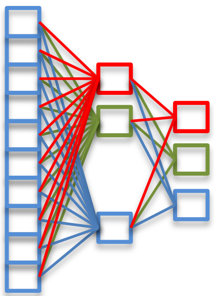
Lecture: Deep Learning and Differential Programming

4.1 Recurrent Neural Networks

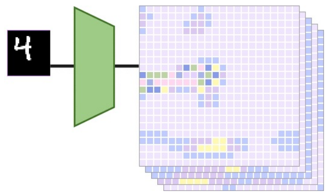
https://liris.cnrs.fr/christian.wolf/teaching

INSA LYON Christian Wolf

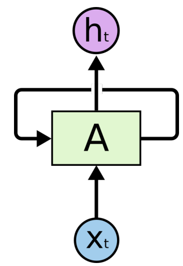
The Deep Toolbox



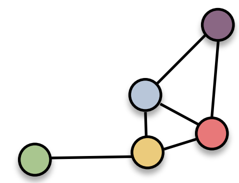
MLP



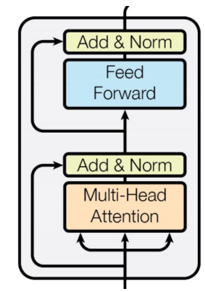
CNN /
Convolutions



RNN /
Recurrence



GN, GCN /
Graphs, geometry



Transformers /
Self-attention

What do I know about the data and the task?

Nothing (vector space)

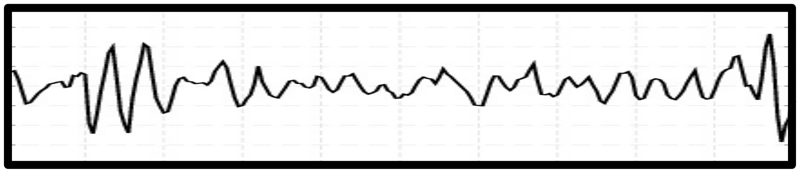
Translation equivariance

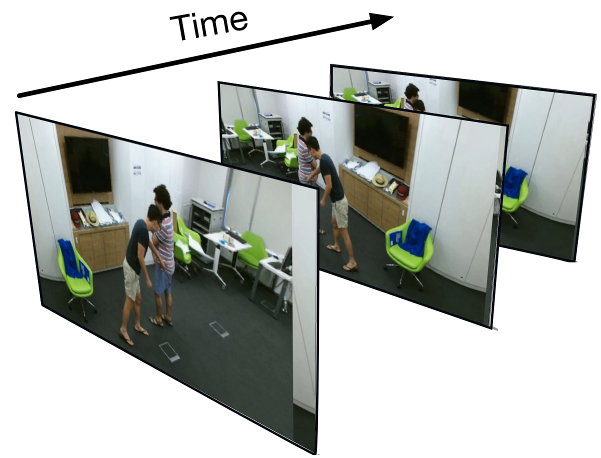
Sequential data, Markov property

Graph structured data

Permutation equivariance

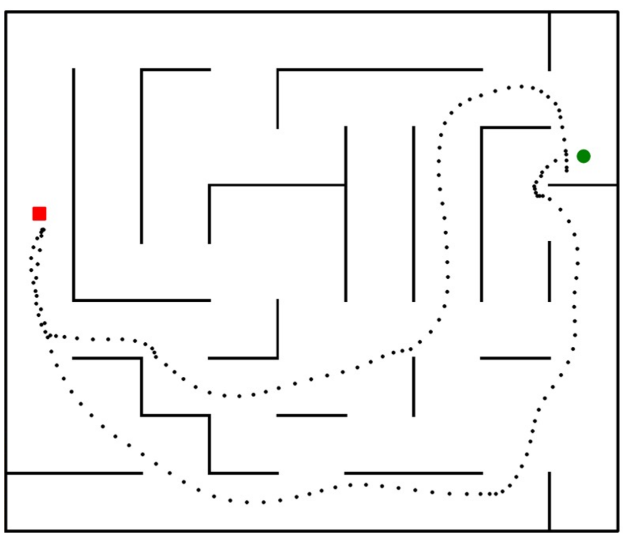
Problem: dealing with sequences



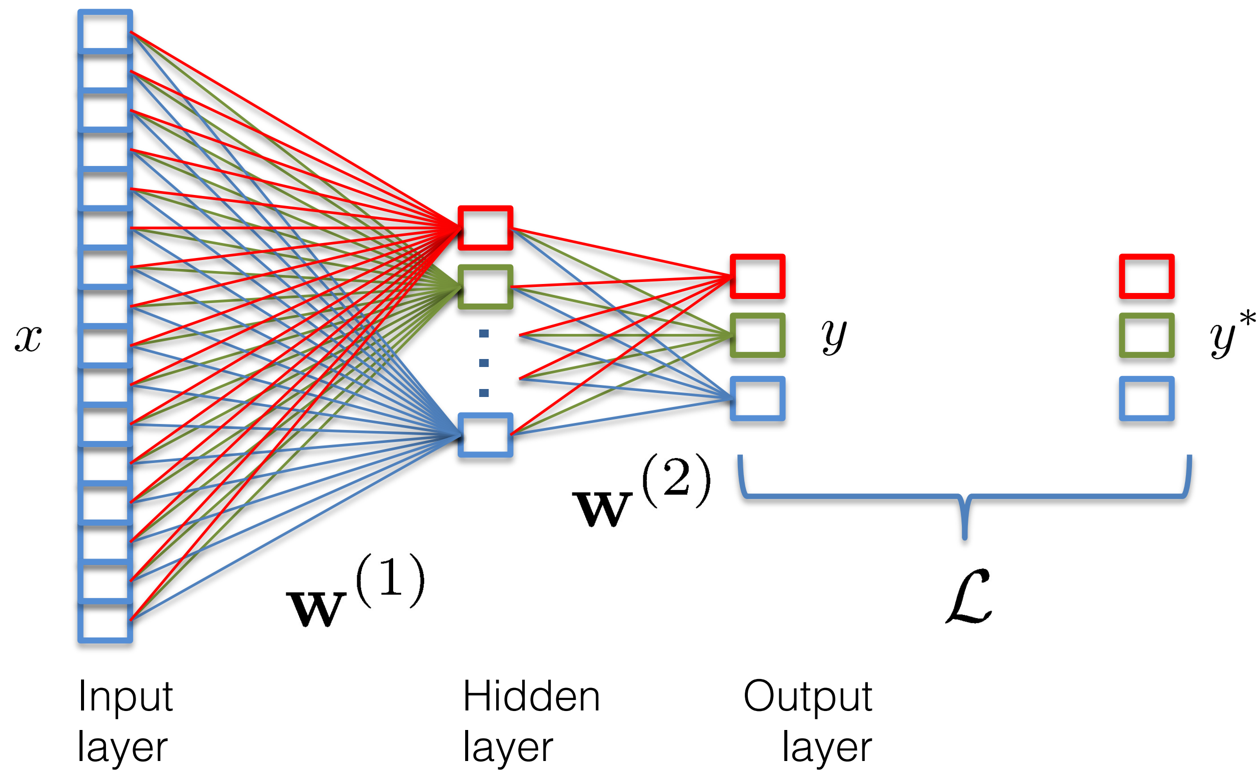


Toutes les familles heureuses se ressemblent. Chaque famille malheureuse, au contraire, l'est à sa façon.

Happy families are all alike. Every unhappy family is unhappy in its own way.

