Deep Sequence Modeling

Ava Soleimany

MIT 6.S191

January 27, 2020
Given an image of a ball, can you predict where it will go next?
Given an image of a ball, can you predict where it will go next?
Given an image of a ball, can you predict where it will go next?
Given an image of a ball, can you predict where it will go next?
Sequences in the Wild

Audio
Sequences in the Wild

character:

word:

6.S.191 Introduction to Deep Learning

Text
A Sequence Modeling Problem: Predict the Next Word
A Sequence Modeling Problem: Predict the Next Word

“This morning I took my cat for a walk.”
A Sequence Modeling Problem: Predict the Next Word

“This morning I took my cat for a walk.”

given these words
A Sequence Modeling Problem: Predict the Next Word

“This morning I took my cat for a walk.”

given these words

predict the next word
Idea #1: Use a Fixed Window

“This morning I took my cat for a walk.”

given these two words predict the next word
Idea #1: Use a Fixed Window

“This morning I took my cat \textcolor{green}{\text{for a walk}}.”

Given these two words, predict the next word.

One-hot feature encoding: tells us what each word is

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

\begin{align*}
\text{for} & \quad \downarrow & \text{a} \\
\text{prediction} & &
\end{align*}
Problem #1: Can’t Model Long-Term Dependencies

“France is where I grew up, but I now live in Boston. I speak fluent ____.”

\[ J’aime\ 6.191! \]

We need information from the distant past to accurately predict the correct word.
Idea #2: Use Entire Sequence as Set of Counts

“This morning I took my cat for a”

“bag of words”

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

prediction
Problem #2: Counts Don’t Preserve Order

The food was good, not bad at all.

vs.

The food was bad, not good at all.
Idea #3: Use a Really Big Fixed Window

“This morning I took my cat for a walk.”

given these words predict the next word

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

morning I took this cat

prediction
Problem #3: No Parameter Sharing

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

this \hspace{0.5cm} \textcolor{red}{\text{morning}} \hspace{0.5cm} \textcolor{red}{\text{took}} \hspace{0.5cm} \textcolor{red}{\text{the}} \hspace{0.5cm} \textcolor{red}{\text{cat}}

Each of these inputs has a separate parameter:
Problem #3: No Parameter Sharing

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

this \hspace{1cm} \text{morning} \hspace{1cm} \text{took} \hspace{1cm} \text{the} \hspace{1cm} \text{cat}

Each of these inputs has a \text{separate parameter}:

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

this \hspace{1cm} \text{morning}
Problem #3: No Parameter Sharing

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

this  morning  took  the  cat

Each of these inputs has a separate parameter:

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

this  morning

Things we learn about the sequence won’t transfer if they appear elsewhere in the sequence.
Sequence Modeling: Design Criteria

To model sequences, we need to:

1. Handle **variable-length** sequences
2. Track **long-term** dependencies
3. Maintain information about **order**
4. Share **parameters** across the sequence

Today: **Recurrent Neural Networks (RNNs)** as an approach to sequence modeling problems
Recurrent Neural Networks (RNNs)
Standard Feed-Forward Neural Network

\[ \hat{y} \]
\[ \downarrow \]
\[ x \]

One to One
“Vanilla” neural network
Recurrent Neural Networks for Sequence Modeling

One to One
"Vanilla" neural network

Many to One
Sentiment Classification
Recurrent Neural Networks for Sequence Modeling

One to One
"Vanilla" neural network

Many to One
Sentiment Classification

Many to Many
Music Generation

6.191 Lab!
Recurrent Neural Networks for Sequence Modeling

One to One
“Vanilla” neural network

Many to One
Sentiment Classification

Many to Many
Music Generation

... and many other architectures and applications

6.191 Lab!
Standard “Vanilla” Neural Network

\[ \hat{y}_t \]

output vector

\[ x_t \]

input vector
Recurrent Neural Network (RNN)

output vector $\hat{y}_t$

RNN

$h_t$

input vector $x_t$
Recurrent Neural Network (RNN)
Recurrent Neural Network (RNN)

Apply a recurrence relation at every time step to process a sequence:

\[
\hat{y}_t, h_t, x_t
\]
Recurrent Neural Network (RNN)

Apply a **recurrence relation** at every time step to process a sequence:

\[
\hat{y}_t = f_W(h_t, x_t)
\]

- \( h_t \): cell state parameterized by \( W \)
- \( h_{t-1} \): old state
- \( x_t \): input vector at time step \( t \)
Recurrent Neural Network (RNN)

