FIT3181 Deep Learning

Week 07: DL for time-series and temporal data- RNNs and LSTMs

Lecturer: Trung Le

Email: trunglm@monash.edu

Department of Data Science and AI
Faculty of Information Technology, Monash University, Australia

Outline

- Time-series and sequence modelling
- Recurrent neural networks NNs
  - Architecture and connection to DNN
  - Learning with Back Propagation Through Time
  - Applications of RNNs
- Long Short Term Memory (LSTM)
  - Long-term dependency matters
  - LSTM Cell: forget gate, input gate and output gate
  - LSTM with Peephole connection
- Gated Recurrent Unit

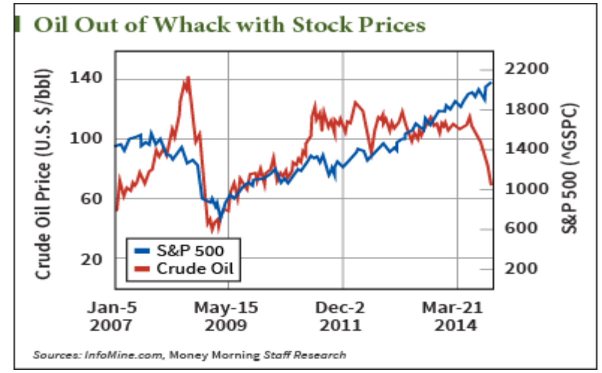
- **Further reading recommendation**
  - [HandsOn, ch15], [DL, ch11]

Sequence modelling

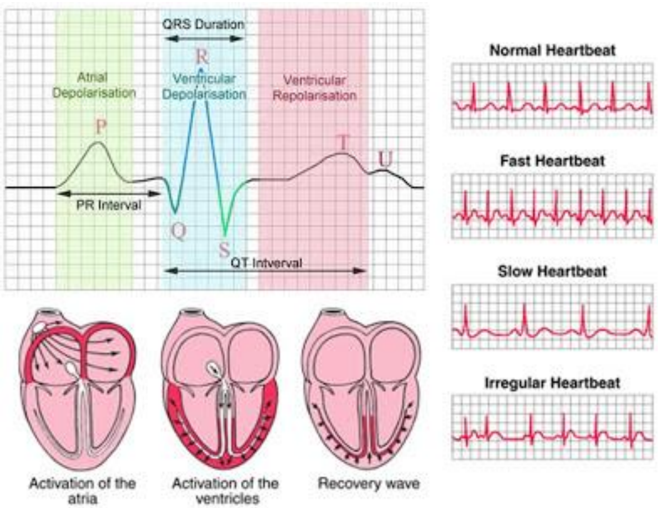
Time series and sequential data

Speech

Video surveillance



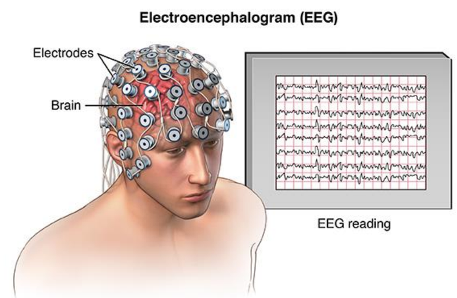
Stock market prices



Electrocardiography signals = electrical activity of the heart over time

- We live in a **time-space universe**
- All data collected has a **timestamp**
- Time-series/sequential data

  = collection of sequential data points indexed by time order!



EEG brain signal

Texts, messages from media

Sequential data examples

- Data can also be viewed from different, more subtle angles

I love deep learning

- **Word level**
  - I, love, deep, learning
- **Character level**
  - I, l, o, v, e, d, e, e, p, l, e, a, r, n, i, n, g

Items

I → love → deep → learning

x_1 x_2 x_3 x_4

How words/items x_1, x_2, x_3, x_4, x_5 **cooccur** in a meaningful text?

$$p(x_1, x_2, x_3, x_4) = p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_1, x_2)p(x_4 \mid x_1, x_2, x_3)$$
$$= p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_{1:2})p(x_4 \mid x_{1:3})$$

Items

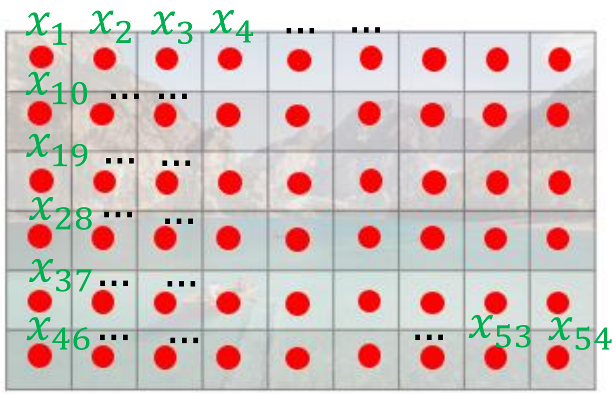
I → l → o → v → … → … → n → g

x_1 x_2 x_3 x_4 x_{16} x_{17}

$$p(x_1, x_2, \dots, x_{17}) = p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_{1:2})p(x_4 \mid x_{1:3}) \dots p(x_{15} \mid x_{1:14})p(x_{16} \mid x_{1:15})$$

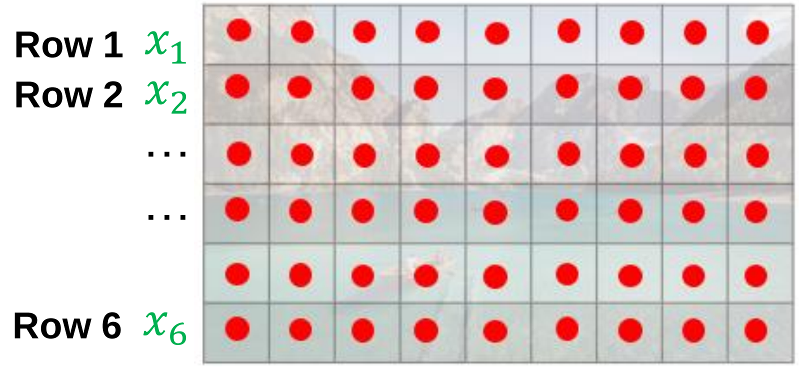
Sequential data examples

- Sequence of pixels or rows of pixels



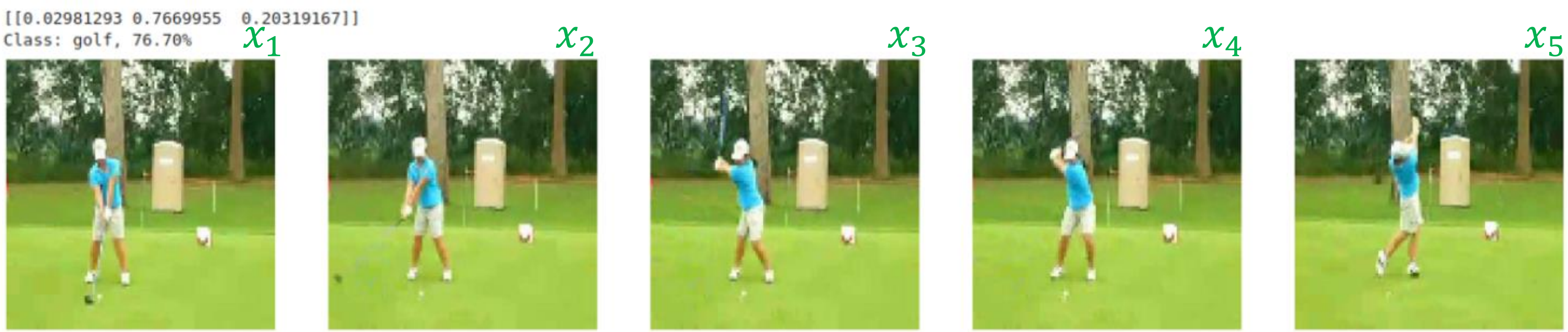
- Sequence of pixels

$p(x_{1:54}) = p(x_1)p(x_2 \mid x_1) \ldots p(x_{54} \mid x_{1:53})$



- Sequence of rows

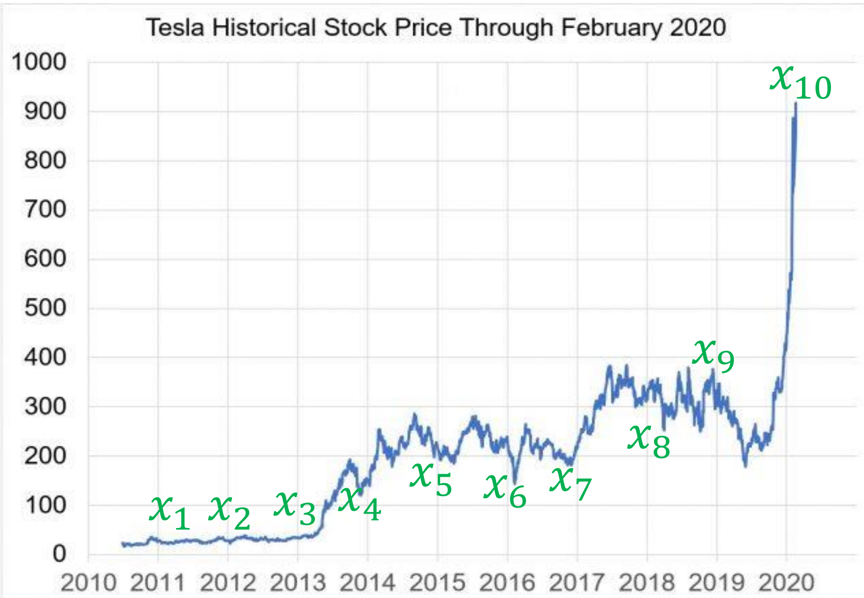
$p(x_{1:6}) = p(x_1)p(x_2 \mid x_1) \ldots p(x_6 \mid x_{1:5})$

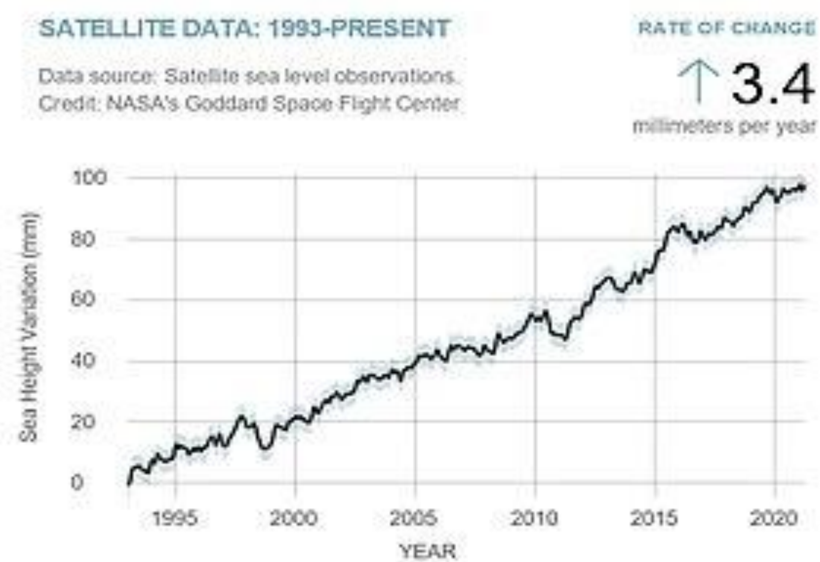


- Video as a **sequence of frames/images** (Source: medium.com)

$p(x_1, \ldots, x_5) = p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_{1:2})p(x_4 \mid x_{1:3})p(x_5 \mid x_{1:4})$

Time series data





$p(x_{1:10}) = p(x_1)p(x_2 \mid x_1) \dots p(x_{10} \mid x_{1:9})$

- Too simple and naïve modelling (i.e., underfitting modelling)
- Historical data are **not sufficient**
- Many **external factors**
  - Human behaviour, the success of Tesla, climate change, weather, and so on

$p(x_t \mid x_{t-1}, \dots, x_{t-k}) = ???$

- If we consider the **stock prices of the last k year**, what the **stock price** of the **current year**?

