Lecture 7: seq2seq + Attention
Sequence Generation

Harvard
AC295/CS287r/CSCI E-115B
Chris Tanner
ScHoolboy Q
[seq2seq] + Attention (2019)
ANNOUNCEMENTS

• HW2 is out! Determine your mystery language.

• **Research Proposals** are due in 7 days, **Sept 30**.

• Office Hours:
  
  • Today, my OH will be pushed back: **3:30pm – 5:30pm**
  
  • Please reserve your **coding questions for the TFs and/or EdStem**, as I hold office hours solo, and debugging code can easily bottleneck the queue.

• **Saturday @ 9am**, I’ll host & record a **review session**. Submit questions on Ed’s Sway
RECAP: L5

• RNNs help capture more context while avoiding sparsity, storage, and compute issues!

• The hidden layer is what we care about. It represents the word’s “meaning”.

• Often suffers from vanishing/exploding gradients
RECAP: L6

• Gradient Clipping may help all NNs

• LSTMs (1997) are usually much better than vanilla RNNs
  • Captures long-range dependencies
  • Doesn’t suffer as much w/ its gradients

Emilia told her project partner Alan about ___ latest idea.

He tends to stress out and put too much pressure on himself
some old memories are “forgotten”

some new memories are made

memory is written, erased, and read by three gates – which are influenced by $x$ and $h$

a nonlinear weighted version of the long-term memory becomes our short-term memory

Diagram: https://colah.github.io/posts/2015-08-Understanding-LSTMs/
RECAP: L6

Diagram: https://colah.github.io/posts/2015-08-Understanding-LSTMs/
Outline

- Long Short-Term Memory (LSTMs)
- Bi-LSTM and ELMo
- seq2seq
- seq2seq + Attention
Outline

- Long Short-Term Memory (LSTMs)
- Bi-LSTM and ELMo
- seq2seq
- seq2seq + Attention
LSTMs

They’ve been around for a while, but essentially unused until 2014!
LSTMs

Figure 1: Architecture of memory cell $c_j$ (the box) and its gate units $in_j$, $out_j$. The self-recurrent connection (with weight 1.0) indicates feedback with a delay of 1 time step. It builds the basis of the “constant error carousel” CEC. The gate units open and close access to CEC. See text and appendix A.1 for details.

Why gate units? To avoid input weight conflicts, $in_j$ controls the error flow to memory cell $c_j$’s input connections $w_{cj}$. To circumvent $c_j$’s output weight conflicts, $out_j$ controls the error flow from unit $j$’s output connections. In other words, the net can use $in_j$ to decide when to keep or override information in memory cell $c_j$, and $out_j$ to decide when to access memory cell $c_j$ and when to prevent other units from being perturbed by $c_j$ (see Figure 1).
Sequential Modelling

If your goal isn’t to predict the next item in a sequence, you could instead perform classification or regression task using the learned, hidden representations.
Sequential Modelling

Language Modelling

Output layer
Hidden layer
Input layer

Auto-regressive

Non-Auto-regressive

1-to-1 tagging/classification
Sequential Modelling

Language Modelling

1-to-1 tagging/classification

If it’s **regressive** or not depends on if each output is fed back in as the next input.

Auto-regressive

Non-Auto-regressive
Sequential Modelling

Many-to-1 classification

Input layer

Hidden layer

Output layer

Sentiment score

Regression
Binary classification
Multi-class classification
Sequential Modelling

Many-to-1 classification

Sentiment score

Output layer

Hidden layer

Input layer

$x_1$, $x_2$, $x_3$, $x_4$

Regression
Binary classification
Multi-class classification
### Types of Prediction

<table>
<thead>
<tr>
<th>Types of Prediction</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regression</td>
<td>I love hiking!</td>
<td>0.9</td>
</tr>
<tr>
<td>Binary Classification</td>
<td>I love hiking!</td>
<td>Positive or negative</td>
</tr>
<tr>
<td>Multi-class Classification</td>
<td>I love hiking!</td>
<td>Very positive, positive, neutral, negative, or very negative</td>
</tr>
<tr>
<td>Structured Prediction</td>
<td>I love hiking!</td>
<td>PRP VBP NN</td>
</tr>
</tbody>
</table>

(difficult scenario when your output has exponential/infinite # of possibilities)
Types of Prediction (an independent axis)

**Unconditioned Prediction**: predict some single variable. $P(X)$

*Example*: language modelling. $X = “I like hiking!”$

**Conditioned Prediction**: predict the probability of an output variable, given the input. $P(Y|X)$

*Example*: text classification. $Y = positive$. $X = “I like hiking!”$
Types of Prediction (an independent axis)

**Unconditioned Prediction**: predict some single variable. \( P(X) \)

Example: language modelling. \( X = \text{“I like hiking!”} \)

(\( \text{un} \))conditioned is referring to if you’re entire model is predicated upon some particular input.

