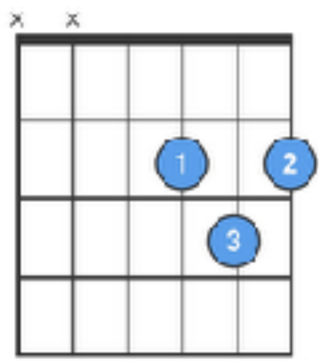
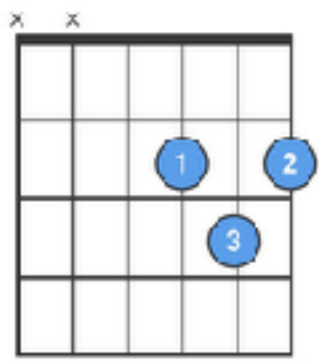
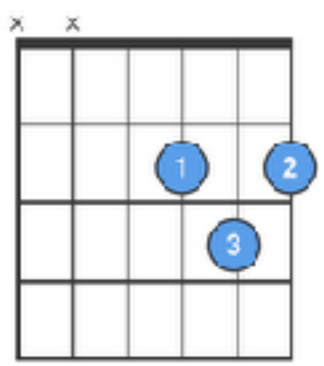

1
2
3



80% D

15% Re

5% chord



80% D

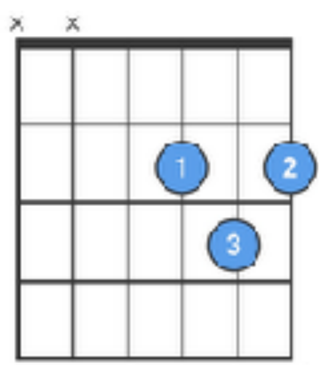
15% Re

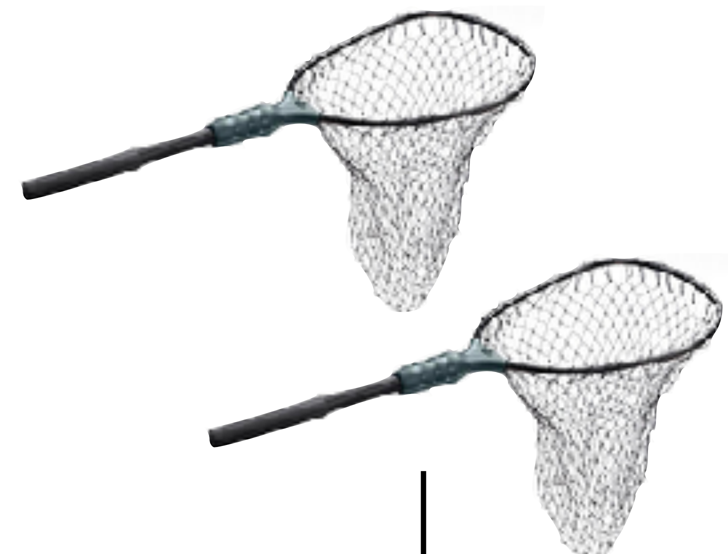
5% chord

50% line

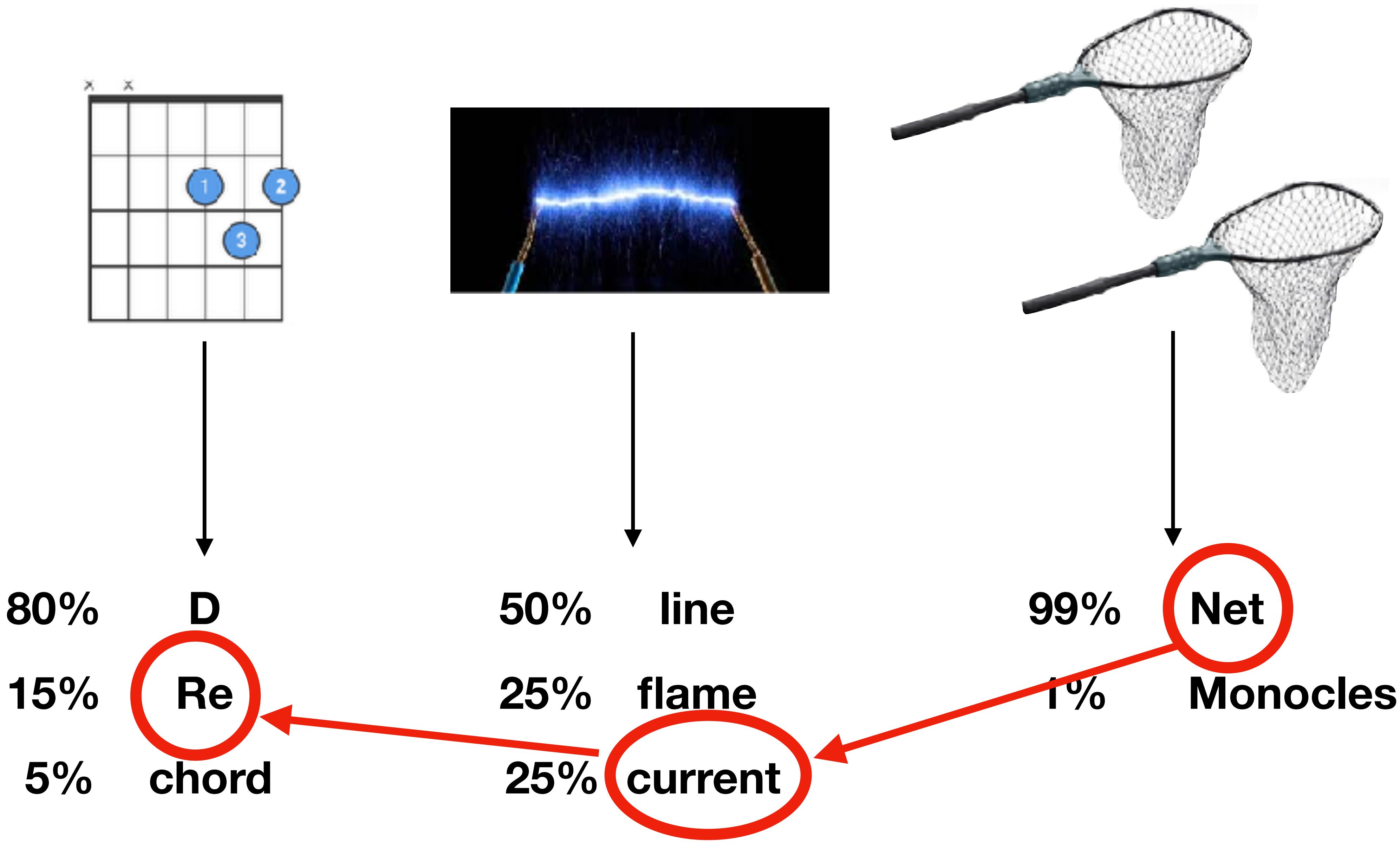
25% flame

25% current





80%	**D**	**50%**	**line**	**99%**	**Net**
15%	**Re**	**25%**	**flame**	**1%**	**Monocles**
5%	**chord**	**25%**	**current**		



80% D
15% Re
5% chord
50% line
25% flame
25% current
99% Net
1% Monocles

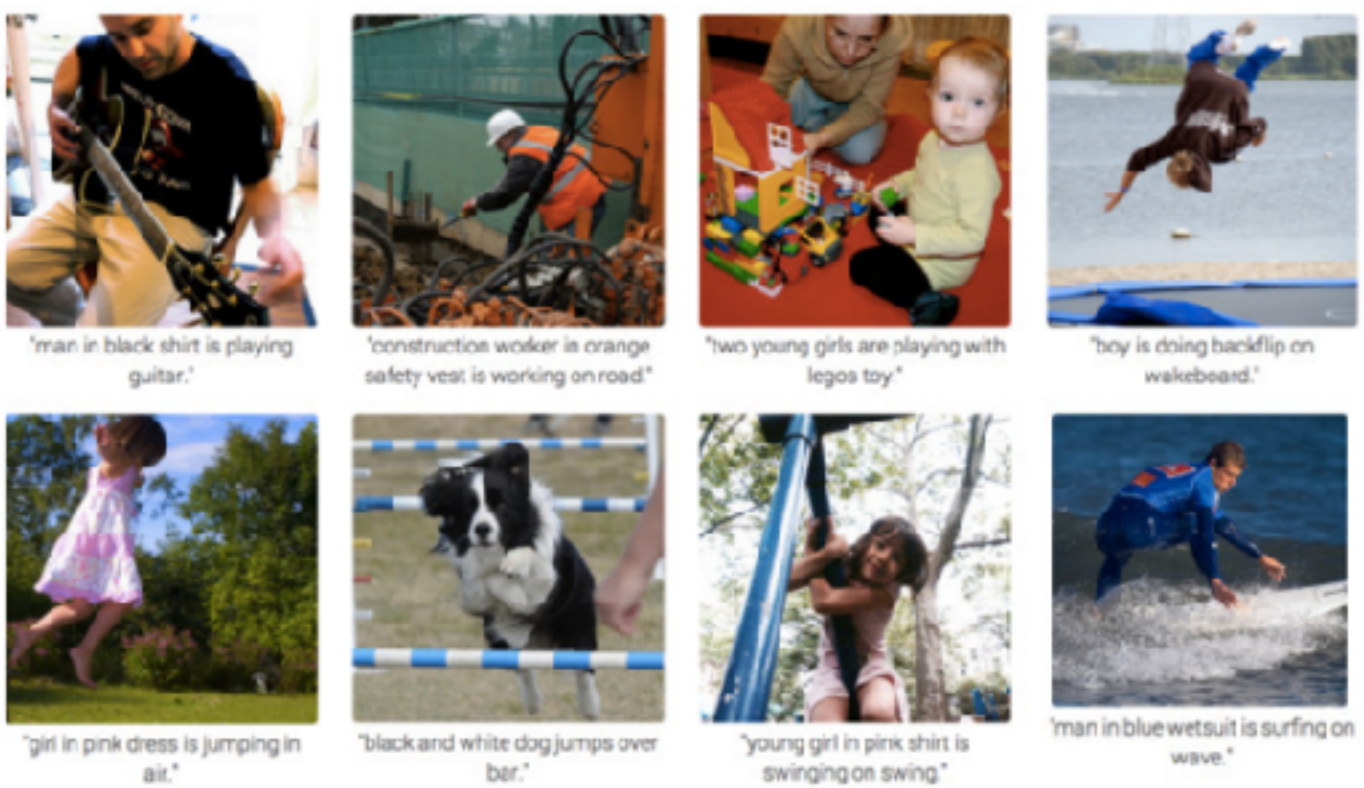
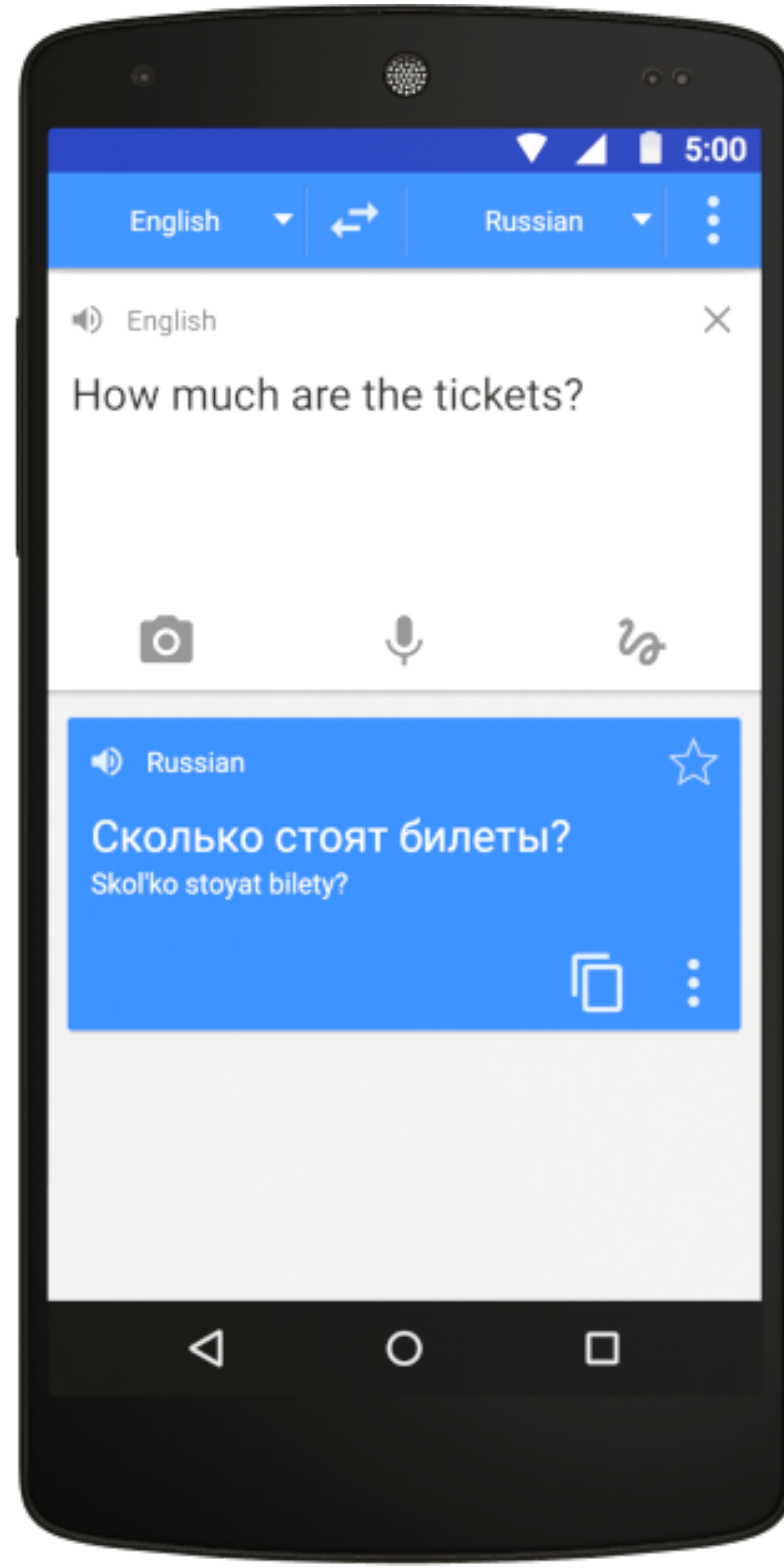


