80%  D
15%  Re
5%  chord
80% D
15% Re
5% chord

50% line
25% flame
25% current

99% Net
1% Monocles
Image captioning
Recurrence is required to capture the representational dynamics of the human visual system

Tim C. Kietzmann\textsuperscript{a,b,1}, Courtney J. Spoerer\textsuperscript{a}, Lynn K. A. Sörensen\textsuperscript{c}, Radoslaw M. Cichy\textsuperscript{d}, Olaf Hauk\textsuperscript{a}, and Nikolaus Kriegeskorte\textsuperscript{e}

\textsuperscript{a}MRC Cognition and Brain Sciences Unit, University of Cambridge, Cambridge CB2 7EF, United Kingdom; \textsuperscript{b}Donders Institute for Brain, Cognition and Behaviour, Radboud University, 6525 HR Nijmegen, The Netherlands; \textsuperscript{c}Department of Psychology, University of Amsterdam, 1018 WD Amsterdam, The Netherlands; \textsuperscript{d}Department of Education and Psychology, Freie Universität Berlin, 14195 Berlin, Germany; and \textsuperscript{e}Department of Psychology, Columbia University, New York, NY 10027

PNAS 2019
Outline

• Conceptual overview or RNNs
  • FFN vs RNN
  • Flavours of RNNs
  • Training RNNs
  • Real-world RNN

• Jupyter Notebook
  • Build simple RNN (Keras)
  • Analyse some properties
Multi-layer perceptron
Why they are powerful interpolators

ReLU
Why they are powerful interpolators

ReLU
Why they are powerful interpolators

Building piecewise local linear models
(even more flexible when stacking multiple layers)
Example: fitting sine function
Don’t use this for extrapolation!
**Feedforward**
(downstream/upstream)

\[
\begin{align*}
\text{output} &= W_4 \ast h_3 \\
\end{align*}
\]

\[
\begin{align*}
h_3 &= W_3 \ast h_2 \\
h_2 &= W_2 \ast h_1 \\
h_1 &= W_1 \ast \text{x}
\end{align*}
\]

\[\text{x}\]

**Recurrent (example)**
(no hierarchy)

\[\text{x}\]
Recurrent (more common)
(only lateral recurrence)

\[ h_1 = W_1 x + V_1 h_1 \]

input = x

How do we know \( h_1 \)?
Recurrent (temporal)

\[ h_1(t) = W_1 x(t) + V_1 h_1(t-1) \]

!! \( W_1 \) and \( V_1 \) don’t depend on time !!
Memories
Output depends on all past input through multiple complex transformations
Flavours of RNNs

• Cardinality of input-output

• How the hidden layers are passed forward
Vanilla RNN

\[ h(t) = \text{nonlin}[ W x(t) + V h(t-1) ] \]
Vanilla RNN

\[ h(t) = \text{nonlin}[W^*x(t) + V^*h(t-1)] \]
Vanilla RNN

\[ h(t) = \text{nonlin}[W x(t) + V h(t-1)] \]

x(t) \rightarrow h(t) \rightarrow h(t-1)

feed through

“Gated” RNN (examples)

\[ x(t) \rightarrow h(t-1) \rightarrow h(t) \]
delete some bits

h(t)

GRU
Vanilla RNN

\[ h(t) = \text{nonlin}[W x(t) + V h(t-1)] \]

feed through

“Gated” RNN (examples)

\[
\begin{align*}
\text{x(t)} & \quad \rightarrow \quad h(t) \\
\text{h(t-1)} & \quad \rightarrow \quad h(t)
\end{align*}
\]

delete some bits

LSTM

\[
\begin{align*}
\text{x(t)} & \quad \rightarrow \quad h(t) \\
\text{h(t-1)} & \quad \rightarrow \quad h(t)
\end{align*}
\]

delete some bits
keep some for later use

GRU
Vanilla RNN

\[ h(t) = \text{nonlin}[W \cdot x(t) + V \cdot h(t-1)] \]

```
x(t) \rightarrow h(t-1) \rightarrow h(t)
```

feed through

"Gated" RNN (examples)

```
x(t) \rightarrow h(t-1) \rightarrow h(t)
```

GRU

!!Learn which bits to delete!!

```
x(t) \rightarrow h(t-1) \rightarrow h(t)
```

LSTM

delete some bits

keep some for later use
How can you delete (and learn what to delete?)

output

1
How can you delete (and learn what to delete?)

\[ \text{input} = \text{Weights} \times [x,h] \]
How can you delete (and learn what to delete?)

input = Weights * [x,h]

(output)

(in practice, smooth sigmoid used instead)
How can you delete (and learn what to delete?)

\[ \text{input} = \text{Weights} \times [x, h] \]

(in practice, smooth sigmoid used instead)

This learns which (combinations of) x’s and h’s to delete
LSTM (approx)

(output depends on input, state, and memory)
(memory explicitly changed by input and state)

feed through

delete some bits
Training FFN

- **One forward pass**
  - Input: $x$
  - Functions
  - Output: Loss

- **Another forward pass**
  - Input: Loss
  - Gradients with respect to inputs
Training RNNs

• Back-propagation through time
A “real-life” RNN

Spanish  
usted es muy

English  
you are very

Google Translate
A “real-life” RNN

Spanish
 usted es muy

English
 you are very

Google Translate
How does it work?
How does it work?

Word -> vector of numbers (word “embedding”, word piece model)
How does it work?

Word -> vector of numbers
Sentence -> trajectory

(word “embedding”, word piece model)
How does it work?

- Word -> vector of numbers (word “embedding”, word piece model)
- Sentence -> trajectory
- Two separate networks (encoding/decoding)
How does it work?

Word -> vector of numbers  
(word “embedding”, word piece model)

Sentence -> trajectory

Two separate networks (encoding/decoding)
How does it work?

Word -> vector of numbers  
Sentence -> trajectory  
Two separate networks (encoding/decoding)
Time to do hands-on playing with RNNs