Neural models of language
Neural networks

- Introduction
- Word-level representations
- Neural language models
- Recurrent neural networks
- Sequence-to-sequence modelling
- Some recent developments

With material from Phil Blunsom, Piotr Mirowski, Adam Kalai, and James Zou
Artificial neural networks

- **Artificial neural networks (ANNs)** were (kind of) inspired from neurobiology (Widrow and Hoff, 1960).
- Each unit has many inputs (**dendrites**), one output (**axon**).
- The **nucleus** fires (sends an electric signal along the axon) given input from other neurons.
- ‘Learning’ occurs at the **synapses** that connect neurons, either by amplifying or attenuating signals.
Perceptron: an artificial neuron

- Each neuron calculates a **weighted sum** of its inputs and compares this to a threshold, $\tau$. If the sum exceeds the threshold, the neuron fires.
- Inputs $a_i$ are activations from adjacent neurons, each weighted by a parameter $w_i$.

$$x = \sum_{i=1}^{M} w_i a_i$$

![McCulloch-Pitts model diagram](image)
Perceptron output

- Perceptron output is determined by activation functions, $g()$, which can be non-linear functions of weighted input.
- Popular activation functions include \texttt{tanh} and the \texttt{sigmoid}:
  \[ g(x) = \sigma(x) = \frac{1}{1 + e^{\rho x}} \]
  - The sigmoid’s derivative is the easily computable $\sigma' = \sigma \cdot (1 - \sigma)$
Rectified Linear Units (ReLUs)

• Since 2011, the ReLU \( S = g(x) = \max(0, x) \) has become more popular.
  • More biologically plausible, sparse activation, limited (vanishing or exploding) gradient problems, efficient computation.

• A smooth approximation is the softplus \( \log(1 + e^x) \), which has a simple derivative \( 1/(1 + e^{-x}) \)

• Why do we care about the derivatives?

Perceptron learning

• Weights are adjusted in **proportion to the error** (i.e., the **difference** between the desired, $y$, and the actual output, $S$).

• The **derivative** $g'$ allows us to assign blame proportionally.

• Given a small learning rate, $\alpha$ (e.g., 0.05), we can repeatedly adjust each of the weight parameters by

$$w_j := w_j + \alpha \cdot \sum_{i=1}^{R} Err_i \cdot g'(x_i) \cdot a_j[i]$$

where $Err_i = (y_i - S_i)$, among $R$ training examples.

Assumes mean-square error objective
Threshold perceptrons and XOR

• Some relatively simple logical functions cannot be learned by threshold perceptrons (since they are not linearly separable).
Artificial neural networks

• Complex functions can be represented by layers of perceptrons (multi-layer perceptrons, MLPs).

• Input are passed to the input layer.
• Activations are propagated through hidden layers to the output layer.
• MLPs are quite robust to noise, and are trained specifically to reduce error.
Deep

It's a cat.

'hidden' representations are learned here
Can we find hidden patterns in words?
Words

- Given a corpus with \( D \) (e.g., \( = 100K \)) unique words, the **classical approach** is to uniquely assign **each word** with an index in \( D \)-dimensional vectors (‘one-hot’ representation).

<table>
<thead>
<tr>
<th>lugubrious</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>. . .</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>...</th>
<th>0</th>
</tr>
</thead>
</table>

- Classic **word-feature representation** assigns **features** to each index in a much denser vector.

<table>
<thead>
<tr>
<th>1</th>
<th>0.8</th>
<th>2.5</th>
<th>0.81</th>
<th>...</th>
<th>99</th>
</tr>
</thead>
</table>

  \( d \ll D \)

- Can we learn a dense representation? What will it give us?
Learning word semantics

"You shall know a word by the company it keeps."
— J.R. Firth (1957)

\[
P(w_t = \text{lugubrious}|w_{t-1} = \text{feeling}, w_{t-2} = \text{been}, \ldots)\]

been  feeling  lugubrious  all  day
felt  a  lugubrious  sadness  in

...  