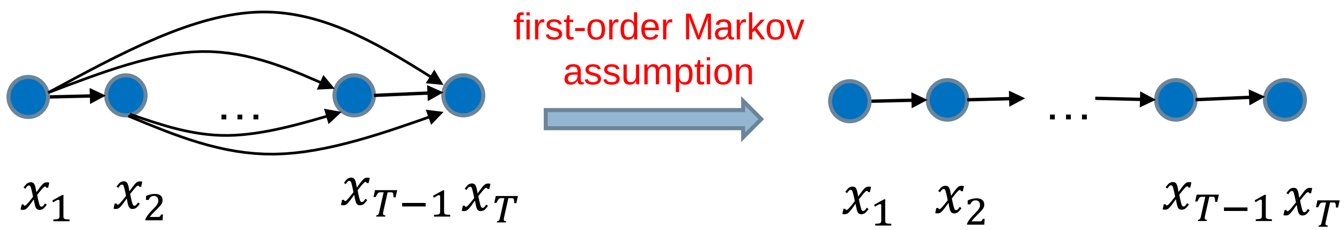
$p(x_{1995:2020}) = p(x_{1995})p(x_{1996} \mid x_{1995}) \dots p(x_{2020} \mid x_{1995:2019})$

x_i is **sea level rise** in year i

$p(x_t \mid x_{t-1}, \dots, x_{t-k}) = ???$

How to model sequential data?

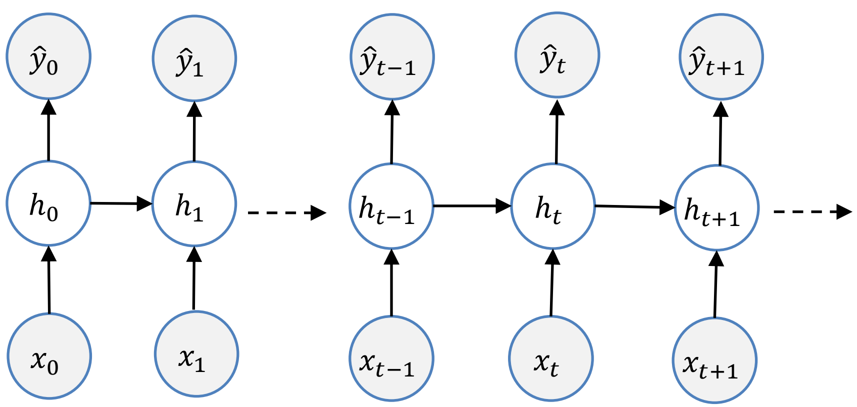
- **Markov models**
  - Hidden Markov Models (HMM), Dynamic Bayesian Networks (DBN), etc.



$p(x_1, .., x_T)$
$= p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_{1:2}) \dots p(x_T \mid x_{1:T-1})$

$p(x_1, .., x_T)$
$= p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_2) \dots p(x_T \mid x_{T-1})$

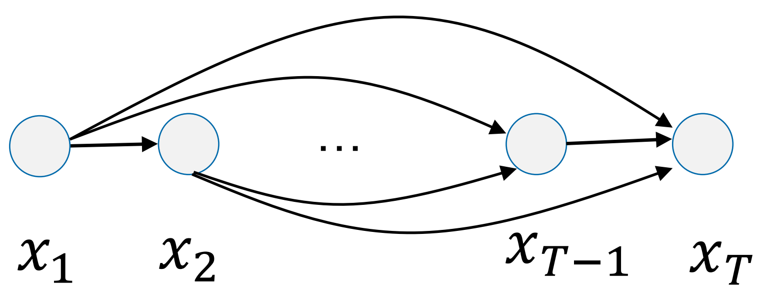
- **Autoregressive Model**



- The hidden state h_t stores **information of past observations** $x_{1:t}$ **= lossy historical summary**
- h_t is used to predict $\hat{y}_t$.
- $h_t = g(h_{t-1}, x_t)$ depends on $\boldsymbol{h}_{\mathbf{t-1}}$ and $\boldsymbol{x}_{\boldsymbol{t}}$.
- $p(x_0, .., x_T) =$
  $p(x_0)p(x_1 \mid x_0)p(x_2 \mid x_{0:1}) \dots p(x_T \mid x_{0:T-1}) =$
  $p(x_0)p(x_1 \mid h_0)p(x_2 \mid h_1) \dots p(x_T \mid h_{T-1})$
- RNNs belong to the family of autoregressive models

What are time-series and sequential data?

- Time-series/sequential data = collection of sequential data points indexed by time order
  - Traditionally, it is often the collection of measurements recorded repeatedly over time for the same object of interest
    - e.g., stock market prices, position of the missile, location of the car
  - However, modern machine learning problems often deal with timestamped data collected from heterogeneous sources:
    - e.g., analysing stock market prices together with real-world events, fusion of missile location bearings + weather information for object tracking
    - DL revolution has enabled much more sophisticated/complex problems
- What to model?
  - When historical observations influence on the future



The central question is to model the joint distribution $p(x_1, x_2, \ldots, x_T)$

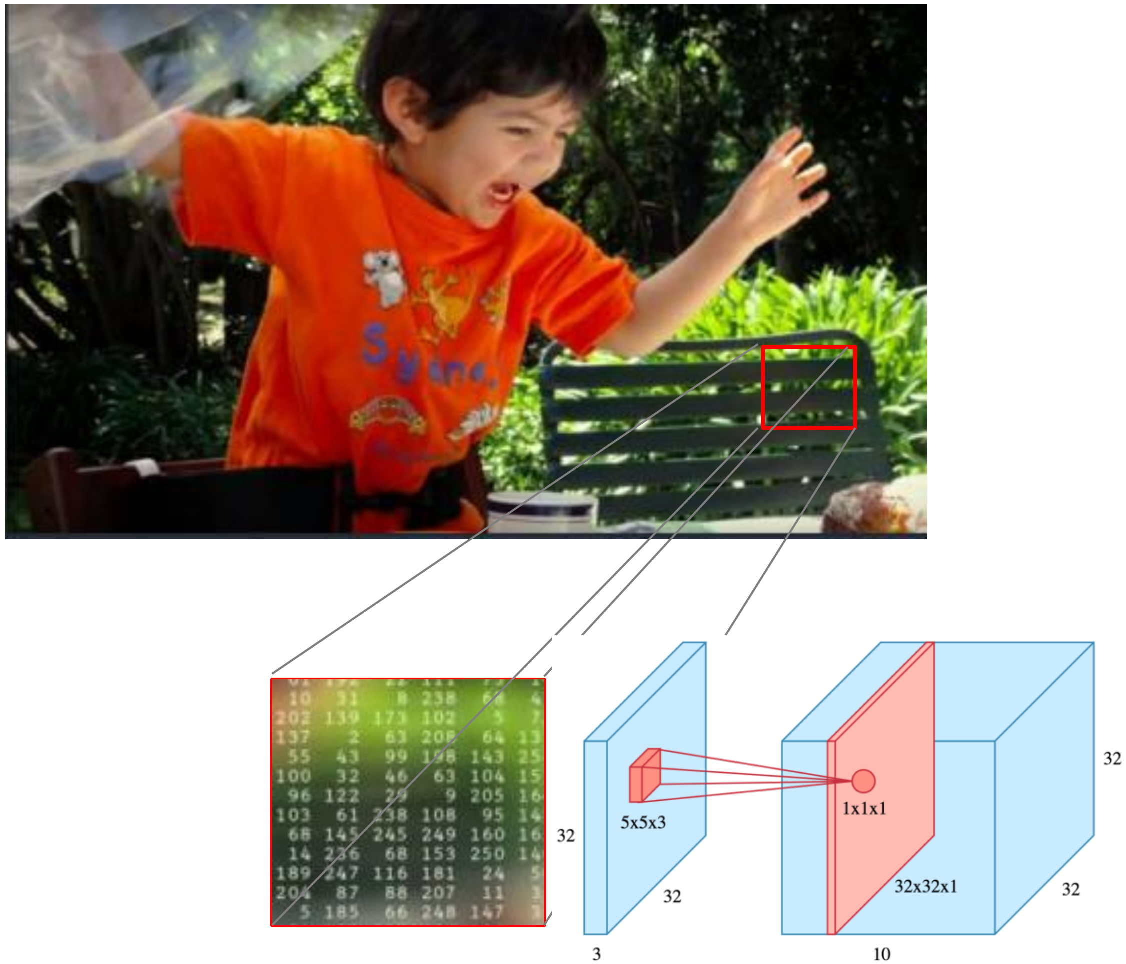
Sequence Modelling: Summary

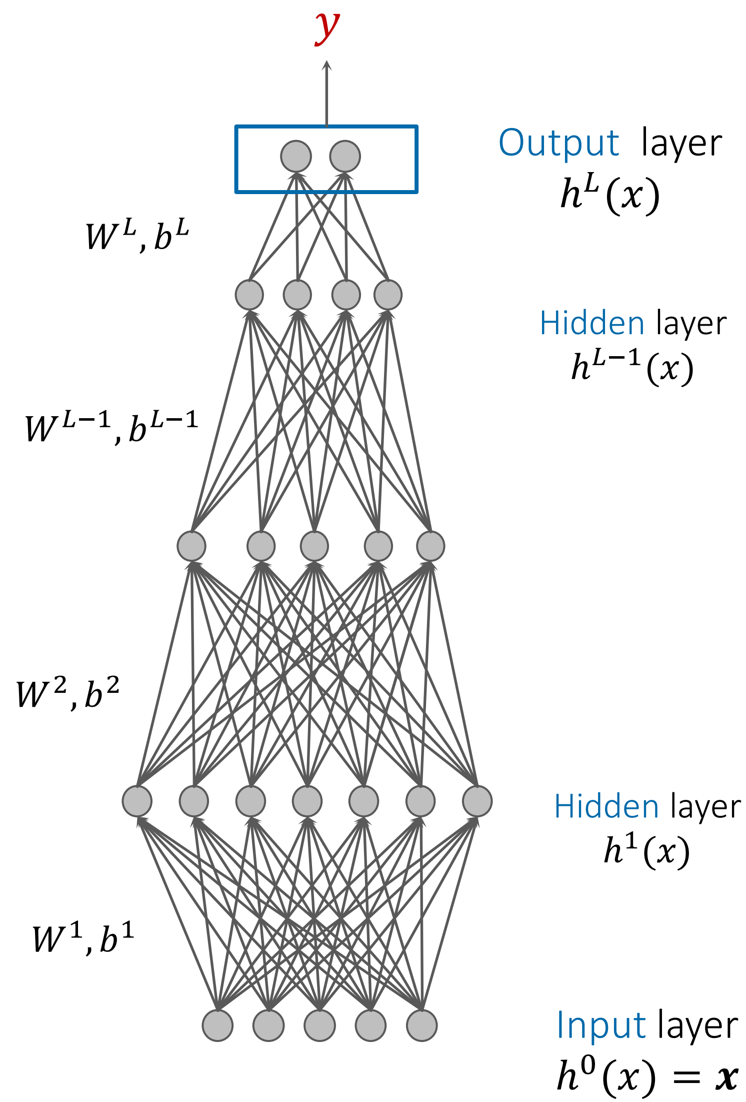
- Pre-Deep Learning Models:
  - Typically exploit or make assumption on the model structures to facilitate inference and model training.
  - Hidden Markov Models (discrete state)
    - Factorial Hidden Markov Models, Coupled HMM, Hierarchical HMMs
  - State-space models (Kalman filters, continuous states):
    - Hidden state is a continuous random variable, usually with some Gaussian assumptions
  - Dynamic Bayesian Networks
  - Conditional Random Fields (CRFs), etc

- Deep learning:
  - Leverage on the power of DNNs to capture time-varying dependency
  - Autoregressive models, Deep Recurrent Neural Networks
  - Long Short-Term Memory (LSTM)
  - Seq2Seq models, etc.

Recurrent NNs (RNNs)

DNN and CNNs



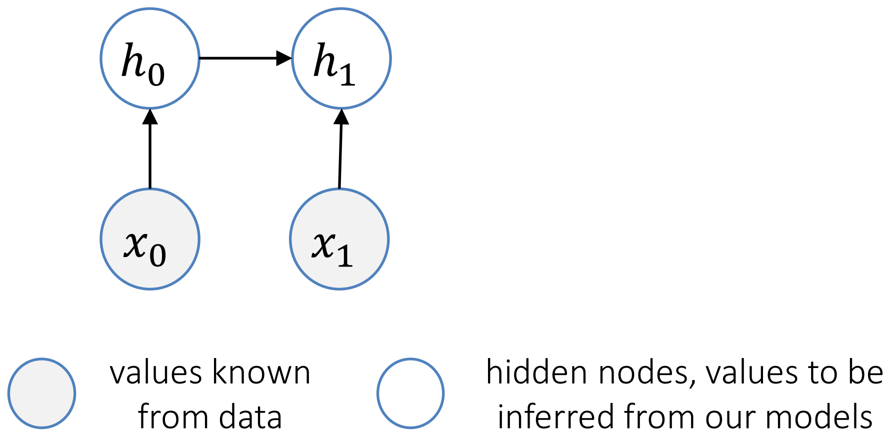


- DNNs: build networks for deep static structures
- CNNs: mainly builds network to exploit spatial patterns
- How to build network for sequences?

