Modeling Sequential Data with Recurrent Networks

Chris Dyer
DeepMind
Carnegie Mellon University
Outline: Part I

• Neural networks as feature inducers
• Recurrent neural networks
  • Application: language models
• Learning challenges and solutions
  • Vanishing gradients
  • Long short-term memories
  • Gated recurrent units
• Break
Outline: Part II

• RNN performance tuning and implementation tricks

• Bidirectional RNNs
  • Application: better word representations

• Sequence to Sequence transduction with RNNs
  • Applications: machine translation & image caption generation

• Sequences as matrices and attention
  • Application: machine translation
Feature Induction

\[ \hat{y} = Wx + b \]

\[ F = \frac{1}{M} \sum_{i=1}^{M} \| \hat{y}_i - y_i \|_2^2 \]

In linear regression, the goal is to learn \( W \) and \( b \) such that \( F \) is minimized for a dataset \( D \) consisting of \( M \) training instances. An engineer must select/design \( x \) carefully.
Feature Induction

\[ \hat{y} = Wx + b \]

\[ F = \frac{1}{M} \sum_{i=1}^{M} ||\hat{y}_i - y_i||^2_2 \]

In linear regression, the goal is to learn \( W \) and \( b \) such that \( F \) is minimized for a dataset \( D \) consisting of \( M \) training instances. An engineer must select/design \( x \) carefully.

\[ h = g(Vx + c) \]

\[ \hat{y} = Wh + b \]

“nonlinear regression”

Use “naive features” \( x \) and \textit{learn} their transformations (conjunctions, nonlinear transformation, etc.) into \( h \).
Feature Induction

\[ h = g(Vx + c) \]
\[ \hat{y} = Wh + b \]

- What functions can this parametric form compute?
  - If \( h \) is big enough (i.e., enough dimensions), it can represent any vector-valued function to any degree of precision
- This is a much more powerful regression model!
Feature Induction

\[ h = g(Vx + c) \]
\[ \hat{y} = Wh + b \]

- What functions can this parametric form compute?
  - If \( h \) is big enough (i.e., enough dimensions), it can represent any vector-valued function to any degree of precision
- This is a much more powerful regression model!
- You can think of \( h \) as “induced features” in a linear classifier
- The network did the job of a feature engineer
Recurrent Neural Networks

- Lots of interesting data is sequential in nature
  - Words in sentences
  - DNA
  - Stock market returns
  - ...  

- How do we represent an arbitrarily long history?
Recurrent Neural Networks

- Lots of interesting data is sequential in nature
  - Words in sentences
  - DNA
  - Stock market returns
  - ...

- How do we represent an arbitrarily long history?
  - we will train neural networks to build a representation of these arbitrarily big sequences
Recurrent Neural Networks

Feed-forward NN

\[ h = g(Vx + c) \]
\[ \hat{y} = Wh + b \]
Recurrent Neural Networks

Feed-forward NN
\[ h = g(Vx + c) \]
\[ \hat{y} = Wh + b \]

Recurrent NN
\[ h_t = g(Vx_t + Uh_{t-1} + c) \]
\[ \hat{y}_t = Wh_t + b \]
Recurrent Neural Networks

Feed-forward NN
\[
\begin{align*}
    h &= g(Vx + c) \\
    \hat{y} &= Wh + b
\end{align*}
\]

Recurrent NN
\[
\begin{align*}
    h_t &= g(Vx_t + Uh_{t-1} + c) \\
    h_t &= g(V[x_t; h_{t-1}] + c) \\
    \hat{y}_t &= Wh_t + b
\end{align*}
\]
Recurrent Neural Networks

Feed-forward NN
\[ h = g(Vx + c) \]
\[ \hat{y} = Wh + b \]

Recurrent NN
\[ h_t = g(Vx_t + Uh_{t-1} + c) \]
\[ h_t = g(V[x_t; h_{t-1}] + c) \]
\[ \hat{y}_t = Wh_t + b \]
Recurrent Neural Networks

\[ h_t = g(Vx_t + Uh_{t-1} + c) \]
\[ \hat{y}_t = Wh_t + b \]
Recurrent Neural Networks

\[ h_t = g(Vx_t + Uh_{t-1} + c) \]
\[ \hat{y}_t = Wh_t + b \]
Recurrent Neural Networks

\[ h_t = g(Vx_t + Uh_{t-1} + c) \]
\[ \hat{y}_t = Wh_t + b \]
Recurrent Neural Networks

\[ h_t = g(Vx_t + Uh_{t-1} + c) \]
\[ \hat{y}_t = Wh_t + b \]
Recurrent Neural Networks

\[ h_t = g(Vx_t + Uh_{t-1} + c) \]
\[ \hat{y}_t = Wh_t + b \]
Recurrent Neural Networks

\[ h_t = g(Vx_t + Uh_{t-1} + c) \]
\[ \hat{y}_t = Wh_t + b \]

How do we train the RNN’s parameters?
Recurrent Neural Networks

\[ h_t = g(Vx_t + Uh_{t-1} + c) \]
\[ \hat{y}_t = Wh_t + b \]
Recurrent Neural Networks

\[ h_t = g(Vx_t + Uh_{t-1} + c) \]
\[ \hat{y}_t = Wh_t + b \]
Recurrent Neural Networks

\[ h_t = g(Vx_t + Uh_{t-1} + c) \]
\[ \hat{y}_t = Wh_t + b \]
Recurrent Neural Networks

\[ h_t = g(Vx_t + Uh_{t-1} + c) \]
\[ \hat{y}_t = Wh_t + b \]
Recurrent Neural Networks

- The unrolled graph is a well-formed (DAG) computation graph—we can run backprop

  - Parameters are tied across time, derivatives are aggregated across all time steps

  - This is historically called “backpropagation through time” (BPTT)
Parameter Tying

\[ h_t = g(Vx_t + Uh_{t-1} + c) \]
\[ \hat{y}_t = Wh_t + b \]
Parameter Tying

\[ h_t = g(Vx_t + Uh_{t-1} + c) \]
\[ \hat{y}_t = Wh_t + b \]
Parameter Tying

\[ \hat{y}_1, \hat{y}_2, \hat{y}_3, \hat{y}_4 \]
Parameter Tying

\[
\frac{\partial F}{\partial U} = \sum_{t=1}^{4} \frac{\partial h_t}{\partial U} \frac{\partial F}{\partial h_t}
\]
Parameter Tying

Parameter tying also came up when learning the filters in convolutional networks (and in the transition matrices for HMMs!).

\[
\frac{\partial F}{\partial U} = \sum_{t=1}^{4} \frac{\partial h_t}{\partial U} \frac{\partial F}{\partial h_t}
\]
Parameter Tying

• Why do we want to tie parameters?
  • Reduce the number of parameters to be learned
  • Deal with arbitrarily long sequences

• What if we always have short sequences?
  • Maybe you might untie parameters, then. But you wouldn’t have an RNN anymore!
What else can we do?

\[ h_t = g(Vx_t + Uh_{t-1} + c) \]
\[ \hat{y}_t = Wh_t + b \]
“Read and summarize”

\[ h_t = g(V x_t + U h_{t-1} + c) \]

\[ \hat{y} = W h_{|x|} + b \]

Summarize a sequence into a single vector. (This will be useful later...)
View 2: Recursive Definition

- Recall how to construct a list recursively:
  base case
    [] is a list (the empty list)
View 2: Recursive Definition

• Recall how to construct a list recursively:
  base case
  [] is a list (the empty list)

  induction
  [t | h] where t is a list and h is an atom is a list
Recall how to construct a list recursively:
- **Base case**: 
  ```
  [] is a list (the empty list)
  ```

- **Induction**: 
  ```
  [t | h] where t is a list and h is an atom is a list
  ```

RNNs define functions that compute representations recursively according to this definition of a list.

