DO NOT DISTURB....
MACHINE IS LEARNING.
Lecture 13: Recurrent Neural Networks

CS109B Data Science 2
Pavlos Protopapas, Mark Glickman, and Chris Tanner
Outline

Why Recurrent Neural Networks (RNNs)
Main Concept of RNNs
More Details of RNNs
RNN training
Gated RNN
Many classification and regression tasks involve data that is assumed to be independent and identically distributed (i.i.d.). For example:

- Detecting lung cancer
- Face recognition
- Risk of heart attack
Background

Much of our data is inherently **sequential**

<table>
<thead>
<tr>
<th>scale</th>
<th>examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORLD</td>
<td>Natural disasters (e.g., earthquakes)</td>
</tr>
<tr>
<td></td>
<td>Climate change</td>
</tr>
<tr>
<td>HUMANITY</td>
<td>Stock market</td>
</tr>
<tr>
<td></td>
<td>Flu outbreaks</td>
</tr>
<tr>
<td>INDIVIDUAL PEOPLE</td>
<td>Speech recognition</td>
</tr>
<tr>
<td></td>
<td>Machine Translation (e.g., English -&gt; French)</td>
</tr>
</tbody>
</table>
Background

Much of our data is inherently sequential

PREDICTING EARTHQUAKES
Background

Much of our data is inherently sequential

STOCK MARKET PREDICTIONS
Background

Much of our data is inherently sequential

**SPEECH RECOGNITION**

“What is the weather today?”

“What is the weather two day?”

“What is the whether too day?”

“What is, the Wrether to Dae?”
Sequence Modeling: Handwritten Text

- Input: Image
- Output: Text

https://towardsdatascience.com/build-a-handwritten-text-recognition-system-using-tensorflow-2326a3487c3d5
Sequence Modeling: Text-to-Speech

- Input: Audio
- Output: Text
Sequence Modeling: Machine Translation

- Input: Text
- Output: Translated Text

Economic growth has slowed down in recent years.

Das Wirtschaftswachstum hat sich in den letzten Jahren verlangsamt.

La croissance économique s’est ralentie ces dernières années.
Outline

Why RNNs
Main Concept of RNNs (part 1)
More Details of RNNs
RNN training
Gated RNN
What can my NN do?

**Training:** Present to the NN examples and learn from them.

[George, Mary, Tom, Suzie]
What can my NN do?

**Prediction:** Given an example

- **George**
- **Mary**
What my NN can NOT do?

WHO IS IT?

?
Learn from previous examples
Recurrent Neural Network (RNN)
Recurrent Neural Network (RNN)

I have seen George moving in this way before.

RNNs recognize the data's sequential characteristics and use patterns to predict the next likely scenario.
Recurrent Neural Network (RNN)

Our model requires context - or contextual information - to understand the subject (he) and the direct object (it) in the sentence.

"He told me I could have it"

WHO IS HE?

I do not know. I need to know who said that and what he said before. Can you tell me more?
After providing sequential information, the model understood the subject (Joe’s brother) and the direct object (sweater) in the sentence.

- Hellen: Nice sweater Joe.
- Joe: Thanks, Hellen. It used to belong to my brother and he told me I could have it.

I see what you mean now!

The noun “he” stands for Joe’s brother while “it” for the sweater.
Sequences

• We want a machine learning model to understand sequences, not isolated samples.
• Can MLP do this?
• Assume we have a sequence of temperature measurements and we want to take 3 sequential measurements and predict the next one.

<table>
<thead>
<tr>
<th>samples</th>
<th>features</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>35</td>
</tr>
<tr>
<td>2</td>
<td>32</td>
</tr>
<tr>
<td>3</td>
<td>45</td>
</tr>
<tr>
<td>4</td>
<td>48</td>
</tr>
<tr>
<td>5</td>
<td>41</td>
</tr>
<tr>
<td>6</td>
<td>39</td>
</tr>
<tr>
<td>7</td>
<td>36</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Sequences

- We want a machine learning model to understand sequences, not isolated samples.
- Can MLP do this?
- Assume we have a sequence of temperature measurements and we want to take 3 sequential measurements and predict the next one.
Sequences

- We want a machine learning model to understand sequences, not isolated samples.
- Can MLP do this?
- Assume we have a sequence of temperature measurements and we want to take 3 sequential measurements and predict the next one.
Sequences

- We want a machine learning model to understand sequences, not isolated samples.
- Can MLP do this?
- Assume we have a sequence of temperature measurements and we want to take 3 sequential measurements and predict the next one.
Sequences

- We want a machine learning model to understand sequences, not isolated samples.
- Can MLP do this?
- Assume we have a sequence of temperature measurements and we want to take 3 sequential measurements and predict the next one.
Windowed dataset

This is called **overlapping windowed** dataset, since we’re windowing observations to create new.

