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
  - Neural machine translation (beam search, attention)
  - Image/video captioning
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.
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:
Recurrent Networks offer a lot of flexibility:

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

e.g. sentiment classification
sequence of words -> sentiment

Andrej Karpathy
Recurrent Networks offer a lot of flexibility:

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

E.g., machine translation: seq of words -> seq of words
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

Andrej Karpathy
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 $h_t$

old state $h_{t-1}$

input vector $x_t$ at some time step

some function $f_W$ 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 $h_t$:

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

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

$$y_t = W_{hy}h_t$$
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”
Example

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?

Andrej Karpathy
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 distribution 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.

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

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

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

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

  the students opened their _____

  - books
  - laptops
  - exams
  - minds

Abigail See
n-gram Language Models

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

\[
P(x^{(t+1)}|x^{(t)}, \ldots, x^{(1)}) = P(x^{(t+1)}|x^{(t)}, \ldots, x^{(t-n+2)})
\]

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

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

\[
\approx \frac{\text{count}(x^{(t+1)}, x^{(t)}, \ldots, x^{(t-n+2)})}{\text{count}(x^{(t)}, \ldots, x^{(t-n+2)})}
\]
**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!

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

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

**Note:** Increasing $n$ makes sparsity problems worse. Typically we can’t have $n$ bigger than 5.

**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*.

Abigail See
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!

\( x^{(1)} \) and \( x^{(2)} \) 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

![Diagram of RNN with hidden states and input sequence](image)
A RNN Language Model

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

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

word embeddings
\[ e(t) = Ex(t) \]

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

$\hat{y}^{(4)} = P(x^{(5)}|\text{the students opened their books laptops zoo})$
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 distribution 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

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

= negative log prob of “students”

Corpus → the

students

opened

their

exams

...
Training a RNN Language Model

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

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

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

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

= negative log prob of “opened”

Corpus → $x^{(1)}$ → $x^{(2)}$ → $x^{(3)}$ → $x^{(4)}$ → exams → ...

Abigail See
Training a RNN Language Model

Loss → $J^{(1)}(\theta) → \hat{y}^{(1)}$ → $\hat{y}^{(2)} → J^{(3)}(\theta) → \hat{y}^{(3)} → J^{(4)}(\theta) → \hat{y}^{(4)}$ → $U → W_h → E$

Predicted prob dists → $J^{(1)}(\theta)$ → $J^{(2)}(\theta)$ → $J^{(3)}(\theta)$ → $J^{(4)}(\theta)$ → $h^{(0)} → h^{(1)} → h^{(2)} → h^{(3)} → h^{(4)} → U → W_h$

$e^{(1)} → e^{(2)} → e^{(3)} → e^{(4)} → E → W_e$

Corpus → the → students → opened → their → exams → ...

= negative log prob of “their”

Abigail See
Training a RNN Language Model

\[ \text{Loss} \rightarrow J^{(1)}(\theta) \rightarrow \hat{y}^{(1)} \rightarrow U \rightarrow h^{(1)} \rightarrow W_h \rightarrow W_e \rightarrow e^{(1)} \rightarrow E \rightarrow \text{the} \]  
\[ J^{(2)}(\theta) \rightarrow \hat{y}^{(2)} \rightarrow U \rightarrow h^{(2)} \rightarrow W_h \rightarrow W_e \rightarrow e^{(2)} \rightarrow E \rightarrow \text{students} \]  
\[ J^{(3)}(\theta) \rightarrow \hat{y}^{(3)} \rightarrow U \rightarrow h^{(3)} \rightarrow W_h \rightarrow W_e \rightarrow e^{(3)} \rightarrow E \rightarrow \text{opened} \]  
\[ J^{(4)}(\theta) \rightarrow \hat{y}^{(4)} \rightarrow U \rightarrow h^{(4)} \rightarrow W_h \rightarrow W_e \rightarrow e^{(4)} \rightarrow E \rightarrow \text{their} \]  
\[ \text{Corpus} \rightarrow x^{(1)} \rightarrow \text{exams} \rightarrow \ldots \]

= negative log prob of “exams”
Training a RNN Language Model

Loss $\rightarrow J^{(1)}(\theta) + J^{(2)}(\theta) + J^{(3)}(\theta) + J^{(4)}(\theta) + \ldots = J(\theta) = \frac{1}{T} \sum_{t=1}^{T} J^{(t)}(\theta)$

Predicted prob dists

Corpus $\rightarrow x^{(1)} \rightarrow x^{(2)} \rightarrow x^{(3)} \rightarrow x^{(4)} \rightarrow \ldots$

Abigail See
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”
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:

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


Abigail See
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:

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

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

Source:

Abigail See
Question: How do we calculate this?

