multi-hop attention and Transformers
Outline

Review of common (old fashioned) neural architectures

bags

Attention

Transformer
Some (historically standard) neural architectures:

- Good (neural) models have existed for some data types for a while:
Some (historically standard) neural architectures:

- Good (neural) models have existed for some data types for a while:
  - Convolutional Networks (CNN) for translation-invariant (and scale invariant/composable) grid-structured data
  - Recurrent Neural Networks (RNN) for (ordered) sequential data.
Some (historically standard) neural architectures:

- Good (neural) models have existed for some data types for a while:
  - Convolutional Networks (CNN) for translation-invariant (and scale invariant/composable) grid-structured data
  - Recurrent Neural Networks (RNN) for (ordered) sequential data.

- Less empirically successful:
  - fully connected feed-forward networks.
(Deep) fully connected feed forward nets have not been nearly as successful as their structured counterparts.
(Deep) fully connected feed forward nets have not been nearly as successful as their structured counterparts.

It’s not that they don’t work; but rather, you can almost always do something better.
Convolutional neural networks:

- The input $x_j$ has a grid structure, and $A_j$ specializes to a convolution.
- The pointwise nonlinearity is followed by a pooling operator.
- Pooling introduces invariance (on the grid) at the cost of lower resolution (on the grid).
- These have been very successful because the invariances and symmetries of the model are well adapted to the invariances and symmetries of the tasks they are used for.
Sequential networks

- Inputs come as a sequence, and the output is a sequence:

  
  - Input sequence $x_0, x_1, ..., x_n, ...$ and output sequence $y_0, y_1, ..., y_n, ...$

  \[
  \hat{y}_i = f(x_i, x_{i-1}, ..., x_0)
  \]

- Two standard strategies for dealing with growing input:
**Sequential networks**

- Inputs come as a sequence, and the output is a sequence:

  - input sequence $x_0, x_1, ..., x_n, ...$ and output sequence $y_0, y_1, ..., y_n, ...$;

  $$\hat{y}_i = f(x_i, x_{i-1}, ..., x_0)$$

- Two standard strategies for dealing with growing input:

  - fixed memory size (that is, $f(x_i, x_{i-1}, ..., x_0) = f(x_i, x_{i-1}, ..., x_{i-m})$ for some fixed, not too big $m$)
Sequential networks

- Inputs come as a sequence, and the output is a sequence:

- input sequence $x_0, x_1, ..., x_n, ...$ and output sequence $y_0, y_1, ..., y_n, ...$

  \[ \hat{y}_i = f(x_i, x_{i-1}, ..., x_0) \]

- Two standard strategies for dealing with growing input:

  - fixed memory size (that is, $f(x_i, x_{i-1}, ..., x_0) = f(x_i, x_{i-1}, ..., x_{i-m})$ for some fixed, not too big $m$)

  - recurrence
Recurrent sequential networks (Elman, Jordan)

- In equations:

- Have input sequence $x_0, x_1, ..., x_n, ...$ and output sequence $y_0, y_1, ..., y_n, ...$;

- and hidden state sequence $h_0, h_1, ..., h_n, ...$.

- the network updates

  $$ h_{i+1} = f(h_i, x_{i+1}) $$

  $$ \hat{y}_i = g(h_i), $$

  where $f$ and $g$ are (perhaps multilayer) neural networks.

- multiplicative interactions seem to be important for recurrent sequential networks (e.g. in LSTM, GRU).

- Thus recurrent nets are as deep as the length of the sequence (if written as a feed-forward network).
What to do if your input is a set (of vectors)?
Why should we want to input sets (or graphs)?

- permutation invariance
- Sparse representations of input
- Make determinations of structure at input time, rather than when building architecture
Why should we want to input sets (or graphs)?