Image captioning

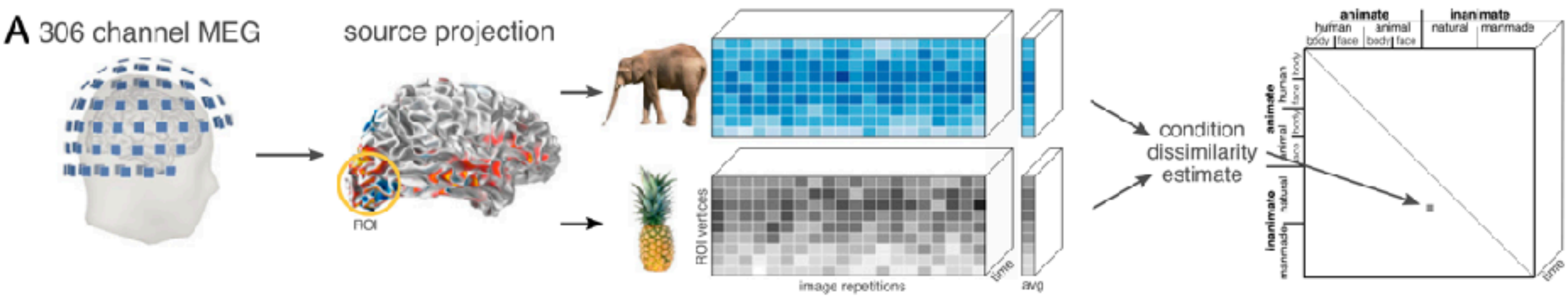


Recurrence is required to capture the representational dynamics of the human visual system

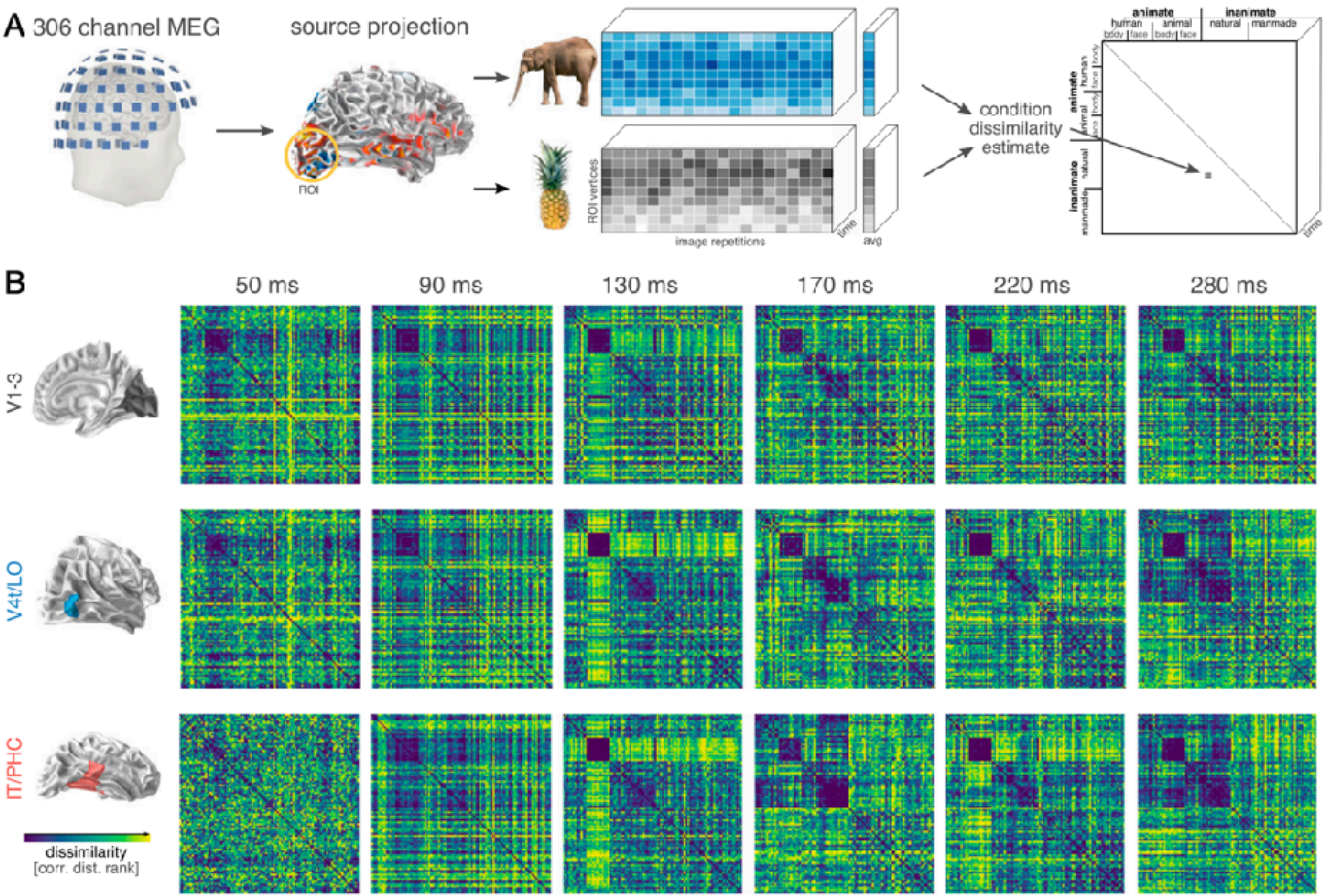
Tim C. Kietzmann[a,b,1], Courtney J. Spoerer[a], Lynn K. A. Sörensen[c], Radoslaw M. Cichy[d], Olaf Hauk[a], and Nikolaus Kriegeskorte[e]

