CS 1699: Deep Learning

Modeling Sequences: Recurrent Neural Networks, Transformers

Prof. Adriana Kovashka
University of Pittsburgh
March 24, 2020
Plan for this lecture

• Recurrent neural networks
  – Basics
  – Training (backprop through time, vanishing gradient)
  – Recurrent networks with gates (GRU, LSTM)

• Applications in NLP and vision
  – Image/video captioning
  – Neural machine translation (beam search, attention)

• Transformers
  – Self-attention
  – BERT
  – Cross-modal transformers for VQA and VCR
Recurrent neural networks
Some pre-RNN captioning results

This is a picture of one sky, one road and one sheep. The gray sky is over the gray road. The gray sheep is by the gray road.

Here we see one road, one sky and one bicycle. The road is near the blue sky, and near the colorful bicycle. The colorful bicycle is within the blue sky.

This is a picture of two dogs. The first dog is near the second furry dog.

Kulkarni et al., CVPR 2011
Results with Recurrent Neural Networks

“man in black shirt is playing guitar.”

“construction worker in orange safety vest is working on road.”

“two young girls are playing with lego toy.”

“boy is doing backflip on wakeboard.”
Recurrent Networks offer a lot of flexibility:

- one to one
- one to many
- many to one
- many to many

vanilla neural networks
Recurrent Networks offer a lot of flexibility:

- **One to one**
  - Input: Image
  - Output: Single word

- **One to many**
  - Input: Image
  - Output: Sequence of words
  - Example: Image captioning
    - Image -> sequence of words

- **Many to one**
  - Input: Sequence of words
  - Output: Single image

- **Many to many**
  - Input: Sequence of words
  - Output: Sequence of images
Recurrent Networks offer a lot of flexibility:

- **one to one**
- **one to many**
- **many to one**
- **many to many**

e.g. **sentiment classification**
sequence of words -> sentiment
Recurrent Networks offer a lot of flexibility:

e.g. machine translation
seq of words -> seq of words

Andrej Karpathy
Recurrent Networks offer a lot of flexibility:

- **One to one**
- **One to many**
- **Many to one**
- **Many to many**

*E.g. video classification on frame level*
Recurrent Neural Network
Recurrent Neural Network

usually want to output a prediction at some time steps

Adapted from Andrej Karpathy
Recurrent Neural Network

We can process a sequence of vectors $\mathbf{x}$ by applying a recurrence formula at every time step:

$$ h_t = f_W(h_{t-1}, x_t) $$

new state \hspace{1cm} old state \hspace{1cm} input vector at some time step

some function with parameters $W$
Recurrent Neural Network

We can process a sequence of vectors $\mathbf{x}$ by applying a recurrence formula at every time step:

$$h_t = f_W(h_{t-1}, x_t)$$

Notice: the same function and the same set of parameters are used at every time step.
(Vanilla) Recurrent Neural Network

The state consists of a single “hidden” vector $h$:

$$h_t = f_W(h_{t-1}, x_t)$$

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$
Example

Character-level language model example

Vocabulary: [h,e,l,o]

Example training sequence: “hello”
Example

Character-level language model example

Vocabulary: [h,e,l,o]

Example training sequence: “hello”

Input layer:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Input chars: “h” “e” “l” “l”
Character-level language model example

Vocabulary: [h,e,l,o]

Example training sequence: “hello”
Character-level language model example

Vocabulary: [h,e,l,o]

Example training sequence: “hello”

What do we still need to specify, for this to work?
What kind of loss can we formulate?
Training a Recurrent Neural Network

- Get a **big corpus of text** which is a sequence of words \( x^{(1)}, \ldots, x^{(T)} \)
- Feed into RNN; compute output distribution \( \hat{y}^{(t)} \) **for every step** \( t \).
  - i.e. predict probability dist of **every word**, given words so far

- **Loss function** on step \( t \) is **cross-entropy** between predicted probability distribution \( \hat{y}^{(t)} \), and true next word \( y^{(t)} \) (one-hot); \( V \) is vocabulary

\[
J^{(t)}(\theta) = CE(y^{(t)}, \hat{y}^{(t)}) = - \sum_{w\in V} y_w^{(t)} \log \hat{y}_w^{(t)} = - \log \hat{y}_{x_{t+1}}^{(t)}
\]

- Average this to get **overall loss** for entire training set:

\[
J(\theta) = \frac{1}{T} \sum_{t=1}^{T} J^{(t)}(\theta) = \frac{1}{T} \sum_{t=1}^{T} - \log \hat{y}_{x_{t+1}}^{(t)}
\]

Adapted from Abigail See
The vanishing/exploding gradient problem

- The error at a time step ideally can tell a previous time step from many steps away to change during backprop
- Multiply the same matrix at each time step during backprop
The vanishing gradient problem

- Total error is the sum of each error at time steps $t$
  \[
  \frac{\partial E}{\partial W} = \sum_{t=1}^{T} \frac{\partial E_t}{\partial W}
  \]

- Chain rule:
  \[
  \frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W}
  \]

- More chain rule:
  \[
  \frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_j-1}
  \]

- Derivative of vector wrt vector is a Jacobian matrix of partial derivatives; norm of this matrix can become very small or very large quickly [Bengio et al. 1994, Pascanu et al. 2013], leading to vanishing/exploding gradient

Adapted from Richard Socher
What uses of language models from everyday life can you think of?
Now in more detail…
Language Modeling

- **Language Modeling** is the task of predicting what word comes next.

\[
\text{the students opened their } \underline{\text{_____}} \]

- More formally: given a sequence of words \( x^{(1)}, x^{(2)}, \ldots, x^{(t)} \), compute the probability distribution of the next word \( x^{(t+1)} \):

\[
P(x^{(t+1)} \mid x^{(t)}, \ldots, x^{(1)})
\]

where \( x^{(t+1)} \) can be any word in the vocabulary \( V = \{ w_1, \ldots, w_{|V|} \} \)

- A system that does this is called a **Language Model**.
n-gram Language Models

• First we make a simplifying assumption: $x^{(t+1)}$ depends only on the preceding $n$-1 words.

$$P(x^{(t+1)} | x^{(t)}, \ldots, x^{(1)}) = P(x^{(t+1)} | x^{(t)}, \ldots, x^{(t-n+2)})$$  \hspace{1cm} (assumption)

• **Question:** How do we get these $n$-gram and $(n-1)$-gram probabilities?

• **Answer:** By counting them in some large corpus of text!

$$\approx \frac{\text{count}(x^{(t+1)}, x^{(t)}, \ldots, x^{(t-n+2)})}{\text{count}(x^{(t)}, \ldots, x^{(t-n+2)})}$$  \hspace{1cm} (statistical approximation)
**Sparsity Problems with n-gram Language Models**

**Sparsity Problem 1**

**Problem:** What if “students opened their $w$” never occurred in data? Then $w$ has probability 0!

**Partial Solution:** Add small $\delta$ to the count for every $w \in V$. This is called *smoothing*.

$$P(w|\text{students opened their}) = \frac{\text{count(students opened their } w)}{\text{count(students opened their)}}$$

**Sparsity Problem 2**

**Problem:** What if “students opened their” never occurred in data? Then we can’t calculate probability for any $w$!

**Partial Solution:** Just condition on “opened their” instead. This is called *backoff*.

**Note:** Increasing $n$ makes sparsity problems worse. Typically we can’t have $n$ bigger than 5.
A fixed-window neural Language Model

output distribution
\[ \hat{y} = \text{softmax}(Uh + b_2) \in \mathbb{R}^{|V|} \]

hidden layer
\[ h = f(We + b_1) \]

concatenated word embeddings
\[ e = [e^{(1)}; e^{(2)}; e^{(3)}; e^{(4)}] \]

words / one-hot vectors
\[ x^{(1)}, x^{(2)}, x^{(3)}, x^{(4)} \]
A fixed-window neural Language Model

**Improvements** over $n$-gram LM:
- No sparsity problem
- Don’t need to store all observed $n$-grams

Remaining **problems**:
- Fixed window is too small
- Enlarging window enlarges $W$
- Window can never be large enough!
- $\mathbf{x}^{(1)}$ and $\mathbf{x}^{(2)}$ are multiplied by completely different weights in $W$. **No symmetry** in how the inputs are processed.

We need a neural architecture that can process *any length input*
Recurrent Neural Networks (RNN)
A family of neural architectures

Core idea: Apply the same weights $W$ repeatedly

outputs (optional) \{ 

hidden states \{ 

input sequence (any length) \{ 

Abigail See
A RNN Language Model

output distribution
\[ \hat{y}(t) = \text{softmax} \left( U h^{(t)} + b_2 \right) \in \mathbb{R}^{V|} \]

hidden states
\[ h^{(t)} = \sigma \left( W_h h^{(t-1)} + W_e e^{(t)} + b_1 \right) \]
\[ h^{(0)} \text{ is the initial hidden state} \]

word embeddings
\[ e^{(t)} = E x^{(t)} \]

words / one-hot vectors
\[ x^{(t)} \in \mathbb{R}^{V|} \]

Note: this input sequence could be much longer, but this slide doesn’t have space!
A RNN Language Model

**RNN Advantages:**
- Can process any length input
- Computation for step $t$ can (in theory) use information from many steps back
- Model size doesn’t increase for longer input
- Same weights applied on every timestep, so there is symmetry in how inputs are processed.

**RNN Disadvantages:**
- Recurrent computation is slow
- In practice, difficult to access information from many steps back

Abigail See
Recall: Training a RNN Language Model

- Get a big corpus of text which is a sequence of words \( x^{(1)}, \ldots, x^{(T)} \)
- Feed into RNN-LM; compute output distribution \( \hat{y}^{(t)} \) for every step \( t \).
  - i.e. predict probability dist of every word, given words so far

- Loss function on step \( t \) is cross-entropy between predicted probability distribution \( \hat{y}^{(t)} \), and the true next word \( y^{(t)} \) (one-hot for \( x^{(t+1)} \)):
  \[
  J^{(t)}(\theta) = CE(y^{(t)}, \hat{y}^{(t)}) = - \sum_{w \in V} y_w^{(t)} \log \hat{y}_w^{(t)} = - \log \hat{y}_{x_{t+1}}^{(t)}
  \]

- Average this to get overall loss for entire training set:
  \[
  J(\theta) = \frac{1}{T} \sum_{t=1}^{T} J^{(t)}(\theta) = \frac{1}{T} \sum_{t=1}^{T} - \log \hat{y}_{x_{t+1}}^{(t)}
  \]
Training a RNN Language Model

= negative log prob of “students”

Loss → $J^{(1)}(\theta) → \hat{y}^{(1)}$ → $U$ → $W_h$ → $W_e$ → $e^{(1)}$ → $E$ → $x^{(1)}$ → the

Predicted prob dists

Corpus → $x^{(1)}$ → students → opened → their → exams → ...
Training a RNN Language Model

Loss $\rightarrow J^{(1)}(\theta) \rightarrow \hat{y}^{(1)} \rightarrow \hat{y}^{(2)} \rightarrow \hat{y}^{(3)} \rightarrow \hat{y}^{(4)} \rightarrow ...$

Predicted prob dists $\rightarrow U \rightarrow U \rightarrow U \rightarrow U \rightarrow U \rightarrow ...$

Corpus $\rightarrow x^{(1)} \rightarrow \text{the} \rightarrow ...$

= negative log prob of "opened"
Training a RNN Language Model

Loss → $J^{(1)}(\theta) → \hat{y}^{(1)} → U → h^{(1)} → W_h → e^{(1)} → E → the$

$J^{(2)}(\theta) → \hat{y}^{(2)} → U → h^{(2)} → W_h → e^{(2)} → E → students$

$J^{(3)}(\theta) → \hat{y}^{(3)} → U → h^{(3)} → W_h → e^{(3)} → E → opened$

$J^{(4)}(\theta) → \hat{y}^{(4)} → U → h^{(4)} → W_h → e^{(4)} → E → their$

$... 

