Deep learning

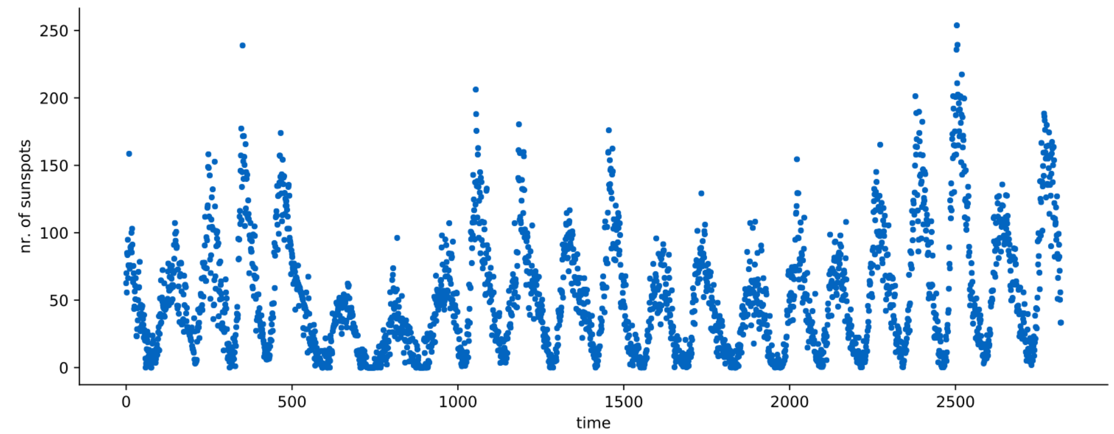
Sequential modelling / RNNs

Today’s program

- 14:00-14:15 Recap
- 14:15-15:00 Machine learning tasks: regression / classification
- 15:00-15:45 Hands-on: multiclass Fashion MNIST
- 15:45-16:15 Break
- 16:15-16:45 Optimizers, regularization techniques
- 16:45-17:30 Hands-on: Regularization techniques on F-MNIST
- **17:30-18:00 Analyzing sequential data, RNNs**
- 18:00-19:00 Diner
- 19:00-19:45 Hands-on: Predicting future temperatures with an RNN
- 19:45-20:15 Types of RNNs: LSTM, GRU
- 20:15-21:00 Hands-on: creating sequences, temperature prediction with GRU-based RNN
- Time left: Improving RNNs: regularization, stacking, stateful and bi-directional RNNs
- Time left: Hands-on: Improved RNNs on temperature prediction

Sequential data

- Data is sequential when the data has some order.
- The whole dataset can consist of a single order (sunspots) or many individual orders (sentences).



the cat sat on the mat .
the book is open .

Sequential data

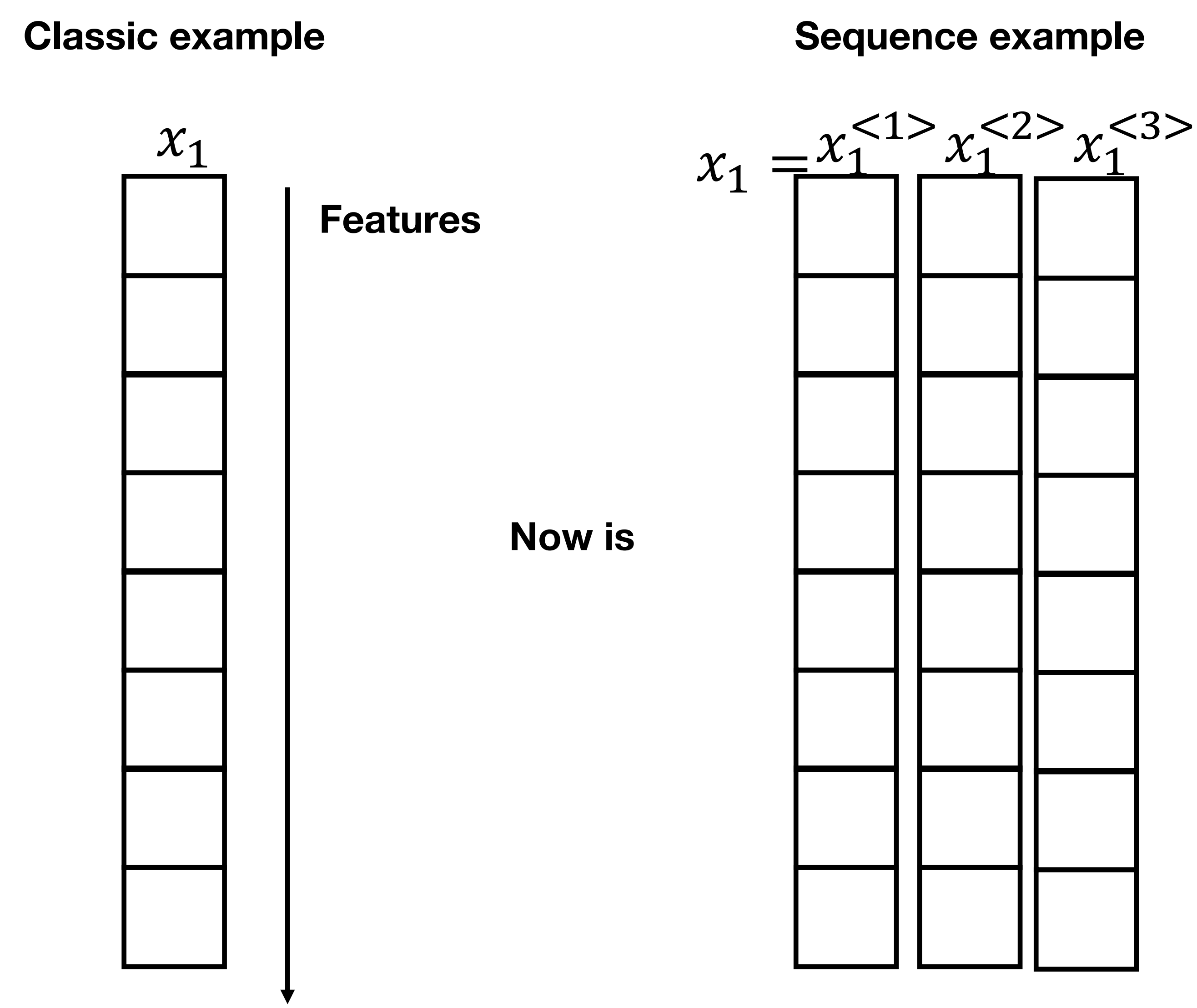
in deep learning

- Machine translation — "Hoi, hoe gaat het?" → "Hi, how are you?"
- Speech recognition —  → "Hi, how are you?"
- Music generation — Some style / nothing → 
- Sentiment classification — "Hoi, hoe gaat het?" → 
- Video activity recognition — 

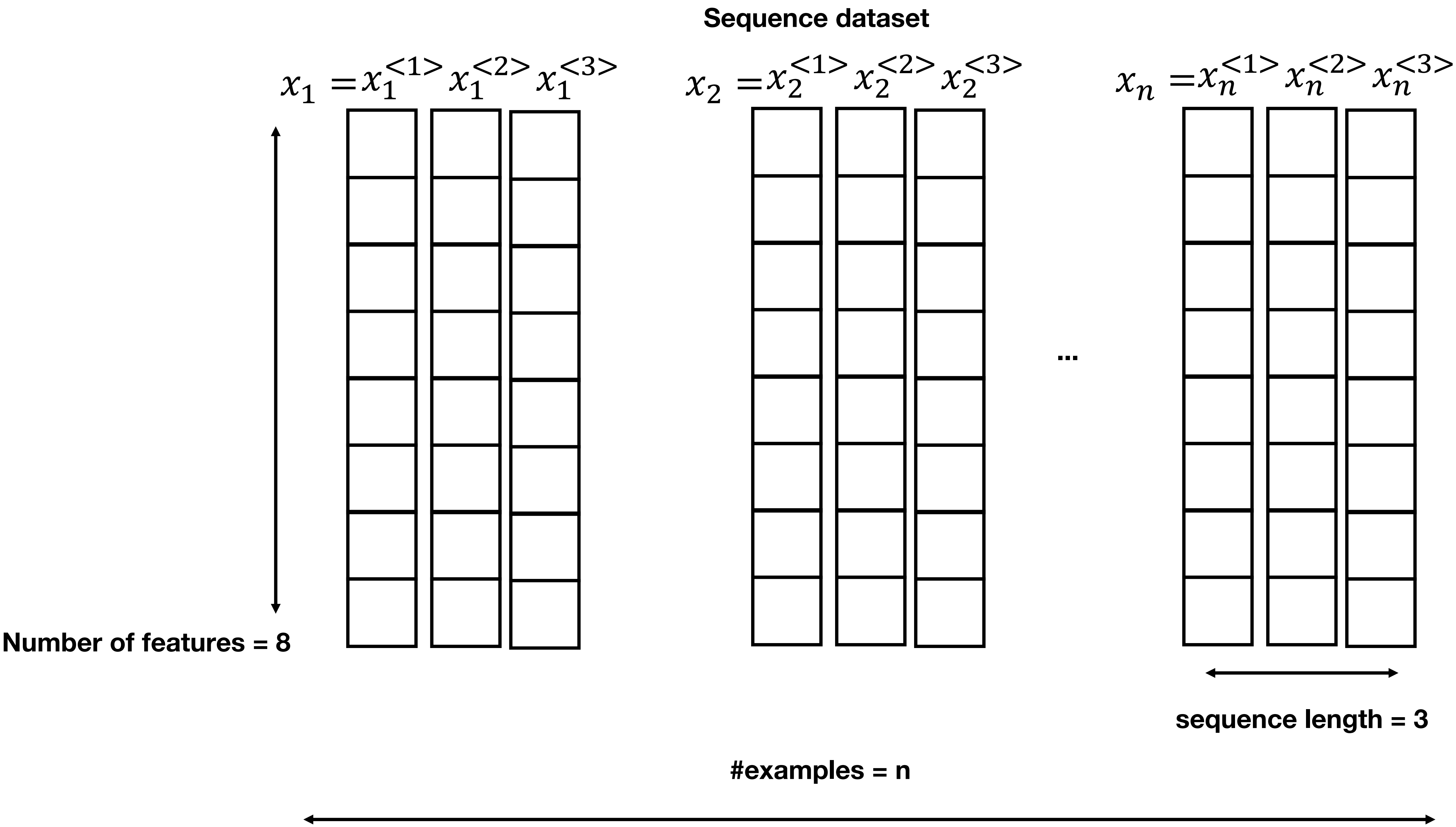
→ Running
- ...

Sequential data

- In previous lectures our data have been made up from a single example.
- A single example can have many features.
    - Temperature, air pressure, etc.
- Now each example is made from a single sequence.
    - "Hi, hoe gaat het?"
- Each sequence has many examples.
    - "Hi", "hoe", "gaat", "het"
- That is, in each iteration we process a single sequence, many examples.