Here, we’re predicting the center word given the context. This is called the ‘continuous bag of words’ (CBOW) model.

https://code.google.com/p/word2vec/
Continuous bag of words (1 word context)

\[ W_I \in (D \times H) \]
\[ W_0 \in (H \times D) \]

\[ D = 100K, 0, 0, 0, \ldots, 1, \ldots, 0 \]
\[ [0,0,0, \ldots, 1, \ldots, 0] \]

Note: we have two vector representations of each word:

\[ \nu_w = x^T W_I (w^{th} \text{ row of } W_I) \]
\[ V_w = W_0^T y (w^{th} \text{ col of } W_0) \]

\[ 'softmax' \: P(w_o|w_i) = \frac{\exp(V_{w_o}^T \nu_{w_i})}{\sum_w \exp(V_w^T \nu_{w_i})} \]

Where

\[ \nu_w \] is the ‘input’ vector for word \( w \),
\[ V_w \] is the ‘output’ vector for word \( w \),

CSC401/2511 – Spring 2020
Continuous bag of words \((C \text{ words context})\)

- If we want to use more context, \(C\), we need to change the network architecture somewhat.
  - Each input word will produce one of \(C\) embeddings
  - We just need to add an intermediate layer, usually this just averages the embeddings.

```
been
feeling
lugubrious
all
felt
a
lugubrious
sadness
... 
```
Skip-grams

- **Skip-grams** invert the task – we predict context words given the current word.

- According to Mikolov, **Skip-gram**: works well with small amounts of training data, represents rare words.

**CBOW**: several times faster to train, slightly better accuracy for frequent words

Actually doing the learning

• Given $H$-dimensional embeddings, and $V$ word types, our parameters, $\theta$, are:

$$\theta = \begin{bmatrix} \nu_a \\ \nu_{aardvark} \\ \vdots \\ \nu_{zymurgy} \\ V_a \\ V_{aardvark} \\ \vdots \\ V_{zymurgy} \end{bmatrix} \in \mathbb{R}^{2V \times H}$$
Actually doing the learning

We have many options. Gradient descent is popular. We want to optimize, given \( T \) tokens of training data,

\[
J(\theta) = \frac{1}{T} \sum_{t=1}^{T} \sum_{-c<j<c, j \neq 0} \log P(w_{t+j}|w_t)
\]

And we want to update vectors \( V_{w_{t+j}} \) then \( v_{w_t} \) within \( \theta \)

\[
\theta^{(new)} = \theta^{(old)} - \eta \nabla_{\theta} J(\theta)
\]

so we’ll need to take the derivative of the (log of the) softmax function:

\[
P(w_o|w_i) = \frac{\exp(V_{w_o}^T v_{w_i})}{\sum_{w=1}^{W} \exp(V_{w}^T v_{w_i})}
\]

Where \( v_{w} \) is the ‘input’ vector for word \( w \),
and \( V_{w} \) is the ‘output’ vector for word \( w \),
Actually doing the learning

We need the derivative of the (log of the) softmax function:

$$\frac{\delta}{\delta \nu_{wt}} \log P(w_{t+j} | w_t) = \frac{\delta}{\delta \nu_{wt}} \log \frac{\exp(V_{wt+j}^T \nu_{wt})}{\sum_{w=1}^{W} \exp(V_{w}^T \nu_{wt})}$$

$$= \frac{\delta}{\delta \nu_{wt}} \left[ \log \exp \left( V_{wt+j}^T \nu_{wt} \right) - \log \sum_{w=1}^{W} \exp \left( V_{w}^T \nu_{wt} \right) \right]$$

$$= V_{wt+j} - \frac{\delta}{\delta \nu_{wt}} \log \sum_{w=1}^{W} \exp \left( V_{w}^T \nu_{wt} \right)$$

