NLP Reading Group — Meeting 3

Matt Smith

UW Data Science Club

14 March, 2019

Coverage

Show and Tell

Did anyone do the practice and get interesting results?

I tried to get GPT-2 to generate memes and failed.

Coverage

Section 4 – Experiment Settings

We evaluate the proposed approach on the task of English-to-French translation. We use the bilingual, parallel corpora provided by ACL WMT ' 14. As a comparison, we also report the performance of an RNN Encoder–Decoder which was proposed recently by Cho et al. (2014a). We use the same training procedures and the same dataset for both models.

WMT ' 14 contains the following English-French parallel corpora: Europarl (61M words), news commentary (5.5M), UN (421M) and two crawled corpora of 90M and 272.5M words respectively, totaling 850M words. Following the procedure described in Cho et al. (2014a), we reduce the size of the combined corpus to have 348M words using the data selection method by Axelrod et al. (2011). We do not use any monolingual data other than the mentioned parallel corpora, although it may be possible to use a much larger monolingual corpus to pretrain an encoder.

Section 4 – Experiment Settings

We concatenate news-test-2012 and news-test-2013 to make a development (validation) set, and evaluate the models on the test set (news-test-2014) from WMT ' 14, which consists of 3003 sentences not present in the training data.

After a usual tokenization, we use a shortlist of 30,000 most frequent words in each language to train our models. Any word not included in the shortlist is mapped to a special token ([UNK]). We do not apply any other special preprocessing, such as lowercasing or stemming, to the data.

Section 4 – Experiment Settings

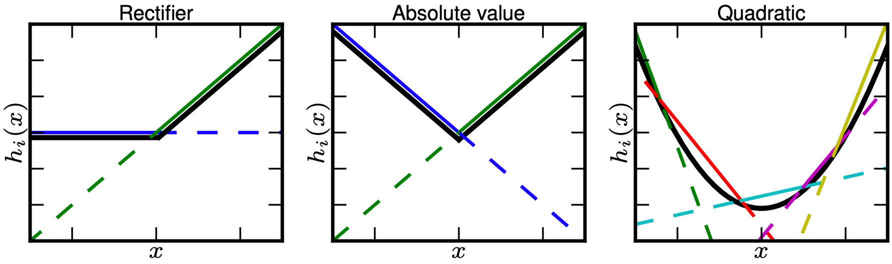
We train two types of models. The first one is an RNN Encoder–Decoder (RNNencdec, Cho et al., 2014a), and the other is the proposed model, to which we refer as RNNsearch. We train each model twice: first with the sentences of length up to 30 words (RNNencdec-30, RNNsearch-30) and then with the sentences of length up to 50 word (RNNencdec-50, RNNsearch-50). The encoder and decoder of the RNNencdec have 1000 hidden units each. The encoder of the RNNsearch consists of forward and backward recurrent neural networks (RNN) each having 1000 hidden units. Its decoder has 1000 hidden units. In both cases, we use a multilayer network with a single maxout (Goodfellow et al., 2013) hidden layer to compute the conditional probability of each target word (Pascanu et al., 2014).

Maxout

A new (as of 2013) type of activation function that has nice properties when trained on networks using dropout.

Given input x, the activation is

$$\text{maxout}(x) := \max_i x_i$$



AFAIK not used much today.

Section 4 – Experiment Settings

We use a minibatch stochastic gradient descent (SGD) algorithm together with Adadelta (Zeiler, 2012) to train each model. Each SGD update direction is computed using a minibatch of 80 sentences. We trained each model for approximately 5 days.

Once a model is trained, we use a beam search to find a translation that approximately maximizes the conditional probability (see, e.g., Graves, 2012; Boulanger-Lewandowski et al., 2013). Sutskever et al. (2014) used this approach to generate translations from their neural machine translation model.

Decoding – Beam Search

Each iteration of the decoder takes a sentence fragment and produces a probability.

However we can't just sample from this probability distribution or pick the most likely word at east position. Doing so produces unnatural outputs as we ignore correlations between words.

Example

Say you're decoding a vector representing 'I am visiting my family'.
After you read 'I am', a greedy choice of the next word could be 'going' as the sentence 'I am going to visit my family' seems partially correct and 'I am going' appears more often in English than 'I am visiting'.
However the sentence 'I am going to visit my family' is less natural and thus has a lower total probability.

