CIS 522 - University Of Pennsylvania, Spring 2020

Lecture 12: RNNs and LSTMs

27 February 2020

Lecturer: Konrad Kording *Scribe: Kushagra Goel, Brandon Lin, Yonah Mann*

1 Recurrent Neural Networks for NLP

1.1 Introduction

In this lecture, we introduce **recurrent neural networks**, which are networks widely used in natural language processing and similar domains, where the input is of variable length. We also explore an extension of RNNs, known as **LSTMs**; both network will give a good sense of how problems in NLP are tackled.

2 Problems with Feedforward Networks in NLP

The main issue with standard feedforward networks is their inability to generalize to variable-length input. A feedforward network could be trained to classify 4-word sentences (such as "I love deep learning.") but would have no way of classifying even the slightest modifications to this sentence (for example, "I really love deep learning."). One could attempt to use some procedure that trims unnecessary words from a sentence so that it matches the input size of the network, but even such a procedure would be difficult to create and dropping words from a sentence may cause it to lose meaning.

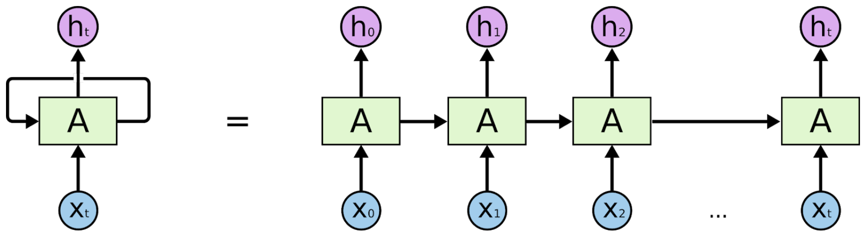
So suppose we have a sentence S with words $[x_1, x_2, \dots, x_m]$. Instead of having a network take in m inputs at a time, what if we ran the network m times, each time on each word of the sentence?

This is precisely what recurrent neural networks do. Essentially, recurrent neural network emulate the action of reading a sentence left to right and runs itself on each word in sequential order. Ideally, what the network should do is learn about previous words in a sentence, accumulate the weights it learns and use those in subsequent words.

More precisely, we assume that *the way the network "thinks" should not be affected by time*; in other words, that we can use the same network structure to iterate through the words instead of using a new network each time. This is the **prior** that we assume on our data when we use an RNN, that perception should remain relatively invariant across a sentence.

3 Recurrent Neural Networks

3.1 Structure of Recurrent Neural Networks



Whenever one step of the recurrent neural network forward pass occurs, the network generates what is known as a "hidden state". We denote the hidden state at step t as $\boldsymbol{h}^{(t)} \in \mathbb{R}^h$. The function the RNN uses can be represented with five tensors:

- $\mathbf{W} \in \mathbb{R}^{h \times h}$, a linear layer transforming the previous hidden state
- $\mathbf{U} \in \mathbb{R}^{h \times d}$, a linear layer transforming the current input word
- $\mathbf{V} \in \mathbb{R}^{C \times h}$, used to transform the new activated hidden state.
- $\boldsymbol{b} \in \mathbb{R}^h, \boldsymbol{c} \in \mathbb{R}^C$, bias vectors

Now, we can represent a single step of the RNN as follows:

1. On step t, the network calculates
$$\boldsymbol{h}^{(t)} = \tanh(\mathbf{W}\boldsymbol{h}^{(t-1)} + \mathbf{U}\boldsymbol{x}^{(t)} + \boldsymbol{b})$$
where $\boldsymbol{x}^{(t)} \in \mathbb{R}^d$ is a vector embedding of the t^{th} word.
2. Then, it computes the intermediate output
$$\boldsymbol{o}^{(t)} = \mathbf{V}\boldsymbol{h}^{(t)} + \boldsymbol{c}$$
3. Finally, it computes a softmax on the intermediate output to obtain a vector of probabilities for that time step:
$$\boldsymbol{y}^{(t)} = \text{softmax}(\boldsymbol{o}^{(t)})$$

The exact unrolling scheme may differ depending on what exactly your learning problem is, but this serves as a general enough framework for how RNNs operate.