- Define (learn) a representation of the base case
- Learn a representation of the inductive step

**Anything you can construct recursively, you can obtain an “embedding” of with neural networks using this general strategy**
Example: Language Model

\[ \mathbf{u} = \mathbf{Wh} + \mathbf{b} \]

\[ p_i = \frac{\exp u_i}{\sum_j \exp u_j} \]

\[ \mathbf{h} \in \mathbb{R}^d \]

|\( |V| = 100,000 \)

What are the dimensions of \( \mathbf{W} \)?
Example: Language Model

\[ u = Wh + b \]

\[ p_i = \frac{\exp u_i}{\sum_j \exp u_j} \]

\[ h \in \mathbb{R}^d \]

\[ |V| = 100,000 \]

What are the dimensions of \( b \)?
Example: Language Model

\[ u = Wh + b \]

\[ p_i = \frac{\exp u_i}{\sum_j \exp u_j} \]

\[ h \in \mathbb{R}^d \]

\[ |V| = 100,000 \]

What are the dimensions of \( b \)?

\[ p(e) = p(e_1) \times \]

\[ p(e_2 \mid e_1) \times \]

\[ p(e_3 \mid e_1, e_2) \times \]

\[ p(e_4 \mid e_1, e_2, e_3) \times \]

\[ \ldots \]
Example: Language Model

\[ u = Wh + b \]

\[ p_i = \frac{\exp u_i}{\sum_j \exp u_j} \]

\[ h \in \mathbb{R}^d \]

|V| = 100,000

What are the dimensions of \( b \)?

\[ p(e) = p(e_1) \times p(e_2 | e_1) \times p(e_3 | e_1, e_2) \times p(e_4 | e_1, e_2, e_3) \times \ldots \]

histories are sequences of words…
Example: Language Model
Example: Language Model
Example: Language Model
Example: Language Model

\[ p(tom | \langle s \rangle) \]
Example: Language Model

\[ p(tom \mid \langle s \rangle) \]
Example: Language Model

\[ p(tom \mid \langle s \rangle) \]
Example: Language Model

\[ p(tom \mid \langle s \rangle) \times p(likes \mid \langle s \rangle, tom) \]
Example: Language Model

\[
p(tom \mid \langle s \rangle) \times p(likes \mid \langle s \rangle, tom)
\times p(beer \mid \langle s \rangle, tom, likes)
\]
Example: Language Model

\[
p(tom \mid \langle s \rangle) \times p(likes \mid \langle s \rangle, tom) \\
\times p(beer \mid \langle s \rangle, tom, likes) \\
\times p(\langle /s \rangle \mid \langle s \rangle, tom, likes, beer)
\]
Language Model Training

$\hat{p}_1$  softmax  $h_1$

$\hat{p}_2$  softmax  $h_2$

$\hat{p}_3$  softmax  $h_3$

$\hat{p}_4$  softmax  $h_4$

$s$s

<x> x_1 \rightarrow h_0 \rightarrow <s>

<x> x_2 \rightarrow h_1 \rightarrow tom \rightarrow \hat{p}_1

<x> x_3 \rightarrow h_2 \rightarrow likes \rightarrow \hat{p}_2

<x> x_4 \rightarrow h_3 \rightarrow beer \rightarrow \hat{p}_3

<s>
Language Model Training

\[ \hat{p}_1 \rightarrow p_1 \rightarrow h_1 \rightarrow \text{softmax} \rightarrow cost_1 \]

\[ \text{softmax} \rightarrow \hat{p}_2 \rightarrow h_2 \rightarrow \text{softmax} \rightarrow cost_2 \]

\[ \text{softmax} \rightarrow \hat{p}_3 \rightarrow h_3 \rightarrow \text{softmax} \rightarrow cost_3 \]

\[ \text{softmax} \rightarrow \hat{p}_4 \rightarrow h_4 \rightarrow \text{softmax} \rightarrow cost_4 \]

\[ x_1 \rightarrow \text{softmax} \rightarrow h_1 \rightarrow \text{softmax} \rightarrow x_2 \]

\[ x_2 \rightarrow \text{softmax} \rightarrow h_2 \rightarrow \text{softmax} \rightarrow x_3 \]

\[ x_3 \rightarrow \text{softmax} \rightarrow h_3 \rightarrow \text{softmax} \rightarrow x_4 \]

\[ \langle s \rangle \rightarrow \text{softmax} \rightarrow h_0 \rightarrow \text{softmax} \rightarrow \langle s \rangle \]
Language Model Training

\[
\begin{align*}
\hat{p}_1 & \rightarrow \text{softmax} & \text{cost}_1 \\
h_1 & \rightarrow \text{softmax} & \text{cost}_2 \\
h_2 & \rightarrow \text{softmax} & \text{cost}_3 \\
h_3 & \rightarrow \text{softmax} & \text{cost}_4 \\
h_4 & \\
\end{align*}
\]

log loss/cross entropy

\[
\begin{align*}
\text{x}_1 & \rightarrow h_0 \\
\text{x}_2 & \\
\text{x}_3 & \\
\text{x}_4 & \\
\langle s \rangle & \rightarrow h_4 \\
\end{align*}
\]
Language Model Training

\[ \hat{p}_1 \]

\[ \text{softmax} \]

\[ \text{cost}_1 \]

\[ \text{softmax} \]

\[ \text{cost}_2 \]

\[ \text{softmax} \]

\[ \text{cost}_3 \]

\[ \text{softmax} \]

\[ \text{cost}_4 \]
Language Model Training

\[
\begin{align*}
\hat{p}_1 & \rightarrow \text{softmax} \rightarrow h_1 \\
\text{tom} & \rightarrow \text{softmax} \rightarrow \hat{h}_1 \\
\text{likes} & \rightarrow \text{softmax} \rightarrow h_2 \\
\text{beer} & \rightarrow \text{softmax} \rightarrow h_3 \\
\text{}</s> & \rightarrow \text{softmax} \rightarrow h_4 \\
\end{align*}
\]

\[
\begin{align*}
\text{cost}_1 & \rightarrow \log \text{ loss/cross entropy} \\
\text{cost}_2 & \rightarrow \text{cost}_3 \\
\text{cost}_4 & \rightarrow \mathcal{F} \\
\end{align*}
\]
RNN Language Models

• Unlike Markov (n-gram) models, RNNs never forget
  • However we will see they might have trouble learning to use their memories (more soon…)

• Algorithms
  • Sample a sequence from the probability distribution defined by the RNN
  • Train the RNN to minimize cross entropy (aka MLE)
  • What about: what is the most probable sequence?
Questions?
Training Challenges

\[ h_t = g(Vx_t + Uh_{t-1} + c) \]

\[ \hat{y} = Wh|x| + b \]

What happens to gradients as you go back in time?

\[ \frac{\partial F}{\partial F} \]
Training Challenges

\[ h_t = g(Vx_t + Uh_{t-1} + c) \]
\[ \hat{y} = Wh|x| + b \]

What happens to gradients as you go back in time?