Loss = negative log prob of “their”

Corpus$ → \mathbf{x}^{(1)}$, $\mathbf{x}^{(2)}$, $\mathbf{x}^{(3)}$, $\mathbf{x}^{(4)}$, $\mathbf{x}^{(5)}$, $...$

Abigail See
Training a RNN Language Model

Loss \rightarrow J^{(1)}(\theta) \rightarrow \hat{y}^{(1)} \rightarrow U \rightarrow W_h \rightarrow e^{(1)} \rightarrow E \rightarrow \text{the} \rightarrow x^{(1)}

J^{(2)}(\theta) \rightarrow \hat{y}^{(2)} \rightarrow U \rightarrow W_h \rightarrow e^{(2)} \rightarrow E \rightarrow \text{students} \rightarrow x^{(2)}

J^{(3)}(\theta) \rightarrow \hat{y}^{(3)} \rightarrow U \rightarrow W_h \rightarrow e^{(3)} \rightarrow E \rightarrow \text{opened} \rightarrow x^{(3)}

J^{(4)}(\theta) \rightarrow \hat{y}^{(4)} \rightarrow U \rightarrow W_h \rightarrow e^{(4)} \rightarrow E \rightarrow \text{their} \rightarrow x^{(4)} \rightarrow \text{exams} \rightarrow \ldots

= \text{negative log prob of “exams”}

Predicted prob dists

Corpus

Abigail See
Training a RNN Language Model

\[
J(\theta) = \frac{1}{T} \sum_{t=1}^{T} J(t)(\theta)
\]

Loss \rightarrow \sum_{t=1}^{T} J(t)(\theta) \rightarrow \hat{y}^{(1)} \rightarrow \hat{y}^{(2)} \rightarrow \hat{y}^{(3)} \rightarrow \hat{y}^{(4)} \rightarrow \ldots

Predicted prob dists

\begin{align*}
&h^{(0)} \\
&\rightarrow W_h \rightarrow h^{(1)} \\
&\rightarrow W_h \rightarrow h^{(2)} \\
&\rightarrow W_h \rightarrow h^{(3)} \\
&\rightarrow W_h \rightarrow h^{(4)} \\
&\rightarrow W_h \rightarrow \ldots
\end{align*}

Corpus

\begin{align*}
&x^{(1)} \rightarrow E \rightarrow e^{(1)} \\
&x^{(2)} \rightarrow E \rightarrow e^{(2)} \\
&x^{(3)} \rightarrow E \rightarrow e^{(3)} \\
&x^{(4)} \rightarrow E \rightarrow e^{(4)} \\
&\ldots
\end{align*}

the students opened their exams

Abigail See
Training a RNN Language Model

• However: Computing loss and gradients across entire corpus \( x^{(1)}, \ldots, x^{(T)} \) is too expensive!

\[
J(\theta) = \frac{1}{T} \sum_{t=1}^{T} J^{(t)}(\theta)
\]

• In practice, consider \( x^{(1)}, \ldots, x^{(T)} \) as a sentence (or a document)

• **Recall:** Stochastic Gradient Descent allows us to compute loss and gradients for small chunk of data, and update.

• Compute loss \( J(\theta) \) for a sentence (actually a batch of sentences), compute gradients and update weights. Repeat.
Backpropagation for RNNs

**Question:** What’s the derivative of \( J^{(t)}(\theta) \) w.r.t. the repeated weight matrix \( W_h \)?

**Answer:**
\[
\frac{\partial J^{(t)}}{\partial W_h} = \sum_{i=1}^{t} \left. \frac{\partial J^{(t)}}{\partial W_h} \right|_{(i)}
\]

“The gradient w.r.t. a repeated weight is the sum of the gradient w.r.t. each time it appears”

Why?

Abigail See
Multivariable Chain Rule

- Given a multivariable function \( f(x, y) \), and two single variable functions \( x(t) \) and \( y(t) \), here's what the multivariable chain rule says:

\[
\frac{d}{dt} f(x(t), y(t)) = \frac{\partial f}{\partial x} \frac{dx}{dt} + \frac{\partial f}{\partial y} \frac{dy}{dt}
\]

Derivative of composition function

Source:
Backpropagation for RNNs: Proof sketch

- Given a multivariable function $f(x, y)$, and two single variable functions $x(t)$ and $y(t)$, here's what the multivariable chain rule says:

$$
\frac{d}{dt} f(x(t), y(t)) = \frac{\partial f}{\partial x} \frac{dx}{dt} + \frac{\partial f}{\partial y} \frac{dy}{dt}
$$

Derivative of composition function

In our example:

Apply the multivariable chain rule:

$$
\frac{\partial J^{(t)}}{\partial W_h} = \sum_{i=1}^{t} \frac{\partial J^{(t)}}{\partial W_h} \bigg|_{(i)} \frac{\partial W_h}{\partial W_h} \bigg|_{(i)} = 1
$$

Source:

Abigail See
Backpropagation for RNNs

Question: How do we calculate this?

Answer: Backpropagate over timesteps $i=t,\ldots,0$, summing gradients as you go. This algorithm is called “backpropagation through time”.

$$\frac{\partial J^{(t)}}{\partial W_h} = \sum_{i=1}^{t} \frac{\partial J^{(t)}}{\partial W_h} \bigg|_{(i)}$$

Abigail See
Vanishing gradient intuition
Vanishing gradient intuition

\[
\frac{\partial J^{(4)}}{\partial h^{(1)}} = ?
\]

Abigail See
Vanishing gradient intuition

$$\frac{\partial J^{(4)}}{\partial h^{(1)}} = \frac{\partial h^{(2)}}{\partial h^{(1)}} \times \frac{\partial J^{(4)}}{\partial h^{(2)}}$$

chain rule!
Vanishing gradient intuition

\[
\frac{\partial J^{(4)}}{\partial h^{(1)}} = \frac{\partial h^{(2)}}{\partial h^{(1)}} \times \frac{\partial h^{(3)}}{\partial h^{(2)}} \times \frac{\partial J^{(4)}}{\partial h^{(3)}}
\]

chain rule!
Vanishing gradient intuition

\[ \frac{\partial J^{(4)}}{\partial h^{(1)}} = \frac{\partial h^{(2)}}{\partial h^{(1)}} \times \frac{\partial h^{(3)}}{\partial h^{(2)}} \times \frac{\partial h^{(4)}}{\partial h^{(3)}} \times \frac{\partial J^{(4)}}{\partial h^{(4)}} \]

chain rule!
Vanishing gradient intuition

What happens if these are small?

Vanishing gradient problem:
When these are small, the gradient signal gets smaller and smaller as it backpropagates further.
Vanishing gradient proof sketch

- Recall:
  \[ h^{(t)} = \sigma \left( W_h h^{(t-1)} + W_x x^{(t)} + b_1 \right) \]

- Therefore:
  \[ \frac{\partial h^{(t)}}{\partial h^{(t-1)}} = \text{diag} \left( \sigma' \left( W_h h^{(t-1)} + W_x x^{(t)} + b_1 \right) \right) W_h \] (chain rule)

- Consider the gradient of the loss \( J^{(i)}(\theta) \) on step \( i \), with respect to the hidden state \( h^{(j)} \) on some previous step \( j \).

\[
\frac{\partial J^{(i)}(\theta)}{\partial h^{(j)}} = \frac{\partial J^{(i)}(\theta)}{\partial h^{(i)}} \prod_{j < t \leq i} \frac{\partial h^{(t)}}{\partial h^{(t-1)}}
\]

\[
= \frac{\partial J^{(i)}(\theta)}{\partial h^{(i)}} W_h^{(i-j)} \prod_{j < t \leq i} \text{diag} \left( \sigma' \left( W_h h^{(t-1)} + W_x x^{(t)} + b_1 \right) \right)
\]

(value of \( \frac{\partial h^{(t)}}{\partial h^{(t-1)}} \))

If \( W_h \) is small, then this term gets vanishingly small as \( i \) and \( j \) get further apart.

Vanishing gradient proof sketch

• Consider matrix L2 norms:

\[
\left\| \frac{\partial J^{(i)}(\theta)}{\partial h^{(j)}} \right\| \leq \left\| \frac{\partial J^{(i)}(\theta)}{\partial h^{(i)}} \right\| \|W_h\|^{(i-j)} \prod_{j < t \leq i} \|\text{diag} \left( \sigma' \left( W_h h^{(t-1)} + W_x x^{(t)} + b_1 \right) \right)\|
\]

• Pascanu et al showed that if the largest eigenvalue of \(W_h\) is less than 1, then the gradient \(\left\| \frac{\partial J^{(i)}(\theta)}{\partial h^{(j)}} \right\|\) will shrink exponentially
  • Here the bound is 1 because we have sigmoid nonlinearity

• There’s a similar proof relating a largest eigenvalue >1 to exploding gradients

Why is vanishing gradient a problem?

Gradient signal from faraway is lost because it’s much smaller than gradient signal from close-by.

So model weights are only updated only with respect to near effects, not long-term effects.
Effect of vanishing gradient on RNN-LM

- **LM task:** *When she tried to print her tickets, she found that the printer was out of toner. She went to the stationery store to buy more toner. It was very overpriced. After installing the toner into the printer, she finally printed her ________*

- To learn from this training example, the RNN-LM needs to model the dependency between “tickets” on the 7th step and the target word “tickets” at the end.