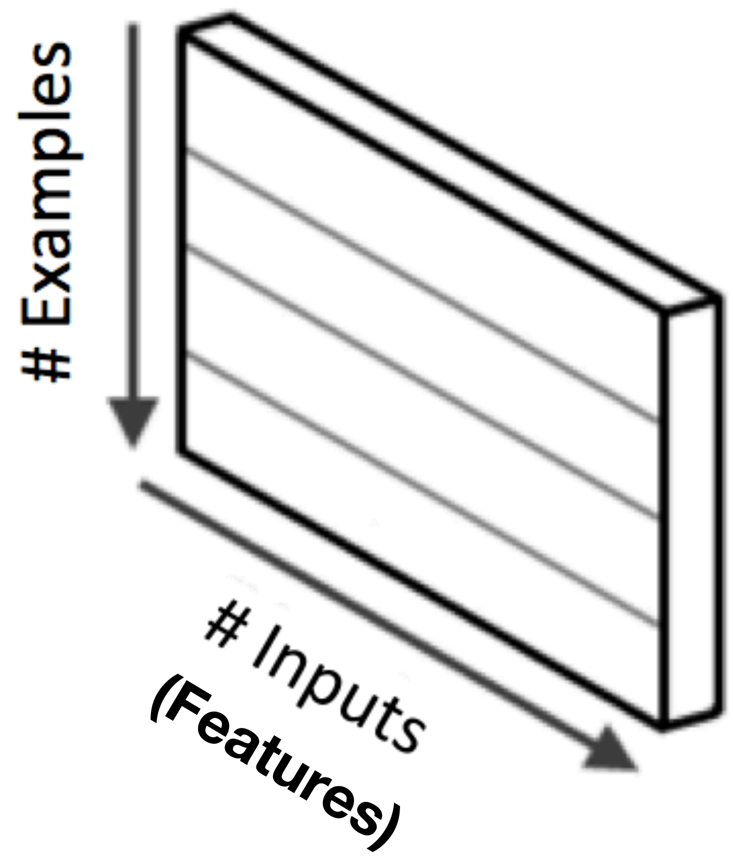


Sequential data



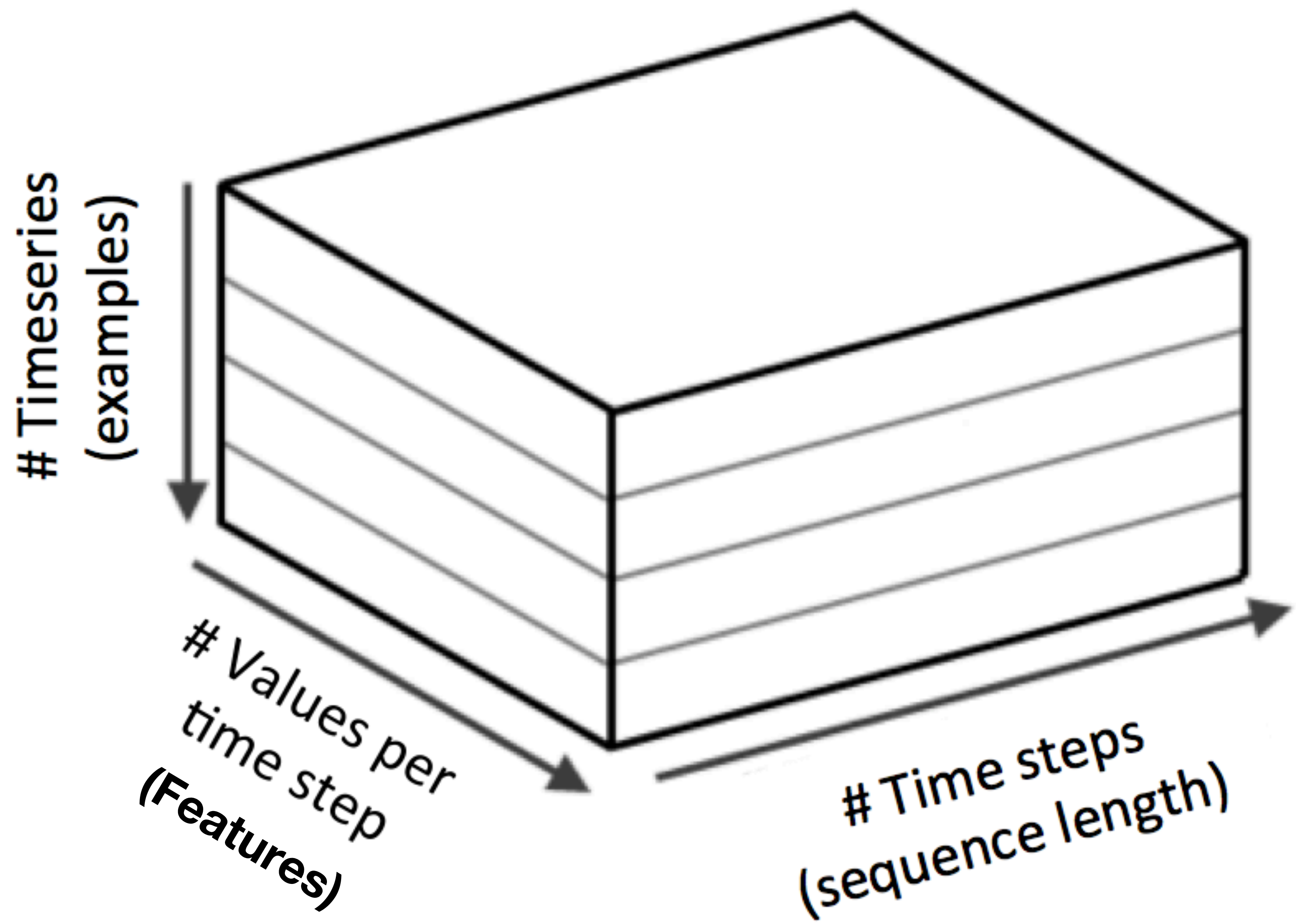
Data = (examples, sequence_length, features)

Feed-Forward Network Data



Data = (examples, features)

Recurrent Network Data

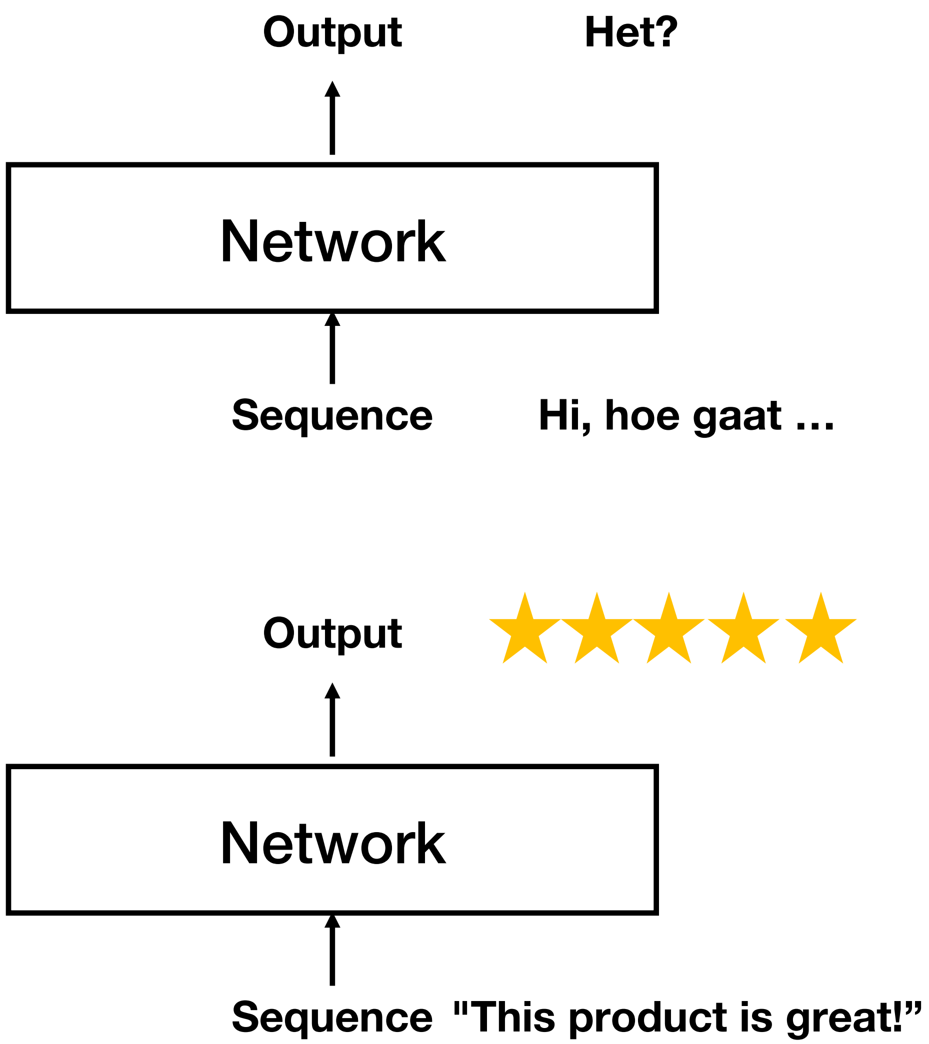


Data = (examples, sequence_length, features)

Source: https://www.oreilly.com/library/view/deep-learning/9781491924570/ch04.html

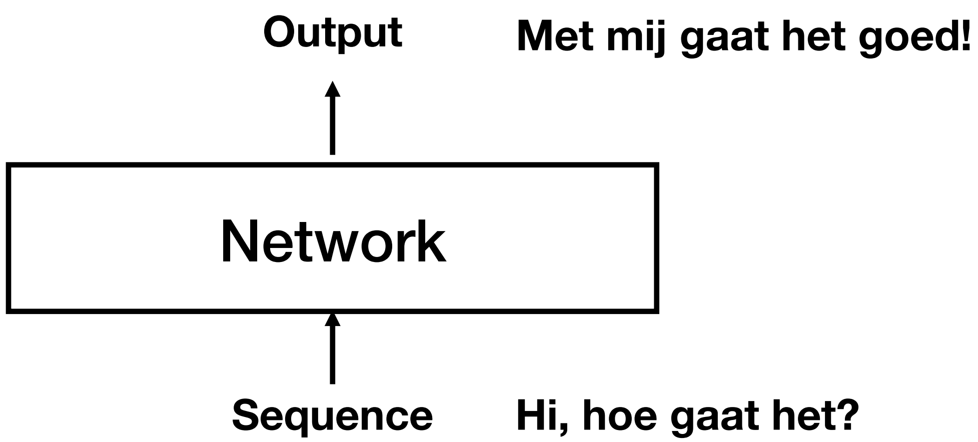
Which task?

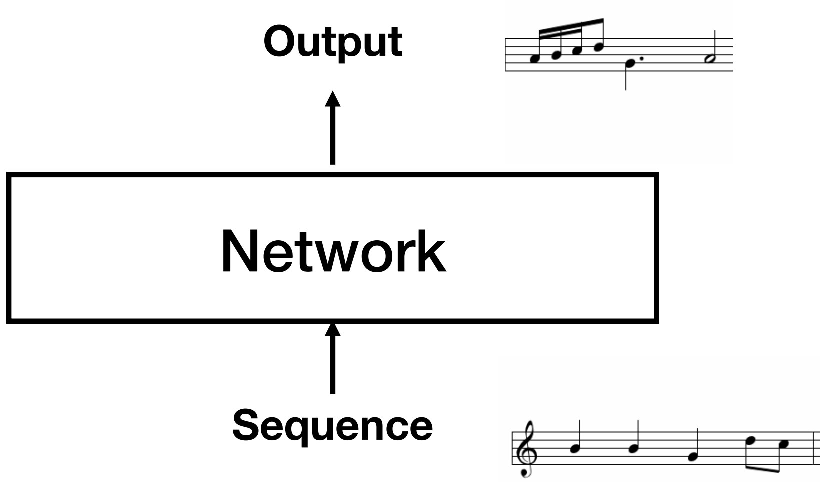
- Many-to-one
  - Regression
    - Tomorrows weather
    - Next word
  - Classification
    - Is this a question?
    - Assign rating based on review



Which task?

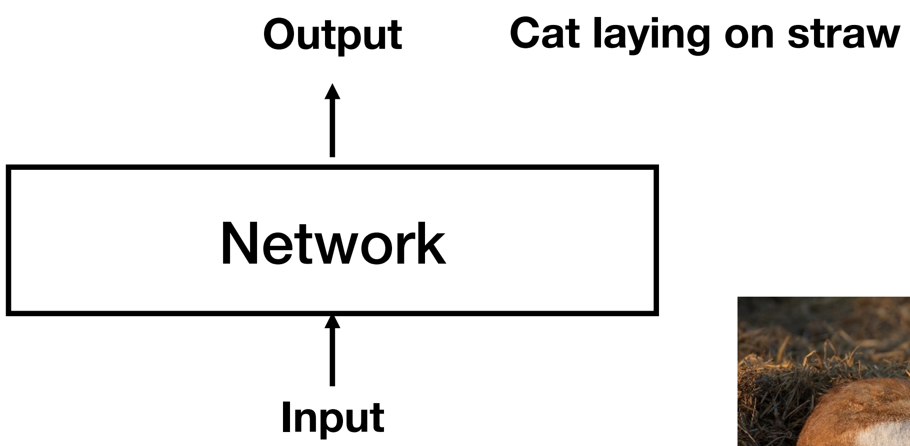
- Many-to-many
  - Regression
    - Daily weather for next week
  - Classification
    - Next sentence
    - Take a piece of music and compose the next section
    - Machine translation





Which task?

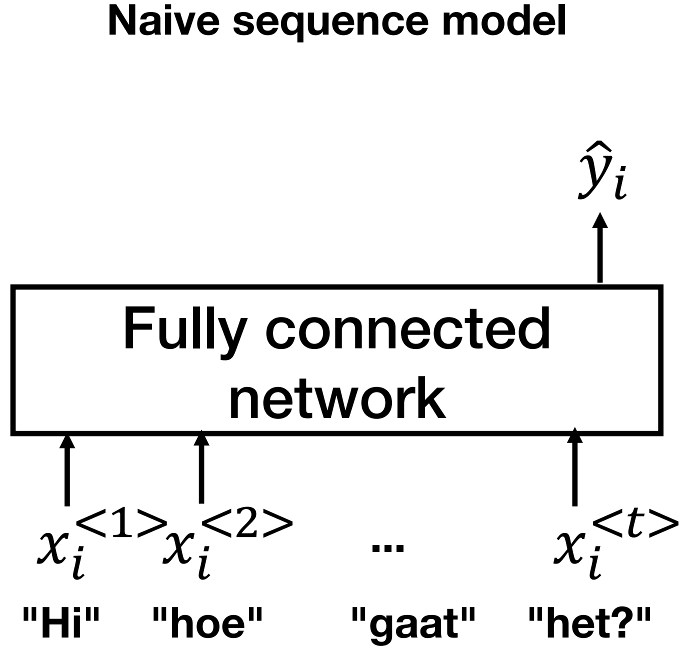
- One-to-many
  - Classification
    - Automatic captioning



How to model?

Naive model

- Why not just a fully connected network?
  - For this we will need VERY many parameters.
  - The sequence length might vary between examples.
  - We can often process each element independently and equally.
    - "gaat" is the same word regardless of position in a sentence.



How to model?

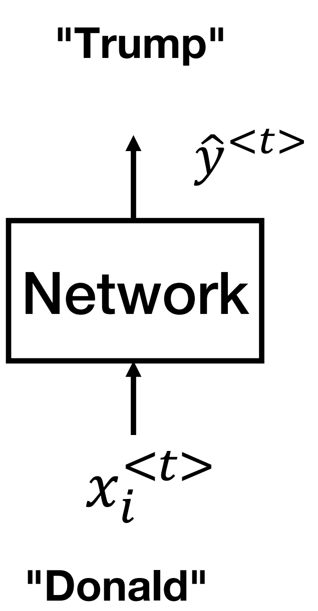
- What about a very simple model that just models the next word?

  Dataset:

  - The president of the United States is Donald Trump.
  - Wherever Donald Trump goes, security is tight.