\[ \frac{\partial F}{\partial \hat{y}} \frac{\partial F}{\partial F} \]
Training Challenges

\[ h_t = g(Vx_t + Uh_{t-1} + c) \]
\[ \hat{y} = Wh|x| + b \]

What happens to gradients as you go back in time?

\[ \frac{\partial \hat{y}}{\partial h_4} \frac{\partial F}{\partial \hat{y}} \frac{\partial F}{\partial F} \]
Training Challenges

\[ h_t = g(Vx_t + Uh_{t-1} + c) \]
\[ \hat{y} = Wh|x| + b \]

What happens to gradients as you go back in time?

\[ \frac{\partial \hat{y}}{\partial h_3} \frac{\partial \hat{y}}{\partial h_4} \frac{\partial \hat{y}}{\partial F} \frac{\partial \hat{y}}{\partial F} \]
Training Challenges

\[ h_t = g(Vx_t + Uh_{t-1} + c) \]
\[ \hat{y} = Wh|x| + b \]

What happens to gradients as you go back in time?

\[
\begin{align*}
\frac{\partial h_3}{\partial h_2} & \quad \frac{\partial h_4}{\partial h_3} \\
\frac{\partial \hat{y}}{\partial h_4} & \quad \frac{\partial F}{\partial \hat{y}} \quad \frac{\partial F}{\partial F}
\end{align*}
\]
Training Challenges

\[ h_t = g(Vx_t + Uh_{t-1} + c) \]
\[ \hat{y} = Wh|x| + b \]

What happens to gradients as you go back in time?

\[ \frac{\partial F}{\partial h_1} = \frac{\partial h_2}{\partial h_1} \frac{\partial h_3}{\partial h_2} \frac{\partial h_4}{\partial h_3} \frac{\partial \hat{y}}{\partial h_4} \frac{\partial F}{\partial \hat{y}} \frac{\partial F}{\partial F} \]
Training Challenges

\[ h_t = g(Vx_t + Uh_{t-1} + c) \]

\[ \hat{y} = Wh_{|x|} + b \]

What happens to gradients as you go back in time?

\[
\frac{\partial F}{\partial h_1} = \left\{ \frac{\partial h_2}{\partial h_1} \frac{\partial h_3}{\partial h_2} \frac{\partial h_4}{\partial h_3} \frac{\partial \hat{y}}{\partial h_4} \frac{\partial F}{\partial \hat{y}} \frac{\partial F}{\partial F} \right\} \prod_{t=2}^{4} \frac{\partial h_t}{\partial h_{t-1}}
\]
Training Challenges

\[ h_t = g(\mathbf{V}x_t + \mathbf{U}h_{t-1} + \mathbf{c}) \]
\[ \hat{y} = \mathbf{W}h|x| + \mathbf{b} \]

What happens to gradients as you go back in time?

\[ \frac{\partial \mathcal{F}}{\partial h_1} = \left( \prod_{t=2}^{\lfloor x \rfloor} \frac{\partial h_t}{\partial h_{t-1}} \right) \frac{\partial \hat{y}}{\partial h|x|} \frac{\partial \mathcal{F}}{\partial \hat{y}} \frac{\partial \mathcal{F}}{\partial \hat{y}} \]
Training Challenges

\[ h_t = g(Vx_t + Uh_{t-1} + c) \]
\[ \hat{y} = Wh|x| + b \]

What happens to gradients as you go back in time?

\[
\frac{\partial \mathcal{F}}{\partial h_1} = \left( \prod_{t=2}^{|x|} \frac{\partial h_t}{\partial h_{t-1}} \right) \frac{\partial \hat{y}}{\partial h|x|} \frac{\partial \mathcal{F}}{\partial \hat{y}} \frac{\partial \mathcal{F}}{\partial \mathcal{F}}
\]
Training Challenges

\[ h_t = g(Vx_t + Uh_{t-1} + c) \]

\[ \hat{y} = Wh|x| + b \]

What happens to gradients as you go back in time?

\[
\frac{\partial F}{\partial h_1} = \left( \prod_{t=2}^{\mid x \mid} \frac{\partial h_t}{\partial z_t} \frac{\partial z_t}{\partial h_{t-1}} \right) \frac{\partial \hat{y}}{\partial h|x|} \frac{\partial F}{\partial \hat{y}} \frac{\partial F}{\partial F}
\]
Training Challenges

\[ h_t = g(Vx_t + Uh_{t-1} + c) \]
\[ \hat{y} = Wh|x| + b \]
\[ \frac{\partial F}{\partial h_1} = \left( \prod_{t=2}^{\left| x \right|} \frac{\partial h_t}{\partial z_t} \frac{\partial z_t}{\partial h_{t-1}} \right) \frac{\partial \hat{y}}{\partial h|x|} \frac{\partial F}{\partial \hat{y}} \frac{\partial F}{\partial F} \]
Training Challenges

\[ h_t = g(Vx_t + Uh_{t-1} + c) \]

\[ \hat{y} = Wh|\alpha| + b \]

\[ \frac{\partial F}{\partial h_1} = \left( \prod_{t=2}^{t=|\alpha|} \frac{\partial h_t}{\partial z_t} \frac{\partial z_t}{\partial h_{t-1}} \right) \frac{\partial \hat{y}}{\partial h|\alpha|} \frac{\partial F}{\partial \hat{y}} \frac{\partial F}{\partial F} \]

\[ \frac{\partial h_t}{\partial z_t} = \text{diag}(g'(z_t)) \]
Training Challenges

\[ h_t = g(Vx_t + Uh_{t-1} + c) \]

\[ \hat{y} = Wh_{|x|} + b \]

\[
\frac{\partial F}{\partial h_1} = \left( \prod_{t=2}^{\mid x \mid} \frac{\partial h_t}{\partial z_t} \frac{\partial z_t}{\partial h_{t-1}} \right) \frac{\partial \hat{y}}{\partial h_{|x|}} \frac{\partial F}{\partial \hat{y}} \frac{\partial F}{\partial F}
\]

\[
\frac{\partial h_t}{\partial z_t} = \text{diag}(g'(z_t))
\]

\[
\frac{\partial z_t}{\partial h_{t-1}} = ?
\]
Training Challenges

\[ h_t = g(\mathbf{V} \mathbf{x}_t + \mathbf{U} h_{t-1} + c) \]

\[ \hat{y} = \mathbf{W} \mathbf{h} |\mathbf{x}| + b \]

\[ \frac{\partial \mathcal{F}}{\partial h_1} = \left( \prod_{t=2}^{\mathbf{x}} \frac{\partial h_t}{\partial \mathbf{z}_t} \frac{\partial \mathbf{z}_t}{\partial h_{t-1}} \right) \]

\[ \frac{\partial h_t}{\partial \mathbf{z}_t} = \text{diag}(g'(\mathbf{z}_t)) \]

\[ \frac{\partial \mathbf{z}_t}{\partial h_{t-1}} = \mathbf{U} \]
Training Challenges

\[ h_t = g(Vx_t + Uh_{t-1} + c) \]

\[ \hat{y} = Wh|_x + b \]

\[ \frac{\partial F}{\partial h_1} = \left( \prod_{t=2}^{x} \frac{\partial h_t}{\partial z_t} \frac{\partial z_t}{\partial h_{t-1}} \right) \frac{\partial \hat{y}}{\partial h|_x} \frac{\partial F}{\partial \hat{y}} \frac{\partial F}{\partial F} \]

\[ \frac{\partial h_t}{\partial z_t} = \text{diag}(g'(z_t)) \]