3.2 Backpropagation Through Time

Let us now consider how backpropagation through time works. Consider the following notation [1]

$$\begin{aligned}
\boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)} \\
\boldsymbol{h}^{(t)} &= \tanh\left(\boldsymbol{a}^{(t)}\right) \\
\boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)} \\
\hat{\boldsymbol{y}}^{(t)} &= \text{softmax}\left(\boldsymbol{o}^{(t)}\right)
\end{aligned}$$

Compute the BPTT gradients recursively.

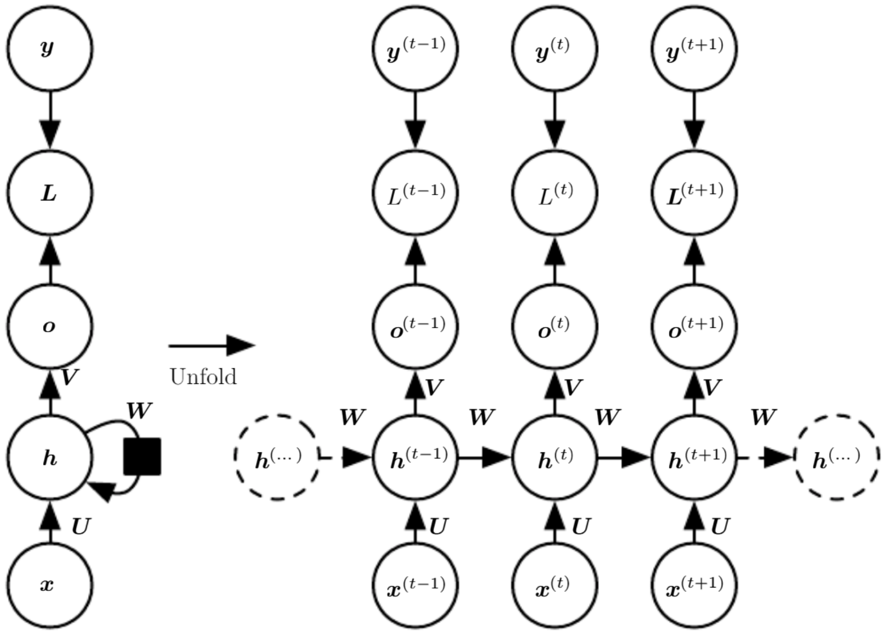


Figure 1: Unrolled RNN

- Begin the recursion with nodes immediately preceding final loss.
- Assuming that the outputs $\boldsymbol{o}^{(t)}$ are used as arguments to softmax to obtain vector $\widehat{y}$ of probabilities over the output with loss being the negative log-likelihood of the true target $y^{(t)}$ given the input so far, the gradient on the outputs at time step t is :

$$(\nabla_{\boldsymbol{o}^{(t)}} L)_i = \frac{\partial L}{\partial o_i^{(t)}} = \frac{\partial L}{\partial L^{(t)}} \frac{\partial L^{(t)}}{\partial o_i^{(t)}} = \hat{y}_i^{(t)} - \mathbf{1}_{i,y^{(t)}}$$

- And since at final time step τ, $\boldsymbol{h}^{(\tau)}$ has only $\boldsymbol{o}^{(t)}$ as a descendent,

$$\nabla_{\boldsymbol{h}^{(\tau)}} L = \boldsymbol{V}^\top \nabla_{\boldsymbol{o}^{(\tau)}} L$$

- Iterating backwards in time to backpropagate gradients, we have

$$\begin{aligned}\nabla_{\boldsymbol{h}^{(t)}} L &= \left(\frac{\partial \boldsymbol{h}^{(t+1)}}{\partial \boldsymbol{h}^{(t)}}\right)^\top (\nabla_{\boldsymbol{h}^{(t+1)}} L) + \left(\frac{\partial \boldsymbol{o}^{(t)}}{\partial \boldsymbol{h}^{(t)}}\right)^\top (\nabla_{\boldsymbol{o}^{(t)}} L) \\ &= \boldsymbol{W}^\top (\nabla_{\boldsymbol{h}^{(t+1)}} L) \operatorname{diag}\left(1 - \left(\boldsymbol{h}^{(t+1)}\right)^2\right) + \boldsymbol{V}^\top (\nabla_{\boldsymbol{o}^{(t)}} L)\end{aligned}$$