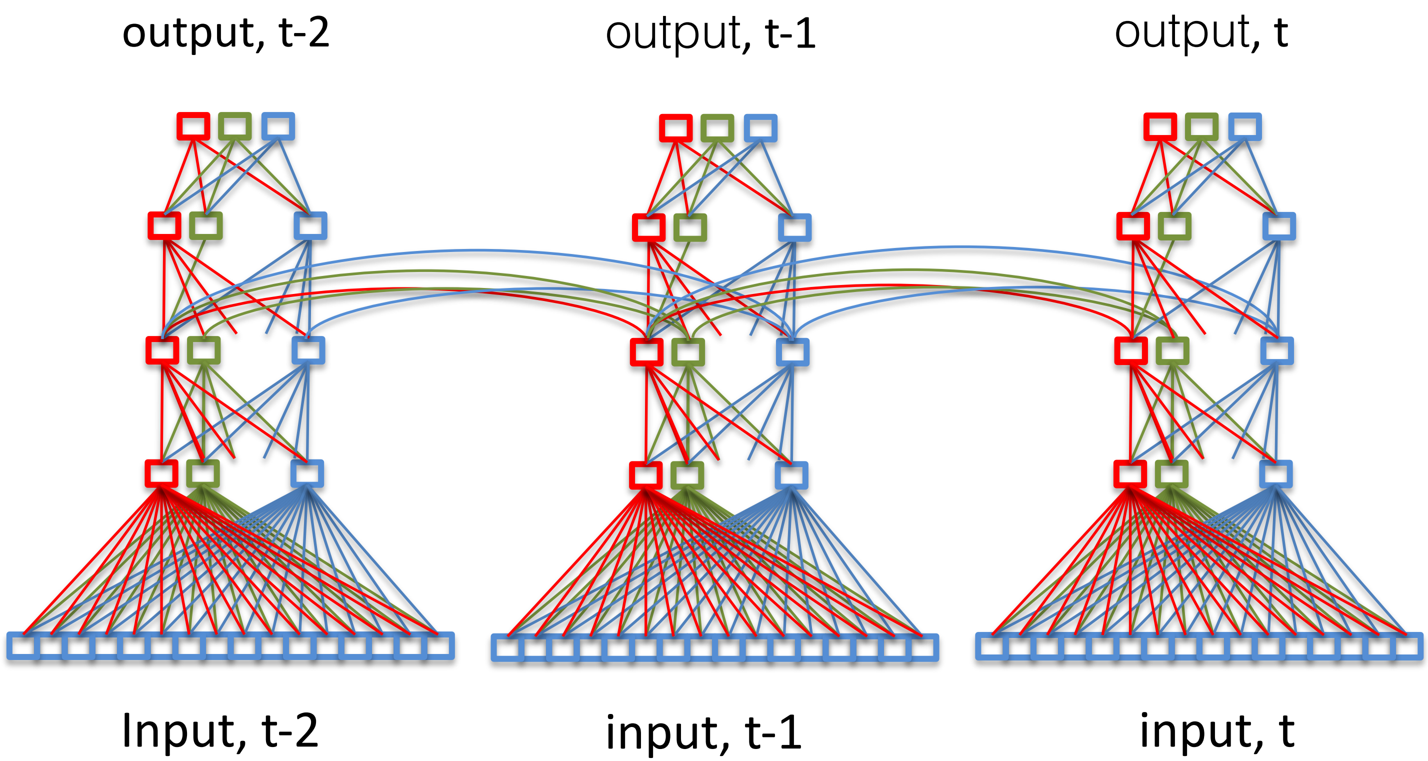


Feed Forward Networks



Prediction: <u>feed-forward</u> computation in a DAG (directed asyclic graph).

Recurrent neural networks



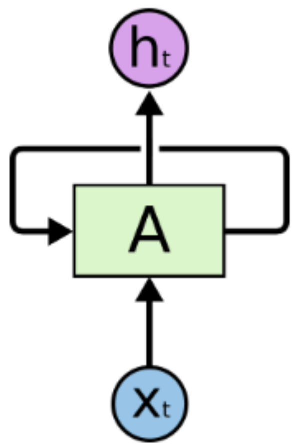
A shout-out ...

... to Chris Olah's excellent blog post on RNN and LSTM networks, which completely dominates lectures on this topic. The following couple of slides are based on his excellent drawings.

`Chris Olah, https://colah.github.io/posts/2015-08-Understanding-LSTMs/`

Recurrent Neural Networks (RNNs)

As feed-forward networks, Recurrent Neural Networks (RNNs) predict some output from a given input.
However, they also pass information over time, from instant $(t-1)$ to (t):

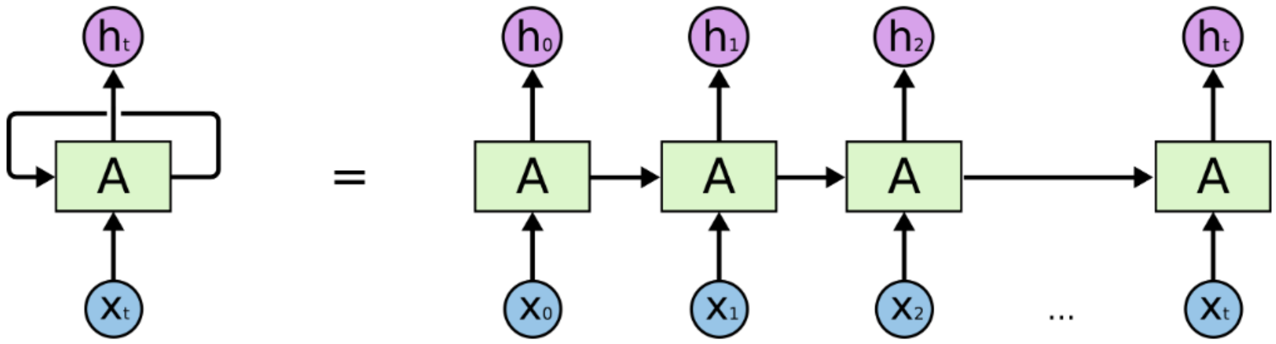


Here, we write h_t for the output, since these networks can be stacked into multiple layers, i.e. h_t is input into a new layer.

`Chris Olah, https://colah.github.io/posts/2015-08-Understanding-LSTMs/`

Recurrent Neural Networks (RNNs)

A more intuitive view of an RNN is to unroll it over time:



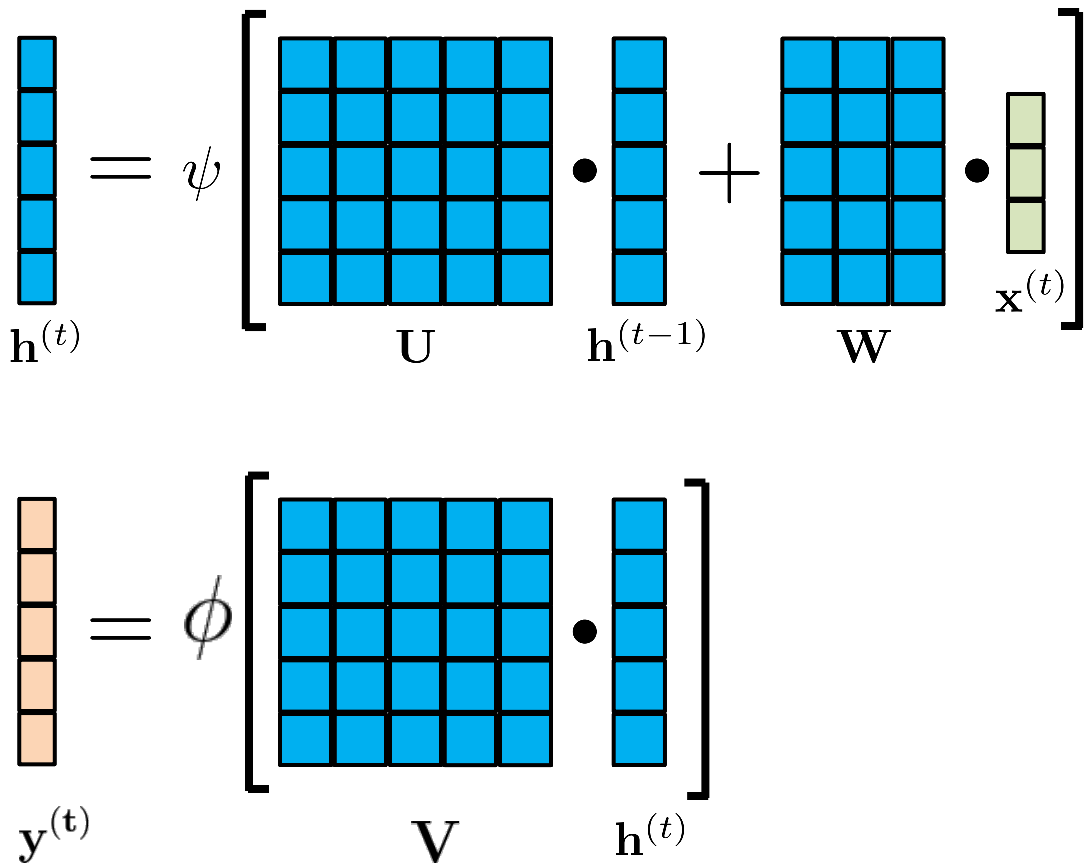
The update equations are:

$$h_t = \sigma \left(h_{t-1} \cdot W_h + x_t \cdot W_x \right)$$

$\Rightarrow$ two weight matrices: one for the recurrent connections over time, one for the input connections.
σ is an activation function.