\[ \frac{\partial z_t}{\partial h_{t-1}} = U \]

\[ \frac{\partial h_t}{\partial h_{t-1} \partial z_t} \frac{\partial z_t}{\partial h_{t-1}} = \text{diag}(g'(z_t))U \]
Training Challenges

\[ h_t = g(Vx_t + Uh_{t-1} + c) \]
\[ \hat{y} = Wh|_x + b \]
\[ \frac{\partial F}{\partial h_1} = \left( \prod_{t=2}^{z_t} \frac{\partial h_t}{\partial z_t} \frac{\partial z_t}{\partial h_{t-1}} \right) \frac{\partial \hat{y}}{\partial h|_x} \frac{\partial F}{\partial \hat{y}} \frac{\partial F}{\partial F} \]
\[ \frac{\partial F}{\partial h_1} = \left( \prod_{t=2}^{z_t} \text{diag}(g'(z_t))U \right) \frac{\partial \hat{y}}{\partial h|_x} \frac{\partial F}{\partial \hat{y}} \frac{\partial F}{\partial F} \]
Training Challenges

\[ h_t = g(Vx_t + Uh_{t-1} + c) \]
\[ \hat{y} = Wh_1 + b \]

\[ \frac{\partial F}{\partial h_1} = \left( \prod_{t=2}^{\left\lfloor x \right\rfloor} \frac{\partial h_t}{\partial z_t} \frac{\partial z_t}{\partial h_{t-1}} \right) \frac{\partial \hat{y}}{\partial h_1} + \frac{\partial F}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial F} \]

\[ \frac{\partial F}{\partial h_1} = \left( \prod_{t=2}^{\left\lfloor x \right\rfloor} \text{diag}(g'(z_t))U \right) \frac{\partial \hat{y}}{\partial h_1} + \frac{\partial F}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial F} \]

Three cases: largest eigenvalue is 

**exactly 1**; gradient propagation is stable

<1; gradient vanishes (exponential decay)

>1; gradient explodes (exponential growth)
Vanishing Gradients

• In practice, the spectral radius of \( U \) is small, and gradients vanish

• In practice, this means that long-range dependencies are difficult to learn (although in theory they are learnable)

• Solutions
  
  • Better optimizers (second order methods, approximate second order methods)

  • Normalization to keep the gradient norms stable across time

  • Clever initialization so that you at least start with good spectra (e.g., start with random orthonormal matrices)

• Alternative parameterizations: LSTMs and GRUs
Alternative RNNs

- Long short-term memories (LSTMs; Hochreiter and Schmidhuber, 1997)
- Gated recurrent units (GRUs; Cho et al., 2014)
- Intuition instead of multiplying across time (which leads to exponential growth), we want the error to be constant.
- What is a function whose Jacobian has a spectral radius of exactly 1: the identity function
Memory cells

\[ c_t = c_{t-1} + f(x_t) \]
Memory cells

\[ c_t = c_{t-1} + f(x_t) \]

\[ f(v) = \tanh(Wv + b) \]
Memory cells

\[ c_t = c_{t-1} + f(x_t) \]
\[ h_t = g(c_t) \]

\[ f(v) = \tanh(Wv + b) \]
Memory cells

\[ c_t = c_{t-1} + f(x_t) \]
\[ h_t = g(c_t) \]

\[ f(v) = \tanh(Wv + b) \]
Memory cells

\[ c_t = c_{t-1} + f(x_t) \]
\[ h_t = g(c_t) \]

Note:

\[ \frac{\partial c_t}{\partial c_{t-1}} = I \]
\[ c_t = c_{t-1} + f([x_t; h_{t-1}]) \]
\[ h_t = g(c_t) \]
Memory cells

\[ c_t = c_{t-1} + f([x_t; h_{t-1}]) \]

\[ h_t = g(c_t) \]

“Almost constant”

\[ \frac{\partial c_t}{\partial c_{t-1}} = I + \varepsilon \]
Memory cells

\[
c_t = f_t \odot c_{t-1} + i_t \odot f([x_t; h_{t-1}])
\]

\[
h_t = g(c_t)
\]

\[
f_t = \sigma(f_f([x_t; h_{t-1}])))
\]

"forget gate"

\[
i_t = \sigma(f_i([x_t; h_{t-1}]))
\]

"input gate"
Memory cells

\[
\begin{align*}
c_t &= \mathbf{f}_t \odot c_{t-1} + \mathbf{i}_t \odot f([x_t; h_{t-1}]) \\
h_t &= g(c_t) \\
f_t &= \sigma(f_f([x_t; h_{t-1}])) & \text{“forget gate”} \\
i_t &= \sigma(f_i([x_t; h_{t-1}])) & \text{“input gate”}
\end{align*}
\]
Memory cells

\[ c_t = f_t \odot c_{t-1} + i_t \odot f([x_t; h_{t-1}]) \]
\[ h_t = g(c_t) \]
\[ f_t = \sigma(f_f([x_t; h_{t-1}])) \quad \text{“forget gate”} \]
\[ i_t = \sigma(f_i([x_t; h_{t-1}])) \quad \text{“input gate”} \]
\[
\begin{align*}
    c_t &= f_t \odot c_{t-1} + i_t \odot f([x_t; h_{t-1}]) \\
    h_t &= o_t \odot g(c_t) \\
    f_t &= \sigma(f_f([x_t; h_{t-1}])) & \text{“forget gate”} \\
    i_t &= \sigma(f_i([x_t; h_{t-1}])) & \text{“input gate”} \\
    o_t &= \sigma(f_o([x_t; h_{t-1}])) & \text{“output gate”}
\end{align*}
\]
LSTM

c_t = f_t \odot c_{t-1} + i_t \odot f([x_t; h_{t-1}])

h_t = o_t \odot g(c_t)

f_t = \sigma(f_f([x_t; h_{t-1}]))

i_t = \sigma(f_i([x_t; h_{t-1}]))

o_t = \sigma(f_o([x_t; h_{t-1}]))

"forget gate"

"input gate"

"output gate"
LSTM Variant

\[ c_t = (1 - i_t) \odot c_{t-1} + i_t \odot f([x_t; h_{t-1}]) \]
\[ h_t = o_t \odot g(c_t) \]
\[ f_t = \sigma(f_f([x_t; h_{t-1}])) \]
\[ f_t = 1 - i_t \]
\[ i_t = \sigma(f_i([x_t; h_{t-1}])) \]
\[ o_t = \sigma(f_o([x_t; h_{t-1}])) \]

“input gate”

“output gate”
LSTM Variant

c_t = (1 - i_t) \odot c_{t-1} + i_t \odot f([x_t; h_{t-1}])

h_t = o_t \odot g(c_t)

\bar{f}_t = \sigma(f_f([x_t; h_{t-1}]))

f_t = 1 - i_t

i_t = \sigma(f_i([x_t; h_{t-1}]))

"input gate"

o_t = \sigma(f_o([x_t; h_{t-1}]))

"output gate"
Another Visualization

Figure credit: Christopher Olah
Another Visualization

Figure credit: Christopher Olah
Another Visualization

Forget some of the past

Figure credit: Christopher Olah
Forget some of the past
Add new memories

Figure credit: Christopher Olah
Gated Recurrent Units (GRUs)

