking have identical labeling. However, the parse trees identify leaf sentence segments that are often smaller than those identified by the chunking tags, as shown by Hollingshead et al. (2005).[19] Instead of relying on Charniak parser, we chose to train a second chunking network to identify the segments delimited by the leaves of the Penn Treebank parse trees (level 0). Our network achieved 92.25% F1 score on this task (we call it PT0), while we evaluated Charniak performance as 91.94% on the same task. As shown in Table 12, feeding our

19. As in Hollingshead et al. (2005), consider the sentence and chunk labels "(NP They) (VP are starting to buy) (NP growth stocks)". The parse tree can be written as "(S (NP They) (VP are (VP starting (S (VP to (VP buy (NP growth stocks)))))))". The tree leaves segmentation is thus given by "(NP They) (VP are) (VP starting) (VP to) (VP buy) (NP growth stocks)".

own PT0 prediction into the SRL system gives similar performance to using Charniak predictions, and is consistently better than the CHUNK feature.

6.6 Word Representations

We have described how we induced useful word embeddings by applying our architecture to a language modeling task trained using a large amount of unlabeled text data. These embeddings improve the generalization performance on all tasks (Section 4.) The literature describes other ways to induce word representations. Mnih and Hinton (2007) proposed a related language model approach inspired from Restricted Boltzmann Machines. However, word representations are perhaps more commonly inferred from *n*-gram language modeling rather than purely continuous language models. One popular approach is the Brown clustering algorithm (Brown et al., 1992a), which builds hierarchical word clusters by maximizing the bigram's mutual information. The induced word representation has been used with success in a wide variety of NLP tasks, including POS (Schütze, 1995), NER (Miller et al., 2004; Ratinov and Roth, 2009), or parsing (Koo et al., 2008). Other related approaches exist, like phrase clustering (Lin and Wu, 2009) which has been shown to work well for NER. Finally, Huang and Yates (2009) have recently proposed a smoothed language modeling approach based on a Hidden Markov Model, with success on POS and Chunking tasks.

While a comparison of all these word representations is beyond the scope of this paper, it is rather fair to question the quality of our word embeddings compared to a popular NLP approach. In this section, we report a comparison of our word embeddings against Brown clusters, when used as features into our neural network architecture. We report as baseline previous results where our word embeddings are *fine-tuned* for each task. We also report performance when our embeddings are kept fixed during task-specific training. Since *convex* machine learning algorithms are common practice in NLP, we finally report performances for the convex version of our architecture.

For the convex experiments, we considered the linear version of our neural networks (instead of having several linear layers interleaved with a non-linearity). While we always picked the sentence approach for SRL, we had to consider the window approach in this particular convex setup, as the sentence approach network (see Figure 2) includes a Max layer. Having only one linear layer in our neural network is not enough to make our architecture convex: all lookup-tables (for each discrete feature) must also be *fixed*. The word-lookup table is simply fixed to the embeddings obtained from our language model LM2. All other discrete feature lookup-tables (caps, POS, Brown Clusters...) are fixed to a standard *sparse* representation. Using the notation introduced in Section 3.2.1, if LT_{W^k} is the lookup-table of the k^{th} discrete feature, we have $W^k \in \mathbb{R}^{|\mathcal{D}^k| \times |\mathcal{D}^k|}$ and the representation of the discrete input w is obtained with:

$$LT_{W^k}(w) = \langle W^k \rangle_w^1 = \left(0, \cdots 0, \underset{\text{at index } w}{1}, 0, \cdots 0\right)^{\mathrm{T}} . \tag{18}$$

Training our architecture in this convex setup with the sentence-level likelihood (13) corresponds to training a CRF. In that respect, these convex experiments show the performance of our word embeddings in a classical NLP framework.

Following the Ratinov and Roth (2009) and Koo et al. (2008) setups, we generated $1,000$ Brown clusters using the implementation[20] from Liang (2005). To make the comparison fair, the clusters were first induced on the concatenation of Wikipedia and Reuters data sets, as we did in Section 4

20. Available at `http://www.eecs.berkeley.edu/~pliang/software`.

Approach	**POS** (PWA)	**CHUNK** (F1)	**NER** (F1)	**SRL** (F1)
	Non-convex Approach			
LM2 (non-linear NN)	97.29	94.32	89.59	76.06
LM2 (non-linear NN, fixed embeddings)	97.10	94.45	88.79	72.24
Brown Clusters (non-linear NN, 130K words)	96.92	94.35	87.15	72.09
Brown Clusters (non-linear NN, all words)	96.81	94.21	86.68	71.44
	Convex Approach			
LM2 (linear NN, fixed embeddings)	96.69	93.51	86.64	59.11
Brown Clusters (linear NN, 130K words)	96.56	94.20	86.46	51.54
Brown Clusters (linear NN, all words)	96.28	94.22	86.63	56.42

