A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning

Ronan Collobert COLLOBER@NEC-LABS.COM
Jason Weston JASONW@NEC-LABS.COM
NEC Labs America, 4 Independence Way, Princeton, NJ 08540 USA

Abstract

We describe a single convolutional neural network architecture that, given a sentence, outputs a host of language processing predictions: part-of-speech tags, chunks, named entity tags, semantic roles, semantically similar words and the likelihood that the sentence makes sense (grammatically and semantically) using a language model. The entire network is trained *jointly* on all these tasks using weight-sharing, an instance of *multitask learning*. All the tasks use labeled data except the language model which is learnt from unlabeled text and represents a novel form of *semi-supervised learning* for the shared tasks. We show how both *multitask learning* and *semi-supervised learning* improve the generalization of the shared tasks, resulting in state-of-the-art performance.

1. Introduction

The field of Natural Language Processing (NLP) aims to convert human language into a formal representation that is easy for computers to manipulate. Current end applications include information extraction, machine translation, summarization, search and human-computer interfaces.

While complete semantic understanding is still a far-distant goal, researchers have taken a divide and conquer approach and identified several sub-tasks useful for application development and analysis. These range from the syntactic, such as part-of-speech tagging, chunking and parsing, to the semantic, such as word-sense disambiguation, semantic-role labeling, named entity extraction and anaphora resolution.

Appearing in *Proceedings of the 25th International Conference on Machine Learning*, Helsinki, Finland, 2008.

Currently, most research analyzes those tasks *separately*. Many systems possess few characteristics that would help develop a unified architecture which would presumably be necessary for deeper semantic tasks. In particular, many systems possess three failings in this regard: (i) they are *shallow* in the sense that the classifier is often linear, (ii) for good performance with a linear classifier they must incorporate many hand-engineered features specific for the task; and (iii) they cascade features learnt separately from other tasks, thus propagating errors.

In this work we attempt to define a unified architecture for Natural Language Processing that *learns features* that are relevant to the tasks at hand given very limited prior knowledge. This is achieved by training a *deep neural network*, building upon work by (Bengio & Ducharme, 2001) and (Collobert & Weston, 2007). We define a rather general convolutional network architecture and describe its application to many well known NLP tasks including part-of-speech tagging, chunking, named-entity recognition, learning a language model and the task of semantic role-labeling.

All of these tasks are integrated into a single system which is trained *jointly*. All the tasks except the language model are supervised tasks with labeled training data. The language model is trained in an unsupervised fashion on the entire Wikipedia website. Training this task jointly with the other tasks comprises a novel form of *semi-supervised learning*.

We focus on, in our opinion, the most difficult of these tasks: the semantic role-labeling problem. We show that both (i) multitask learning and (ii) semi-supervised learning significantly improve performance on this task *in the absence of hand-engineered features*.

We also show how the combined tasks, and in particular the unsupervised task, learn powerful features with clear semantic information given no human supervision other than the (labeled) data from the tasks (see Table 1).

The article is structured as follows. In Section 2 we describe each of the NLP tasks we consider, and in Section 3 we define the general architecture that we use to solve all the tasks. Section 4 describes how this architecture is employed for *multitask* learning on all the *labeled* tasks we consider, and Section 5 describes the *unlabeled* task of building a language model in some detail. Section 6 gives experimental results of our system, and Section 7 concludes with a discussion of our results and possible directions for future research.

2. NLP Tasks

We consider six standard NLP tasks in this paper.

Part-Of-Speech Tagging (POS) aims at labeling each word with a unique tag that indicates its syntactic role, e.g. plural noun, adverb, ...

Chunking, also called shallow parsing, aims at labeling segments of a sentence with syntactic constituents such as noun or verb phrase (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).

Named Entity Recognition (NER) labels atomic elements in the sentence into categories such as "PERSON", "COMPANY", or "LOCATION".

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 predicate in the sentence, e.g. 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 13 modifier tags such as ARGM-LOC (locational) and ARGM-TMP (temporal) that operate in a similar way for all verbs.