[apply the chain rule $\frac{\delta f}{\delta \nu_{wt}} = \frac{\delta f}{\delta z} \frac{\delta z}{\delta \nu_{wt}}$]

$$= V_{wt+j} - \sum_{w=1}^{W} p(w|w_t) V_w$$

Using word representations

Without a latent space,

\[
\text{lugubrious} = [0, 0, 0, \ldots, 0, 1, 0, \ldots, 0], \quad & \quad \text{sad} = [0, 0, 0, \ldots, 0, 0, 1, \ldots, 0]
\]

\[\text{Similarity} = \cos(x, y) = 0.0\]

In latent space,

\[
\text{lugubrious} = [0.8, 0.69, 0.4, \ldots, 0.05]_H, \quad & \quad \text{sad} = [0.9, 0.7, 0.43, \ldots, 0.05]_H
\]

\[\text{Similarity} = \cos(x, y) = 0.9\]

Reminder:

\[\cos(u, v) = \frac{u \cdot v}{||u|| \times ||v||}\]
Skip-grams with negative sampling

• The default process is inefficient.
  • For one – what a waste of time! We don’t want to update $H \times D$ weights!
  • For two – we want to avoid confusion! ‘Hallucinated’ contexts should be minimized.

• For the observed pair (lugubrious, sadness), only the output neuron for sadness should be 1, and all $D - 1$ others should be 0.
Skip-grams with negative sampling

• We want to **maximize** the association of **observed** (positive) contexts:
  - lugubrious  sad
  - lugubrious  feeling
  - lugubrious  tired

• We want to **minimize** the association of ‘**hallucinated**’ (negative) contexts:
  - lugubrious  happy
  - lugubrious  roof
  - lugubrious  truth
Skip-grams with negative sampling

- Choose a small number $\eta$ of ‘negative’ words, and just update the weights for the ‘positive’ word plus the $\eta$ ‘negative’ words.
  - $5 \leq \eta \leq 20$ can work in practice for smaller datasets.
  - For $D = 100K$, we only update 0.006% of the weights in the output layer.

- The authors suggest choosing the top $\eta$ words by modified unigram probability:

$$P^*(w_{t+1}) = \frac{C(w_{t+1})^3}{\sum_w C(w)^4}$$
Smell the GloVe

• **GloVe** (‘**Glo**bal **Ve**ctors’) is an alternative method of obtaining word embeddings.
  • Instead of predicting words at particular positions, look at the **co-occurrence matrix**.

\[
X = \begin{bmatrix}
  0 & 2 & 1 & 0 & 0 & 0 & 0 \\
  2 & 0 & 0 & 1 & 0 & 1 & 0 \\
  1 & 0 & 0 & 0 & 0 & 1 & 0 \\
  0 & 1 & 0 & 0 & 1 & 0 & 0 \\
  0 & 0 & 0 & 1 & 0 & 0 & 1 \\
  0 & 1 & 0 & 0 & 0 & 0 & 1 \\
  0 & 0 & 0 & 0 & 1 & 1 & 1 \\
  0 & 0 & 0 & 0 & 1 & 1 & 0
\end{bmatrix}
\]

Word \( w_i \) occurs \( X_{i,j} (= X_{j,i}) \) times with word \( w_j \), within some context window (e.g., 10 words, a sentence, ...).

Smell the GloVe

• Populating the co-occurrence matrix requires a complete pass through the corpus, but needs only be done once.
• Let $P_{i,j} = P(w_j | w_i) = X_{i,j} / X_i$.

<table>
<thead>
<tr>
<th>Probability and Ratio</th>
<th>$k = \text{solid}$</th>
<th>$k = \text{gas}$</th>
<th>$k = \text{water}$</th>
<th>$k = \text{fashion}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P(k</td>
<td>\text{ice})$</td>
<td>$1.9 \times 10^{-4}$</td>
<td>$6.6 \times 10^{-5}$</td>
<td>$3.0 \times 10^{-3}$</td>
</tr>
<tr>
<td>$P(k</td>
<td>\text{steam})$</td>
<td>$2.2 \times 10^{-5}$</td>
<td>$7.8 \times 10^{-4}$</td>
<td>$2.2 \times 10^{-3}$</td>
</tr>
<tr>
<td>$P(k</td>
<td>\text{ice})/P(k</td>
<td>\text{steam})$</td>
<td>8.9</td>
<td>$8.5 \times 10^{-2}$</td>
</tr>
</tbody>
</table>