Recurrent Neural Networks

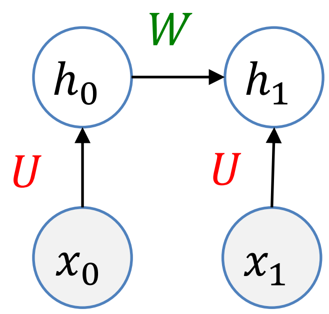
[Rumelhart, et al., 1986]

Simplest RNN with two time-slices (no output)



Recurrent Neural Networks

Simplest RNN with two time-slices (<u>no output</u>)



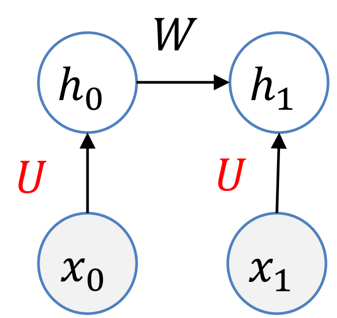
values known from data

hidden nodes, values to be inferred from our models

- Input: $x_0 \in \mathbb{R}^{in\_size}$, $x_1 \in \mathbb{R}^{in\_size}$
- $h_0 = \tanh(Ux_0 + b) \in \mathbb{R}^{hidden\_size}$
  - $U \in \mathbb{R}^{hidden\_size \times in\_size}$
- $h_1 = \text{some function of } h_0 \text{ and } x_1$
  $= \tanh(Wh_0 + Ux_1 + b)$

Recurrent Neural Networks

Simplest RNN with two time-slices (no output)



values known from data

hidden nodes, values to be inferred from our models

- Input: $x_0 \in \mathbb{R}^{in\_size}$, $x_1 \in \mathbb{R}^{in\_size}$
- $h_0 = \tanh(Ux_0 + b) \in \mathbb{R}^{hidden\_size}$
  - $U \in \mathbb{R}^{hidden\_size \times in\_size}$
- $h_1 =$ some function of h_0 and x_1
  $= \tanh(Wh_0 + Ux_1 + b)$

```
import numpy as np

X0 = np.array([[0.0, 1.0, -2.0],
               [-3.0, 4.0, 5.0],
               [6.0, 7.0, -8.0],
               [6.0, -1.0, 2.0], dtype= np.float32) # t = 0
X1 = np.array([[9.0, 8.0, 7.0],
               [0.0, 0.0, 0.0],
               [6.0, 5.0, 4.0],
               [1.0, 2.0, 3.0]], dtype= np.float32) # t = 1
```

batch_size = 4

```
hidden_size = 5
input_size = 3

U = tf.Variable(tf.random.normal(shape=[input_size, hidden_size],dtype=tf.float32))
W = tf.Variable(tf.random.normal(shape=[hidden_size, hidden_size],dtype=tf.float32))
b = tf.Variable(tf.zeros([1, hidden_size], dtype=tf.float32))

h0 = tf.tanh(tf.matmul(X0, U) + b)
h1 = tf.tanh(tf.matmul(X1, U) + tf.matmul(h0, W)  + b)
```

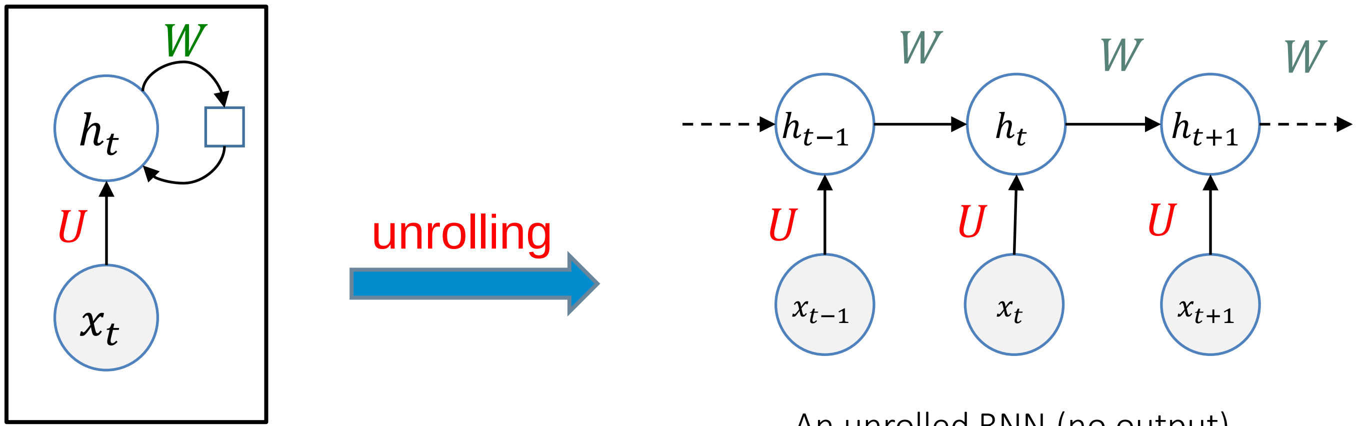
```
print("h0= {}".format(h0.numpy()))
```

```
h0= [[-0.92953503 -0.9916096  -0.9950185   0.90235376 -0.03616969]
 [-0.99999934  1.         -0.98861784 -0.99981123 -0.99951625]
 [-1.         -0.99999905 -1.          0.63526183  0.80846786]
 [ 0.88406056  0.99978167  0.9999999  -0.99995536  0.9902843 ]]
```

```
print("h1= {}".format(h1.numpy()))
```

Recurrent Neural Networks

with no output



An unrolled RNN (no output)

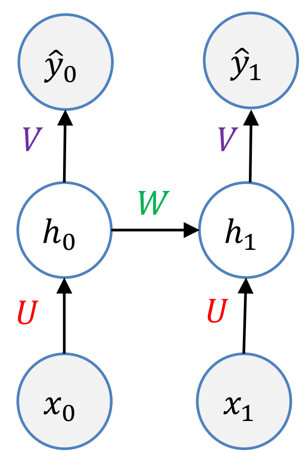
- Idea: sharing parameters for each data of the time index
- Given a data sequence $x_1, x_2, \ldots, x_T$
- RNN models a dynamic system driven by an external signal x_t

$$h_t = f(h_{t-1}, x_t) = f(f(h_{t-2}, x_{t-1}), x_t) = \cdots = \text{summary}(x_{1:t}, h_0)$$

- h_t can be considered as a kind of lossy summary of the history $x_{1:t}$

Recurrent Neural Networks

Parameterization - 2 time slices

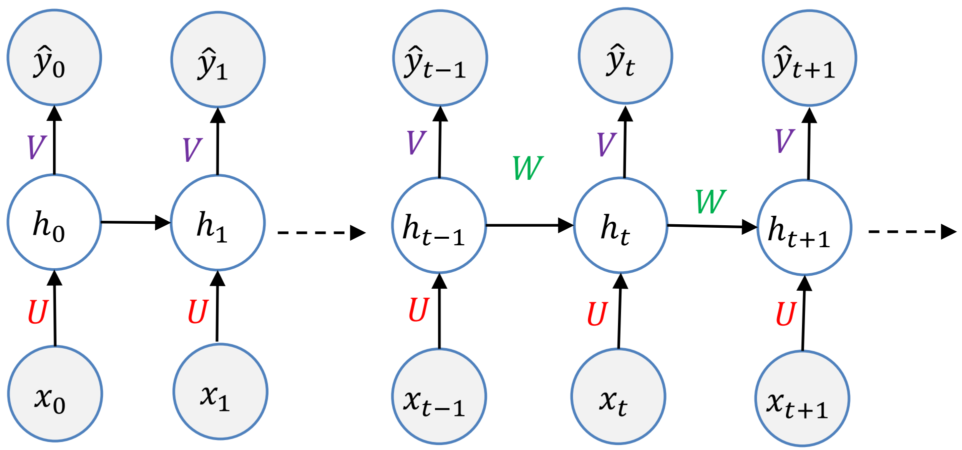


Simplest RNN with two time-slices (with output)

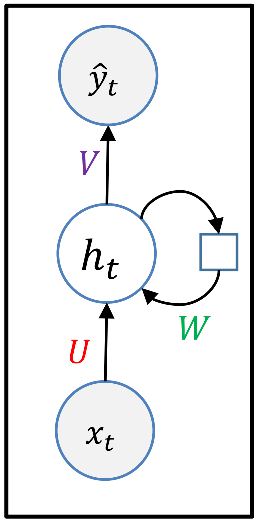
- Input: $x_0, x_1 \in \mathbb{R}^{in\_size}$, $y_0, y_1 \in Y$
- $h_0 = \tanh(Ux_0 + b)$
- $\hat{y}_0 = \begin{cases} Vh_0 + c \quad (\text{regression}) \\ \text{softmax}\,(Vh_0 + c) \quad (\text{classification}) \end{cases}$
  - Suffer loss $l(\hat{y}_0, y_0)$
- $h_1 = \text{some function of } h_0 \text{ and } x_1$
  $= \tanh(Wh_0 + Ux_1 + b)$
- $\hat{y}_1 = \begin{cases} Vh_1 + c \quad (\text{regression}) \\ \text{softmax}\,(Vh_1 + c) \quad (\text{classification}) \end{cases}$
  - Suffer loss $l(\hat{y}_1, y_1)$

Recurrent Neural Networks

Parametrization over multiple time slices



Simplest RNN with multiple time-slices (with output)

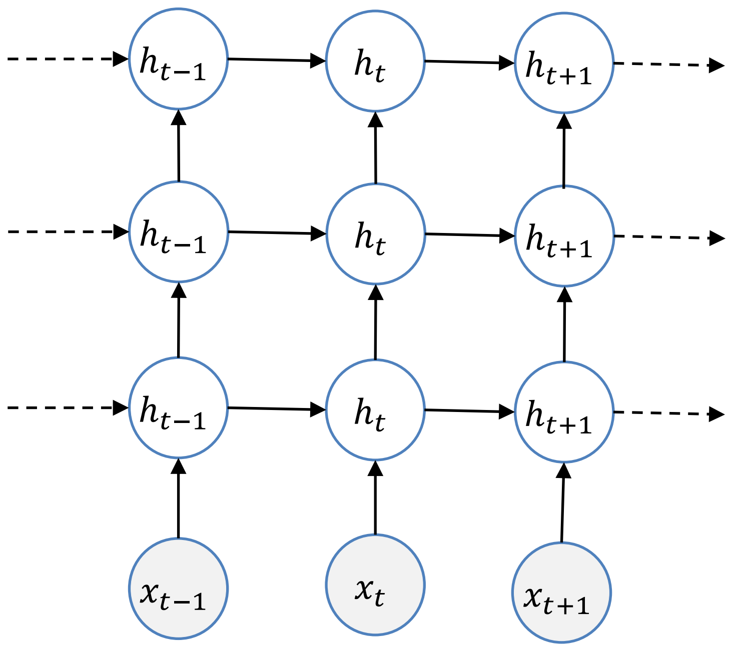


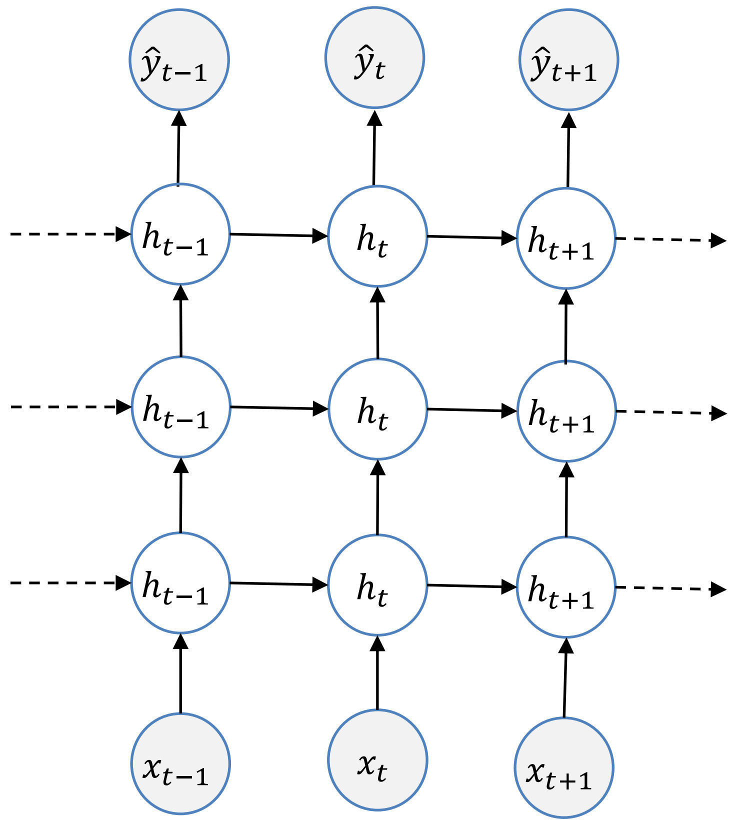
- **Input**: $x_0, x_1, x_2, .., x_t, \ldots \in \mathbb{R}^{in\_size}$, $y_0, y_1, \ldots \in Y$
- $h_0 = \tanh(Ux_0 + b)$
- $\hat{y}_0 = \begin{cases} Vh_0 + c & \text{(regression)} \\ \text{softmax}\,(Vh_0 + c) & \text{(classification)} \end{cases}$
  - Suffer loss $l(\hat{y}_0, y_0)$
- **for** t=1,2,...