Decoding – Beam Search

We want to pick an output sentence with the highest total probability. Beam search is a heuristic BFS-style search that can do this.

Beam search stores the B most likely sentence fragments so far and extends them until the translation is done.

In our previous example we would see ‘I am going’ as more likely than ‘I am visiting’, but beam search would store both as candidates and keep extending them. After extension we'd find that $P(\text{I am visiting my family}) > P(\text{I am going to visit my family})$ and get the better decoding.

Think about how you'd pick B & the performance concerns. Some research papers set it ridiculously high in order to get good results.

Decoding

Another common type of decoding is called the CRF. CRFs are more common when decoding tagging (I've never seen one used in NMT).

CRFs

I won't go into detail about how CRFs work but they are somewhat like Markov models that learn a transition matrix between adjacent words and maximize the resulting probability exactly.
It's enough to know that if you see a CRF, you should think 'decoding'.

Proper decoding is extremely slow. Some papers (like the GPT-2 paper) use greedy decoding or top-k decoding (only sample from the k most likely words).

Section 5 – Experimental Results

In Table 1, we list the translation performances measured in BLEU score. It is clear from the table that in all the cases, the proposed RNNsearch outperforms the conventional RNNencdec. More importantly, the performance of the RNNsearch is as high as that of the conventional phrase-based translation system (Moses), when only the sentences consisting of known words are considered. This is a significant achievement, considering that Moses uses a separate monolingual corpus (418M words) in addition to the parallel corpora we used to train the RNNsearch and RNNencdec.

BLEU Score

BLEU (bilingual evaluation understudy) score is a model evaluation metric that compares machine-generated translations against a handful of known good human-generated translations.

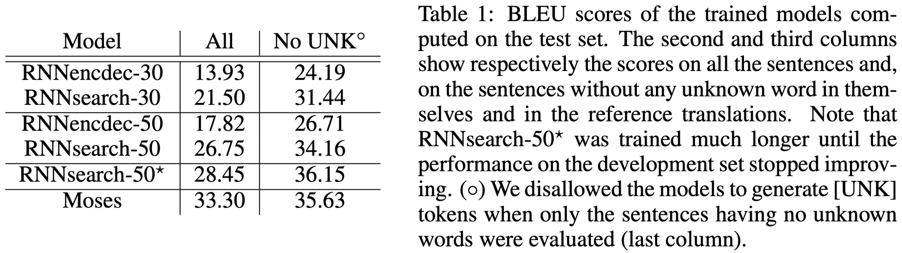
Bad BLEU

There is some debate on if BLEU is a good metric or not.
Read *Re-evaluating the Role of BLEU in Machine Translation Research*.

Calculating BLEU is complicated (there is not one definitive algorithm) but the main idea is that you want to measure the proportion of n-grams in the candidate text that appear in a translated text.

Other penalties are applied for excess brevity or repeated n-grams.

Section 5 – Experimental Results



Model	All	No UNK°
RNNencdec-30	13.93	24.19
RNNsearch-30	21.50	31.44
RNNencdec-50	17.82	26.71
RNNsearch-50	26.75	34.16
RNNsearch-50*	28.45	36.15
Moses	33.30	35.63

Table 1: BLEU scores of the trained models computed on the test set. The second and third columns show respectively the scores on all the sentences and, on the sentences without any unknown word in themselves and in the reference translations. Note that RNNsearch-50* was trained much longer until the performance on the development set stopped improving. ($\circ$) We disallowed the models to generate [UNK] tokens when only the sentences having no unknown words were evaluated (last column).

Section 5 – Experimental Results

One of the motivations behind the proposed approach was the use of a fixed-length context vector in the basic encoder–decoder approach. We conjectured that this limitation may make the basic encoder–decoder approach to underperform with long sentences. In Fig. 2, we see that the performance of RNNencdec dramatically drops as the length of the sentences increases. On the other hand, both RNNsearch-30 and RNNsearch-50 are more robust to the length of the sentences. RNNsearch50, especially, shows no performance deterioration even with sentences of length 50 or more. This superiority of the proposed model over the basic encoder–decoder is further confirmed by the fact that the RNNsearch-30 even outperforms RNNencdec-50 (see Table 1).