- Once gradients on internal nodes of the computational graph are obtained, we can obtain gradients on the parameters :

$$
\begin{aligned}
\nabla_{\boldsymbol{c}} L &= \sum_t \left(\frac{\partial \boldsymbol{o}^{(t)}}{\partial \boldsymbol{c}} \right)^\top \nabla_{\boldsymbol{o}^{(t)}} L = \sum_t \nabla_{\boldsymbol{o}^{(t)}} L \\
\nabla_{\boldsymbol{b}} L &= \sum_t \left(\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{b}^{(t)}} \right)^\top \nabla_{\boldsymbol{h}^{(t)}} L = \sum_t \operatorname{diag}\left(1 - \left(\boldsymbol{h}^{(t)}\right)^2\right) \nabla_{\boldsymbol{h}^{(t)}} L \\
\nabla_{\boldsymbol{V}} L &= \sum_t \sum_i \left(\frac{\partial L}{\partial o_i^{(t)}} \right) \nabla_{\boldsymbol{V}} o_i^{(t)} = \sum_t \left(\nabla_{\boldsymbol{o}^{(t)}} L\right) \boldsymbol{h}^{(t)\top} \\
\nabla_{\boldsymbol{W}} L &= \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}} \right) \nabla_{\boldsymbol{W}^{(t)}} h_i^{(t)} \\
&= \sum_t \operatorname{diag}\left(1 - \left(\boldsymbol{h}^{(t)}\right)^2\right) \left(\nabla_{\boldsymbol{h}^{(t)}} L\right) \boldsymbol{h}^{(t-1)\top} \\
\nabla_{\boldsymbol{U}} L &= \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}} \right) \nabla_{U^{(t)}} h_i^{(t)} \\
&= \sum_t \operatorname{diag}\left(1 - \left(\boldsymbol{h}^{(t)}\right)^2\right) \left(\nabla_{\boldsymbol{h}^{(t)}} L\right) \boldsymbol{x}^{(t)\top}
\end{aligned}
$$

3.3 Pros and cons of RNNs

RNNs solve some of the aforementioned issues with feedforward networks:

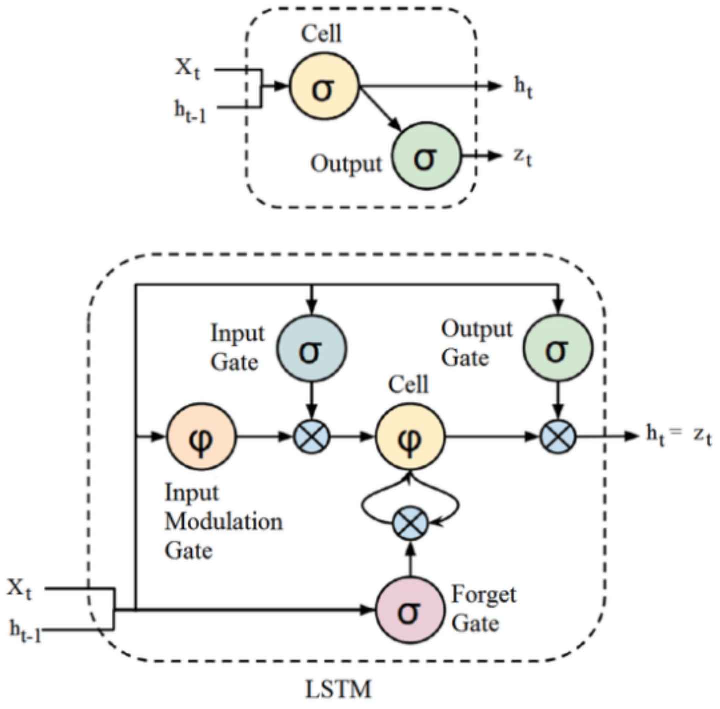
1. They model sequential data so that each state is dependant on previous states.
2. They can handle variable length inputs.

But, RNNs have one major problem – they don't actually remember very much. In models with many hidden states, a RNN will suffer from the vanishing gradient problem. By the time you reach a later hidden state, odds are the RNN has forgotten everything form the first hidden states. To fix this, we turn to the LSTM.