Table 13: Generalization performance of our neural network architecture trained with our language model (LM2) word embeddings, and with the word representations derived from the Brown Clusters. As before, all networks are fed with a capitalization feature. Additionally, POS is using a word suffix of size 2 feature, CHUNK is fed with POS, NER uses the CoNLL 2003 gazetteer, and SRL is fed with levels 1–5 of the Charniak parse tree, as well as a verb position feature. We report performance with both convex and non-convex architectures (300 hidden units for all tasks, with an additional 500 hidden units layer for SRL). We also provide results for Brown Clusters induced with a 130K word dictionary, as well as Brown Clusters induced with all words of the given tasks.

for training our largest language model LM2, using a 130K word dictionary. This dictionary covers about 99% of the words in the training set of each task. To cover the last 1%, we augmented the dictionary with the missing words (reaching about 140K words) and induced Brown Clusters using the concatenation of WSJ, Wikipedia, and Reuters.

The Brown clustering approach is hierarchical and generates a binary tree of clusters. Each word in the vocabulary is assigned to a node in the tree. Features are extracted from this tree by considering the path from the root to the node containing the word of interest. Following Ratinov & Roth, we picked as features the path prefixes of size 4, 6, 10 and 20. In the non-convex experiments, we fed these four Brown Cluster features to our architecture using four different lookup tables, replacing our word lookup table. The size of the lookup tables was chosen to be 12 by validation. In the convex case, we used the classical sparse representation (18), as for any other discrete feature.

We first report in Table 13 generalization performance of our best non-convex networks trained with our LM2 language model and with Brown Cluster features. Our embeddings perform at least as well as Brown Clusters. Results are more mitigated in a convex setup. For most tasks, going non-convex is better for both word representation types. In general, "fine-tuning" our embeddings for each task also gives an extra boost. Finally, using a better word coverage with Brown Clusters ("all words" instead of "130K words" in Table 13) did not help.

More complex features could be possibly combined instead of using a non-linear model. For instance, Turian et al. (2010) performed a comparison of Brown Clusters and embeddings trained in the same spirit as ours[21], with additional features combining labels and tokens. We believe this

21. However they did not reach our embedding performance. There are several differences in how they trained their models that might explain this. Firstly, they may have experienced difficulties because they train 50-dimensional

Task	Features
POS	Suffix of size 2
CHUNK	POS
NER	CoNLL 2003 gazetteer
PT0	POS
SRL	PT0, verb position

Table 14: Features used by SENNA implementation, for each task. In addition, all tasks use "low caps word" and "caps" features.

Task		**Benchmark**	**SENNA**
Part of Speech (POS)	(Accuracy)	97.24 %	97.29 %
Chunking (CHUNK)	(F1)	94.29 %	94.32 %
Named Entity Recognition (NER)	(F1)	89.31 %	89.59 %
Parse Tree level 0 (PT0)	(F1)	91.94 %	92.25 %
Semantic Role Labeling (SRL)	(F1)	77.92 %	75.49 %

Table 15: Performance of the engineered sweet spot (SENNA) on various tagging tasks. The PT0 task replicates the sentence segmentation of the parse tree leaves. The corresponding benchmark score measures the quality of the Charniak parse tree leaves relative to the Penn Treebank gold parse trees.

type of comparison should be taken with care, as combining a given feature with different word representations might not have the same effect on each word representation.

6.7 Engineering a Sweet Spot

We implemented a standalone version of our architecture, written in the C language. We gave the name "SENNA" (Semantic/syntactic Extraction using a Neural Network Architecture) to the resulting system. The parameters of each architecture are the ones described in Table 5. All the networks were trained separately on each task using the sentence-level likelihood (SLL). The word embeddings were initialized to LM2 embeddings, and then fine-tuned for each task. We summarize features used by our implementation in Table 14, and we report performance achieved on each task in Table 15. The runtime version[22] contains about 2500 lines of C code, runs in less than 150MB of memory, and needs less than a millisecond per word to compute all the tags. Table 16 compares the tagging speeds for our system and for the few available state-of-the-art systems: the Toutanova et al. (2003) POS tagger[23], the Shen et al. (2007) POS tagger[24] and the Koomen et al. (2005) SRL

embeddings for 269K distinct words using a comparatively small training set (RCV1, 37M words), unlikely to contain enough instances of the rare words. Secondly, they predict the correctness of the final word of each window instead of the center word (Turian et al., 2010), effectively restricting the model to unidirectional prediction. Finally, they do not fine tune their embeddings after unsupervised training.

22. Available at `http://ml.nec-labs.com/senna`.
23. Available at `http://nlp.stanford.edu/software/tagger.shtml`. We picked the 3.0 version (May 2010).
24. Available at `http://www.cis.upenn.edu/~xtag/spinal`.