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

\[
z_t = \sigma(f_z([h_{t-1}; x_t]))
\]

\[
r_t = \sigma(f_r([h_{t-1}; x_t]))
\]

\[
\tilde{h}_t = f([r_t \odot h_{t-1}; x_t]))
\]
Summary

- Better gradient propagation is possible when you use additive rather than multiplicative/highly non-linear recurrent dynamics

\[ RNN \quad h_t = f([x_t; h_{t-1}]) \]

\[ LSTM \quad c_t = f_t \odot c_{t-1} + i_t \odot f([x_t; h_{t-1}]) \]

\[ GRU \quad h_t = (1 - z_t) \odot h_{t-1} + z_t \odot f([x_t; r_t \odot h_{t-1}]) \]
Summary

• Better gradient propagation is possible when you use **additive** rather than multiplicative/highly non-linear recurrent dynamics

RNN  \( h_t = f([x_t; h_{t-1}]) \)

LSTM  \( c_t = f_t \odot c_{t-1} + i_t \odot f([x_t; h_{t-1}]) \)

GRU  \( h_t = (1 - z_t) \odot h_{t-1} + z_t \odot f([x_t; r_t \odot h_{t-1}]) \)
Summary

• Better gradient propagation is possible when you use additive rather than multiplicative/highly non-linear recurrent dynamics

\[
\begin{align*}
\text{RNN} \quad h_t &= f([x_t; h_{t-1}]) \\
\text{LSTM} \quad c_t &= f_t \odot c_{t-1} + i_t \odot f([x_t; h_{t-1}]) \\
\text{GRU} \quad h_t &= (1 - z_t) \odot h_{t-1} + z_t \odot f([x_t; r_t \odot h_{t-1}])
\end{align*}
\]

• Recurrent architectures are an active area of research, requires a mix of mathematical analysis, creativity, problem-specific knowledge

• (LSTMs are hard to beat though!)
Questions?

Break?
A Few Tricks of the Trade

- Depth
- Dropout
- Implementation tricks
“Deep” LSTMs

• This term has been defined several times, but the following is the most standard convention
“Deep” LSTMs

- This term has been defined several times, but the following is the most standard convention
“Deep” LSTMs

- This term has been defined several times, but the following is the most standard convention
“Deep” LSTMs

• This term has been defined several times, but the following is the most standard convention
“Deep” LSTMs

- This term has been defined several times, but the following is the most standard convention
Does Depth Matter?

• Yes, it helps

• It seems to play a less significant role in text than in audio/visual processing
  
  • H1: More transformation of the input is required for ASR, image recognition, etc., than for common text applications (word vectors become customized to be “good inputs” to RNNs whereas you’re stuck with what nature gives you for speech/vision)
  
  • H2: less effort has been made to find good architectures (RNNs are expensive to train; have been widely used for less long)
  
  • H3: back prop through time + depth is hard and we need better optimizers
  
  • Many other possibilities…
  
• 2-8 layers seems to be standard

• Input “skip” connections are used often but by no means universally
Dropout and Deep LSTMs

• Applying dropout layers requires some care
Dropout and Deep LSTMs

• Apply dropout between layers, but not on the recurrent connections
Implementation Details

• **For speed**
  - Use diagonal matrices instead of full matrices (esp. for gates)
  - Concatenate parameter matrices for all gates and do a single matrix-vector/matrix multiplication
  - Use optimized implementations (from NVIDIA)
  - Use GRUs or reduced-gate variant of LSTMs

• **For learning speed and performance**
  - Initialize so that the bias on the forget gate is large (intuitively: at the beginning of training, the signal from the past is unreliable)
  - Use random orthogonal matrices to initialize the square matrices
Implementation Details: Minibatching

- GPU hardware is
  - pretty fast for elementwise operations (IO bound- can’t get enough data through the GPU)
  - very fast for matrix-matrix multiplication (usually compute bound - the GPU will work at 100% capacity, and GPU cores are fast)

- RNNs, LSTMs, GRUs all consist of
  - lots of elementwise operations (addition, multiplication, nonlinearities, …)
  - lots of matrix-vector products

- Minibatching: convert many matrix-vector products into a single matrix-matrix multiplication
Minibatching

Single-instance RNN

\[ h_t = g(Vx_t + Uh_{t-1} + c) \]
\[ \hat{y}_t = Wh_t + b \]
Minibatching

Single-instance RNN

\[ h_t = g(Vx_t + Uh_{t-1} + c) \]
\[ \hat{y}_t = Wh_t + b \]

Minibatch RNN

\[ H_t = g(VX_t + UH_{t-1} + c) \]
\[ \hat{Y}_t = WH_t + b \]

We batch across instances, not across time.
Minibatching

Single-instance RNN

\[ h_t = g(Vx_t + Uh_{t-1} + c) \]
\[ \hat{y}_t = Wh_t + b \]

Minibatch RNN

\[ H_t = g(VX_t + UH_{t-1} + c) \]
\[ \hat{Y}_t = WH_t + b \]

We batch across instances, not across time.
Minibatching

Single-instance RNN

$h_t = g(Vx_t + Uh_{t-1} + c)$

$\hat{y}_t = Wh_t + b$

Minibatch RNN

$H_t = g(VX_t + UH_{t-1} + c)$

$\hat{Y}_t = WH_t + b$

We batch across instances, not across time.
Minibatching

Single-instance RNN
\[ h_t = g(Vx_t + Uh_{t-1} + c) \]
\[ \hat{y}_t = Wh_t + b \]

Minibatch RNN
\[ H_t = g(VX_t + UH_{t-1} + c) \]
\[ \hat{Y}_t = WH_t + b \]

We batch across instances, not across time.
Minibatching

- The challenge with working with mini batches of sequences is ... sequences are of different lengths

- This usually means you bucket training instances based on similar lengths, and pad with 0’s

  - Be careful when padding not to back propagate a non-zero value!

- Manual minibatching convinces me that this is the era of assembly language programming for neural networks. Make the future an easier place to program!
Questions?
Bidirectional RNNs

• We can read a sequence from left to right to obtain a representation

• Or we can read it from right to left

• Or we can read it from both and combine the representations
Word Embedding Models

car

Memorize

Generalize
Word Embedding Models

Memorize

Generalize
Word Embedding Models

![Diagram of word embedding models]

- **Memorize**
- **Generalize**
Word Embedding Models

Memorize

Generalize
Word Embedding Models

Memorize

Generalize
Word Embedding Models

Memorize

Generalize
Word Embedding Models

Memorize

Generalize
Word Embedding Models

Memorize

Generalize
Word Embedding Models

Memorize

Generalize
Word Embedding Models

Memorize

Generalize
## Language modeling

CharLSTM > Word Lookup

<table>
<thead>
<tr>
<th>Language</th>
<th>ppl Words</th>
<th>ppl Chars</th>
<th>Δ</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Analytic</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>English</td>
<td>59.4</td>
<td>57.4</td>
<td>-2.0</td>
</tr>
</tbody>
</table>
## Language modeling

**CharLSTM > Word Lookup**

<table>
<thead>
<tr>
<th>Language</th>
<th>ppl (Words)</th>
<th>ppl (Chars)</th>
<th>Δ</th>
</tr>
</thead>
<tbody>
<tr>
<td>English</td>
<td>59.4</td>
<td>57.4</td>
<td>-2.0</td>
</tr>
<tr>
<td>Turkish</td>
<td>44.0</td>
<td>32.9</td>
<td>-11.1</td>
</tr>
<tr>
<td>German</td>
<td>59.1</td>
<td>43.0</td>
<td>-16.1</td>
</tr>
</tbody>
</table>