Language Models A language model traditionally estimates the probability of the next word being w in a sequence. We consider a different setting: predict whether the given sequence exists in nature, or not, following the methodology of (Okanohara & Tsujii, 2007). This is achieved by labeling real texts as positive examples, and generating "fake" negative text.

Semantically Related Words ("Synonyms") This is the task of predicting whether two words are semantically related (synonyms, holonyms, hypernyms...) which is measured using the WordNet database (`http://wordnet.princeton.edu`) as ground truth.

Our main interest is SRL, as it is, in our opinion, the most complex of these tasks. We use all these tasks to: (i) show the generality of our proposed architecture; and (ii) improve SRL through multitask learning.

3. General Deep Architecture for NLP

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 classical *shallow* classification algorithm, e.g. a Support Vector Machine (SVM), often with a linear kernel. The choice of features is a completely empirical process, mainly based on 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 complex features (e.g., extracted from a parse tree) which makes such systems slow and intractable for large-scale applications.

Instead we advocate a deep neural network (NN) architecture, trained in an end-to-end fashion. The input sentence is processed by several layers of feature extraction. The features in deep layers of the network are *automatically trained* by backpropagation to be relevant to the task. We describe in this section a general deep architecture suitable for all our NLP tasks, and easily generalizable to other NLP tasks.

Our architecture is summarized in Figure 1. The first layer extracts features for each word. The second layer extracts features from the 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 classical NN layers.

3.1. Transforming Indices into Vectors

As our architecture deals with raw words and not engineered features, the first layer has to map words into real-valued vectors for processing by subsequent layers of the NN. For simplicity (and efficiency) we consider words as indices in a finite dictionary of words $\mathcal{D} \subset \mathbb{N}$.

Lookup-Table Layer Each word $i \in \mathcal{D}$ is *embedded* into a d-dimensional space using a *lookup table* $LT_W(\cdot)$:

$$LT_W(i) = W_i \,,$$

where $W \in \mathbb{R}^{d \times |\mathcal{D}|}$ is a matrix of parameters to be learnt, $W_i \in \mathbb{R}^d$ is the i^{th} column of W and d is the word vector size (wsz) to be chosen by the user. In the first layer of our architecture an input sentence $\{s_1, s_2, \ldots s_n\}$ of n words in $\mathcal{D}$ is thus transformed into a series of vectors $\{W_{s_1}, W_{s_2}, \ldots W_{s_n}\}$ by apply-

ing the lookup-table to each of its words.

It is important to note that the parameters W of the layer are automatically *trained* during the learning process using backpropagation.

Variations on Word Representations In practice, one may want to introduce some basic pre-processing, such as word-stemming or dealing with upper and lower case. In our experiments, we limited ourselves to converting all words to lower case, and represent the capitalization as a separate feature (yes or no).

When a word is decomposed into K elements (features), it can be represented as a tuple $\boldsymbol{i} = \{i^1, i^2, \dots i^K\} \in \mathcal{D}^1 \times \cdots \times \mathcal{D}^K$, where $\mathcal{D}^k$ is the dictionary for the k^{th}-element. We associate to each element a lookup-table $LT_{W^k}(\cdot)$, with parameters $W^k \in \mathbb{R}^{d^k \times |\mathcal{D}^k|}$ where $d_k \in \mathbb{N}$ is a user-specified vector size. A word $\boldsymbol{i}$ is then embedded in a $d = \sum_k d^k$ dimensional space by concatenating all lookup-table outputs:

$$LT_{W^1,\dots,W^K}(\boldsymbol{i})^{\mathrm{T}} = (LT_{W^1}(i^1)^{\mathrm{T}}, \dots, LT_{W^K}(i^K)^{\mathrm{T}})$$

Classifying with Respect to a Predicate In a complex task like SRL, the class label of each word in a sentence depends on a given *predicate*. It is thus necessary to encode in the NN architecture which predicate we are considering in the sentence.

We propose to add a feature for each word that encodes its relative distance to the chosen predicate. For the i^{th} word in the sentence, if the predicate is at position pos_p we use an additional lookup table $LT^{dist_p}(i - pos_p)$.