Section 5 – Experimental Results

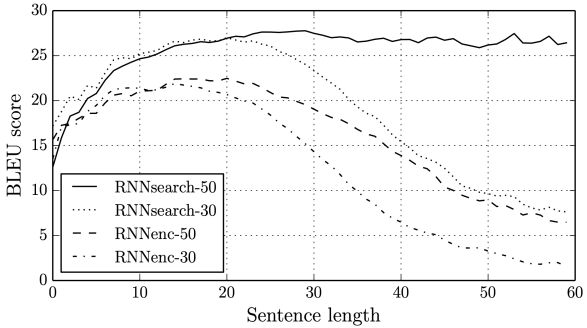
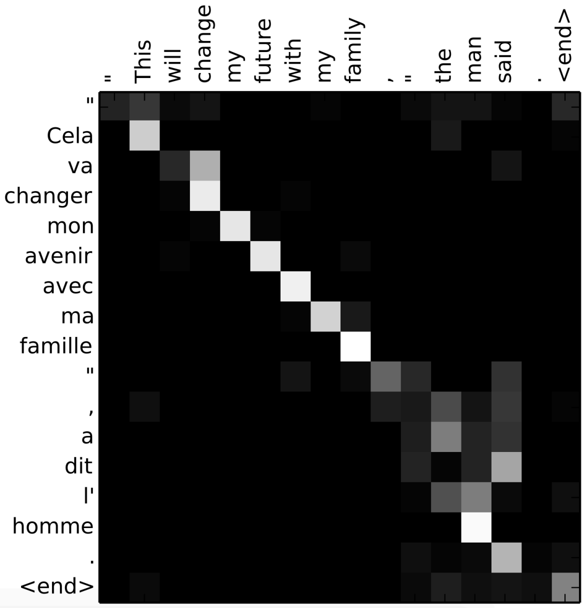


Figure 2: The BLEU scores of the generated translations on the test set with respect to the lengths of the sentences. The results are on the full test set which includes sentences having unknown words to the models.

Section 5 – Evaluating Alignments

The strength of the soft-alignment, opposed to a hard-alignment, is evident, for instance, from Fig. 3. Consider the source phrase [the man] which was translated into [l' homme]. Any hard alignment will map [the] to [l'] and [man] to [homme]. This is not helpful for translation, as one must consider the word following [the] to determine whether it should be translated into [le], [la], [les] or [l']. Our soft-alignment solves this issue naturally by letting the model look at both [the] and [man], and in this example, we see that the model was able to correctly translate [the] into [l']. We observe similar behaviors in all the presented cases in Fig. 3. An additional benefit of the soft alignment is that it naturally deals with source and target phrases of different lengths, without requiring a counter-intuitive way of mapping some words to or from nowhere ([NULL]) (see, e.g., Chapters 4 and 5 of Koehn, 2010).

Section 5 – Evaluating Alignments



Section 5 – Long Sentences

As clearly visible from Fig. 2 the proposed model (RNNsearch) is much better than the conventional model (RNNencdec) at translating long sentences. This is likely due to the fact that the RNNsearch does not require encoding a long sentence into a fixed-length vector perfectly, but only accurately encoding the parts of the input sentence that surround a particular word.

In conjunction with the quantitative results presented already, these qualitative observations confirm our hypotheses that the RNNsearch architecture enables far more reliable translation of long sentences than the standard RNNencdec model.

Section 5 – Long Sentences

The authors also give a list of translations showing where RNNencdec gets confused within larger sentences.

The failure modes are common in encoder-decoders: forgetting to close quotations or end with a quote mark, changing topics partway through the sentence or using different names to refer to the same person or thing.

Coverage

Character-Aware Neural Language Models

Necessary to understand the ELMO paper.

ELMO paper is required for understanding the BERT paper.

BERT was SOTA until GPT-2 came out, and BERT is still very good considering it has way fewer parameters than GPT-2.