- But if gradient is small, the model can’t learn this dependency
  - So the model is unable to predict similar long-distance dependencies at test time
Effect of vanishing gradient on RNN-LM

• **LM task:** *The writer of the books ___*

• **Correct answer:** *The writer of the books *is* planning a sequel*

• **Syntactic recency:** *The writer of the books *is* (correct)*

• **Sequential recency:** *The writer of the books *are* (incorrect)*

• Due to vanishing gradient, RNN-LMs are better at learning from sequential recency than syntactic recency, so they make this type of error more often than we’d like [Linzen et al 2016]

Why is *exploding* gradient a problem?

- If the gradient becomes too big, then the SGD update step becomes too big:

  \[ \theta_{\text{new}} = \theta_{\text{old}} - \alpha \nabla_{\theta} J(\theta) \]

- This can cause **bad updates**: we take too large a step and reach a bad parameter configuration (with large loss)

- In the worst case, this will result in **Inf** or **NaN** in your network (then you have to restart training from an earlier checkpoint)
Gradient clipping: solution for exploding gradient

- **Gradient clipping**: if the norm of the gradient is greater than some threshold, scale it down before applying SGD update.

\[
\hat{g} \leftarrow \frac{\partial E}{\partial \theta} \\
\text{if } \|\hat{g}\| \geq \text{threshold} \text{ then} \\
\hat{g} \leftarrow \frac{\text{threshold}}{\|\hat{g}\|} \hat{g} \\
\text{end if}
\]

- **Intuition**: take a step in the same direction, but a smaller step.

RNNs with Gates
How to fix vanishing gradient problem?

• The main problem is that it’s too difficult for the RNN to learn to preserve information over many timesteps.

• In a vanilla RNN, the hidden state is constantly being rewritten

\[ h^{(t)} = \sigma \left( W_h h^{(t-1)} + W_x x^{(t)} \right) \]

• How about a RNN with separate memory?
Gated Recurrent Units (GRUs)

• More complex hidden unit computation in recurrence!

• Introduced by Cho et al. 2014

• Main ideas:
  • keep around memories to capture long distance dependencies
  • allow error messages to flow at different strengths depending on the inputs
Gated Recurrent Units (GRUs)

• Standard RNN computes hidden layer at next time step directly:
  \[ h_t = f \left( W^{(hh)} h_{t-1} + W^{(hx)} x_t \right) \]

• GRU first computes an update gate (another layer) based on current input word vector and hidden state
  \[ z_t = \sigma \left( W^{(z)} x_t + U^{(z)} h_{t-1} \right) \]

• Compute reset gate similarly but with different weights
  \[ r_t = \sigma \left( W^{(r)} x_t + U^{(r)} h_{t-1} \right) \]
Gated Recurrent Units (GRUs)

- **Update gate**
  \[ z_t = \sigma \left( W^{(z)} x_t + U^{(z)} h_{t-1} \right) \]

- **Reset gate**
  \[ r_t = \sigma \left( W^{(r)} x_t + U^{(r)} h_{t-1} \right) \]

- **New memory content:**
  \[ \tilde{h}_t = \tanh (W x_t + r_t \circ U \tilde{h}_{t-1}) \]
  If reset gate unit is \( \sim 0 \), then this ignores previous memory and only stores the new word information.

- **Final memory at time step combines current and previous time steps:**
  \[ h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \tilde{h}_t \]
Gated Recurrent Units (GRUs)

\[
\begin{align*}
    z_t &= \sigma \left( W^{(z)} x_t + U^{(z)} h_{t-1} \right) \\
    r_t &= \sigma \left( W^{(r)} x_t + U^{(r)} h_{t-1} \right) \\
    \tilde{h}_t &= \tanh \left( W x_t + r_t \circ U h_{t-1} \right) \\
    h_t &= z_t \circ h_{t-1} + (1 - z_t) \circ \tilde{h}_t
\end{align*}
\]
Gated Recurrent Units (GRUs)

- If reset $r$ is close to 0, ignore previous hidden state: Allows model to drop information that is irrelevant in the future

\[
z_t = \sigma \left( W^{(z)} x_t + U^{(z)} h_{t-1} \right)
\]

\[
r_t = \sigma \left( W^{(r)} x_t + U^{(r)} h_{t-1} \right)
\]

\[
\tilde{h}_t = \tanh \left( W x_t + r_t \circ U h_{t-1} \right)
\]

\[
h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \tilde{h}_t
\]

- If update $z$ is close to 1, can copy information through many time steps, i.e. copy paste state: Less vanishing gradient!

- Units with short-term dependencies often have reset gates $(r)$ very active; ones with long-term dependencies have active update gates $(z)$
Long-short-term-memories (LSTMs)

• Proposed by Hochreiter and Schmidhuber in 1997

• We can make the units even more complex

• Allow each time step to modify
  • Input gate (current cell matters) \(i_t = \sigma \left(W^{(i)} x_t + U^{(i)} h_{t-1} \right)\)
  • Forget (gate 0, forget past) \(f_t = \sigma \left(W^{(f)} x_t + U^{(f)} h_{t-1} \right)\)
  • Output (how much cell is exposed) \(o_t = \sigma \left(W^{(o)} x_t + U^{(o)} h_{t-1} \right)\)
  • New memory cell \(\tilde{c}_t = \tanh \left(W^{(c)} x_t + U^{(c)} h_{t-1} \right)\)
  • Final memory cell: \(c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t\)
  • Final hidden state: \(h_t = o_t \circ \tanh(c_t)\)

Adapted from Richard Socher
Long-short-term-memories (LSTMs)

Intuition: memory cells can keep information intact, unless inputs make them forget it or overwrite it with new input.

Cell can decide to output this information or just store it.

Richard Socher, figure from wildml.com
Review on your own: Gated Recurrent Units (GRU)

- Proposed by Cho et al. in 2014 as a simpler alternative to the LSTM.
- On each timestep \( t \) we have input \( x(t) \) and hidden state \( h(t) \) (no cell state).

**Update gate:** controls what parts of hidden state are updated vs preserved

\[
u(t) = \sigma \left( W_u h(t-1) + U_u x(t) + b_u \right)
\]

**Reset gate:** controls what parts of previous hidden state are used to compute new content

\[
r(t) = \sigma \left( W_r h(t-1) + U_r x(t) + b_r \right)
\]

**New hidden state content:** reset gate selects useful parts of prev hidden state. Use this and current input to compute new hidden content.

\[
\tilde{h}(t) = \tanh \left( W_h (r(t) \circ h(t-1)) + U_h x(t) + b_h \right)
\]

\[
h(t) = (1 - u(t)) \circ h(t-1) + u(t) \circ \tilde{h}(t)
\]

**Hidden state:** update gate simultaneously controls what is kept from previous hidden state, and what is updated to new hidden state content

**How does this solve vanishing gradient?**
GRU makes it easier to retain info long-term (e.g. by setting update gate to 0)


Abigail See
We have a sequence of inputs $\mathbf{x}^{(t)}$, and we will compute a sequence of hidden states $\mathbf{h}^{(t)}$ and cell states $\mathbf{c}^{(t)}$. On timestep $t$:

**Forget gate:** controls what is kept vs forgotten, from previous cell state

**Input gate:** controls what parts of the new cell content are written to cell

**Output gate:** controls what parts of cell are output to hidden state

**New cell content:** this is the new content to be written to the cell

**Cell state:** erase ("forget") some content from last cell state, and write ("input") some new cell content

**Hidden state:** read ("output") some content from the cell

### Sigmoid function: all gate values are between 0 and 1

For timestep $t$:

- $f^{(t)} = \sigma \left( W_f h^{(t-1)} + U_f \mathbf{x}^{(t)} + b_f \right)$
- $i^{(t)} = \sigma \left( W_i h^{(t-1)} + U_i \mathbf{x}^{(t)} + b_i \right)$
- $o^{(t)} = \sigma \left( W_o h^{(t-1)} + U_o \mathbf{x}^{(t)} + b_o \right)$

- $\tilde{c}^{(t)} = \tanh \left( W_c h^{(t-1)} + U_c \mathbf{x}^{(t)} + b_c \right)$
- $c^{(t)} = f^{(t)} \circ c^{(t-1)} + i^{(t)} \circ \tilde{c}^{(t)}$
- $h^{(t)} = o^{(t)} \circ \tanh c^{(t)}$

Abigail See
Review on your own: Long Short-Term Memory (LSTM)

You can think of the LSTM equations visually like this:

Source: [http://colah.github.io/posts/2015-08-Understanding-LSTMs/](http://colah.github.io/posts/2015-08-Understanding-LSTMs/)
LSTM vs GRU

- Researchers have proposed many gated RNN variants, but LSTM and GRU are the most widely-used.

- The biggest difference is that GRU is quicker to compute and has fewer parameters.

- There is no conclusive evidence that one consistently performs better than the other.

- LSTM is a good default choice (especially if your data has particularly long dependencies, or you have lots of training data).

- **Rule of thumb**: start with LSTM, but switch to GRU if you want something more efficient.
LSTMs: real-world success

• In 2013-2015, LSTMs started achieving state-of-the-art results
  • Successful tasks include: handwriting recognition, speech recognition, machine translation, parsing, image captioning
  • LSTM became the dominant approach

• Now (2019), other approaches (e.g. Transformers) have become more dominant for certain tasks.
  • For example in WMT (a MT conference + competition):
    • In WMT 2016, the summary report contains ”RNN” 44 times
    • In WMT 2018, the report contains “RNN” 9 times and “Transformer” 63 times

Is vanishing/exploding gradient just a RNN problem?

- No! It can be a problem for all neural architectures (including feed-forward and convolutional), especially deep ones.
  - Due to chain rule / choice of nonlinearity function, gradient can become vanishingly small as it backpropagates
  - Thus lower layers are learnt very slowly (hard to train)
  - Solution: lots of new deep feedforward/convolutional architectures that add more direct connections (thus allowing the gradient to flow)

For example:
- Residual connections aka “ResNet”
- Also known as skip-connections
- The identity connection preserves information by default
- This makes deep networks much easier to train

Figure 2. Residual learning: a building block.

Is vanishing/exploding gradient just a RNN problem?

• No! It can be a problem for all neural architectures (including feed-forward and convolutional), especially deep ones.
  • Due to chain rule / choice of nonlinearity function, gradient can become vanishingly small as it backpropagates
  • Thus lower layers are learnt very slowly (hard to train)
  • Solution: lots of new deep feedforward/convolutional architectures that add more direct connections (thus allowing the gradient to flow)

For example:
• Dense connections aka “DenseNet”
• Directly connect everything to everything!

Bidirectional RNNs: motivation

Task: Sentiment Classification

We can regard this hidden state as a representation of the word “terribly” in the context of this sentence. We call this a contextual representation.

Sentence encoding

We can regard this hidden state as a representation of the word “terribly” in the context of this sentence. We call this a contextual representation.

These contextual representations only contain information about the left context (e.g. “the movie was”).

What about right context?