- A simple network could learn that ‘Donald’ is always followed by ‘Trump’

- Pretty easy to learn, not so realistic…



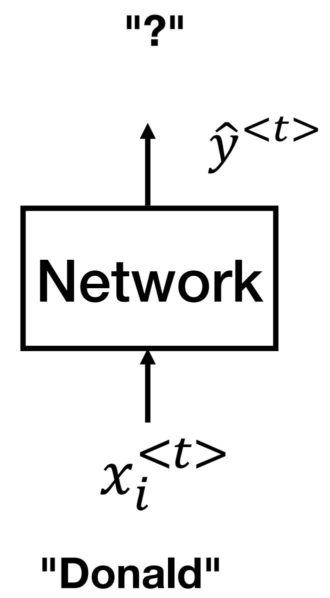
How to model?

- What about a very simple model that just models the next word?

  Dataset:

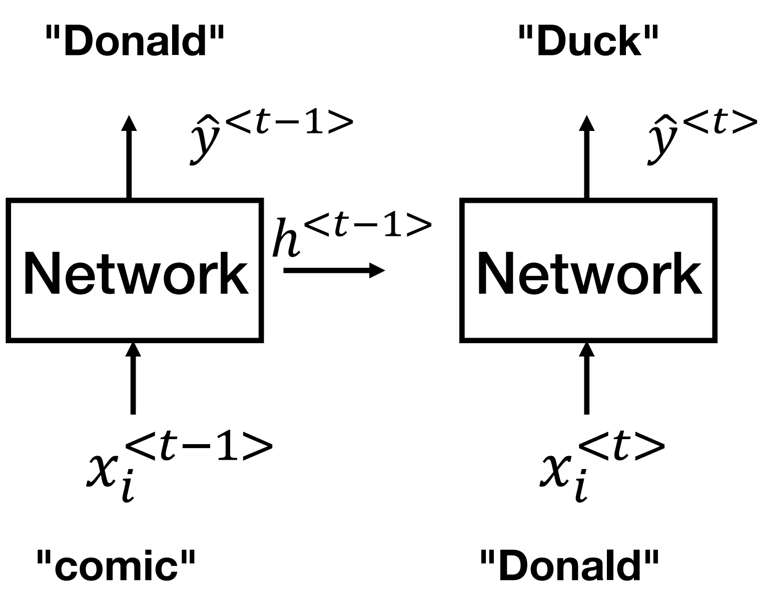
  - The president of the United States is Donald Trump.
  - Wherever Donald Trump goes, security is tight.
  - The comic Donald Duck is very famous.

- Now what?



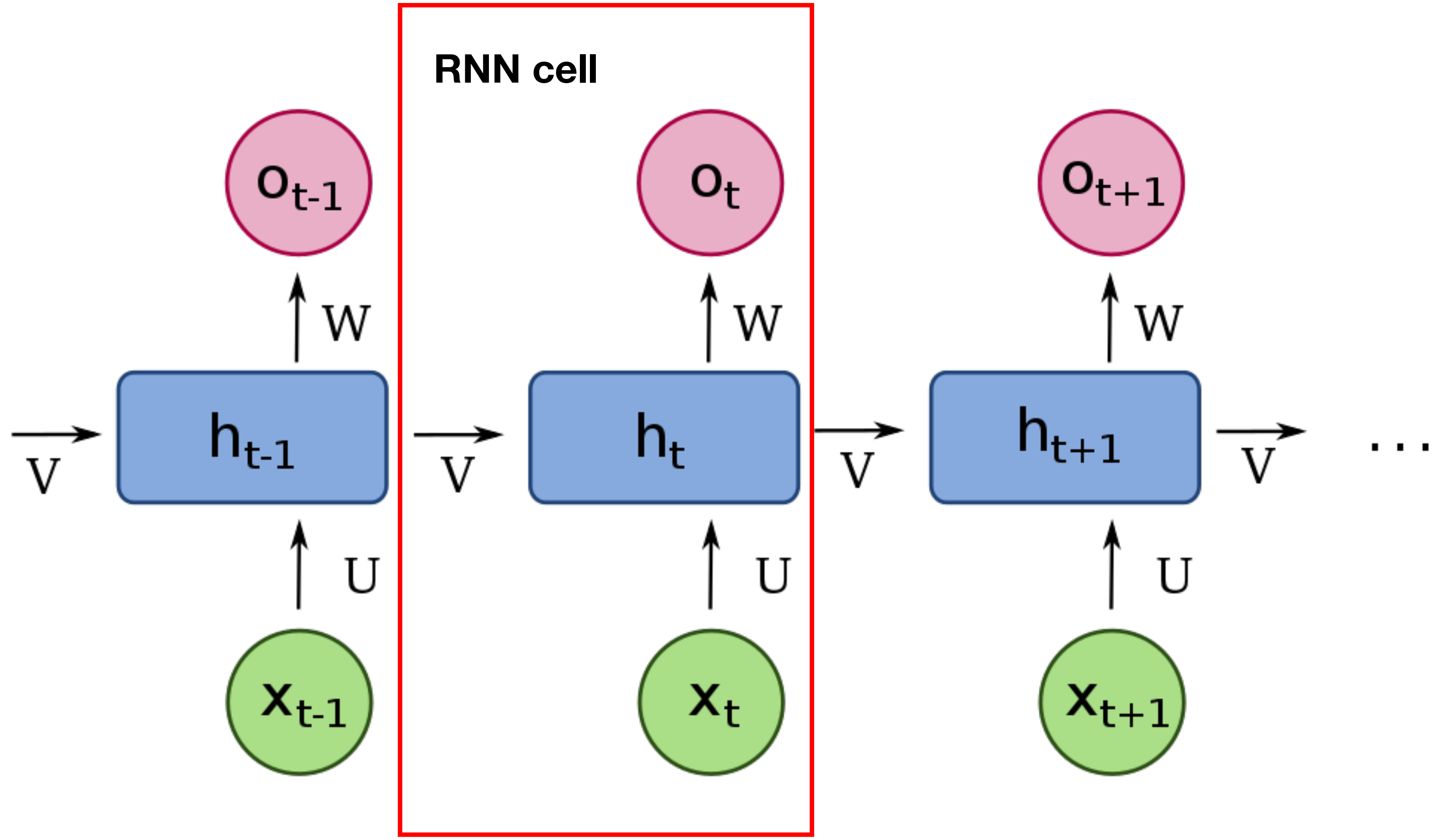
How to model?

- Meaning depends on context!
- Recurrent networks have a memory (a ‘state’), $h^{<t-1>}$ in the figure below
- The next prediction $\hat{y}^{<t>}$ depends not only on the input $x_i^{<t>}$, but also on the context ‘remembered’ by the network, $h^{<t-1>}$.



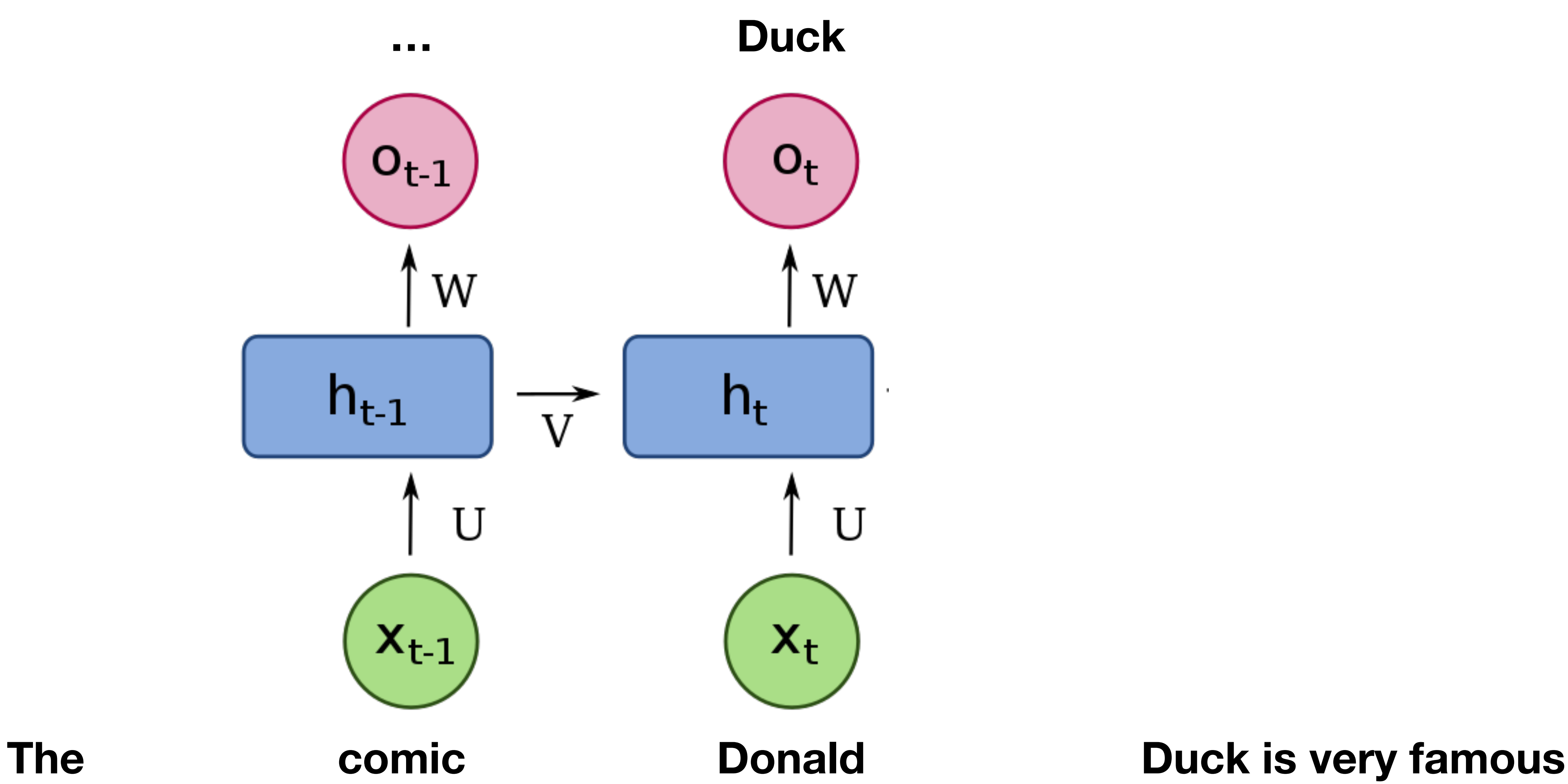
Simple RNN

- U, V, W are the weight matrices
- Weights are reused!



Simple RNN

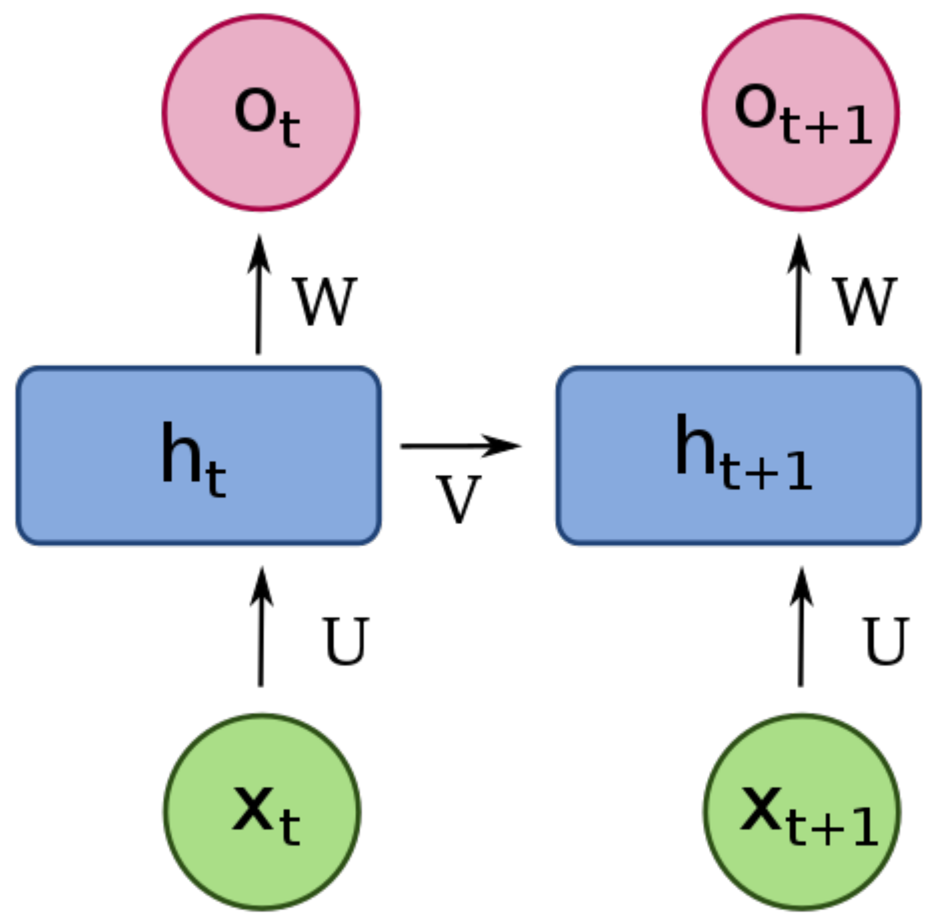
- Weights are reused: predictions the same, irrespective of where "comic Donald" occurs in a sequence



Simple RNN

- Weights are reused: predictions the same, irrespective of where "comic Donald" occurs in a sequence