3.2. Variable Sentence Length

The lookup table layer maps the original sentence into a sequence $\boldsymbol{x}(\cdot)$ of n identically sized vectors:

$$(\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_n), \quad \forall t \ \boldsymbol{x}_t \in \mathbb{R}^d \,. \tag{1}$$

Obviously the size n of the sequence varies depending on the sentence. Unfortunately normal NNs are not able to handle sequences of variable length.

The simplest solution is to use a *window approach*: consider a window of fixed size ksz around each word we want to label. While this approach works with great success on simple tasks like POS, it fails on more complex tasks like SRL. In the latter case it is common for the role of a word to depend on words far away in the sentence, and hence outside of the considered window.

When modeling long-distance dependencies is important, Time-Delay Neural Networks (TDNNs) (Waibel et al., 1989) are a better choice. Here, *time* refers

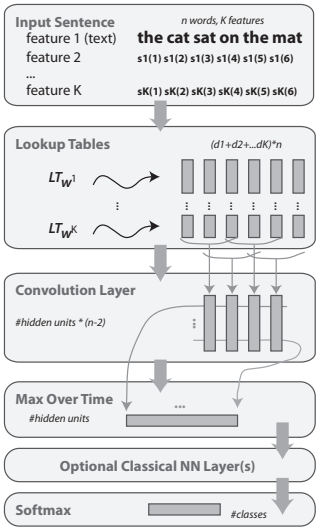

Figure 1. A general deep NN architecture for NLP. Given an input sentence, the NN outputs class probabilities for *one* chosen word. A classical window approach is a special case where the input has a fixed size ksz, and the TDNN kernel size is ksz; in that case the TDNN layer outputs only one vector and the Max layer performs an identity.

to the idea that a sequence has a notion of order. A TDNN "reads" the sequence in an online fashion: at *time* $t \geq 1$, one sees $\boldsymbol{x}_t$, the t^{th} word in the sentence.

A classical TDNN layer performs a *convolution* on a given sequence $\boldsymbol{x}(\cdot)$, outputting another sequence $\boldsymbol{o}(\cdot)$ whose value at time t is:

$$\boldsymbol{o}(t) = \sum_{j=1-t}^{n-t} \boldsymbol{L}_j \cdot \boldsymbol{x}_{t+j} \,, \tag{2}$$

where $\boldsymbol{L}_j \in \mathbb{R}^{n_{hu} \times d}$ $(-n \leq j \leq n)$ are the parameters of the layer (with n_{hu} hidden units) trained by backpropagation. One usually constrains this convolution by defining a *kernel width*, ksz, which enforces

$$\forall \ |j| > (ksz - 1)/2, \quad \boldsymbol{L}_j = \boldsymbol{0} \,. \tag{3}$$

A classical window approach only considers words in a window of size ksz around the word to be labeled. Instead, if we use (2) and (3), a TDNN considers at the same time *all* windows of ksz words in the sentence.

TDNN layers can also be *stacked* so that one can extract local features in lower layers, and more global features in subsequent ones. This is an approach typically used in convolutional networks for vision tasks, such as the LeNet architecture (LeCun et al., 1998).

We then add to our architecture a layer which captures the *most relevant features over the sentence* by feeding

the TDNN layer(s) into a "Max" Layer, which takes the maximum over time (over the sentence) in (2) for each of the n_{hu} output features.

As the layer's output is of fixed dimension (independent of sentence size) subsequent layers can be classical NN layers. Provided we have a way to indicate to our architecture the word *to be labeled*, it is then able to use features extracted from *all* windows of ksz words in the sentence to compute the label of *one* word of interest.

We indicate the word to be labeled to the NN with an additional lookup-table, as suggested in Section 3.1. Considering the word at position pos_w we encode the relative distance between the i^{th} word in the sentence and this word using a lookup-table $LT^{dist_w}(i - pos_w)$.

