Sequence Modeling with Neural Networks

Harini Suresh
What is a sequence?

- “I took the dog for a walk this morning.”  
  - sentence
-  
  - function
-  
  - speech waveform
Successes of deep models

Machine translation

- Perfect translation
- Human
- Neural (GNMT)
- Phrase-based (PBMT)

<table>
<thead>
<tr>
<th>Translation model</th>
<th>English &gt; Spanish</th>
<th>English &gt; French</th>
<th>Spanish &gt; English</th>
<th>French &gt; English</th>
<th>Chinese &gt; English</th>
</tr>
</thead>
<tbody>
<tr>
<td>Translation quality</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

Question Answering

Super Bowl 50 was an American football game to determine the champion of the National Football League (NFL) for the 2015 season. The American Football Conference (AFC) champion Denver Broncos defeated the National Football Conference (NFC) champion Carolina Panthers 24–10 to earn their third Super Bowl title. The game was played on February 7, 2016, at Levi's Stadium in the San Francisco Bay Area at Santa Clara, California. As this was the 50th Super Bowl, the league emphasized the “golden anniversary” with various gold-themed initiatives, as well as temporarily suspending the tradition of naming each Super Bowl game with Roman numerals (under which the game would have been known as “Super Bowl L”), so that the logo could prominently feature the Arabic numerals 50.

Super Bowl 50 decided the NFL champion for what season?

Ground Truth Answers: 2015, the 2015 season, 2015
Prediction: 2015

Right: https://rajpurkar.github.io/SQuAD-explorer/
how do we model sequences?
idea: represent a sequence as a bag of words

“I dislike rain.”

[0 1 0 1 0 0 0 1]

prediction
problem: bag of words does not preserve order
**problem:** bag of words does not preserve order

“The food was good, not bad at all.”

vs

“The food was bad, not good at all.”
idea: maintain an ordering within feature vector

[0 0 0 1 0 0 0 1 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1]

On Monday it was snowing

One hot feature vector indicates what each word is

prediction
**problem:** hard to deal with different word orders

“On Monday, it was snowing.”

vs

“It was snowing on Monday.”
**problem:** hard to deal with different word orders

\[
\begin{bmatrix}
0 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 \\
\end{bmatrix}
\]

On Monday it was snowing

vs

\[
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 \\
\end{bmatrix}
\]

It was snowing on Monday
problem: hard to deal with different word orders

“On Monday it was snowing.”
vs
“It was snowing on Monday.”

We would have to relearn the rules of language at each point in the sentence.
idea: markov models
**Problem:** we can’t model long-term dependencies

**Markov assumption:** each state depends only on the last state.
problem: we can’t model long-term dependencies

“In France, I had a great time and I learnt some of the _____ language.”

We need information from the far past and future to accurately guess the correct word.
let’s turn to **recurrent neural networks**! (RNNs)

1. to maintain **word order**
2. to **share parameters** across the sequence
3. to keep track of **long-term dependencies**
example network:

input

hidden

output
let's take a look at this one hidden unit
RNNS **remember** their previous state:

\[ x_0 : \text{vector representing first word} \]
\[ s_0 : \text{cell state at } t = 0 \text{ (some initialization)} \]
\[ s_1 : \text{cell state at } t = 1 \]

\[ s_1 = \tanh(Wx_0 + Us_0) \]

\[ W, U : \text{weight matrices} \]
RNNS remember their previous state:

\[ x_1 : \text{vector representing second word} \]
\[ s_1 : \text{cell state at } t = 1 \]
\[ s_2 : \text{cell state at } t = 2 \]

\[ s_2 = \tanh(Wx_1 + Us_1) \]

\( W, U : \text{weight matrices} \)
“unfolding” the RNN across time:
“unfolding” the RNN across time:

\[ x_0, W \rightarrow s_0, U \rightarrow x_1, W \rightarrow s_1, U \rightarrow x_2, W \rightarrow s_2, \ldots \]

notice that W and U stay the same!
“unfolding” the RNN across time:

$s_n$ can contain information from all past timesteps
KING LEAR:
O, 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.
possible task: language model

\[ y_0 \quad alas \]

\[ y_1 \quad my \]

\[ y_2 \quad honor \]

\[ v \]

\[ w \]

\[ s_0 \quad x_0 \]

\[ s_1 \quad x_1 \]

\[ s_2 \quad x_2 \]

\[ \ldots \]

\[ y_i \] is actually a probability distribution over possible next words, aka a softmax
possible task: language model

37:29 The righteous shall inherit the land, and leave it for an inheritance unto the children of Gad according to the number of steps that is linear in $b$.

hath it not been for the singular taste of old Unix, “new Unix” would not exist.

http://kingjamesprogramming.tumblr.com/
possible task: classification (i.e. sentiment)

@HVSVN

Don't fly with @British_Airways. They can't keep track of your luggage.