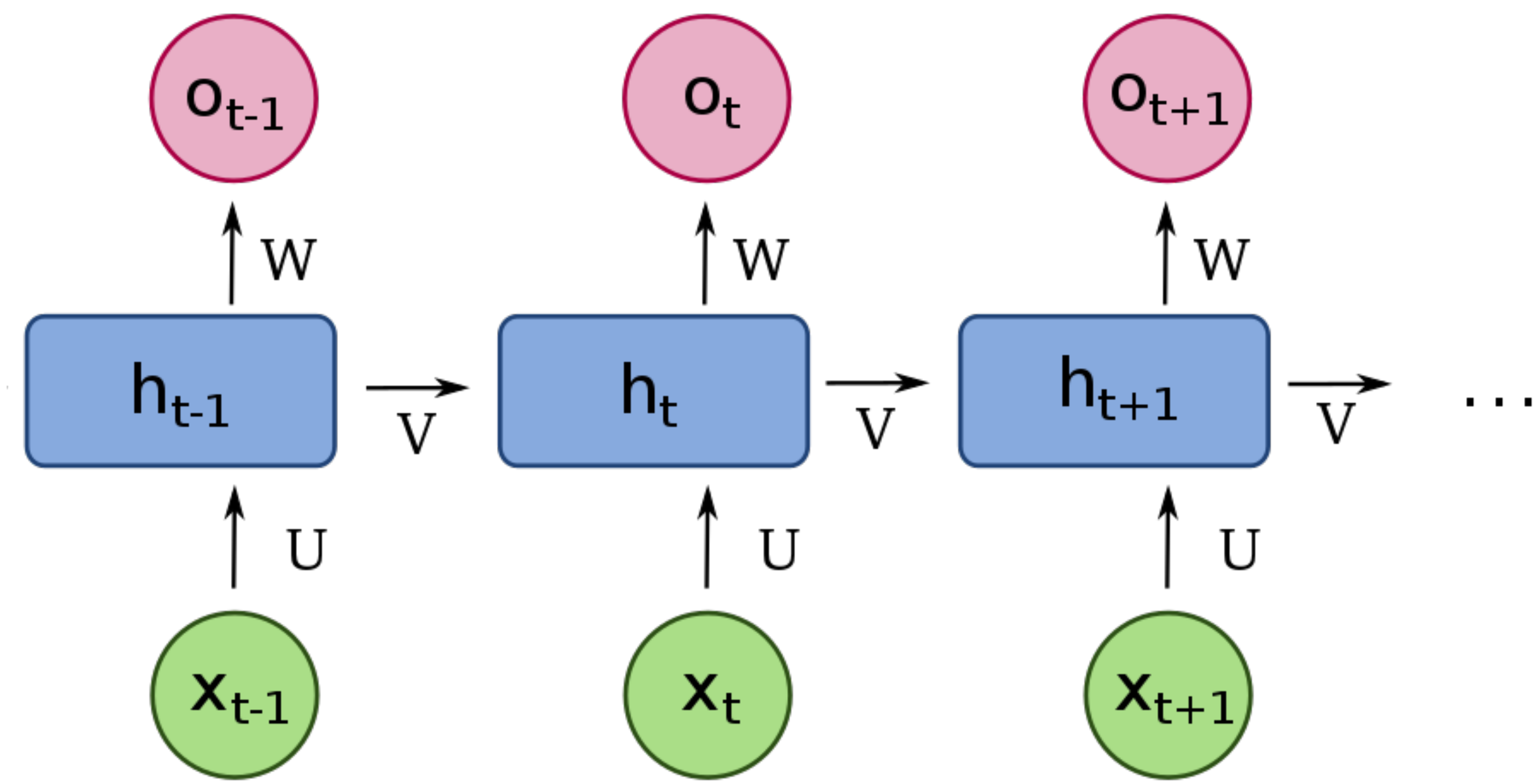
... **Duck**



Do you know the **comic** **Donald** **Duck?**

Simple RNN

- Of course, this changes if we use longer sequence length (i.e. look at more context/history)
- Memory h_{t-1} *could* alter the prediction



Do you know **the** **comic** **Donald** **Duck?**

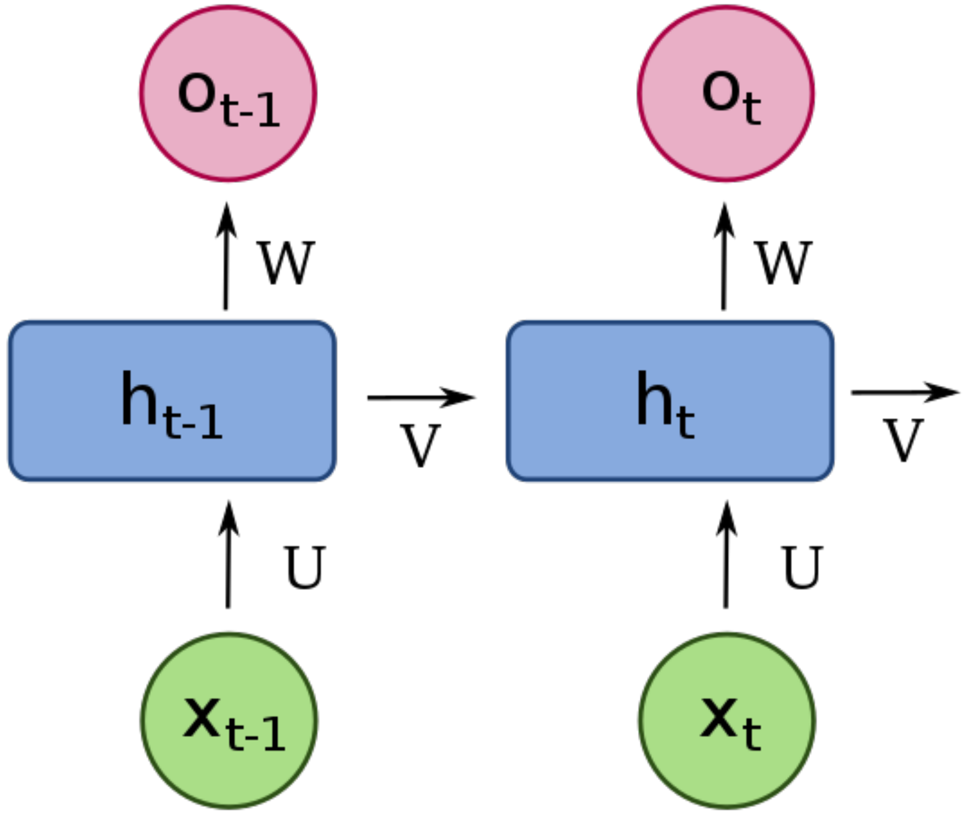
Simple RNN

- The math inside a simple RNN cell…

$$h_t = \sigma(U \cdot x_t + V \cdot h_{t-1} + b_h)$$
$$o_t = \sigma(W \cdot h_t + b_o)$$

This is often considered the next dense layer

This is often considered the RNN cell



- Compare to dense layer

$$o_t = \sigma(W \cdot x_t + b)$$

Simple RNN

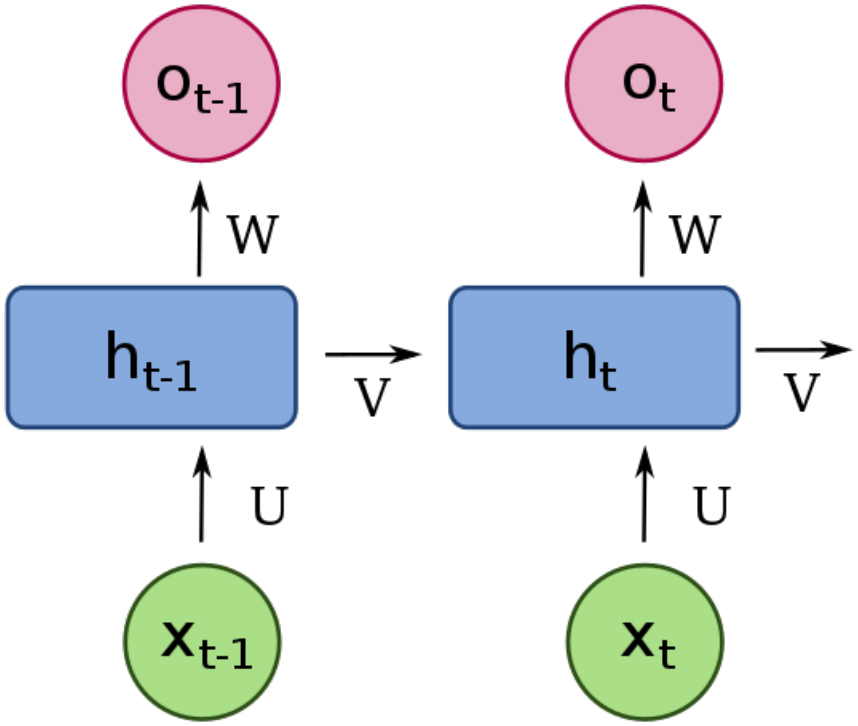
- How many parameters do we have?

$(\boldsymbol{n},\boldsymbol{m})(\boldsymbol{m})\quad(\boldsymbol{n},\boldsymbol{n})(\boldsymbol{n})\qquad(\boldsymbol{n})$

$$h_t = \sigma(U \cdot x_t + V \cdot h_{t-1} + b_h)$$ Layer_simple_rnn

$$o_t = \sigma(W \cdot h_t + b_o)$$ Layer_dense

$(\boldsymbol{k},\boldsymbol{n})\ (\boldsymbol{n})\qquad(\boldsymbol{k})$



- Length of vector x_t: # features (m)
- Length of hidden state vector h_t: we choose it! (based on how complex a memory we think we need) (n)
- Length of vector o_t: # outputs (classes / numbers) per timestep (k)
- It does NOT depend on sequence length! (because weights are ‘recycled’)

Simple RNN

- How many parameters do we have?

$$\underset{(\boldsymbol{n},\boldsymbol{m})(\boldsymbol{m})\ (\boldsymbol{n},\boldsymbol{n})(\boldsymbol{n})\quad (\boldsymbol{n})}{}$$

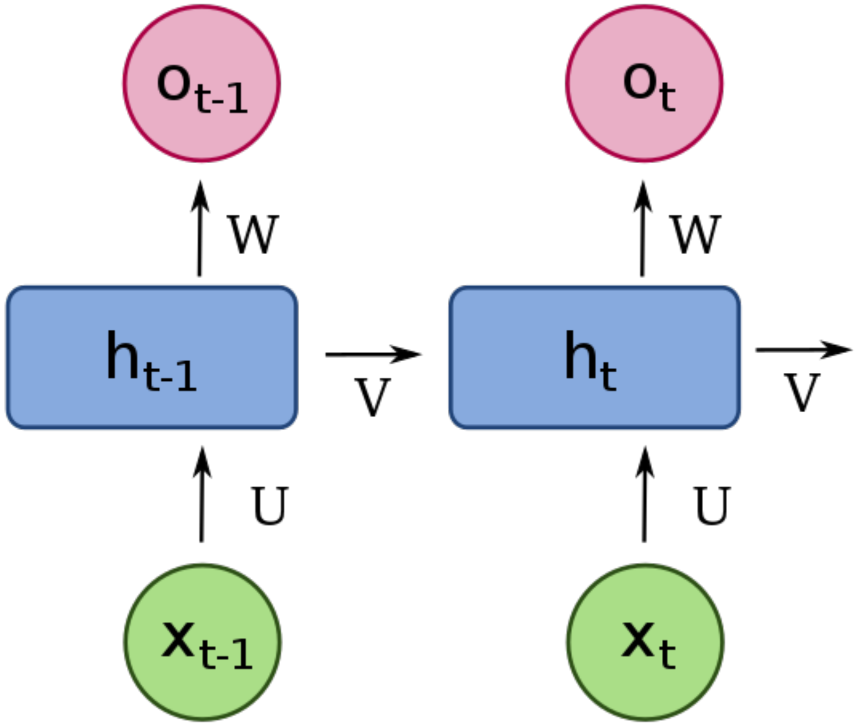
$$\boxed{h_t = \sigma(U \cdot x_t + V \cdot h_{t-1} + b_h)}$$

Layer_simple_rnn

$$\boxed{o_t = \sigma(W \cdot h_t + b_o)}$$

Layer_dense

$$\underset{(\boldsymbol{k},\boldsymbol{n})\ (\boldsymbol{n})\quad (\boldsymbol{k})}{}$$



- Sum up all weights and biases:

$$\#params = (n \cdot m + n \cdot n + n) + (k \cdot n + k)$$

Simple RNN

- How many parameters do we have?