POS System	RAM (MB)	Time (s)
Toutanova et al. (2003)	800	64
Shen et al. (2007)	2200	833
SENNA	32	4

SRL System	RAM (MB)	Time (s)
Koomen et al. (2005)	3400	6253
SENNA	124	51

Table 16: Runtime speed and memory consumption comparison between state-of-the-art systems and our approach (SENNA). We give the runtime in seconds for running both the POS and SRL taggers on their respective testing sets. Memory usage is reported in megabytes.

system.[25] All programs were run on a single 3GHz Intel core. The POS taggers were run with Sun Java 1.6 with a large enough memory allocation to reach their top tagging speed. The beam size of the Shen tagger was set to 3 as recommended in the paper. Regardless of implementation differences, it is clear that our neural networks run considerably faster. They also require much less memory. Our POS and SRL tagger runs in 32MB and 120MB of RAM respectively. The Shen and Toutanova taggers slow down significantly when the Java machine is given less than 2.2GB and 800MB of RAM respectively, while the Koomen tagger requires at least 3GB of RAM.

We believe that a number of reasons explain the speed advantage of our system. First, our system only uses rather simple input features and therefore avoids the nonnegligible computation time associated with complex handcrafted features. Secondly, most network computations are *dense* matrix-vector operations. In contrast, systems that rely on a great number of *sparse* features experience memory latencies when traversing the sparse data structures. Finally, our compact implementation is self-contained. Since it does not rely on the outputs of disparate NLP system, it does not suffer from communication latency issues.

7. Critical Discussion

Although we believe that this contribution represents a step towards the "NLP from scratch" objective, we are keenly aware that both our goal and our means can be criticized.

The main criticism of our goal can be summarized as follows. Over the years, the NLP community has developed a considerable expertise in engineering effective NLP features. Why should they forget this painfully acquired expertise and instead painfully acquire the skills required to train large neural networks? As mentioned in our introduction, we observe that no single NLP task really covers the goals of NLP. Therefore we believe that task-specific engineering (i.e., that does not generalize to other tasks) is not desirable. But we also recognize how much our neural networks owe to previous NLP task-specific research.

The main criticism of our means is easier to address. Why did we choose to rely on a twenty year old technology, namely multilayer neural networks? We were simply attracted by their ability to discover hidden representations using a stochastic learning algorithm that scales linearly with

25. Available at `http://l2r.cs.uiuc.edu/~cogcomp/asoftware.php?skey=SRL`.

the number of examples. Most of the neural network technology necessary for our work has been described ten years ago (e.g., Le Cun et al., 1998). However, if we had decided ten years ago to train the language model network LM2 using a vintage computer, training would only be nearing completion today. Training algorithms that scale linearly are most able to benefit from such tremendous progress in computer hardware.

8. Conclusion

We have presented a multilayer neural network architecture that can handle a number of NLP tasks with both speed and accuracy. The design of this system was determined by our desire to avoid task-specific engineering as much as possible. Instead we rely on large unlabeled data sets and let the training algorithm discover internal representations that prove useful for all the tasks of interest. Using this strong basis, we have engineered a fast and efficient "all purpose" NLP tagger that we hope will prove useful to the community.

Acknowledgments

We acknowledge the persistent support of NEC for this research effort. We thank Yoshua Bengio, Samy Bengio, Eric Cosatto, Vincent Etter, Hans-Peter Graf, Ralph Grishman, and Vladimir Vapnik for their useful feedback and comments.

Appendix A. Neural Network Gradients

We consider a neural network $f_\theta(\cdot)$, with parameters θ. We maximize the likelihood (8), or minimize ranking criterion (17), with respect to the parameters θ, using stochastic gradient. By negating the likelihood, we now assume it all corresponds to minimize a cost $C(f_\theta(\cdot))$, with respect to θ.

Following the classical "back-propagation" derivations (LeCun, 1985; Rumelhart et al., 1986) and the modular approach shown in Bottou (1991), any feed-forward neural network with L layers, like the ones shown in Figure 1 and Figure 2, can be seen as a composition of functions $f_\theta^l(\cdot)$, corresponding to each layer l:

$$f_\theta(\cdot) = f_\theta^L(f_\theta^{L-1}(\dots f_\theta^1(\cdot)\dots))$$

Partitioning the parameters of the network with respect to each layers $1 \le l \le L$, we write:

$$\theta = (\theta^1, \dots, \theta^l, \dots, \theta^L)\,.$$

We are now interested in computing the gradients of the cost with respect to each θ^l. Applying the chain rule (generalized to vectors) we obtain the classical backpropagation recursion:

$$\frac{\partial C}{\partial \theta^l} = \frac{\partial f_\theta^l}{\partial \theta^l}\frac{\partial C}{\partial f_\theta^l} \tag{19}$$

$$\frac{\partial C}{\partial f_\theta^{l-1}} = \frac{\partial f_\theta^l}{\partial f_\theta^{l-1}}\frac{\partial C}{\partial f_\theta^l}\,. \tag{20}$$

In other words, we first initialize the recursion by computing the gradient of the cost with respect to the last layer output $\partial C/\partial f_\theta^L$. Then each layer l computes the gradient respect to its own parameters with (19), given the gradient coming from its output $\partial C / \partial f_\theta^l$. To perform the backpropagation, it also computes the gradient with respect to its own inputs, as shown in (20). We now derive the gradients for each layer we used in this paper.

