NLP Paper Reading 1: A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning (2008)

Jorge Balazs

February 2, 2019

Paper Structure

Paper Structure

Abstract

1. Introduction
2. NLP Tasks
3. General Deep Architecture for NLP
    - 3.1 Transforming Indices into Vectors
    - 3.2 Variable Sentence Length
    - 3.3 Deep Architecture
    - 3.4 Related Architectures
4. Multitasking with Deep NN
    - 4.1 Deep Joint Training
    - 4.2 Previous Work in MLT for NLP
5. Leveraging Unlabeled Data
6. Experiments
7. Conclusion

Abstract

Abstract

What:

- Show that both *multitask learning* (MTL) and *semi-supervised learning* improve the generalization of shared tasks.

How:

- Network is trained *jointly* on several NLP tasks.
- Sentence $\xrightarrow{\text{input}}$ CNN Architecture $\xrightarrow{\text{output}}$ host of NLP predictions.

Introduction

Introduction - Background

- NLP = human language ⟶ formal representation understandable by computers.
- Some applications: information extraction, machine translation, summarization, search, human-computer interfaces. Can you think of any others?
- Semantic understanding still is a far distant goal. This is still true today.

> NLP is usually addressed by solving several *syntactic* and *semantic* subtasks (eg, POS tagging, syntactic parsing, and semantic-role labeling).

- It is common to analyze these tasks separately.

Introduction - Current Limitations

- Common failings of current approaches
  1. They are shallow, ie classifiers are linear. What does this mean?
  2. Linear classifiers need good hand-engineered features. Why is this not desirable?
  3. Features learned in different tasks are *cascaded*, which propagates errors.

Introduction - Proposal

A unified architecture that *learns features* relevant to the tasks at hand, with little prior knowledge.

How?

By *jointly* training a Deep Neural Network for solving several NLP tasks.

The language model is an exception; we'll talk about this later.

Introduction - Contributions

They show that:

- Multitask learning and semi-supervised learning significantly improve performance of Semantic-role labeling, *without hand-engineered features*.
- Combined tasks (MTL), and the unsupervised task (Language Model), learn features that encode semantic information with no other supervision than the labeled data from the tasks.

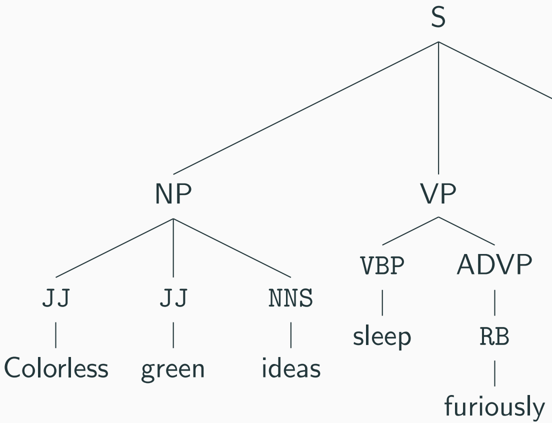
Does this sound familiar? Remember this paper was written more than 8 years ago, before AlexNet in 2012 and before the Deep Learning wave.

NLP Tasks

Part-of-Speech Tagging (POS)

Colorless	green	ideas	sleep	furiously
JJ	JJ	NNS	VBP	RB

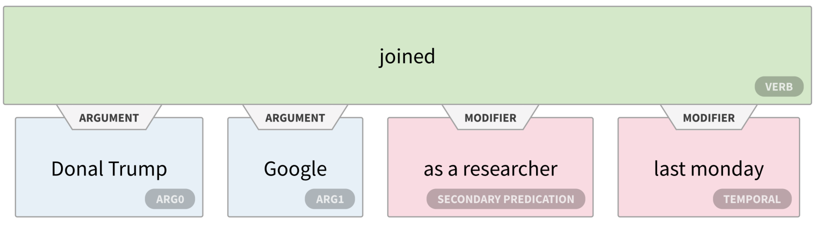
Chunking


S
NP
VP
.
JJ
JJ
NNS
VBP
ADVP
.
Colorless
green
ideas
sleep
RB
furiously

Named Entity Recognition (NER)

Donal Trump	joined	Google	as a researcher	last monday
PERSON		ORG		DATE

Semantic Role Labeling (SRL)



joined

VERB

ARGUMENT

Donal Trump

ARG0

ARGUMENT

Google

ARG1

MODIFIER

as a researcher

SECONDARY PREDICATION

MODIFIER

last monday

TEMPORAL

Language Models

Predict the next word, given the previous words.