Apply a recurrence relation at every time step to process a sequence:

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

- cell state
- function parameterized by \( W \)
- old state
- input vector at time step \( t \)

Note: the same function and set of parameters are used at every time step.
RNN Intuition

```
my_rnn = RNN()
hidden_state = [0, 0, 0, 0]

sentence = ["I", "love", "recurrent", "neural"]

for word in sentence:
    prediction, hidden_state = my_rnn(word, hidden_state)

next_word_prediction = prediction
# >>> "networks!"
```
RNN Intuition

```
my_rnn = RNN()
hidden_state = [0, 0, 0, 0]

sentence = ["I", "love", "recurrent", "neural"]

for word in sentence:
    prediction, hidden_state = my_rnn(word, hidden_state)

next_word_prediction = prediction
# >>> "networks!"
```
my_rnn = RNN()
hidden_state = [0, 0, 0, 0]

sentence = ['I', 'love', 'recurrent', 'neural']

for word in sentence:
    prediction, hidden_state = my_rnn(word, hidden_state)

next_word_prediction = prediction
# >>> "networks!"
RNN State Update and Output

\[ \hat{y}_t \]

\[ h_t \]

\[ x_t \]
RNN State Update and Output

\[ \hat{y}_t \]

\[ h_t \]

Input Vector \( x_t \)

output vector

RNN recurrent cell
RNN State Update and Output

\[
\hat{y}_t = \text{RNN} \left( x_t, h_t \right)
\]

output vector $\hat{y}_t$

input vector $x_t$

Update Hidden State

\[
h_t = \tanh \left( W_{hh}^T h_{t-1} + W_{xh}^T x_t \right)
\]

Input Vector

$x_t$
RNN State Update and Output

Output Vector
\[ \hat{y}_t = W_{hy}^T h_t \]

Update Hidden State

\[ h_t = \tanh(W_{hh}^T h_{t-1} + W_{xh}^T x_t) \]

Input Vector
\[ x_t \]
RNNs: Computational Graph Across Time

\[ \hat{y}_t \]

\[ x_t \]

\[ \text{RNN} \]

Represent as computational graph unrolled across time
RNNs: Computational Graph Across Time

\[ \hat{y}_t \rightarrow \text{RNN} \rightarrow \hat{y}_0 \]

\[ x_t \rightarrow \text{RNN} \rightarrow x_0 \]
RNNs: Computational Graph Across Time
RNNs: Computational Graph Across Time
RNNs: Computational Graph Across Time

\[ \hat{y}_t = W_{xh} x_t \]

\[ \hat{y}_0 = W_{xh} x_0 \]

\[ \hat{y}_1 = W_{xh} x_1 \]

\[ \hat{y}_2 = W_{xh} x_2 \]

\[ \ldots \]

\[ \hat{y}_t = W_{xh} x_t \]
RNNs: Computational Graph Across Time
RNNs: Computational Graph Across Time
RNNs: Computational Graph Across Time

Re-use the same weight matrices at every time step
RNNs: Computational Graph Across Time

→ Forward pass

\[
\begin{align*}
\hat{y}_t & \quad \hat{y}_0 \\
W_{hy} & \quad W_{hy} \\
RNN & \quad L_0 \\
W_{xh} & \quad W_{xh} \\
x_t & \quad x_0 \\
\end{align*}
\]

\[
\begin{align*}
\hat{y}_1 & \quad \hat{y}_1 \\
W_{hy} & \quad W_{hy} \\
RNN & \quad L_1 \\
W_{xh} & \quad W_{xh} \\
x_1 & \quad x_1 \\
\end{align*}
\]

\[
\begin{align*}
\hat{y}_2 & \quad \hat{y}_2 \\
W_{hy} & \quad W_{hy} \\
RNN & \quad L_2 \\
W_{xh} & \quad W_{xh} \\
x_2 & \quad x_2 \\
\end{align*}
\]

\[
\begin{align*}
\ldots & \quad \ldots \\
W_{hy} & \quad W_{hy} \\
RNN & \quad L_3 \\
W_{xh} & \quad W_{xh} \\
x_t & \quad x_t \\
\end{align*}
\]
RNNs: Computational Graph Across Time