*Analytic*

*Agglutinative*
## Language modeling

**CharLSTM > Word Lookup**

<table>
<thead>
<tr>
<th>Language</th>
<th>ppl Words</th>
<th>ppl Chars</th>
<th>Δ</th>
</tr>
</thead>
<tbody>
<tr>
<td>English</td>
<td>59.4</td>
<td>57.4</td>
<td>-2.0</td>
</tr>
<tr>
<td>Turkish</td>
<td>44.0</td>
<td>32.9</td>
<td>-11.1</td>
</tr>
<tr>
<td>German</td>
<td>59.1</td>
<td>43.0</td>
<td>-16.1</td>
</tr>
<tr>
<td>Portuguese</td>
<td>46.2</td>
<td>40.9</td>
<td>-5.3</td>
</tr>
<tr>
<td>Catalan</td>
<td>35.3</td>
<td>34.9</td>
<td>-0.4</td>
</tr>
</tbody>
</table>

- **Analytic**
- **Agglutinative**
- **Fusional**
## Language modeling

**CharLSTM > Word Lookup**

| Language   | ppl Words | ppl Chars | $\Delta$ | $|\theta|$ Words | $|\theta|$ Chars |
|------------|-----------|-----------|----------|----------------|----------------|
| English    | 59.4      | 57.4      | -2.0     | 4.3M           | 0.18M          |
| Turkish    | 44.0      | 32.9      | -11.1    | 5.7M           | 0.17M          |
| German     | 59.1      | 43.0      | -16.1    | 6.3M           | 0.18M          |
| Portuguese | 46.2      | 40.9      | -5.3     | 4.2M           | 0.18M          |
| Catalan    | 35.3      | 34.9      | -0.4     | 4.3M           | 0.18M          |

**Categories**

- **Analytic**
- **Agglutinative**
- **Fusional**
Language modeling
Word similarities

<table>
<thead>
<tr>
<th>increased</th>
<th>John</th>
</tr>
</thead>
<tbody>
<tr>
<td>reduced</td>
<td>Richard</td>
</tr>
<tr>
<td>improved</td>
<td>George</td>
</tr>
<tr>
<td>expected</td>
<td>James</td>
</tr>
<tr>
<td>decreased</td>
<td>Robert</td>
</tr>
<tr>
<td>targeted</td>
<td>Edward</td>
</tr>
<tr>
<td>increased</td>
<td>John</td>
</tr>
<tr>
<td>-----------</td>
<td>------</td>
</tr>
<tr>
<td>reduced</td>
<td>Richard</td>
</tr>
<tr>
<td>improved</td>
<td>George</td>
</tr>
<tr>
<td>expected</td>
<td>James</td>
</tr>
<tr>
<td>decreased</td>
<td>Robert</td>
</tr>
<tr>
<td>targeted</td>
<td>Edward</td>
</tr>
</tbody>
</table>
Questions?
Recurrent Neural Networks (RNNs)

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

c = RNN(x)

What is a vector representation of a sequence $x$?
Recurrent Neural Networks (RNNs)

What is a vector representation of a sequence \( \mathbf{x} \)?
Recurrent Neural Networks (RNNs)

What is a vector representation of a sequence $\mathbf{x}$?

$h_t = f(h_{t-1}, \mathbf{x}_t)$

c = RNN($\mathbf{x}$)

$x = \text{START} \ x_1 \ x_2 \ x_3 \ x_4$
Recurrent Neural Networks (RNNs)

What is a vector representation of a sequence $\mathbf{x}$?
What is a vector representation of a sequence $x$?
Recurrent Neural Networks (RNNs)

What is a vector representation of a sequence $\mathbf{x}$?

\[ c = \text{RNN}(\mathbf{x}) \]

\[ h_t = f(h_{t-1}, x_t) \]

$0 \rightarrow \mathbf{x} = \text{START} \xrightarrow{} x_1 \xrightarrow{} x_2 \xrightarrow{} x_3 \xrightarrow{} x_4$
RNN Encoder-Decoders

\[ c = \text{RNN}(x) \]

What is the probability of a sequence \( y \mid x \)?

Cho et al. (2014); Sutskever et al. (2014)
RNN Encoder-Decoders

$c = \text{RNN}(x)$

*What is the probability of a sequence $y \mid x$?*

Cho et al. (2014); Sutskever et al. (2014)
RNN Encoder-Decoders

$$c = \text{RNN}(x)$$

What is the probability of a sequence $y | x$?

Cho et al. (2014); Sutskever et al. (2014)
RNN Encoder-Decoder

\[ c = \text{RNN}(x) \]

What is the probability of a sequence \( y \mid x \) ?

Cho et al. (2014); Sutskever et al. (2014)
RNN Encoder-Decoders

\[ c = \text{RNN}(x) \]

What is the probability of a sequence \( y \mid x \) ?

Cho et al. (2014); Sutskever et al. (2014)
RNN Encoder-Decoder

What is the probability of a sequence $y \mid x$?

Cho et al. (2014); Sutskever et al. (2014)
What is the probability of a sequence $y \mid x$?

Cho et al. (2014); Sutskever et al. (2014)
RNN Encoder-Decoder

What is the probability of a sequence \( y \mid x \)?

Cho et al. (2014); Sutskever et al. (2014)

\[
c = \text{RNN}(x) \]
\[
y \mid c \sim \text{RNNLM}(c)
\]
RNN Encoder-Decoder

\[ c = \text{RNN}(x) \]
\[ y \mid c \sim \text{RNNLM}(c) \]

**What is the probability of a sequence** \( y \mid x \) **?**

Cho et al. (2014); Sutskever et al. (2014)
RNN Encoder-Decoders

\[ c = \text{RNN}(x) \]
\[ y \mid c \sim \text{RNNLM}(c) \]

What is the probability of a sequence \( y \mid x \)?

Cho et al. (2014); Sutskever et al. (2014)
RNN Encoder-Decoders

$c = \text{RNN}(x)$

$y \mid c \sim \text{RNNLM}(c)$

**What is the probability of a sequence $y \mid x$?**

Cho et al. (2014); Sutskever et al. (2014)
ich habe Hunger
I'm hungry

ich habe Hunger
## Sutskever et al. (2014)

<table>
<thead>
<tr>
<th>Method</th>
<th>test BLEU score (ntst14)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bahdanau et al. [2]</td>
<td>28.45</td>
</tr>
<tr>
<td>Baseline System [29]</td>
<td>33.30</td>
</tr>
<tr>
<td>Single forward LSTM, beam size 12</td>
<td>26.17</td>
</tr>
<tr>
<td>Single reversed LSTM, beam size 12</td>
<td>30.59</td>
</tr>
<tr>
<td>Ensemble of 5 reversed LSTMs, beam size 1</td>
<td>33.00</td>
</tr>
<tr>
<td>Ensemble of 2 reversed LSTMs, beam size 12</td>
<td>33.27</td>
</tr>
<tr>
<td>Ensemble of 5 reversed LSTMs, beam size 2</td>
<td>34.50</td>
</tr>
<tr>
<td>Ensemble of 5 reversed LSTMs, beam size 12</td>
<td><strong>34.81</strong></td>
</tr>
</tbody>
</table>
Ensembles of NNs

• Sutskever noticed that their single models did not work well

• But by combining N independently trained models and obtaining a “consensus”, the performance could be improved a lot

• This is called ensembling.
Encode anything as a vector!
Encode anything as a vector!
Encode anything as a vector!