We can easily do using a MLS:

![Diagram showing overlapping windowed dataset]

But re-arranging the order of the inputs like:

```
3  45
1  35
2  32
4

2  32
3  45
5

4  48
5  41
6
```

will produce the same results.
Why not CNNs or MLPs?

1. MLPs/CNNs require fixed input and output size
2. MLPs/CNNs can’t classify inputs in multiple places
Windowed dataset

What follows after: ‘I got in the car and’?

drove away

What follows after: ‘In car the and I’?

Not obvious it should be ‘drove away’

The order of words matters. This is true for most sequential data. A fully connected network will not distinguish the order and therefore missing some information.
Outline

Why RNNs

Main Concept of RNNs

More Details of RNNs

RNN training

Gated RNN
Somehow the computational unit should remember what it has seen before.
Memory

Somehow the computational unit should remember what it has seen before.
Somehow the computational unit should remember what it has seen before. We’ll call the information the unit’s state.
Memory

In neural networks, once training is over, the weights do not change. This means that the network is done learning and done changing.

Then, we feed in values, and it simply applies the operations that make up the network, using the values it has learned.

But the RNN units can remember new information after training has completed.

That is, they’re able to keep changing after training is over.
Memory

**Question:** How can we do this? How can build a unit that remembers the past?

The memory or **state** can be written to a file but in RNNs, we keep it inside the recurrent unit.

In an array or in a vector!

**Work with an example:**

*Anna Sofia said her shoes are too ugly. Her here means Anna Sofia.*

*Nikolas put his keys on the table. His here means Nikolas*
**Question:** How can we do this? How can we build a unit that remembers the past?

The memory or **state** can be written to a file but in RNNs, we keep it inside the recurrent unit.

In an array or in a vector!
Building an RNN

\[ Y_t \]

\[ X_t \]

\[ Y_{t+1} \]

\[ X_{t+1} \]

\[ Y_{t+2} \]

\[ X_{t+2} \]

\[ Y_{t+3} \]

\[ X_{t+3} \]
Outline

Why RNNs
Main Concept of RNNs
More Details of RNNs
RNN training
Gated RNN
Structure of an RNN cell
Anatomy of an RNN unit
Outline

Why RNNs
Main Concept of RNNs
More Details of RNNs
**RNN training**
Gated RNN
Backprop Through Time

• For each input, unfold network for the sequence length $T$
• Back-propagation: apply forward and backward pass on unfolded network
• Memory cost: $O(T)$
Backprop Through Time

RNN

$Y_t$

output weights

update weights

$X_t$

input weights

State
Backprop Through Time

\[ Y_t \]

output weights

\[ X_t \]

input weights

update weights

RNN

State
Backprop Through Time

\[ Y_t \]
\[ Y_t \rightarrow \text{Output Weights: } W \rightarrow h_t \rightarrow \text{RNN} \rightarrow \text{State} \rightarrow X_t \rightarrow \text{Input Weights: } V \rightarrow \text{Update Weights: } U \]
You have two activation functions $g_h$ which serves as the activation for the hidden state and $g_y$ which is the activation of the output. In the example shown before $g_y$ was the identity.
Backprop Through Time

\[ \hat{y}_{t-2} \]
\[ w \]
\[ h_{t-2} \]
\[ U \]
\[ v \]
\[ X_{t-2} \]

\[ \hat{y}_{t-1} \]
\[ w \]
\[ h_{t-1} \]
\[ U \]
\[ v \]
\[ X_{t-1} \]

\[ \hat{y}_t \]
\[ w \]
\[ h_t \]
\[ U \]
\[ v \]
\[ X_t \]
Backprop Through Time

\[ \hat{y}_t = g_y(Wh_t + b) \]

\[ L = \sum_t L_t \quad L_t = L_t(\hat{y}_t) \]

\[ \frac{dL}{dW} = \sum_t \frac{dL_t}{dW} = \sum_t \frac{\partial L_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial W} \]

