Deep Sequence Modeling

Ava Soleimany

MIT 6.S191

January 19, 2021
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
Sequence Modeling Applications

One to One
Binary Classification

Many to One
Sentiment Classification

One to Many
Image Captioning

Many to Many
Machine Translation

"Will I pass this class?"
Student → Pass?

"A baseball player throws a ball."
Neurons with Recurrence
The Perceptron Revisited

\[ y = g(z) \]

\[ z = w_1 x^{(1)} + w_2 x^{(2)} + \ldots + w_m x^{(m)} \]
Feed-Forward Networks Revisited

\[ x^{(1)} \rightarrow x^{(2)} \rightarrow x^{(m)} \rightarrow \hat{y}^{(1)} \rightarrow \hat{y}^{(2)} \rightarrow \hat{y}^{(3)} \rightarrow \hat{y}^{(n)} \]

\[ x \in \mathbb{R}^m \quad \hat{y} \in \mathbb{R}^n \]
Feed-Forward Networks Revisited

\[ x_t \in \mathbb{R}^m \quad \hat{y}_t \in \mathbb{R}^n \]
Handling Individual Time Steps

\[ \hat{y}_t = f(x_t) \]
Neurons with Recurrence

\[ \hat{y}_t = f(x_t, h_{t-1}) \]

output vector
\[ \hat{y}_t \]

input vector
\[ x_t \]

\[ \hat{y}_0 \]
\[ x_0 \]
\[ h_0 \]
\[ \hat{y}_1 \]
\[ x_1 \]
\[ h_1 \]
\[ \hat{y}_2 \]
\[ x_2 \]
Neurons with Recurrence

\[ \hat{y}_t = f(x_t, h_{t-1}) \]
Recurrent Neural Networks (RNNs)
Recurrent Neural Networks (RNNs)

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

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

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

RNNs have a state, \( h_t \), that is updated at each time step as a sequence is processed.
RNN Intuition

```python
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

```python
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

```python
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

output vector $\hat{y}_t$

RNN

$h_t$

input vector $x_t$
RNN State Update and Output

\[ \hat{y}_t \]

\[ \text{RNN} \]

\[ h_t \]

\[ x_t \]

\[ \text{Input Vector} \]

\[ x_t \]
RNN State Update and Output

Update Hidden State

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

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

\[ \hat{y}_t \]
\[ \text{RNN} \]
\[ x_t \]

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

Re-use the same weight matrices at every time step

\[
\hat{y}_t, \hat{y}_0, \hat{y}_1, \hat{y}_2, \ldots
\]

\[
RNN = \begin{align*}
&L_0 \\
&L_1 \\
&L_2 \\
&L_3
\end{align*}
\]

\[
\begin{align*}
&x_t \\
&x_0 \\
&x_1 \\
&x_2 \\
&\ldots
\end{align*}
\]

\[
\begin{align*}
&W_{hx} \\
&W_{hh} \\
&W_{hy} \\
&W_{hx}
\end{align*}
\]
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

\[
\text{tf\.keras\.layers\.SimpleRNN}(\text{rnn_units})
\]
RNNs for Sequence Modeling

\[ \hat{y} \]

\[ x \]

One to One
"Vanilla" NN
Binary classification

Many to One
Sentiment Classification

One to Many
Text Generation
Image Captioning

Many to Many
Translation & Forecasting
Music Generation

... and many other architectures and applications

6.5191 Lab!
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

Recurrent Neural Networks (RNNs) meet these sequence modeling design criteria
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
A Sequence Modeling Problem: Predict the Next Word

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

given these words
predict the next word

Representing Language to a Neural Network

Neural networks cannot interpret words
Neural networks require numerical inputs
Encoding Language for a Neural Network

Neural networks cannot interpret words

Neural networks require numerical inputs

Embedding: transform indexes into a vector of fixed size.

1. Vocabulary: Corpus of words
2. Indexing: Word to index
3. Embedding: Index to fixed-sized vector

One-hot embedding

```
"cat" = [0, 1, 0, 0, 0, 0]
```

Learned embedding

```
run
walk

day
sun

dog
happy
sad
```
Handle Variable Sequence Lengths

The food was great

vs.

We visited a restaurant for lunch

vs.

We were hungry but cleaned the house before eating
Model Long-Term Dependencies

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

 retorno. J’aime 6.5191!

We need information from the distant past to accurately predict the correct word.
Capture Differences in Sequence Order

The food was good, not bad at all.

vs.

The food was bad, not good at all.
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

**Recurrent Neural Networks (RNNs)** meet these sequence modeling design criteria
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
RNNs: Backpropagation Through Time

\[
\begin{align*}
\hat{y}_t & \leftarrow W_{hy} \hat{y}_0 \\
\hat{y}_0 & \leftarrow W_{hx} x_0 \\
\hat{y}_1 & \leftarrow W_{hy} \hat{y}_1 \\
\hat{y}_2 & \leftarrow W_{hy} \hat{y}_2 \\
\cdots & \\
\hat{y}_t & \leftarrow W_{hy} \hat{y}_t
\end{align*}
\]

L

\[
\begin{align*}
x_t & \rightarrow W_{hx} x_t \\
x_0 & \rightarrow W_{hx} x_0 \\
x_1 & \rightarrow W_{hx} x_1 \\
x_2 & \rightarrow W_{hx} x_2 \\
\cdots & \\
x_t & \rightarrow W_{hx} x_t
\end{align*}
\]
Standard RNN Gradient Flow

\[ h_0 \xrightarrow{w_{hh}} h_t \]

\[ w_{xh} \]

\[ x_0 \xrightarrow{w_{xh}} x_1 \xrightarrow{w_{xh}} x_2 \xrightarrow{\ldots} x_t \xrightarrow{w_{xh}} h_t \]
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 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 clouds are in the___”

“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 $x > 0$. 

- ReLU derivative
- tanh derivative
- sigmoid derivative
Trick #2: Parameter Initialization

Initialize weights to identity matrix

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

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.

```python
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
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

Listening to 3rd movement

6.S191 Lab!
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
Example Task: Machine Translation

Potential Issues

Encoding bottleneck

Encoder (English)  Decoder (French)

the  dog  eats  le  chien  mange
Example Task: Machine Translation

Potential Issues

- 🔄 Encoding bottleneck
- 🕒 Slow, no parallelization

Encoder (English) → Decoder (French)

- the dog eats
- le chien mange
Example Task: Machine Translation

Potential Issues

- 🌊 Encoding bottleneck
- ⏳ Slow, no parallelization
- 🧠 Not long memory

Encoder (English)
- the
dog
eats

Decoder (French)
- le
chien
mange
- le
chien

The diagram illustrates the process of machine translation, showing the flow from the English encoder to the French decoder, with potential issues highlighted.
Example Task: Machine Translation

Attention mechanisms in neural networks provide **learnable memory access**

```
Encoder (English)      Decoder (French)
```

- the
- dog
- eats

```
Attention
```

```
le  chien  mange
```

Sutskever+, NeurIPS 2014; Bahdanau+ ICLR 2015; Vaswani+, NeurIPS 2017
Application: Trajectory Prediction for Self-Driving Cars
Application: 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? Come to the class Gather.Town!