3.3. Deep Architecture

A TDNN (or window) layer performs a linear operation over the input words. While linear approaches work fairly well for POS or NER, more complex tasks like SRL require nonlinear models. One can add to the NN one or more classical NN layers. The output of the l^{th} layer containing n_{hu_l} hidden units is computed with $\boldsymbol{o^l} = \tanh(\boldsymbol{L^l} \cdot \boldsymbol{o^{l-1}})$, where the matrix of parameters $\boldsymbol{L^l} \in \mathbb{R}^{n_{hu_l} \times n_{hu_{l-1}}}$ is trained by backpropagation.

The size of the last (parametric) layer's output $\boldsymbol{o^{last}}$ is the number of classes considered in the NLP task. This layer is followed by a *softmax* layer (Bridle, 1990) which makes sure the outputs are positive and sum to 1, allowing us to interpret the outputs of the NN as probabilities for each class. The i^{th} output is given by $e^{o_i^{last}} / \sum_j e^{o_j^{last}}$. The whole network is trained with the cross-entropy criterion (Bridle, 1990).

3.4. Related Architectures

In (Collobert & Weston, 2007) we described a NN suited for SRL. This work also used a lookup-table to generate word features (see also (Bengio & Ducharme, 2001)). The issue of labeling with respect to a predicate was handled with a special hidden layer: its output, given input sequence (1), predicate position pos_p, and the word of interest pos_w was defined as:

$$\boldsymbol{o}(t) = \mathcal{C}(t - pos_w,\, t - pos_p) \cdot \boldsymbol{x}_t.$$

The function $\mathcal{C}(\cdot)$ is shared through time t: one could say that this is a variant of a TDNN layer with a kernel width $ksz = 1$ but where the parameters are *conditioned* with other variables (distances with respect to the verb and word of interest).

The fact that $\mathcal{C}(\cdot)$ does not combine several words in the same neighborhood as in our TDNN approach limits the dependencies between words it can model. Also $\mathcal{C}(\cdot)$ is itself a NN inside a NN. Not only does one have to carefully design this additional architecture, but it also makes the approach more complicated to train and implement. Integrating all the desired features in $\boldsymbol{x}()$ (including the predicate position) via lookup-tables makes our approach simpler, more general and easier to tune.

4. Multitasking with Deep NN

Multitask learning (MTL) is the procedure of learning several tasks at the same time with the aim of mutual benefit. This an old idea in machine learning; a good overview, especially focusing on NNs, can be found in (Caruana, 1997).

4.1. Deep Joint Training

If one considers *related* tasks, it makes sense that features useful for one task might be useful for other ones. In NLP for example, POS predictions are often used as features for SRL and NER. Improving generalization on the POS task might therefore improve both SRL and NER.

A NN automatically learns features for the desired tasks in the deep layers of its architecture. In the case of our general architecture for NLP presented in Section 3, the deepest layer (consisting of lookup-tables) *implicitly learns* relevant features for each word in the dictionary. It is thus reasonable to expect that when training NNs on *related* tasks, sharing deep layers in these NNs would improve features produced by these deep layers, and thus improve generalization performance. The last layers of the network can then be task specific.

In this paper we show this procedure performs very well for NLP tasks when sharing the lookup-tables of each considered task, as depicted in Figure 2. Training is achieved in a stochastic manner by looping over the tasks:

1. Select the next task.
2. Select a random training example for this task.
3. Update the NN for this task by taking a gradient step with respect to this example.
4. Go to 1.

It is worth noticing that labeled data for training each task can come from completely different datasets.

4.2. Previous Work in MTL for NLP

The NLP field contains many related tasks. This makes it a natural field for applying MTL, and sev-

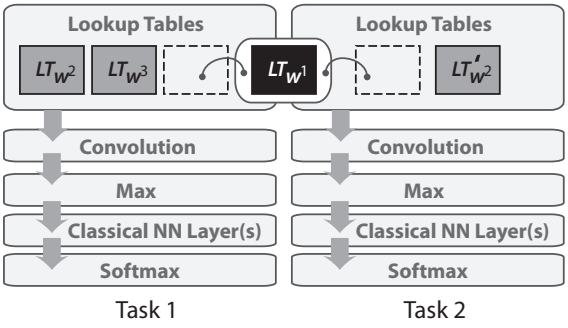

Figure 2. Example of deep multitasking with NN. Task 1 and Task 2 are two tasks trained with the architecture presented in Figure 1. One lookup-table (in black) is shared (the other lookup-tables and layers are task specific). The principle is the same with more than two tasks.

eral techniques have already been explored.