4 LSTM

4.1 Structure of LSTM

A Long Short-Term Memory (LSTM) network is a modified RNN with an emphasis on capturing both short and long term memory (i.e to solve the vanishing gradient problem in RNNs). To do this, three gates are introduced which together make up a single LSTM cell.



4.1.1 Input Gate

The input gate determines which information should enter the cell state. It does this via two gates: the input gate itself which decides whether information should enter and the input modulation gate which adds non-linearity and makes the information zero-mean via the tanh activation function.

Input gate:

$$\mathbf{i}_t = \sigma(\mathbf{W}_i[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i)$$

Input modulation gate:

$$\tilde{\mathbf{c}}_t = \tanh(\mathbf{W}_c[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_c)$$

4.1.2 Forget Gate

The forget gate decides which old data to discard.

Forget gate:

$$\mathbf{f}_t = \sigma(\mathbf{W}_f[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f)$$

4.1.3 Cell state

The long-term memory is called the cell state. It takes in the outputs of the input and forget gates and thus stores old information from previous hidden states.

Cell state:

$$\mathbf{c}_t = \mathbf{f}_t \circ \mathbf{c}_{t-1} + \mathbf{i}_t \circ \tilde{\mathbf{c}}_t$$

4.1.4 Output gate

The output gate decides what output to generate from the current cell state.

Output gate:

$$\mathbf{o}_t = \sigma(\mathbf{W}_o[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o)$$

Hidden state:

$$\mathbf{h}_t = \mathbf{o}_t \circ \tanh(\mathbf{c}_t)$$

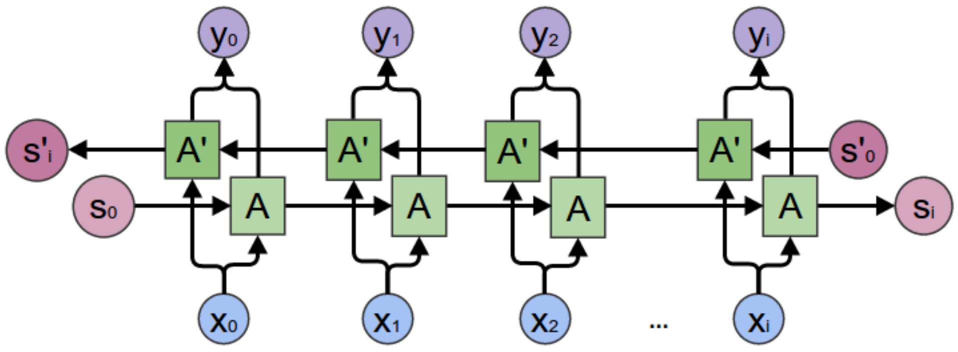
4.2 Success of LSTMs

In the last few years, there have been incredible success applying RNNs to a variety of problems: speech recognition, language modeling, translation, image captioning... The list goes on. As you might expect, the sequence regime of operation is much more powerful compared to fixed networks that are doomed from the get-go by a fixed number of computational steps, and hence also much more appealing for those of us who aspire to build more intelligent systems. Moreover, RNNs combine the input vector with their state vector with a fixed (but learned) function to produce a new state vector. This can in programming terms be interpreted as running a fixed program with certain inputs and some internal variables. Viewed this way, RNNs essentially describe programs. [3] Essentially all of these are achieved using LSTMs. They really work a lot better for most tasks! LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn! [2]

We strongly advise students to read the blogs "The Unreasonable Effectiveness of Recurrent Neural Networks" and "Understanding LSTM Networks".

4.3 Bi-LSTM

The bi-directional LSTM is just two LSTMs placed together. In the LSTM that runs forwards, you preserve information from the past and in the LSTM that runs backward, you preserve information from the future. Bi-LSTMs work well in practice, but no one really understands why they are better than normal LSTMs.



5 InferSent

In the InferSent paper[4], the authors study the task of learning universal representations of sentences, i.e., a sentence encoder model that is trained on a large corpus and subsequently transferred to other tasks. Unlike in computer vision, where convolutional neural networks are predominant, there are multiple ways to encode a sentence using neural networks. Hence, they investigate the impact of the sentence encoding architecture on representational transferability, and compare convolutional, recurrent and even simpler word composition schemes. The experiments show that an encoder based on a bi-directional LSTM architecture with max pooling, trained on the Stanford Natural Language Inference (SNLI) dataset (Bowman et al., 2015), yields state-of-the-art sentence embeddings compared to all existing alternative unsupervised approaches like SkipThought or FastSent, while being much faster to train.