- permutation invariance
- Sparse representations of input
- Make determinations of structure at input time, rather than when building architecture
- No choice, the input is given that way, and we really want to use a neural architecture.
Outline

Review of common (old fashioned) neural architectures

bags

Attention

Transformer
Simplest possibility: Bag of (vectors)

- Given a featurization of each element of the input set into some vector $m \in \mathbb{R}^d$, take the average:

$$\{m_1, \ldots, m_s\} \rightarrow \frac{1}{s} \sum_i m_i$$
Simplest possibility: Bag of (vectors)

- Given a featurization of each element of the input set into some vector \( m \in \mathbb{R}^d \), take the average:

\[
\{m_1, \ldots, m_s\} \rightarrow \frac{1}{s} \sum_i m_i
\]

- Use domain knowledge to pick a good featurization, and perhaps to arrange “pools” so that not all structural information from the set is lost.

- This can be surprisingly effective.
Simplest possibility: Bag of (vectors)

- Given a featurization of each element of the input set into some vector \( m \in \mathbb{R}^d \), take the average:

\[
\{m_1, \ldots, m_s\} \rightarrow \frac{1}{s} \sum_i m_i
\]

- Use domain knowledge to pick a good featurization, and perhaps to arrange “pools” so that not all structural information from the set is lost.

- This can be surprisingly effective.

- or, depending on your viewpoint, demonstrate bias in data or poorly designed tasks.
Some empirical “successes” of bags

- recommender systems (writing users as a bag of items, or items as bags of users)
- generic word embeddings (e.g. word2vec)
- success as a generic baseline in language (retrieval) tasks
“Failures” of bags:
- Convolutional nets and vision
- Usually beaten in NLP by contextualized word vectors (ELMO → BERT)
Outline

Review of common (old fashioned) neural architectures

bags

Attention

Transformer
Attention

- “Attention”: weighting or probability distribution over inputs that depends on computational state and inputs.

- Attention can be “hard”, that is, described by discrete variables, or “soft”, described by continuous variables.
Attention in vision

- Humans use attention at multiple scales (Saccades, etc...)
- Long history in computer vision [P.N. Rajesh et al., 1996; Butko et. al., 2009; Larochelle et al., 2010; Mnih et. al. 2014;]
- This is usually attention over the grid: given a machine's current state/history of glimpses, where and at what scale should it look next
Attention in NLP

- Alignment in machine translation: for each word in the target, get a distribution over words in the source [Brown et. al. 1993], (lots more)

(Figure from Latent Alignment and Variational Attention by Deng et. al.)
Attention in NLP

- Alignment in machine translation: for each word in the target, get a distribution over words in the source [Brown et. al. 1993], (lots more)

- Used differently than the vision version: optimized over, rather than focused on.

- Attention as “focusing” in NLP: [Bahdanau et. al. 2014].
Attention and bags:

- Attention can be used for dynamically weighted averages

For example in [Bahdanau et. al. 2014], $u$ is the hidden state at a given token in an LSTM.
Attention and bags:

- Attention can be used for dynamically weighted averages

\[ \{m_1, \ldots, m_n\} \rightarrow \sum_j a_j m_j \]

where \(a_j\) depends on the state of the machine and the \(m\).
Attention and bags:

- Attention can be used for dynamically weighted averages

\[ \{m_1, ..., m_n\} \rightarrow \sum_j a_j m_j \]

where \(a_j\) depends on the state of the machine and the \(m\).

- One standard approach (soft attention): state given by a vector \(u\) and

\[ a_j = \frac{e^{u^T m_j}}{\sum_j e^{u^T m_j}} \]

- For example in [Bahdanau et. al. 2014], \(u\) is the hidden state at given token in an LSTM.
attention is a “generic” computational mechanism; it allows complex processing of any “unstructured” inputs.
attention is a “generic” computational mechanism; it allows complex processing of any “unstructured” inputs.

:) 

but really,
attention is a “generic” computational mechanism; it allows complex processing of any “unstructured” inputs.

but really,

Helps solve problems with long term dependencies

deals cleanly with sparse inputs

allows practitioners to inject domain knowledge and structure at run time instead of at architecting time.
Attention for dynamically weighted bags history

- This seems to be a surprisingly new development

- for handwriting generation: [Graves, 2013] location based

- for translation: [Bahdanau et. al. 2014] content based

- more generally: [Weston et. al. 2014; Graves et. al. 2014; Vinyals 2015] content + location
“Learning to Jointly Align and Translate”

Add an attention layer to LSTM translation model
Multi-hop attention

- “hop” → “layer”