> *Language modeling is the task of assigning a probability to sentences in a language [...]. Besides assigning a probability to each sequence of words, the language models also assigns a probability for the likelihood of a given word (or a sequence of words) to follow a sequence of words [...] [Goldberg(2017)].*

Semantically Related Words

Predicting whether two words are semantically related:

- Mammal $\xrightleftharpoons[HYPONYM]{HYPERNYM}$ Cat
- Car $\xrightleftharpoons[MERONYM]{HOLONYM}$ Wheel
- PC $\xleftrightarrow{SYNONYM}$ Computer

General Deep Architecture for NLP

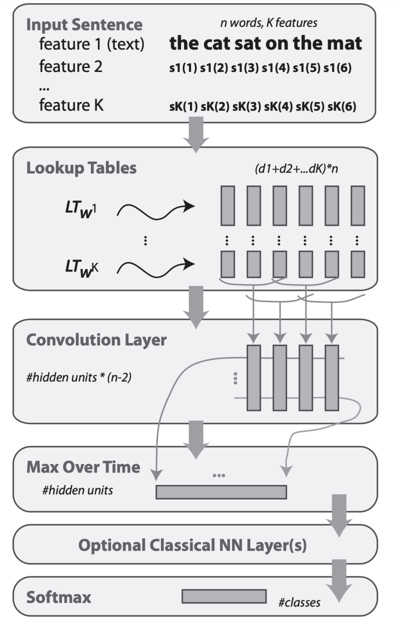
General Deep Architecture for NLP - Traditional Approaches

- The previous tasks can all be seen as assigning labels to words.
- The common approach is to manually transform input sentences into vectors (feature engineering), and feed these vectors to a shallow classifier (eg, SVM).
- The choice of features (feature engineering), is an empirical process dependent on the task at hand, mainly based on trial and error, and difficult to scale.

General Deep Architecture for NLP - Proposed Approach

- They propose a Deep Neural Network (NN) trained in an end-to-end fashion.
- Features (intermediate representations between the input and output) are *learned automatically* by the NN.

General Deep Architecture for NLP - Proposed Approach



General Deep Architecture for NLP - Transforming Indices into Vectors

- They assume a mapping between words and a set of indices $\mathcal{D}$ which is a subset of the Natural numbers ($\mathcal{D} \subset \mathbb{N}$).
  - $|\mathcal{D}|$ is the number of elements in $\mathcal{D}$, ie the words we "know" (words that appear in the training set).
  - $\mathcal{D} = \{1, 2, \ldots, |\mathcal{D}|\}$

General Deep Architecture for NLP - Transforming Indices into Vectors

- They also define the relationship between these indices and an *embedding lookup table*.
    - This lookup table is simply a matrix $\boldsymbol{W}$ where each column corresponds to a vector of dimension d that represents one of the indices in $\mathcal{D}$. Therefore $\boldsymbol{W}$ has d rows and $|\mathcal{D}|$ columns, and since it contains only real numbers we say that $\boldsymbol{W} \in \mathbb{R}^{d \times |\mathcal{D}|}$.
    - $LT_W(i)$ is a function that takes an index i in $\mathcal{D}$ as input, and returns the i-th column of $\boldsymbol{W}$; a vector $\boldsymbol{W}_i$ of dimension d ($\boldsymbol{W}_i \in \mathbb{R}^d$).
    - These vectors are learned during training.

General Deep Architecture for NLP - Transforming Indices into Vectors

- In general, we can have mappings between any kind of word-level feature and a set of indices.
  - For example, capitalization of words is a feature. Our "vocabulary" of capitalization features is $\mathcal{D}^{cap} = \{1, 2\}$, where 1 corresponds to non-capitalized and 2 to capitalized.
  - Similar to the embedding lookup table $\boldsymbol{W}$ mentioned previously, we can define $\boldsymbol{W}^{cap} \in \mathbb{R}^{m \times 2}$.[1]
  - Equivalently, we will also have a mapping between the elements in $\mathcal{D}^{cap}$ and $\boldsymbol{W}^{cap}$ given by $LT_{W^{cap}}(i) = \boldsymbol{W}_i^{cap}$.

[1] m is the dimensionality we chose for these specific vectors, and $|\mathcal{D}^{cap}| = 2$. Also note that m does not necessarily equal d.