Chris Olah, https://colah.github.io/posts/2015-08-Understanding-LSTMs/

From state h to output y



Toy example with handcrafted parameters

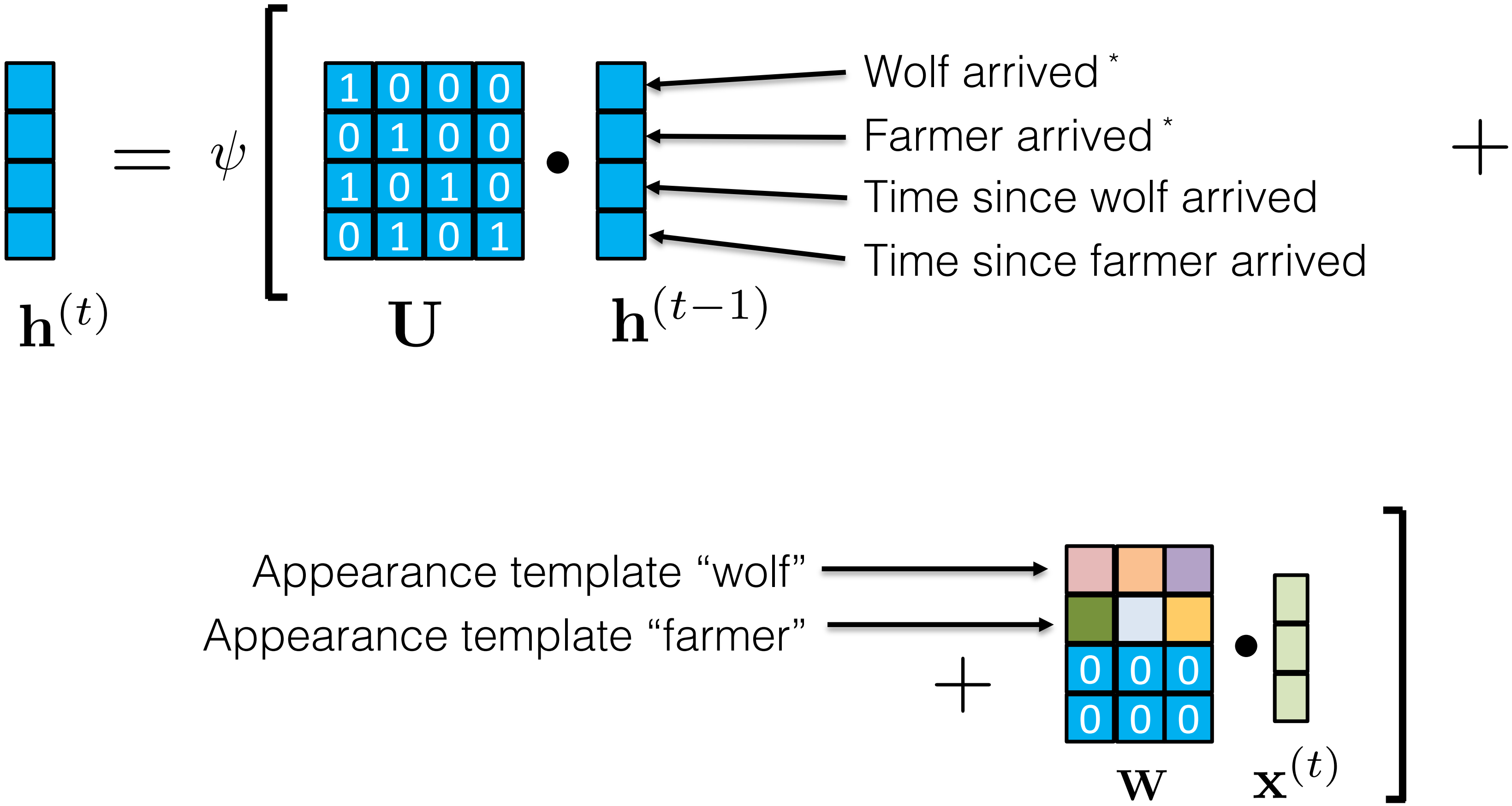
In a sequential problem, we surveil a farm and watch for the appearance of objects. At each instant t we oberve a vector  which can indicate the apparence of

- a wolf 
- a farmer 

The objective is to output an estimate of danger, i.e.

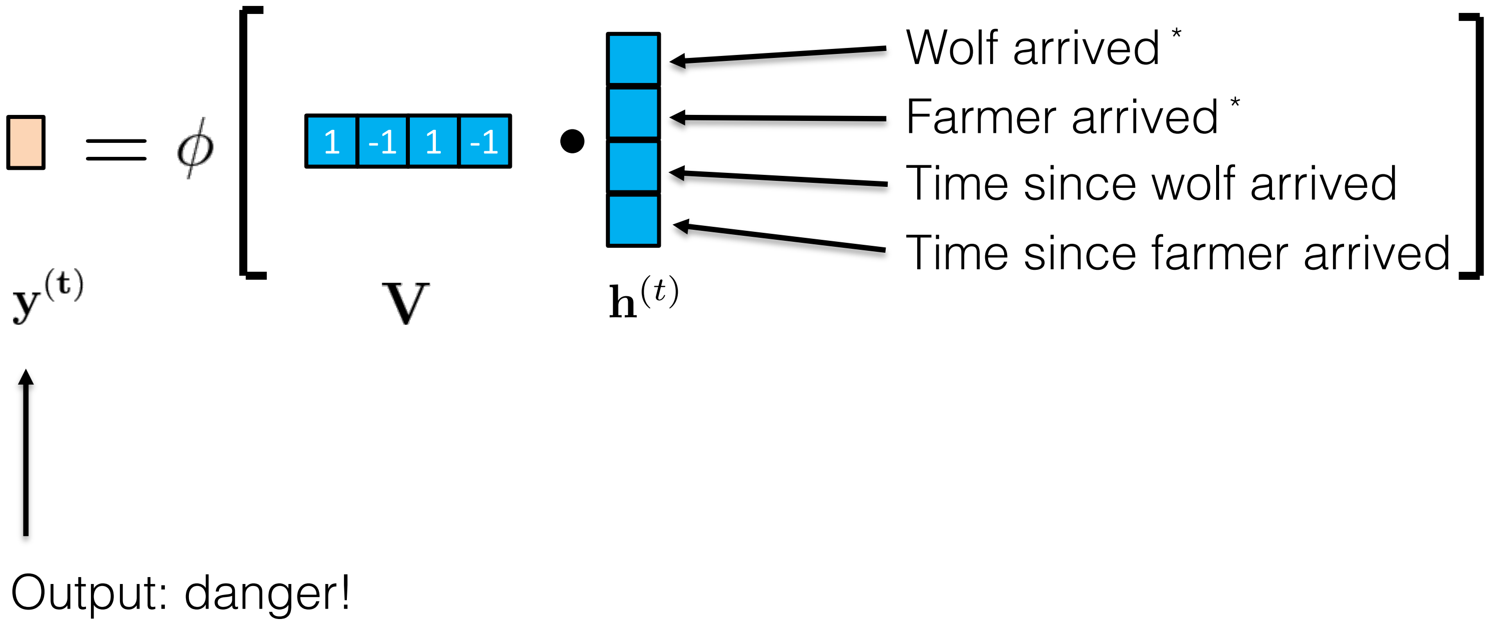
- presence of the wolf w/o the farmer, or
- presence of both, arrival of the wolf before the farmer.

Toy example with handcrafted parameters



*Appropriate activation functions required to normalize state values

Toy example with handcrafted parameters



RNN training & problems

RNNs are trained with backpropagation through time (BPTT): the graph is unrolled, and the loss derived w.r.t. the parameters of all different time instants.
Standard vanilla RNNs are difficult to train and suffer from shortcomings:

- Vanishing gradients: small gradients vanish over long time ranges.
- Exploding gradients: high gradients explode over long ranges.
- Lack of long-term dependency handling: short-term updates between individual time instances dominate.

Solution: gating mechanisms (LSTM and GRU networks).

Gating

Problem: hidden state is updated at <u>**each**</u> time step

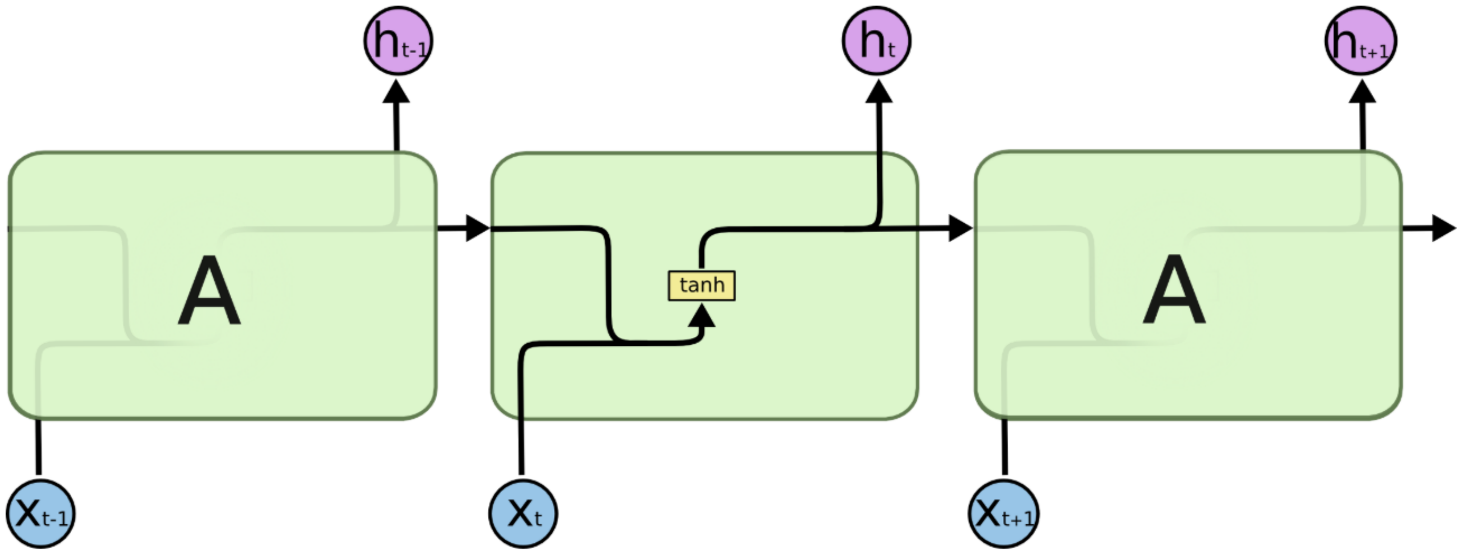
Solution: at each instant, for each state value, decide how much to