InferSent is a sentence embeddings method that provides semantic representations for English sentences. It is trained on natural language inference data and generalizes well to many different tasks. InferSent is essentially a BiLSTM encoder with max pooling as described below:
For a sequence of T words $\{w_t\}_{t=1,\ldots,T}$ a bidirectional LSTM computes a set of T vectors $\{h_t\}_t$. For $t \in [1, \ldots, T]$, h_t is the concatenation of a forward LSTM and a backward LSTM that read the sentences in two opposite directions:

$$\begin{aligned}
\overrightarrow{h_t} &= \overrightarrow{\text{LSTM}}_t\,(w_1, \ldots, w_T) \\
\overleftarrow{h_t} &= \overleftarrow{\text{LSTM}}_t\,(w_1, \ldots, w_T) \\
h_t &= \left[\overrightarrow{h_t}, \overleftarrow{h}_t\right]
\end{aligned}$$

One can choose from the two ways of combining the varying number of $\{h+\}t$ t to form a fixed-size vector, either by selecting the maximum value over each dimension of the hidden units (max pooling, preferred) or by considering the average of the representations (mean pooling).
Considering the following tasks and their datasets:

- Binary and Multi-Class Classification
  The authors use a set of binary classification tasks (see Figure 2) that covers various types of sentence classification, including sentiment analysis (MR, SST), question-type (TREC), product reviews (CR), subjectivity/objectivity (SUBJ) and opinion polarity (MPQA). The authors generate sentence vectors and train a logistic regression on top. A linear classifier requires fewer parameters than an MLP and is thus suitable for small datasets, where transfer learning is especially well-suited. The authors tune the L2 penalty of the logistic regression with grid-search on the validation set.

name	N	task	C	examples
MR	11k	sentiment (movies)	2	”Too slow for a younger crowd , too shallow for an older one.” (neg)
CR	4k	product reviews	2	”We tried it out christmas night and it worked great .” (pos)
SUBJ	10k	subjectivity/objectivity	2	”A movie that doesn’t aim too high , but doesn’t need to.” (subj)
MPQA	11k	opinion polarity	2	”don’t want”; ”would like to tell”; (neg, pos)
TREC	6k	question-type	6	”What are the twin cities ?” (LOC:city)
SST	70k	sentiment (movies)	2	”Audrey Tautou has a knack for picking roles that magnify her [..]” (pos)

Figure 2: Classification tasks. C is the number of class and N is the number of samples

- Entailment and Semantic Relatedness
  The authors evaluate on the SICK dataset for both entailment (SICK-E) and semantic relatedness (SICK-R). We use the same matching methods as in SNLI and learn a Logistic Regression on top of the joint representation. For semantic relatedness evaluation, The authors follow the approach of (Tai et al., 2015) and learn to predict the probability distribution of relatedness scores. The authors report Pearson correlation.

- STS14 - Semantic Textual Similarity
  While semantic relatedness is supervised in the case of SICK-R, The authors also evaluate our embeddings on the 6 unsupervised SemEval tasks of STS14 (Agirre et al., 2014). This dataset includes subsets of news articles, forum discussions, image descriptions and headlines from news articles containing pairs of sentences (lower-cased), labeled with a similarity score between 0 and 5. These tasks evaluate how the cosine distance between two sentences correlate with a human-labeled similarity score through Pearson and Spearman correlations.

- Paraphrase Detection
  The Microsoft Research Paraphrase Corpus is composed of pairs of sentences which have been extracted from news sources on the Web. Sentence pairs have been human-annotated according to whether they capture a paraphrase/semantic equivalence relationship. The authors use the same approach as with SICK-E, except that our classifier has only 2 classes.