General Deep Architecture for NLP - Transforming Indices into Vectors

- For each word, we will have several word-level features. For each of these features we will have a single vector.
- The representation of each word, will be the *concatenation* of these vectors.
- For example, if we only consider the words themselves and their capitalization (see previous slides), each word will be represented by a vector $\in \mathbb{R}^{m+d}$.

General Deep Architecture for NLP - Variable Sentence Length

- Let's assume that we have a sequence $\{s_1, \ldots, s_n\}$ of n words, each with a corresponding vector $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ obtained through the procedure described earlier.
- Each $\boldsymbol{x}_i$ will have the same dimension, say d, ie $\boldsymbol{x}_i \in \mathbb{R}^d$.[2]
- Clearly the number of vectors we end up with, n, depends on the length of the original sentence, however NNs know only how to deal with fixed-length inputs.

How do we solve this problem?

[2] Not the same d mentioned earlier; just a redefinition.

General Deep Architecture for NLP - Variable Sentence Length

- A simple approach is to use *window approach* with a window size of ksz. What exactly do they mean by a window approach?
- This approach works well for modeling local phenomena, which is beneficial for tasks such as POS tagging.
- It fails, however, when modeling long-range dependencies, which is required for more complex tasks such as SRL.

Why is it ok to model local phenomena in POS tagging but not in SRL?

General Deep Architecture for NLP - Variable Sentence Length

- An alternative approach is to use Time-Delay Neural Networks (roughly equivalent to CNNs), which are capable of modeling long-range dependencies.
- $\boldsymbol{o}(t) = \sum_{j=1-t}^{n-t} \boldsymbol{L}_j \cdot \boldsymbol{x}_{t+j}$, where $\boldsymbol{L}_j \in \mathbb{R}^{n_{hu} \times d}$. Remember that our word vectors $\boldsymbol{x}_t \in \mathbb{R}^d$, which means that $\boldsymbol{o}(t) \in \mathbb{R}^{n_{hu}}$ for every t.

General Deep Architecture for NLP - Variable Sentence Length

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

$$\boldsymbol{o}(1) = \textstyle\sum_{j=0}^{n-1} \boldsymbol{L}_j \cdot \boldsymbol{x}_{1+j} = \boldsymbol{L}_0\boldsymbol{x}_1 + \boldsymbol{L}_1\boldsymbol{x}_2 + \ldots + \boldsymbol{L}_{n-1}\boldsymbol{x}_n$$

$$\boldsymbol{o}(2) = \textstyle\sum_{j=-1}^{n-2} \boldsymbol{L}_j \cdot \boldsymbol{x}_{2+j} = \boldsymbol{L}_{-1}\boldsymbol{x}_1 + \boldsymbol{L}_0\boldsymbol{x}_2 + \ldots + \boldsymbol{L}_{n-2}\boldsymbol{x}_n$$

$$\vdots$$

$$\boldsymbol{o}(t) = \textstyle\sum_{j=1-t}^{n-t} \boldsymbol{L}_j \cdot \boldsymbol{x}_{t+j} = \boldsymbol{L}_{1-t}\boldsymbol{x}_1 + \boldsymbol{L}_{2-t}\boldsymbol{x}_2 + \ldots + \boldsymbol{L}_{n-t}\boldsymbol{x}_n$$

$$\vdots$$

$$\boldsymbol{o}(n) = \textstyle\sum_{j=1-n}^{0} \boldsymbol{L}_j \cdot \boldsymbol{x}_{n+j} = \boldsymbol{L}_{1-n}\boldsymbol{x}_1 + \boldsymbol{L}_{2-n}\boldsymbol{x}_2 + \ldots + \boldsymbol{L}_0\boldsymbol{x}_n$$

General Deep Architecture for NLP - Variable Sentence Length

Let's call the window size w, i.e., $w = ksz$. We impose $\boldsymbol{L}_j = \mathbf{0}$ if $|j| > \frac{(w-1)}{2}$, or, in other words, $\boldsymbol{L}_j$ will be non-zero iff

$$-\frac{(w-1)}{2} \leq j \leq \frac{(w-1)}{2}$$

Let's set $w = 3$, i.e., $\boldsymbol{L}_j \neq \mathbf{0}$ if $-1 \leq j \leq 1$

Our convolution operation becomes:

$$\boldsymbol{o}(t) = \sum_{-1 \leq j \leq 1} \boldsymbol{L}_j \cdot \boldsymbol{x}_{t+j}$$

Which means that each sum will have only $w = 3$ elements.