Forward pass

\[ \hat{y}_t \]

\[ \hat{y}_0 \]

\[ \hat{y}_1 \]

\[ \hat{y}_2 \]

\[ \cdots \]

\[ \hat{y}_t \]

\[ L_0 \]

\[ L_1 \]

\[ L_2 \]

\[ L_3 \]

\[ x_t \]

\[ x_0 \]

\[ x_1 \]

\[ x_2 \]

\[ \cdots \]

\[ x_t \]

\[ W_{hy} \]

\[ W_{hx} \]

\[ W_{hh} \]
class MyRNNCell(tf.keras.layers.Layer):
    def __init__(self, rnn_units, input_dim, output_dim):
        super(MyRNNCell, self).__init__()

        # Initialize weight matrices
        self.W_xh = self.add_weight([rnn_units, input_dim])
        self.W_hh = self.add_weight([rnn_units, rnn_units])
        self.W_hy = self.add_weight([output_dim, rnn_units])

        # Initialize hidden state to zeros
        self.h = tf.zeros([rnn_units, 1])

    def call(self, x):
        # Update the hidden state
        self.h = tf.math.tanh( self.W_hh * self.h + self.W_xh * x )

        # Compute the output
        output = self.W_hy * self.h

        # Return the current output and hidden state
        return output, self.h
RNN Implementation in TensorFlow

tf.keras.layers.SimpleRNN(rnn_units)
Backpropagation Through Time (BPTT)
Recall: Backpropagation in Feed Forward Models

Backpropagation algorithm:

1. Take the derivative (gradient) of the loss with respect to each parameter
2. Shift parameters in order to minimize loss
RNNs: Backpropagation Through Time

Forward pass

\[
\begin{align*}
\hat{y}_t &\rightarrow W_{hy} \\
\hat{y}_0 &\rightarrow W_{hy} \\
\hat{y}_1 &\rightarrow W_{hy} \\
\hat{y}_2 &\rightarrow W_{hy} \\
\vdots &\rightarrow W_{hy} \\
\hat{y}_t &\rightarrow W_{hy} \\
RNN &\rightarrow W_{xh} \\
\cdots
\end{align*}
\]
RNNs: Backpropagation Through Time
Standard RNN Gradient Flow
Computing the gradient wrt $h_0$ involves many factors of $W_{hh}$ + repeated gradient computation!
Standard RNN Gradient Flow: Exploding Gradients

Computing the gradient wrt $h_0$ involves many factors of $W_{hh}$ + repeated gradient computation!

Many values > 1: exploding gradients
Gradient clipping to scale big gradients
Standard RNN Gradient Flow: Vanishing Gradients

Computing the gradient wrt $h_0$ involves many factors of $W_{hh}$ + repeated gradient computation!

Many values > 1: exploding gradients
Gradient clipping to scale big gradients

Many values < 1: vanishing gradients
1. Activation function
2. Weight initialization
3. Network architecture
The Problem of Long-Term Dependencies

Why are vanishing gradients a problem?
The Problem of Long-Term Dependencies

Why are vanishing gradients a problem?

Multiply many small numbers together
The Problem of Long-Term Dependencies

Why are vanishing gradients a problem?

Multiply many small numbers together

Errors due to further back time steps have smaller and smaller gradients
The Problem of Long-Term Dependencies

Why are vanishing gradients a problem?

Multiply many small numbers together

Errors due to further back time steps
have smaller and smaller gradients

Bias parameters to capture short-term dependencies
The Problem of Long-Term Dependencies

Why are vanishing gradients a problem?

Multiply many small numbers together

Errors due to further back time steps have smaller and smaller gradients

Bias parameters to capture short-term dependencies

“The clouds are in the ___”
The Problem of Long-Term Dependencies

Why are vanishing gradients a problem?

Multiply many small numbers together

Errors due to further back time steps have smaller and smaller gradients

Bias parameters to capture short-term dependencies
The Problem of Long-Term Dependencies

Why are vanishing gradients a problem?

Multiply many small numbers together

Errors due to further back time steps have smaller and smaller gradients

Bias parameters to capture short-term dependencies

“The clouds are in the ___”

“\( \hat{y}_0 \) \rightarrow \hat{y}_1 \rightarrow \hat{y}_2 \rightarrow \hat{y}_3 \rightarrow \hat{y}_4 \)"

“I grew up in France, … and I speak fluent ___”
The Problem of Long-Term Dependencies

Why are vanishing gradients a problem?