In this example, “exciting” is in the right context and this modifies the meaning of “terribly” (from negative to positive)

Abigail See
Bidirectional RNNs

This contextual representation of “terribly” has both left and right context!

Abigail See
Bidirectional RNNs

On timestep $t$:

Forward RNN

$$\overrightarrow{h}(t) = \text{RNN}_{FW}(\overrightarrow{h}(t-1), x(t))$$

Backward RNN

$$\overleftarrow{h}(t) = \text{RNN}_{BW}(\overleftarrow{h}(t+1), x(t))$$

Concatenated hidden states

$$h(t) = [\overrightarrow{h}(t); \overleftarrow{h}(t)]$$

This is a general notation to mean “compute one forward step of the RNN” – it could be a vanilla, LSTM or GRU computation.

We regard this as “the hidden state” of a bidirectional RNN. This is what we pass on to the next parts of the network.

Generally, these two RNNs have separate weights.
Multi-layer RNNs

The hidden states from RNN layer $i$ are the inputs to RNN layer $i+1$
Evaluating Language Models

- The standard evaluation metric for Language Models is perplexity.

$$\text{perplexity} = \prod_{t=1}^{T} \left( \frac{1}{P_{LM}(x^{(t+1)} | x^{(t)}, \ldots, x^{(1)})} \right)^{1/T}$$

Normalized by number of words

Inverse probability of corpus, according to Language Model

- This is equal to the exponential of the cross-entropy loss $J(\theta)$:

$$\prod_{t=1}^{T} \left( \frac{1}{\hat{y}_{x_{t+1}}^{(t)}} \right)^{1/T} = \exp \left( \frac{1}{T} \sum_{t=1}^{T} - \log \hat{y}_{x_{t+1}}^{(t)} \right) = \exp(J(\theta))$$

Lower perplexity is better!
Recap thus far

- **Language Model**: A system that predicts the next word

- **Recurrent Neural Network**: A family of neural networks that:
  - Take sequential input of any length
  - Apply the same weights on each step
  - Can optionally produce output on each step

- **Vanishing gradient problem**: what it is, why it happens, and why it’s bad for RNNs

- **LSTMs and GRUs**: more complicated RNNs that use gates to control information flow; they are more resilient to vanishing gradients
Plan for this lecture

- Recurrent neural networks
  - Basics
  - Training (backprop through time, vanishing gradient)
  - Recurrent networks with gates (GRU, LSTM)
- Applications in NLP and vision
  - Image/video captioning
  - Neural machine translation (beam search, attention)
- Transformers
  - Self-attention
  - BERT
  - Cross-modal transformers for VQA and VCR
Applications
Why should we care about Language Modeling?

- Language Modeling is a **benchmark task** that helps us **measure our progress** on understanding language.

- Language Modeling is a **subcomponent** of many NLP tasks, especially those involving **generating text** or **estimating the probability of text**:
  - Predictive typing
  - Speech recognition
  - Handwriting recognition
  - Spelling/grammar correction
  - Authorship identification
  - Machine translation
  - Summarization
  - Dialogue
  - etc.

Abigail See
Generating text with a RNN Language Model

You can use a RNN Language Model to generate text by repeated sampling. Sampled output is next step’s input.

Abigail See
Generating text with a RNN Language Model

• Let’s have some fun!
• You can train a RNN-LM on any kind of text, then generate text in that style.
• RNN-LM trained on Obama speeches:

The United States will step up to the cost of a new challenges of the American people that will share the fact that we created the problem. They were attacked and so that they have to say that all the task of the final days of war that I will not be able to get this done.

Source: https://medium.com/@samim/obama-rnn-machine-generated-political-speeches-c8abd18a2ea0

Abigail See
Generating text with a RNN Language Model

- Let’s have some fun!
- You can train a RNN-LM on any kind of text, then generate text in that style.
- RNN-LM trained on *Harry Potter*:

  “Sorry,” Harry shouted, panicking—“I’ll leave those brooms in London, are they?”

  “No idea,” said Nearly Headless Nick, casting low close by Cedric, carrying the last bit of treacle Charms, from Harry’s shoulder, and to answer him the common room perched upon it, four arms held a shining knob from when the spider hadn’t felt it seemed. He reached the teams too.

Generating text with a RNN Language Model

- Let’s have some fun!
- You can train a RNN-LM on any kind of text, then generate text in that style.
- RNN-LM trained on paint color names:

```
<table>
<thead>
<tr>
<th>Color Name</th>
<th>Hex Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ghasty Pink 231 137 165</td>
<td>231 137 165</td>
</tr>
<tr>
<td>Power Gray 151 124 112</td>
<td>151 124 112</td>
</tr>
<tr>
<td>Navel Tan 199 173 140</td>
<td>199 173 140</td>
</tr>
<tr>
<td>Bock Coe White 221 215 236</td>
<td>221 215 236</td>
</tr>
<tr>
<td>Horble Gray 178 181 196</td>
<td>178 181 196</td>
</tr>
<tr>
<td>Homestar Brown 133 104 85</td>
<td>133 104 85</td>
</tr>
<tr>
<td>Snader Brown 144 106 74</td>
<td>144 106 74</td>
</tr>
<tr>
<td>Golder Craam 237 217 177</td>
<td>237 217 177</td>
</tr>
<tr>
<td>Hurky White 232 223 215</td>
<td>232 223 215</td>
</tr>
<tr>
<td>Burf Pink 223 173 179</td>
<td>223 173 179</td>
</tr>
<tr>
<td>Rose Hork 230 215 198</td>
<td>230 215 198</td>
</tr>
<tr>
<td>Sand Dan 201 172 143</td>
<td>201 172 143</td>
</tr>
<tr>
<td>Grade Bat 48 94 83</td>
<td>48 94 83</td>
</tr>
<tr>
<td>Light Of Blast 175 150 147</td>
<td>175 150 147</td>
</tr>
<tr>
<td>Grass Bat 176 99 108</td>
<td>176 99 108</td>
</tr>
<tr>
<td>Sindis Poop 204 205 194</td>
<td>204 205 194</td>
</tr>
<tr>
<td>Dope 219 209 179</td>
<td>219 209 179</td>
</tr>
<tr>
<td>Testing 156 101 106</td>
<td>156 101 106</td>
</tr>
<tr>
<td>Stoner Blue 152 165 159</td>
<td>152 165 159</td>
</tr>
<tr>
<td>Burble Simp 226 181 132</td>
<td>226 181 132</td>
</tr>
<tr>
<td>Stanky Bean 197 162 171</td>
<td>197 162 171</td>
</tr>
<tr>
<td>Turdly 190 164 116</td>
<td>190 164 116</td>
</tr>
</tbody>
</table>
```

This is an example of a character-level RNN-LM (predicts what character comes next)

Generating poetry with RNNs

Sonnet 116 – Let me not ...

by William Shakespeare

Let me not to the marriage of true minds
Admit impediments. Love is not love
Which alters when it alteration finds,
Or bends with the remover to remove:
O no! it is an ever-fixed mark
That looks on tempests and is never shaken;
It is the star to every wandering bark,
Whose worth's unknown, although his height be taken.
Love's not Time's fool, though rosy lips and cheeks
Within his bending sickle's compass come:
Love alters not with his brief hours and weeks,
But bears it out even to the edge of doom.
If this be error and upon me proved,
I never writ, nor no man ever loved.
Generating poetry with RNNs

at first:

```
tyntd-iafhatawiaohrdemot lytdws e ,tfti, astai f ogho eoase rrranbyne 'nhthnee e
plia tklergd t o idoe ns,smtt h ne etie h,hregtrs nigtkke,aoenms lng
```

train more

"Tmont thithey" fomesscerliund
Keushey. Thom here
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwy fil on aseterlome
coaniogennnc Phe lism thond hon at. MeiDimorotion in ther thize."

train more

Aftair fall unsuch that the hall for Prince Velzonski's that me of
her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort
how, and Gogition is so overelical and ofter.

train more

"Why do what that day," replied Natasha, and wishing to himself the fact the
princess, Princess Mary was easier, fed in had oftened him.
Pierre aking his soul came to the packs and drove up his father-in-law women.

More info: [http://karpathy.github.io/2015/05/21/rnn-effectiveness/](http://karpathy.github.io/2015/05/21/rnn-effectiveness/)
Generating poetry with RNNs

PANDARUS:
Alas, I think he shall be come approached and the day
When little strain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:
They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:
Well, your wit is in the care of side and that.

Second Lord:
They would be ruled after this chamber, and
my fair nues begun out of the fact, to be conveyed,
Whose noble souls I’ll have the heart of the wars.

Clown:
Come, sir, I will make did behold your worship.

VIOLA:
I’ll drink it.

VIOLA:
Why, Salisbury must find his flesh and thought
That which I am not aps, not a man and in fire,
To show the reining of the raven and the wars
To grace my hand reproach within, and not a fair are hand,
That Caesar and my goodly father's world;
When I was heaven of presence and our fleets,
We spare with hours, but cut thy council I am great,
Murdered and by thy master's ready there
My power to give thee but so much as hell:
Some service in the noble bondman here,
Would show him to her wine.

KING LEAR:
0, if you were a feeble sight, the courtesy of your law,
Your sight and several breath, will wear the gods
With his heads, and my hands are wonder'd at the deeds,
So drop upon your lordship's head, and your opinion
Shall be against your honour.
Generating textbooks with RNNs

open source textbook on algebraic geometry

Latex source
Generating textbooks with RNNs

Lemma 0.1. Assume (3) and (3) by the construction in the description. Suppose \( X = \lim |X| \) (by the formal open covering \( X \) and a single map \( \text{Proj}_X(A) = \text{Spec}(B) \) over \( U \) compatible with the complex \( \text{Set}(A) = \Gamma(X, \mathcal{O}_X) \)).

When in this case of to show that \( Q \rightarrow \mathcal{C}/X \) is stable under the following result in the second conditions of (1), and (3). This finishes the proof. By Definition ?? (without element is when the closed subschemes are cut by \( f \)). If \( T \) is surjective we may assume \( T \) is connected with residue fields of \( S \). Moreover there exists a closed subspace \( Z \subset X \) of \( X \) where \( U \) in \( X \) is proper (some defining as a closed subset of the uniqueness it suffices to check the fact that the following theorem

(1) \( f \) is locally of finite type. Since \( S = \text{Spec}(R) \) and \( Y = \text{Spec}(R) \).

Proof. This is form all sheaves of sheaves on \( X \). But given a scheme \( X \) and a surjective étale morphism \( U \rightarrow X \). Let \( U \cap U = \bigcup_{i=1, \ldots, n} U_i \) be the sheaf \( S \) over \( S \) at the schemes \( X_i \rightarrow X \) and \( U = \lim U_i \).

The following lemma surjective restroscomposes of this implies that \( \mathcal{F}_{X_0} = \mathcal{F}_{X_0} \).