- Caption-Image retrieval
  The caption-image retrieval task evaluates joint image and language feature models (Hodosh et al., 2013; Lin et al., 2014). The goal is either to rank a large collection of images by their relevance with respect to a given query caption (Image Retrieval), or ranking captions by their relevance for a given query image (Caption Retrieval). The authors use a pairwise rankingloss $\mathcal{L}_{\text{cir}}(x, y)$:

$$\begin{gathered}\sum_y \sum_k \max\left(0, \alpha - s(Vy, Ux) + s\left(Vy, Ux_k\right)\right) + \\ \sum_x \sum_{k'} \max\left(0, \alpha - s(Ux, Vy) + s\left(Ux, Vy_{k'}\right)\right)\end{gathered}$$

  where (x, y) consists of an image y with one of its associated captions x, $(y_k)_k$ and $(y_{k'})_{k'}$ are negative examples of the ranking loss, α is the margin and s corresponds to the cosine similarity. U and V are learned linear transformations that project the caption x and the image y to the same embedding space.

Model	MR	CR	SUBJ	MPQA	SST	TREC	MRPC	SICK-R	SICK-E	STS14
Unsupervised representation training (unordered sentences)										
Unigram-TFIDF	73.7	**79.2**	90.3	82.4	-	85.0	73.6/81.7	-	-	.58/.57
ParagraphVec (DBOW)	60.2	66.9	76.3	70.7	-	59.4	72.9/81.1	-	-	.42/.43
SDAE	74.6	78.0	90.8	86.9	-	78.4	**73.7**/80.7	-	-	.37/.38
SIF (GloVe + WR)	-	-	-	-	82.2	-	-	-	**84.6**	**.69**/ -
word2vec BOW†	77.7	79.8	90.9	88.3	79.7	83.6	72.5/81.4	0.803	78.7	.65/.64
fastText BOW†	78.3	81.0	**92.4**	87.8	**81.9**	84.8	**73.9/82.0**	0.815	78.3	.63/.62
GloVe BOW†	**78.7**	78.5	91.6	87.6	79.8	83.6	72.1/80.9	0.800	78.6	.54/.56
GloVe Positional Encoding†	78.3	77.4	91.1	87.1	80.6	83.3	72.5/81.2	0.799	77.9	.51/.54
BiLSTM-Max (untrained)†	77.5	**81.3**	89.6	**88.7**	80.7	**85.8**	73.2/81.6	**0.860**	83.4	.39/.48
Unsupervised representation training (ordered sentences)										
FastSent	70.8	78.4	88.7	80.6	-	76.8	72.2/80.3	-	-	**.63/.64**
FastSent+AE	71.8	76.7	88.8	81.5	-	80.4	71.2/79.1	-	-	.62/.62
SkipThought	76.5	80.1	93.6	87.1	82.0	**92.2**	**73.0/82.0**	**0.858**	82.3	.29/.35
SkipThought-LN	**79.4**	**83.1**	**93.7**	**89.3**	82.9	88.4	-	**0.858**	79.5	.44/.45
Supervised representation training										
CaptionRep (bow)	61.9	69.3	77.4	70.8	-	72.2	73.6/81.9	-	-	.46/.42
DictRep (bow)	76.7	78.7	90.7	87.2	-	81.0	68.4/76.8	-	-	**.67/.70**
NMT En-to-Fr	64.7	70.1	84.9	81.5	-	82.8	69.1/77.1	-		.43/.42
Paragram-phrase	-	-	-	-	79.7	-	-	0.849	83.1	**.71**/ -
BiLSTM-Max (on SST)†	(*)	83.7	90.2	89.5	(*)	86.0	72.7/80.9	0.863	83.1	.55/.54
BiLSTM-Max (on SNLI)†	79.9	84.6	92.1	**89.8**	83.3	**88.7**	75.1/82.3	**0.885**	**86.3**	.68/.65
BiLSTM-Max (on AllNLI)†	**81.1**	**86.3**	**92.4**	**90.2**	**84.6**	88.2	**76.2/83.1**	**0.884**	**86.3**	**.70/.67**
Supervised methods (directly trained for each task – no transfer)										
Naive Bayes - SVM	79.4	81.8	93.2	86.3	83.1	-	-	-	-	-
AdaSent	83.1	86.3	95.5	93.3	-	92.4	-	-	-	-
TF-KLD	-	-	-	-	-	-	80.4/85.9	-	-	-
Illinois-LH	-	-	-	-	-	-	-	-	84.5	-
Dependency Tree-LSTM	-	-	-	-	-	-	-	0.868	-	-

Figure 3: Transfer test results for various architectures trained in different ways. Underlined are best results for transfer learning approaches, in bold are best results among the models trained in the same way. † indicates methods that The authors trained, other transfer models have been extracted from (Hill et al., 2016). For best published supervised methods (no transfer), The authors consider AdaSent (Zhao et al., 2015), TF-KLD (Ji and Eisenstein, 2013), Tree-LSTM (Tai et al., 2015) and Illinois-LH system (Lai and Hockenmaier, 2014). (*) Our model trained on SST obtained 83.4 for MR and 86.0 for SST (MR and SST come from the same source), which The authors do not put in the tables for fair comparison with transfer methods.