$params = (n \cdot m + n \cdot n + n) + (k \cdot n + k)$

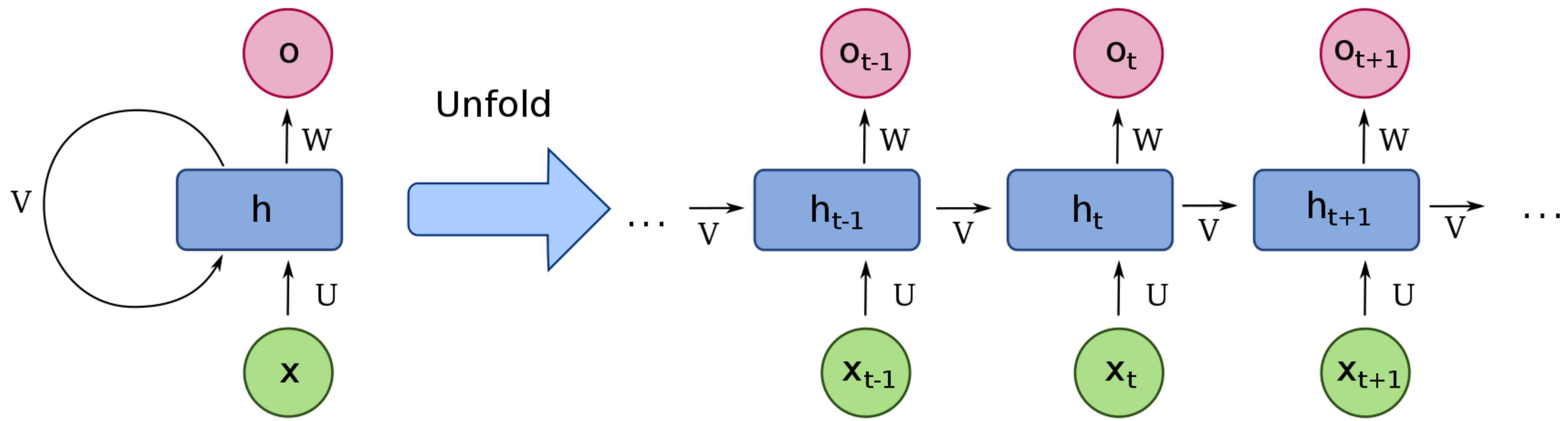
Example: trying to predict temperature for tomorrow

- 14 features (e.g. pressure, temperature, wind velocity, etc)
- 16 hidden states (we choose it!)
- 1 output per timepoint (only temperature)

$\#params = (16 \cdot 14 + 16 \cdot 16 + 16) + (16 \cdot 1 + 1) = 496 + 17 = 513$

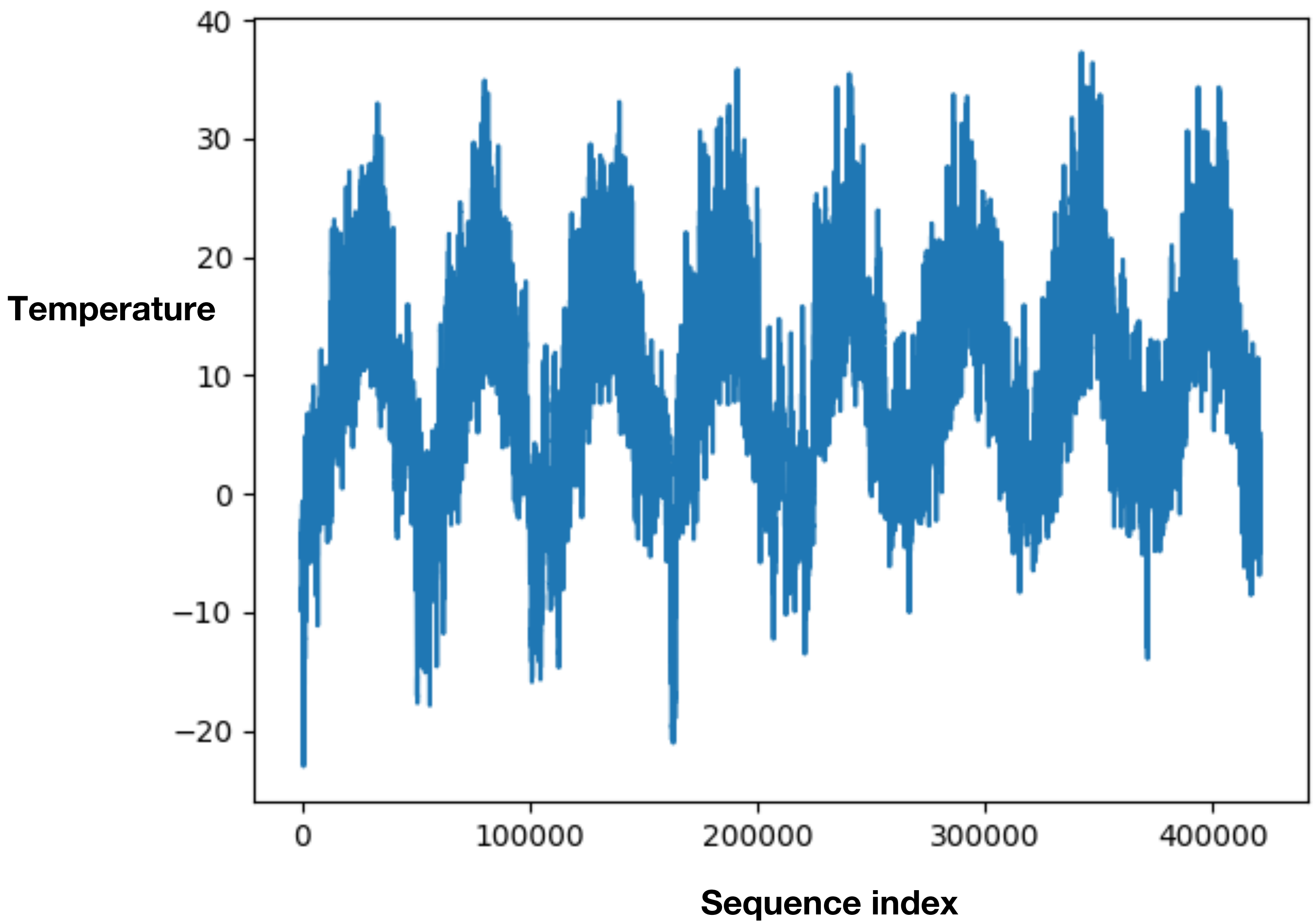
Unfolding an RNN

- Sometimes, literature shows RNN as a loop. We have looked at ‘unfolded’ RNNs. Same thing, just different visual representation.



Notebook

- The Jena weather dataset.
- A long sequence of weather measurements.
- Each measurement consists of 15 variables.
- We see a time-dependent pattern in the data.



Today's program

- 14:00-14:15 Recap
- 14:15-15:00 Machine learning tasks: regression / classification
- 15:00-15:45 Hands-on: multiclass Fashion MNIST
- 15:45-16:15 Break
- 16:15-16:45 Optimizers, regularization techniques
- 16:45-17:30 Hands-on: Regularization techniques on F-MNIST
- 17:30-18:00 Analyzing sequential data, RNNs
- **18:00-19:00 Diner**
- 19:00-19:45 Hands-on: Predicting future temperatures with an RNN
- 19:45-20:15 Types of RNNs: LSTM, GRU
- 20:15-21:00 Hands-on: creating sequences, temperature prediction with GRU-based RNN
- Time left: Improving RNNs: regularization, stacking, stateful and bi-directional RNNs
- Time left: Hands-on: Improved RNNs on temperature prediction

Hands-on

Go to https://jupyter.lisa.surfsara.nl:8000/

Or https://dba.projects.sda.surfsara.nl/

Notebook: `04a-time-series-prediction.ipynb`

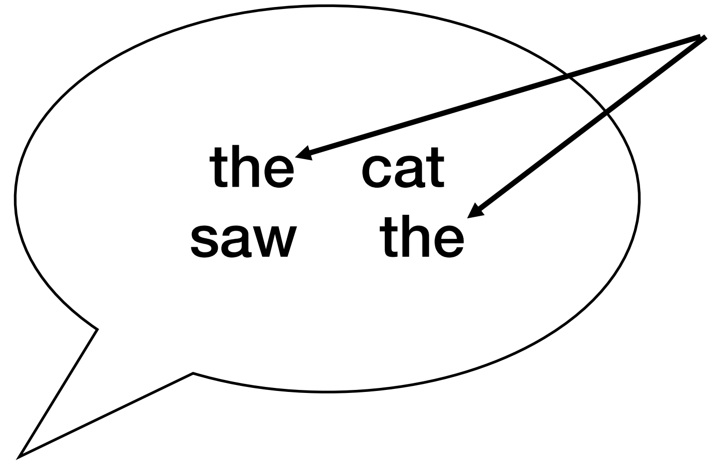
19:00-19:45

Hands-on recap

sharing parameters

- Each time-step we use the **same network** (same weight matrices), it just gets **different "memory"** (hidden state) passed to it.
- Sharing parameters **reduces** the number of **parameters** in the model.
- Reduces model complexity and risk of overfitting. This is the value of RNNs over fully connected!
- Assumption behind RNN: meaning of an input depends on
    - Input
    - Current hidden state
    - *Not* on location in the sequence

We process "the" in the same way.



layer_simple_rnn(units = 12, input_shape = c(sequence_length, features))

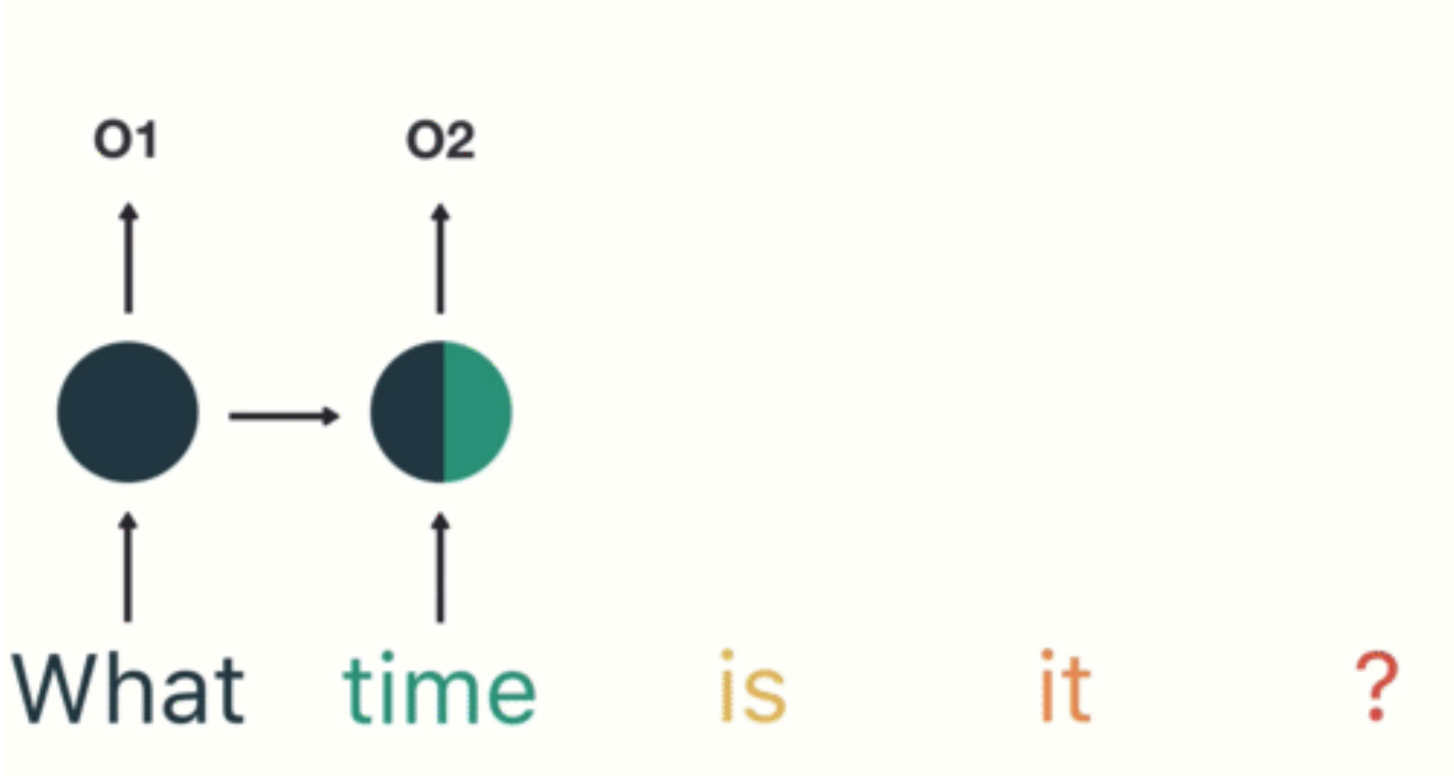
Today's program

- 14:00-14:15 Recap
- 14:15-15:00 Machine learning tasks: regression / classification
- 15:00-15:45 Hands-on: multiclass Fashion MNIST
- 15:45-16:15 Break
- 16:15-16:45 Optimizers, regularization techniques
- 16:45-17:30 Hands-on: Regularization techniques on F-MNIST
- 17:30-18:00 Analyzing sequential data, RNNs
- 18:00-19:00 Diner
- 19:00-19:45 Hands-on: Predicting future temperatures with an RNN
- **19:45-20:15 Types of RNNs: LSTM, GRU**
- 20:15-21:00 Hands-on: creating sequences, temperature prediction with GRU-based RNN
- Time left: Improving RNNs: regularization, stacking, stateful and bi-directional RNNs
- Time left: Hands-on: Improved RNNs on temperature prediction

Sequential data

Long term dependencies

- After each time-step we store some information.
- When we train RNNs we are training them to do two things.
  - Store the correct information between time-steps. This is **hard**.
  - Map the stored information to solve the task. This is **easy**.