**Conditioned Prediction**: predict the probability of an output variable, given the input. \( P(Y|X) \)

Example: text classification. \( Y = \text{positive.} \ X = \text{“I like hiking!”} \)
Types of Prediction (an independent axis)

**Unconditioned Prediction**: predict some single variable. $P(X)$

**Example**: language modelling. $X = \text{“I like hiking!”}$

Language modelling is **unconditional prediction**, but one could do so by making use of **conditional probabilities** of $X$.

**Conditioned Prediction**: predict the probability of an output variable, given. $P(Y|X)$

**Example**: text classification. $Y = \text{positive}$. $X = \text{“I like hiking!”}$
Types of Unconditional Prediction

**Independent Prediction**

\[
P(X) = \prod_{i=1}^{\mid X \mid} P(x_i)
\]

(e.g. unigram model)

**Left-to-right Markov Chain (order n-1)**

\[
P(X) = \prod_{i=1}^{\mid X \mid} P(x_i | x_{i-n+1}, \ldots, x_{i-1})
\]

(e.g. n-gram LM, feed-forward LM)

**Left-to-right Autoregressive Prediction**

\[
P(X) = \prod_{i=1}^{\mid X \mid} P(x_i | x_1, \ldots, x_{i-1})
\]

(e.g. RNN LM)
Types of Unconditional Prediction

Formally, a language model estimates the probability of a sequence, so this is illegal. It cheats in a manner that we call them masked language models (not proper prob. dist and they don’t estimate sequences)
Types of Conditional Prediction

**Many-to-1 classification**

\[ P(y|X) \]

**Many-to-many classification**

- **Non-autoregressive Conditioned Prediction**
  \[ P(Y|X) = \prod_{i=1}^{|Y|} P(y_i|X) \]
  (e.g. sequence labeling, non-autoregressive MT)

- **Autoregressive Conditioned Prediction**
  \[ P(Y|X) = \prod_{i=1}^{|Y|} P(y_i|X, y_1, \ldots, y_{i-1}) \]
  (e.g. seq2seq model)
This concludes the foundation in sequential representation.

Most state-of-the-art advances are based on those core RNN/LSTM ideas. But, with tens of thousands of researchers and hackers exploring deep learning, there are many tweaks that haven proven useful.

(aka this is where things get crazy.)
Outline

- Long Short-Term Memory (LSTMs)
- Bi-LSTM and ELMo
- seq2seq
- seq2seq + Attention
Outline

- Long Short-Term Memory (LSTMs)
- Bi-LSTM and ELMo
- seq2seq
- seq2seq + Attention
RNN Extensions: Bi-directional LSTMs

RNNs/LSTMs use the left-to-right context and sequentially process data.

If you have full access to the data at testing time, why not make use of the flow of information from right-to-left, also?
RNN Extensions: Bi-directional LSTMs

For brevity, let’s use the follow schematic to represent an RNN
RNN Extensions: Bi-directional LSTMs

For brevity, let's use the follow schematic to represent an RNN.
RNN Extensions: Bi-directional LSTMs

Concatenate the hidden layers

Hidden layer

Input layer

\[ h^L_1 \rightarrow h^L_2 \rightarrow h^L_3 \rightarrow h^L_4 \]

\[ h^R_1 \rightarrow h^R_2 \rightarrow h^R_3 \rightarrow h^R_4 \]
RNN Extensions: Bi-directional LSTMs

Output layer

\[ \hat{y}_1 \quad \hat{y}_2 \quad \hat{y}_3 \quad \hat{y}_4 \]

Concatenate the hidden layers

\[ h_1^R \quad h_2^R \quad h_3^R \quad h_4^R \]

\[ h_1^L \quad h_2^L \quad h_3^L \quad h_4^L \]