- input
- forget
- output

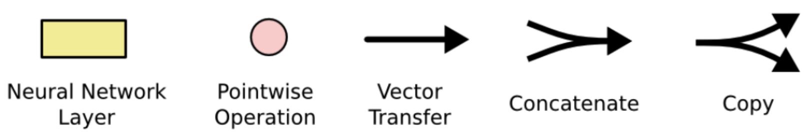
- These decisions are also learned

LSTM Networks: walk through

We start by illustrating a vanilla RNN in a new way:



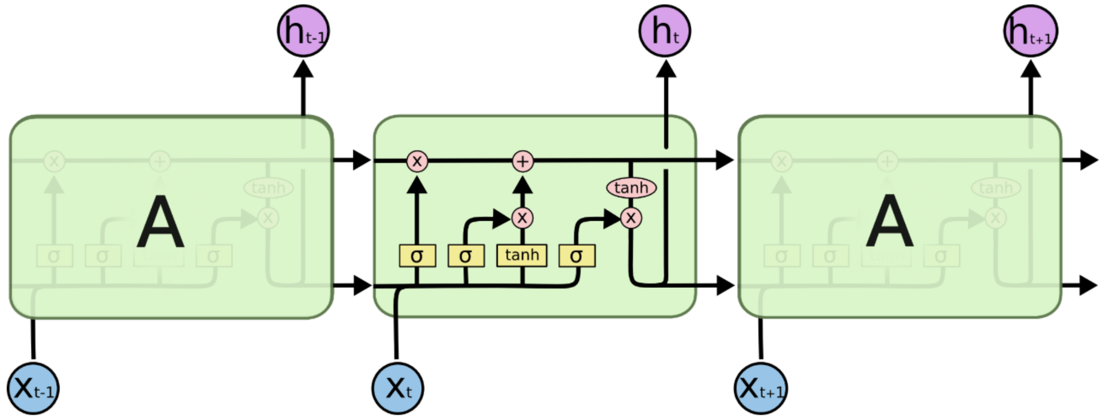
where:



Chris Olah, https://colah.github.io/posts/2015-08-Understanding-LSTMs/

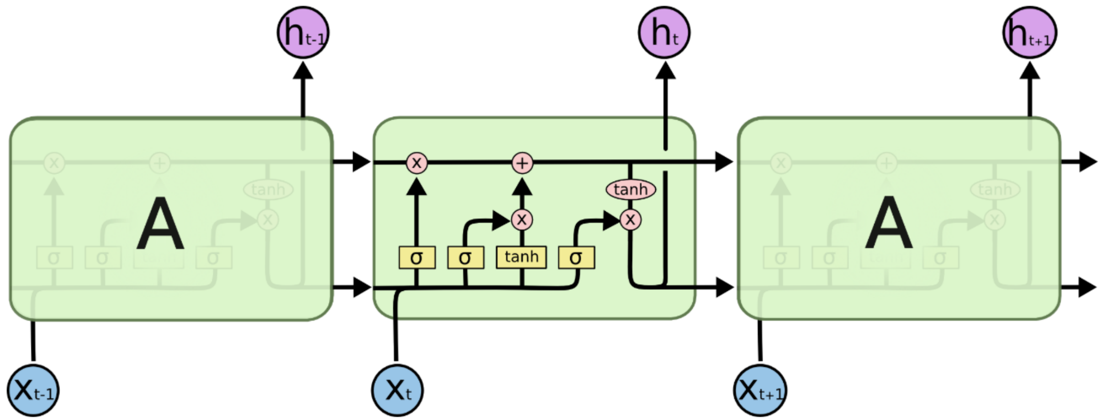
LSTM Networks

LSTM (=Long-short term Memory) networks use gating mechanisms, which handle information flow in a fully trainable way.



LSTM Networks

LSTM (=Long-short term Memory) networks use gating mechanisms, which handle information flow in a fully trainable way.



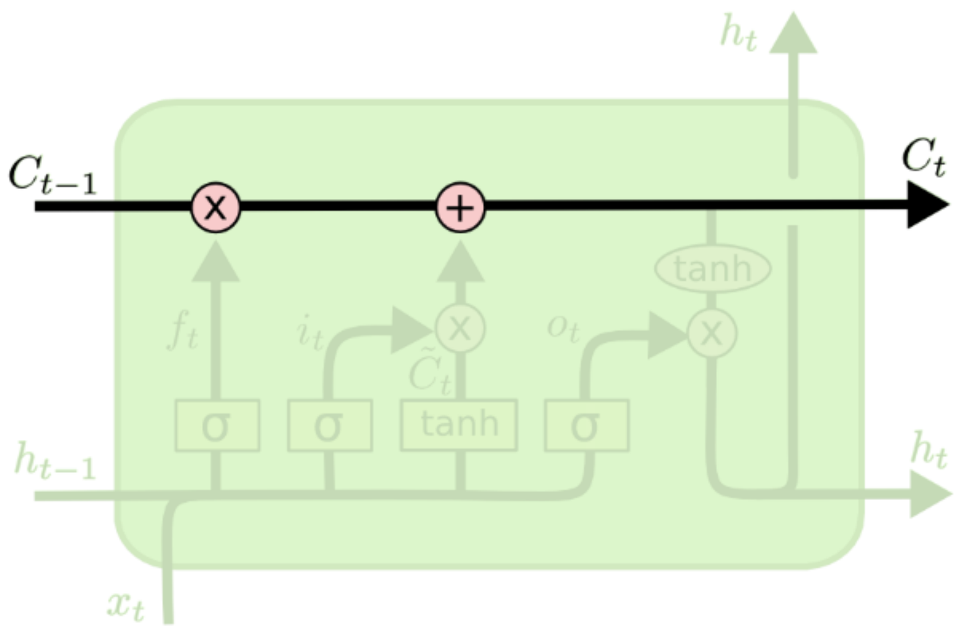
$$
\begin{aligned}
f_t &= \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right) \\
i_t &= \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right) \\
\tilde{C}_t &= \tanh\left(W_C \cdot [h_{t-1}, x_t] + b_C\right) \\
C_t &= f_t * C_{t-1} + i_t * \tilde{C}_t \\
o_t &= \sigma\left(W_o\, [h_{t-1}, x_t] + b_o\right) \\
h_t &= o_t * \tanh\left(C_t\right)
\end{aligned}
$$

`Chris Olah, https://colah.github.io/posts/2015-08-Understanding-LSTMs/`

LSTM Networks: walk through

An LSTM has two different memory representations:

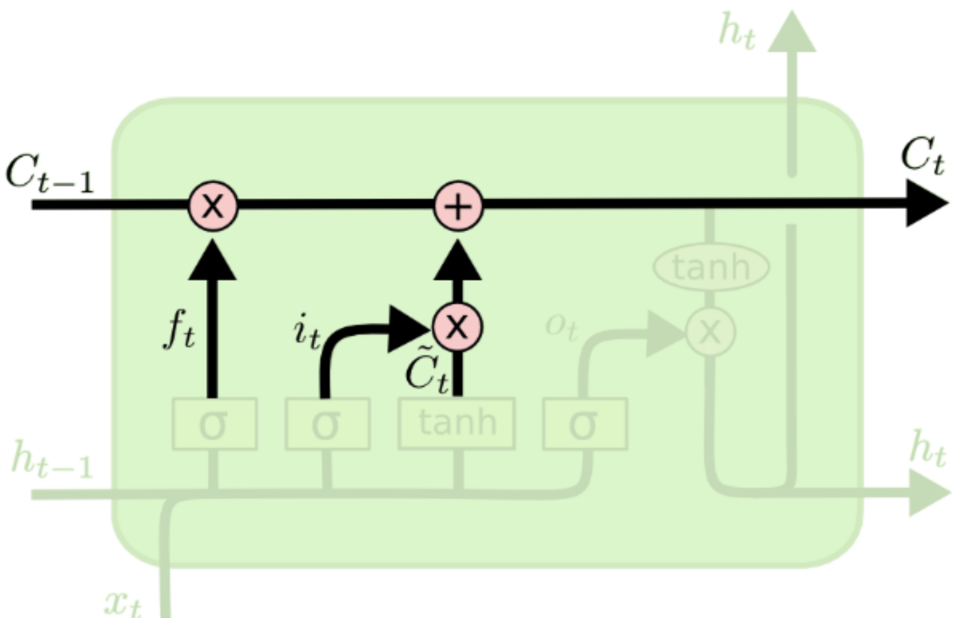
- A cell state C,
- the hidden state h.