Aside – smell the GloVe

- Minimize $J = \sum_{i,j=1}^{V} f(X_{i,j}) \left( \nu_{w_i} V_{w_j} + b_i + \tilde{b}_j - \log X_{i,j} \right)^2$
  where $b_i$ and $\tilde{b}_j$ are input and output bias terms associated with $w_i$ and $w_j$, respectively

1. $f(0) = 0$. If $f$ is viewed as a continuous function, it should vanish as $x \to 0$ fast enough that the $\lim_{x \to 0} f(x) \log^2 x$ is finite.

2. $f(x)$ should be non-decreasing so that rare co-occurrences are not overweighted.

3. $f(x)$ should be relatively small for large values of $x$, so that frequent co-occurrences are not overweighted.

Of course a large number of functions satisfy these properties, but one class of functions that we found to work well can be parameterized as,

$$f(x) = \begin{cases} (x/x_{\text{max}})^\alpha & \text{if } x < x_{\text{max}} \\ 1 & \text{otherwise} \end{cases}$$  \hspace{1cm} (9)
Aside – smell the GloVe

• **Intrinsic evaluation**: popular method is to cherry-pick a few $k$-nearest neighbours examples that match expectations.

  0. frog 1. frogs 2. toad 3. litoria 4. leptodactylidae 5. rana 6. lizard 7. eleutherodactylus

  3. litoria 4. leptodactylidae 5. rana 7. eleutherodactylus

• **Extrinsic evaluation**: embed resulting vectors into a variety of tasks.

Redacted. See [https://github.com/sebastianruder/NLP-progress](https://github.com/sebastianruder/NLP-progress)
Linguistic regularities in vector space

Trained on the Google news corpus with over 300 billion words.
Linguistic regularities in vector space

(from GloVe)
Linguistic regularities in vector space

<table>
<thead>
<tr>
<th>Expression</th>
<th>Nearest token</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paris – France + Italy</td>
<td>Rome</td>
</tr>
<tr>
<td>Bigger – big + cold</td>
<td>Colder</td>
</tr>
<tr>
<td>Sushi – Japan + Germany</td>
<td>bratwurst</td>
</tr>
<tr>
<td>Cu – copper + gold</td>
<td>Au</td>
</tr>
<tr>
<td>Windows – Microsoft + Google</td>
<td>Android</td>
</tr>
</tbody>
</table>

**Analogies:** apple:apples :: octopus:octopodes  
**Hypernymy:** shirt:clothing :: chair:furniture
Importance of in-domain data

Let’s talk about gender at the UofT

However, in word2vec trained on Google News, 
\texttt{man:woman::programmer:homemaker}.

Let’s talk about gender at the UofT

Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings

Tolga Bolukbasi\textsuperscript{1}, Kai-Wei Chang\textsuperscript{2}, James Zou\textsuperscript{2}, Venkatesh Saligrama\textsuperscript{1,2}, Adam Kalai\textsuperscript{2}

\textsuperscript{1}Boston University, 8 Saint Mary’s Street, Boston, MA
\textsuperscript{2}Microsoft Research New England, 1 Memorial Drive, Cambridge, MA
tolgab@bu.edu, kw@kwchang.net, jamesyzou@gmail.com, srv@bu.edu, adam.kalai@microsoft.com

Abstract

The blind application of machine learning runs the risk of amplifying biases present in data. Such a danger is facing us with word embedding, a popular framework to represent text data as vectors which has been used in many machine learning and
Solution?

1. Hand-pick words $S_0$ that are ‘gender definitional’. ‘Neutral’ words are the complement, $N = V \setminus S_0$. 

![Diagram showing gender-definitional words](image)
Solution?

2. Project away gender subspace from gender-neutral words, \( w := w - w \cdot B \) for \( w \in N \), where \( B \) is the gender subspace.
Solution?