A.1 Lookup Table Layer

Given a matrix of parameters $\theta^1 = W^1$ and word (or discrete feature) indices $[w]_1^T$, the layer outputs the matrix:

$$f_\theta^l([w]_l^T) = \begin{pmatrix} \langle W \rangle_{[w]_1}^1 & \langle W \rangle_{[w]_2}^1 & \dots & \langle W \rangle_{[w]_T}^1 \end{pmatrix}.$$

The gradients of the weights $\langle W \rangle_i^1$ are given by:

$$\frac{\partial C}{\partial \langle W \rangle_i^1} = \sum_{\{1 \le t \le T \,/\, [w]_t = i\}} \langle \frac{\partial C}{\partial f_\theta^l} \rangle_i^1$$

This sum equals zero if the index i in the lookup table does not corresponds to a word in the sequence. In this case, the i^{th} column of W does not need to be updated. As a Lookup Table Layer is always the first layer, we do not need to compute its gradients with respect to the inputs.

A.2 Linear Layer

Given parameters $\theta^l = (W^l, b^l)$, and an input *vector* f_θ^{l-1} the output is given by:

$$f_\theta^l = W^l f_\theta^{l-1} + b^l. \tag{21}$$

The gradients with respect to the parameters are then obtained with:

$$\frac{\partial C}{\partial W^l} = \left[\frac{\partial C}{\partial f_\theta^l}\right] \left[f_\theta^{l-1}\right]^{\mathrm{T}} \text{ and } \frac{\partial C}{\partial b^l} = \frac{\partial C}{\partial f_\theta^l}, \tag{22}$$

and the gradients with respect to the inputs are computed with:

$$\frac{\partial C}{\partial f_\theta^{l-1}} = \left[W^l\right]^{\mathrm{T}} \frac{\partial C}{\partial f_\theta^l}. \tag{23}$$

A.3 Convolution Layer

Given a input *matrix* f_θ^{l-1}, a Convolution Layer $f_\theta^l(\cdot)$ applies a Linear Layer operation (21) successively on each window $\langle f_\theta^{l-1} \rangle_t^{d_{win}}$ $(1 \le t \le T)$ of size d_{win}. Using (22), the gradients of the parameters are thus given by summing over all windows:

$$\frac{\partial C}{\partial W^l} = \sum_{t=1}^{T} \left[\langle \frac{\partial C}{\partial f_\theta^l} \rangle_t^1\right] \left[\langle f_\theta^{l-1} \rangle_t^{d_{win}}\right]^{\mathrm{T}} \text{ and } \frac{\partial C}{\partial b^l} = \sum_{t=1}^{T} \langle \frac{\partial C}{\partial f_\theta^l} \rangle_t^1.$$

After initializing the input gradients $\partial C / \partial f_\theta^{l-1}$ to zero, we iterate (23) over all windows for $1 \le t \le T$, leading the *accumulation*[26]:

$$\langle \frac{\partial C}{\partial f_\theta^{l-1}} \rangle_t^{d_{win}} \mathrel{+}= \left[W^l\right]^{\mathrm{T}} \langle \frac{\partial C}{\partial f_\theta^l} \rangle_t^1.$$

26. We denote "+=" any accumulation operation.

A.4 Max Layer

Given a *matrix* f_θ^{l-1}, the Max Layer computes

$$\left[f_\theta^l\right]_i = \max_t \left[\langle f_\theta^{l-1}\rangle_t^1\right]_i \text{ and } a_i = \operatorname*{argmax}_t \left[\langle f_\theta^{l-1}\rangle_t^1\right]_i \; \forall i,$$

where a_i stores the index of the largest value. We only need to compute the gradient with respect to the inputs, as this layer has no parameters. The gradient is given by

$$\left[\langle \frac{\partial C}{\partial f_\theta^{l-1}} \rangle_t^1\right]_i = \begin{cases} \left[\langle \frac{\partial C}{\partial f_\theta^l}\rangle_t^1\right]_i & \text{if } t = a_i \\ 0 & \text{otherwise} \end{cases} .$$

A.5 HardTanh Layer

Given a *vector* f_θ^{l-1}, and the definition of the HardTanh (5) we get

$$\left[\frac{\partial C}{\partial f_\theta^{l-1}}\right]_i = \begin{cases} 0 & \text{if } \left[f_\theta^{l-1}\right]_i < -1 \\ \left[\frac{\partial C}{\partial f_\theta^l}\right]_i & \text{if } -1 <= \left[f_\theta^{l-1}\right]_i <= 1 \\ 0 & \text{if } \left[f_\theta^{l-1}\right]_i > 1 \end{cases} ,$$

if we ignore non-differentiability points.