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

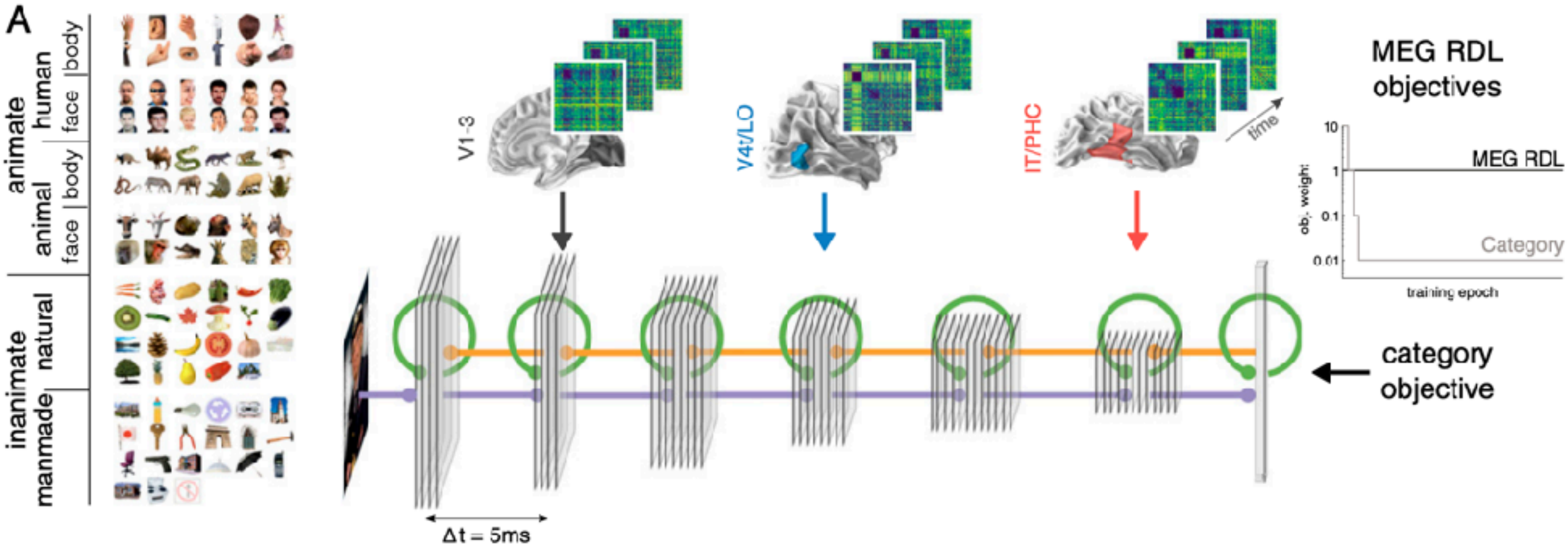
PNAS 2019



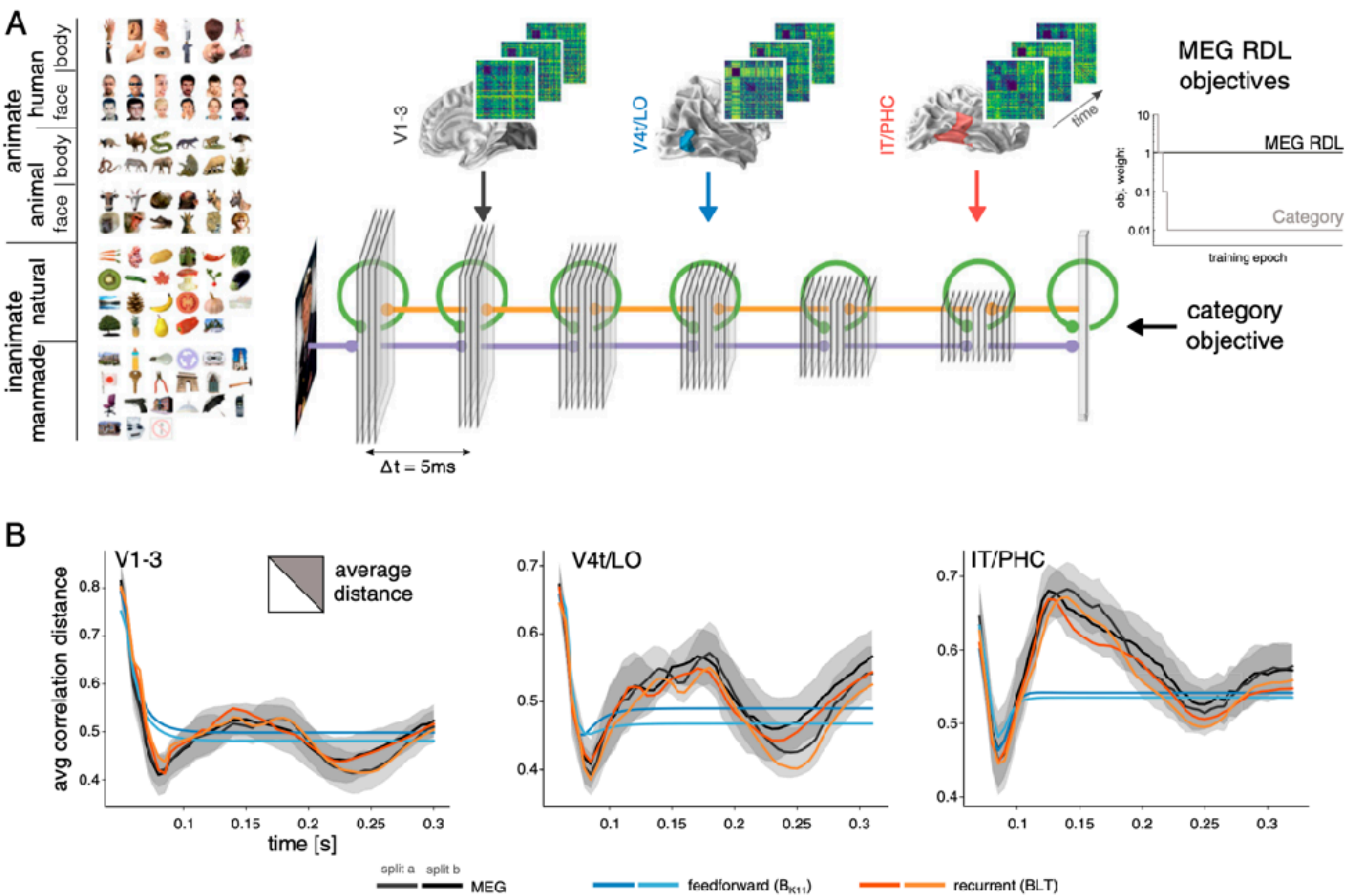
Kietzmann 2019



Kietzmann 2019



Kietzmann 2019



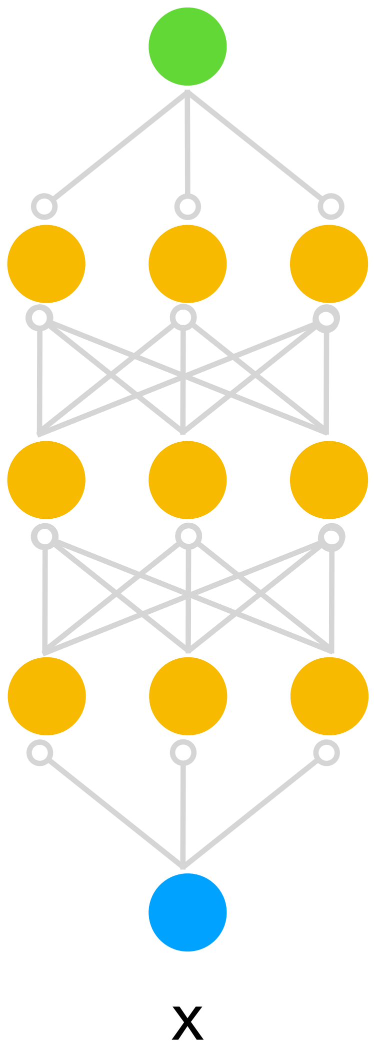
Kietzmann 2019

Outline

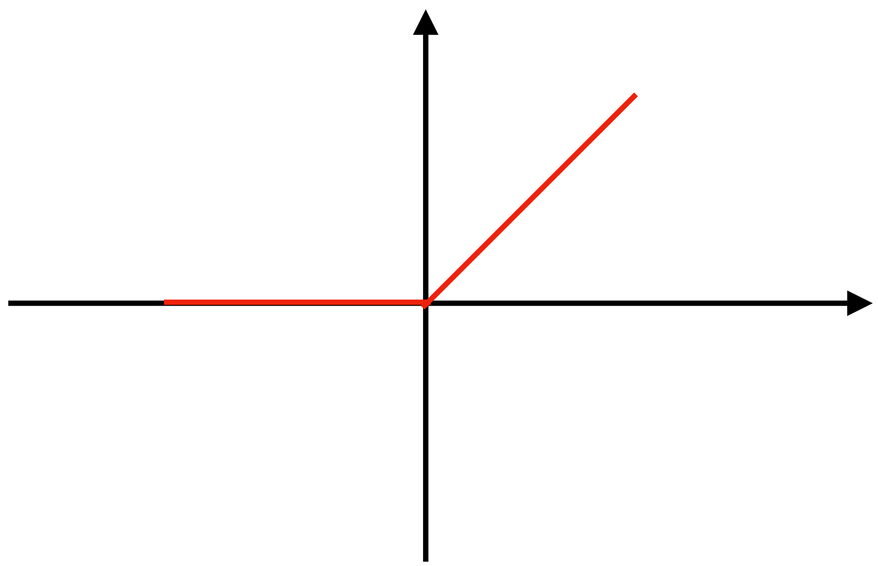
- Conceptual overview or RNNs
  - FFN vs RNN
  - Flavours of RNNs
  - Training RNNs
  - Real-world RNN