Encode anything as a vector!
Limitations

• A possible conceptual problem
  • Sentences have unbounded lengths
  • Vectors have finite capacity

• A possible practical problem
  • Distance between “translations” and their sources are distant- can LSTMs learn this?
Two Goals

• Represent a source sentence as a matrix
• Generate a target sentence from a matrix

• These two steps are:
  • An algorithm for neural MT
  • A way of introducing attention
Sentences as Matrices

• Problem with the fixed-size vector model in translation (maybe in images?)

  • Sentences are of different sizes but vectors are of the same size

• Solution: use matrices instead

  • Fixed number of rows, but number of columns depends on the number of words

  • Usually $|f| = \#\text{cols}$
Sentences as Matrices

Ich möchte ein Bier
Sentences as Matrices

Ich möchte ein Bier

Mach’s gut
Sentences as Matrices

Ich möchte ein Bier
Mach’s gut
Die Wahrheiten der Menschen sind die unwiderlegbaren Irrtümer
Sentences as Matrices

Ich möchte ein Bier  Mach’s gut  Die Wahrheiten der Menschen sind die unwiderlegbaren Irrtümer

Question: How do we build these matrices?
With Concatenation

- Each word type is represented by an n-dimensional vector
- Take all of the vectors for the sentence and concatenate them into a matrix
- Simplest possible model
  - So simple, no one has bothered to publish how well/badly it works!
Ich möchte ein Bier.
Ich möchte ein Bier
Ich möchte ein Bier

$f_i = x_i$

$x_1 \rightarrow x_2 \rightarrow x_3 \rightarrow x_4$

Ich möchte ein Bier

$F \in \mathbb{R}^{n \times |f|}$
With Convolutional Nets

• Apply convolutional networks to transform the naive concatenated matrix to obtain a context-dependent matrix.

• Closely related to the first “modern” neural translation model proposed (Kalchbrenner et al., 2013).

  • No one has been using convnets lately in MT (including Kalchbrenner et al, who are using BiLSTMs these days).

• Note: convnets usually have a “pooling” operation at the top level that results in a fixed-sized representation. For sentences, it is probably good to leave this out.
Ich möchte ein Bier.
Ich möchte ein Bier
Ich möchte ein Bier.
Ich möchte ein Bier
Ich möchte ein Bier

\[
F \in \mathbb{R}^{f(n) \times g(|\mathbf{f}|)}
\]

Ich möchte ein Bier
With Bidirectional RNNs

- By far the most widely used matrix representation, due to Bahdanau et al (2015)

- One column per word

- Each column (word) has two halves concatenated together:
  - a “forward representation”, i.e., a word and its left context
  - a “reverse representation”, i.e., a word and its right context

- Implementation: bidirectional RNNs (GRUs or LSTMs) to read from left to right and right to left, concatenate representations
Ich möchte ein Bier
Ich möchte ein Bier!
Ich möchte ein Bier!
Ich möchte ein Bier
Ich möchte ein Bier!

$f_i = [\text{h}_i; \overrightarrow{h}_i]$
Ich möchte ein Bier!

\[
f_i = [\vec{h}_i; \vec{h}_i]
\]
Ich möchte ein Bier
Ich möchte ein Bier

\[ f_i = [\vec{h}_i; \vec{h}_i] \]

\[ \vec{h}_1 \quad \vec{h}_2 \quad \vec{h}_3 \quad \vec{h}_4 \]

\[ \rightarrow \vec{h}_1 \quad \rightarrow \vec{h}_2 \quad \rightarrow \vec{h}_3 \quad \rightarrow \vec{h}_4 \]

\[ x_1 \quad x_2 \quad x_3 \quad x_4 \]

\[ \text{Ich} \quad \text{möchte} \quad \text{ein} \quad \text{Bier} \]

\[ F \in \mathbb{R}^{2n \times |f|} \]
Where are we in 2016?

• There are lots of ways to construct $F$

  • Very little (published?) work comparing them

  • There are many more undiscovered things out there

    • convolutions are particularly interesting and under-explored

    • syntactic information could help

  • My intuition is simpler/faster models will work well for the matrix encoding part—context dependencies are limited in language.

  • try something with phrase types instead of word types?
Generation from Matrices

• We have a matrix $F$ representing the input, now we need to generate from it

• Bahdanau et al. (2015) were the first to propose using **attention** for translating from matrix-encoded sentences

• High-level idea
  
  • Generate the output sentence word by word using an RNN
  
  • At each output position $t$, the RNN receives **two** inputs (in addition to any recurrent inputs)
    
    • a fixed-size vector embedding of the previously generated output symbol $e_{t-1}$
    
    • a fixed-size vector encoding a “view” of the input matrix
  
  • How do we get a fixed-size vector from a matrix that changes over time?
    
    • Bahdanau et al: do a weighted sum of the columns of $F$ (i.e., words) based on how important they are **at the current time step**. (i.e., just a matrix-vector product $F a_t$)
    
    • The weighting of the input columns at each time-step $(a_t)$ is called **attention**
Recall RNNs...
Recall RNNs…
Recall RNNs...
Recall RNNs...
Recall RNNs...
Recall RNNs...
Ich möchte ein Bier
Ich möchte ein Bier
Ich möchte ein Bier
Ich möchte ein Bier
Ich möchte ein Bier
Ich möchte ein Bier
I'd like a beer

Ich möchte ein Bier
I'd like a beer STOP

Ich möchte ein Bier

Attention history:
Attention

• How do we know what to attend to at each time-step?

• That is, how do we compute $a_t$?
Computing Attention

• At each time step (one time step = one output word), we want to be able to “attend” to different words in the source sentence

• We need a weight for every word: this is an $|f|$-length vector $a_t$

• Here is a simplified version of Bahdanau et al.’s solution

  • Use an RNN to predict model output, call the hidden states $s_t$
    ($s_t$ has a fixed dimensionality, call it $m$)
Computing Attention

• At each time step (one time step = one output word), we want to be able to “attend” to different words in the source sentence

• We need a weight for every word: this is an $|f|$-length vector $a_t$

• Here is a simplified version of Bahdanau et al.’s solution

  • Use an RNN to predict model output, call the hidden states $s_t$
    (the $s_t$ has a fixed dimensionality, call it $m$)
  • At time $t$ compute the **expected input embedding** $r_t = Vs_{t-1}$
    ($V$ is a learned parameter)
Computing Attention

• At each time step (one time step = one output word), we want to be able to “attend” to different words in the source sentence

• We need a weight for every word: this is an $|f|$-length vector $a_t$

• Here is a simplified version of Bahdanau et al.’s solution

  • Use an RNN to predict model output, call the hidden states $s_t$  
    ($s_t$ has a fixed dimensionality, call it $m$)
  
  • At time $t$ compute the expected input embedding $r_t = Vs_{t-1}$  
    ($V$ is a learned parameter)
  
  • Take the dot product with every column in the source matrix to compute the attention energy $u_t = F^\top r_t$ (called $e_t$ in the paper)  
    (Since $F$ has $|f|$ columns, $u_t$ has $|f|$ rows)
Computing Attention

• At each time step (one time step = one output word), we want to be able to “attend” to different words in the source sentence

• We need a weight for every word: this is an $|f|$-length vector $a_t$

• Here is a simplified version of Bahdanau et al.’s solution

  • Use an RNN to predict model output, call the hidden states $s_t$ ($s_t$ has a fixed dimensionality, call it $m$)

  • At time $t$ compute the **expected input embedding** $r_t = Vs_{t-1}$
    (V is a learned parameter)