Cascading Features The most obvious way to achieve MTL is to train one task, and then use this task as a feature for another task. This is a very common approach in NLP. For example, in the case of SRL, several methods (e.g., (Pradhan et al., 2004)) train a POS classifier and use the output as features for training a parser, which is then used for building features for SRL itself. Unfortunately, tasks (features) are learnt separately in such a cascade, thus propagating errors from one classifier to the next.

Shallow Joint Training If one possesses a dataset labeled for several tasks, it is then possible to train these tasks jointly in a *shallow* manner: one unique model can predict all task labels at the same time. Using this scheme, the authors of (Sutton et al., 2007) proposed a conditional random field approach where they showed improvements from joint training on POS tagging and noun-phrase chunking tasks. However the requirement of jointly annotated data is a limitation, as this is often not the case. Similarly, in (Miller et al., 2000) NER, parsing and relation extraction were jointly trained in a statistical parsing model achieving improved performance on all tasks. This work has the same joint labeling requirement problem, which the authors avoided by using a predictor to fill in the missing annotations.

In (Sutton & McCallum, 2005a) the authors showed that one could learn the tasks independently, hence using different training sets, by only leveraging predictions jointly in a *test time* decoding step, and still obtain improved results. The problem is, however, that this will not make use of the shared tasks at *training time*. The NN approach used here seems more flexible in these regards.

Finally, the authors of (Musillo & Merlo, 2006) made an attempt at improving the semantic role labeling task by joint inference with syntactic parsing, but their results are not state-of-the-art. The authors of (Sutton & McCallum, 2005b) also describe a negative result at the same joint task.

5. Leveraging Unlabeled Data

Labeling a dataset can be an expensive task, especially in NLP where labeling often requires skilled linguists. On the other hand, unlabeled data is abundant and freely available on the web. Leveraging unlabeled data in NLP tasks seems to be a very attractive, and challenging, goal.

In our MTL framework presented in Figure 2, there is nothing stopping us from jointly training supervised tasks on labeled data *and* unsupervised tasks on unlabeled data. We now present an unsupervised task suitable for NLP.

Language Model We consider a *language model* based on a simple fixed window of text of size ksz using our NN architecture, given in Figure 2. We trained our language model to discriminate a two-class classification task: if the word in the middle of the input window is related to its context or not. We construct a dataset for this task by considering all possible ksz windows of text from the entire of English Wikipedia (`http://en.wikipedia.org`). Positive examples are windows from Wikipedia, negative examples are the same windows but where the middle word has been replaced by a random word.

We train this problem with a ranking-type cost:

$$\sum_{s \in \mathcal{S}} \sum_{w \in \mathcal{D}} \max\left(0,\, 1 - f(s) + f(s^w)\right) , \qquad (4)$$

where $\mathcal{S}$ is the set of sentence windows of text, $\mathcal{D}$ is the dictionary of words, and $f(\cdot)$ represents our NN architecture without the softmax layer and s^w is a sentence window where the middle word has been replaced by the word w. We sample this cost *online* w.r.t. (s, w).

We will see in our experiments that the features (embedding) learnt by the lookup-table layer of this NN clusters semantically similar words. These discovered features will prove very useful for our shared tasks.

Previous Work on Language Models (Bengio & Ducharme, 2001) and (Schwenk & Gauvain, 2002) already presented very similar language models. However, their goal was to give a *probability* of a word given *previous* ones in a sentence. Here, we only want to have a good representation of words: we take advantage of the complete context of a word (before and af-

ter) to predict its relevance. Perhaps this is the reason the authors were never able to obtain a good embedding of their words. Also, using probabilities imposes using a cross-entropy type criterion and can require many tricks to speed-up the training, due to normalization issues. Our criterion (4) is much simpler in that respect.