- Memory networks [Weston et. al. 2014, Sukhbaatar et. al. 2015]:
  - The network keeps a vector of state variables \( u \); and operates by sequential updates to the \( u \).
  - each update to \( u \) is modulated by attention over the input set.
  - outputs a fixed size vector
Multi-hop attention

- Fix a number of “hops” (layers) $p$, initialize $u = 0 \in \mathbb{R}^d$, $i = 0$,

- input $M = \{m_1, ..., m_N\}$, $m_j \in \mathbb{R}^d$

The memory network then operates with

1: increment $i \leftarrow i + 1$

2: set $a = \sigma(u^T M)$ ($\sigma$ is the vector softmax function)

3: update $u \leftarrow \sum_j a_j m_j$

4: if $i < p$ return to 1:, else output $u$. 
\[ a^i = \sigma(u^i \mathbf{T} \mathbf{M}) \]
\[ u^{i+1} \leftarrow \sum_j a_j^i u_j^i \]
If the inputs have an underlying geometry, can include geometric information in the weighted “bags”

Important example: for sequential data, use position encoding
- For each input $m_i$ add to it a vector $l(i)$

$l(i)$ can be fixed during training or learned
<table>
<thead>
<tr>
<th>Story (1: 1 supporting fact)</th>
<th>Support</th>
<th>Hop 1</th>
<th>Hop 2</th>
<th>Hop 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Daniel went to the bathroom.</td>
<td>0.00</td>
<td>0.00</td>
<td>0.03</td>
<td></td>
</tr>
<tr>
<td>Mary travelled to the hallway.</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>John went to the bedroom.</td>
<td>0.37</td>
<td>0.02</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>John travelled to the bathroom.</td>
<td>0.60</td>
<td>0.98</td>
<td>0.96</td>
<td></td>
</tr>
<tr>
<td>Mary went to the office.</td>
<td>0.01</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
</tr>
</tbody>
</table>

Where is John?  Answer: bathroom  Prediction: bathroom

<table>
<thead>
<tr>
<th>Story (2: 2 supporting facts)</th>
<th>Support</th>
<th>Hop 1</th>
<th>Hop 2</th>
<th>Hop 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>John dropped the milk.</td>
<td></td>
<td>0.06</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>John took the milk there.</td>
<td>yes</td>
<td>0.88</td>
<td>1.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Sandra went back to the bathroom.</td>
<td></td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>John moved to the hallway.</td>
<td>yes</td>
<td>0.00</td>
<td>0.00</td>
<td>1.00</td>
</tr>
<tr>
<td>Mary went back to the bedroom.</td>
<td></td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Where is the milk?  Answer: hallway  Prediction: hallway

<table>
<thead>
<tr>
<th>Story (16: basic induction)</th>
<th>Support</th>
<th>Hop 1</th>
<th>Hop 2</th>
<th>Hop 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brian is a frog.</td>
<td>yes</td>
<td>0.00</td>
<td>0.98</td>
<td>0.00</td>
</tr>
<tr>
<td>Lily is gray.</td>
<td></td>
<td>0.07</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Brian is yellow.</td>
<td>yes</td>
<td>0.07</td>
<td>0.00</td>
<td>1.00</td>
</tr>
<tr>
<td>Julius is green.</td>
<td></td>
<td>0.06</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Greg is a frog.</td>
<td>yes</td>
<td>0.76</td>
<td>0.02</td>
<td>0.00</td>
</tr>
</tbody>
</table>

What color is Greg?  Answer: yellow  Prediction: yellow

<table>
<thead>
<tr>
<th>Story (18: size reasoning)</th>
<th>Support</th>
<th>Hop 1</th>
<th>Hop 2</th>
<th>Hop 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>The suitcase is bigger than the chest.</td>
<td>yes</td>
<td>0.00</td>
<td>0.88</td>
<td>0.00</td>
</tr>
<tr>
<td>The box is bigger than the chocolate.</td>
<td></td>
<td>0.04</td>
<td>0.05</td>
<td>0.10</td>
</tr>
<tr>
<td>The chest is bigger than the chocolate.</td>
<td>yes</td>
<td>0.17</td>
<td>0.07</td>
<td>0.90</td>
</tr>
<tr>
<td>The chest fits inside the container.</td>
<td></td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>The chest fits inside the box.</td>
<td></td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Does the suitcase fit in the chocolate?  Answer: no  Prediction: no
(sequential) Recurrent networks for language modeling (again)