- Jupyter Notebook
  - Build simple RNN (Keras)
  - Analyse some properties

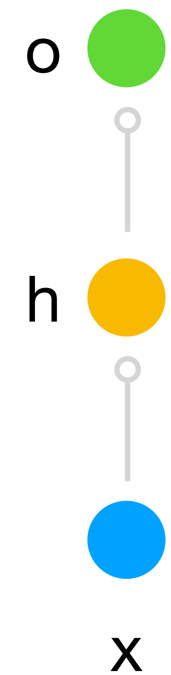
Multi-layer perceptron

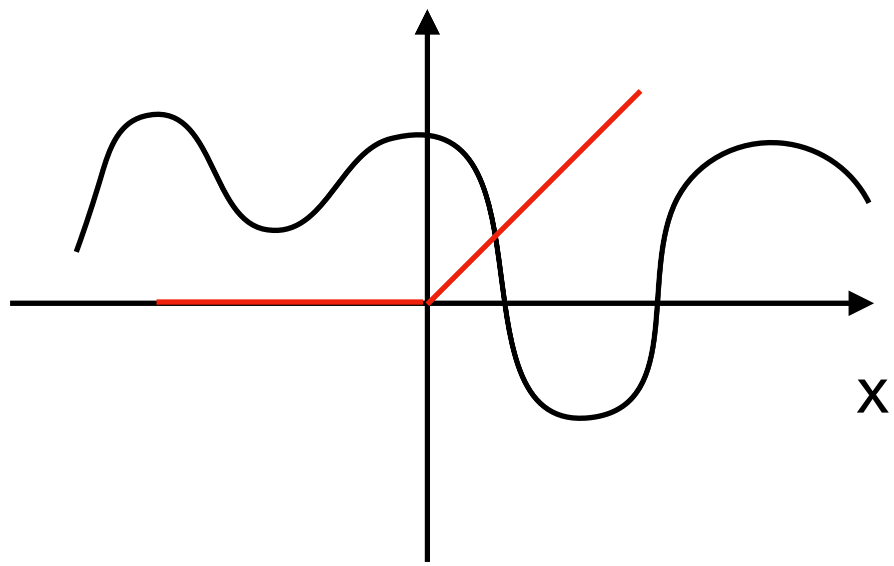


Why they are powerful interpolators

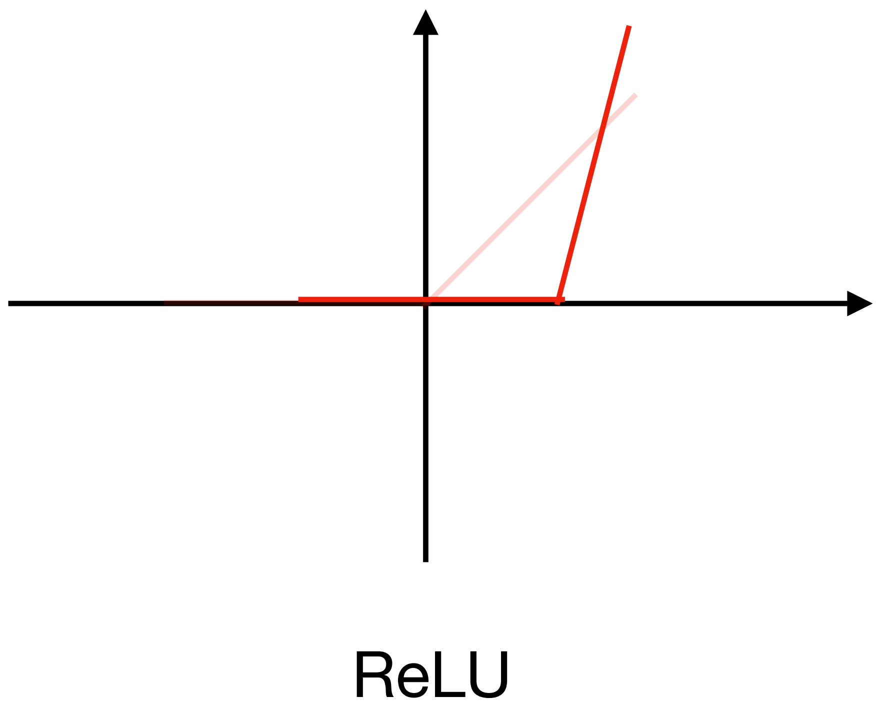


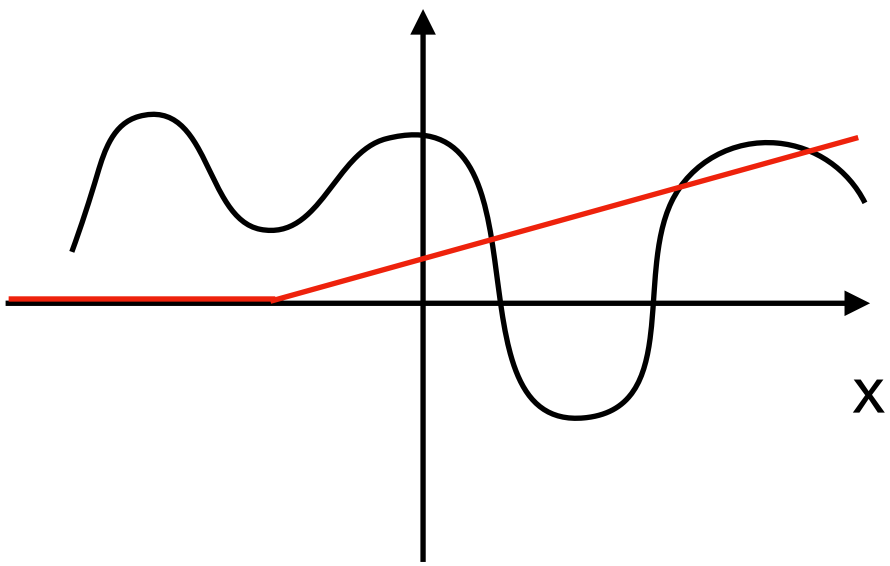
ReLU



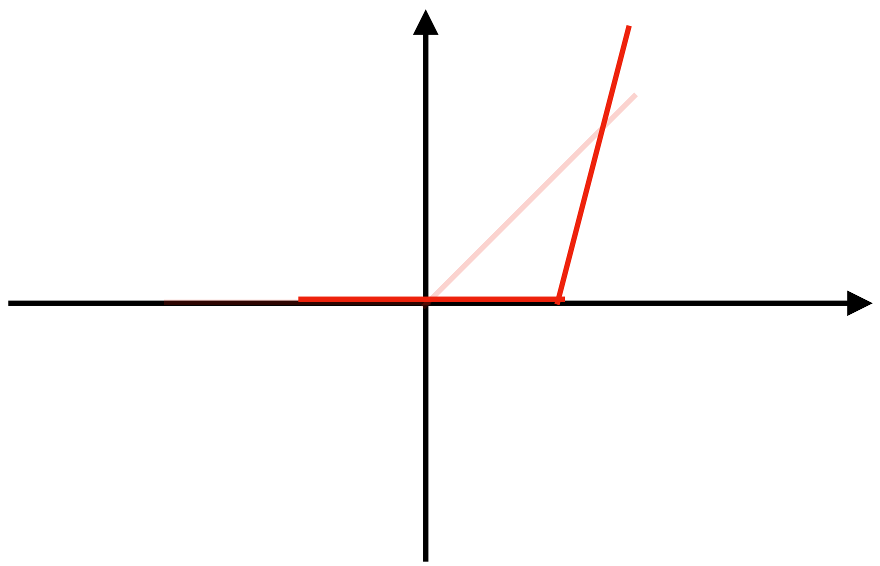


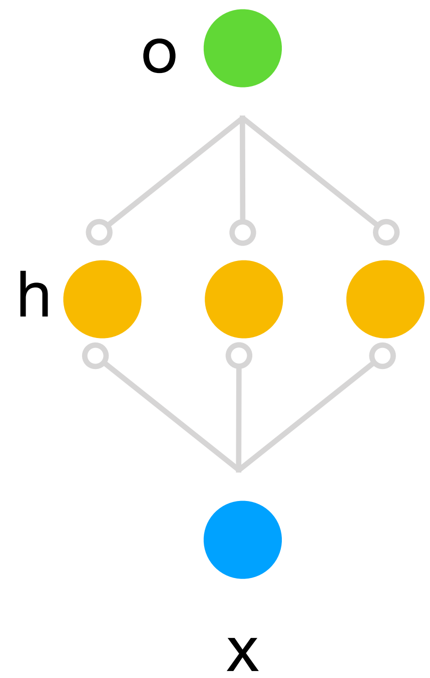
Why they are powerful interpolators

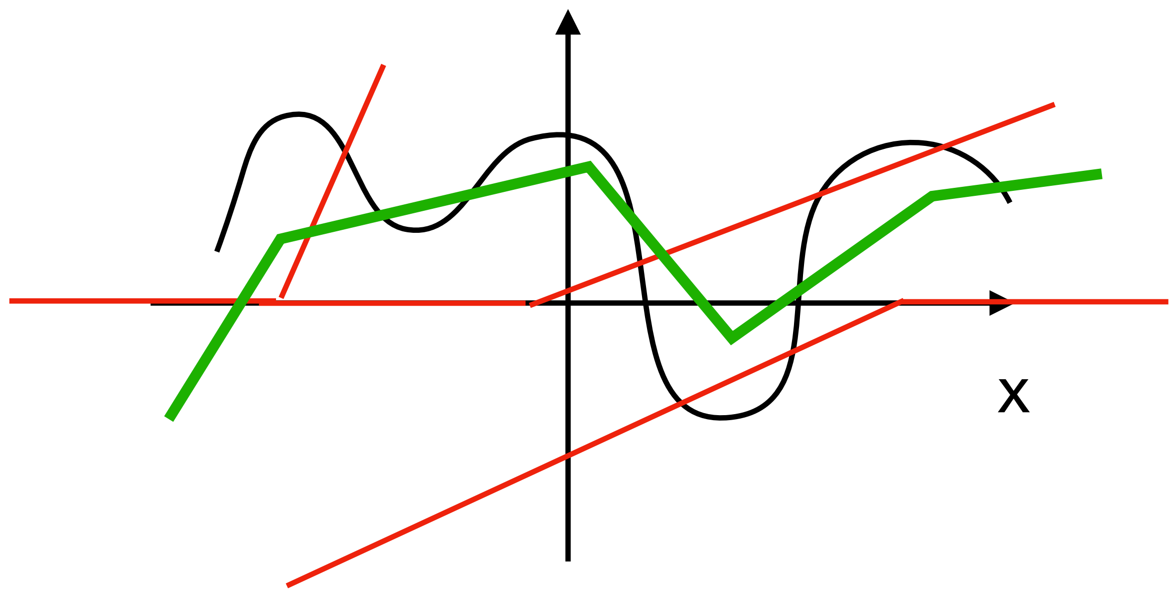




Why they are powerful interpolators

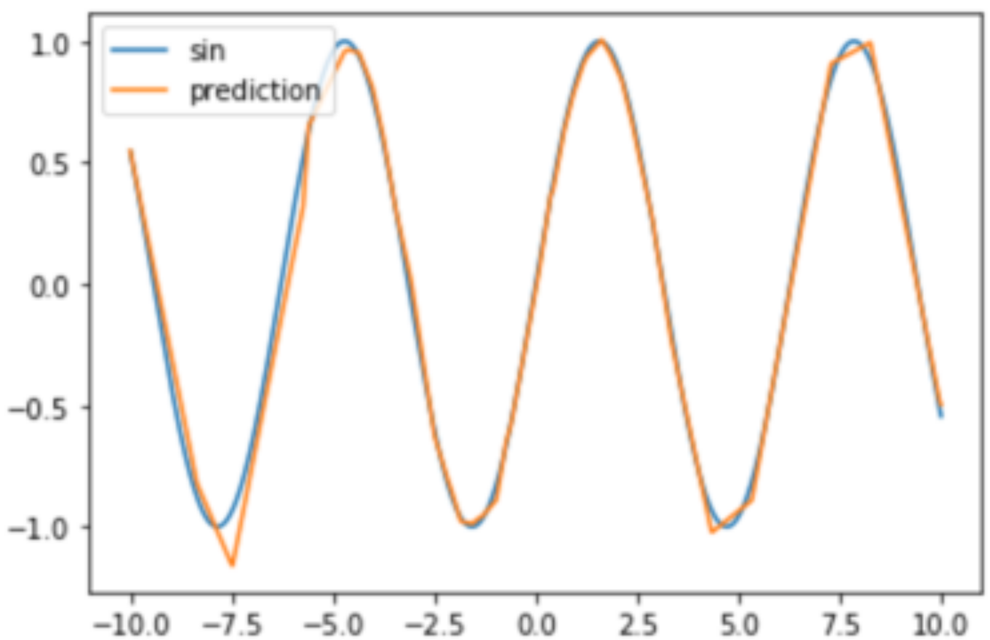


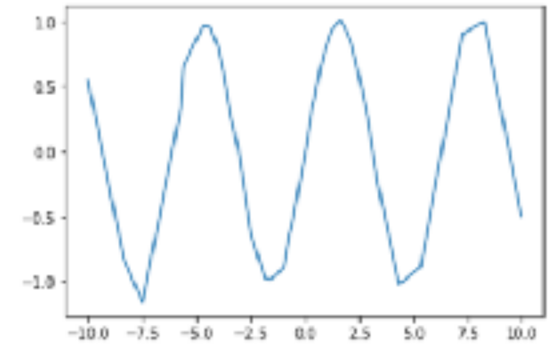




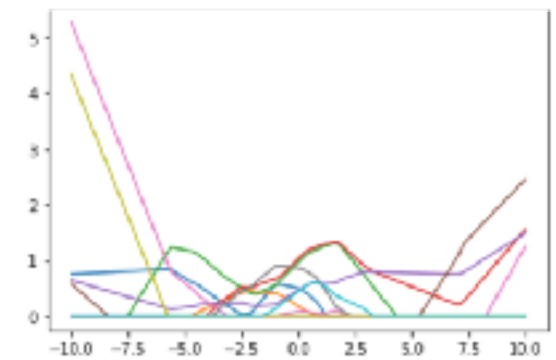
Building piecewise local linear models
(even more flexible when stacking multiple layers)