Answer: Backpropagate over timesteps \( i=t,...,0 \), summing gradients as you go. This algorithm is called “backpropagation through time”
Vanishing gradient intuition

\[ h^{(1)} \rightarrow W \rightarrow h^{(2)} \rightarrow W \rightarrow h^{(3)} \rightarrow W \rightarrow h^{(4)} \rightarrow J^{(4)}(\theta) \]
Vanishing gradient intuition

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

Abigail See
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.

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

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


Adapted from Abigail See
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______, 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 tickets.*

• 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

Adapted from Abigail See
Effect of vanishing gradient on RNN-LM

- **LM task:** *The writer of the books ___*
  - is
  - are

- **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:

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

  - learning rate
  - gradient

- 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

```
Algorithm 1 Pseudo-code for norm clipping

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

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

  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)

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

Diagram:

- Final memory
- Memory (reset)
- Update gate
- Reset gate
- Input:

Richard Socher
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

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

- 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 (\( z \))

Adapted from Richard Socher
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 makes them forget it or overwrite it with new input.

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

\[
\begin{align*}
    u^{(t)} &= \sigma \left( W_u h^{(t-1)} + U_u x^{(t)} + b_u \right) \\
    r^{(t)} &= \sigma \left( W_r h^{(t-1)} + U_r x^{(t)} + b_r \right)
\end{align*}
\]

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

\[
\begin{align*}
    \tilde{h}^{(t)} &= \tanh \left( W_h (r^{(t)} \circ h^{(t-1)}) + U_h x^{(t)} + b_h \right) \\
    h^{(t)} &= (1 - u^{(t)}) \circ h^{(t-1)} + u^{(t)} \circ \tilde{h}^{(t)}
\end{align*}
\]

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

**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 $x^{(t)}$, and we will compute a sequence of hidden states $h^{(t)}$ and cell states $c^{(t)}$. 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

$$f^{(t)} = \sigma \left( W_f h^{(t-1)} + U_f x^{(t)} + b_f \right)$$

$$i^{(t)} = \sigma \left( W_i h^{(t-1)} + U_i x^{(t)} + b_i \right)$$

$$o^{(t)} = \sigma \left( W_o h^{(t-1)} + U_o x^{(t)} + b_o \right)$$

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

$$c^{(t)} = f^{(t)} \circ c^{(t-1)} + i^{(t)} \circ \tilde{c}^{(t)}$$

$$h^{(t)} = o^{(t)} \circ \tanh c^{(t)}$$

Gates are applied using element-wise product

All these are vectors of same length $n$
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/
Cell State vs Hidden State

P(Snorlax is being attacked) = 0.9
P(Snorlax is showering) = 0.05
P(Pokemon is drinking water) = 0.05

http://blog.echen.me/2017/05/30/exploring-lstms/
Activity

http://blog.echen.me/2017/05/30/exploring-lstms/
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 for sequence modeling
  - Successful tasks include: handwriting recognition, speech recognition, machine translation, parsing, image captioning
  - LSTM became the dominant approach

- Starting in **2019**, other approaches (e.g. Transformers) became more dominant for certain NLP tasks (will discuss next lecture)
  - For example in WMT (machine translation competition):
    - In WMT **2016**, the summary report contains “RNN” 44 times
    - In WMT **2018**, the report contains “RNN” 9 times and “Transformer” 63 times


Adapted from Abigail See
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

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

element-wise mean/max

element-wise mean/max

positive

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!

Concatenated hidden states

Backward RNN

Forward RNN

the  movie  was  terribly  exciting  !
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.

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

Inverse probability of corpus, according to Language Model

Normalized by number of words

• 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; 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
  – Neural machine translation (beam search, attention)
  – Image/video captioning
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.
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.