:(

Kim Kardashian

Happy Birthday to my best friend, the heart of my life, my soul!!! I love you beyond words! instagram.com/p/aTgfI-OS-a/

:)
possible task: classification (i.e. sentiment)

\[ y \text{ is a probability distribution over possible classes (like positive, negative, neutral), aka a softmax} \]
possible task: machine translation
how do we train an RNN?
how do we **train** an RNN?

**backpropagation!**

(through time)
remember: **backpropagation**

1. **take the derivative** (gradient) of the loss with respect to each parameter

2. **shift parameters in the opposite direction** in order to minimize loss
we have a **loss at each timestep:**

(since we’re making a prediction at each timestep)
we have a **loss at each timestep**: 
(since we’re making a prediction at each timestep)
we sum the losses across time:

\[ \text{loss at time } t = J_t(\Theta) \]

\[ \text{total loss} = J(\Theta) = \sum_t J_t(\Theta) \]

\[ \Theta = \text{our parameters, like weights} \]
what are our gradients?

we sum gradients across time for each parameter $P$:

$$\frac{\partial J}{\partial P} = \sum_t \frac{\partial J_t}{\partial P}$$
let's try it out for $W$ with the chain rule:

\[
\frac{\partial J}{\partial W} = \sum_t \frac{\partial J_t}{\partial W}
\]
let’s try it out for $W$ with the **chain rule**:

$$
\frac{\partial J}{\partial W} = \sum_t \frac{\partial J_t}{\partial W}
$$

so let’s take a single timestep $t$:
let’s try it out for $W$ with the **chain rule**:

\[
\frac{\partial J}{\partial W} = \sum_{t} \frac{\partial J_t}{\partial W}
\]

so let’s take a single timestep $t$: 

\[
\frac{\partial J_2}{\partial W}
\]
let’s try it out for $W$ with the **chain rule**:

\[
\frac{\partial J}{\partial W} = \sum_t \frac{\partial J_t}{\partial W}
\]

so let’s take a single timestep $t$:

\[
\frac{\partial J_t}{\partial W} = \frac{\partial J_t}{\partial y_t}
\]
let’s try it out for $W$ with the **chain rule**:

\[
\frac{\partial J}{\partial W} = \sum_t \frac{\partial J_t}{\partial W}
\]

so let’s take a single timestep $t$:

\[
\frac{\partial J_t}{\partial W} = \frac{\partial J_t}{\partial y_t} \frac{\partial y_t}{\partial s_t} \frac{\partial s_t}{\partial W}
\]
let’s try it out for $W$ with the **chain rule**:

$$\frac{\partial J}{\partial W} = \sum_t \frac{\partial J_t}{\partial W}$$

so let’s take a single timestep $t$:

$$\frac{\partial J_2}{\partial W} = \frac{\partial J_2}{\partial y_2} \frac{\partial y_2}{\partial s_2} \frac{\partial s_2}{\partial W}$$
let’s try it out for $W$ with the chain rule:

$$\frac{\partial J}{\partial W} = \sum_t \frac{\partial J_t}{\partial W}$$

so let’s take a single timestep $t$:

$$\frac{\partial J_2}{\partial W} = \frac{\partial J_2}{\partial y_2} \frac{\partial y_2}{\partial s_2} \frac{\partial s_2}{\partial W}$$

but wait...
let’s try it out for $W$ with the **chain rule**:

\[
\frac{\partial J}{\partial W} = \sum_t \frac{\partial J_t}{\partial W}
\]

so let’s take a single timestep $t$:

\[
\frac{\partial J_2}{\partial W} = \frac{\partial J_2}{\partial y_2} \frac{\partial y_2}{\partial s_2} \frac{\partial s_2}{\partial W}
\]

but wait...