Source: https://towardsdatascience.com/illustrated-guide-to-recurrent-neural-networks-79e5eb8049c9

Sequential data

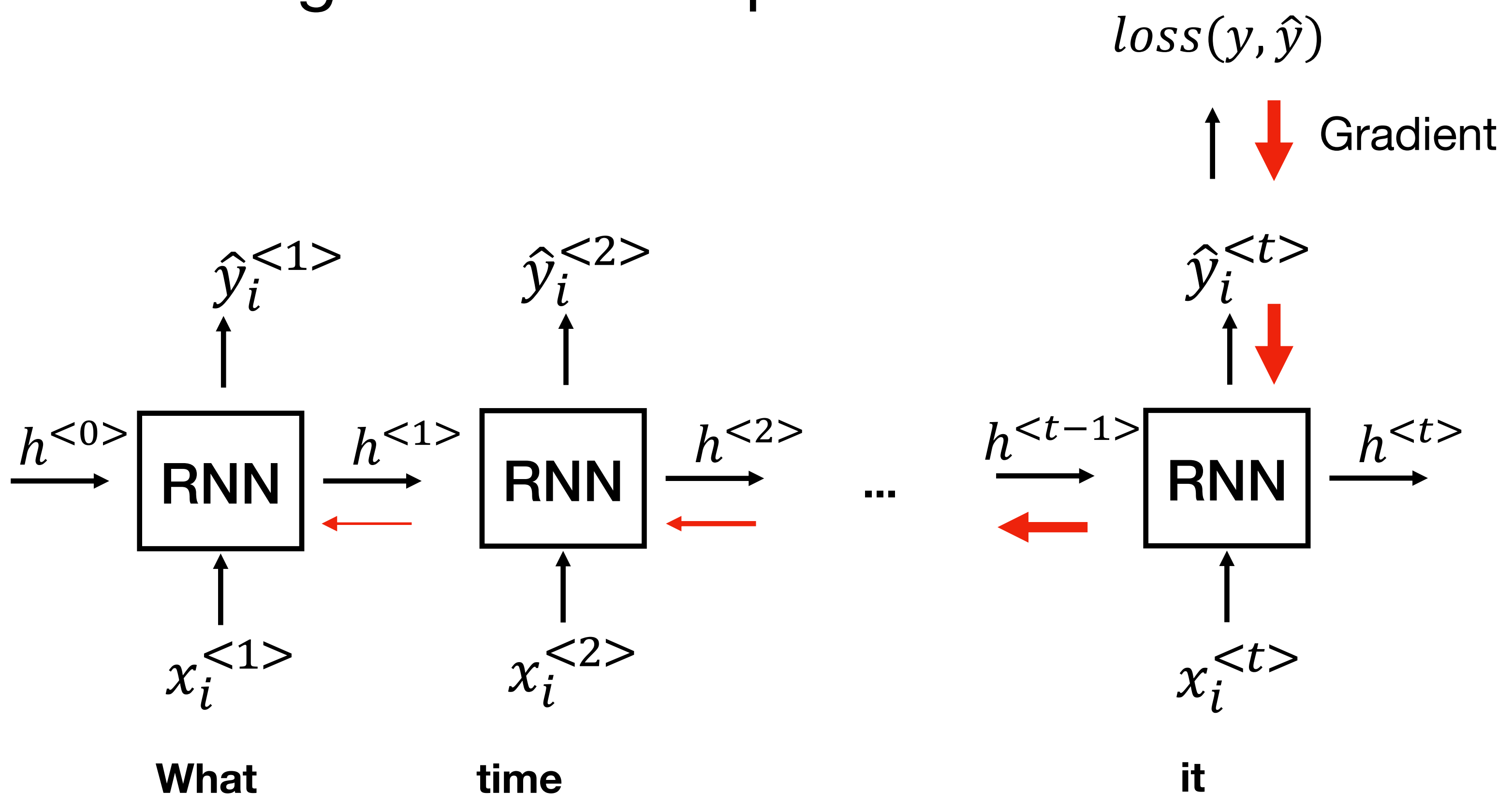
Long term dependencies

- Why is it hard?
- RNNs are trying to learn to **represent sequences by remembering what they contain**.
    - In question detection we want to remember if we saw "what", ...
- We learn to represent the sequence in order to solve the task at hand.
    - At **start** we are doing **poorly** (random weights) and we see almost no indication that "what" was used previously.
    - We want to update our RNN cell so that next time we remember when we see "what".

Sequential data

Long term dependencies

- If the sequence is long, little information is passed all the way to the end so a small error signal is sent back for the word "what".
- Small error signal = small updates

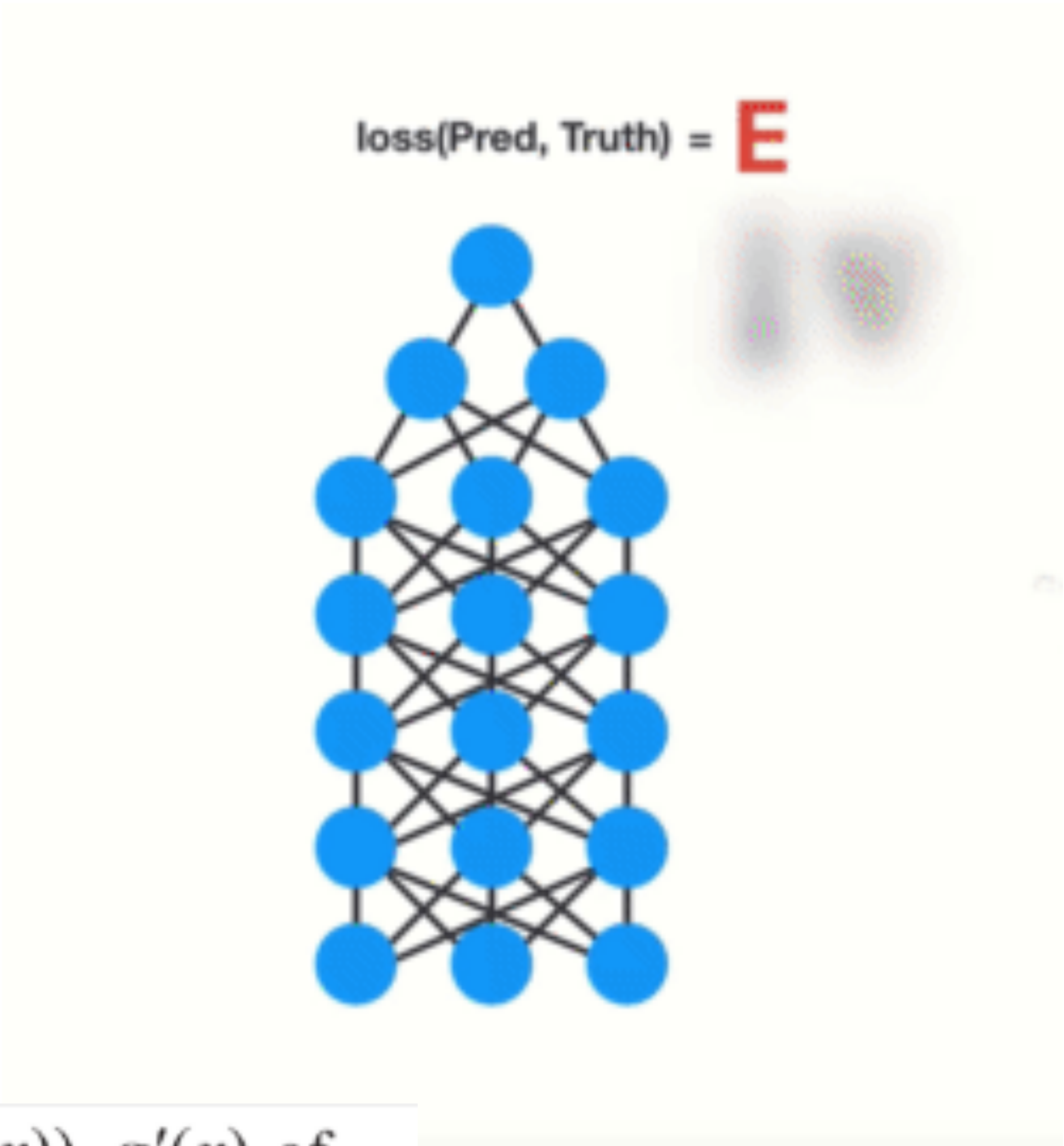


Sequential data

Long term dependencies

- The is the problem of **long term dependencies**, which is because the **gradient vanishes** (the error signal).

- This is **a general problem** in neural networks trained with gradient descent, but very tangible in RNNs due to their depth.



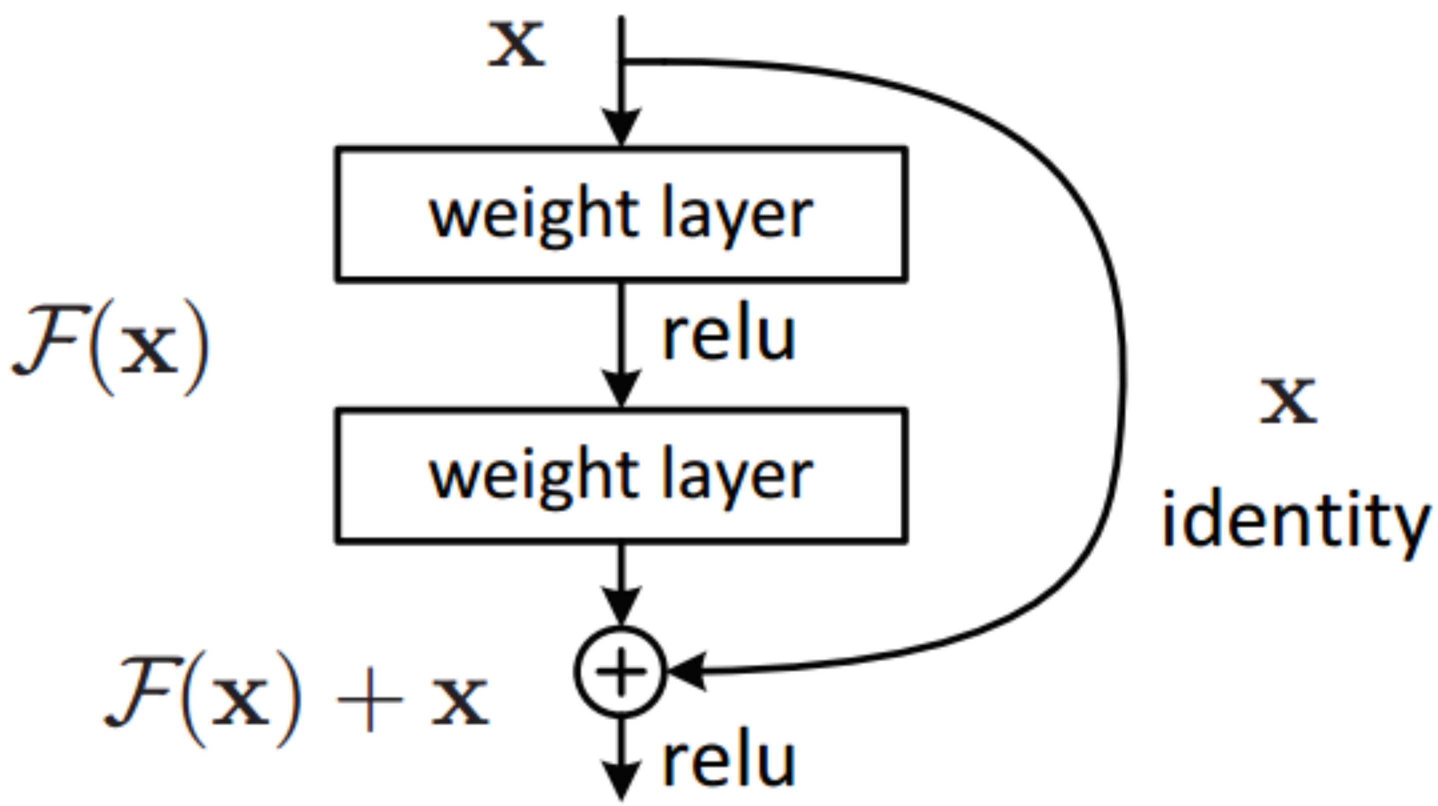
$k(x) = f(g(x))$	$k'(x) = f'(g(x)) \cdot g'(x)$ of $\frac{dk}{dx} = \frac{df}{dg} \cdot \frac{dg}{dx}$

Source: https://towardsdatascience.com/illustrated-guide-to-recurrent-neural-networks-79e5eb8049c9

Residual connections

Long term dependencies

- The solution to this problem in general are **residual connections**, (ResNet, 2015).
- We add connections which **bypass non-linear activations** (or go through fewer).
- This allows the error signal to flow directly to earlier layers.