A.6 Word-Level Log-Likelihood

The network outputs a score $[f_\theta]_i$ for each tag indexed by i. Following (11), if y is the true tag for a given example, the stochastic score to minimize can be written as

$$C(f_\theta) = \operatorname*{logadd}_j \, [f_\theta]_j - [f_\theta]_y$$

Considering the definition of the logadd (10), the gradient with respect to f_θ is given by

$$\frac{\partial C}{\partial\, [f_\theta]_i} = \frac{e^{[f_\theta]_i}}{\sum_k e^{[f_\theta]_k}} - 1_{i=y} \quad \forall i.$$

A.7 Sentence-Level Log-Likelihood

The network outputs a matrix where each element $\left[f_\theta\right]_{i,t}$ gives a score for tag i at word t. Given a tag sequence $[y]_1^T$ and a input sequence $[x]_1^T$, we maximize the likelihood (13), which corresponds to minimizing the score

$$C(f_\theta, A) = \underbrace{\operatorname*{logadd}_{\forall [j]_1^T} s([x]_1^T,\, [j]_1^T,\, \tilde{\theta})}_{C_{logadd}} - s([x]_1^T,\, [y]_1^T,\, \tilde{\theta}),$$

with

$$s([x]_1^T,\, [y]_1^T,\, \tilde{\theta}) = \sum_{t=1}^T \left([A]_{[y]_{t-1},\, [y]_t} + [f_\theta]_{[y]_t,\, t} \right).$$

We first initialize all gradients to zero

$$\frac{\partial C}{\partial \left[f_\theta\right]_{i,t}} = 0 \;\forall i,t \;\text{ and }\; \frac{\partial C}{\partial \left[A\right]_{i,j}} = 0 \quad \forall i,j\,.$$

We then *accumulate* gradients over the second part of the cost $-s([x]_1^T, [y]_1^T, \tilde{\theta})$, which gives:

$$\left.\begin{aligned} \frac{\partial C}{\partial \left[f_\theta\right]_{[y]_t,t}} &\mathrel{+}= 1 \\ \frac{\partial C}{\partial \left[A\right]_{[y]_{t-1},[y]_t}} &\mathrel{+}= 1 \end{aligned}\right. \qquad \forall t\,.$$

We now need to accumulate the gradients over the first part of the cost, that is C_{logadd}. We differentiate C_{logadd} by applying the chain rule through the recursion (14). First we initialize our recursion with

$$\frac{\partial C_{logadd}}{\partial \delta_T(i)} = \frac{e^{\delta_T(i)}}{\sum_k e^{\delta_T(k)}} \quad \forall i\,.$$

We then compute iteratively:

$$\frac{\partial C_{logadd}}{\partial \delta_{t-1}(i)} = \sum_j \frac{\partial C_{logadd}}{\partial \delta_t(j)} \frac{e^{\delta_{t-1}(i)+[A]_{i,j}}}{\sum_k e^{\delta_{t-1}(k)+[A]_{k,j}}}\,,$$

where at each step t of the recursion we accumulate of the gradients with respect to the inputs f_θ, and the transition scores $[A]_{i,j}$:

$$\begin{aligned} \frac{\partial C}{\partial \left[f_\theta\right]_{i,t}} \mathrel{+}= \frac{\partial C_{logadd}}{\partial \delta_t(i)} \frac{\partial \delta_t(i)}{\partial \left[f_\theta\right]_{i,t}} &= \frac{\partial C_{logadd}}{\partial \delta_t(i)} \\ \frac{\partial C}{\partial \left[A\right]_{i,j}} \mathrel{+}= \frac{\partial C_{logadd}}{\partial \delta_t(j)} \frac{\partial \delta_t(j)}{\partial \left[A\right]_{i,j}} &= \frac{\partial C_{logadd}}{\partial \delta_t(j)} \frac{e^{\delta_{t-1}(i)+[A]_{i,j}}}{\sum_k e^{\delta_{t-1}(k)+[A]_{k,j}}}\,. \end{aligned}$$

A.8 Ranking Criterion

We use the ranking criterion (17) for training our language model. In this case, given a "positive" example x and a "negative" example $x^{(w)}$, we want to minimize:

$$C(f_\theta(x), f_\theta(x^w)) = \max\left\{\, 0\,,\, 1 - f_\theta(x) + f_\theta(x^{(w)}) \,\right\}.$$

Ignoring the non-differentiability of $max(0,\cdot)$ in zero, the gradient is simply given by:

$$\begin{pmatrix} \frac{\partial C}{\partial f_\theta(x)} \\ \frac{\partial C}{\partial f_\theta(x^w)} \end{pmatrix} = \begin{cases} \begin{pmatrix} -1 \\ 1 \end{pmatrix} & \text{if } 1 - f_\theta(x) + f_\theta(x^{(w)}) > 0 \\ \begin{pmatrix} 0 \\ 0 \end{pmatrix} & \text{otherwise} \end{cases}\,.$$

References

R. K. Ando and T. Zhang. A framework for learning predictive structures from multiple tasks and unlabeled data. *Journal of Machine Learning Research (JMLR)*, 6:1817–1953, 2005.