General Deep Architecture for NLP - Variable Sentence Length

$$\boldsymbol{o}(t) = \sum_{-1 \le j \le 1} \boldsymbol{L}_j \cdot \boldsymbol{x}_{t+j}$$

$$\begin{aligned}
\boldsymbol{o}(1) &= \boldsymbol{L}_{-1}\boldsymbol{x}_0 + \boldsymbol{L}_0\boldsymbol{x}_1 + \boldsymbol{L}_1\boldsymbol{x}_2 \\
\boldsymbol{o}(2) &= \boldsymbol{L}_{-1}\boldsymbol{x}_1 + \boldsymbol{L}_0\boldsymbol{x}_2 + \boldsymbol{L}_1\boldsymbol{x}_3 \\
&\vdots \\
\boldsymbol{o}(t) &= \boldsymbol{L}_{-1}\boldsymbol{x}_{t-1} + \boldsymbol{L}_0\boldsymbol{x}_t + \boldsymbol{L}_1\boldsymbol{x}_{t+1} \\
&\vdots \\
\boldsymbol{o}(n-1) &= \boldsymbol{L}_{-1}\boldsymbol{x}_{n-2} + \boldsymbol{L}_0\boldsymbol{x}_{n-1} + \boldsymbol{L}_1\boldsymbol{x}_n \\
\boldsymbol{o}(n) &= \boldsymbol{L}_{-1}\boldsymbol{x}_{n-1} + \boldsymbol{L}_0\boldsymbol{x}_n + \boldsymbol{L}_1\boldsymbol{x}_{n+1}
\end{aligned}$$

General Deep Architecture for NLP - Variable Sentence Length

- Before the TDNN we had n word vectors $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\},\ \boldsymbol{x}_t \in \mathbb{R}^d$; one for each word of our sequence, obtained from a lookup embedding table.
- After the TDNN, we have n vectors $\{\boldsymbol{o}_1, \ldots, \boldsymbol{o}_n\},\ \boldsymbol{o}_t \in \mathbb{R}^{n_{hu}}$, still one for each word of our sequence, where each $\boldsymbol{o}_t$ encodes a notion of its neighborhood.
- Also note that the previous statement is true independently of the window size we chose.
- This allows to easily pass the output to the first TDNN to another TDNN with different parameters and maybe a different window size (what the authors refer to as *stacking*).

General Deep Architecture for NLP - Variable Sentence Length

- After the previous steps we still have n vectors though. To obtain a single vector for representing the sentence the authors use a “Max” Layer (aka maxpooling), over the sequence dimension.
- The maxpooling operation reduces the sequence dimension and returns a single vector $\boldsymbol{o} \in \mathbb{R}^{n_{hu}}$ representing the whole sentence.
- This sentence representation can then be passed to a Feedforward (aka dense) NN layer.
- Finally, they encode the position of the word to be labelled with an additional lookup table $LT^{dist_w}(i - pos_w)$.

General Deep Architecture for NLP - Deep Architecture

- This part describes how they transform the maxpooled output of the TDNN $\boldsymbol{o} \in \mathbb{R}^{n_{hu}}$ into a vector $\boldsymbol{o}^{last} \in \mathbb{R}^{k}$ where k is the number of classes in the NLP classification task, for example, the number of possible POS tags.
- Finally, $\boldsymbol{o}^{last} \in \mathbb{R}^{k}$ is fed to a softmax layer, which returns a vector representing a probability distribution over the possible classes: $\boldsymbol{p} \in \mathbb{R}^{k}$, where $0 \leq \boldsymbol{p}_i \leq 1$ and $\sum_{i=1}^{k} \boldsymbol{p}_i = 1$

Multitasking with Deep NN

Multitasking with Deep NN

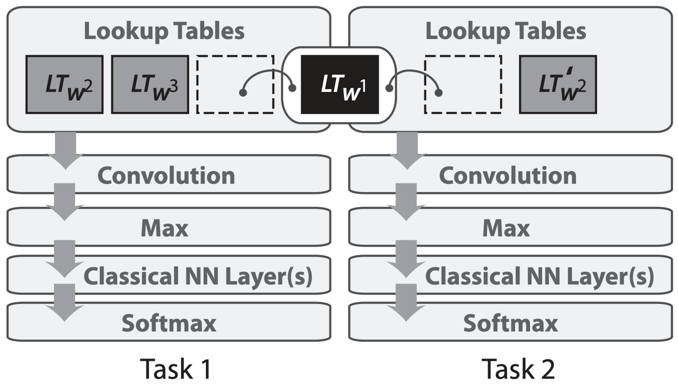
Multitask learning is the procedure of learning several tasks at the same time with the aim of mutual benefit.