6 Embedding from Language Models ELMo

6.1 Motivation

In this paper[5], authors introduce a new type of deep contextualized word representation that models both

- Complex characteristics of word use (e.g., syntax and semantics)
- How these uses vary across linguistic contexts (i.e., to model polysemy)

The word vectors are learned functions of the internal states of a deep bidirectional language model (biLM), which is pretrained on a large text corpus. The authors 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.
The learned representations differ from traditional word type embeddings in that each token is assigned a representation that is a function of the entire input sentence. The 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, they are called 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.

6.2 How ELMo works

ELMo is a task specific combination of the intermediate layer representations in the biLM. For each token t_k a L-layer biLM computes a set of $2L + 1$ representations

$$\begin{aligned} R_k &= \left\{ \mathbf{x}_k^{LM}, \overrightarrow{\mathbf{h}}_{k,j}^{LM}, \overleftarrow{\mathbf{h}}_{k,j}^{LM} | j = 1, \ldots, L \right\} \\ &= \left\{ \mathbf{h}_{k,j}^{LM} | j = 0, \ldots, L \right\} \end{aligned}$$

where $\mathbf{h}_{k,0}^{LM}$ is the token layer and $\mathbf{h}_{k,j}^{LM} = \left[\overrightarrow{\mathbf{h}}_{k,j}^{LM}; \overleftarrow{\mathbf{h}}_{k,j}^{LM} \right]$, for each biLSTM layer. For inclusion in a downstream model, ELMo collapses all layers in R into a single vector, $\mathbf{ELMo}_k = E\left(R_k; \mathbf{\Theta}_e\right)$. In the simplest case, ELMo just selects the top layer, $E\left(R_k\right) = \mathbf{h}_{k,L}^{LM}$, more generally, ELMo computes a task specific weighting of all biLM layers:

$$\mathbf{ELMo}_k^{task} = E\left(R_k; \Theta^{task}\right) = \gamma^{task} \sum_{j=0}^{L} s_j^{task} \mathbf{h}_{k,j}^{LM}$$

where $\mathbf{s}^{task}$ are softmax-normalized weights and the scalar parameter γ^{task} allows the task model to scale the entire ELMo vector.

6.3 Using ELMo

Given a pre-trained biLM and a supervised architecture for a target NLP task, it is a simple process to use the biLM to improve the task model. We simply run the biLM and record all of the layer representations for each word. Then, we let the end task model learn a linear combination of these representations, as described below.
To add ELMo to the supervised model, we first freeze the weights of the biLM and then concatenate the ELMo vector $\mathbf{ELMo}_k^{task}$ with $\mathbf{X}_k$ and pass the ELMo enhanced representation $\left[\mathbf{x}_k; \mathbf{ELMo}_k^{task}\right]$ into the task RNN. For some tasks (e.g., SNLI, SQuAD), further improvements are observed by also including ELMo at the output of the task RNN by introducing another set of output specific linear weights and replacing Λk with $\left[\mathbf{h}_k; \mathbf{ELMo}_k^{task}\right]$. As the remainder of the supervised model

remains unchanged, these additions can happen within the context of more complex neural models.

References

[1] Computing the gradient in an rnn. https://cedar.buffalo.edu/ srihari/CSE676/10.2.2%20RNN-Gradients.pdf.

[2] Understanding lstm networks. http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[3] The unreasonable effectiveness of recurrent neural networks. http://karpathy.github.io/2015/05/21/rnn-effectiveness/.

[4] Alexis Conneau, Douwe Kiela, Holger Schwenk, Loic Barrault, and Antoine Bordes. Supervised learning of universal sentence representations from natural language inference data, 2017.

[5] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations, 2018.