Chris Olah, https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM Networks: walk through

The new cell state C_t is a linear combination of the old cell state C_{t-1} and some new updated information $\tilde{C}_t$ (described later):



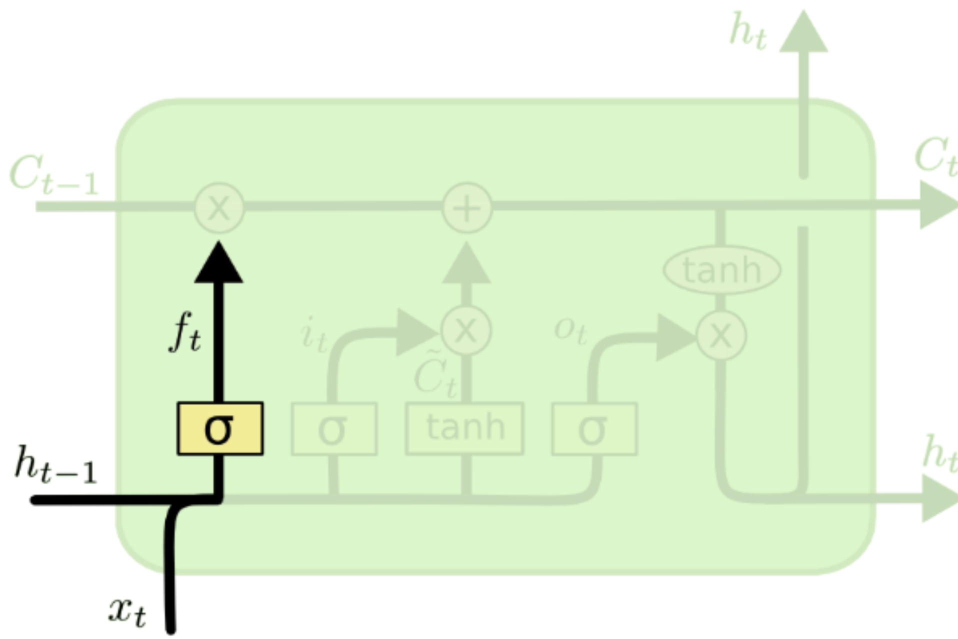
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

f_t and i_t are "**gates**" (trainable functions), which govern the information flow (based on the hidden state h_t).

Chris Olah, https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM Networks: walk through

The forget gate controls how much of the old cell state is forgotten or passed through:



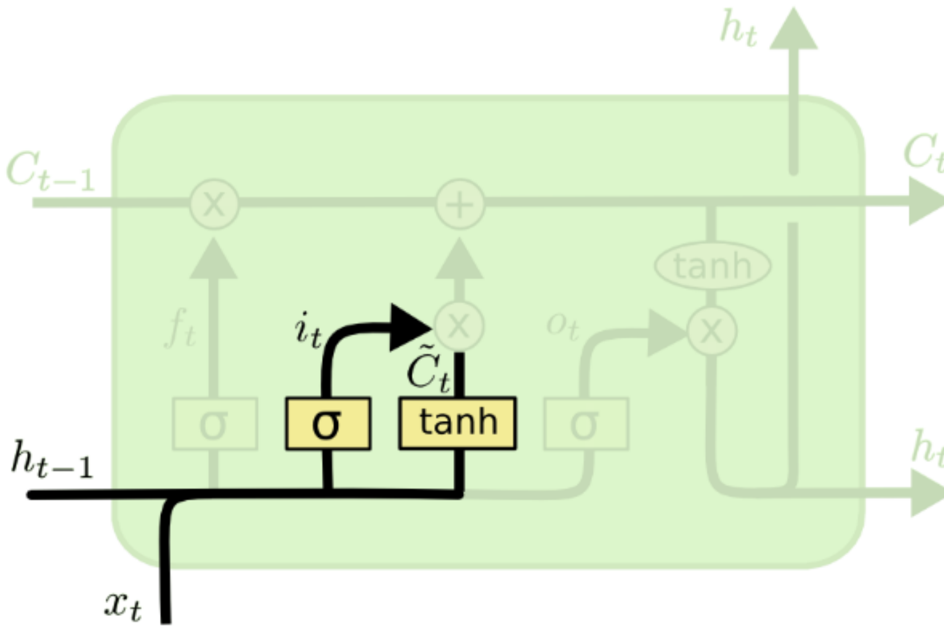
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$$

Chris Olah, https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM Networks: walk through

The input gate controls how much new information is passed over to the cell state.

The new information is predicted from the hidden state h:

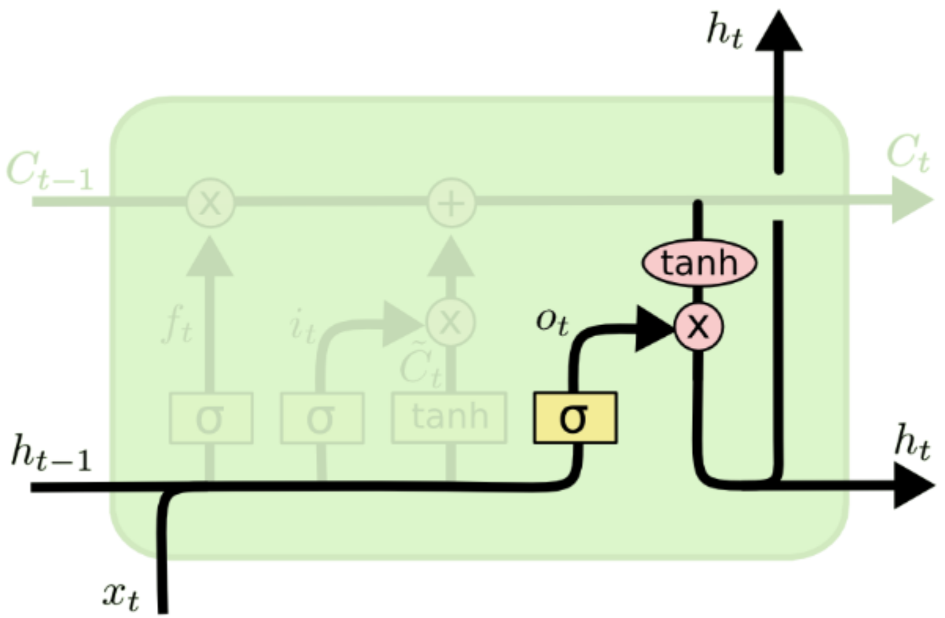


$$
\begin{aligned}
i_t &= \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right) \\
\tilde{C}_t &= \tanh\left(W_C \cdot [h_{t-1}, x_t] + b_C\right)
\end{aligned}
$$

Chris Olah, https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM Networks: walk through

The output gate controls, how the information from the cell state is translated into the hidden state h:

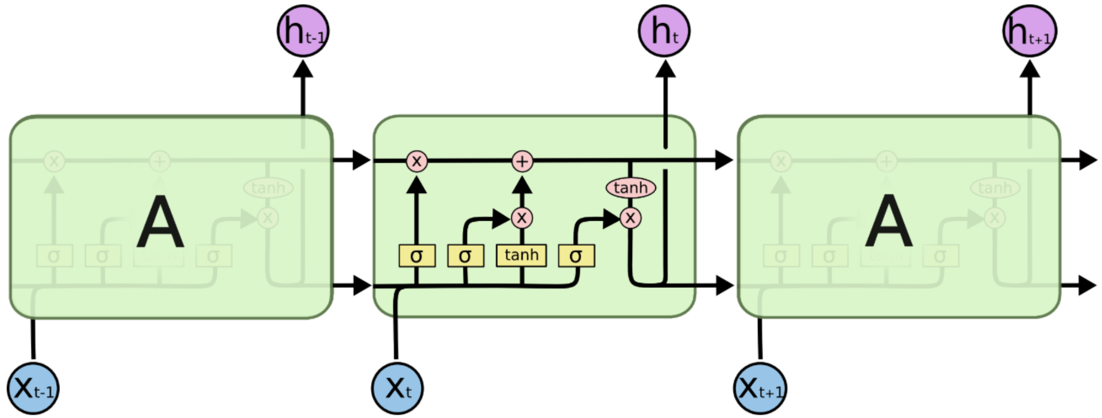


$$
\begin{aligned}
o_t &= \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right) \\
h_t &= o_t * \tanh\left(C_t\right)
\end{aligned}
$$

Chris Olah, https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM Networks

The full model, again:



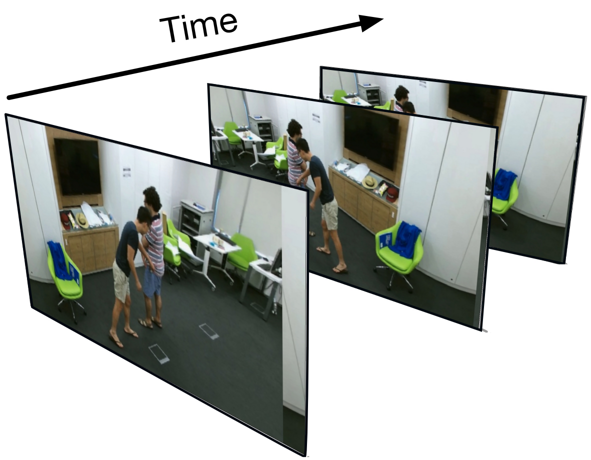
$$
\begin{aligned}
f_t &= \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right) \\
i_t &= \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right) \\
\tilde{C}_t &= \tanh\left(W_C \cdot [h_{t-1}, x_t] + b_C\right) \\
C_t &= f_t * C_{t-1} + i_t * \tilde{C}_t \\
o_t &= \sigma\left(W_o\, [h_{t-1}, x_t] + b_o\right) \\
h_t &= o_t * \tanh\left(C_t\right)
\end{aligned}
$$