Example: fitting sine function

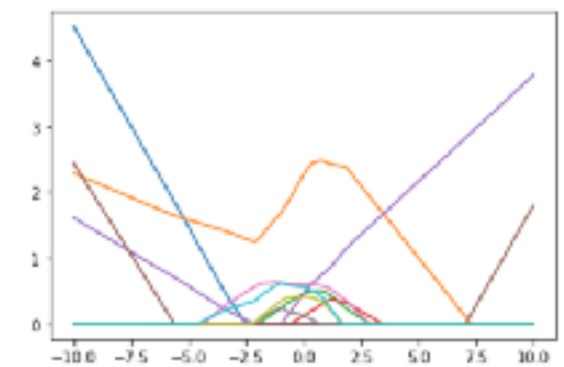




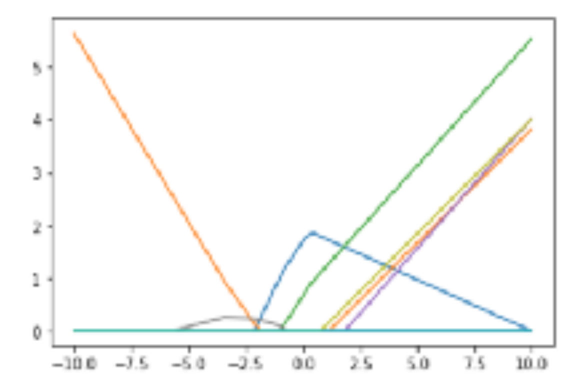
output



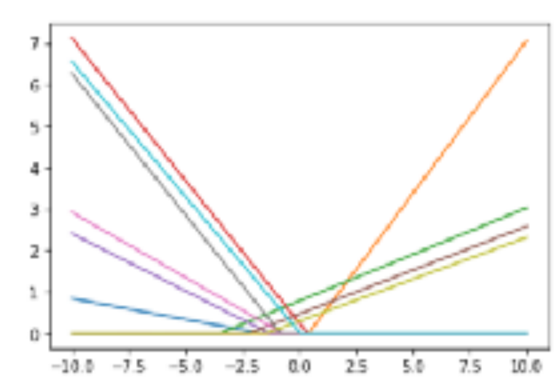
Layer 4



Layer 3

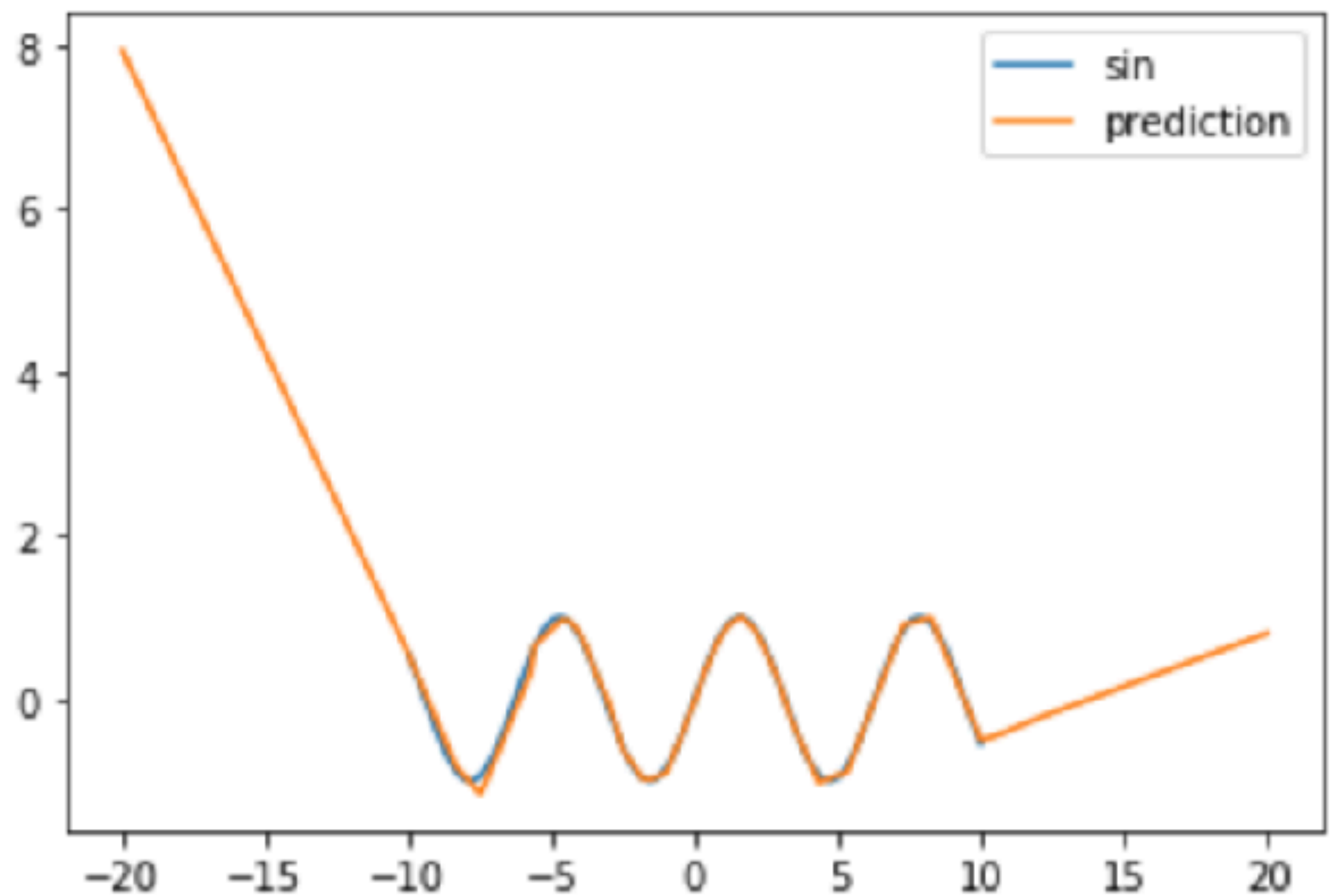


Layer 2

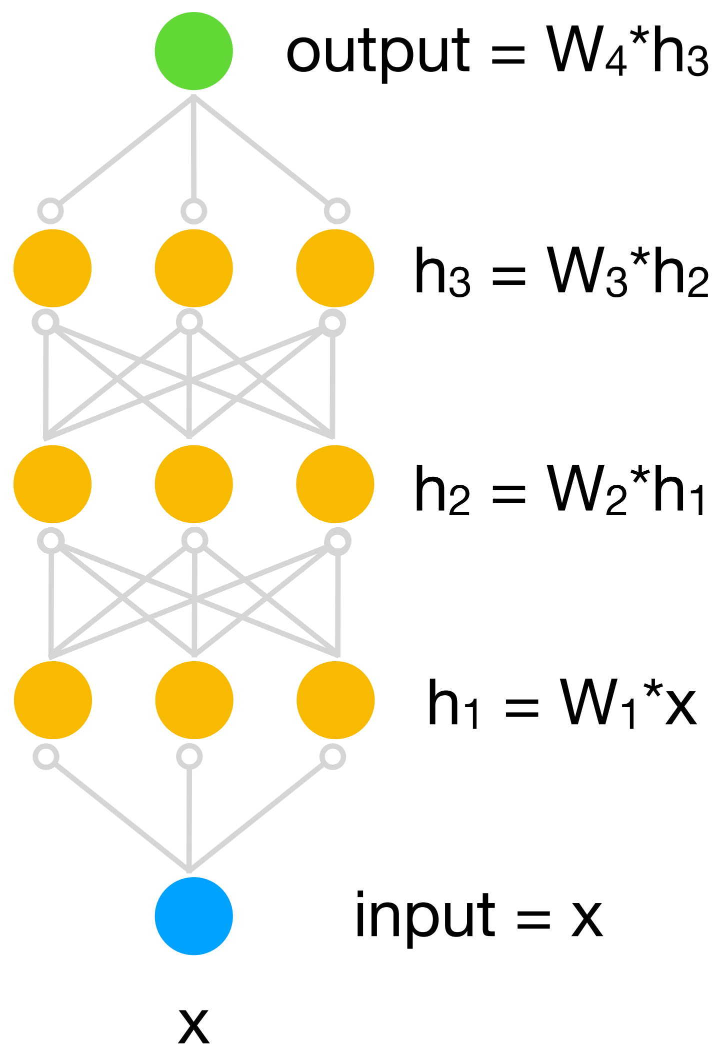


Layer 1

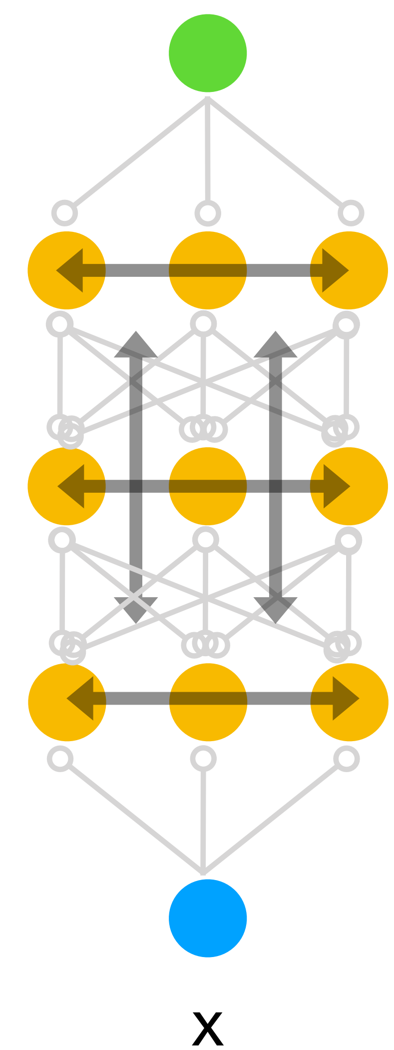
Don’t use this for extrapolation!



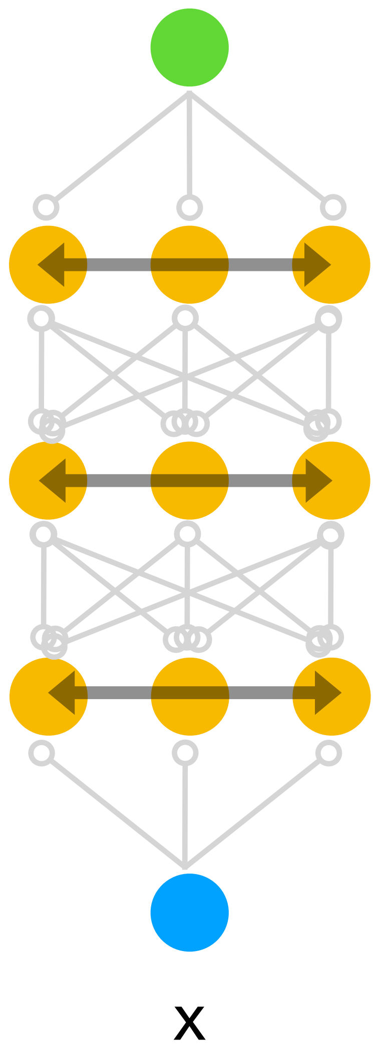
Feedforward
(downstream/upstream)



Recurrent (example)
(no hierarchy)



Recurrent (more common)
(only lateral recurrence)

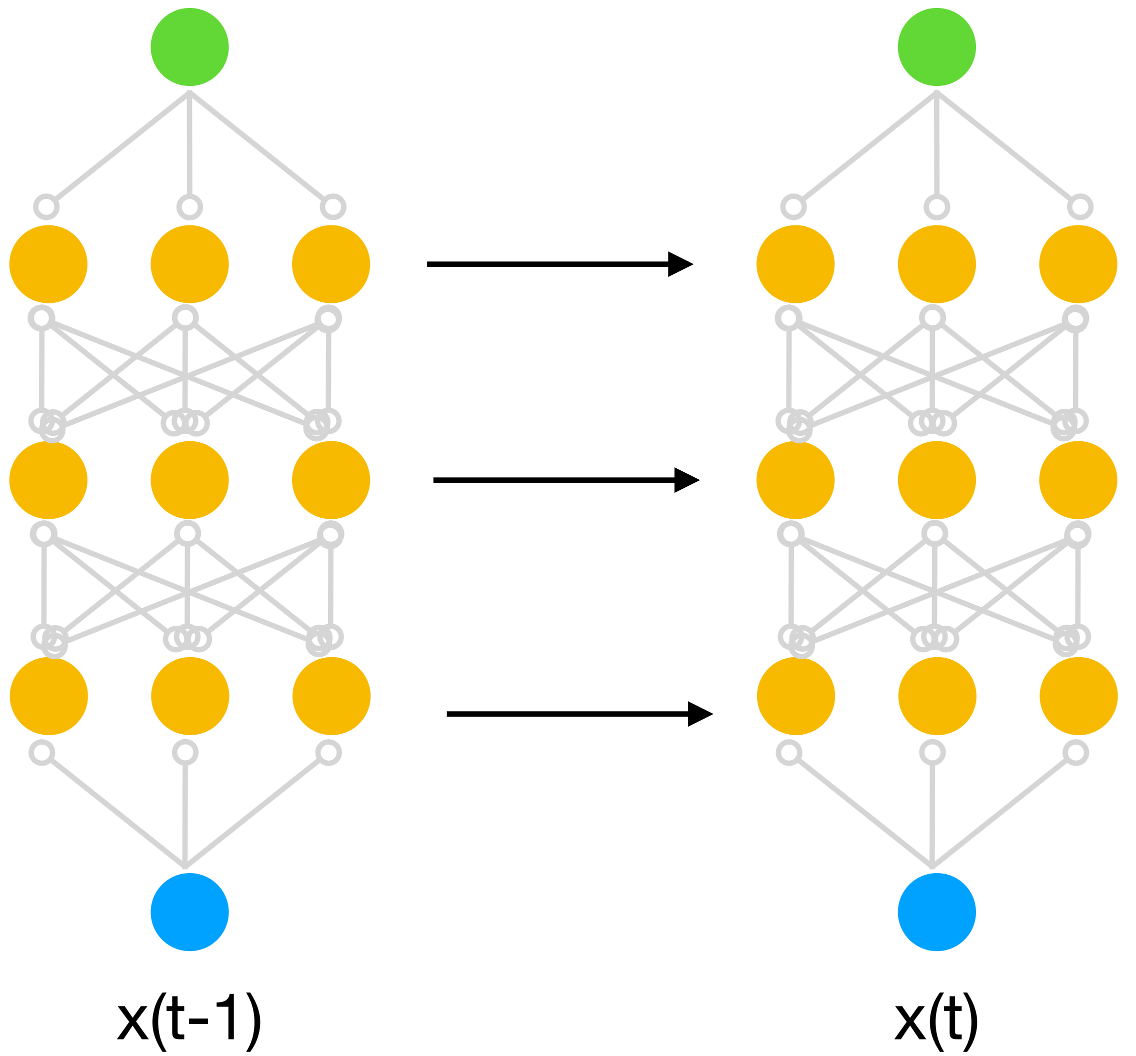


$h_1 = W_1 * x + V_1 * h_1$

input = x

How do we know h_1?

Recurrent (temporal)

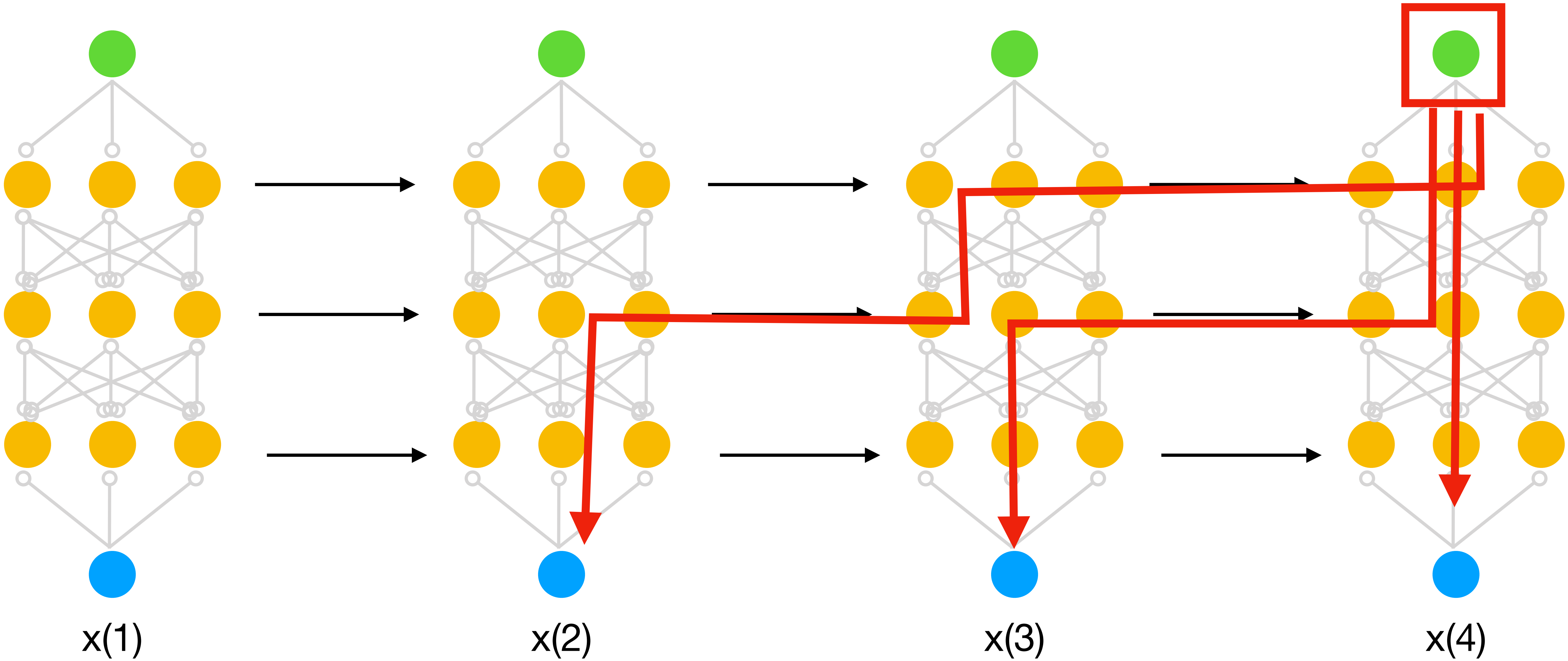


$h_1(t) = W_1 * x(t) + V_1 * h_1(t-1)$

!! W_1 and V_1 don't depend on time !!