2. Project away gender subspace from gender-neutral words, \( w := w - w \cdot B \) for \( w \in N \), where \( B \) is the gender subspace.
Results

- Generate many analogies, see which ones preserve gender stereotypes.

  - He: *Blue* :: She: ?
  - He: *Doctor* :: She: ?
  - He: *Brother* :: She: ?
NEURAL LANGUAGE MODELS
Trigram models

- CBOW: prediction of current word $w_t$ given $w_{t-1}$.
- Let’s reconsider predicting $w_t$ given multiple $w_{t-j}$?
  - I.e., let’s think about language modelling.

Here:
- $w_i$ is a one-hot vector,
- $p_t$ is a distribution, and
- $|w_i| = |p_t| = |V|$ (i.e., the size of the vocabulary)
Sampling from trigram models

- Since $p_t \sim P(w_t|w_{t-2} w_{t-1})$, we just feed forward and sample from the output vector.
Training trigram models

• Here’s one approach:

1. Randomly choose a batch (e.g., 10K consecutive words)
2. Propagate words through the current model
3. Obtain word likelihoods (loss)
4. Back-propagate loss
5. Gradient step to update model
6. Go to (1)
Training trigram models

- The typical training objective is the cross entropy (see Lecture 3) of the corpus $C$ given the model $M$:

\[ F = H(C; M) = -\frac{\log_2 P_M(C)}{||C||} \]

Minimize

Maximize

\[ \log_2 P_M(C) = \log_2 \prod_{t=0}^{T} P(w_t) = \sum_{t=0}^{T} \log_2 P(w_t) \]

\[ \log_2 P(w_t) = w_t^\top \log p_t \]

Here:
- $w_i$ is a one-hot vector, and
- $p_t$ is a distribution.
Training trigram models

- Compute our gradients, using $F = -\frac{\log_2 P_M(C)}{\|C\|}$ and $\log_2 P(w_t) = w_t^\top \log p_t$ and backpropagate.

- $h_t = g(W_I[w_{t-2}; w_{t-1}] + c)$
- $p_t = \text{softmax}(W_O h_t + b)$

Here:
- $w_i$ is a one-hot vector, and
- $p_t$ is a distribution.
So what?

• 😊 Neural language models of this type:
  • Can generalize better than MLE LMs to unseen \( n \)-grams,
  • Can be modified to use \textit{semantic} information as in word2vec.

\[
P(\text{the cat sat on the mat}) \approx P(\text{the cat sat on the rug})
\]

• 😞 Neural language models of this type:
  • Can take \textit{relatively} long to train
  • Number of parameters scale poorly with increasing context.

\textit{Let’s improve both of these issues...}
Dealing with that bottleneck

- Traditional datasets for neural language modeling include:
  - AP News (14M tokens, 17K types)
  - HUB-4 (1M tokens, 25K types)
  - Google News (6B tokens, 1M types)
  - Wikipedia (3.2B tokens, 2M types)

- Awesome datasets for medical/clinical LM include:
  - EMRALD/ICES (3.5B tokens, 13M types)

- Much of the computational effort is in the initial embedding, and in the softmax.
  - Can we simplify and speed up the process?
Dealing with that bottleneck

• **Replace** rare words with `<out-of-vocabulary>` token.
• **Subsample** frequent words.

• Hierarchical softmax.
• Noise-contrastive estimation.
• Negative sampling.

Hierarchical softmax with grouping

• Group words into distinct classes, $c$, e.g., by frequency.
  • E.g., $c_1$ is top 5% of words by frequency, $c_2$ is the next 5%, ...