Hidden layer

\[ h_1 \quad h_2 \quad h_3 \quad h_4 \]

Input layer

\[ x_1 \quad x_2 \quad x_3 \quad x_4 \]
RNN Extensions: Bi-directional LSTMs

**BI-LSTM STRENGTHS?**

- Usually performs at least as well as uni-directional RNNs/LSTMs

**BI-LSTM ISSUES?**

- Slower to train
- Only possible if access to full data is allowed
Type of Unconditional Prediction

Bidirectional Prediction

\[ P(X) \neq \prod_{i=1}^{\left|X\right|} P(x_i | x \neq i) \] (e.g. masked language model)
RNN Extensions: Stacked LSTMs

Hidden layers provide an abstraction (holds “meaning”).

Stacking hidden layers provides increased abstractions.
RNN Extensions: Stacked LSTMs

Hidden layers provide an abstraction (holds “meaning”). Stacking hidden layers provides increased abstractions.
RNN Extensions: Stacked LSTMs

Hidden layers provide an abstraction (holds “meaning”).

Stacking hidden layers provides increased abstractions.
ELMo: Stacked Bi-directional LSTMs

General Idea:

• Goal is to get highly rich, contextualized embeddings (word tokens)

• Use both directions of context (bi-directional), with increasing abstractions (stacked)

• Linearly combine all abstract representations (hidden layers) and optimize w.r.t. a particular task (e.g., sentiment classification)
ELMo: Stacked Bi-directional LSTMs

Illustration: http://jalammar.github.io/illustrated-bert/
Embedding of “stick” in “Let’s stick to” - Step #2

1- Concatenate hidden layers

Forward Language Model

Backward Language Model

2- Multiply each vector by a weight based on the task

3- Sum the (now weighted) vectors

ELMo embedding of “stick” for this task in this context

Illustration: http://jalammar.github.io/illustrated-bert/
ELMo: Stacked Bi-directional LSTMs

<table>
<thead>
<tr>
<th>Task</th>
<th>Previous SOTA</th>
<th>Our Baseline</th>
<th>ELMo + Baseline</th>
<th>Increase (Absolute/Relative)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQuAD</td>
<td>Liu et al. (2017)</td>
<td>84.4</td>
<td>81.1</td>
<td>85.8</td>
</tr>
<tr>
<td>SNLI</td>
<td>Chen et al. (2017)</td>
<td>88.6</td>
<td>88.0</td>
<td>88.7 ± 0.17</td>
</tr>
<tr>
<td>SRL</td>
<td>He et al. (2017)</td>
<td>81.7</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>67.2</td>
<td>70.4</td>
</tr>
<tr>
<td>NER</td>
<td>Peters et al. (2017)</td>
<td>91.93 ± 0.19</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>51.4</td>
<td>54.7 ± 0.5</td>
</tr>
</tbody>
</table>

**ELMo: Stacked Bi-directional LSTMs**

<table>
<thead>
<tr>
<th>Model</th>
<th>$F_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>WordNet 1st Sense Baseline</td>
<td>65.9</td>
</tr>
<tr>
<td>Raganato et al. (2017a)</td>
<td>69.9</td>
</tr>
<tr>
<td>Iacobacci et al. (2016)</td>
<td>70.1</td>
</tr>
<tr>
<td>CoVe, First Layer</td>
<td>59.4</td>
</tr>
<tr>
<td>CoVe, Second Layer</td>
<td>64.7</td>
</tr>
<tr>
<td>biLM, First Layer</td>
<td>67.4</td>
</tr>
<tr>
<td>biLM, Second layer</td>
<td>69.0</td>
</tr>
</tbody>
</table>

**Model** | **Acc.**
---|---
Collobert et al. (2011) | 97.3 |
Ma and Hovy (2016)       | 97.6 |
Ling et al. (2015)       | 97.8 |
CoVe, First Layer        | 93.3 |
CoVe, Second Layer       | 92.8 |
biLM, First Layer        | 97.3 |
biLM, Second Layer       | 96.8 |

Table 5: All-words fine grained WSD $F_1$. For CoVe and the biLM, we report scores for both the first and second layer biLSTMs.

The higher layer seems to learn **semantics** while the lower layer probably captured **syntactic** features.