The authors of (Okanohara & Tsujii, 2007), like us, also take a two-class approach (true/fake sentences). They use a *shallow* (kernel) classifier.

Previous Work in Semi-Supervised Learning For an overview of semi-supervised learning, see (Chapelle et al., 2006). There have been several uses of semi-supervised learning in NLP before, for example in NER (Rosenfeld & Feldman, 2007), machine translation (Ueffing et al., 2007), parsing (McClosky et al., 2006) and text classification (Joachims, 1999). The first work is a highly problem-specific approach whereas the last three all use a self-training type approach (Transductive SVMs in the case of text classification, which is a kind of self-training method). These methods augment the training set with labeled examples from the unlabeled set which are predicted by the model itself. This can give large improvements in a model, but care must be taken as the predictions are of course prone to noise.

The authors of (Ando & Zhang, 2005) propose a setup more similar to ours: they learn from unlabeled data as an auxiliary task in a MTL framework. The main difference is that they use *shallow* classifiers; however they report positive results on POS and NER tasks.

Semantically Related Words Task We found it interesting to compare the embedding obtained with a language model on unlabeled data with an embedding obtained with labeled data. WordNet is a database which contains semantic relations (synonyms, holonyms, hypernyms, ...) between around 150,000 words. We used it to train a NN similar to the language model one. We considered the problem as a two-class classification task: positive examples are pairs with a relation in Wordnet, and negative examples are random pairs.

6. Experiments

We used Sections 02-21 of the PropBank dataset version 1 (about 1 million words) for training and Section 23 for testing as standard in all SRL experiments. POS and chunking tasks use the same data split via the Penn TreeBank. NER labeled data was obtained by running the Stanford Named Entity Recognizer (a CRF based classifier) over the same data.

Table 1. Language model performance for learning an embedding in $wsz = 50$ dimensions (dictionary size: 30,000). For each column the queried word is followed by its index in the dictionary (higher means more rare) and its 10 nearest neighbors (arbitrary using the Euclidean metric).

FRANCE	JESUS	XBOX	REDDISH	SCRATCHED
454	1973	6909	11724	29869
SPAIN	CHRIST	PLAYSTATION	YELLOWISH	SMASHED
ITALY	GOD	DREAMCAST	GREENISH	RIPPED
RUSSIA	RESURRECTION	PSNUMBER	BROWNISH	BRUSHED
POLAND	PRAYER	SNES	BLUISH	HURLED
ENGLAND	YAHWEH	WII	CREAMY	GRABBED
DENMARK	JOSEPHUS	NES	WHITISH	TOSSED
GERMANY	MOSES	NINTENDO	BLACKISH	SQUEEZED
PORTUGAL	SIN	GAMECUBE	SILVERY	BLASTED
SWEDEN	HEAVEN	PSP	GREYISH	TANGLED
AUSTRIA	SALVATION	AMIGA	PALER	SLASHED

Language models were trained on Wikipedia. In all cases, any numeric number was converted as "NUMBER". Accentuated characters were transformed to their non-accentuated versions. All paragraphs containing other non-ASCII characters were discarded. For Wikipedia, we obtain a database of *631M* words. We used WordNet to train the "synonyms" (semantically related words) task.

All tasks use the same dictionary of the 30,000 most common words from Wikipedia, converted to lower case. Other words were considered as *unknown* and mapped to a special word.

Architectures All tasks were trained using the NN shown in Figure 1. POS, NER, and chunking tasks were trained with the window version with $ksz = 5$. We chose linear models for POS and NER. For chunking we chose a hidden layer of 200 units. The language model task had a window size $ksz = 11$, and a hidden layer of 100 units. All these tasks used two lookup-tables: one of dimension wsz for the word in lower case, and one of dimension 2 specifying if the first letter of the word is a capital letter or not.

For SRL, the network had a convolution layer with $ksz = 3$ and 100 hidden units, followed by another hidden layer of 100 hidden units. It had three lookup-tables in the first layer: one for the word (in lower case), and two that encode relative distances (to the word of interest and the verb). The last two lookup-tables embed in 5 dimensional spaces. Verb positions are obtained with our POS classifier.