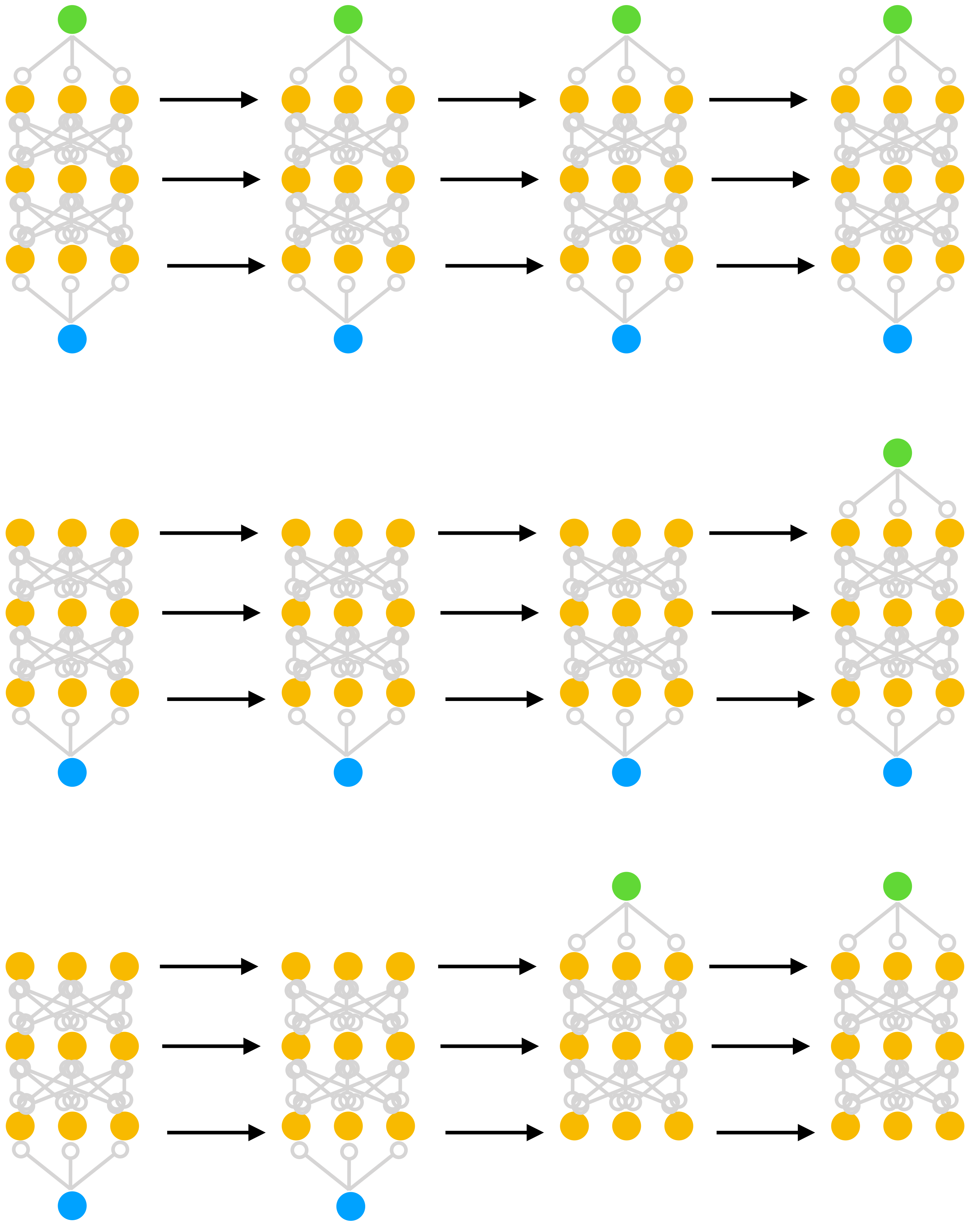
Memories

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) = nonlin[W*x(t) + V*h(t-1)]

Vanilla RNN

h(t) = nonlin[W*x(t) + V*h(t-1)]

$$\begin{matrix} x(t) \\ h(t-1) \end{matrix} \xrightarrow[\text{feed through}]{} h(t)$$

Vanilla RNN

h(t) = nonlin[W*x(t) + V*h(t-1)]

x(t)
h(t-1) $\xrightarrow[\text{feed through}]{}$ h(t)

“Gated” RNN

(examples)

x(t)
h(t-1) $\xrightarrow[\text{delete some bits}]{}$ h(t)

GRU

Vanilla RNN

h(t) = nonlin[W*x(t) + V*h(t-1)]

x(t)
h(t-1) $\longrightarrow$ h(t)
feed through

“Gated” RNN

(examples)

x(t)
h(t-1) $\longrightarrow$ h(t)
delete some bits

GRU

x(t)
h(t-1) $\longrightarrow$ h(t)
delete some bits
keep some for later use

LSTM

Vanilla RNN

h(t) = nonlin[W*x(t) + V*h(t-1)]

x(t)
h(t-1) $\longrightarrow$ h(t)
feed through

"Gated" RNN

(examples)

x(t)
h(t-1) $\longrightarrow$ h(t)
delete some
bits

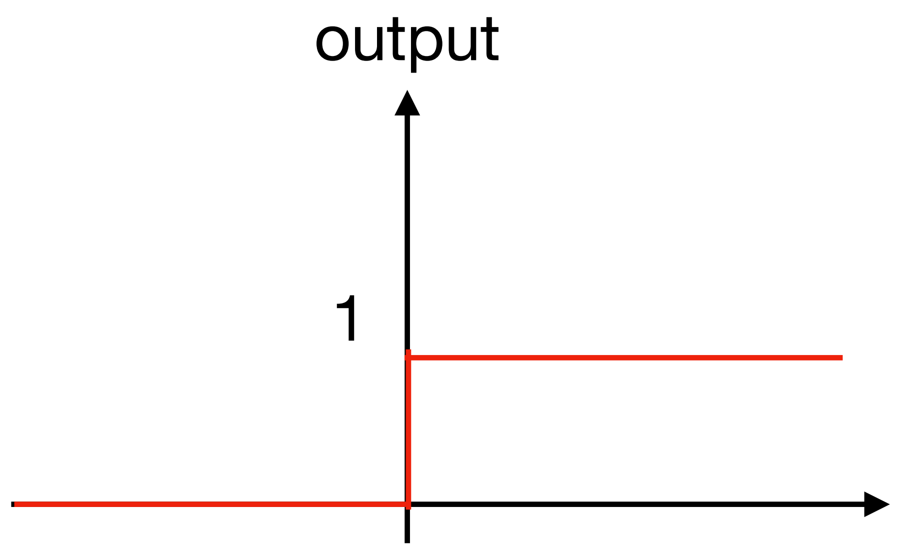
GRU

!!Learn which
bits to delete!!

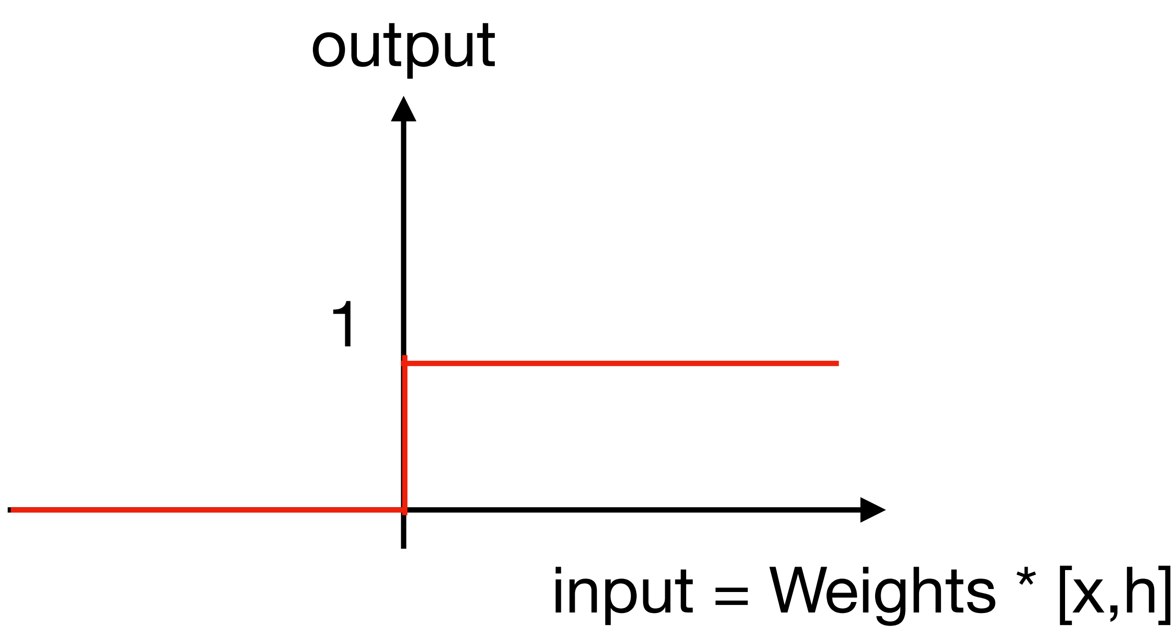
x(t)
h(t-1) $\longrightarrow$ h(t)
delete some
bits
keep some
for later use

LSTM

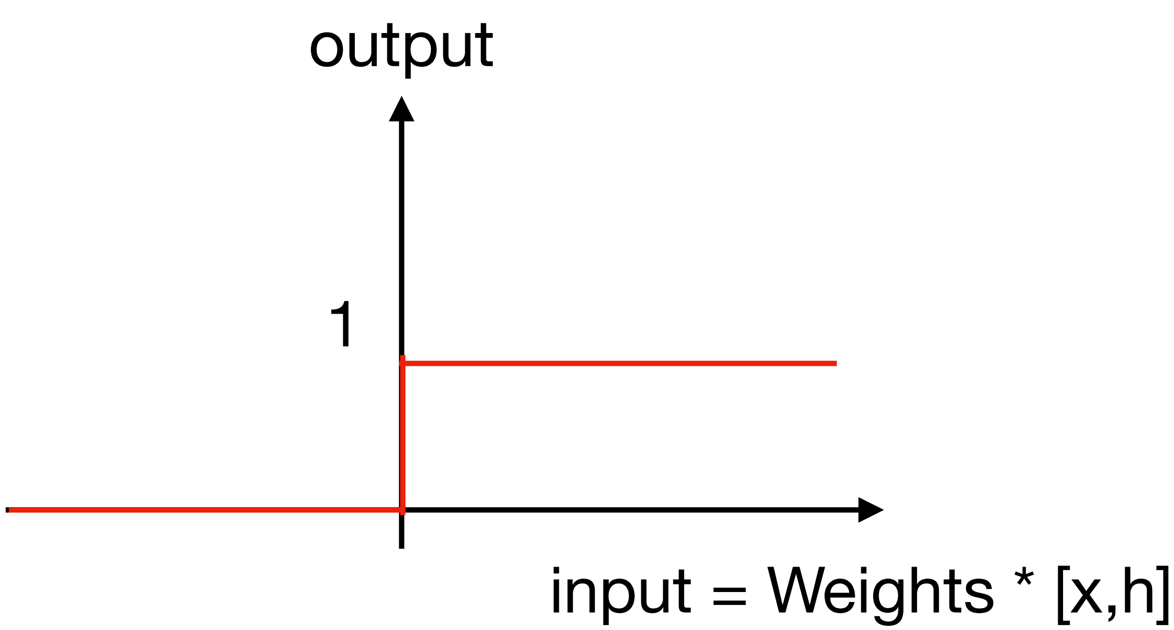
How can you delete (and learn what to delete?)



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

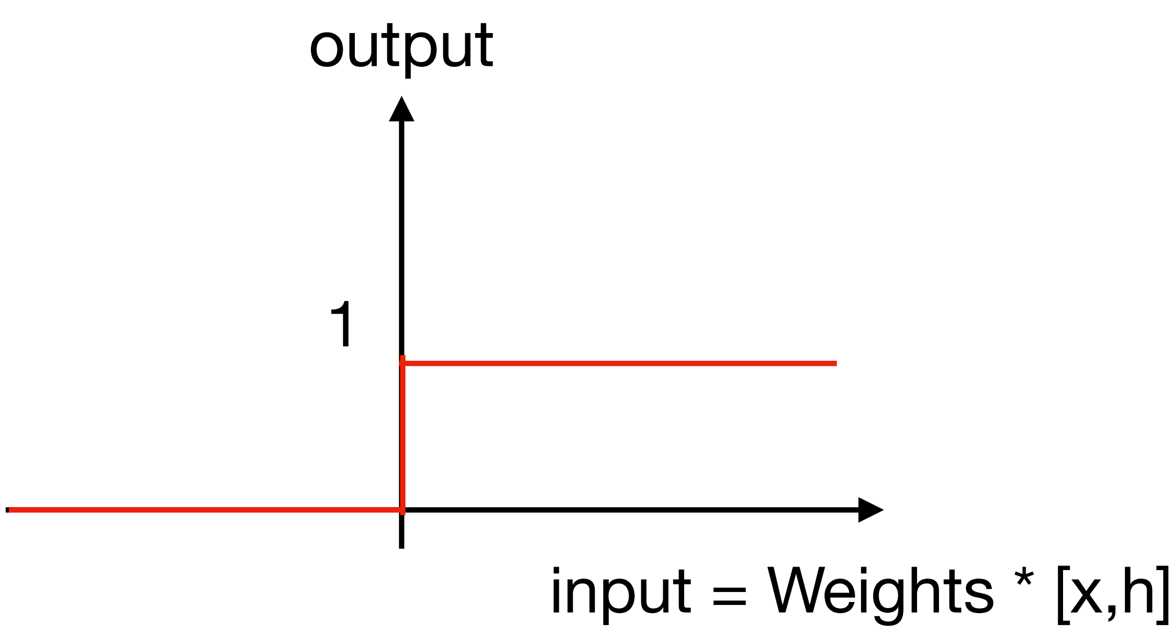


How can you delete (and learn what to delete?)