\[
s_2 = \tanh(Us_1 + Wx_2)
\]
let’s try it out for $W$ with the **chain rule**:

\[
\frac{\partial J}{\partial W} = \sum_t \frac{\partial J_t}{\partial W}
\]

so let’s take a single timestep $t$:

\[
\frac{\partial J_2}{\partial W} = \frac{\partial J_2}{\partial y_2} \frac{\partial y_2}{\partial s_2} \frac{\partial s_2}{\partial W}
\]

but wait…

\[s_2 = \tanh(U s_1 + W x_2)\]

$s_1$ also depends on $W$ so we can’t just treat $\frac{\partial s_2}{\partial W}$ as a constant!
how does $s_2$ depend on $W$?
how does $s_2$ depend on $W$?
how does $s_2$ depend on $W$?

\[
\frac{\partial s_2}{\partial W} + \frac{\partial s_2}{\partial s_1} \frac{\partial s_1}{\partial W}
\]
how does $s_2$ depend on $W$?

$$\begin{align*}
\frac{\partial s_2}{\partial W} &+ \frac{\partial s_2}{\partial s_1} \frac{\partial s_1}{\partial W} \\
&+ \frac{\partial s_2}{\partial s_0} \frac{\partial s_0}{\partial W}
\end{align*}$$
backpropagation through time:

\[
\frac{\partial J_2}{\partial W} = \sum_{k=0}^{2} \frac{\partial J_2}{\partial y_2} \frac{\partial y_2}{\partial s_2} \frac{\partial s_2}{\partial s_k} \frac{\partial s_k}{\partial W}
\]

Contributions of \( W \) in previous timesteps to the error at timestep \( t \)
backpropagation through time:

\[
\frac{\partial J_t}{\partial W} = \sum_{k=0}^{t} \frac{\partial J_t}{\partial y_t} \frac{\partial y_t}{\partial s_t} \frac{\partial s_t}{\partial s_k} \frac{\partial s_k}{\partial W}
\]

Contributions of $W$ in previous timesteps to the error at timestep $t$
why are RNNs hard to train?
problem: vanishing gradient

\[ \frac{\partial J_2}{\partial W} = \sum_{k=0}^{2} \frac{\partial J_2}{\partial y_2} \frac{\partial y_2}{\partial s_2} \frac{\partial s_2}{\partial s_k} \frac{\partial s_k}{\partial W} \]
problem: vanishing gradient

\[
\frac{\partial J_2}{\partial W} = \sum_{k=0}^{2} \frac{\partial J_2}{\partial y_2} \frac{\partial y_2}{\partial s_2} \frac{\partial s_2}{\partial s_k} \frac{\partial s_k}{\partial W}
\]
problem: vanishing gradient

\[
\frac{\partial J_2}{\partial W} = \sum_{k=0}^{2} \frac{\partial J_2}{\partial y_2} \frac{\partial y_2}{\partial s_2} \frac{\partial s_2}{\partial s_k} \frac{\partial s_k}{\partial W}
\]

at \( k = 0 \):

\[
\frac{\partial s_2}{\partial s_0} = \frac{\partial s_2}{\partial s_1} \frac{\partial s_1}{\partial s_0}
\]
problem: vanishing gradient

\[
\frac{\partial J_n}{\partial W} = \sum_{k=0}^{n} \frac{\partial J_n}{\partial y_n} \frac{\partial y_n}{\partial s_n} \frac{\partial s_n}{\partial s_k} \frac{\partial s_k}{\partial W}
\]

\[
\frac{\partial s_n}{\partial s_{n-1}} \frac{\partial s_{n-1}}{\partial s_{n-2}} \cdots \frac{\partial s_3}{\partial s_2} \frac{\partial s_2}{\partial s_1} \frac{\partial s_1}{\partial s_0}
\]

\[
\begin{align*}
y_0 & \quad s_0 & \quad x_0 \\
y_1 & \quad s_1 & \quad x_1 \\
y_2 & \quad s_2 & \quad x_2 \\
y_3 & \quad s_3 & \quad x_3 \\
\vdots & \quad \vdots & \quad \vdots \\
y_n & \quad s_n & \quad x_n
\end{align*}
\]
**problem: vanishing gradient**

\[
\frac{\partial J_n}{\partial W} = \sum_{k=0}^{n} \frac{\partial J_n}{\partial y_n} \frac{\partial y_n}{\partial s_n} \frac{\partial s_n}{\partial s_k} \frac{\partial s_k}{\partial W}
\]

as the gap between timesteps gets bigger, this product gets longer and longer!
problem: vanishing gradient

\[
\frac{\partial s_n}{\partial s_{n-1}} \frac{\partial s_{n-1}}{\partial s_{n-2}} \cdots \frac{\partial s_3}{\partial s_2} \frac{\partial s_2}{\partial s_1} \frac{\partial s_1}{\partial s_0}
\]
problem: vanishing gradient

what are each of these terms?