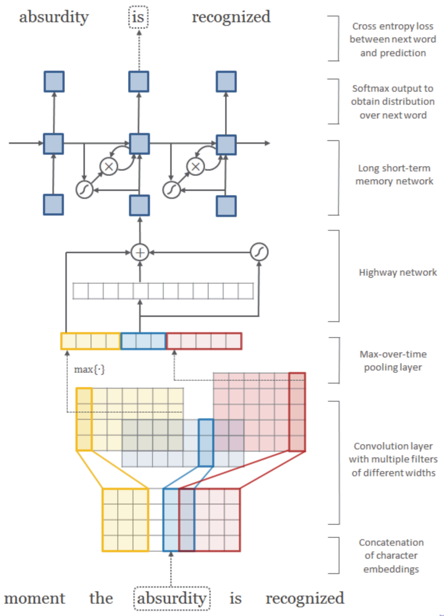
Abstract

We describe a simple neural language model that relies only on character-level inputs. Predictions are still made at the word-level. Our model employs a convolutional neural network (CNN) and a highway network over characters, whose output is given to a long short-term memory (LSTM) recurrent neural network language model (RNN-LM). On the English Penn Treebank the model is on par with the existing state-of-the-art despite having 60% fewer parameters. On languages with rich morphology (Arabic, Czech, French, German, Spanish, Russian), the model outperforms word-level/morpheme-level LSTM baselines, again with fewer parameters. The results suggest that on many languages, character inputs are sufficient for language modeling. Analysis of word representations obtained from the character composition part of the model reveals that the model is able to encode, from characters only, both semantic and orthographic information.

Section 1 – Introduction

While NLMs have been shown to outperform count-based n-gram language models (Mikolov et al. 2011), they are blind to subword information (e.g. morphemes). For example, they do not know, a priori, that eventful, eventfully, uneventful, and uneventfully should have structurally related embeddings in the vector space. Embeddings of rare words can thus be poorly estimated, leading to high perplexities for rare words (and words surrounding them). This is especially problematic in morphologically rich languages with long-tailed frequency distributions or domains with dynamic vocabularies (e.g. social media).

Model


absurdity
is
recognized
Cross entropy loss between next word and prediction
Softmax output to obtain distribution over next word
Long short-term memory network
Highway network
Max-over-time pooling layer
max{·}
Convolution layer with multiple filters of different widths
Concatenation of character embeddings
moment
the
absurdity
is
recognized

CNNs

A full treatment of CNNs would take an hour by itself.

CNNs apply a sliding filter to the input to get an output of similar shape.

The notable property of CNNs is that this filter naturally understands spacial relationships within the input space.

In the case of NLP, the spacial relationship we care about is the arrangements of letters.

CNNs are more biased towards spacial relationships than RNNs are.

Highway Networks

A generalization of residual connections.

Given input x and dense layer D, the output of a highway connection is

$$z := T(x)D(x) + V(x)x$$

where $T, V : \mathbb{R}^n \to [0,1]$. Commonly we set $V(x) = 1 - T(x)$ and $T(x) = \sigma(Wx + b)$.

Residual Connections

Residual connections are the special case where $T = V = 1$.

Results

The learned representations of OOV words (computeraided, misinformed) are positioned near words with the same part-of-speech. The model is also able to correct for incorrect/non-standard spelling (loooooook), indicating potential applications for text normalization in noisy domains.

Coverage

Deep contextualized word representations

AKA ELMO.

This is the first paper we've covered that I hadn't read before starting this reading group so bear with me.

Tensorflow Hub

You can play around with this model yourself. Tensorflow Hub makes it easy to import pre-trained models inside of Tensorflow and use them as layers.
Check out `https://tfhub.dev/google/elmo/2`.

Abstract

We introduce a new type of deep contextualized word representation that models both (1) complex characteristics of word use (e.g., syntax and semantics), and (2) how these uses vary across linguistic contexts (i.e., to model polysemy). Our word vectors are learned functions of the internal states of a deep bidirectional language model (biLM), which is pretrained on a large text corpus. We show that these representations can be easily added to existing models and significantly improve the state of the art across six challenging NLP problems, including question answering, textual entailment and sentiment analysis. We also present an analysis showing that exposing the deep internals of the pre-trained network is crucial, allowing downstream models to mix different types of semi-supervision signals.

Contextualized Word Vectors

The authors claim the problem with normal word vectors is that they are static.

Each word maps to a fixed vector irrespective of context.

ELMO produces contextualized word vectors, which means that the continuous representation of the word depends on the surrounding context. This accounts for polysemy.

Polysemy

Polysemy is the property that a word can mean different things in different contexts.
For example ‘a blue ball’ vs ‘feeling blue’.