---

ELMo: Stacked Bi-directional LSTMs

• ELMo yielded incredibly good contextualized embeddings, which yielded SOTA results when applied to many NLP tasks.

• Main ELMo takeaway: given enough training data, having tons of explicit connections between your vectors is useful (system can determine how to best use context)

ELMo Slides: https://www.slideshare.net/shuntaroy/a-review-of-deep-contextualized-word-representations-peters-2018
RECAP

• **Language Modelling** may help us for other tasks

• **LSTMs** do a great job of capturing “meaning”, which can be used for almost every task
  
  • Given a sequence of $N$ words, we can produce 1 output
  
  • Given a sequence of $N$ words, we can produce $N$ outputs
RECAP

• **Language Modelling** may help us for other tasks

• **LSTMs** do a great job of capturing “meaning”, which can be used for almost every task
  
  • Given a sequence of \( N \) words, we can produce 1 output
  • Given a sequence of \( N \) words, we can produce \( N \) outputs
  • What if we wish to have \( M \) outputs?
We want to produce a **variable-length** output
(e.g., $n \rightarrow m$ predictions)

Thank you for visiting!

Děkujeme za návštěvu!
Outline

- Long Short-Term Memory (LSTMs)
- Bi-LSTM and ELMo
- seq2seq
- seq2seq + Attention
Outline

- Long Short-Term Memory (LSTMs)
- Bi-LSTM and ELMo
- seq2seq
- seq2seq + Attention
Sequence-to-Sequence (seq2seq)

• If our input is a sentence in Language A, and we wish to translate it to Language B, it is clearly sub-optimal to translate word by word (like our current models are suited to do).

• Instead, let a sequence of tokens be the unit that we ultimately wish to work with (a sequence of length $N$ may emit a sequences of length $M$).

• seq2seq models are comprised of 2 RNNs: 1 encoder, 1 decoder.
The brown dog ran
Sequence-to-Sequence (seq2seq)

The final hidden state of the encoder RNN is the initial state of the decoder RNN.
The final hidden state of the encoder RNN is the initial state of the decoder RNN.
Sequence-to-Sequence (seq2seq)

The final hidden state of the encoder RNN is the initial state of the decoder RNN.
The final hidden state of the encoder RNN is the initial state of the decoder RNN.
The brown dog ran.

The final hidden state of the encoder RNN is the initial state of the decoder RNN.
Sequence-to-Sequence (seq2seq)

The final hidden state of the encoder RNN is the initial state of the decoder RNN.

The brown dog ran

ENCODER RNN

DECODER RNN
Sequence-to-Sequence (seq2seq)

The final hidden state of the encoder RNN is the initial state of the decoder RNN.

ENCODER RNN

DECODER RNN

The brown dog ran

Le chien brun
The final hidden state of the encoder RNN is the initial state of the decoder RNN.
The final hidden state of the encoder RNN is the initial state of the decoder RNN.
Sequence-to-Sequence (seq2seq)

The final hidden state of the encoder RNN is the initial state of the decoder RNN.

The brown dog ran

ENCODER RNN

DECODER RNN
Sequence-to-Sequence (seq2seq)

The final hidden state of the encoder RNN is the initial state of the decoder RNN.

ENCODER RNN

DECODER RNN

The brown dog ran

Le chien brun a couru
Sequence-to-Sequence (seq2seq)

The final hidden state of the encoder RNN is the initial state of the decoder RNN.

ENCODER RNN

DECODER RNN
The final hidden state of the encoder RNN is the initial state of the decoder RNN.
Sequence-to-Sequence (seq2seq)

The brown dog ran

ENCODER RNN

DECODER RNN

Input layer

Hidden layer

$h^E_1$ $h^E_2$ $h^E_3$ $h^E_4$

$h^D_1$ $h^D_2$ $h^D_3$ $h^D_4$

$\hat{y}_1$ $\hat{y}_2$ $\hat{y}_3$ $\hat{y}_4$ $\hat{y}_5$ $\hat{y}_6$

The brown dog ran

<s> Le chien brun a couru
Sequence-to-Sequence (seq2seq)

Training occurs like RNNs typically do; the loss (from the decoder outputs) is calculated, and we update weights all the way to the beginning (encoder).
The brown dog ran.