\[ \frac{\partial \hat{y}_t}{\partial W} = g'_y h_t \]
Backprop Through Time

$$\hat{y}_t = g_y(W h_t + b)$$
$$h_t = g_h(V x_t + U h_{t-1} + b')$$
$$\hat{y}_t = g_y(W g_h(V x_t + U h_{t-1} + b') + b)$$

$$L = \sum_t L_t$$
$$L_t = L_t(\hat{y}_t)$$

$$\frac{dL}{dU} = \sum_t \frac{dL_t}{d\hat{y}_t} \frac{d\hat{y}_t}{dh_t} \frac{dh_t}{dU}$$

$$\frac{dh_t}{dU} = \sum_{k=1}^{t} \frac{dh_t}{dh_k} \frac{dh_k}{dU}$$

$$\frac{\partial h_t}{\partial h_k} = \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial h_{t-2}} \ldots \frac{\partial h_{k+1}}{\partial h_k} = \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}}$$

$$\frac{\partial L_t}{dU} = \frac{\partial L_t}{d\hat{y}_t} \frac{d\hat{y}_t}{dh_t} \left( \frac{d h_t}{dU} + \frac{d h_t}{dU} \frac{d h_{t-1}}{dU} + \frac{d h_t}{dU} \frac{d h_{t-1}}{dU} \frac{d h_{t-2}}{dU} + \ldots \right)$$

$$\frac{\partial h_j}{\partial h_{j-1}} = g'_h U$$
Gradient Clipping

Prevents exploding gradients
Clip the norm of gradient before update.
For some derivative $g$, and some threshold $u$

\[
\text{if } ||g|| > u \\
g \leftarrow \frac{gu}{||g||}
\]
Gradient Clipping

Without clipping

With clipping

$J(w, b)$

$w$

$b$
Outline

Why RNNs
Main Concept of RNNs
More Details of RNNs
RNN training
Gated RNN
Long-term Dependencies

Unfolded networks can be very deep.

Long-term interactions are given exponentially smaller weights than small-term interactions.

Gradients tend to either vanish or explode.
Long Short-Term Memory

Handles long-term dependencies
Leaky units where weight on self-loop $\alpha$ is context-dependent
Allow network to decide whether to accumulate or forget past info
Notation

Using conventional and convenient notation
Simple RNN again

\[ z_t = V x_t + U h_{t-1} + b_r \]

\[ h_t = g(z_t) \]

\[ Y_t = \sigma(W h_t) \]
Simple RNN again

\[ z_t = V x_t + U h_{t-1} + b_r \]

\[ h_t = g(z_t) \]

\[ Y_t = V h_t + b_y \]

Activation functions:

- \[ \sigma \]

Weights:

- \[ W \]
- \[ U \]
- \[ V \]
- \[ W_1, W_2, b_1, b_2 \]
Simple RNN again: **Memories**

\[ Y_t = f(W_h + \sigma U X_t + V) \]

- \( h_t \): Previous hidden state
- \( X_t \): Current input
- \( Y_t \): Current output
- \( W, U, V, \sigma \): Weight matrices and activation function
- \( f \): Activation function (e.g., sigmoid, tanh)

Diagram shows the flow of information in an RNN, with arrows indicating the direction of data processing.
Simple RNN again: **Memories - Forgetting**

\[
Y_t = \sigma(Wh_t + UY_t + VX_t)
\]

\[
h_t = \sigma(UY_t + VX_t)
\]

\[
\text{State}
\]

\[
X_t
\]

**V**

**W**

**σ**

**σ**

**U**

**+**

**Y_t**

**h_t**
Simple RNN again: **New Events**

\[ Y_t = \sigma(W h_t + U v + b) \]
Simple RNN again: **New Events Weighted**
Simple RNN again: **Updated memories**

\[ Y_t \]

\[ h_t \]

\[ V \]

\[ X_t \]

\[ W \]

\[ \sigma \]

\[ U \]

\[ + \]

\[ \sigma \]

\[ \text{State} \]
Ref


Continue on Wednesday
Is it raining? We build an RNN to the probability if it is raining:

RNN

dog barking

white shirt

apple pie

knee hurts

get dark
RNN + Memory
RNN + Memory + Output

- Dog barking
- White shirt
- Apple pie
- Knee hurts
- Get dark

RNN + Memory + Output diagram with connections and probabilities.
LSTM: Long short term memory
Before to really understand LSTM let's see the big picture …

**Forget Gate**

\[ f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \]

**Input Gate**

\[ i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \]

**Cell State**

\[ C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t \]

**Output Gate**

\[ o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \]

\[ h_t = o_t \cdot \tanh(C_t) \]

\[ \tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_f) \]
Before to really understand LSTM lets see the big picture ...