$$\frac{\partial s_n}{\partial s_{n-1}} \frac{\partial s_{n-1}}{\partial s_{n-2}} \cdots \frac{\partial s_3}{\partial s_2} \frac{\partial s_2}{\partial s_1} \frac{\partial s_1}{\partial s_0}$$
**Problem:** vanishing gradient

What are each of these terms?

\[
\frac{\partial s_n}{\partial s_{n-1}} = W^T \text{diag}\left[f'(W_{s_{n-1}} + Ux_j)\right]
\]

\[W = \text{sampled from standard normal distribution} = \text{mostly} < 1\]

\[f = \text{tanh or sigmoid so } f' < 1\]
problem: vanishing gradient

what are each of these terms?

\[
\frac{\partial s_n}{\partial s_{n-1}} = W^T \text{diag}[f'(W_{s_{n-1}} + Ux_j)]
\]

\(W = \text{sampled from standard normal distribution} = \text{mostly } < 1\)

\(f = \text{tanh or sigmoid so } f' < 1\)

we’re multiplying a lot of small numbers together.
we’re multiplying a lot of **small numbers** together.

so what?

errors due to further back timesteps have increasingly smaller gradients.

so what?

parameters become biased to **capture shorter-term dependencies**.
“In France, I had a great time and I learnt some of the ______ language.”

our parameters are not trained to capture long-term dependencies, so the word we predict will mostly depend on the previous few words, not much earlier ones
solution #1: activation functions

ReLU derivative prevents $f'$ from shrinking the gradients

tanh derivative

sigmoid derivative
solution #2: initialization

weights initialized to identity matrix
biases initialized to zeros

$W = \begin{pmatrix}
1 & 0 & 0 & \cdots & 0 \\
0 & 1 & 0 & \cdots & 0 \\
0 & 0 & 1 & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & \cdots & 1 
\end{pmatrix}$

prevents $W$ from shrinking the gradients
solution #3: **gated cells**

rather each node being just a simple RNN cell, make each node a more **complex unit with gates** controlling what information is passed through.
solution #3: more on **LSTMs**
solution #3: more on **LSTMs**

forget irrelevant parts of previous state
solution #3: more on **LSTMs**

$s_j \quad \rightarrow \quad s_{j+1}$

selectively update cell state values
solution #3: more on LSTMs

\[ S_j \rightarrow S_{j+1} \]

output certain parts of cell state
solution #3: more on **LSTMs**

Forget irrelevant parts of previous state

Selectively update cell state values

Output certain parts of cell state

$S_j \rightarrow S_{j+1}$
why do LSTMs help?

1. forget gate allows information to pass through unchanged
   → when taking the derivative, \( f' \) is 1 for what we want to keep!

2. \( s_j \) depends on \( s_{j-1} \) through addition!
   → when taking the derivative, not lots of small \( W \) terms!
in practice: machine translation.
basic encoder-decoder model:
add LSTM cells:

MIT 6.S191 | Intro to Deep Learning | IAP 2017
problem: a fixed-length encoding is limiting

all the decoder knows about the input sentence is in one fixed length vector, $s_2$
solution: attend over all encoder states
solution: attend over all encoder states
solution: attend over all encoder states
now we can model **sequences**!

- why recurrent neural networks?
- building models for language, classification, and machine translation
- training them with backpropagation through time
- solving the vanishing gradient problem with activation functions, initialization, and gated cells (like LSTMs)
- using attention mechanisms
and there’s lots more to do!

- extending our models to timeseries + waveforms
- complex language models to generate long text or books
- language models to generate code
- controlling cars + robots
- predicting stock market trends
- summarizing books + articles
- handwriting generation
- multilingual translation models
- … many more!