Lemma 0.2. Let \( X \) be a locally Noether scheme over \( S \), \( E = \mathcal{F}_{X/S} \). Set \( \mathcal{I} = \mathcal{J}_1 \subset T_0^* \). Since \( T_0^* \subset T^* \) are nonzero over \( i_0 \leq p \) is a subset of \( \mathcal{J}_{n,0} = \mathcal{A}_2 \) works.

Lemma 0.3. In Situation ??, Hence we may assume \( q_0' = 0 \).

Proof. We will use the property we see that \( p \) is the next functor (??). On the other hand, by Lemma ?? we see that

\[
\text{D}(\mathcal{O}_X) = \mathcal{O}_X(\text{D})
\]

where \( K \) is an \( F \)-algebra where \( \delta_{n+1} \) is a scheme over \( S \).
Proof. Omitted.

**Lemma 0.1.** Let $C$ be a set of the construction.

Let $C$ be a gerber covering. Let $F$ be a quasi-coherent sheaves of $O$-modules. We have to show that

$$O_{O_X} = O_X(L)$$

Proof. This is an algebraic space with the composition of sheaves $F$ on $X_{etale}$ we have

$$O_X(F) = \{\text{morph}_1 \times_{O_X}(G, F)\}$$

where $G$ defines an isomorphism $F \to F$ of $O$-modules.

**Lemma 0.2.** This is an integer $Z$ is injective.

Proof. See Spaces, Lemma ??.

**Lemma 0.3.** Let $S$ be a scheme. Let $X$ be a scheme and $X$ is an affine open covering. Let $U \subset X$ be a canonical and locally of finite type. Let $X$ be a scheme.

Let $X$ be a scheme which is equal to the formal complex.

The following to the construction of the lemma follows.

Let $X$ be a scheme. Let $X$ be a scheme covering. Let

$$b: X \to Y' \to Y \to Y' \to X$$

be a morphism of algebraic spaces over $S$ and $Y$.

Proof. Let $X$ be a nonzero scheme of $X$. Let $X$ be an algebraic space. Let $F$ be a quasi-coherent sheaf of $O_X$-modules. The following are equivalent

1. $F$ is an algebraic space over $S$.
2. If $X$ is an affine open covering.

Consider a common structure on $X$ and $X$ the functor $O_X(U)$ which is locally of finite type.

This since $\mathcal{F} \in \mathcal{F}$ and $x \in \mathcal{G}$ the diagram

$$\begin{array}{c}
\Spec(K_0) \\
\Mor_{\mathcal{S}_{\mathcal{X}}} \\
\mathcal{O}_{X_0}/(\mathcal{O}_{X_0}) \\
\end{array}
$$

is a limit. Then $\mathcal{G}$ is a finite type and assume $S$ is a flat and $\mathcal{F}$ and $\mathcal{G}$ is a finite type $f$. This is of finite type diagrams, and

- the composition of $\mathcal{G}$ is a regular sequence,
- $\mathcal{O}_{X'}$ is a sheaf of rings.

Proof. We have see that $X = \Spec(R)$ and $\mathcal{F}$ is a finite type representable by algebraic space. The property $\mathcal{F}$ is a finite morphism of algebraic stacks. Then the cohomology of $X$ is an open neighbourhood of $U$.

Proof. This is clear that $\mathcal{G}$ is a finite presentation, see Lemmas ??.

A random above we consider that $U$ is an open covering of $C$. The functor $\mathcal{F}$ is a field

$$\mathcal{O}_{X_0} \to \mathcal{F} \to \epsilon(\mathcal{O}_{X_0}) \to \mathcal{O}_{X_0}(\mathcal{O}_{X_0})$$

is an isomorphism of covering of $\mathcal{O}_{X_0}$. If $\mathcal{F}$ is the unique element of $\mathcal{F}$ such that $X$ is an isomorphism.

The property $\mathcal{F}$ is a disjoint union of Proposition ?? and we can filtered set of presentations of a scheme $\mathcal{O}_X$-algebra with $\mathcal{F}$ are opens of finite type over $S$.

If $\mathcal{F}$ is a scheme theoretic image points.

If $\mathcal{F}$ is a finite direct sum $\mathcal{O}_{X_0}$ is a closed immersion, see Lemma ??.

This is a sequence of $\mathcal{F}$ is a similar morphism.
Generating code with RNNs

```c
static void do_command(struct seq_file *m, void *v)
{
    int column = 32 << (cmd[2] & 0x80);
    if (state)
        cmd = (int)(int_state ^ (in_8(&ch->ch_flags) & Cmd) ? 2 : 1);
    else
        seq = 1;
    for (i = 0; i < 16; i++) {
        if (k & (1 << 1))
            pipe = (in_use & UMPTHREAD_UNCC) +
            ((count & 0x00000000fffffff8) & 0x000000ff) << 8;
        if (count == 0)
            sub(pid, ppc_md.kexec_handle, 0x20000000);
        pipe_set_bytes(i, 0);
    }
    /* Free our user pages pointer to place camera if all dash */
    subsystem_info = &of_changes[PAGE_SIZE];
    rek_controls(offset, idx, &soffset);
    /* Now we want to deliberately put it to device */
    control_check_polarity(&context, val, 0);
    for (i = 0; i < COUNTER; i++)
        seq_puts(s, "policy ");
}
```

Generated C code
Image Captioning

CVPR 2015:
Deep Visual-Semantic Alignments for Generating Image Descriptions, Karpathy and Fei-Fei
Show and Tell: A Neural Image Caption Generator, Vinyals et al.
Long-term Recurrent Convolutional Networks for Visual Recognition and Description, Donahue et al.
Learning a Recurrent Visual Representation for Image Caption Generation, Chen and Zitnick

Adapted from Andrej Karpathy
Image Captioning

Recurrent Neural Network

Convolutional Neural Network

Andrej Karpathy
Image Captioning
Image Captioning

Andrej Karpathy
Image Captioning

before:
\[ h = \tanh(W_{xh} \cdot x + W_{hh} \cdot h) \]

now:
\[ h = \tanh(W_{xh} \cdot x + W_{hh} \cdot h + W_{ih} \cdot im) \]
Image Captioning

Andrej Karpathy
Image Captioning

Andrej Karpathy
Image Captioning

Andrej Karpathy
Image Captioning

Andrej Karpathy
Image Captioning

Caption generated: “straw hat”

Sample <END> token => finish.
Image Captioning

- "man in black shirt is playing guitar."
- "construction worker in orange safety vest is working on road."
- "two young girls are playing with lego toy."
- "boy is doing backflip on wakeboard."
- "a young boy is holding a baseball bat."
- "a cat is sitting on a couch with a remote control."
- "a woman holding a teddy bear in front of a mirror."
- "a horse is standing in the middle of a road."

Andrei Karpathy
Video Captioning

Key Insight:
Generate feature representation of the video and “decode” it to a sentence

Venugopalan et al., “Translating Videos to Natural Language using Deep Recurrent Neural Networks”, NAACL-HTL 2015
A boy is playing golf.<EOS>

Venugopalan et al., “Translating Videos to Natural Language using Deep Recurrent Neural Networks”, NAACL-HTL 2015
Video Captioning

FGM: A person is dancing with the person on the stage.
YT: A group of men are riding the forest.
I+V: A group of people are dancing.
GT: Many men and women are dancing in the street.

FGM: A person is cutting a potato in the kitchen.
YT: A man is slicing a tomato.
I+V: A man is slicing a carrot.
GT: A man is slicing carrots.

FGM: A person is walking with a person in the forest.
YT: A monkey is walking.
I+V: A bear is eating a tree.
GT: Two bear cubs are digging into dirt and plant matter at the base of a tree.

FGM: A person is riding a horse on the stage.
YT: A group of playing are playing in the ball.
I+V: A basketball player is playing.
GT: Dwayne wade does a fancy layup in an allstar game.

Venugopalan et al., “Translating Videos to Natural Language using Deep Recurrent Neural Networks”, NAACL-HTL 2015
Video Captioning

Venugopalan et al., “Sequence to Sequence - Video to Text”, ICCV 2015
Video Captioning

S2VT Overview

Encoding stage

Decoding stage

Venugopalan et al., “Sequence to Sequence - Video to Text”, ICCV 2015
Neural Machine Translation?

- **Neural Machine Translation (NMT)** is a way to do Machine Translation with a *single neural network*.

- The neural network architecture is called *sequence-to-sequence* (aka *seq2seq*) and it involves *two RNNs*. 
**Neural Machine Translation (NMT)**

The sequence-to-sequence model

Encoding of the source sentence. Provides initial hidden state for Decoder RNN.

- **Source sentence (input)**
  - Encoder RNN
    - \(`il\)
    - \(`a\)
    - \(`m'\)
    - \(`entarté\)

- **Target sentence (output)**
  - Decoder RNN
    - \(`<START>\)
    - \(`he\)
    - \(`hit\)
    - \(`me\)
    - \(`with\)
    - \(`a\)
    - \(`pie\)
    - \(`<END>\)

**Decoder RNN** is a Language Model that generates target sentence, *conditioned on encoding*.

- **Note:** This diagram shows test time behavior: decoder output is fed in \(\rightarrow\) as next step’s input

Abigail See
Greedy decoding

- We saw how to generate (or “decode”) the target sentence by taking argmax on each step of the decoder

- This is greedy decoding (take most probable word on each step)

- Problems with this method?
Problems with greedy decoding

- Greedy decoding has no way to undo decisions!
  - **Input**: *il am’entarté*  (*he hit me with a pie*)
  - → *he _____*
  - → *he hit _____*
  - → *he hit a _____*  (*whoops! no going back now…*)

- How to fix this?
Exhaustive search decoding

- Ideally we want to find a (length $T$) translation $y$ that maximizes

$$
P(y|x) = P(y_1|x) P(y_2|y_1, x) P(y_3|y_1, y_2, x) \ldots, P(y_T|y_1, \ldots, y_{T-1}, x)
$$

$$
= \prod_{t=1}^{T} P(y_t|y_1, \ldots, y_{t-1}, x)
$$

- We could try computing all possible sequences $y$
  - This means that on each step $t$ of the decoder, we’re tracking $V^T$ possible partial translations, where $V$ is vocabulary size
  - This $O(V^T)$ complexity is far too expensive!
Beam search decoding

- **Core idea:** On each step of decoder, keep track of the $k$ most probable partial translations (**hypotheses**)
  - $k$ is the beam size (in practice around 5 to 10)

- A hypothesis $y_1, \ldots, y_t$ has a **score** which is its log probability:

$$
score(y_1, \ldots, y_t) = \log P_{LM}(y_1, \ldots, y_t|x) = \sum_{i=1}^{t} \log P_{LM}(y_i|y_1, \ldots, y_{i-1}, x)
$$