Source: https://towardsdatascience.com/illustrated-guide-to-recurrent-neural-networks-79e5eb8049c9

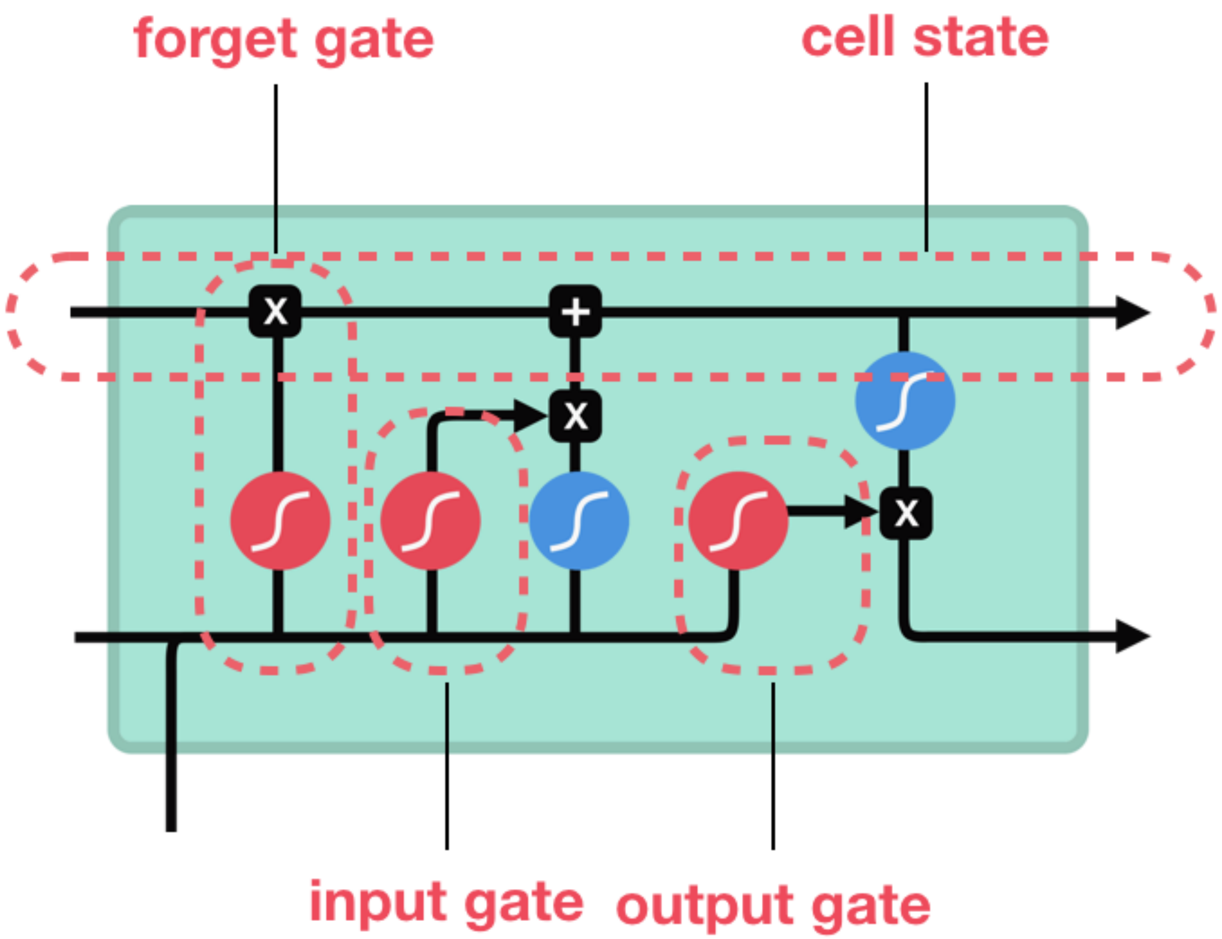
LSTMs and GRUs

Long term dependencies

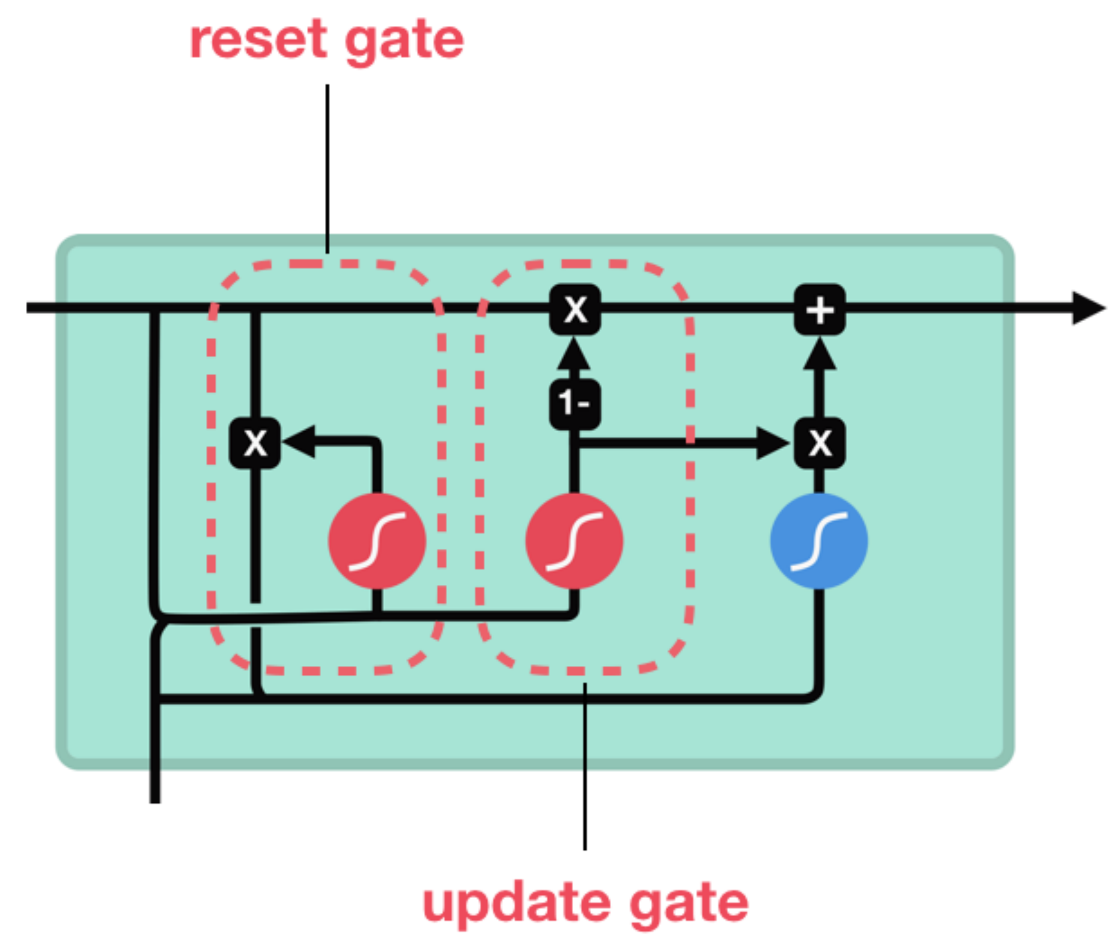
- The solution to the vanishing gradient problem in RNNs was a different implementation of the RNN cell.
  - LSTM (1997)
  - GRU (2014)
- They are more complex and expensive but are able to deal better with long term dependencies.
  - LSTM is heavier than GRU.

```
layer_gru(units = 10)          layer_lstm(units = 10)
```

LSTM



GRU



Source: https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

Generating sequences

We want to break this long sequence into many sequences

(1, ...) (2, ...) (3, ...) (4, ...) (5, ...) (6, ...) (7, ...) (8, ...) (9, ...) (10, ...) (11, ...) (12, ...) (13, ...) (14, ...)

Reshape approach - sequence length = 7

(1, ...) (2, ...) (3, ...) (4, ...) (5, ...) (6, ...) (7, ...) (8, ...) (9, ...) (10, ...) (11, ...) (12, ...) (13, ...) (14, ...)

1st example	(1, ...) (2, ...) (3, ...) (4, ...) (5, ...) (6, ...) (7, ...)
2nd example	(8, ...) (9, ...) (10, ...) (11, ...) (12, ...) (13, ...) (14, ...)

Generating sequences

We want to break this long sequence into many sequences

(1, ...) (2, ...) (3, ...) (4, ...) (5, ...) (6, ...) (7, ...) (8, ...) (9, ...) (10, ...) (11, ...) (12, ...) (13, ...) (14, ...)

Shift approach, using shift = 2, sequence length = 7

(1, ...) (2, ...) (3, ...) (4, ...) (5, ...) (6, ...) (7, ...) (8, ...) (9, ...) (10, ...) (11, ...) (12, ...) (13, ...) (14, ...)

↓

1st example (1, ...) (2, ...) (3, ...) (4, ...) (5, ...) (6, ...) (7, ...)

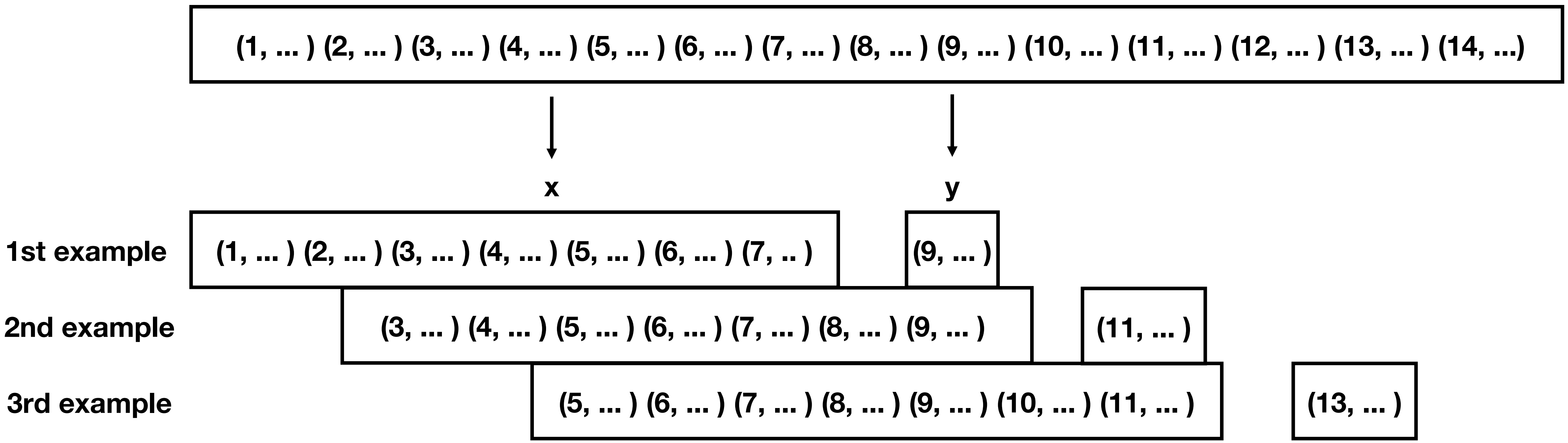
2nd example (3, ...) (4, ...) (5, ...) (6, ...) (7, ...) (8, ...) (9, ...)

3rd example (5, ...) (6, ...) (7, ...) (8, ...) (9, ...) (10, ...) (11, ...)

4th example (7, ...) (8, ...) (9, ...) (10, ...) (11, ...) (12, ...) (13, ...)

Generating sequences

Shift approach, using shift = 2, sequence length = 7, target shift = 1



Hands-on

Go to https://jupyter.lisa.surfsara.nl:8000/

Or https://dba.projects.sda.surfsara.nl/

Notebook: `05a-rnns.ipynb`

20:15-21:00

Today's program

- 14:00-14:15 Recap
- 14:15-15:00 Machine learning tasks: regression / classification
- 15:00-15:45 Hands-on: multiclass Fashion MNIST
- 15:45-16:15 Break
- 16:15-16:45 Optimizers, regularization techniques
- 16:45-17:30 Hands-on: Regularization techniques on F-MNIST
- 17:30-18:00 Analyzing sequential data, RNNs
- 18:00-19:00 Diner
- 19:00-19:45 Hands-on: Predicting future temperatures with an RNN
- 19:45-20:15 Types of RNNs: LSTM, GRU
- 20:15-21:00 Hands-on: creating sequences, temperature prediction with GRU-based RNN
- **Time left: Improving RNNs: regularization, stacking, stateful and bi-directional RNNs**
- **Time left: Hands-on: Improved RNNs on temperature prediction**

Improving RNNs

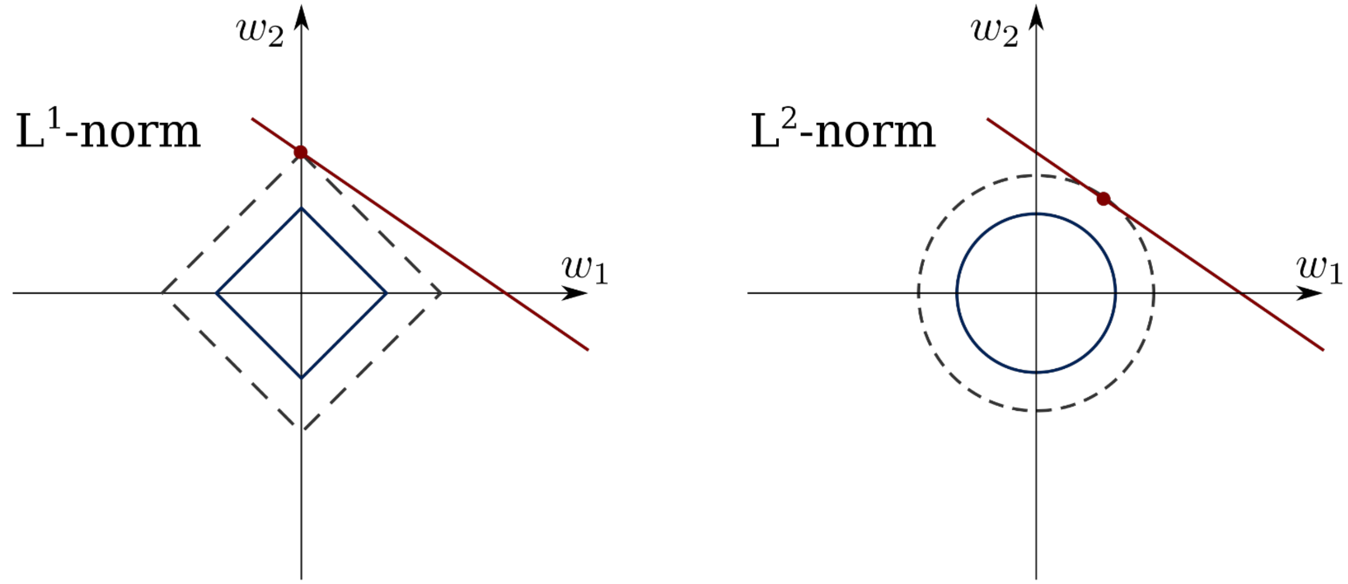
- Regularisation
  - L1/L2
  - Dropout, recurrent dropout
- Improving RNNs
  - Stacking
  - Stateful
  - Bi-directional