Multitasking with Deep NN - Deep joint Training

- The benefit comes from learning features (representations) that are beneficial for several tasks (*more general* features).
- Usually, POS predictions are often used for solving the NER and SRL tasks. Having good representations for POS tagging, might improve both NER and SRL.
- Earliest layers (lookup tables) implicitly learn relevant word representations for each word. It's reasonable to expect that these features will perform well in related tasks.
- Earlier layers can be shared accross tasks while later ones can be task-specific.

Multitasking with Deep NN - Deep joint Training

- Their model is trained one example at a time.
- At each iteration they pick a random example from a random task, feed it forward, obtain the loss, calculate the gradients, and backpropagate them.



Leveraging Unlabeled Data

Leveraging Unlabeled Data

- Labeling is expensive.
- Leveraging unlabeled data in NLP seems like a good idea. 10 years later this still rings true.

Leveraging Unlabeled Data - Language Model

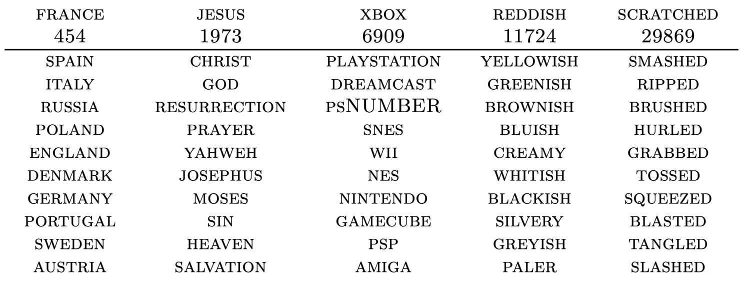
- 2-class classification task: Predict whether the middle word of a window is related to its context or not.
- Training dataset was built from Wikipedia. Positive examples are real extracts from Wikipedia, negative ones are the same ones but with the middle word replaced by a random one.
- Ranking-type (hinge) loss:

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

$\mathcal{S}$ is the set of sentence windows, and $\mathcal{D}$ the set of words in the vocabulary.

Leveraging Unlabeled Data - Language Model

Word representations learned by this model cluster for semantically-similar words. In other words, the embedding space learned by this model encodes semantic similarity.



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

Experiments

Experiments - Results

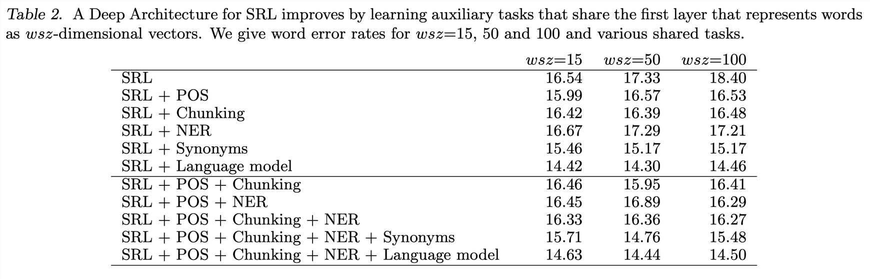
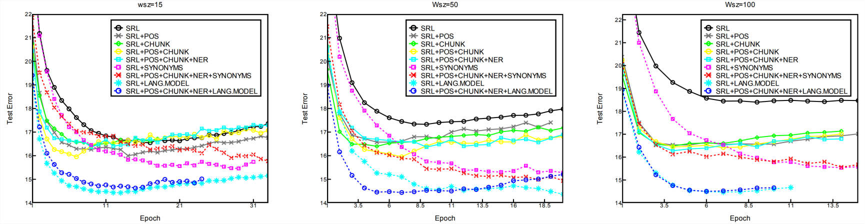


Table 2. A Deep Architecture for SRL improves by learning auxiliary tasks that share the first layer that represents words as wsz-dimensional vectors. We give word error rates for wsz=15, 50 and 100 and various shared tasks.

	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

Experiments - Results



Conclusion

Conclusion

- They showed a deep NN for NLP.
- Architecture is extremely fast.
- Architecture can be applied to several tasks.
- **Learning tasks simultaneously improved generalization performance.**
- Achieved state of the art in SRL when training this task jointly with their language model *without any explicit syntactic features*.

Discussion

Yoav Goldberg. 2017.
Neural Network Methods for Natural Language Processing.
Synthesis Lectures on Human Language Technologies 10(1):1–309.