  - Scores are all negative, and higher score is better
  - We search for high-scoring hypotheses, tracking top $k$ on each step

- Beam search is **not guaranteed** to find optimal solution
- But **much more efficient** than exhaustive search!
Beam search decoding: example

Beam size = $k = 2$. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{LM}(y_i | y_1, \ldots, y_{i-1}, x)$
Beam search decoding: example

Beam size = $k = 2$. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{LM}(y_i|y_1, \ldots, y_{i-1}, x)$

-0.7 = $\log P_{LM}(he|<\text{START}>)$

-0.9 = $\log P_{LM}(l|<\text{START}>)$

Take top $k$ words and compute scores
Beam search decoding: example

Beam size = $k = 2$. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{LM}(y_i|y_1, \ldots, y_{i-1}, x)$

For each of the $k$ hypotheses, find top $k$ next words and calculate scores

-1.7 = $\log P_{LM}(\text{hit}|<\text{START}> \text{he}) + 0.7$

-2.9 = $\log P_{LM}(\text{struck}|<\text{START}> \text{he}) + 0.7$

-1.6 = $\log P_{LM}(\text{was}|<\text{START}> I) + 0.9$

-1.8 = $\log P_{LM}(\text{got}|<\text{START}> I) + 0.9$
Beam search decoding: example

Beam size = \( k = 2 \). Blue numbers =

\[
\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i|y_1, \ldots, y_{i-1}, x)
\]

Of these \( k^2 \) hypotheses, just keep \( k \) with highest scores

\[
\begin{align*}
\langle \text{START} \rangle & \quad -0.7 \\
\text{he} & \quad -0.7 \\
\text{hit} & \quad -1.7 \\
\text{struck} & \\
\text{I} & \quad -0.9 \\
\text{got} & \quad -1.8 \\
\text{was} & \quad -1.6 \\
\end{align*}
\]
Beam search decoding: example

Beam size = \( k = 2 \). Blue numbers = \( \text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{LM}(y_i|y_1, \ldots, y_{i-1}, x) \)

For each of the \( k \) hypotheses, find top \( k \) next words and calculate scores

-2.8 = \( \log P_{LM}(a|<\text{START}> \text{ he hit}) \) + -1.7

-2.5 = \( \log P_{LM}(me|<\text{START}> \text{ he hit}) \) + -1.7

-2.9 = \( \log P_{LM}(hit|<\text{START}> \text{ I was}) \) + -1.6

-3.8 = \( \log P_{LM}(struck|<\text{START}> \text{ I was}) \) + -1.6

Abigail See
Beam search decoding: example

Beam size = k = 2. Blue numbers = \( \text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{LM}(y_i|y_1, \ldots, y_{i-1}, x) \)

Of these \( k^2 \) hypotheses, just keep \( k \) with highest scores
Beam search decoding: example

Beam size = $k = 2$. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{LM}(y_i|y_1, \ldots, y_{i-1}, x)$

For each of the $k$ hypotheses, find top $k$ next words and calculate scores

Abigail See
Beam search decoding: example

Beam size = \( k = 2 \). Blue numbers = \( \text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{LM}(y_i|y_1, \ldots, y_{i-1}, x) \)

Of these \( k^2 \) hypotheses, just keep \( k \) with highest scores

Abigail See
Beam search decoding: example

Beam size = $k = 2$. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{LM}(y_i|y_1, \ldots, y_{i-1}, x)$

For each of the $k$ hypotheses, find top $k$ next words and calculate scores
Beam search decoding: example

Beam size = $k = 2$. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{LM}(y_i|y_1, \ldots, y_{i-1}, x)$

Of these $k^2$ hypotheses, just keep $k$ with highest scores
Beam search decoding: example

Beam size = $k = 2$. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{LM}(y_i|y_1, \ldots, y_{i-1}, x)$

For each of the $k$ hypotheses, find top $k$ next words and calculate scores.
Beam search decoding: example

Beam size = \( k = 2 \). Blue numbers = \( \text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{LM}(y_i|y_1, \ldots, y_{i-1}, x) \)

This is the top-scoring hypothesis!
Beam search decoding: example

Beam size = \( k = 2 \). Blue numbers = \( \text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{LM}(y_i | y_1, \ldots, y_{i-1}, x) \)

Abigail See
Beam search decoding: finishing up

- We have our list of completed hypotheses.
- How to select top one with highest score?

- Each hypothesis $y_1, \ldots , y_t$ on our list has a score

$$\text{score}(y_1, \ldots , y_t) = \log P_{LM}(y_1, \ldots , y_t | x) = \sum_{i=1}^{t} \log P_{LM}(y_i | y_1, \ldots , y_{i-1}, x)$$

- **Problem with this:** longer hypotheses have lower scores

- **Fix:** Normalize by length. Use this to select top one instead:

$$\frac{1}{t} \sum_{i=1}^{t} \log P_{LM}(y_i | y_1, \ldots , y_{i-1}, x)$$
How do we evaluate Machine Translation?

**BLEU (Bilingual Evaluation Understudy)**

- BLEU compares the machine-written translation to one or several human-written translation(s), and computes a similarity score based on:
  - *n*-gram precision (usually for 1, 2, 3 and 4-grams)
  - Plus a penalty for too-short system translations

- BLEU is useful but imperfect
  - There are many valid ways to translate a sentence
  - So a good translation can get a poor BLEU score because it has low *n*-gram overlap with the human translation 😞

MT progress over time
[Edinburgh En-De WMT newstest2013 Cased BLEU; NMT 2015 from U. Montréal]

NMT: the biggest success story of NLP Deep Learning

Neural Machine Translation went from a fringe research activity in 2014 to the leading standard method in 2016

- 2014: First seq2seq paper published
- 2016: Google Translate switches from SMT to NMT
- This is amazing!
  - SMT systems, built by hundreds of engineers over many years, outperformed by NMT systems trained by a handful of engineers in a few months
So is Machine Translations solved?

• Nope!

• Many difficulties remain:
  • Out-of-vocabulary words
  • Domain mismatch between train and test data
  • Maintaining context over longer text
  • Low-resource language pairs

Further reading: “Has AI surpassed humans at translation? Not even close!”
https://www.skynettoday.com/editorials/state_of_nmt
So is Machine Translations solved?

- Nope!
- Using **common sense** is still hard

```
| English | | Spanish |
|---------|-----------------|
| paper jam Edit | Mermelada de papel |
```

Abigail See
So is Machine Translations solved?

- **Nope!**
- **NMT picks up biases in training data**

Source: [https://hackernoon.com/bias-sexist-or-this-is-the-way-it-should-be-ce1f7c8c683c](https://hackernoon.com/bias-sexist-or-this-is-the-way-it-should-be-ce1f7c8c683c)
NMT research continues

NMT is the flagship task for NLP Deep Learning

- NMT research has pioneered many of the recent innovations of NLP Deep Learning

- In 2019: NMT research continues to thrive
  - Researchers have found many, many improvements to the “vanilla” seq2seq NMT system
  - But one improvement is so integral that it is the new vanilla…

ATTENTION
Sequence-to-sequence: the bottleneck problem

Encoding of the source sentence.

Source sentence (input):

```
il a m' entarté
```

Target sentence (output):

```
<START> he hit me with a pie <END>
```

Problems with this architecture?

Abigail See
Sequence-to-sequence: the bottleneck problem

Encoding of the source sentence. This needs to capture all information about the source sentence. Information bottleneck!

Source sentence (input)

Target sentence (output)

He hit me with a pie

Abigail See
Attention

- **Attention** provides a solution to the bottleneck problem.

- **Core idea**: on each step of the decoder, use *direct connection to the encoder* to *focus on a particular part* of the source sequence.

- First we will show via diagram (no equations), then we will show with equations.
Sequence-to-sequence with attention

Encoder RNN

Attention scores

dot product

Decoder RNN

Source sentence (input)

Abigail See
Sequence-to-sequence with attention

Encoder RNN

Attention scores

Decoder RNN

Source sentence (input)

il a m' entarté <START>

dot product

Abigail See
Sequence-to-sequence with attention

Encoder RNN

Attention scores

Decoder RNN

Source sentence (input)

il a m’ entarté <START>

dot product

Abigail See
Sequence-to-sequence with attention

Encoder RNN

Attention scores

Decoder RNN

Source sentence (input)

dot product

il  a  m'  entarté  <START>
Sequence-to-sequence with attention

On this decoder timestep, we’re mostly focusing on the first encoder hidden state (“he”)

Take softmax to turn the scores into a probability distribution

Source sentence (input)

Abigail See
Sequence-to-sequence with attention

Use the attention distribution to take a **weighted sum** of the encoder hidden states.

The attention output mostly contains information from the hidden states that received high attention.
Sequence-to-sequence with attention

Encoder RNN

Attention distribution

Attention scores

Source sentence (input)

<START>

il a m' entarté

Concatenate attention output with decoder hidden state, then use to compute $y_1$ as before
Sequence-to-sequence with attention

Sometimes we take the attention output from the previous step, and also feed it into the decoder (along with the usual decoder input).
Attention: in equations

- We have encoder hidden states $h_1, \ldots, h_N \in \mathbb{R}^h$
- On timestep $t$, we have decoder hidden state $s_t \in \mathbb{R}^h$
- We get the attention scores $e^t$ for this step:
  \[
  e^t = [s_t^T h_1, \ldots, s_t^T h_N] \in \mathbb{R}^N
  \]
- We take softmax to get the attention distribution $\alpha^t$ for this step (this is a probability distribution and sums to 1)
  \[
  \alpha^t = \text{softmax}(e^t) \in \mathbb{R}^N
  \]
- We use $\alpha^t$ to take a weighted sum of the encoder hidden states to get the attention output $a_t$
  \[
  a_t = \sum_{i=1}^{N} \alpha_i^t h_i \in \mathbb{R}^h
  \]
- Finally we concatenate the attention output $a_t$ with the decoder hidden state $s_t$ and proceed as in the non-attention seq2seq model
  \[
  [a_t; s_t] \in \mathbb{R}^{2h}
  \]
Attention is great

- Attention significantly improves NMT performance
  - It’s very useful to allow decoder to focus on certain parts of the source
- Attention solves the bottleneck problem
  - Attention allows decoder to look directly at source; bypass bottleneck
- Attention helps with vanishing gradient problem
  - Provides shortcut to faraway states
- Attention provides some interpretability
  - By inspecting attention distribution, we can see what the decoder was focusing on
  - We get (soft) alignment for free!
  - This is cool because we never explicitly trained an alignment system
  - The network just learned alignment by itself
Attention is a general Deep Learning technique

- We’ve seen that attention is a great way to improve the sequence-to-sequence model for Machine Translation.
- **However**: You can use attention in many architectures (not just seq2seq) and many tasks (not just MT).