ELMO at its core

Hey ELMo, what's the embedding of the word "stick"?
There are multiple possible embeddings! Use it in a sentence.
Oh, okay. Here: "Let's stick to improvisation in this skit"
Oh in that case, the embedding is: -0.02, -0.16, 0.12, -0.1 ....etc

Exercise

Exercise: what are the pros and cons of this?

Exercise

Exercise: what are the pros and cons of this?

Pros:

- word embeddings can take context into account
- word embeddings for words with lots of polysemy are not ‘overloaded’

Cons:

- you need to run a biLM on your input to get the word vectors
- running ahead of time + storing the result takes a lot of space

Deep Word Vectors

ELMO produces these vectors by a bidirectional language model, combining the lower layers of the model to produce the word representation.

This is what is meant by a deep language model.

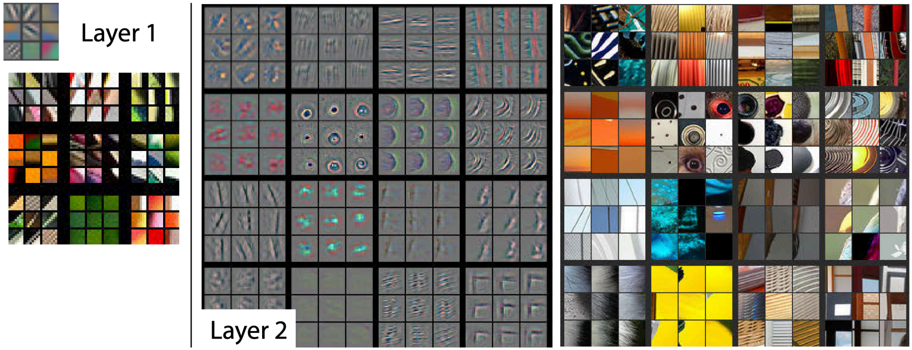
I see a similarity/inspiration to CNNs.

The lower layers of CNNs compute lower-level features like edges or shapes, while the higher layers compute high-level features like objects or faces.

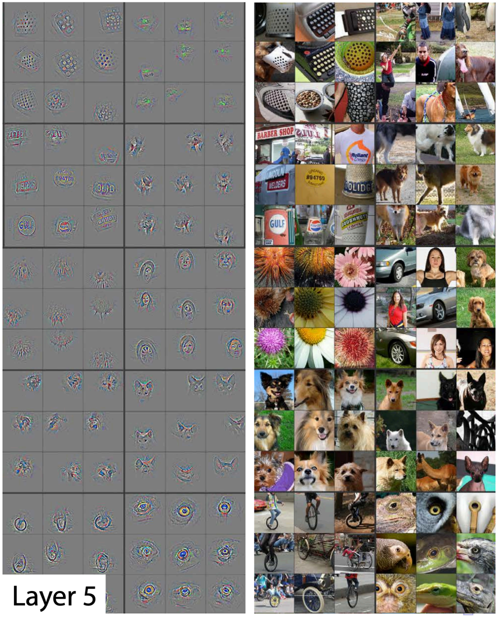
Likewise the lower layers of LMs compute low-level language features; upper layers of LMs compute high-level language features.

You need both to make better word embeddings, according to this paper.

Visualizing CNN Layers



Visualizing CNN Layers



Section 1 – Introduction

Our representations differ from traditional word type embeddings in that each token is assigned a representation that is a function of the entire input sentence. We use vectors derived from a bidirectional LSTM that is trained with a coupled language model (LM) objective on a large text corpus. For this reason, we call them ELMo (Embeddings from Language Models) representations. Unlike previous approaches for learning contextualized word vectors (Peters et al., 2017; McCann et al., 2017), ELMo representations are deep, in the sense that they are a function of all of the internal layers of the biLM. More specifically, we learn a linear combination of the vectors stacked above each input word for each end task, which markedly improves performance over just using the top LSTM layer.

Section 1 – Introduction

Combining the internal states in this manner allows for very rich word representations. Using intrinsic evaluations, we show that the higher-level LSTM states capture context-dependent aspects of word meaning (e.g., they can be used without modification to perform well on supervised word sense disambiguation tasks) while lowerlevel states model aspects of syntax (e.g., they can be used to do part-of-speech tagging). Simultaneously exposing all of these signals is highly beneficial, allowing the learned models select the types of semi-supervision that are most useful for each end task.