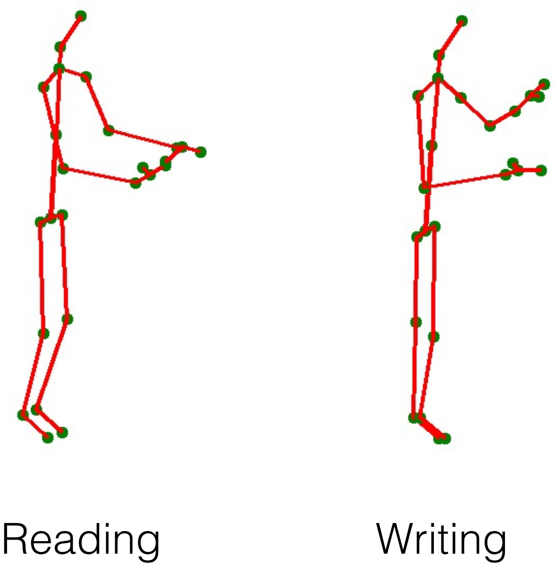
Chris Olah, https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Example applictions: Motion

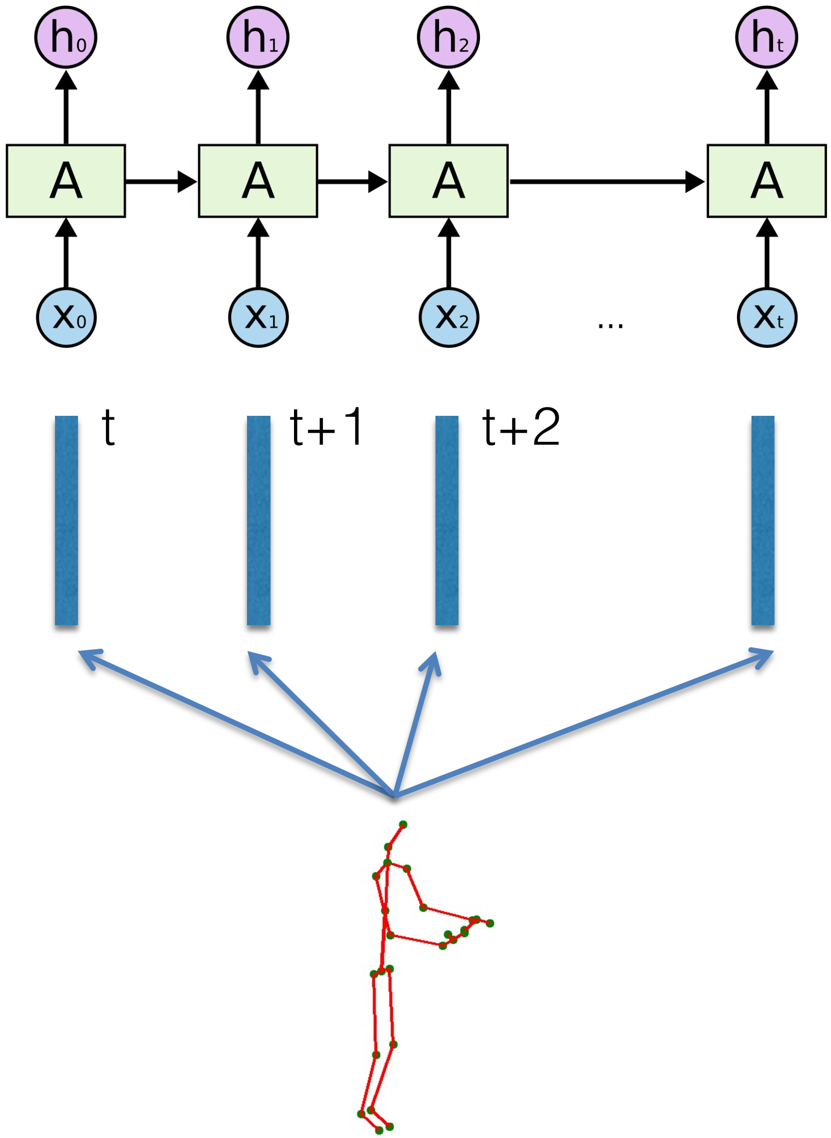
[Figure: Scott Eaton]

Example application: activity recognition





Example application: activity recognition

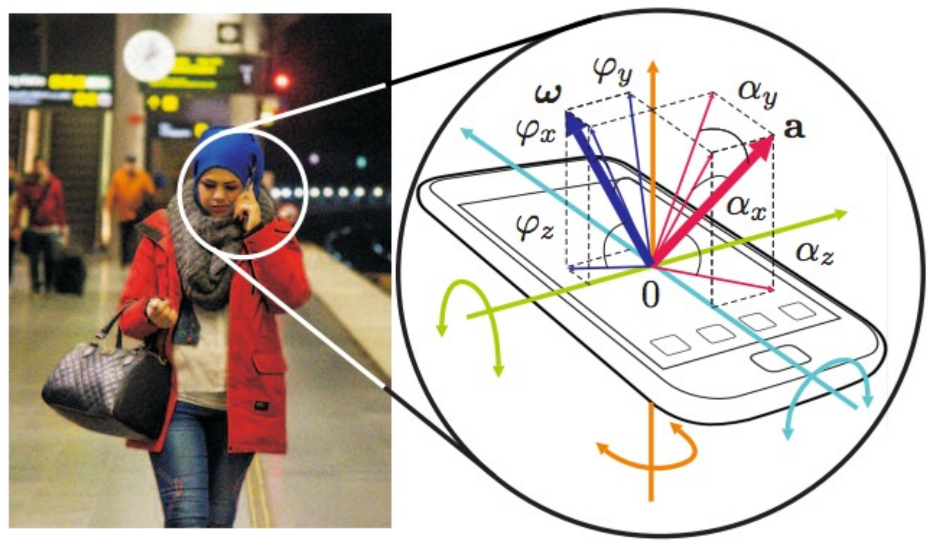


Entering PINs on smartphones is painful

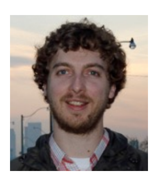
Automatically authenticate smartphone users given their behavior (=interaction style). Shut off phone when theft is detected.

Project "Abacus » (Google)

- 1500 volunteers, 1500 Nexus 5 smartphones
- Several months of natural daily usage, 27.6 TB of data
- Multiple sensors: camera, touchscreen, GPS, bluetooth, wifi, cell antenna, inertial, magnetometer
- <u>This work</u>: inertial sensors only, recorded at 200Hz



Work of Natalia Neverova
Phd student at LIRIS



With Graham W. Taylor,
University of Guelph, Canada

Google

[Neverova, Wolf, Lacey, Fridmann, Chandra, Barbello, Taylor, *IEEE Access 2016*]

Experimental results

Feature extraction

Model	Accuracy, %	# parameters
ST Convnet	37.13	6 102 137
LT Convnet	56.46	6 102 137
Conv-RNN	64.57	1 960 295
Conv-CWRNN	68.83	1 964 254
Conv-LSTM	68.92	1 965 403
Conv-DCWRNN	**69.41**	**1 964 254**

Biometric framework (GMM)

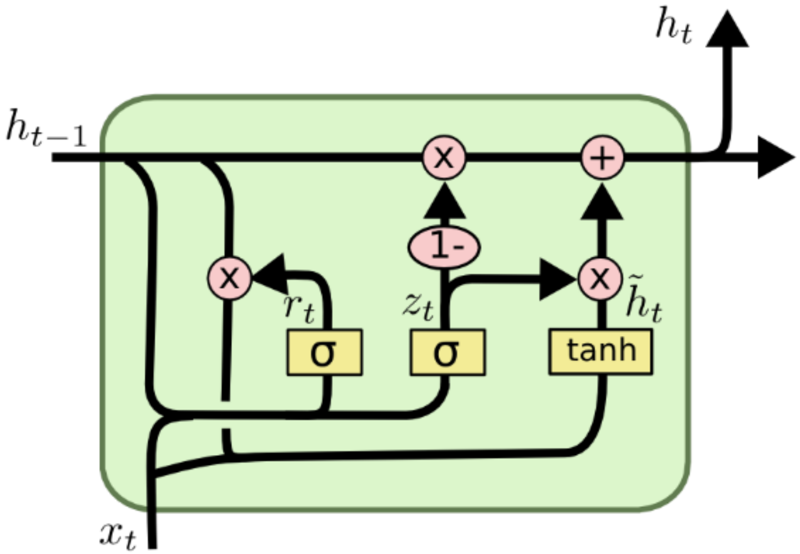
Model	EER, %	HTER, %
Raw features	36.21	42.17
ST Convnet	32.44	34.89
LT Convnet	28.15	29.01
Conv-RNN	22.32	22.49
Conv-CWRNN	21.52	21.92
Conv-LSTM	21.13	21.41
Conv-DCWRNN	20.01	20.52
Conv-DCWRNN, zt-norm	**18.17**	**19.29**
Conv-DCWRNN (per device)	*15.84*	*16.13*
Conv-DCWRNN (per session)	*8.82*	*9.37*

[Neverova, Wolf, Lacey, Fridmann, Chandra, Barbello, Taylor, *IEEE Access 2016*]

Appendix:
Other RNN Variants

Variant: GRU (Gated Recurrent Unit)

GRUs are simpler: they merge cell state and h, and merge gates:



$$
\begin{aligned}
z_t &= \sigma\left(W_z \cdot [h_{t-1}, x_t]\right) \\
r_t &= \sigma\left(W_r \cdot [h_{t-1}, x_t]\right) \\
\tilde{h}_t &= \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right) \\
h_t &= (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t
\end{aligned}
$$

`Chris Olah, https://colah.github.io/posts/2015-08-Understanding-LSTMs/`

What's the Frequency Kenneth?