  **More general definition of attention**:  
  - Given a set of vector values, and a vector query, attention is a technique to compute a weighted sum of the values, dependent on the query.

- We sometimes say that the query attends to the values.
- For example, in the seq2seq + attention model, each decoder hidden state (query) attends to all the encoder hidden states (values).
Plan for this lecture

• Recurrent neural networks
  – Basics
  – Training (backprop through time, vanishing gradient)
  – Recurrent networks with gates (GRU, LSTM)

• Applications in NLP and vision
  – Image/video captioning
  – Neural machine translation (beam search, attention)

• Transformers
  – Self-attention
  – BERT
  – Cross-modal transformers for VQA and VCR
Transformers

(meaning representation through context, representation learning, unsupervised learning)
How do we represent the meaning of a word?

Definition: meaning (Webster dictionary)
- the idea that is represented by a word, phrase, etc.
- the idea that a person wants to express by using words, signs, etc.
- the idea that is expressed in a work of writing, art, etc.

Commonest linguistic way of thinking of meaning:

\[
\text{signifier (symbol)} \iff \text{signified (idea or thing)}
\]

= denotational semantics

Christopher Manning
How do we have usable meaning in a computer?

Common solution: Use e.g. WordNet, a thesaurus containing lists of synonym sets and hypernyms (“is a” relationships).

e.g. synonym sets containing “good”:

```python
from nltk.corpus import wordnet as wn
poses = { 'n': 'noun', 'v': 'verb', 's': 'adj (s)', 'a': 'adj', 'r': 'adv'}
for synset in wn.synsets("good"):
    print("{}: ",", ".join([l.name() for l in synset.lemmas()])))
```

```
noun: good
noun: good, goodness
noun: good, goodness
noun: commodity, trade_good, good
adj: good
adj (sat): full, good  adj:
    good
adj (sat): estimable, good, honorable, respectable  adj (sat):
    beneficial, good
adj (sat): good
adj (sat): good, just, upright ...
adverb: well, good
adverb: thoroughly, soundly, good
```

e.g. hypernyms of “panda”:

```python
from nltk.corpus import wordnet as wn
panda = wn.synset("panda.n.01")
hyper = lambda s: s.hypernyms()
list(panda.closure(hyper))
```

```
[Synset('procyonid.n.01'),
 Synset('carnivore.n.01'),
 Synset('placental.n.01'),
 Synset('mammal.n.01'),
 Synset('vertebrate.n.01'),
 Synset('chordate.n.01'),
 Synset('animal.n.01'),
 Synset('organism.n.01'),
 Synset('living_thing.n.01'),
 Synset('whole.n.02'),
 Synset('object.n.01'),
 Synset('physical_entity.n.01'),
 Synset('entity.n.01')]
```
Problems with resources like WordNet

- Great as a resource but missing nuance
  - e.g. "proficient" is listed as a synonym for "good". This is only correct in some contexts.

- Missing new meanings of words
  - e.g., wicked, badass, nifty, wizard, genius, ninja, bombest
  - Impossible to keep up-to-date!

- Subjective
- Requires human labor to create and adapt
- Can’t compute accurate word similarity
Representing words as discrete symbols

In traditional NLP, we regard words as discrete symbols: hotel, conference, motel - a localist representation

Means one 1, the rest 0s

Words can be represented by one-hot vectors:

motel = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
hotel = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]

Vector dimension = number of words in vocab (e.g. 500,000)
Problem with words as discrete symbols

Example: in web search, if user searches for “Seattle motel”, we would like to match documents containing “Seattle hotel”.

But:

\[
\text{motel} = [0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0] \\
\text{hotel} = [0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0]
\]

These two vectors are orthogonal.
There is no natural notion of similarity for one-hot vectors!

Solution:
- Could try to rely on WordNet’s list of synonyms to get similarity?
  - But it is well-known to fail badly: incompleteness, etc.
- Instead: learn to encode similarity in the vectors themselves
Representing words by their context

- **Distributional semantics**: A word’s meaning is given by the words that frequently appear close-by
  - “You shall know a word by the company it keeps” (J. R. Firth 1957)
  - One of the most successful ideas of modern statistical NLP!

- When a word \( w \) appears in a text, its **context** is the set of words that appear nearby (within a fixed-size window).

- Use the many contexts of \( w \) to build up a representation of \( w \)

  …government debt problems turning into banking crises as happened in 2009…
  …saying that Europe needs unified banking regulation to replace the hodgepodge…
  …India has just given its banking system a shot in the arm…

These context words will represent **banking**.
Word vectors

We will build a dense vector for each word, chosen so that it is similar to vectors of words that appear in similar contexts.

\[
\text{banking} = \begin{pmatrix}
0.286 \\
0.792 \\
eg0.177 \\
eg0.107 \\
0.109 \\
-0.542 \\
0.349 \\
0.271 \\
\end{pmatrix}
\]

Note: word vectors are sometimes called word embeddings or word representations. They are a distributed representation.
Word meaning as a neural word vector - visualization

\[
\text{expect} = \\
\begin{pmatrix}
0.286 \\
0.792 \\
-0.177 \\
-0.107 \\
0.109 \\
-0.542 \\
0.349 \\
0.271 \\
0.487 \\
\end{pmatrix}
\]
3. Word2vec: Overview

Word2vec (Mikolov et al. 2013) is a framework for learning word vectors

Idea:
• We have a large corpus of text
• Every word in a fixed vocabulary is represented by a vector
• Go through each position $t$ in the text, which has a center word $c$ and context ("outside") words $o$
• Use the similarity of the word vectors for $c$ and $o$ to calculate the probability of $o$ given $c$ (or vice versa)
• Keep adjusting the word vectors to maximize this probability
**Word2Vec Overview**

- Example windows and process for computing $P(w_{t+j}/w_t)$
Word2Vec Overview

- Example windows and process for computing $P(w_{t+j} | w_t)$
**Word2vec: objective function**

For each position \( t = 1, \ldots, T \), predict context words within a window of fixed size \( m \), given center word \( w_j \).

\[
\text{Likelihood} = L(\theta) = \prod_{t=1}^{T} \prod_{-m \leq j \leq m, j \neq 0} P(w_{t+j} \mid w_t; \theta)
\]

\( \theta \) is all variables to be optimized

Sometimes called cost or loss function

The **objective function** is the (average) negative log likelihood:

\[
J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{-m \leq j \leq m, j \neq 0} \log P(w_{t+j} \mid w_t; \theta)
\]

**Minimizing objective function \( \Leftrightarrow \) Maximizing predictive accuracy**
Word2vec: objective function

• We want to minimize the objective function:

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

• **Question:** How to calculate \( P(w_{t+j} | w_t; \theta) \)?

• **Answer:** We will use two vectors per word \( w \):
  
  • \( v_w \) when \( w \) is a center word
  
  • \( u_w \) when \( w \) is a context word

• Then for a center word \( c \) and a context word \( o \):

\[
P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}
\]
**Word2vec: prediction function**

The softmax function maps arbitrary values to a probability distribution $p_i$.

- “max” because amplifies probability of largest $x_i$
- “soft” because still assigns some probability to smaller $x_i$

Frequently used in Deep Learning

**Formally**

$$P(o | c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

- Exponentiation makes anything positive
- Dot product compares similarity of $o$ and $c$.
  $$u^T v = u \cdot v = \sum_{i=1}^{n} u_i v_i$$
  Larger dot product = larger probability
- Normalize over entire vocabulary to give probability distribution

**Mathematically**

$$\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)} = p_i$$

Christopher Manning
Peters et al. (2018): ELMo: Embeddings from Language Models


- Breakout version of word token vectors or contextual word vectors
- Learn word token vectors using long contexts not context windows (here, whole sentence, could be longer)
- Learn a deep Bi-NLM and use all its layers in prediction
Peters et al. (2018): ELMo: Embeddings from Language Models

- Train a bidirectional LM
- Aim at performant but not overly large LM:
  - Use 2 biLSTM layers
  - Use character CNN to build initial word representation
  - Use 4096 dim hidden/cell LSTM states with 512 dim projections to next input
  - Use a residual connection
  - Tie parameters of token input and output (softmax) and tie these between forward and backward LMs
ELMo used in a sequence tagger

ELMo results: Great for all tasks

<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: Weighting of layers

- The two biLSTM NLM layers have differentiated uses/meanings
  - Lower layer is better for lower-level syntax, etc.
    - Part-of-speech tagging, syntactic dependencies, NER
  - Higher layer is better for higher-level semantics
    - Sentiment, Semantic role labeling, question answering, SNLI
Let’s scale it up!

ULMfit
Jan 2018
Training: 1 GPU day

GPT
June 2018
Training
240 GPU days

BERT
Oct 2018
Training
256 TPU days
~320–560 GPU days

GPT-2
Feb 2019
Training
~2048 TPU v3 days according to a reddit thread

Christopher Manning
**GPT-2 language model cherry-picked output**

<table>
<thead>
<tr>
<th>SYSTEM PROMPT (HUMAN-WRITTEN)</th>
<th>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.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MODEL COMPLETION (MACHINE-WRITTEN, 10 TRIES)</td>
<td>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 Pérez, 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. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow. Pérez and the others then ventured further into the valley. …</td>
</tr>
</tbody>
</table>

Christopher Manning
Elon Musk’s scientists have announced the creation of a terrifying artificial intelligence that’s so smart they refused to release it to the public.

OpenAI’s GPT-2 is designed to write just like a human and is an impressive leap forward capable of penning chillingly convincing text.

It was ‘trained’ by analysing eight million web pages and is capable of writing large tracts based upon a ‘prompt’ written by a real person.

But the machine mind will not be released in its fully-fledged form because of the risk of it being used for ‘malicious purposes’ such as generating fake news, impersonating people online, automating the production of spam or churning out ‘abusive or faked content to post on social media’.

OpenAI wrote: ‘Due to our concerns about malicious applications of the technology, we are not releasing the trained model.'
The Motivation for Transformers

• We want parallelization but RNNs are inherently sequential

• Despite GRUs and LSTMs, RNNs still need attention mechanism to deal with long range dependencies – path length between states grows with sequence otherwise

• But if attention gives us access to any state… maybe we can just use attention and don’t need the RNN?
Transformer Overview

Attention is all you need. 2017. Aswani, Shazeer, Parmar, Uszkoreit, Jones, Gomez, Kaiser, Polosukhin

- Non-recurrent sequence-to-sequence encoder-decoder model
- Task: machine translation with parallel corpus
- Predict each translated word
- Final cost/error function is standard cross-entropy error on top of a softmax classifier

This and related figures from paper ↑

Christopher Manning
Dot-Product Attention (Extending our previous def.)