Section 1 – Introduction

Extensive experiments demonstrate that ELMo representations work extremely well in practice. We first show that they can be easily added to existing models for six diverse and challenging language understanding problems, including textual entailment, question answering and sentiment analysis. The addition of ELMo representations alone significantly improves the state of the art in every case, including up to 20% relative error reductions. For tasks where direct comparisons are possible, ELMo outperforms CoVe (McCann et al., 2017), which computes contextualized representations using a neural machine translation encoder. Finally, an analysis of both ELMo and CoVe reveals that deep representations outperform those derived from just the top layer of an LSTM. Our trained models and code are publicly available, and we expect that ELMo will provide similar gains for many other NLP problems.

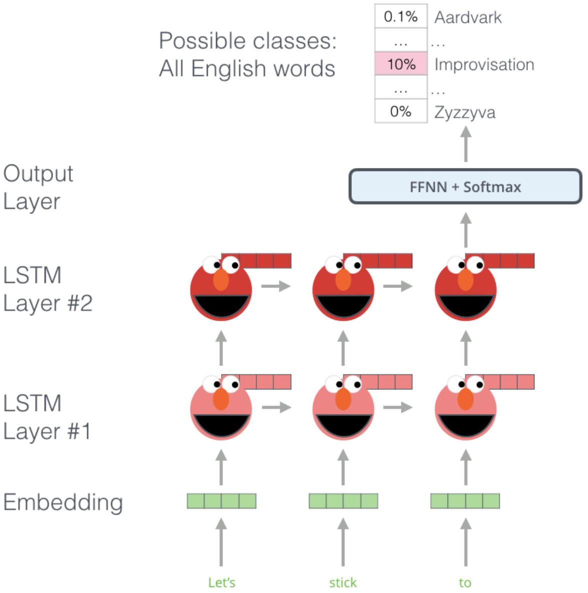
Section 2 – Related Work

Other recent work has also focused on learning context-dependent representations. context2vec (Melamud et al., 2016) uses a bidirectional Long Short Term Memory (LSTM; Hochreiter and Schmidhuber, 1997) to encode the context around a pivot word.

...

Previous work has also shown that different layers of deep biRNNs encode different types of information. For example, introducing multi-task syntactic supervision (e.g., part-of-speech tags) at the lower levels of a deep LSTM can improve overall performance of higher level tasks such as dependency parsing (Hashimoto et al., 2017) or CCG super tagging (Søgaard and Goldberg, 2016). In an RNN-based encoder-decoder machine translation system, Belinkov et al. (2017) showed that the representations learned at the first layer in a 2- layer LSTM encoder are better at predicting POS tags then second layer.

Model



Possible classes:
All English words
0.1% Aardvark
... ...
10% Improvisation
... ...
0% Zyzzyva
Output Layer
FFNN + Softmax
LSTM Layer #2
LSTM Layer #1
Embedding
Let's
stick
to

Getting embeddings from ELMO

The embedding of the tth word is the weighted sum of each layer's output at time t.

For example: Suppose there are L layers. If the output of layer i at word t is x_i^t, then the output embedding for the ith word is

$$\gamma \sum_{i=i}^{L} s_i x_i^t$$

γ and s_i are trainable in the model that uses ELMO weights, while the rest of the weights in the model are fixed.

Other Embeddings

So far we have seen:

- *A Neural Probabilistic Language Model*
- word2vec
- ELMO

I think today the two most common word embeddings are these, which we might not cover:

- GloVe
- fastText

These are fixed embeddings like word2vec and unlike ELMO, but they perform very wel without ELMO's performance considerations.

Coverage

Idea

Combines:

- ELMO
- OpenAI-style transformer
- Bidirectional transformers via bidirectional masking
- ULMfit (finetuning LMs)

Homework

Practice: Read the BERT paper. `https://arxiv.org/abs/1810.04805`

Exercises:

- Familiarize yourself with CNNs
- Use the TensorflowHub version of ELMO to create a NLP model for a task of your choice, compare it with other embeddings
- Implement ELMO
- Try to reproduce the results from *Neural Machine Translation by Jointly Learning to Align and Translate*, either on the paper's dataset or try generalizing the results to your own dataset