- At train time:
  - Have input sequence $x_0, x_1, ..., x_n, ...$ and output sequence $y_0 = x_1, y_1 = x_2, ...$;
  - and state sequence $h_0, h_1, ..., h_n, ...$

- the network runs via

\[ h_{i+1} = \sigma(Wh_i + Ux_{i+1}) \]

\[ \hat{y}_i = Vg(h_i), \]

- $\sigma$ is a nonlinearity, $W, U, V$ are matrices of appropriate size
(sequential) Recurrent networks for language modeling (again)

- At generation time:

- Have seed hidden state $h_0$, perhaps given by running on a seed sequence;

- Output

\[
sample \ x_{i+1} \sim \sigma(Vg(h_i)),
\]
\[
h_{i+1} = \sigma(Wh_i + Ux_{i+1})
\]
Traditional RNN (recurrent in inputs)
MemN2N
(recurrent in hops)
Outline

Review of common (old fashioned) neural architectures

bags

Attention

Transformer
Transformer [Vaswani et. al. 2017] is a multi-hop attention model that is currently state of the art in most language tasks (and in many other things)

- Has significantly superior performance compared to previous attention based architectures. Improvements:
  - Multi-query hidden-state propagation
  - Multi-head attention
  - Residual blocks
Multi-query hidden-state propagation (Transformer):

- input $M = \{m_1, ..., m_N\}, m_j \in \mathbb{R}^d$

- Fix a number of “hops” $p$, initialize $U = M$, $i = 0$,

The transformer self-attention then operates with

1: increment $i \leftarrow i + 1$

2: set $a_j = \sigma(u_j^T U)$ ($\sigma$ is the vector softmax function)

3: update $u_j \leftarrow \sum_k a_{jk} u_k$ for all $j$

4: if $i < p$ return to 1:, else output $U$. 

\[ a_j^i = \sigma(u_j^iT \mathbf{U}) \]

All \( u_k^{i+1} \) are updated \( u_k^{i+1} \leftarrow \Sigma_j a_{jk}^i u_j^i \)
Multi-head attention

- Multi-head attention combines multiple attention ‘heads’ being trained in the same way on the same data - but with different weight matrices, and yielding different values.

- Each of the $L$ attention heads yields values for each token - these values are then multiplied by trained parameters and added.
Multi-head attention

- Single head attention: given hidden state \( u = \{ u_1, ..., u_N \} \)

\[
u_j \rightarrow \sum_k a_{jk} u_k
\]

with

\[
a_{jk} = \frac{e^{u_j^T u_k}}{\sum_s e^{u_j^T u_s}}
\]

- Multi-head attention with \( L \) heads:

\[
u_j \rightarrow F \left( \begin{bmatrix} \sum_k a_{jk}^1 G_1(u_k) \\ \sum_k a_{jk}^2 G_2(u_k) \\ \vdots \\ \sum_k a_{jk}^L G_L(u_k) \end{bmatrix} \right)
\]

with

\[
a_{jk}^L = \frac{e^{u_j^T G_L(u_j)}}{\sum_s e^{u_j^T G_L(u_k)}},
\]

and \( F \) and \( G \) fully connected networks
1) Concatenate all the attention heads

\[
\begin{align*}
Z_0 & \quad Z_1 & \quad Z_2 & \quad Z_3 & \quad Z_4 & \quad Z_5 & \quad Z_6 & \quad Z_7 \\
\end{align*}
\]

2) Multiply with a weight matrix \( W^o \) that was trained jointly with the model

\[
X
\]

3) The result would be the \( Z \) matrix that captures information from all the attention heads. We can send this forward to the FFNN

\[
Z
\]
Residual connections

- Connections between non-adjacent layers (e.g., if each layer fed into the next 2 layers directly, rather than only feeding into the next layer directly, and into all subsequent layers indirectly.)