(in practice, smooth
sigmoid used instead)

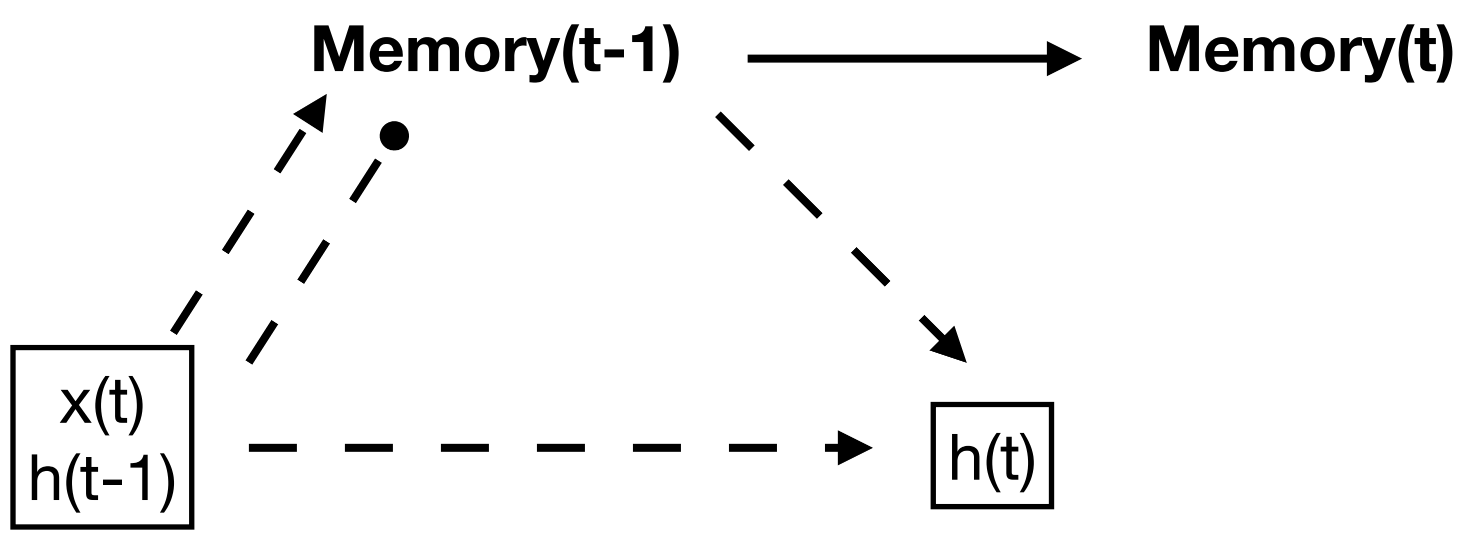
How can you delete (and learn what to delete?)



(in practice, smooth sigmoid used instead)

this learns which (combinations of) x’s and h’s to delete

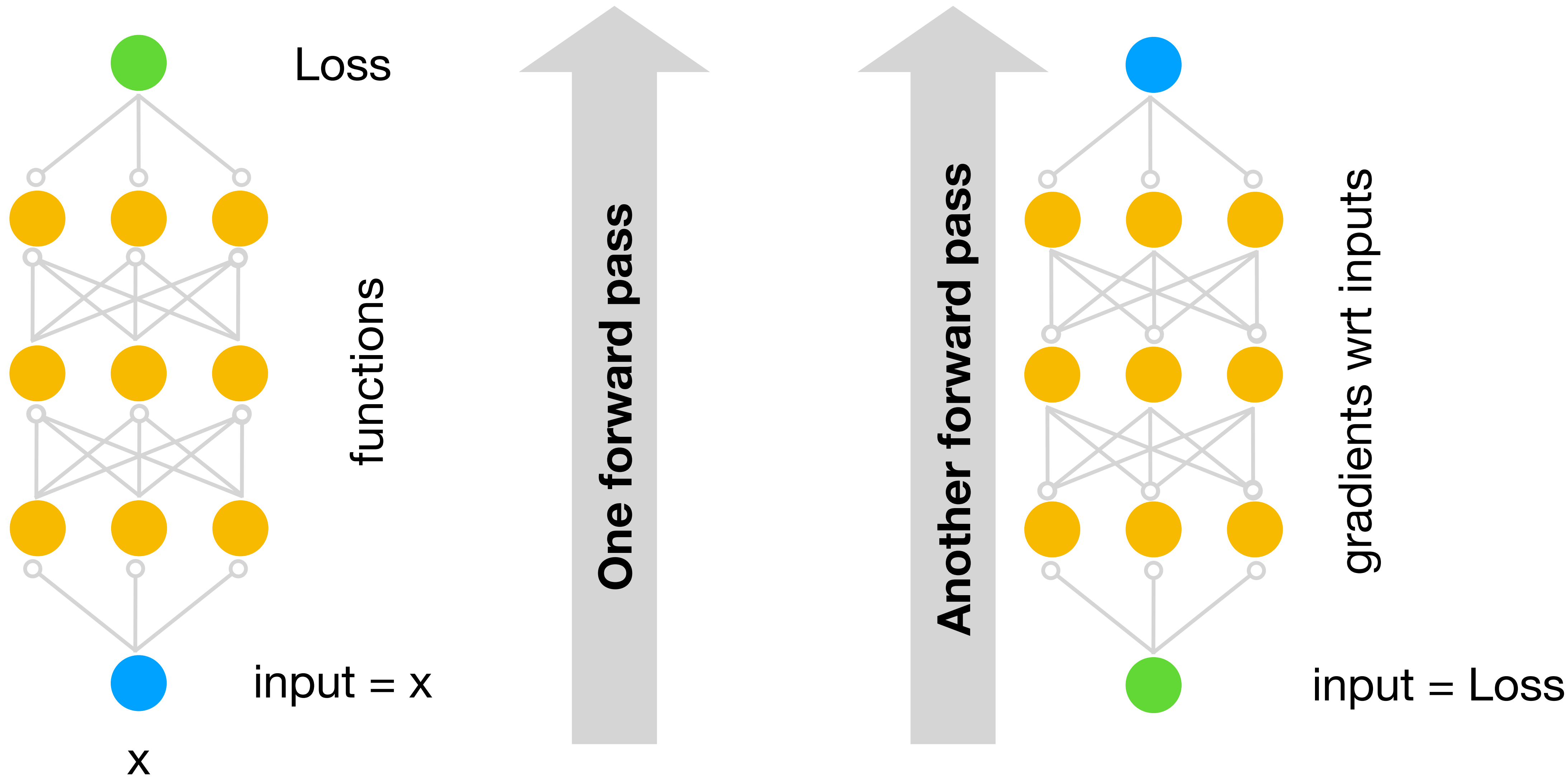
LSTM (approx)



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

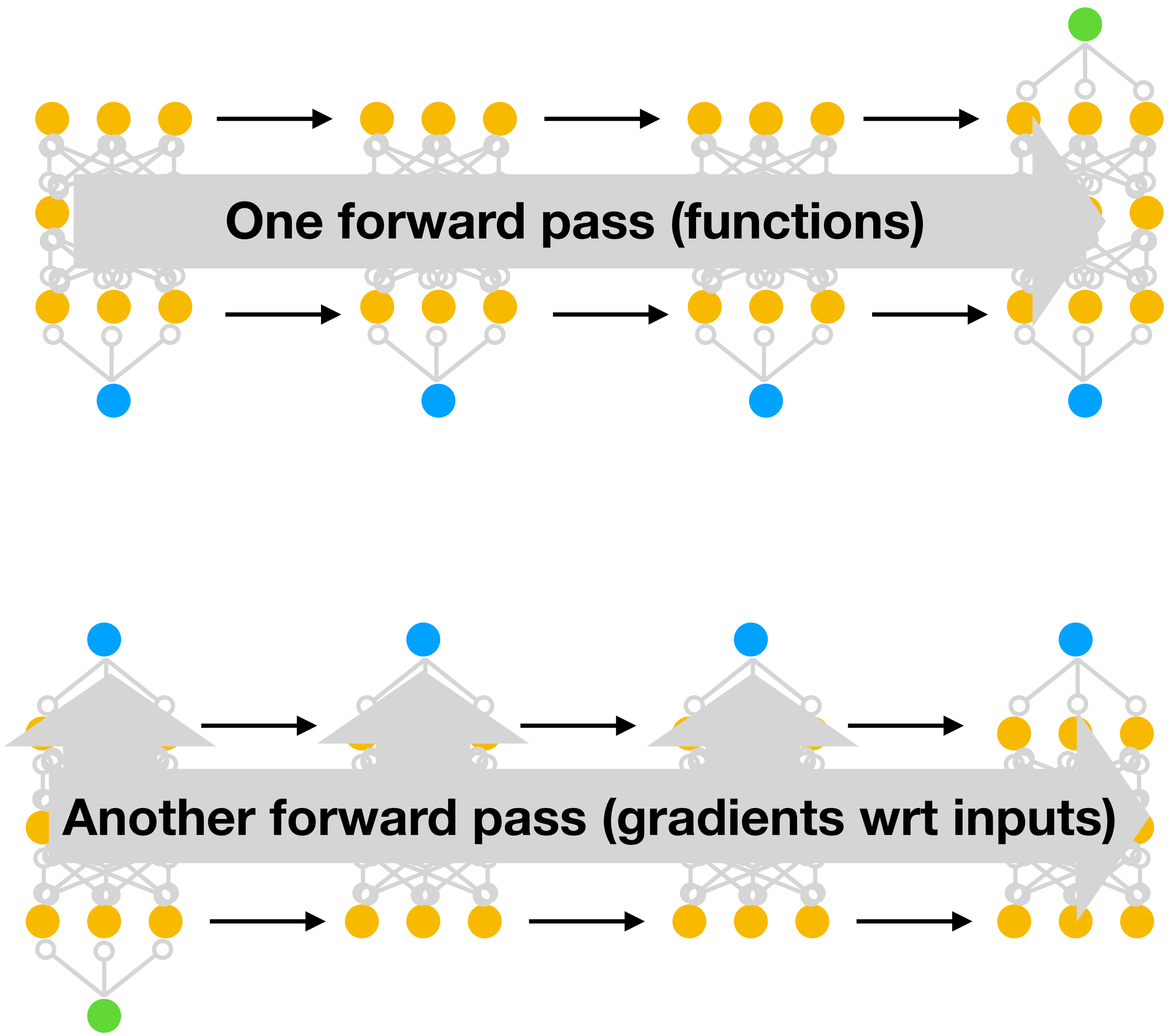
feed through
delete some bits

Training FFN



Training RNNs

- Back-propagation through time



A “real-life” RNN

A “real-life” RNN

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 (word “embedding”, word piece model)

Sentence -> trajectory

How does it work?