- Inputs: a query $q$ and a set of key-value (k-v) pairs to an output
- Query, keys, values, and output are all vectors

- Output is weighted sum of values, where
- Weight of each value is computed by an inner product of query and corresponding key
- Queries and keys have same dimensionality $d_k$, values have $d_v$

$$A(q, K, V) = \sum_i \frac{e^{q \cdot k_i}}{\sum_j e^{q \cdot k_j}} v_i$$
Dot-Product Attention - Matrix notation

- When we have multiple queries q, we stack them in a matrix Q:

\[ A(q, K, V) = \sum_i \frac{e^{q \cdot k_i}}{\sum_j e^{q \cdot k_j}} v_i \]

- Becomes:

\[ A(Q, K, V) = \text{softmax}(QK^T)V \]

\[
\begin{bmatrix} |Q| \times d_k \end{bmatrix} \times \begin{bmatrix} d_k \times |K| \end{bmatrix} \times \begin{bmatrix} |K| \times d_v \end{bmatrix} \]

softmax row-wise

\[ = \begin{bmatrix} |Q| \times d_v \end{bmatrix} \]
Scaled Dot-Product Attention

- Problem: As $d_k$ gets large, the variance of $QK^T$ increases $\rightarrow$ some values inside the softmax get large $\rightarrow$ the softmax gets very peaked $\rightarrow$ hence its gradient gets smaller.

- Solution: Scale by length of query/key vectors:

$$A(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

- The input word vectors are the queries, keys and values
- In other words: the word vectors select each other

Adapted from Christopher Manning
Multi-head attention

- Problem with simple self-attention:
  - Only one way for words to interact with one-another
- Solution: Multi-head attention
  - First map Q, K, V into h=8 many lower dimensional spaces via W matrices
  - Then apply attention, then concatenate outputs and pipe through linear layer

\[
\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O
\]
\[
\text{where } \text{head}_i = \text{Attention}(QW^Q_i, KW^K_i, VW^V_i)
\]

Complete Encoder

- For encoder, at each block, we use the same Q, K and V from the previous layer

- Blocks are repeated 6 times (in vertical stack)
Attention visualization in layer 5

- Words start to pay attention to other words in sensible ways
Attention visualization: Implicit anaphora resolution

In 5th layer. Isolated attentions from just the word ‘its’ for attention heads 5 and 6. Note that the attentions are very sharp for this word.

Christopher Manning
Parallel attention heads

I kicked the ball

Who

Did what?

To whom?

Ashish Vaswani
Transformer Decoder

- 2 sublayer changes in decoder
- Masked decoder self-attention on previously generated outputs:

Blocks repeated 6 times also

- Encoder-Decoder Attention, where queries come from previous decoder layer and keys and values come from output of encoder

Christopher Manning
BERT (Bidirectional Encoder Representations from Transformers):
Pre-training of Deep Bidirectional Transformers for Language Understanding

Based on slides from Jacob Devlin

- Mask out $k\%$ of the input words, and then predict the masked words
  - They always use $k = 15\%$

  
  store  
  ↑

  the man went to the [MASK] to buy a [MASK] of milk

  gallon  
  ↑

- Too little masking: Too expensive to train
- Too much masking: Not enough context
Additional task: Next sentence prediction

• To learn relationships between sentences, predict whether Sentence B is actual sentence that proceeds Sentence A, or a random sentence

Sentence A = The man went to the store.
Sentence B = He bought a gallon of milk.
Label = IsNextSentence

Sentence A = The man went to the store.
Sentence B = Penguins are flightless.
Label = NotNextSentence

Adapted from Christopher Manning
BERT sentence pair encoding

<table>
<thead>
<tr>
<th>Input</th>
<th>[CLS]</th>
<th>my</th>
<th>dog</th>
<th>is</th>
<th>cute</th>
<th>[SEP]</th>
<th>he</th>
<th>likes</th>
<th>play</th>
<th>#ing</th>
<th>[SEP]</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Token Embeddings</strong></td>
<td>$E_{[CLS]}$</td>
<td>$E_{my}$</td>
<td>$E_{dog}$</td>
<td>$E_{is}$</td>
<td>$E_{cute}$</td>
<td>$E_{[SEP]}$</td>
<td>$E_{he}$</td>
<td>$E_{likes}$</td>
<td>$E_{play}$</td>
<td>$E_{#ing}$</td>
<td>$E_{[SEP]}$</td>
</tr>
<tr>
<td><strong>Segment Embeddings</strong></td>
<td>$E_A$</td>
<td>$E_A$</td>
<td>$E_A$</td>
<td>$E_A$</td>
<td>$E_A$</td>
<td>$E_B$</td>
<td>$E_B$</td>
<td>$E_B$</td>
<td>$E_B$</td>
<td>$E_B$</td>
<td>$E_B$</td>
</tr>
<tr>
<td><strong>Position Embeddings</strong></td>
<td>$E_0$</td>
<td>$E_1$</td>
<td>$E_2$</td>
<td>$E_3$</td>
<td>$E_4$</td>
<td>$E_5$</td>
<td>$E_6$</td>
<td>$E_7$</td>
<td>$E_8$</td>
<td>$E_9$</td>
<td>$E_{10}$</td>
</tr>
</tbody>
</table>

Token embeddings are word pieces
Learned segmented embedding represents each sentence
Positional embedding

Adapted from Christopher Manning
BERT model architecture and training

- Transformer encoder (as before)
- Self-attention ⇒ no locality bias
  - Long-distance context has “equal opportunity”
- Single multiplication per layer ⇒ efficiency on GPU/TPU

- Train on Wikipedia + BookCorpus
- Train 2 model sizes:
  - BERT-Base: 12-layer, 768-hidden, 12-head
  - BERT-Large: 24-layer, 1024-hidden, 16-head
- Trained on 4x4 or 8x8 TPU slice for 4 days
BERT model fine tuning

- Simply learn a classifier built on the top layer for each task that you fine tune for

Christopher Manning
# SQuAD 2.0 leaderboard, 2019-02-07

<table>
<thead>
<tr>
<th>Rank</th>
<th>Model</th>
<th>EM</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BERT + MMFT + ADA (ensemble)</td>
<td>85.082</td>
<td>87.615</td>
</tr>
<tr>
<td></td>
<td><em>Microsoft Research Asia</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>BERT + Synthetic Self-Training (ensemble)</td>
<td>84.292</td>
<td>86.967</td>
</tr>
<tr>
<td></td>
<td><em>Google AI Language</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><a href="https://github.com/google-research/bert">https://github.com/google-research/bert</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>BERT finetune baseline (ensemble)</td>
<td>83.536</td>
<td>86.096</td>
</tr>
<tr>
<td></td>
<td><em>Anonymous</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Lunet + Verifier + BERT (ensemble)</td>
<td>83.469</td>
<td>86.043</td>
</tr>
<tr>
<td></td>
<td><em>Layer 6 AI NLP Team</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>PAML+BERT (ensemble model)</td>
<td>83.457</td>
<td>86.122</td>
</tr>
<tr>
<td></td>
<td><em>PINGAN GammaLab</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Lunet + Verifier + BERT (single model)</td>
<td>82.995</td>
<td>86.035</td>
</tr>
<tr>
<td></td>
<td><em>Layer 6 AI NLP Team</em></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Cross-modal transformers

Figure 1: Our ViLBERT model consists of two parallel streams for visual (green) and linguistic (purple) processing that interact through novel co-attentional transformer layers. This structure allows for variable depths for each modality and enables sparse interaction through co-attention. Dashed boxes with multiplier subscripts denote repeated blocks of layers.

Cross-modal transformers

Figure 1: The LXMERT model for learning vision-and-language cross-modality representations. ‘Self’ and ‘Cross’ are abbreviations for self-attention sub-layers and cross-attention sub-layers, respectively. ‘FF’ denotes a feed-forward sub-layer.
Cross-modal transformers

Figure 1: Overview of the proposed UNITER model (best viewed in color), consisting of an Image Embedder, a Text Embedder and a multi-layer self-attention Transformer, learned through three pre-training tasks.

Chen et al., “UNITER: Learning UNiversal Image-TExt Representations”, arxiv 2019
### Visual Commonsense Reasoning Leaderboard

#### Why is [person4] pointing at [person1]?

<table>
<thead>
<tr>
<th>Rank</th>
<th>Model</th>
<th>Q-&gt;A</th>
<th>QA-&gt;R</th>
<th>Q-&gt;AR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Human Performance</td>
<td>91.0</td>
<td>93.0</td>
<td>85.0</td>
</tr>
<tr>
<td></td>
<td>University of Washington</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(Zellers et al. `18)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>UNITER-large (ensemble)</td>
<td>79.8</td>
<td>83.4</td>
<td>66.8</td>
</tr>
<tr>
<td></td>
<td>M5 D365 AI</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>UNITER-large (single model)</td>
<td>77.3</td>
<td>80.8</td>
<td>62.8</td>
</tr>
<tr>
<td></td>
<td>M5 D365 AI</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>VilBERT (ensemble of 10 models)</td>
<td>76.4</td>
<td>78.0</td>
<td>59.8</td>
</tr>
<tr>
<td></td>
<td>Georgia Tech &amp; Facebook AI Research</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><a href="https://arxiv.org/abs/1908.02265">https://arxiv.org/abs/1908.02265</a></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>VL-BERT (single model)</td>
<td>75.8</td>
<td>78.4</td>
<td>59.7</td>
</tr>
<tr>
<td></td>
<td>MSRA &amp; USTC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>VilBERT (ensemble of 5 models)</td>
<td>75.7</td>
<td>77.5</td>
<td>58.8</td>
</tr>
<tr>
<td></td>
<td>Georgia Tech &amp; Facebook AI Research</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><a href="https://arxiv.org/abs/1908.02265">https://arxiv.org/abs/1908.02265</a></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Date*
- September 30, 2019
- September 23, 2019
- August 9, 2019
- September 23, 2019
- August 9, 2019

[https://visualcommonsense.com/leaderboard/](https://visualcommonsense.com/leaderboard/)

**Rationale**: I think so because...
- [person3] has the pancakes in front of him.
- [person2] is taking everyone's order and asked for clarification.
- [person2] is smiling slightly.
- [person1] is delivering food to the table, and she might not know whose order is which.
Additional resource

• Learning about transformers on your own?
  • Key recommended resource:
    • [http://nlp.seas.harvard.edu/2018/04/03/attention.html](http://nlp.seas.harvard.edu/2018/04/03/attention.html)
    • The Annotated Transformer by Sasha Rush
  • An Jupyter Notebook using PyTorch that explains everything!
Recap

- Language modeling is an effective form of unsupervised pretraining for many different supervised tasks
- Attention captures relationships effectively, helps with vanishing gradients
- Attention is cheap to compute and allows better parallelization during training
- Language/sequence models can be extended to settings beyond NLP
- You will know the meaning of a concept/word/image by the company it keeps