• Factorize $p(w_o | w_i) = p(c | w_i)p(w_o | w_i, c)$

\[
\text{softmax': } P(w_o | w_i) = \frac{\exp(V_{w_o}^T v_{w_i})}{\sum_{w=1}^{W} \exp(V_{w}^T v_{w_i})}
\]

\[
\text{Where }
\begin{align*}
\nu_w & \text{ is the ‘input’ vector for word } w, \\
V_w & \text{ is the ‘output’ vector for word } w,
\end{align*}
\]

\[
\exp(c_j v_{w_i}) \times \frac{\exp(V_{w_o}^T v_{w_i})}{\sum_{c \in C} \exp(c v_{w_i})} \times \frac{\exp(V_{w_o}^T v_{w_i})}{\sum_{w \in c} \exp(V_{w}^T v_{w_i})}
\]

[Mikolov et al, 2011, Auli et al, 2013]
RECURRENT NEURAL NETWORKS
Recurrent neural networks (RNNs)

- An RNN has **feedback** connections in its structure so that it ‘remembers’ previous states, when reading a sequence.
  - i.e., it passes information from one step to the next.

![Diagram of RNNs](image)

- Elman network feed hidden units back
- Jordan network (not shown) feed output units back
Unrolling the $h_i$

- Copies of the same network can be applied (i.e., unrolled) at each point in a time series.
- These can be applied to various tasks.

\[
h_t = g(W_i[x; h_{t-1}] + c)
\]

\[
y_t = W_0 h_t + b
\]
Sampling from a RNN LM

• If $|h_i| < |V|$, we’ve already reduced the number of parameters from the trigram NN.
• In ‘theory’, information is maintained in $h_i$ across arbitrary lengths of time...

$h_t = g(W_t[x; h_{t-1}] + c)$
$y_t = W_0 h_t + b$

Karpathy (2015),
The Unreasonable Effectiveness of Recurrent Neural Networks
RNNs and retrograde amnesia

- Unfortunately, catastrophic forgetting is common.
- E.g., the relevant context in “The sushi the sister of your friend’s programming teacher told you about was...” has likely been overwritten by the time $h_{13}$ is produced.

RNNs and retrograde amnesia

• The challenge with RNNs is that the gradient decays quickly as one pushes it back in time. Can we store relevant information?

Imagery and sequence from http://colah.github.io/posts/2015-08-Understanding-LSTMs/
Long short-term memory (LSTM)

• In each module, in an LSTM, there are four interacting neural network layers.

The cell state is a special vector stream that runs through the entire chain and stores the long-term information.
Long short-term memory (LSTM)

• In each module, in an LSTM, there are four interacting neural network layers.

Gates decide what information should be withheld in the cell state. They are a sigmoid followed by a pointwise $\times$. Values near 0 block information; values near 1 pass information.
LSTM step 1: decide what to forget

• The **forget gate layer** compares $h_{t-1}$ and the current input $x_t$ to decide which elements in cell state $C_{t-1}$ to keep and which to turn off.
  • E.g., the cell state might ‘remember’ the number (sing./plural) of the current subject, in order to predict appropriately conjugated verbs, but decide to forget it when a new subject is mentioned at $x_t$.
  • (There’s scanty evidence that such information is so explicit.)
LSTM step 2: decide what to store

• The **input gate layer** has two steps.
  • First, a sigmoid layer $\sigma$ decides which cell units to update.
  • Next, a tanh layer creates new candidate values $\tilde{C}_t$.
  • E.g., the $\sigma$ can turn on the ‘number’ units, and the tanh can push information on the current subject.
  • The $\sigma$ layer is important – we don’t want to push information on units (i.e., latent dimensions) for which we have no information.

\[
i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\
\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)
\]
LSTM step 3: update the cell state

- Update $C_{t-1}$ to $C_t$.
  - First, forget what we want to forget: multiply $C_{t-1}$ by $f_t$.
  - Then, create a ‘mask vector’ of information we want to store, $i_t \times \widehat{C}_t$.
  - Finally, write this information to the new cell state $C_t$.

\[
C_t = f_t \times C_{t-1} + i_t \times \widehat{C}_t
\]
LSTM step 4: output and feedback

- Output something, \( o_t \), based on the current \( x_t \) and \( h_{t-1} \).
- Combine the output with the cell to give your \( h_t \).
  - Normalize cell \( C_t \) on \([-1,1]\) using \( \tanh \) and combine with \( o_t \).
- In some sense, \( C_t \) is long-term memory and \( h_t \) is the short-term memory (hence the name).