All presented variants model a single time frame (update from one time step to the next).

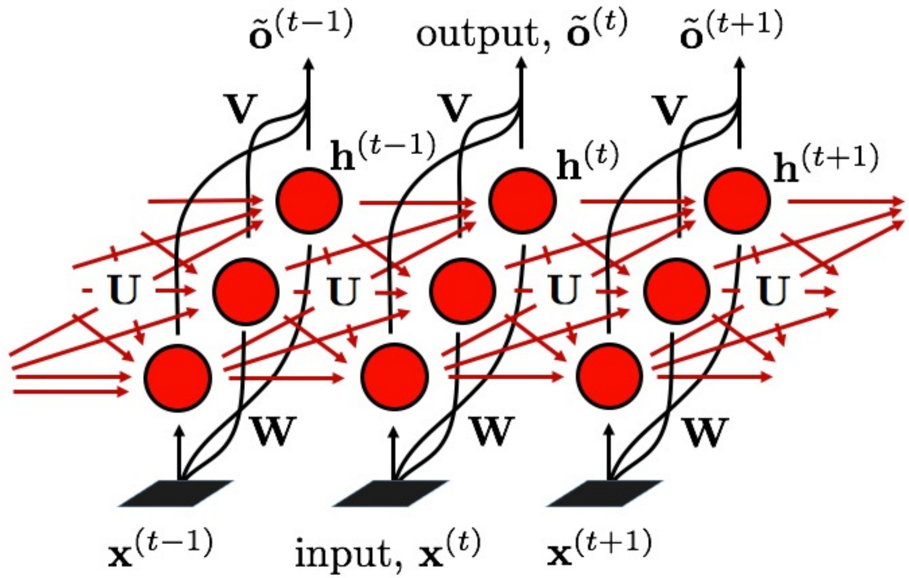
In signal processing, we like to decompose a signal into frequency components.

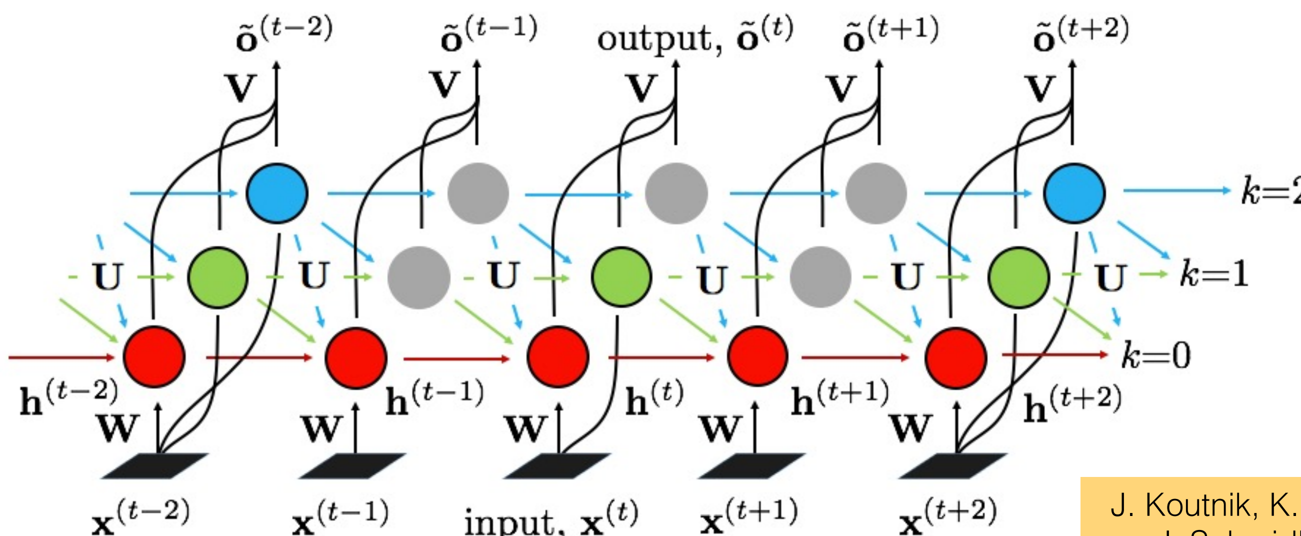
Can we create RNNs with multiple frequency bands?

Clock Work RNNs do this.

J. Koutnik, K. Greff, F. Gomez, and J. Schmidhuber, ICML 2014.

Vanilla RNN vs. Clockwork RNN





J. Koutnik, K. Greff, F. Gomez, and J. Schmidhuber, ICML 2014.

CWRNN problems

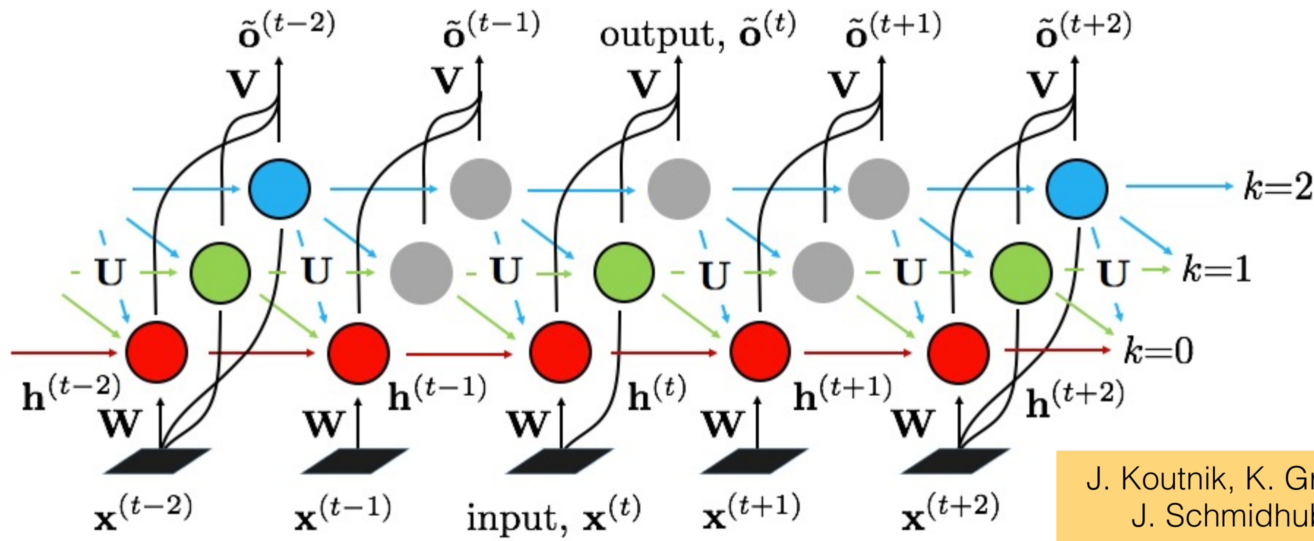
CWRNNs suffer from a couple of problems

- They are not shift-invariant (output depends on time t).
- Lower frequency weights tend to overfit since they are updated slower.

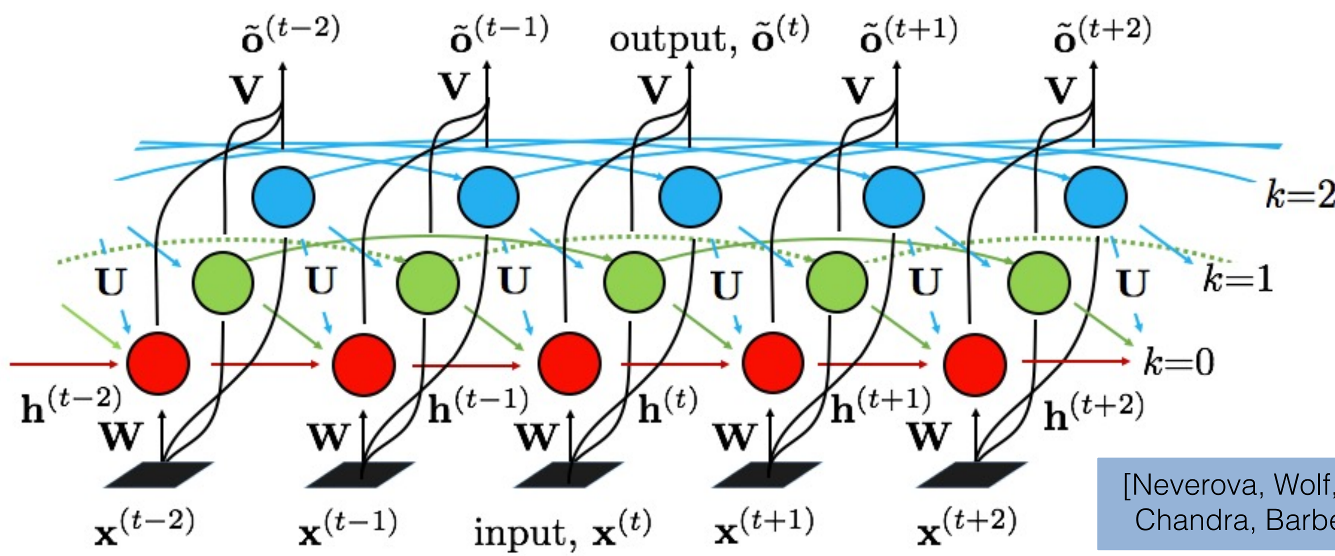
Dense CWRNNs solve this problem.