**Testing** generates decoder outputs one word at a time, until we generate a `<S>` token.

Each decoder's $\hat{y}_i$ becomes the input $x_{i+1}$.
Sequence-to-Sequence (seq2seq)

See any issues with this traditional seq2seq paradigm?
Sequence-to-Sequence (seq2seq)

ENCODER RNN

DECODER RNN

Input layer: The brown dog ran

Hidden layer: $h^E_1 \rightarrow h^E_2 \rightarrow h^E_3 \rightarrow h^E_4$

$\hat{y}_1 \rightarrow \hat{y}_2 \rightarrow \hat{y}_3 \rightarrow \hat{y}_4 \rightarrow \hat{y}_5 \rightarrow \hat{y}_6$

Output: Le chien brun a couru
Sequence-to-Sequence (seq2seq)

It’s crazy that the entire “meaning” of the 1st sequence is expected to be packed into this one embedding, and that the encoder then never interacts with the decoder again. Hands free.
Sequence-to-Sequence (seq2seq)

Instead, what if the decoder, at each step, pays attention to a distribution of all of the encoder’s hidden states?
Sequence-to-Sequence (seq2seq)

Instead, what if the decoder, at each step, pays attention to a distribution of all of the encoder’s hidden states?

**Intuition:** when we (humans) translate a sentence, we don’t just consume the original sentence, reflect on the meaning of the last word, then regurgitate in a new language; we continuously think back at the original sentence while focusing on different parts.
Attention

The concept of attention within cognitive neuroscience and psychology dates back to the 1800s. [William James, 1890].

Nadaray-Watson kernel regression proposed in 1964. It locally weighted its predictions.
seq2seq + Attention

Q: How do we determine how much to pay attention to each of the encoder’s hidden layers?

\[
\begin{align*}
\text{The} & \Rightarrow \text{brown} \Rightarrow \text{dog} \Rightarrow \text{ran} \\
& \Rightarrow h_1^E \Rightarrow h_2^E \Rightarrow h_3^E \Rightarrow h_4^E
\end{align*}
\]
seq2seq + Attention

Q: How do we determine how much to pay attention to each of the encoder’s hidden layers?

A: Let’s base it on our decoder’s current hidden state (our current representation of meaning) and all of the encoder’s hidden layers!

![Diagram showing encoder and decoder RNNs with attention mechanism.]
seq2seq + Attention

Q: How do we determine how much to pay attention to each of the encoder’s hidden layers?

A: Let’s base it on our decoder’s current hidden state (our current representation of meaning) and all of the encoder’s hidden layers!

![Diagram showing encoder and decoder RNNs with attention](image-url)
seq2seq + Attention

Q: How do we determine how much to pay attention to each of the encoder’s hidden layers?

A: Let's base it on our decoder’s current hidden state (our current representation of meaning) and all of the encoder’s hidden layers!

Separate FFNN
seq2seq + Attention

Q: How do we determine how much to pay attention to each of the encoder’s hidden layers?

A: Let’s base it on our decoder’s current hidden state (our current representation of meaning) and all of the encoder’s hidden layers!
seq2seq + Attention

Q: How do we determine how much to pay attention to each of the encoder’s hidden layers?

A: Let’s base it on our decoder’s current hidden state (our current representation of meaning) and all of the encoder’s hidden layers!

ENCODER RNN

DECODER RNN

Separate FFNN
**Q:** How do we determine how much to pay attention to each of the encoder’s hidden layers?

**A:** Let’s base it on our decoder’s current hidden state (our current representation of meaning) and all of the encoder’s hidden layers!

Attention (raw scores)

\[ e_1 = 1.5 \]
\[ e_2 = 0.9 \]
\[ e_3 = 0.2 \]
\[ e_4 = -0.5 \]
**seq2seq + Attention**

**Q:** How do we determine how much to pay attention to each of the encoder’s hidden layers?

**A:** Let’s base it on our decoder’s current hidden state (our current representation of meaning) and all of the encoder’s hidden layers!

\[
\begin{align*}
A_i &= \exp(e_i) / \sum_i \exp(e_i) \\
&= \frac{\exp(e_1)}{\sum_i \exp(e_i)}
\end{align*}
\]
**seq2seq + Attention**

**Q:** How do we determine how much to pay attention to each of the encoder’s hidden layers?

**A:** Let’s base it on our decoder’s current hidden state (our current representation of meaning) and all of the encoder’s hidden layers!