1. LSTM are recurrent neural network with a cell and a hidden state, boths of these are updated in each step and can be thought as memories.

2. Cell states work as a long term memory and the updates depends on the relation between the hidden state in $t-1$ and the input.

3. The hidden state of the next step is a transformation of the cell state and the output (which is the section that is in general used to calculate our loss, ie information that we want in a short memory).
Let's think about my cell state

\[ x_t = \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix} \quad h_{t-1} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad C_{t-1} = \begin{bmatrix} 0.7 \\ 0.3 \end{bmatrix} \]

Let's predict if I will help you in the homework in time t
The forget gate tries to estimate what features of the cell state should be forgotten.

\[
f_t = \sigma(W_f \cdot [h_{t-1}, x_{t-1}] + b_f)
\]

\[
f_t = \sigma\left(\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 1 & 1 & -100 \\ 0.1 & 0.1 & -100 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}\right)
\]

\[
f_t = \sigma\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} -100 \\ -100 \end{bmatrix}\right)
\]

Erase everything!
Input Gate

\[
i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)
\]

\[
\tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_f)
\]

The input gate layer works in a similar way that the forget layer, the input gate layer estimate the degree of confidence of \( \tilde{C}_t \) and \( \tilde{C}_t \) is a new estimation of the cell state.

Let’s say that my input gate estimation is:

\[
i_t = \begin{bmatrix} 1 \\ 1 \end{bmatrix}
\]

\[
\tilde{C}_t = \tanh \left( \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 10 & 1 & -1 \\ -1 & 1 & 10 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right)
\]

\[
\tilde{C}_t = \tanh \left( \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 10 \\ 10 \end{bmatrix} \right)
\]

\[
\begin{bmatrix} \text{😊} & 1 \\ 1 & 1 \end{bmatrix}
\]
After the calculation of forget gate and input gate we can update our cell state.

\[ C_t = f_t \ast C_{t-1} + i_t \ast \tilde{C}_t \]

\[
C_t = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \ast \begin{bmatrix} 0.7 \\ 0.3 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} \ast \begin{bmatrix} 1 \\ 1 \end{bmatrix}
\]

\[
C_t = \begin{bmatrix} 1 \\ 1 \end{bmatrix}
\]
The output gate layer is calculated using the information of the input $x$ in time $t$ and hidden state of the last step.

It is important to notice that hidden state used in the next step is obtained using the output gate layer which is usually the function that we optimize.

\[
o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)
\]

\[
h_t = o_t \times \tanh(C_t)
\]

\[
o_t = \sigma \left( \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} 😄 & 😔 \end{bmatrix} + \begin{bmatrix} 1 & 1 & -1 \end{bmatrix} \begin{bmatrix} 😴 & 😴 & 😴 \end{bmatrix} \right)
\]

\[
o_t \approx 0.9
\]

\[
h_t \approx 0.9 \times \begin{bmatrix} 😄 & 1 \end{bmatrix} = \begin{bmatrix} 😄 & 0.9 \end{bmatrix}
\]
To optimize my parameters I basically need to do: Let's calculate all the derivatives in some time t!

\[
\frac{\partial L}{\partial W_f} = \frac{\partial L}{\partial C^t} \frac{\partial C^t}{\partial f^t} \frac{\partial f^t}{\partial W_f} C^{t-1}
\]

\[
\frac{\partial L}{\partial W_c} = \frac{\partial L}{\partial C^t} \frac{\partial C^t}{\partial (i^t \odot \hat{C}^t)} \frac{\partial (i^t \odot \hat{C}^t)}{\partial W_c} 1
\]

\[
\frac{\partial L}{\partial W_i} = \frac{\partial L}{\partial C^t} \frac{\partial C^t}{\partial (i^t \odot \hat{C}^t)} \frac{\partial (i^t \odot \hat{C}^t)}{\partial W_i} 1
\]

\[
\frac{\partial L}{\partial W_o} = \frac{\partial L}{\partial h^t} \frac{\partial h^t}{\partial o^t} \frac{\partial o^t}{\partial W_o} \tanh C^t
\]