\[
o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)
\]
\[
h_t = o_t \times \tanh(C_t)
\]
Variants of LSTMs

• There are various variations on LSTMs.
  • ‘Bidirectional LSTMs’ (and bidirectional RNNs generally), learn

Variants of LSTMs

• There are various variations on LSTMs.
  • Gers & Schmidhuber (2000) add ‘peepholes’ that allow all sigmoids to read the cell state.
  • We can couple the ‘forget’ and ‘input’ gates.
    • E.g., it’s a bit of a waste to decide to forget number, then decide to store a new number.
  • Gated Recurrent units (GRUs; Cho et al (2014)) go a step further and also merge the cell and hidden states.

Are there examples where GRUs are used instead of LSTMs?
RECENT-ISH BREAKTHROUGHS
Deep contextualized representations

• What does the word play mean?

**ELMo: Embeddings from Language Models**

- Instead of a fixed embedding for each word type, ELMo considers the entire sentence before embedding each token.
- It uses a bi-directional LSTM trained on a specific task.
- Outputs are softmax probabilities on words, as before.
ELMo: Embeddings from Language Models

• Producing the final embedding for word token $k$.

$$\text{ELMo}_{k}^{task} = E(R_k; \Theta^{task}) = \gamma^{task} \sum_{j=0}^{L} s_j^{task} h_{k,j}^{LM}$$

where $R_K$ is the set of all $L$ hidden layers, $h_{k,j}$
$s_j^{task}$ is the task’s weight on the layer, and
$\gamma^{task}$ is a weight on the entire task
**ELMo: Embeddings from Language Models**

1. **Concatenate**

2. **Multiply by weight vectors**

3. **Sum**

\[ \text{ELMo}_{k=1}^{\text{task}} \]
ELMo: Embeddings from Language Models

- What does the word *play* mean?

<table>
<thead>
<tr>
<th>Source</th>
<th>Nearest Neighbors</th>
</tr>
</thead>
<tbody>
<tr>
<td>GloVe</td>
<td>playing, game, games, played, players, plays, player, Play, football, multiplayer</td>
</tr>
<tr>
<td>biLM</td>
<td>Chico Ruiz made a spectacular play on Alusik’s grounder {...} Kieffer, the only junior in the group, was commended for his ability to hit in the clutch, as well as his all-round excellent play. Olivia De Havilland signed to do a Broadway play for Garson {...} they were actors who had been handed fat roles in a successful play, and had talent enough to fill the roles competently, with nice understatement.</td>
</tr>
</tbody>
</table>

Table 4: Nearest neighbors to “play” using GloVe and the context embeddings from a biLM.

ELMo: Embeddings from Language Models

<table>
<thead>
<tr>
<th>TASK</th>
<th>PREVIOUS SOTA</th>
<th>OUR ELMo + BASELINE BASELINE</th>
<th>INCREASE (ABSOLUTE/RELATIVE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQuAD</td>
<td>Liu et al. (2017)</td>
<td>81.1</td>
<td>85.8</td>
</tr>
<tr>
<td>SNLI</td>
<td>Chen et al. (2017)</td>
<td>88.0</td>
<td>88.7 ± 0.17</td>
</tr>
<tr>
<td>SRL</td>
<td>He et al. (2017)</td>
<td>81.4</td>
<td>84.6</td>
</tr>
<tr>
<td>Coref</td>
<td>Lee et al. (2017)</td>
<td>67.2</td>
<td>70.4</td>
</tr>
<tr>
<td>NER</td>
<td>Peters et al. (2017)</td>
<td>90.15</td>
<td>92.22 ± 0.10</td>
</tr>
<tr>
<td>SST-5</td>
<td>McCann et al. (2017)</td>
<td>53.7</td>
<td>54.7 ± 0.5</td>
</tr>
</tbody>
</table>

Table 1: Test set comparison of ELMo enhanced neural models with state-of-the-art single model baselines across six benchmark NLP tasks. The performance metric varies across tasks – accuracy for SNLI and SST-5; F₁ for SQuAD, SRL and NER; average F₁ for Coref. Due to the small test sizes for NER and SST-5, we report the mean and standard deviation across five runs with different random seeds. The “increase” column lists both the absolute and relative improvements over our baseline.