```
my favorite season is spring
```

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
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.

*Source: https://medium.com/deep-writing/harry-potter-written-by-artificial-intelligence-8a9431803da6*
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:

  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 ogoh eoase rrranbyne 'nhthnee e plia tkrlrgd t o idoe ns,smtt h ne etie h,hregrts nigtkie,aoaenns lng

train more

"Tmont thityey" fomesscerliund
Keushey. Thom here
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwy fil on aseterlome
cooniegennnc 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 overetical 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

**FANDARUS:**
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 mues 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:**
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.

I’ll drink it.
Generating textbooks with RNNs

Lemma 0.1. Assume (3) and (4) 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 \( Q \to C_{Z/J_X} \) 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 catenary. If \( T \) is surjective we may assume that \( T \) 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

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

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

The following lemma surjective restcompose of this implies that \( F_{X_0} = F_{X_0} = F_{X_0} \).

Lemma 0.2. Let \( X \) be a locally Noetherian scheme over \( S \), \( E = F_{X/S} \). Set \( I_1 = I_0 \cap I_0 \). Since \( I_1 \cap I_0 \) are nonzero over \( i_0 \leq p \) is a subset of \( F_{X/S} \).

Lemma 0.3. In Situation ??, Hence we may assume \( \phi_0 = 0 \).

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

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

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

The result for any open covering follows from the less of Example ??, It may replace \( S \) by \( X_{\text{spars}, \text{etale}} \) which gives an open subspace of \( X \) and \( T \) equal to \( S_{\text{etale}} \), see Descent, Lemma ??, Namely, by Lemma ?? we see that the \( R \) is geometrically regular over \( S \).
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 & UMXTREADING_UNCCA) +
            ((count & 0x00000000fffffff8) & 0x0000000f) << 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 ");
}
```
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

Target sentence (output)

Decoder RNN

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

Enclosing: This diagram shows test time behavior: decoder output is fed in ... 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 a m’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

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

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

• 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:

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

  - 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}>)]

\[he\]

\[<\text{START}>\]

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

\[l\]

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

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

-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}> \text{ I}) + -0.9

-1.8 = \log P_{LM}(\text{got}|<\text{START}> \text{ I}) + -0.9

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

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

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

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

Backtrack to obtain the full hypothesis


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

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

- Problem with this: longer hypotheses have lower scores

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

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

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

```
paper jam    Mermelada de papel
```

Abigail See
So is Machine Translation 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. This needs to capture all information about the source sentence. Information bottleneck!

Source sentence (input)

Target sentence (output)

he hit me with a pie <END>

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

Decoder RNN

Source sentence (input)

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

Encoder RNN

Attention scores

Decoder RNN

dot product

Source sentence (input)

il a m’ entarté <START>

Abigail See
Sequence-to-sequence with attention

Encoder RNN

Attention scores

Decoder RNN

Source sentence (input)

\begin{align*}
\text{il} & \quad a & \quad m' & \quad \text{entarté} & \quad <\text{START}> \\
\end{align*}
Sequence-to-sequence with attention

Encoder RNN

Attention scores

Decoder RNN

Source sentence (input)

dot product

Abigail See
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.
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

Source sentence (input)

Decoder RNN

Concatenate attention output with decoder hidden state, then use to compute \( y_1 \) as before

Attention distribution

Attention scores

Abigail See
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 score $e^t$ 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 \textit{values}, and a vector \textit{query}, attention is a technique to compute a weighted sum of the values, dependent on the query.

- We sometimes say that the \textit{query attends to the values}.
- For example, in seq2seq + attention model, each decoder hidden state (query) \textit{attends to} all encoder hidden states (values).
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
Image Captioning
Image Captioning

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

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

test image

Andrej Karpathy
Image Captioning

Andrej Karpathy
Image Captioning

test image

Andrej Karpathy
Image Captioning

Andrej Karpathy
Image Captioning

Caption generated: “straw hat”

<START> 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."

Andrej 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

- English Sentence → RNN encoder → RNN decoder → French Sentence
  - [Sutskever et al. NIPS’14]
- Encode → RNN decoder → Sentence
  - [Donahue et al. CVPR’15] [Vinyals et al. CVPR’15]
- Encode → RNN decoder → Sentence
  - [Venugopalan et. al. NAACL’15]
- RNN encoder → RNN decoder → Sentence
  - [Venugopalan et. al. ICCV’15] (this work)

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

S2VT Overview

Now decode it to a sentence!

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