R. M. Bell, Y. Koren, and C. Volinsky. The BellKor solution to the Netflix Prize. Technical report, AT&T Labs, 2007. `http://www.research.att.com/~volinsky/netflix`.

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

Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. In *Advances in Neural Information Processing Systems (NIPS 19)*, 2007.

Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *International Conference on Machine Learning (ICML)*, 2009.

L. Bottou. Stochastic gradient learning in neural networks. In *Proceedings of Neuro-Nîmes*. EC2, 1991.

L. Bottou. Online algorithms and stochastic approximations. In David Saad, editor, *Online Learning and Neural Networks*. Cambridge University Press, Cambridge, UK, 1998.

L. Bottou and P. Gallinari. A framework for the cooperation of learning algorithms. In *Advances in Neural Information Processing Systems (NIPS 3)*. 1991.

L. Bottou, Y. LeCun, and Yoshua Bengio. Global training of document processing systems using graph transformer networks. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 489–493, 1997.

J. S. Bridle. 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, 1990.

P. F. Brown, P. V. deSouza, R. L. Mercer, V. J. D. Pietra, and J C. Lai. Class-based n-gram models of natural language. *Computational Linguistics*, 18(4):467–479, 1992a.

P. F. Brown, V. J. Della Pietra, R. L. Mercer, S. A. Della Pietra, and J. C. Lai. An estimate of an upper bound for the entropy of english. *Computational Linguistics*, 18(1):31–41, 1992b.

C. J. C. Burges, R. Ragno, and Quoc Viet Le. Learning to rank with nonsmooth cost functions. In *Advances in Neural Information Processing Systems (NIPS 19)*, pages 193–200. 2007.

R. Caruana. Multitask Learning. *Machine Learning*, 28(1):41–75, 1997.

O. Chapelle, B. Schlkopf, and A. Zien. *Semi-Supervised Learning*. Adaptive computation and machine learning. MIT Press, Cambridge, Mass., USA, September 2006.

E. Charniak. A maximum-entropy-inspired parser. In *Conference of the North American Chapter of the Association for Computational Linguistics & Human Language Technologies (NAACL-HLT)*, pages 132–139, 2000.

H. L. Chieu. Named entity recognition with a maximum entropy approach. In *Conference on Natural Language Learning (CoNLL)*, pages 160–163, 2003.

N. Chomsky. Three models for the description of language. *IRE Transactions on Information Theory*, 2(3):113–124, September 1956.

S. Clémençon and N. Vayatis. Ranking the best instances. *Journal of Machine Learning Research (JMLR)*, 8:2671–2699, 2007.

W. W. Cohen, R. E. Schapire, and Y. Singer. Learning to order things. *Journal of Artificial Intelligence Research (JAIR)*, 10:243–270, 1998.

T. Cohn and P. Blunsom. Semantic role labelling with tree conditional random fields. In *Conference on Computational Natural Language (CoNLL)*, 2005.

M. Collins. *Head-Driven Statistical Models for Natural Language Parsing*. PhD thesis, University of Pennsylvania, 1999.

R. Collobert. *Large Scale Machine Learning*. PhD thesis, Université Paris VI, 2004.

R. Collobert. Deep learning for efficient discriminative parsing. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2011.

T. Cover and R. King. A convergent gambling estimate of the entropy of english. *IEEE Transactions on Information Theory*, 24(4):413–421, July 1978.

R. Florian, A. Ittycheriah, H. Jing, and T. Zhang. Named entity recognition through classifier combination. In *Conference of the North American Chapter of the Association for Computational Linguistics & Human Language Technologies (NAACL-HLT)*, pages 168–171, 2003.

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

D. Gildea and M. Palmer. The necessity of parsing for predicate argument recognition. *Meeting of the Association for Computational Linguistics (ACL)*, pages 239–246, 2002.

J. Giménez and L. Màrquez. SVMTool: A general POS tagger generator based on support vector machines. In *Conference on Language Resources and Evaluation (LREC)*, 2004.

A. Haghighi, K. Toutanova, and C. D. Manning. A joint model for semantic role labeling. In *Conference on Computational Natural Language Learning (CoNLL)*, June 2005.