- want to be able to keep information from the original item embeddings through all the transformations.

- many shallow models interpretation
Transformer NLP dominance

- Translation
- Language modeling (GPT2, MegatronLM, ...)
- Generic sentence vectors (BERT)
- ...

but architecture is completely generic!
Transformer NLP dominance

- Translation
- Language modeling (GPT2, MegatronLM,...)
- Generic sentence vectors (BERT)
- ...
- but architecture is completely generic!
Transformers for Language modeling: GPT2

- GPT2 is a Transformer-based LM trained on next token prediction.
- It was (at publishing) boundary-pushing in number of parameters.
- It achieved state of the art results on multiple NLP tasks without fine-tuning.
On 10 June, Artificer William Collins was promoted to corporal, probably to fill a combat leadership void for the crossing of the James River. Collins’s service record does not reflect the qualities he demonstrated to earn this promotion, but he had obviously overcome some serious problems. Born in Sacketts Harbor, New York, Collins enlisted in the company in December 1853 at the age of twenty-two, and reenlisted in December 1858. Just a month before the war began in April 1861, Collins went "over the hill" and was not caught until three years later. Returned to the company on 22 March 1864, he was tried...
On 10 June, Artificer William Collins was promoted to corporal, probably to fill a combat leadership void for the crossing of the James River. Collins’s service record does not reflect the qualities he demonstrated to earn this promotion, but he had obviously overcome some serious problems. Born in Sacketts Harbor, New York, Collins enlisted in the company in December 1853 at the age of twenty-two, and reenlisted in December 1858. Just a month before the war began in April 1861, Collins went "over the hill" and was not caught until three years later. Returned to the company on 22 March 1864, he was tried in June of that year. Sentenced to be shot for desertion, he remained on active duty. Captain James J. Dall was born in Virginia in 1829 during the Revolution and joined his father in the Army in 1836. He served with the 2d Virginia Infantry until 1845, when he went to the Western Reserve Regiment; and, on his discharge, he reenlisted with Company I, 25th New York Volunteers (which in 1845 numbered about 4,500 men)…
Context (human-written): In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.
Context (human-written): In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

GPT-2: The scientist named the population, after their distinctive horn, Ovid’s Unicorn. These four-horned, silver-white unicorns were previously unknown to science. Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved. Dr. Jorge Perez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Perez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow...
Contextual embeddings

- State of the art 2017 for word embeddings (for downstream NLP tasks):
  - embed each word in isolation (e.g. word2vec)
  - trained by predicting a token based on the context, *both* left and right (Masked Language Modeling, or MLM).

- Current SOTA: the embedding depends on context.

- ELMo used the hidden state of Bidirectional LSTM:

- BERT uses transformers...
Masked Language Modeling (MLM) and BERT

- BERT style Transformers are SOTA as an input encoder for almost every NLP standard task:
  - Question Answering
  - Sentiment Analysis
  - Natural Language Inference
  - Coreference Resolution
  - ...and more
Transformers have also been used with success in many other areas:


- Fang, et. al. Scene Memory Transformer for Embodied Agents in Long-Horizon Tasks (https://arxiv.org/abs/1903.03878)

Optimization

- Batch size
- Warmup & Learning Rate schedulers
Optimization: Batch size

- recall “batch size” or “minibatch size” is \# of examples in a single gradient update
Optimization: Batch size

- Old and busted: “Minibatch size 1 is best”
  - Theoretical generalization benefits

- Faster convergence per examples seen

- New hotness: use as big a batch as possible
  - Regularization benefits empirically not important in modern settings

- GPUs + distributed processing make huge difference in wall-clock

- Many models in modern settings fail to converge with small batches (esp in Reinforcement Learning)
Optimization: Batch size

- Transformers unstable with small batches...

- Common choice for batch size is simply the largest possible with the memory/computational constraints
Learning rate (LR) is a multiplier on gradient updates: Too-low LR will cause a model to converge very slowly. Too-high LR will lead to non-converging training.

Warmup steps - gradient update steps at the beginning during which LR increases from some starting point to the maximum.

Warmup steps are usually necessary when using Transformers (unlike other models).

After maximum standard LR decay schedules apply.
Thanks!