The language model network had only one lookup-table (the word in lower case) and 100 hidden units. It used a window of size $ksz = 11$.

We show results for different encoding sizes of the word in lower case: $wsz = 15$, 50 and 100.

Table 2. A Deep Architecture for SRL improves by learning auxiliary tasks that share the first layer that represents words as wsz-dimensional vectors. We give word error rates for wsz=15, 50 and 100 and various shared tasks.

	wsz=15	wsz=50	wsz=100
SRL	16.54	17.33	18.40
SRL + POS	15.99	16.57	16.53
SRL + Chunking	16.42	16.39	16.48
SRL + NER	16.67	17.29	17.21
SRL + Synonyms	15.46	15.17	15.17
SRL + Language model	14.42	14.30	14.46
SRL + POS + Chunking	16.46	15.95	16.41
SRL + POS + NER	16.45	16.89	16.29
SRL + POS + Chunking + NER	16.33	16.36	16.27
SRL + POS + Chunking + NER + Synonyms	15.71	14.76	15.48
SRL + POS + Chunking + NER + Language model	14.63	14.44	14.50

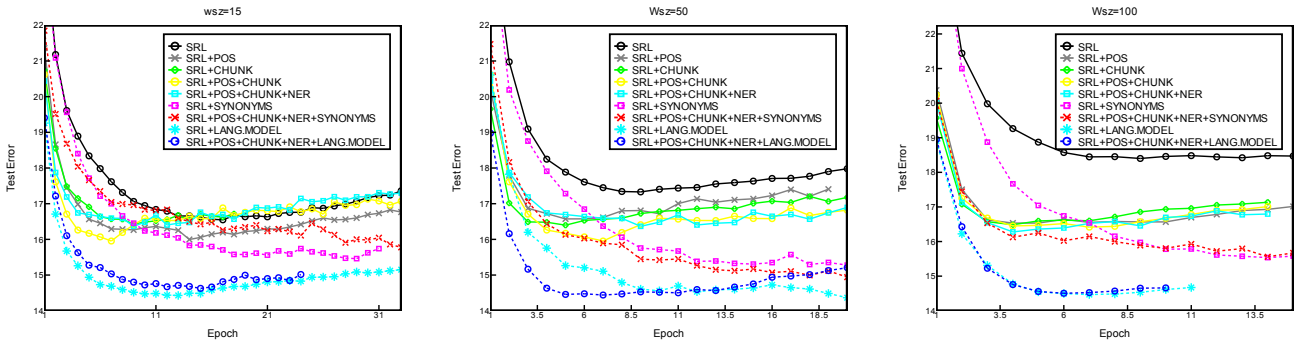

Figure 3. Test error versus number of training epochs over PropBank, for the SRL task alone and SRL jointly trained with various other NLP tasks, using deep NNs.

Results: Language Model Because the language model was trained on a huge database we first trained it *alone*. It takes about a week to train on one computer. The embedding obtained in the word lookup-table was extremely good, even for uncommon words, as shown in Table 1. The embedding obtained by training on labeled data from WordNet "synonyms" is also good (results not shown) however the coverage is not as good as using unlabeled data, e.g. "Dreamcast" is not in the database.

The resulting word lookup-table from the language model was used as an initializer of the lookup-table used in MTL experiments with a language model.

Results: SRL Our main interest was improving SRL performance, the most complex of our tasks. In Table 2, we show results comparing the SRL task alone with the SRL task jointly trained with different combinations of the other tasks. For all our experiments, training was achieved in a few epochs (about a day) over the PropBank dataset as shown in Figure 3. Testing takes 0.015s to label a complete sentence (given one verb).

All MTL experiments performed better than SRL alone. With larger wsz (and thus large capacity) the relative improvement becomes larger from using MTL compared to the task alone, which shows MTL is a good way of regularizing: in fact with MTL results are fairly stable with capacity changes.

The *semi-supervised* training of SRL using the language model performs better than other combinations. Our best model performed as low as 14.30% in per-word error rate, which is to be compared to previously published results of 16.36% with an NN architecture (Collobert & Weston, 2007) and 16.54% for a state-of-the-art method based on parse trees (Pradhan et al., 2004)[1]. Further, our system is the only one not to use POS tags or parse tree features.