So... every derivative is wrt the cell state or the hidden state

\[
W = W - \eta \frac{\partial L}{\partial W}
\]
Let’s calculate the cell state and the hidden state

\[
\frac{\partial L}{\partial h_{t-1}} = \frac{\partial L}{\partial C^t} \left( \frac{\partial C^t}{\partial f^t} \frac{\partial f^t}{\partial h^t} + \frac{\partial C^t}{\partial (i^t \odot C^t)} \frac{\partial (i^t \odot C^t)}{\partial h^t} \right) + \frac{\partial L}{\partial h^t} \frac{\partial h^t}{\partial \sigma^t} \frac{\partial \sigma^t}{\partial h_{t-1}}
\]

\[
\frac{\partial L}{\partial C^t} = \frac{\partial L}{\partial (f^{t+1} \odot C^t + i^{t+1} \odot \hat{C}^t)} \frac{\partial (f^{t+1} \odot C^t + i^{t+1} \odot \hat{C}^t)}{\partial C^t} \left( \frac{\partial L}{\partial C^{t+1}} + \frac{\partial L}{\partial h^{t+1}} \frac{\partial h^{t+1}}{\partial C^{t+1}} \right) \odot f^{t+1}
\]
RNN Structures

• The **one to one** structure is useless.
• It takes a single input and it produces a single output.
• Not useful because the RNN cell is making little use of its unique ability to remember things about its input sequence.

\[ Y_t \]

\[ X_t \]
The many to one structure reads in a sequence and gives us back a single value. Example: Sentiment analysis, where the network is given a piece of text and then reports on some quality inherent in the writing. A common example is to look at a movie review and determine if it was positive or negative.
RNN Structures (cont)

The **one to many** takes in a single piece of data and produces a sequence. For example we give it the starting note for a song, and the network produces the rest of the melody for us.
The **many to many** structures are in some ways the most interesting. Used for machine translation. Example: Predict if it will rain given some inputs.
This form of **many to many** can be used for machine translation.

For example, the English sentence: “**The black dog jumped over the cat**”
In Italian as: “**Il cane nero saltò sopra il gatto**”
In the Italia, the adjective “nero” (black) follows the noun “cane” (dog), so we need to have some kind of buffer so we can produce the words in their proper English.
Bidirectional

LSTM and RNN are designed to analyze sequence of values.

For example: *Patrick said he needs a vacation.*
*he* here means *Patrick* and we know this because *Patrick* was before the word *he*.

However consider the following sentence:
*He needs to work more, Pavlos said about Patrick.*

Bidirectional RNN or BRNN or bidirectional LSTM or BLSTM when using LSTM units.
Bidirectional (cond)

Symbol for a BRNN

\[ Y_t \]

\[ X_t \]

Previous state

\[ Y_{t-2} \]

\[ Y_{t-1} \]

\[ Y_t \]

\[ X_{t-2} \]

\[ X_{t-1} \]

\[ X_t \]

Previous state
Deep RNN

LSTM units can be arranged in layers, so that each the output of each unit is the input to the other units. This is called a deep RNN, where the adjective “deep” refers to these multiple layers.

- Each layer feeds the LSTM on the next layer
- First time step of a feature is fed to the first LSTM, which processes that data and produces an output (and a new state for itself).
- That output is fed to the next LSTM, which does the same thing, and the next, and so on.
- Then the second time step arrives at the first LSTM, and the process repeats.
Deep RNN

\[ Y \quad Y_{t-2} \quad Y_{t-1} \quad Y_t \quad Y_{t+1} \quad Y_{t+2} \]

\[ X \quad X_{t-2} \quad X_{t-1} \quad X_t \quad X_{t+1} \quad X_{t+2} \]
Skip Connections

Add additional connections between units \( d \) time steps apart
Creating paths through time where gradients neither vanish or explode
Leaky Units

Linear self-connections
Maintain cell state: running average of past hidden activations
Standard RNN

\[ C^{(t)} = \tanh(W h^{(t-1)} + U x^{(t-1)}) \]

\[ h^{(t)} = C^{(t)} \]
Leaky Unit

\[
C^{(t)} = \tanh(W h^{(t-1)} + U x^{(t-1)})
\]

\[
h^{(t)} = \alpha h^{(t-1)} + (1-\alpha)C^{(t)}
\]