<table>
<thead>
<tr>
<th>Attention (raw scores)</th>
</tr>
</thead>
</table>
| $e_1$ | 1.5  
| $e_2$ | 0.9  
| $e_3$ | 0.2  
| $e_4$ | -0.5  

<table>
<thead>
<tr>
<th>Attention (softmax’d)</th>
</tr>
</thead>
</table>
| $a_1^1$ | 0.51  
| $a_2^1$ | 0.28  
| $a_3^1$ | 0.14  
| $a_3^1$ | 0.07  

**ENCODER RNN**

- $h_1^E$
- $h_2^E$
- $h_3^E$
- $h_4^E$

**DECODER RNN**

- $h_1^D$
- The
- brown
- dog
- ran
- `<s>`
seq2seq + Attention

We multiply each encoder’s hidden layer by its $a_i^1$ attention weights to create a context vector $c_1^D$

Attention (softmax’d)

- $a_1 = 0.51$
- $a_2 = 0.28$
- $a_3 = 0.14$
- $a_3 = 0.07$
seq2seq + Attention

**REMEMBER:** each attention weight $a_i^j$ is based on the decoder's current hidden state, too.
The brown dog ran

**REMEMBER:** each attention weight $a_i^j$ is based on the decoder’s current hidden state, too.

ENCODER RNN

DECODER RNN
seq2seq + Attention

REMEMBER: each attention weight $a^D_i$ is based on the decoder’s current hidden state, too.

ENCODER RNN

DECODER RNN
seq2seq + Attention

REMEMBER: each attention weight $a^j_i$ is based on the decoder's current hidden state, too.

The brown dog ran

ENCODER RNN

DECODER RNN
seq2seq + Attention

REMEMBER: each attention weight $a_i^j$ is based on the decoder's current hidden state, too.
For convenience, here’s the Attention calculation summarized on 1 slide

Attention output

\[ c(t) = a_1(t) s_1 + a_2(t) s_2 + \cdots + a_m(t) s_m = \sum_{k=1}^{m} a_k(t) s_k \]

“source context for decoder step \(t\)”

Attention weights

\[ a_k(t) = \frac{\exp(\text{score}(h_t, s_k))}{\sum_{l=1}^{m} \exp(\text{score}(h_t, s_l))}, \quad k = 1..m \]

“attention weight for source token \(k\) at decoder step \(t\)”

Attention scores

\[ \text{score}(h_t, s_k), \quad k = 1..m \]

“How relevant is source token \(k\) for target step \(t\)?”

Attention input

\( s_1, s_2, \ldots, s_m \), \( h_t \)

all encoder states, one decoder state
The **Attention mechanism** that produces scores doesn’t have to be a **FFNN** like I illustrated. It can be any function you wish.
Popular Attention Scoring functions:

- **Dot-product**: \( \text{score}(h_t, s_k) = h_t^T s_k \)
- **Bilinear**: \( \text{score}(h_t, s_k) = h_t^T W s_k \)
- **Multi-Layer Perceptron**: \( \text{score}(h_t, s_k) = w_2^T \cdot \tanh(W_1[h_t, s_k]) \)

Photo credit: https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html
seq2seq + Attention

Attention:

• greatly improves seq2seq results
• allows us to visualize the contribution each encoding word gave for each decoder’s word

Image source: Fig 3 in Bahdanau et al., 2015
Takeaway:

Having a separate encoder and decoder allows for $n \rightarrow m$ length predictions.

Attention is powerful; allows us to conditionally weight our focus.
Constituency Parsing

Input: dogs chase cats

Output:

```
  S
 / \ \
NP   VP
 / \
|   |
dogs chase NP
cats
```

or a flattened representation

```
(S (NP dogs )_NP (VP chase (NP cats )_NP )_VP )_S
```
Constituency Parsing

Input: I shot an elephant in my pajamas

Output:

Figure 13.2 Two parse trees for an ambiguous sentence. The parse on the left corresponds to the humorous reading in which the elephant is in the pajamas, the parse on the right corresponds to the reading in which Captain Spaulding did the shooting in his pajamas.