Z. S. Harris. *Mathematical Structures of Language*. John Wiley & Sons Inc., 1968.

D. Heckerman, D. M. Chickering, C. Meek, R. Rounthwaite, and C. Kadie. Dependency networks for inference, collaborative filtering, and data visualization. *Journal of Machine Learning Research (JMLR)*, 1:49–75, 2001.

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

K. Hollingshead, S. Fisher, and B. Roark. Comparing and combining finite-state and context-free parsers. In *Conference on Human Language Technology and Empirical Methods in Natural Language Processing (HLT-EMNLP)*, pages 787–794, 2005.

F. Huang and A. Yates. Distributional representations for handling sparsity in supervised sequence-labeling. In *Meeting of the Association for Computational Linguistics (ACL)*, pages 495–503, 2009.

F. Jelinek. Continuous speech recognition by statistical methods. *Proceedings of the IEEE*, 64(4): 532–556, 1976.

T. Joachims. Transductive inference for text classification using support vector machines. In *International Conference on Machine learning (ICML)*, 1999.

D. Klein and C. D. Manning. Natural language grammar induction using a constituent-context model. In *Advances in Neural Information Processing Systems (NIPS 14)*, pages 35–42. 2002.

T. Koo, X. Carreras, and M. Collins. Simple semi-supervised dependency parsing. In *Meeting of the Association for Computational Linguistics (ACL)*, pages 595–603, 2008.

P. Koomen, V. Punyakanok, D. Roth, and W. Yih. Generalized inference with multiple semantic role labeling systems (shared task paper). In *Conference on Computational Natural Language Learning (CoNLL)*, pages 181–184, 2005.

T. Kudo and Y. Matsumoto. Chunking with support vector machines. In *Conference of the North American Chapter of the Association for Computational Linguistics & Human Language Technologies (NAACL-HLT)*, pages 1–8, 2001.

T. Kudoh and Y. Matsumoto. Use of support vector learning for chunk identification. In *Conference on Natural Language Learning (CoNLL) and Second Learning Language in Logic Workshop (LLL)*, pages 142–144, 2000.

J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *International Conference on Machine Learning (ICML)*, 2001.

Y. Le Cun, L. Bottou, Y. Bengio, and P. Haffner. Gradient based learning applied to document recognition. *Proceedings of IEEE*, 86(11):2278–2324, 1998.

Y. LeCun. A learning scheme for asymmetric threshold networks. In *Proceedings of Cognitiva*, pages 599–604, Paris, France, 1985.

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

D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. Rcv1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research (JMLR)*, 5:361–397, 2004.

P. Liang. Semi-supervised learning for natural language. Master's thesis, Massachusetts Institute of Technology, 2005.

P. Liang, H. Daumé, III, and D. Klein. Structure compilation: trading structure for features. In *International Conference on Machine learning (ICML)*, pages 592–599, 2008.

D. Lin and X. Wu. Phrase clustering for discriminative learning. In *Meeting of the Association for Computational Linguistics (ACL)*, pages 1030–1038, 2009.

N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. In *Machine Learning*, pages 285–318, 1988.

A. McCallum and Wei Li. Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons. In *Conference of the North American Chapter of the Association for Computational Linguistics & Human Language Technologies (NAACL-HLT)*, pages 188–191, 2003.

D. McClosky, E. Charniak, and M. Johnson. Effective self-training for parsing. *Conference of the North American Chapter of the Association for Computational Linguistics & Human Language Technologies (NAACL-HLT)*, 2006.

R. McDonald, K. Crammer, and F. Pereira. Flexible text segmentation with structured multilabel classification. In *Conference on Human Language Technology and Empirical Methods in Natural Language Processing (HLT-EMNLP)*, pages 987–994, 2005.

S. Miller, H. Fox, L. Ramshaw, and R. Weischedel. A novel use of statistical parsing to extract information from text. *Applied Natural Language Processing Conference (ANLP)*, 2000.

S. Miller, J. Guinness, and A. Zamanian. Name tagging with word clusters and discriminative training. In *Conference of the North American Chapter of the Association for Computational Linguistics & Human Language Technologies (NAACL-HLT)*, pages 337–342, 2004.

A Mnih and G. E. Hinton. Three new graphical models for statistical language modelling. In *International Conference on Machine Learning (ICML)*, pages 641–648, 2007.

G. Musillo and P. Merlo. Robust Parsing of the Proposition Bank. *ROMAND 2006: Robust Methods in Analysis of Natural language Data*, 2006.

R. M. Neal. *Bayesian Learning for Neural Networks*. Number 118 in Lecture Notes in Statistics. Springer-Verlag, New York, 1996.

D. Okanohara and J. Tsujii. A discriminative language model with pseudo-negative samples. *Meeting of the Association for Computational Linguistics (ACL)*, pages 73–80, 2007.

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

J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufman, San Mateo, 1988.

D. C. Plaut and G. E. Hinton. Learning sets of filters using back-propagation. *Computer Speech and Language*, 2:35–61, 1987.