Results: POS and Chunking Training takes about 30 min for these tasks alone. Testing time for labeling a complete sentence is about 0.003s. We obtained modest improvements to POS and chunking results us-

[1]Our loss function optimized per-word error rate. We note that many SRL results e.g. the CONLL 2005 evaluation use F1 as a standard measure.

ing MTL. Without MTL (for $wsz = 50$) we obtain 2.95% test error for POS and 4.5% (91.1 F-measure) for chunking. With MTL we obtain 2.91% for POS and 3.8% (92.71 F-measure) for chunking. POS error rates in the 3% range are state-of-the-art. For chunking, although we use a different train/test setup to the CoNLL-2000 shared task (`http://www.cnts.ua.ac.be/conll2000/chunking`) our system seems competitive with existing systems (better than 9 of the 11 submitted systems). However, our system is the only one that does *not use* POS tags as input features.

Note, we did not evaluate NER error rates because we used non-gold standard annotations in our setup. Future work will more thoroughly evaluate these tasks.

7. Conclusion

We proposed a general deep NN architecture for NLP. Our architecture is extremely fast enabling us to take advantage of huge databases (e.g. 631 million words from Wikipedia). We showed our deep NN could be applied to various tasks such as SRL, NER, POS, chunking and language modeling. We demonstrated that learning tasks simultaneously can improve generalization performance. In particular, when training the SRL task jointly with our language model our architecture achieved state-of-the-art performance in SRL without any explicit syntactic features. This is an important result, given that the NLP community considers syntax as a mandatory feature for semantic extraction (Gildea & Palmer, 2001).

References

Ando, R., & Zhang, T. (2005). A Framework for Learning Predictive Structures from Multiple Tasks and Unlabeled Data. *JMLR*, *6*, 1817–1853.

Bengio, Y., & Ducharme, R. (2001). A neural probabilistic language model. *NIPS 13*.

Bridle, J. (1990). Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In F. F. Soulié and J. Hérault (Eds.), *Neurocomputing: Algorithms, architectures and applications*, 227–236. NATO ASI Series.

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

Chapelle, O., Schlkopf, B., & Zien, A. (2006). *Semi-supervised learning*. Adaptive computation and machine learning. Cambridge, Mass., USA: MIT Press.

Collobert, R., & Weston, J. (2007). Fast semantic extraction using a novel neural network architecture. *Proceedings of the 45th Annual Meeting of the ACL* (pp. 560–567).

Gildea, D., & Palmer, M. (2001). The necessity of parsing for predicate argument recognition. *Proceedings of the 40th Annual Meeting of the ACL*, 239–246.

Joachims, T. (1999). Transductive inference for text classification using support vector machines. *ICML*.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, *86*.

McClosky, D., Charniak, E., & Johnson, M. (2006). Effective self-training for parsing. *Proceedings of HLT-NAACL 2006*.

Miller, S., Fox, H., Ramshaw, L., & Weischedel, R. (2000). A novel use of statistical parsing to extract information from text. *6th Applied Natural Language Processing Conference*.

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

Okanohara, D., & Tsujii, J. (2007). A discriminative language model with pseudo-negative samples. *Proceedings of the 45th Annual Meeting of the ACL*, 73–80.

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

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

Rosenfeld, B., & Feldman, R. (2007). Using Corpus Statistics on Entities to Improve Semi-supervised Relation Extraction from the Web. *Proceedings of the 45th Annual Meeting of the ACL*, 600–607.

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

Sutton, C., & McCallum, A. (2005a). Composition of conditional random fields for transfer learning. *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, 748–754.

Sutton, C., & McCallum, A. (2005b). Joint parsing and semantic role labeling. *Proceedings of CoNLL-2005* (pp. 225–228).

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

Ueffing, N., Haffari, G., & Sarkar, A. (2007). Transductive learning for statistical machine translation. *Proceedings of the 45th Annual Meeting of the ACL*, 25–32.

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