Improving RNNs
L2/L1 regularisation

- Just like with normal dense layers.
- we add L2/L1 regularisation to the weights learnt in the RNN cell.

```
layer_gru(units = 10, kernel_regularizer = regularizer_l2(l = 0.001))

layer_gru(units = 10, kernel_regularizer = regularizer_l1(l = 0.001))
```



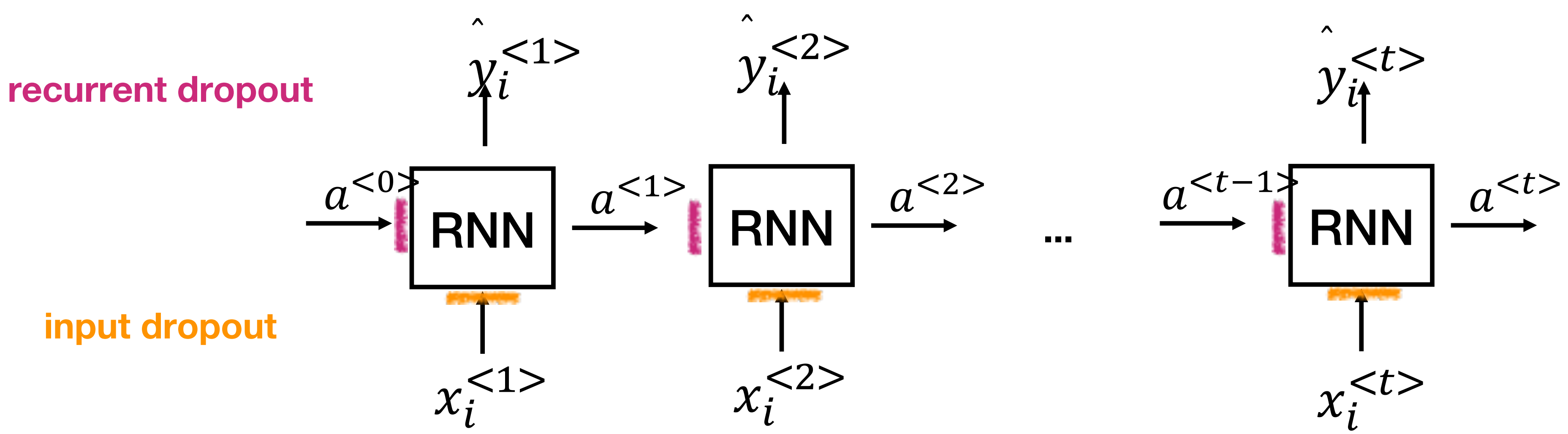
Red line: identical predictions of the NN (too many degrees of freedom)
Red point: solution for (w_1, w_2) preferred by each norm.

Improving RNNs

Dropout

- In RNNs we consider dropouts in two locations.
  - **Input**
  - **Recurrent dropout**

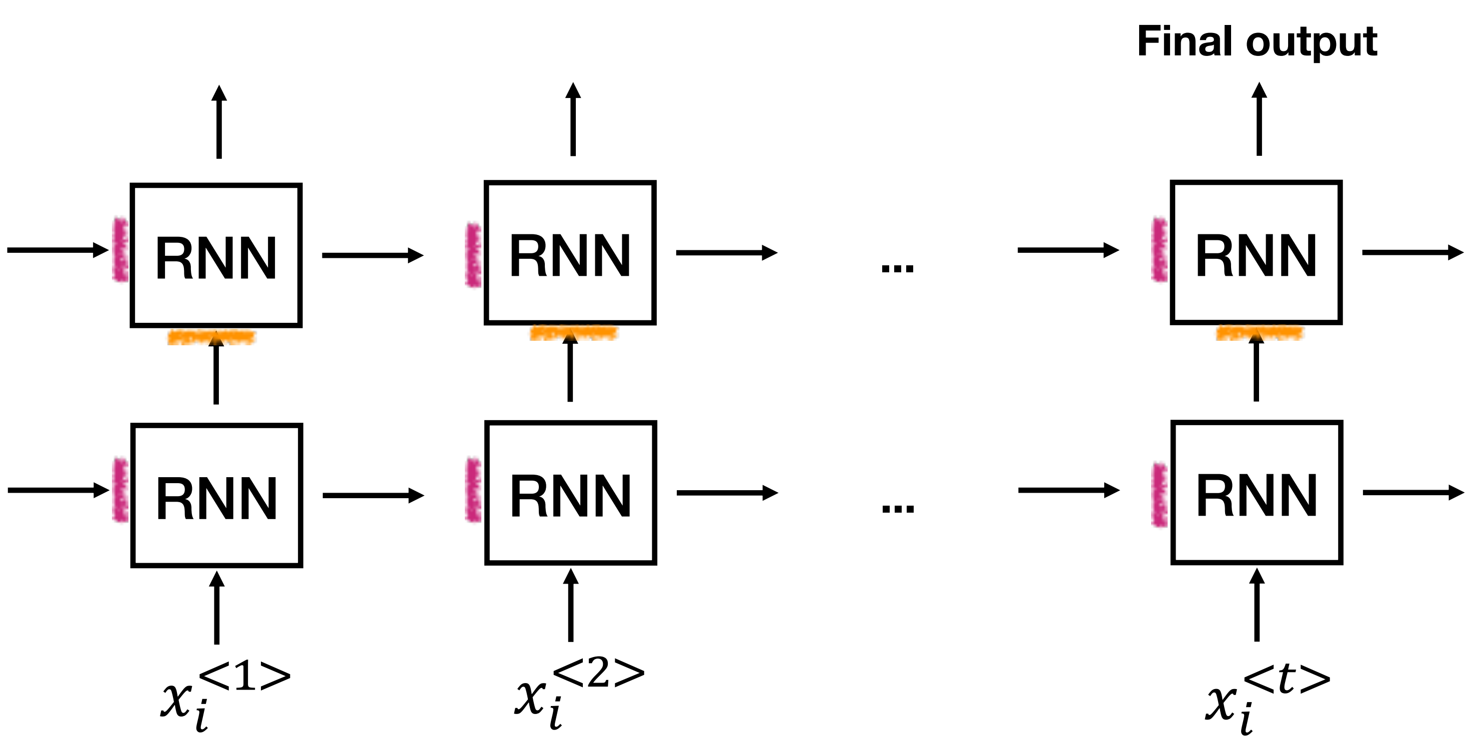
layer_gru(units = 10, dropout = 0.2, recurrent_dropout = 0.3)



Improving RNNs

Stacking RNNs

- Why would we consider input dropout?
  - Maybe in production we might not always get all inputs.
- More likely, we are stacking RNNs.
- Stacking RNNs is like adding additional layers in a dense network.
  - We never go that deep, 1-6 layers. Long training time.

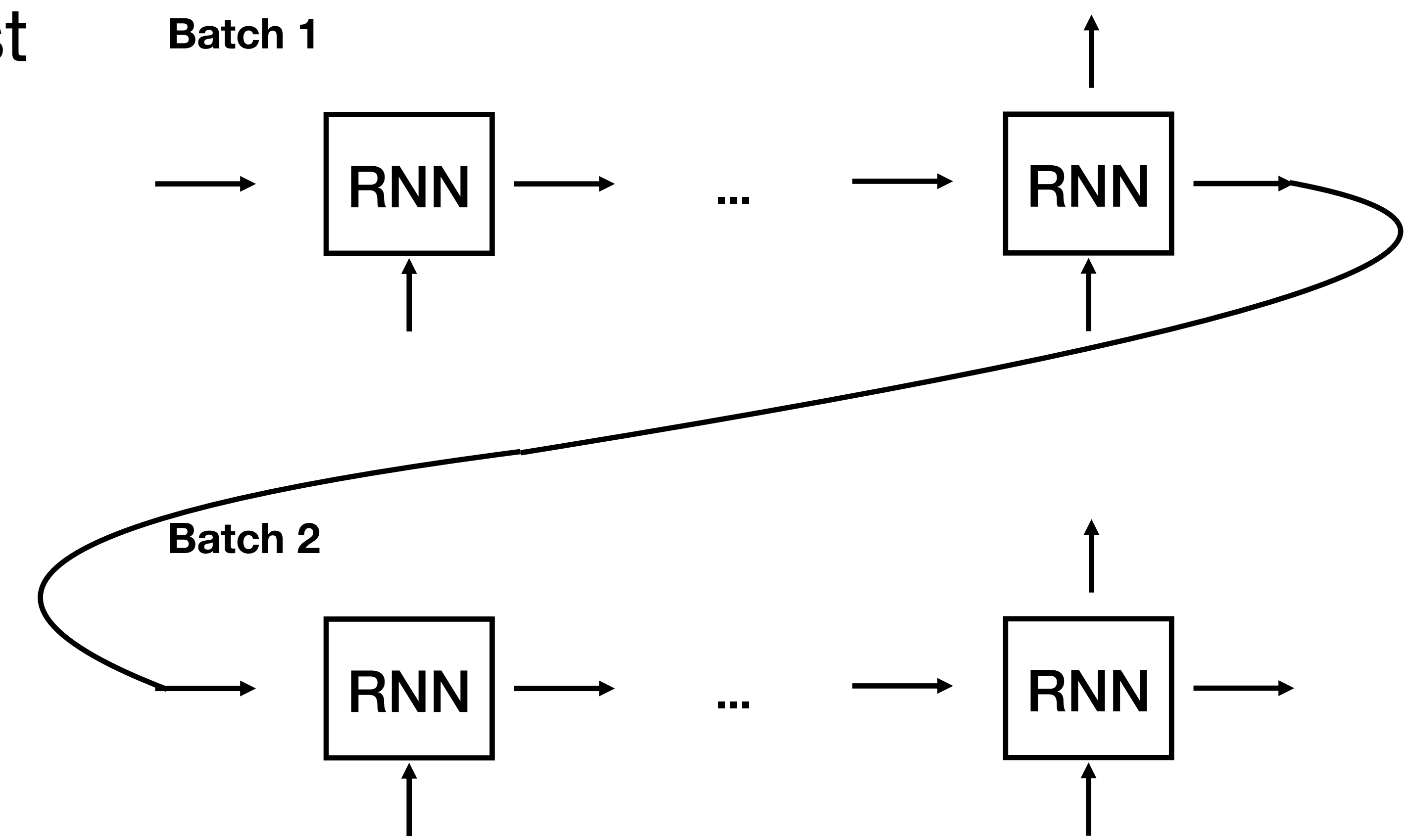



```
layer_gru(units = 10, return_sequences = TRUE) %>%
layers_gru(units = 10)
```

Improving RNNs

Stateful

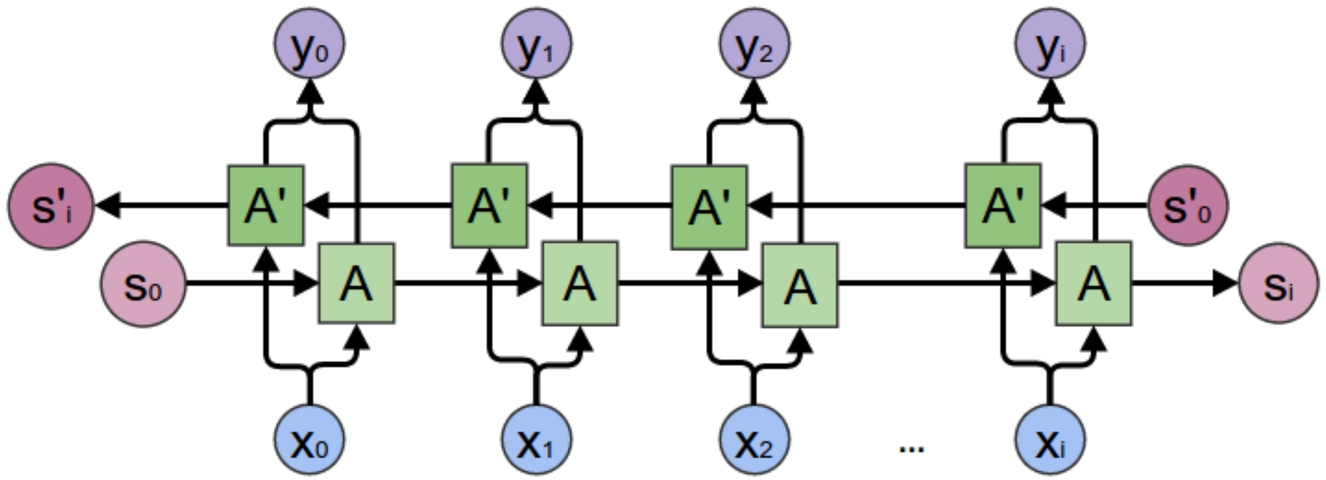
- A stateful RNN passes the last state of the previous batch to as an initial state to the next batch.
  - Otherwise the initial state is "all zeroes".
- This is useful if there is some connection between batches.
  - For example, the batches are in sequence.



Improving RNNs

Bi-directional

- We process the sequence in both directions.
- Very helpful for example in named entity recognition, in which we classify every word as a "person", "place", ...
    - He said, "Teddy Roosevelt ...
    - He said, "Teddy bears ...



Summary

- Regularisation
  - L1/L2
  - Dropout, recurrent dropout
- Improving RNNs
  - Stacking
  - Stateful
  - Bi-directional

Hands-on

Go to https://jupyter.lisa.surfsara.nl:8000/

Or https://dba.projects.sda.surfsara.nl/

Notebook: `05b-rnns-improved.ipynb`

Duration: +/- 45 mins

Will not cover

- We did not cover any natural language processing (NLP).
    - **Word embeddings**, representing words as vectors
- RNNs have been very successful in NLP over the years.
- NLP requires **a lot** of data preprocessing and large models.
- Same models used.
- New Paradigm: Transformers