M. F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

S. Pradhan, W. Ward, K. Hacioglu, J. Martin, and D. Jurafsky. Shallow semantic parsing using support vector machines. *Conference of the North American Chapter of the Association for Computational Linguistics & Human Language Technologies (NAACL-HLT)*, 2004.

S. Pradhan, K. Hacioglu, W. Ward, J. H. Martin, and D. Jurafsky. Semantic role chunking combining complementary syntactic views. In *Conference on Computational Natural Language Learning (CoNLL)*, pages 217–220, 2005.

V. Punyakanok, D. Roth, and W. Yih. The necessity of syntactic parsing for semantic role labeling. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1117–1123, 2005.

L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

L. Ratinov and D. Roth. Design challenges and misconceptions in named entity recognition. In *Conference on Computational Natural Language Learning (CoNLL)*, pages 147–155. Association for Computational Linguistics, 2009.

A. Ratnaparkhi. A maximum entropy model for part-of-speech tagging. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 133–142, 1996.

B. Rosenfeld and R. Feldman. Using Corpus Statistics on Entities to Improve Semi-supervised Relation Extraction from the Web. *Meeting of the Association for Computational Linguistics (ACL)*, pages 600–607, 2007.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by back-propagating errors. In D.E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1, pages 318–362. MIT Press, 1986.

H. Schütze. Distributional part-of-speech tagging. In *Meeting of the Association for Computational Linguistics (ACL)*, pages 141–148, 1995.

H. Schwenk and J. L. Gauvain. Connectionist language modeling for large vocabulary continuous speech recognition. In *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 765–768, 2002.

F. Sha and F. Pereira. Shallow parsing with conditional random fields. In *Conference of the North American Chapter of the Association for Computational Linguistics & Human Language Technologies (NAACL-HLT)*, pages 134–141, 2003.

C. E. Shannon. Prediction and entropy of printed english. *Bell Systems Technical Journal*, 30: 50–64, 1951.

H. Shen and A. Sarkar. Voting between multiple data representations for text chunking. *Advances in Artificial Intelligence*, pages 389–400, 2005.

L. Shen, G. Satta, and A. K. Joshi. Guided learning for bidirectional sequence classification. In *Meeting of the Association for Computational Linguistics (ACL)*, 2007.

N. A. Smith and J. Eisner. Contrastive estimation: Training log-linear models on unlabeled data. In *Meeting of the Association for Computational Linguistics (ACL)*, pages 354–362, 2005.

S. C. Suddarth and A. D. C. Holden. Symbolic-neural systems and the use of hints for developing complex systems. *International Journal of Man-Machine Studies*, 35(3):291–311, 1991.

X. Sun, L.-P. Morency, D. Okanohara, and J. Tsujii. Modeling latent-dynamic in shallow parsing: a latent conditional model with improved inference. In *International Conference on Computational Linguistics (COLING)*, pages 841–848, 2008.

C. Sutton and A. McCallum. Joint parsing and semantic role labeling. In *Conference on Computational Natural Language (CoNLL)*, pages 225–228, 2005a.

C. Sutton and A. McCallum. Composition of conditional random fields for transfer learning. *Conference on Human Language Technology and Empirical Methods in Natural Language Processing (HLT-EMNLP)*, pages 748–754, 2005b.

C. Sutton, A. McCallum, and K. Rohanimanesh. Dynamic Conditional Random Fields: Factorized Probabilistic Models for Labeling and Segmenting Sequence Data. *Journal of Machine Learning Research (JMLR)*, 8:693–723, 2007.

J. Suzuki and H. Isozaki. Semi-supervised sequential labeling and segmentation using giga-word scale unlabeled data. In *Conference of the North American Chapter of the Association for Computational Linguistics & Human Language Technologies (NAACL-HLT)*, pages 665–673, 2008.

W. J. Teahan and J. G. Cleary. The entropy of english using ppm-based models. In *Data Compression Conference (DCC)*, pages 53–62. IEEE Computer Society Press, 1996.

K. Toutanova, D. Klein, C. D. Manning, and Y. Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Conference of the North American Chapter of the Association for Computational Linguistics & Human Language Technologies (NAACL-HLT)*, 2003.

J. Turian, L. Ratinov, and Y. Bengio. Word representations: A simple and general method for semi-supervised learning. In *Meeting of the Association for Computational Linguistics (ACL)*, pages 384–392, 2010.

N. Ueffing, G. Haffari, and A. Sarkar. Transductive learning for statistical machine translation. In *Meeting of the Association for Computational Linguistics (ACL)*, pages 25–32, 2007.

A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K.J. Lang. Phoneme recognition using time-delay neural networks. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37(3): 328–339, 1989.

J. Weston, F. Ratle, and R. Collobert. Deep learning via semi-supervised embedding. In *International Conference on Machine learning (ICML)*, pages 1168–1175, 2008.