[Neverova, Wolf, Lacey, Fridmann, Chandra, Barbello, Taylor, *2016*]

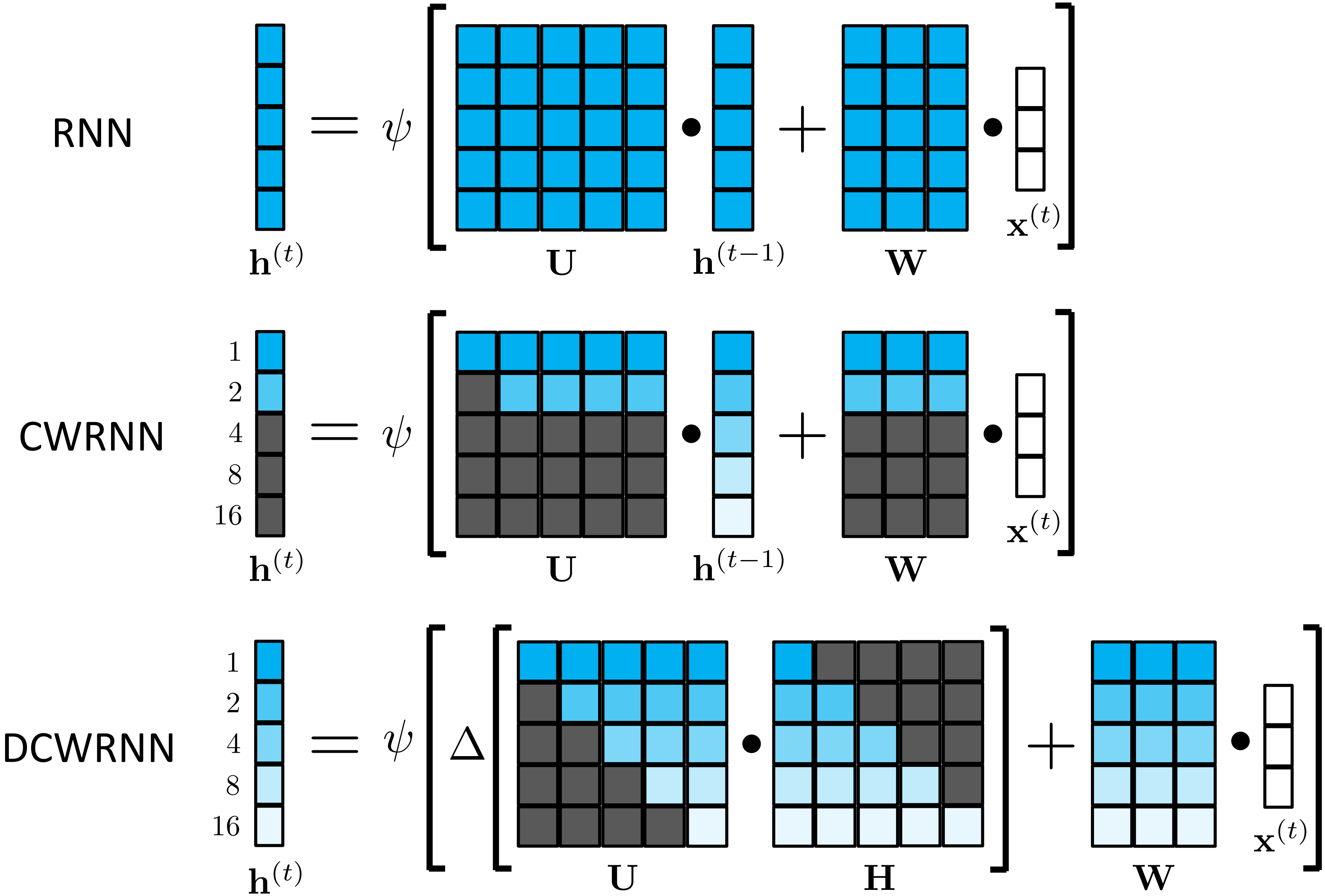
Vanilla RNN vs. Clockwork RNN



J. Koutnik, K. Greff, F. Gomez, and J. Schmidhuber, ICML 2014.

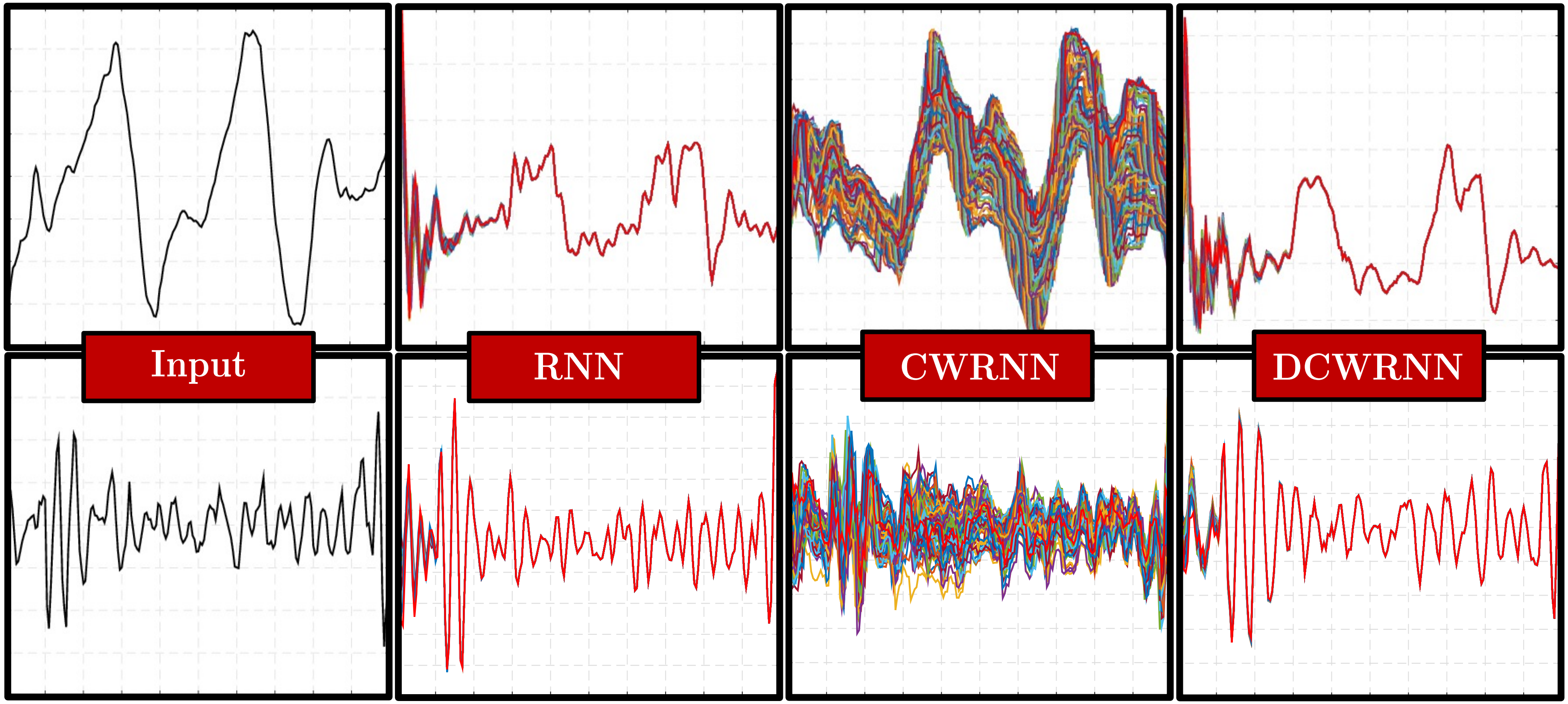


[Neverova, Wolf, Lacey, Fridmann, Chandra, Barbello, Taylor, *2016*]



[Neverova, Wolf, Lacey, Fridmann, Chandra, Barbello, Taylor, *2016*]

On shift-invariance



[Neverova, Wolf, Lacey, Fridmann, Chandra, Barbello, Taylor, *2016*]