$$h_t = \text{some function of } h_{t-1} \text{ and } x_t$$
$$= \tanh(Wh_{t-1} + Ux_t + b)$$
$$\hat{y}_t = \begin{cases} Vh_t + c & \text{(regression)} \\ \text{softmax}\,(Vh_t + c) & \text{(classification)} \end{cases}$$

  - Suffer loss $l(\hat{y}_t, y_t)$

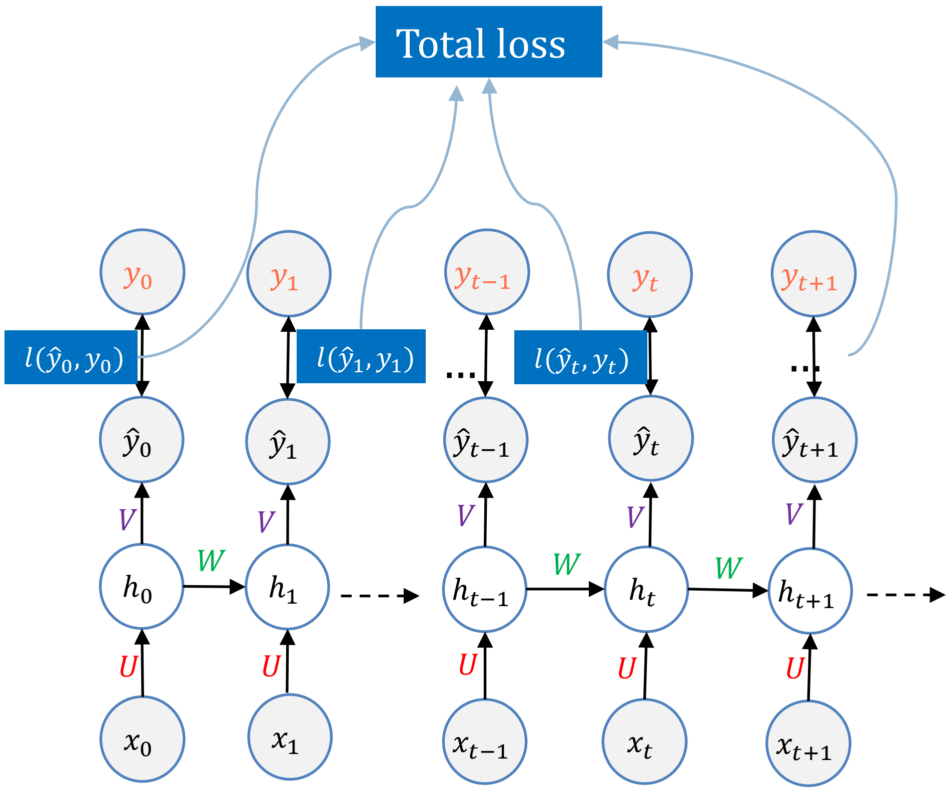
Deeper RNNs


h_{t-1}
h_t
h_{t+1}
h_{t-1}
h_t
h_{t+1}
h_{t-1}
h_t
h_{t+1}
x_{t-1}
x_t
x_{t+1}


$\hat{y}_{t-1}$
$\hat{y}_t$
$\hat{y}_{t+1}$
h_{t-1}
h_t
h_{t+1}
h_{t-1}
h_t
h_{t+1}
h_{t-1}
h_t
h_{t+1}
x_{t-1}
x_t
x_{t+1}

Training RNNs with Back Propagation Through Time (BPTT)

Forward propagation through the time



- Input: $x_0, x_1, x_2, ..., x_t, ... \in \mathbb{R}^{in\_size}$, $y_0, y_1, ..., y_t, ... \in Y$
- $\bar{h}_0 = Ux_0 + b, h_0 = \tanh(\bar{h}_0)$
- $\hat{\boldsymbol{y}}_0 = \begin{cases} Vh_0 + c & \text{(regression)} \\ \text{softmax}\,(Vh_0 + c) & \text{(classification)} \end{cases}$
  - Suffer loss $\text{loss}_0 = l(\hat{y}_0, y_0)$
- for t=1,2,...

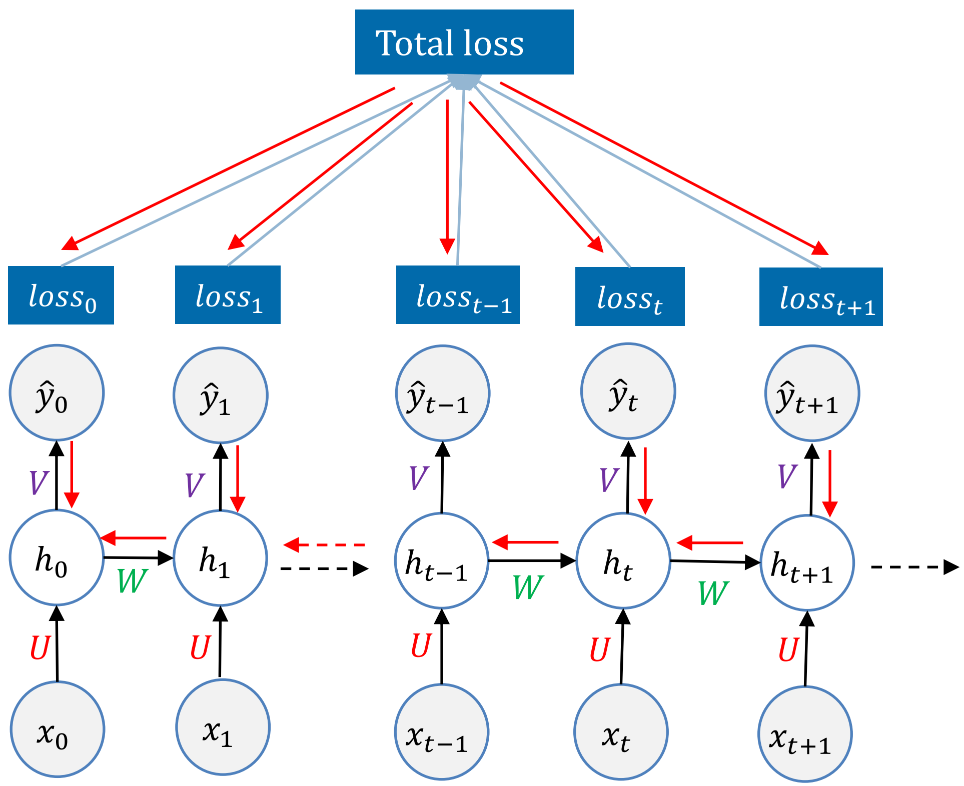
  Pre-activation: $\bar{h}_t = Wh_{t-1} + Ux_t + b$

  After-activation: $h_t = \tanh(\bar{h}_t)$

  Prediction: $\hat{\boldsymbol{y}}_t = \begin{cases} Vh_t + c & \text{(regression)} \\ \text{softmax}\,(Vh_t + c) & \text{(classification)} \end{cases}$
  - Suffer loss: $\text{loss}_t = l(\hat{y}_t, y_t)$
- Total loss
  - $Total\ loss = \sum_t loss_t$

Back Propagation Through Time

Not in assessment



Pre-activation: $\bar{h}_t = W h_{t-1} + U x_t + b$

After-activation: $h_t = \tanh(\bar{h}_t)$

Prediction: $\hat{y}_t = \begin{cases} V h_t + c & \text{(regression)} \\ \text{softmax}\,(V h_t + c) & \text{(classification)} \end{cases}$

Suffer loss $loss_t = l(\hat{y}_t, y_t)$

- Total loss
  - $L = \sum_t loss_t = \sum_t l_t$
- Let us compute $\frac{\partial L}{\partial h_0}$?

- $$\frac{\partial L}{\partial h_0} = \sum_t \frac{\partial l_t}{\partial h_0}$$
  $$= \sum_t \frac{\partial l_t}{\partial \hat{y}_t} \times \frac{\partial \hat{y}_t}{\partial h_t} \times \frac{\partial h_t}{\partial \bar{h}_t} \times \frac{\partial \bar{h}_t}{\partial h_{t-1}} \dots \frac{\partial h_1}{\partial \bar{h}_1} \times \frac{\partial \bar{h}_1}{\partial h_0}$$

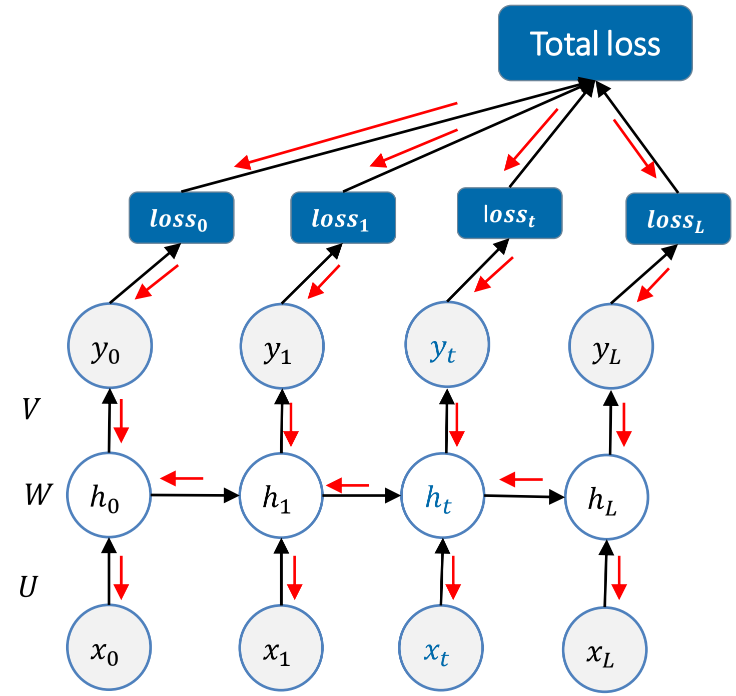
- $$\frac{\partial L}{\partial h_0} = \sum_t \frac{\partial l_t}{\partial \hat{y}_t} \times \frac{\partial \hat{y}_t}{\partial h_t} \times diag(1 - \bar{h}_t^2) W \dots diag(1 - \bar{h}_1^2) W$$