<table>
<thead>
<tr>
<th>Model</th>
<th>English LR</th>
<th>English LP</th>
<th>English F1</th>
<th>Chinese LR</th>
<th>Chinese LP</th>
<th>Chinese F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shen et al. (2018)</td>
<td>92.0</td>
<td>91.7</td>
<td>91.8</td>
<td>86.6</td>
<td>86.4</td>
<td>86.5</td>
</tr>
<tr>
<td>Fried and Klein (2018)</td>
<td>-</td>
<td>-</td>
<td>92.2</td>
<td>-</td>
<td>-</td>
<td>87.0</td>
</tr>
<tr>
<td>Teng and Zhang (2018)</td>
<td>92.2</td>
<td>92.5</td>
<td>92.4</td>
<td>86.6</td>
<td>88.0</td>
<td>87.3</td>
</tr>
<tr>
<td>Vaswani et al. (2017)</td>
<td>-</td>
<td>-</td>
<td>92.7</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Dyer et al. (2016)</td>
<td>-</td>
<td>-</td>
<td>93.3</td>
<td>-</td>
<td>-</td>
<td>84.6</td>
</tr>
<tr>
<td>Kuncoro et al. (2017)</td>
<td>-</td>
<td>-</td>
<td>93.6</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Charniak et al. (2016)</td>
<td>-</td>
<td>-</td>
<td>93.8</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Liu and Zhang (2017b)</td>
<td>91.3</td>
<td>92.1</td>
<td>91.7</td>
<td>85.9</td>
<td>85.2</td>
<td>85.5</td>
</tr>
<tr>
<td>Liu and Zhang (2017a)</td>
<td>-</td>
<td>-</td>
<td>94.2</td>
<td>-</td>
<td>-</td>
<td>86.1</td>
</tr>
<tr>
<td>Suzuki et al. (2018)</td>
<td>-</td>
<td>-</td>
<td>94.32</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Takase et al. (2018)</td>
<td>-</td>
<td>-</td>
<td>94.47</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Fried et al. (2017)</td>
<td>-</td>
<td>-</td>
<td>94.66</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Kitaev and Klein (2018)</td>
<td>94.85</td>
<td>95.40</td>
<td>95.13</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Kitaev et al. (2018)</td>
<td>95.51</td>
<td>96.03</td>
<td>95.77</td>
<td>91.55</td>
<td>91.96</td>
<td>91.75</td>
</tr>
<tr>
<td>Zhou and Zhao (2019) (BERT)</td>
<td>95.70</td>
<td>95.98</td>
<td>95.84</td>
<td>92.03</td>
<td>92.33</td>
<td>92.18</td>
</tr>
<tr>
<td>Zhou and Zhao (2019) (XLNet)</td>
<td>96.21</td>
<td>96.46</td>
<td>96.33</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Our work</td>
<td><strong>96.24</strong></td>
<td><strong>96.53</strong></td>
<td><strong>96.38</strong></td>
<td>91.85</td>
<td><strong>93.45</strong></td>
<td><strong>92.64</strong></td>
</tr>
</tbody>
</table>

Table 3: Constituency Parsing on PTB & CTB test sets.
Image Captioning

**Input**: image

**Output**: generated text

A woman is throwing a **frisbee** in a park.

A **dog** is standing on a hardwood floor.

*Figure 3. Examples of attending to the correct object (white indicates the attended regions, underlines indicated the corresponding word)*

Image Captioning

Input: image
Output: generated text

A stop sign is on a road with a mountain in the background.

A little girl sitting on a bed with a teddy bear.

Figure 3. Examples of attending to the correct object (white indicates the attended regions, underlines indicated the corresponding word)

A large white bird standing in a forest. A woman holding a clock in her hand.

*Figure 5.* Examples of mistakes where we can use attention to gain intuition into what the model saw.
A woman is sitting at a table with a large pizza.

A person is standing on a beach with a surfboard.

Figure 5. Examples of mistakes where we can use attention to gain intuition into what the model saw.
• **LSTMs** yielded state-of-the-art results on most NLP tasks (2014-2018)

• **seq2seq+Attention** was an even more revolutionary idea (Google Translate used it)

• **Attention** allows us to place appropriate weight to the encoder’s hidden states

• But, **LSTMs** require us to iteratively scan each word and wait until we’re at the end before we can do anything
BACKUP