BERT: Bidirectional encoder representations from transformers

• Unlike ELMo, BERT is deep**ly** bidirectional.
  • i.e., every embedding conditions every other in the next layer.

• This is difficult, because when predicting word \( x_t \), you would already have ‘seen’ that word in modelling its own contexts.

Code and models: https://github.com/google-research/bert

BERT: Bidirectional encoder representations from transformers
BERT: **Bidirectional encoder representations from transformers**

• This can be solved by **masking** the word being predicted.

  ![Input: The man went to the [MASK]₁. He bought a [MASK]₂ of milk. Labels: [MASK]₁ = store; [MASK]₂ = gallon](image)

  • (actually, 80% we use [MASK]. 10% we replace the target word with another actual word; 10% we keep the word as-is, to bias ‘towards the observation’.)

• We can also predict other relationships, like whether one sentence follows another.

  ![Sentence A = The man went to the store. Sentence B = He bought a gallon of milk. Label = IsNextSentence](image)  ![Sentence A = The man went to the store. Sentence B = Penguins are flightless. Label = NotNextSentence](image)

  • (actually, you can fine-tune on *many* different tasks)
BERT: Bidirectional encoder representations from transformers

(From http://jalammar.github.io/illustrated-bert/)
BERT: Bidirectional encoder representations from transformers

- The age of humans is over?

<table>
<thead>
<tr>
<th>Rank</th>
<th>Model</th>
<th>EM</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Human Performance Stanford University (Rajpurkar et al. ‘16)</td>
<td>82.304</td>
<td>91.221</td>
</tr>
<tr>
<td>2</td>
<td>BERT (ensemble) Google AI Language</td>
<td>87.433</td>
<td>93.160</td>
</tr>
<tr>
<td>3</td>
<td>nlnet (ensemble) Microsoft Research Asia</td>
<td>85.356</td>
<td>91.202</td>
</tr>
<tr>
<td>4</td>
<td>QANet (ensemble) Google Brain &amp; CMU</td>
<td>84.454</td>
<td>90.490</td>
</tr>
</tbody>
</table>
Aside – ClosedAI

• There are, of course, alternatives.

• **FastText**: Represent each word as a bag of character-grams
  
  
  Code: [https://fasttext.cc](https://fasttext.cc)

• **ULMFit**: Model fine-tuning for classification tasks
  
  
  Code: [Here](https://fasttext.cc)

• **GPT-2**: Spooky uni-directional model
  
  Paper: [Here](https://fasttext.cc)
  
  Blog: [Here](https://fasttext.cc)
OTHER APPLICATIONS
Sentiment analysis

• The traditional \textit{bag-of-words} approach to sentiment analysis used dictionaries of \textit{happy} and \textit{sad} words, simple counts, and either \textit{regression} or \textit{binary} classification.

• But consider these:

  - \textbf{Best} movie of the year
  - \textbf{Slick} and \textbf{entertaining}, despite a \textbf{weak} script
  - \textbf{Fun} and \textbf{sweet} but ultimately \textbf{unsatisfying}
Tree-based sentiment analysis

• We can combine pairs of words into phrase structures.
• Similarly, we can combine phrase and word structures hierarchically for classification.
Tree-based sentiment analysis

(currently broken) demo:
http://nlp.stanford.edu/sentiment/
Neural networks

• Research in neural networks is exciting, expansive, and explorative.
• We have many hyperparameters we can tweak (e.g., activation functions, number and size of layers).
• We have many architectures we can use (e.g., deep networks, LSTMs, attention mechanisms).

• Given the fevered hype, it’s important to retain our scientific skepticism.
  • What are our biases and expectations?
  • When are neural networks the wrong choice?
  • How are we actually evaluating these systems?