- Multiplying W multiple times
- Gradient vanishing exploding problem

Training RNN

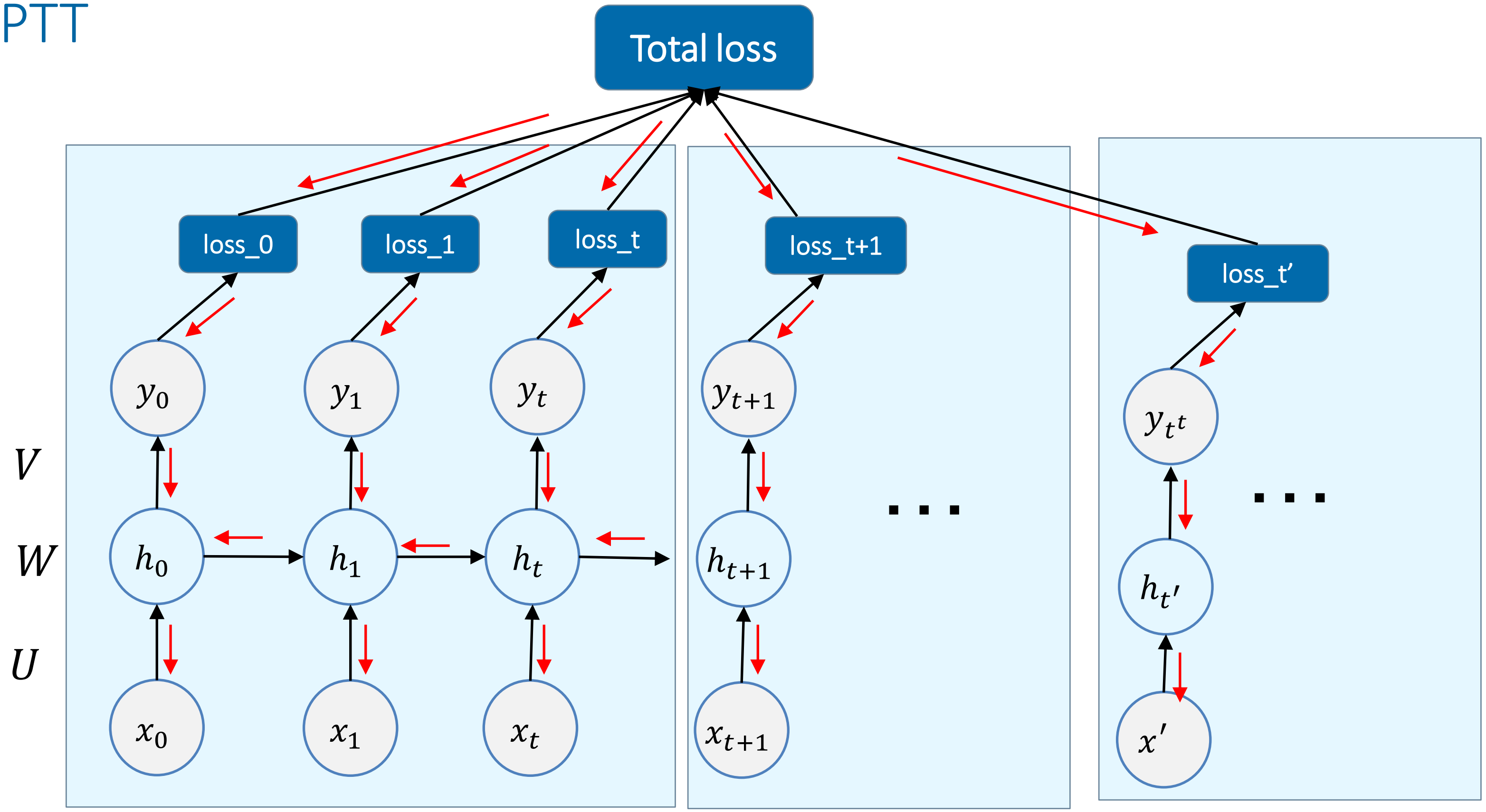
Summary

- Use BackProp Through Time (BPTT)
  - Create unrolled network
  - Forward pass, store values and calculate loss at each time slide
  - Backward propagation through time, starting from the last step, to calculate the local gradients at each time step.
  - Sum up local gradients to obtain the end gradients for each parameters
  - Update the parameters via GD/SGD



Not in assessment

Training RNN
Truncated BPTT



- Truncated BPTT
  - For long sequence, divide the network into consecutive segments, perform BTT within each segment

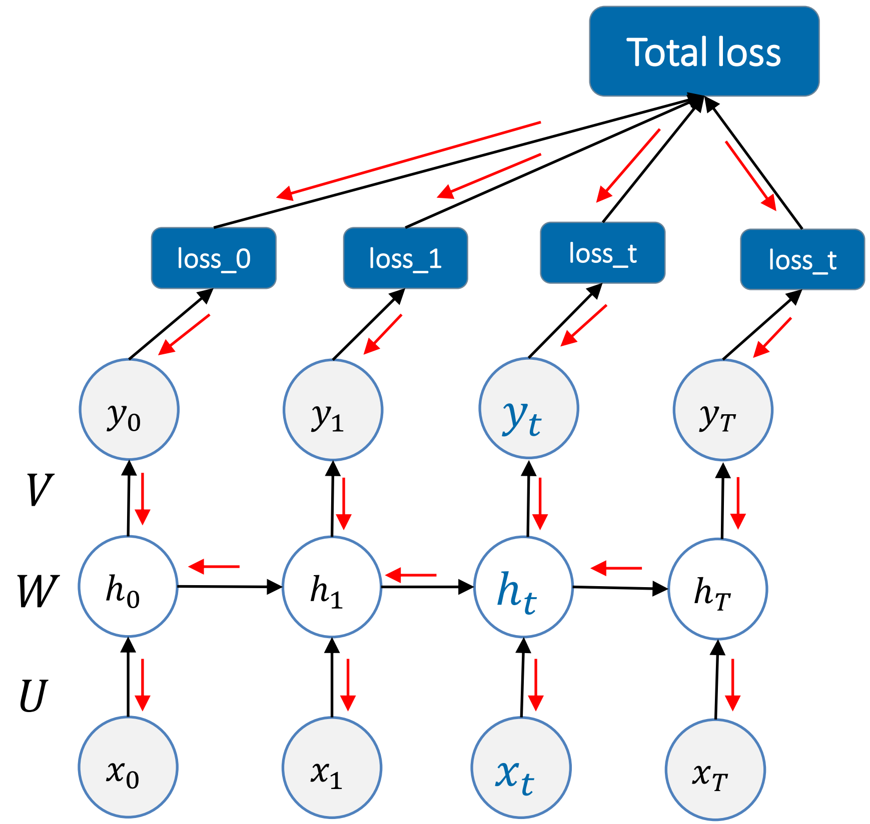
Not in assessment

Training RNN

- Challenges with training RNN:
  - Without using activation function the values can either explode or diminish quickly depending on the spectrum of U, V, W

    $$h_t = W^\top h_{t-1} = (W^t)^\top h_0 = [(Q\Lambda Q^\top)^t]^\top h_t$$
    $$= Q\Lambda^t Q^\top h_0$$

  - Eigenvalues in Λ will dictate the behaviours (why?)
  - Gradient vanishing/exploding
    - $\frac{\partial L}{\partial h_0} = \sum_t \frac{\partial loss_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial h_t} diag(1 - \bar{h}_t^2) W \ldots diag(1 - \bar{h}_1^2) W$



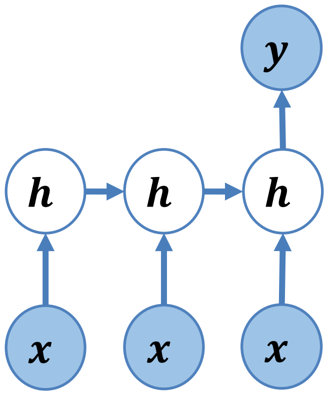
- How to address this?
  - Use tanh activation as **the squash function to scale the output to (-1, 1)** at each time step.
  - But, we can still run into the gradient vanishing problem. Solution: **LSTM, GRU cells**

Not in assessment

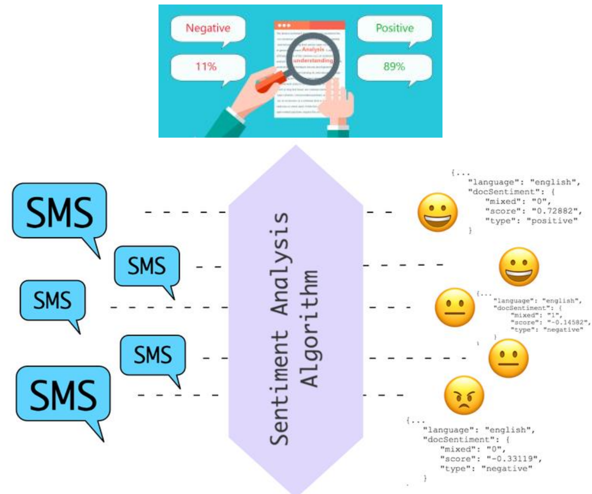
RNNs applications

RNNs Architecture Zoo

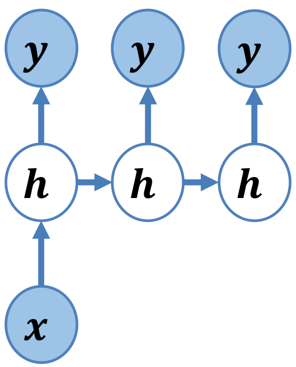
- RNNS architecture is very flexible for many real-world tasks



many to one
(sentiment analysis, image classification)



Sentiment analysis
[https://www.twilio.com]



one to many
(image captioning)

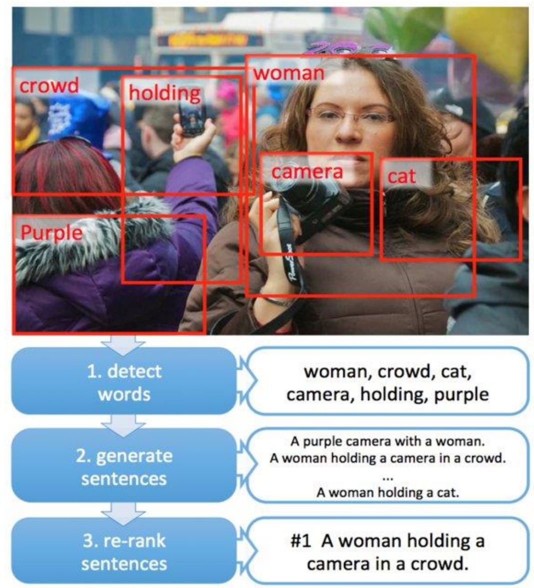
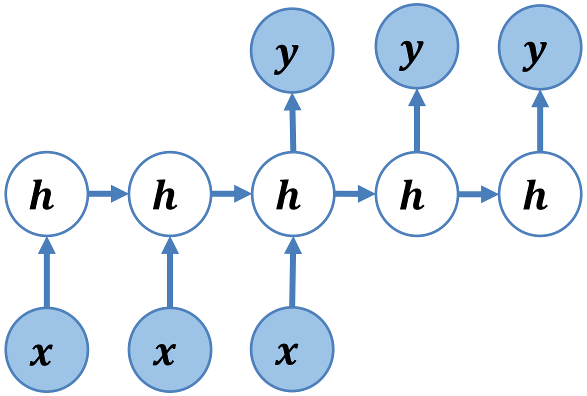


Image captioning
[https://www.pcworld.com]

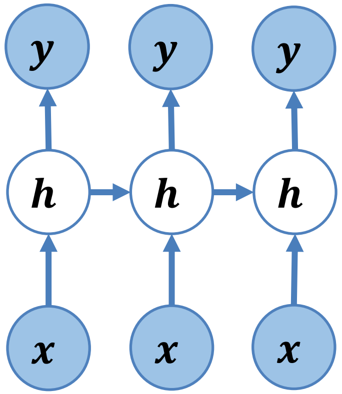
RNNs Architecture Zoo

- RNNS architecture is very flexible for many real-world tasks



many to many (1)

(machine translation)



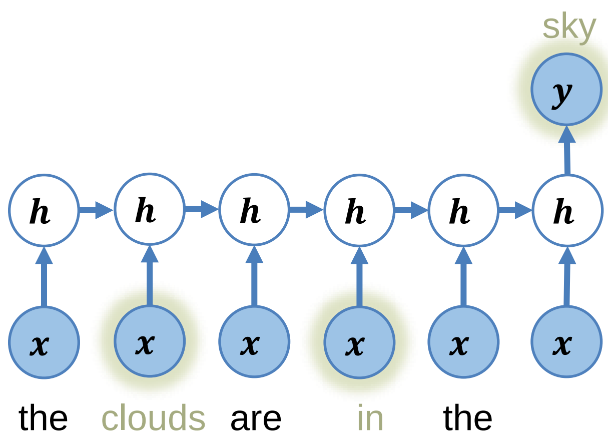
many to many (2)

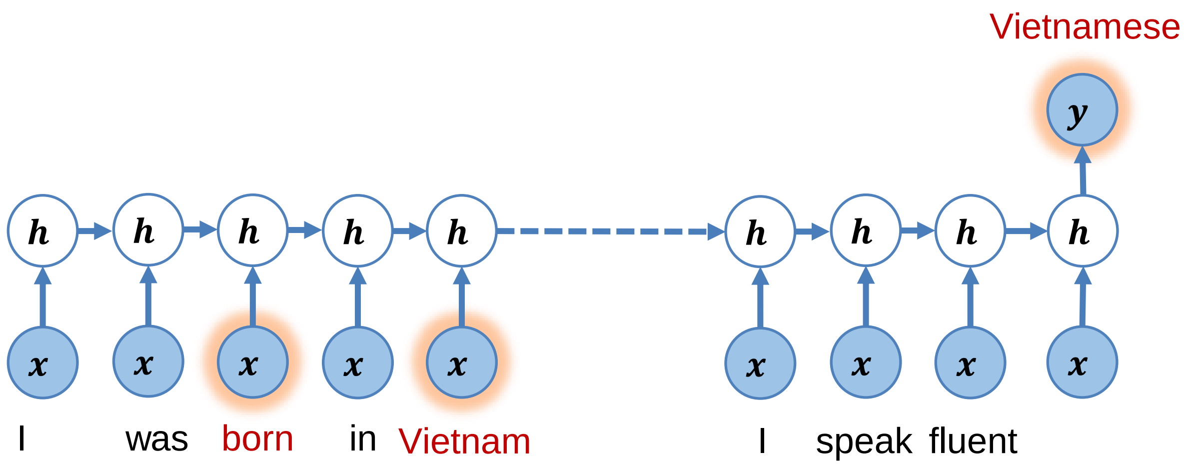
(video classification)