  • Take the dot product with every column in the source matrix to compute the **attention energy** $u_t = F^\top r_t$ (called $e_t$ in the paper)
    (Since $F$ has $|f|$ columns, $u_t$ has $|f|$ rows)

  • Exponentiate and normalize to 1: $a_t = \text{softmax}(u_t)$
    (called $\alpha_t$ in the paper)
Computing Attention

- At each time step (one time step = one output word), we want to be able to “attend” to different words in the source sentence

- We need a weight for every word: this is an $|f|$-length vector $a_t$

- Here is a simplified version of Bahdanau et al.’s solution

  - Use an RNN to predict model output, call the hidden states $s_t$ ($s_t$ has a fixed dimensionality, call it $m$)
  
  - At time $t$ compute the **expected input embedding** $r_t = Vs_{t-1}$
    
    (V is a learned parameter)
  
  - Take the dot product with every column in the source matrix to compute the **attention energy** $u_t = F^\top r_t$ (called $e_t$ in the paper)
    
    (Since $F$ has $|f|$ columns, $u_t$ has $|f|$ rows)
  
  - Exponentiate and normalize to 1: $a_t = \text{softmax}(u_t)$
    
    (called $\alpha_t$ in the paper)
  
  - Finally, the **input source vector** for time $t$ is $c_t = Fa_t$
Nonlinear Attention-Energy Model

• In the actual model, Bahdanau et al. replace the dot product between the columns of $\mathbf{F}$ and $\mathbf{r}_t$ with an MLP:

$$u_t = \mathbf{F}^\top \mathbf{r}_t$$

(simple model)
Nonlinear Attention-Energy Model

• In the actual model, Bahdanau et al. replace the dot product between the columns of $F$ and $r_t$ with an MLP:

$$u_t = F^\top r_t \quad \text{(simple model)}$$

$$u_t = v^\top \tanh(WF + r_t) \quad \text{(Bahdanau et al)}$$
Nonlinear Attention-Energy Model

• In the actual model, Bahdanau et al. replace the dot product between the columns of $F$ and $r_t$ with an MLP:

$$u_t = F^\top r_t$$  \hspace{1cm} \text{(simple model)}

$$u_t = v^\top \tanh(WF + r_t) \hspace{1cm} \text{(Bahdanau et al)}$$

• Here, $W$ and $v$ are learned parameters of appropriate dimension and $+$ “broadcasts” over the $|f|$ columns in $WF$

• This can learn more complex interactions

• It is unclear if the added complexity is necessary for good performance
Putting it all together

\[
F = \text{EncodeAsMatrix}(f)
\]
\[
e_0 = \langle s \rangle
\]
\[
s_0 = w \quad (\text{Learned initial state; Bahdanau uses } U \hat{h}_1)
\]
\[
t = 0
\]
\[
\text{while } e_t \neq \langle /s \rangle : 
\]
\[
t = t + 1
\]
\[
r_t = Vs_{t-1}
\]
\[
\{ u_t = v^\top \tanh(WF + r_t) \} \quad (\text{Compute attention})
\]
\[
a_t = \text{softmax}(u_t)
\]
\[
c_t = Fa_t
\]
\[
s_t = \text{RNN}(s_{t-1}, [e_{t-1}; c_t]) \quad (e_{t-1} \text{ is a learned embedding of } e_t)
\]
\[
y_t = \text{softmax}(Ps_t + b) \quad (P \text{ and } b \text{ are learned parameters})
\]
\[
e_t \mid e_{<t} \sim \text{Categorical}(y_t)
\]
Putting it all together

\[ F = \text{EncodeAsMatrix}(f) \]
\[ e_0 = \langle s \rangle \]
\[ s_0 = w \text{ (Learned initial state; Bahdanau uses } U\bar{h}_1) \]
\[ t = 0 \]
\[ \text{while } e_t \neq \langle /s \rangle : \]
\[ t = t + 1 \]
\[ r_t = Vs_{t-1} \]
\[ u_t = v^\top \tanh(WF + r_t) \]
\[ a_t = \text{softmax}(u_t) \]
\[ c_t = Fa_t \]
\[ s_t = \text{RNN}(s_{t-1}, [e_{t-1}; c_t]) \text{ (} e_{t-1} \text{ is a learned embedding of } e_t) \]
\[ y_t = \text{softmax}(Ps_t + b) \text{ (} P \text{ and } b \text{ are learned parameters}) \]
\[ e_t | e_{<t} \sim \text{Categorical}(y_t) \]
Putting it all together

\[ F = \text{EncodeAsMatrix}(f) \quad \text{(Part 1 of lecture)} \]
\[ e_0 = \langle s \rangle \]
\[ s_0 = w \quad \text{(Learned initial state; Bahdanau uses } U \widehat{h}_1) \]
\[ t = 0 \]
\[ X = WF \]

while \( e_t \neq \langle /s \rangle \):
\[ t = t + 1 \]
\[ r_t = Vs_{t-1} \]
\[ u_t = v^\top \tanh(WF + r_t) \]
\[ a_t = \text{softmax}(u_t) \]
\[ c_t = Fa_t \]
\[ s_t = \text{RNN}(s_{t-1}, [e_{t-1}; c_t]) \quad \text{(} e_{t-1} \text{ is a learned embedding of } e_t) \]
\[ y_t = \text{softmax}(Ps_t + b) \quad \text{(} P \text{ and } b \text{ are learned parameters)} \]
\[ e_t \mid e_{<t} \sim \text{Categorical}(y_t) \]
Putting it all together

\[
F = \text{EncodeAsMatrix}(f) \quad \text{(Part 1 of lecture)}
\]

\[
e_0 = \langle s \rangle
\]

\[
s_0 = w \quad \text{(Learned initial state; Bahdanau uses } U \hat{h}_1 \text{)}
\]

\[
t = 0
\]

\[
X = WF
\]

\[
\text{while } e_t \neq \langle /s \rangle : \\
\quad t = t + 1 \\
\quad r_t = Vs_{t-1} \\
\quad u_t = v^\top \tanh(X + r_t) \\
\quad a_t = \text{softmax}(u_t) \\
\quad c_t = Fa_t \\
\quad s_t = \text{RNN}(s_{t-1}, [e_{t-1}; c_t]) \quad \text{(} e_{t-1} \text{ is a learned embedding of } e_t \text{)} \\
\quad y_t = \text{softmax}(Ps_t + b) \quad \text{(} P \text{ and } b \text{ are learned parameters)} \\
\quad e_t | e_{<t} \sim \text{Categorical}(y_t)
\]
Summary

• Attention is closely related to “pooling” operations in convnets (and other architectures)

• Bahdanau’s attention model seems to only cares about “content”
  • No obvious bias in favor of diagonals, short jumps, fertility, etc.
  • Some work has begun to add other “structural” biases (Luong et al., 2015; Cohn et al., 2016), but there are lots more opportunities

• Attention is similar to alignment, but there are important differences
  • alignment makes stochastic but hard decisions. Even if the alignment probability distribution is “flat”, the model picks one word or phrase at a time
  • attention is “soft” (you add together all the words). Big difference between “flat” and “peaked” attention weights
Attention and Translation

• Cho’s question: does a translator read and memorize the input sentence/document and then generate the output?

• Compressing the entire input sentence into a vector basically says “memorize the sentence”

• Common sense experience says translators refer back and forth to the input. (also backed up by eye-tracking studies)

• Should humans be a model for machines?
Questions?