Multiply many small numbers together

Errors due to further back time steps have smaller and smaller gradients

Bias parameters to capture short-term dependencies

“The clouds are in the ___”

“I grew up in France, … and I speak fluent ___”
Trick #1: Activation Functions

Using ReLU prevents $f'$ from shrinking the gradients when $x > 0$.
Trick #2: Parameter Initialization

Initialize weights to identity matrix

\[ I_n = \begin{pmatrix} 1 & 0 & 0 & \ldots & 0 \\ 0 & 1 & 0 & \ldots & 0 \\ 0 & 0 & 1 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & 1 \end{pmatrix} \]

Initialize biases to zero

This helps prevent the weights from shrinking to zero.
Solution #3: Gated Cells

Idea: use a more complex recurrent unit with gates to control what information is passed through

Long Short Term Memory (LSTMs) networks rely on a gated cell to track information throughout many time steps.
Long Short Term Memory (LSTM) Networks
Standard RNN

In a standard RNN, repeating modules contain a simple computation node
Long Short Term Memory (LSTMs)

LSTM modules contain computational blocks that control information flow.

LSTM cells are able to track information throughout many timesteps.

```
tf.keras.layers.LSTM(num_units)
```
Long Short Term Memory (LSTMs)

Information is **added** or **removed** through structures called **gates**

Gates optionally let information through, for example via a sigmoid neural net layer and pointwise multiplication
Long Short Term Memory (LSTMs)

How do LSTMs work?

1) Forget  2) Store  3) Update  4) Output
Long Short Term Memory (LSTMs)

1) Forget  
2) Store  
3) Update  
4) Output

LSTMs forget irrelevant parts of the previous state
Long Short Term Memory (LSTMs)

1) Forget  2) Store  3) Update  4) Output

LSTMs store relevant new information into the cell state
Long Short Term Memory (LSTMs)

1) Forget  2) Store  3) Update  4) Output

LSTMs selectively update cell state values
Long Short Term Memory (LSTMs)

1) Forget  2) Store  3) Update  4) Output

The **output gate** controls what information is sent to the next time step.
Long Short Term Memory (LSTMs)

1) Forget  2) Store  3) Update  4) Output

\[ \begin{align*}
    c_{t-1} & \rightarrow f_t \rightarrow \times \rightarrow c_t \\
    h_{t-1} & \rightarrow \sigma \rightarrow \times \rightarrow \sigma \\
    x_t & \rightarrow \sigma \rightarrow \times \rightarrow \sigma \\
    y_t & \rightarrow c_t \\
    h_t & \rightarrow \tanh \\
\end{align*} \]
LSTM Gradient Flow

Uninterrupted gradient flow!
LSTMs: Key Concepts

1. Maintain a separate cell state from what is outputted
2. Use gates to control the flow of information
   - Forget gate gets rid of irrelevant information
   - Store relevant information from current input
   - Selectively update cell state
   - Output gate returns a filtered version of the cell state
3. Backpropagation through time with uninterrupted gradient flow
RNN Applications
Example Task: Music Generation

- **Input:** sheet music
- **Output:** next character in sheet music
Example Task: Sentiment Classification

Input: sequence of words
Output: probability of having positive sentiment

\[ \text{loss} = \text{tf.nn.softmax_cross_entropy_with_logits}(y, \text{predicted}) \]
Example Task: Sentiment Classification

Tweet sentiment classification

Ivar Hagendoorn
@IvarHagendoorn

The @MIT Introduction to #DeepLearning is definitely one of the best courses of its kind currently available online introtodeeplearning.com
12:45 PM - 12 Feb 2018

Angels-Cave
@AngelsCave

Replying to @Kazuki2048

I wouldn’t mind a bit of snow right now. We haven’t had any in my bit of the Midlands this winter! :(
2:19 AM - 25 Jan 2019
Example Task: Machine Translation

Encoder (English)

Decoder (French)

the dog eats

<start> le chien mange
Example Task: Machine Translation

Encoder (English) — encoding bottleneck — Decoder (French)

the → dog → eats → <start> → le → chien → mange
Attention mechanisms in neural networks provide **learnable memory access**

Encoder (English) → Decoder (French)

*the* → *dog* → *eats* → "<start>" → *le* → *chien* → *mange*
Trajectory Prediction: Self-Driving Cars
Environmental Modeling
Deep Learning for Sequence Modeling: Summary

1. RNNs are well suited for sequence modeling tasks
2. Model sequences via a recurrence relation
3. Training RNNs with backpropagation through time
4. Gated cells like LSTMs let us model long-term dependencies
5. Models for music generation, classification, machine translation, and more
6.S191: Introduction to Deep Learning

Lab 1: Introduction to TensorFlow and Music Generation with RNNs

Link to download labs:
http://introtodeeplearning.com#schedule

1. Open the lab in Google Colab
2. Start executing code blocks and filling in the #TODOs
3. Need help? Find a TA or come to the front!!