... more to come in our next lectures

Problem of simple RNN and rethinking the memory cell

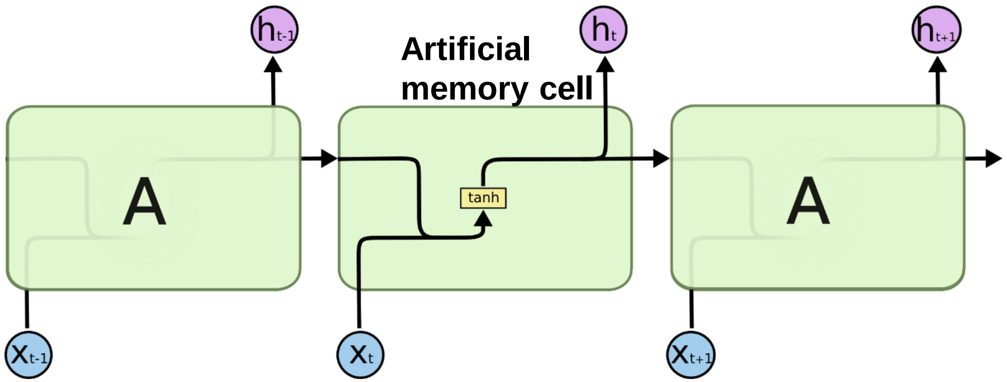
Problems of RNN



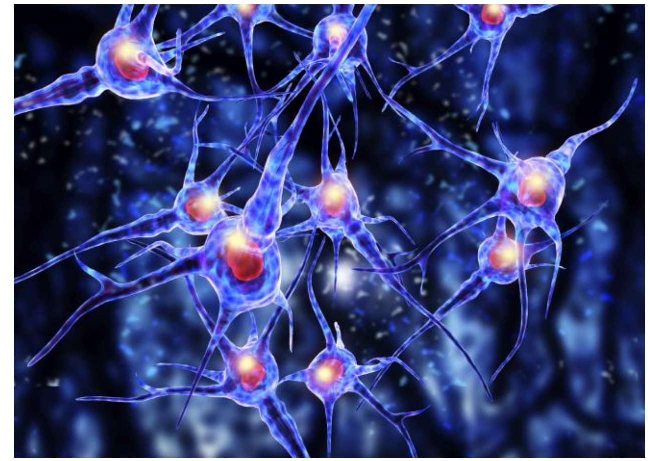


- RNNs **don't** capture **long-term dependency** adequately
  - A hidden state is computed based on only one previous state → can only capture **short-term dependency**
- Modelling drawbacks
  - Technical problem when training long sequences
    - vanishing gradient problem
  - Many layers of nonlinear transformation prevent the data signals and gradient from flowing easily through the network.
- How to address this?
  - Using gating mechanism: adding linear component from previous layer!
    - RNN → LSTM/GRU

Memory cells

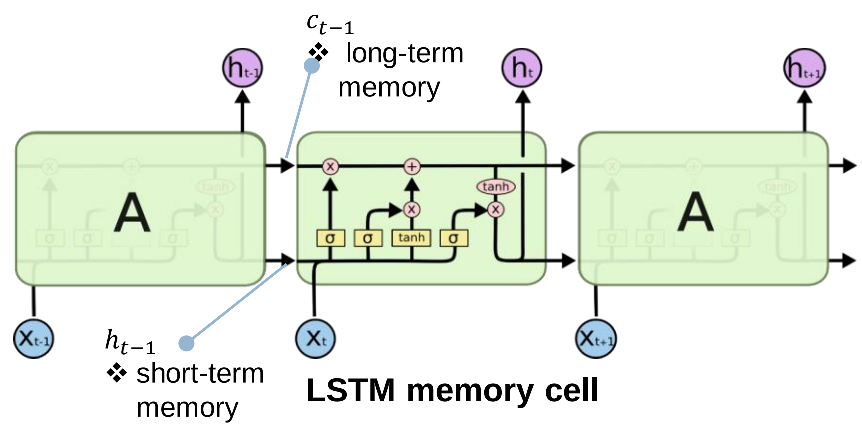


(Source: colah)



Biological **memory cells** in human brain
(Source: news.feinberg.northwestern.edu)

- Our RNN includes many **simple RNN cells**
  - **Input to a cell**: h_{t-1} (previous hidden state) and x_t (current input token)
  - **Output**: $h_t = \tanh(Ux_t + Wh_{t-1} + b)$
  - h_t can only capture **short-term dependency** → **short-term memory**
- How to capture **long-term memory** more efficiently?
  - **LSTM** cell and **GRU** cell

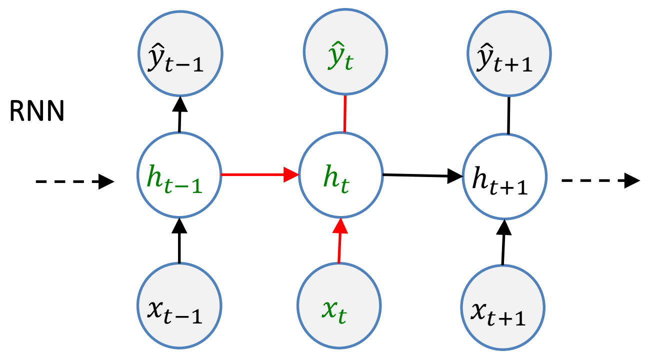


Long-Short Term Memory Models (LSTM)

[Hochreiter and Schimdhuber '97]

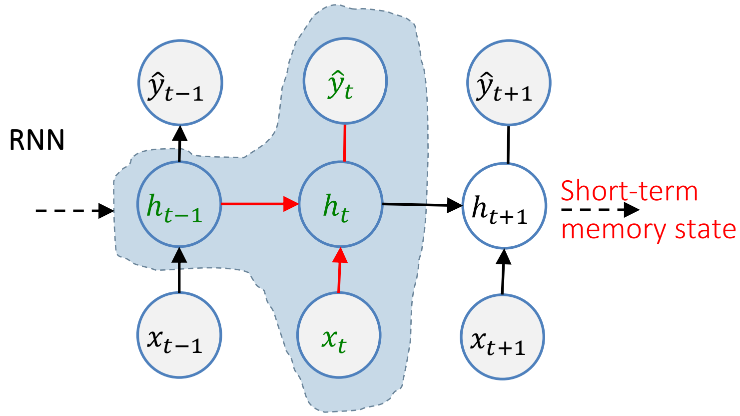
Long Short-Term Memory (LSTM)

[Hochreiter and Schimdhuber '97]



Long Short-Term Memory (LSTM)

[Hochreiter and Schimdhuber '97]

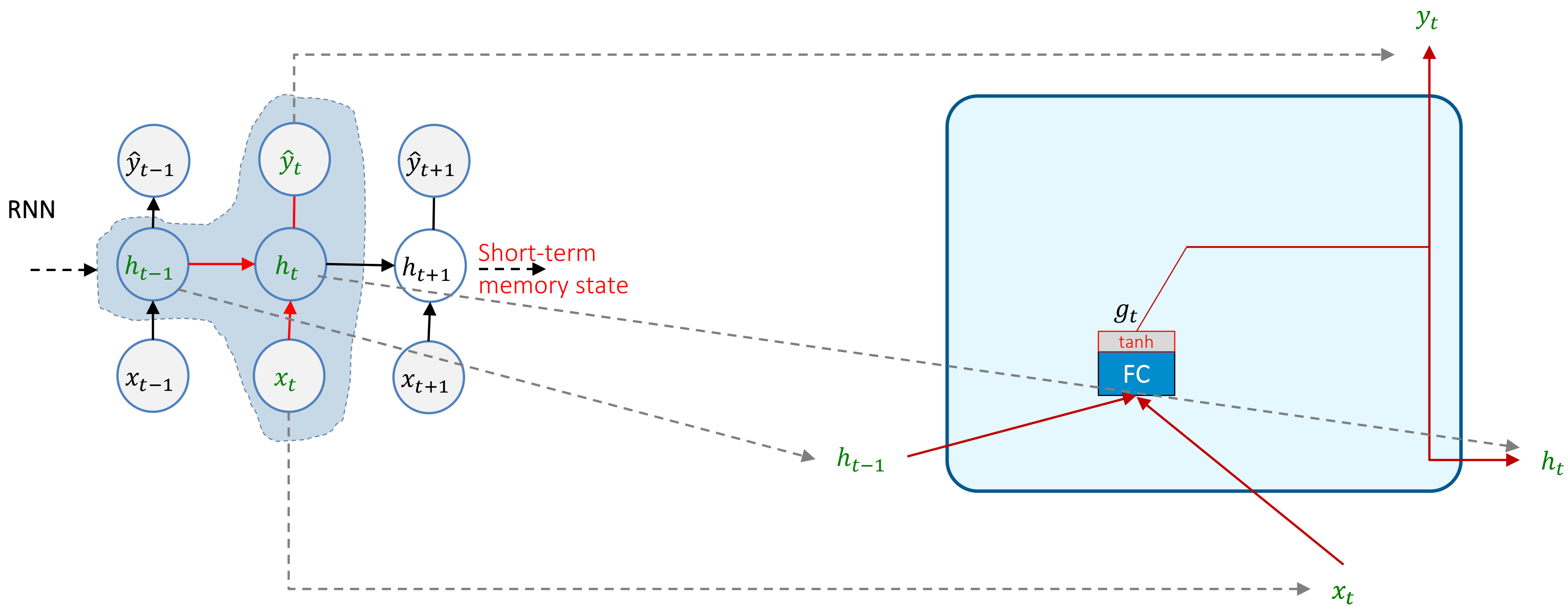


$$h_t = f(h_{t-1}, x_t) = f(f(h_{t-2}, x_{t-1}), x_t) = \cdots = \text{summary}(x_{1:t}, h_0)$$

- $h_t = \tanh(W h_{t-1} + U x_t + b)$
- $\hat{y}_t = \text{softmax}(V h_t + c)$

Long Short-Term Memory (LSTM)

[Hochreiter and Schimdhuber '97]

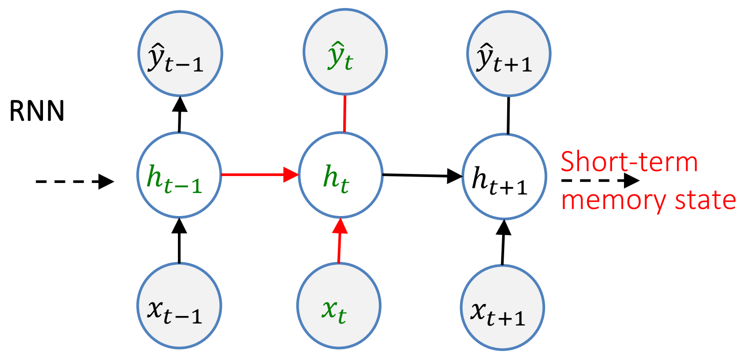


- $h_t = \tanh(Wh_{t-1} + Ux_t + b)$
- $\hat{y}_t = \text{softmax}(Vh_t + c)$

- $g_t = \tanh(Wh_{t-1} + Ux_t + b)$
- Short-term memory: $h_t = g_t$
- $\hat{y}_t = \text{softmax}(Vg_t + c)$

Long Short-Term Memory (LSTM)

[Hochreiter and Schmiddhuber '97]