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

Sentence -> trajectory

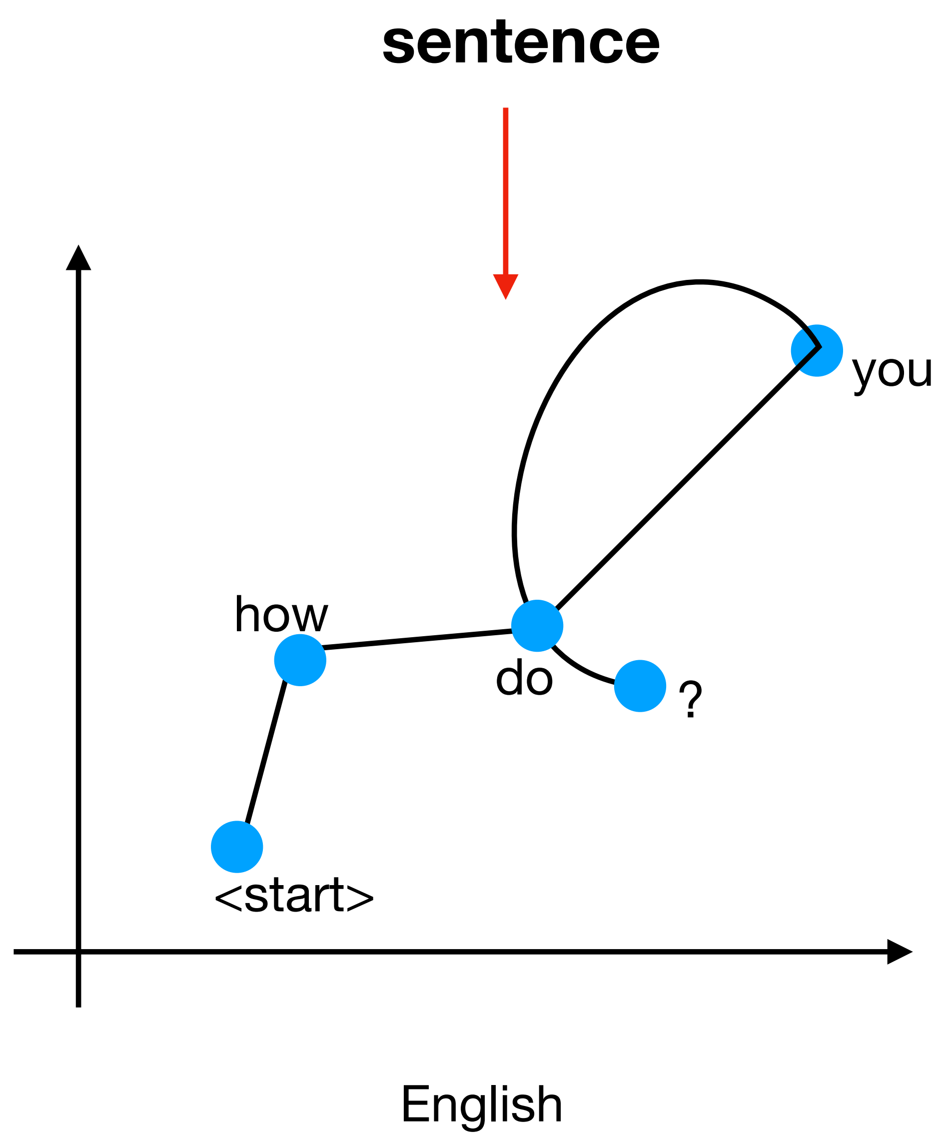
Two separate networks (encoding/decoding)

How does it work?

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

Sentence -> trajectory

Two separate networks (encoding/decoding)

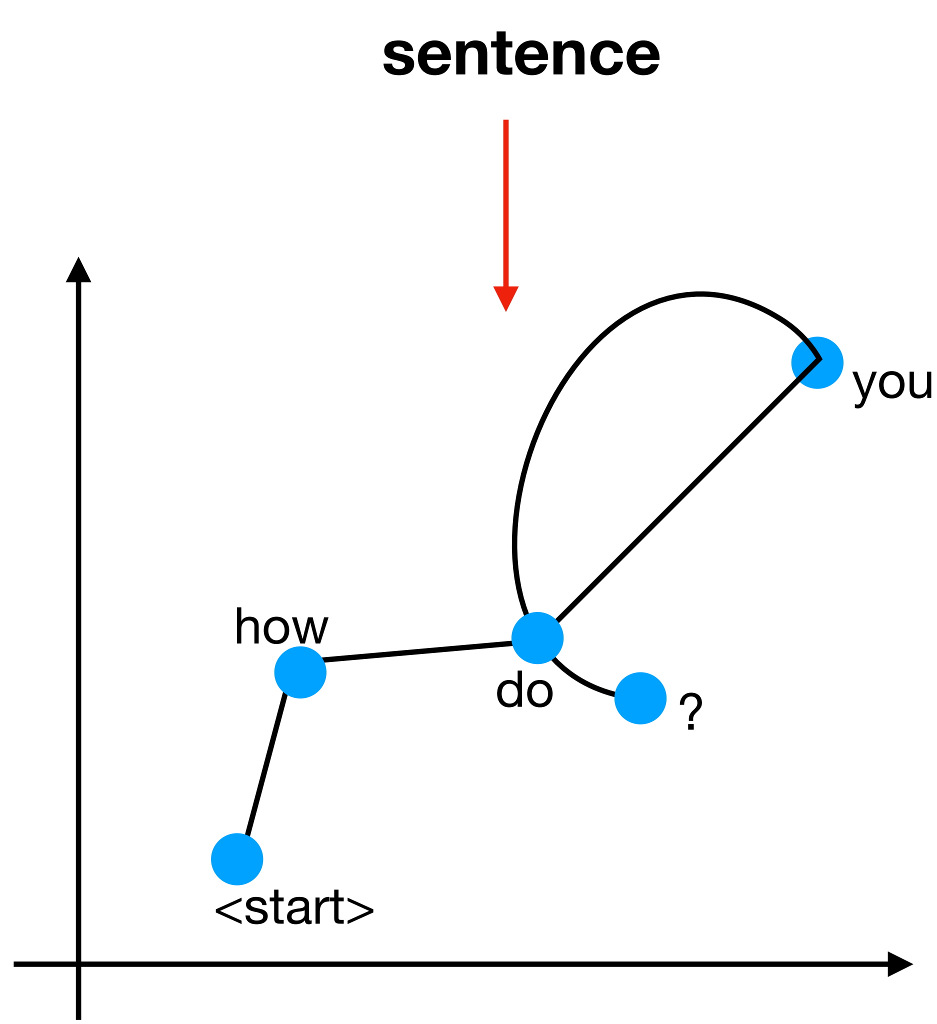


How does it work?

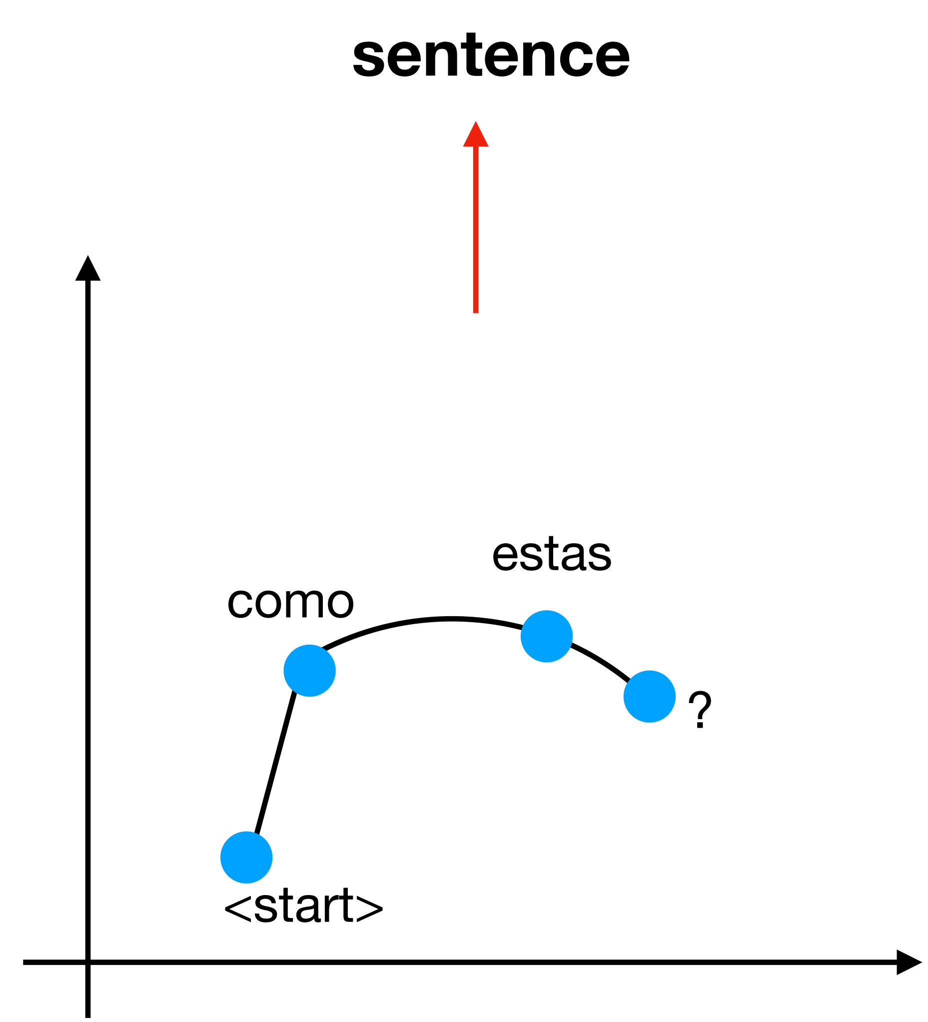
Word -> vector of numbers (word “embedding”, word piece model)

Sentence -> trajectory

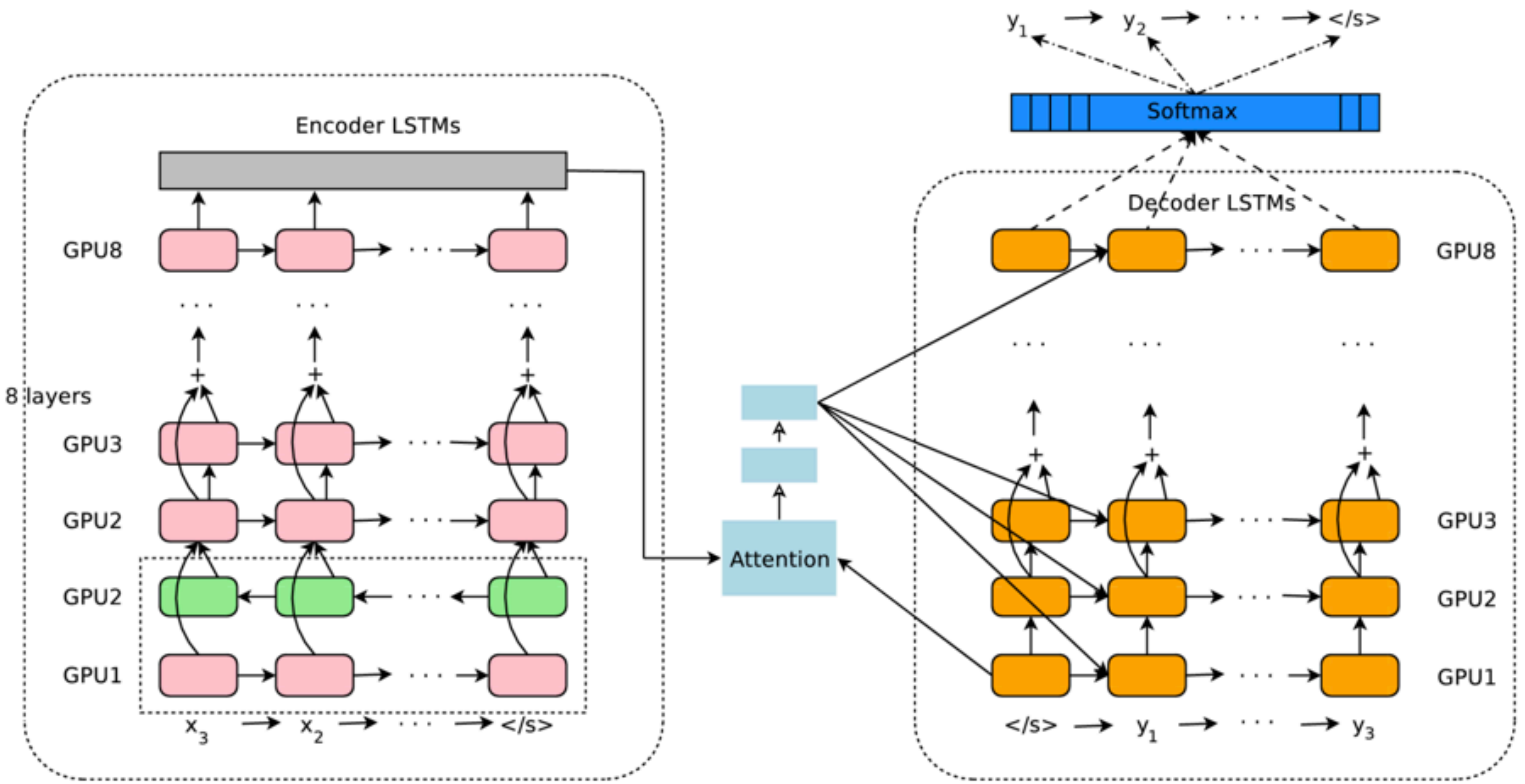
Two separate networks (encoding/decoding)



English



Spanish



https://arxiv.org/pdf/1609.08144.pdf

Time to do hands-on playing with RNNs