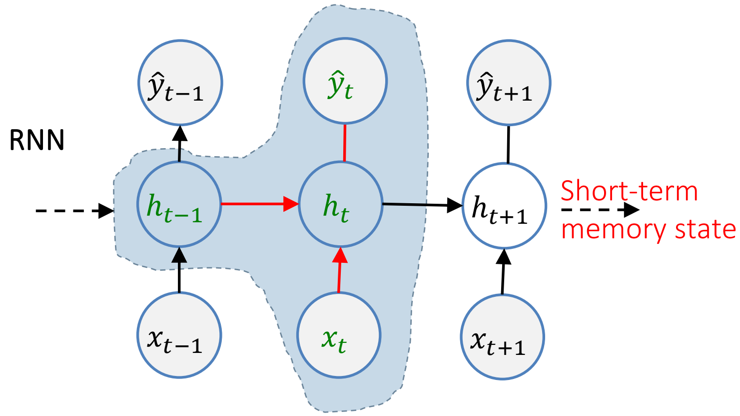


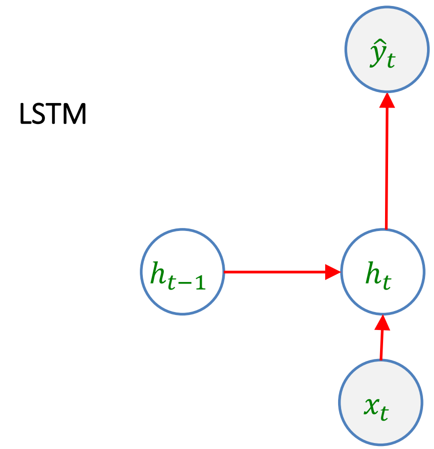
LSTM

- Introduced in 1997 by Hohreiter and Schmidhuber; improved over the years: Sak et. al 2014, Zaremba, 2015, etc.
- <u>Address the long-term dependency problem by introducing a long-term state memory c_t</u>
- Help the gradient flow significantly over a long duration, hence capture long-term dependency

Long Short-Term Memory (LSTM)

[Hochreiter and Schmiddhuber '97]





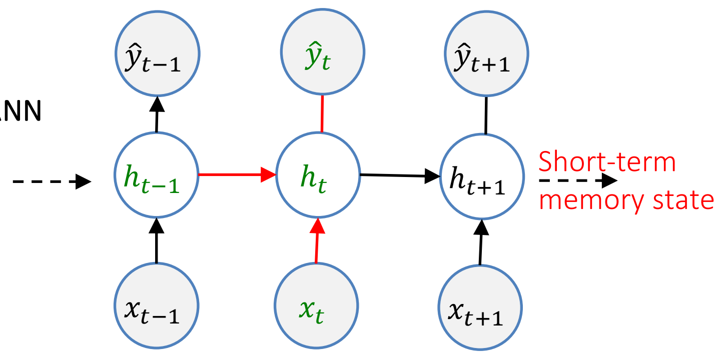
LSTM

- Introduced in 1997 by Hohreiter and Schmidhuber; improved over the years: Sak et. al 2014, Zaremba, 2015, etc.
- Address the long-term dependency problem by introducing a long-term state memory c_t
- Help the gradient flow significantly over a long duration, hence capture long-term dependency

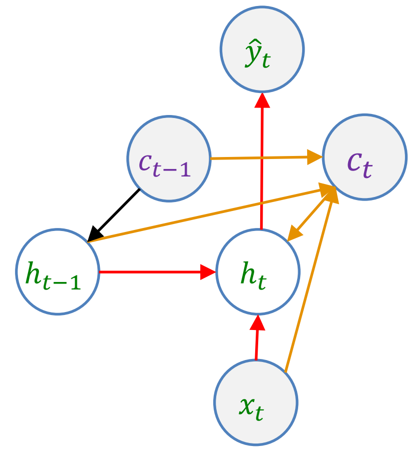
Long Short-Term Memory (LSTM)

[Hochreiter and Schmiddhuber '97]

RNN



LSTM

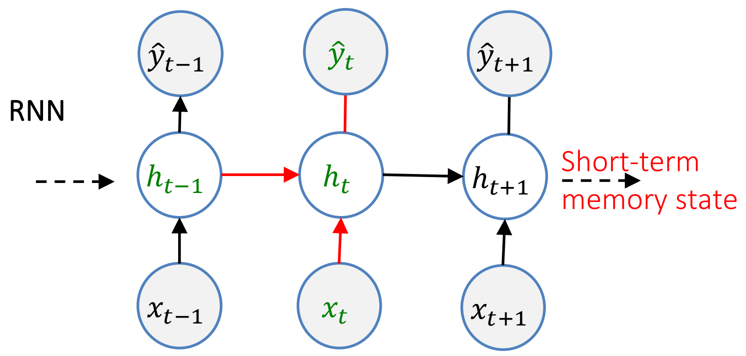


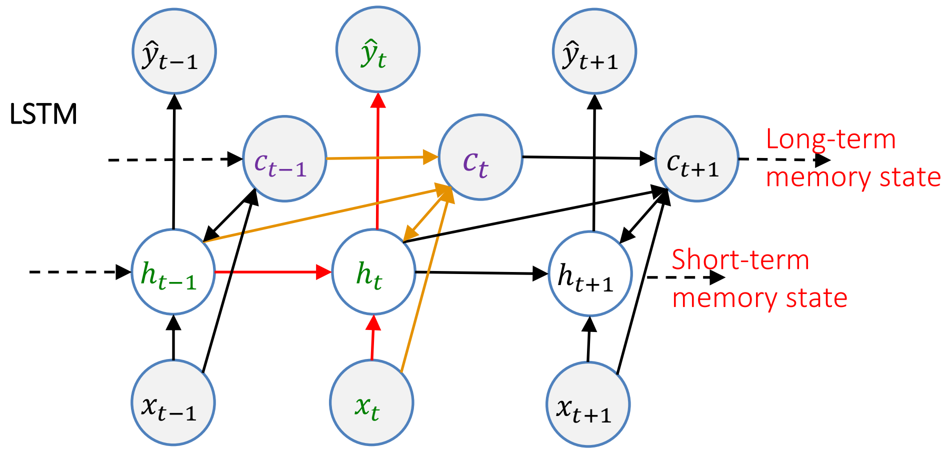
LSTM

- Introduced in 1997 by Hohreiter and Schmidhuber; improved over the years: Sak et. al 2014, Zaremba, 2015, etc.
- Address the long-term dependency problem by introducing a long-term state memory c_t
- Help the gradient flow significantly over a long duration, hence capture long-term dependency

Long Short-Term Memory (LSTM)

[Hochreiter and Schmiddhuber '97]



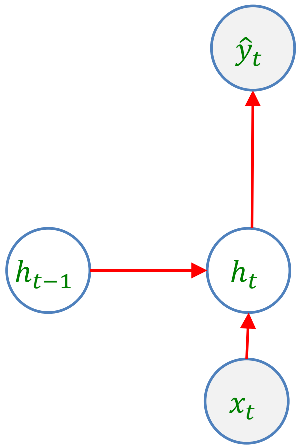


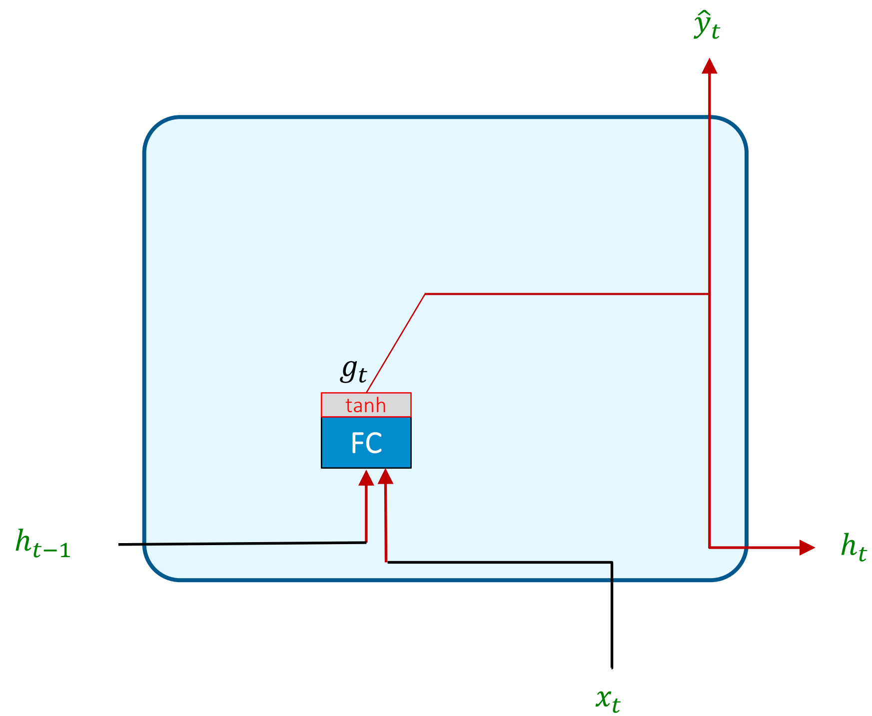
LSTM

- Introduced in 1997 by Hohreiter and Schmidhuber; improved over the years: Sak et. al 2014, Zaremba, 2015, etc.
- Address the long-term dependency problem by introducing a long-term state memory c_t
- Help the gradient flow significantly over a long duration, hence capture long-term dependency

Long Short-Term Memory (LSTM)

[Hochreiter and Schmiddhuber '97]



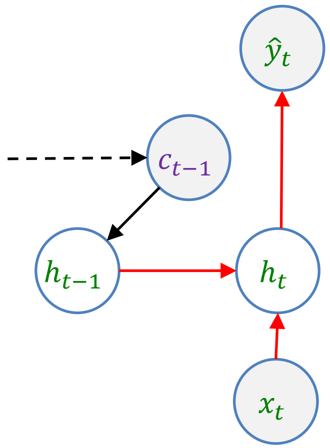


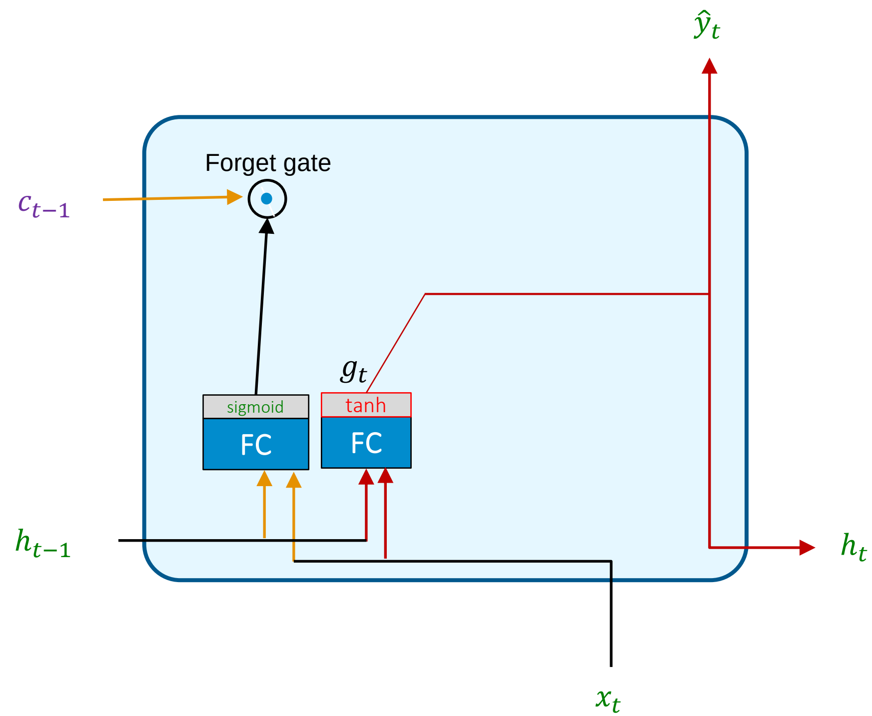
- $h_t = \tanh(Wh_{t-1} + Ux_t + b)$
- $\hat{y}_t = \text{softmax}(Vh_t + c)$

- $g_t = \tanh(Wh_{t-1} + Ux_t + b)$
- RNN short-term: $h_t = g_t$
- RNN output $y_t = \text{softmax}(Vg_t + c)$

Long Short-Term Memory (LSTM)

Forget Gate



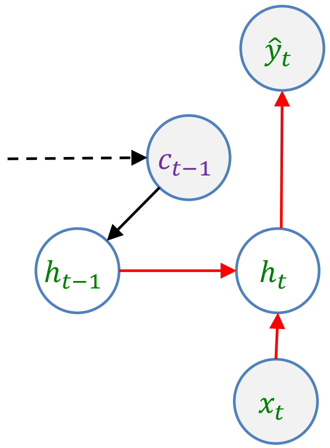


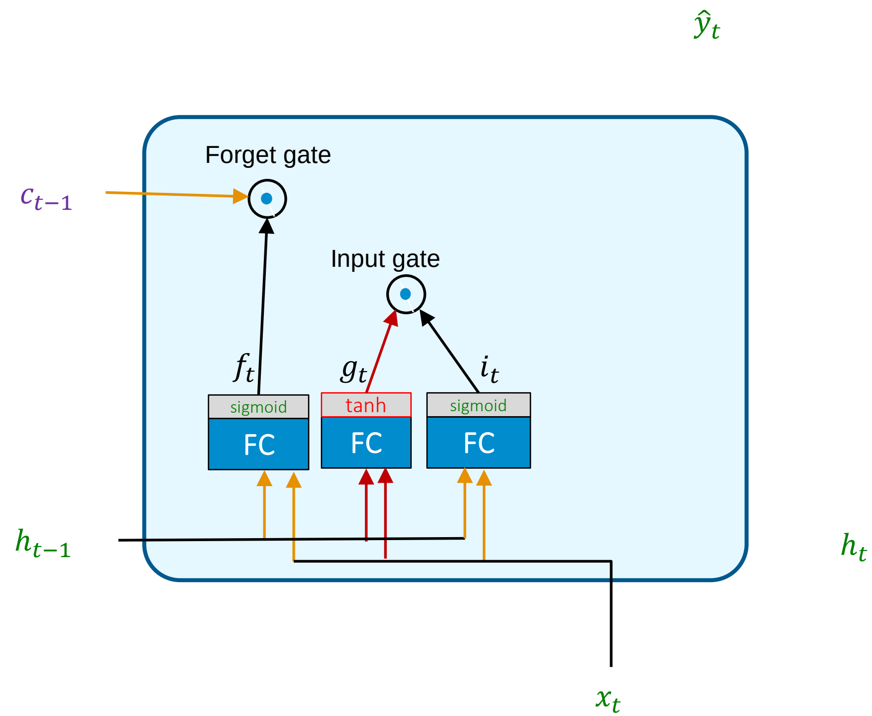
- Introduce three gate controllers:
  - Use sigmoid to ensure outputs' range between 0 and 1
  - Forget gate f_t: with element wise multiplication $\odot$ control which parts of long-term state c_{t-1} should be erased.

- $g_t = \tanh(W h_{t-1} + U x_t + b\ )$
- Forget gate: $f_t = \sigma\left(U^f x_t + W^f h_{t-1} + b^f\right)$

Long Short-Term Memory (LSTM)

Input Gate



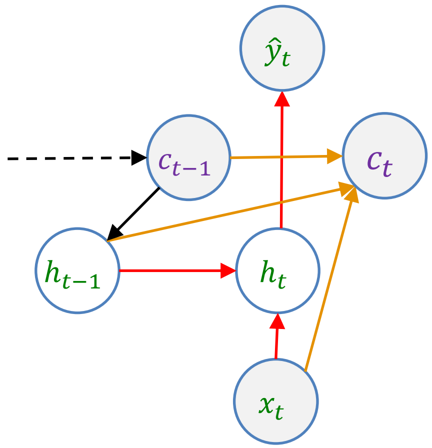


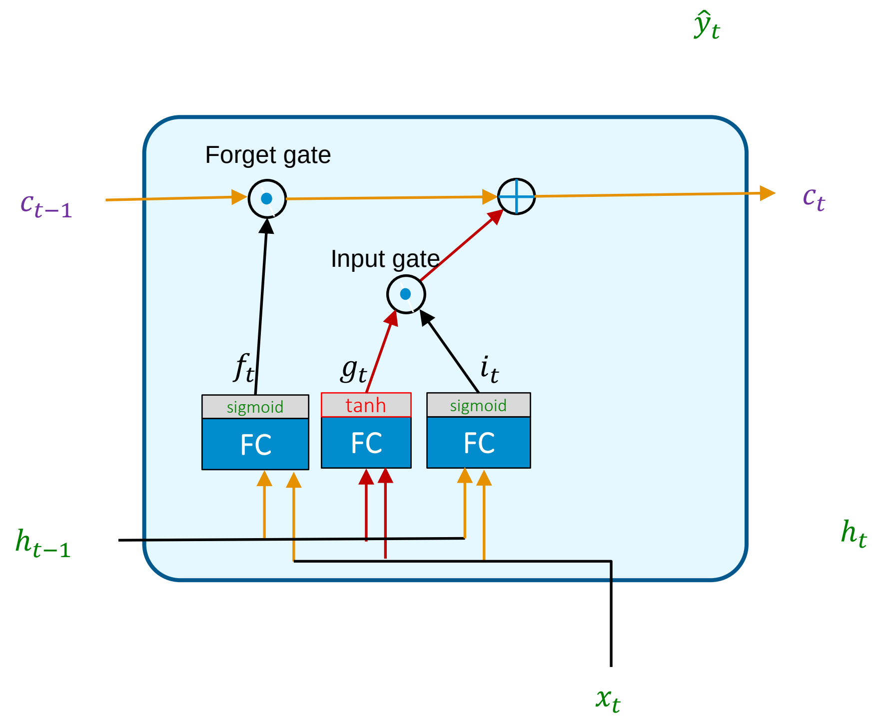
- Introduce three gate controllers:
  - Use sigmoid to ensure outputs' range between 0 and 1
  - Forget gate f_t controls how much c_{t-1} be 'forgotten'
  - Input gate i_t controls which how much g_t be remembered

- $g_t = \tanh(W h_{t-1} + U x_t + b\ )$
- Forget gate: $f_t = \sigma\left(U^f x_t + W^f h_{t-1} + b^f\right)$
- Input gate: $i_t = \sigma(U^i x_t + W^i h_{t-1} + b^i)$

Long Short-Term Memory (LSTM)

Long-term Cell State



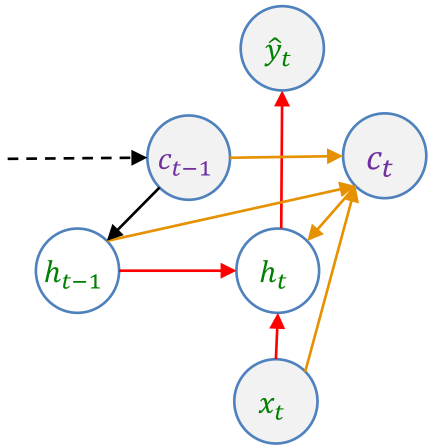


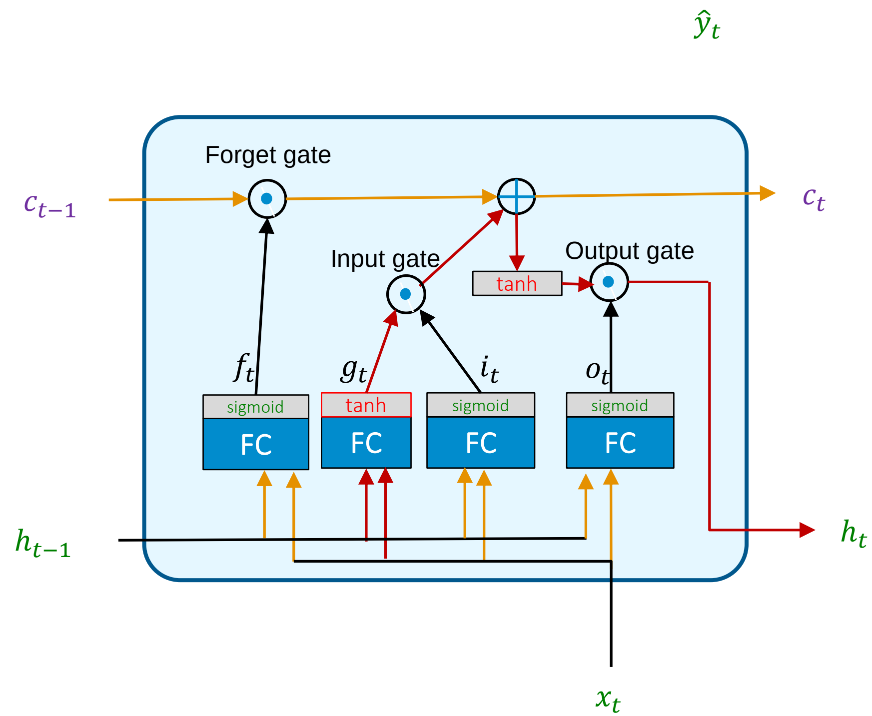
- Introduce three gate controllers:
  - Use sigmoid to ensure outputs' range between 0 and 1
  - Forget gate f_t controls how much c_{t-1} be 'forgotten'
  - Input gate i_t controls which how much g_t be remembered

- $g_t = \tanh(W h_{t-1} + U x_t + b)$
- Forget gate: $f_t = \sigma\left(U^f x_t + W^f h_{t-1} + b^f\right)$
- Input gate: $i_t = \sigma(U^i x_t + W^i h_{t-1} + b^i)$
- LSTM long-term state: $c_t = f_t \odot c_{t-1} + g_t \odot i_t$

Long Short-Term Memory (LSTM)

Output Gate



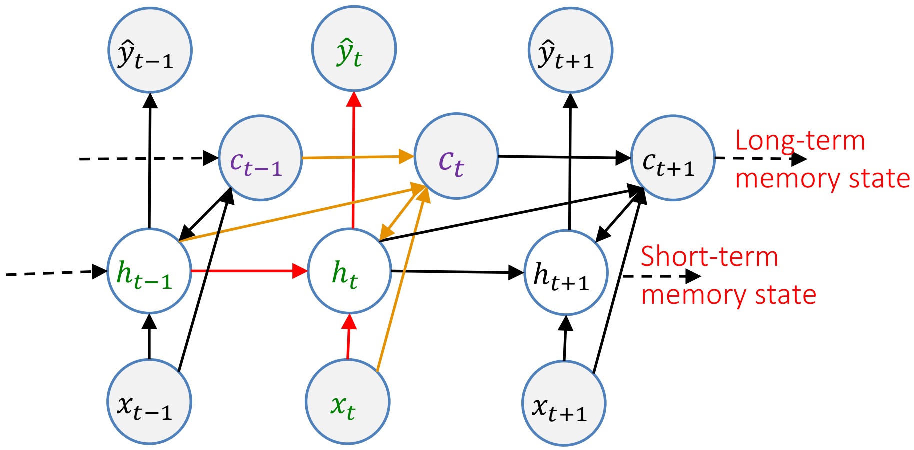


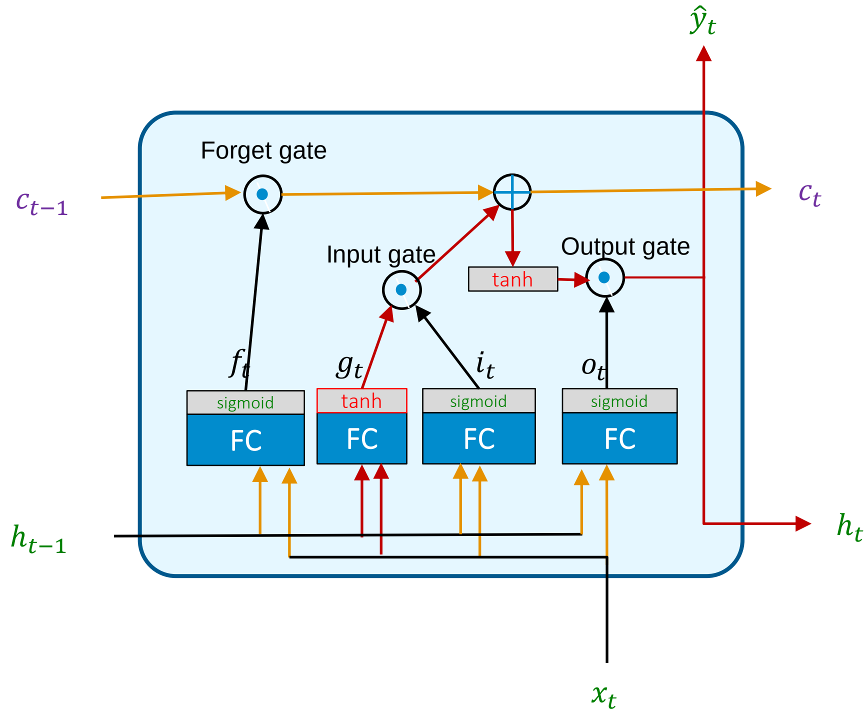
- Introduce three gate controllers:
  - Use sigmoid to ensure outputs' range between 0 and 1
  - Forget gate f_t controls how much c_{t-1} be 'forgotten'
  - Input gate i_t controls which how much g_t be remembered
  - Output gate o_t controls how much long-term c_t should be carried on to the next time slice:
    - to contribute to short-term state: h_t

- $g_t = \tanh(Wh_{t-1} + Ux_t + b\ )$
- Forget gate: $f_t = \sigma\left(U^f x_t + W^f h_{t-1} + b^f\right)$
- Input gate: $i_t = \sigma(U^i x_t + W^i h_{t-1} + b^i)$
- LSTM long-term state: $c_t = f_t \odot c_{t-1} + g_t \odot i_t$
- Output gate: $o_t = \sigma(U^o x_t + W^o h_{t-1} + b^o)$
- LSTM short-term state: $h_t = o_t \odot \tanh(c_t\ )$

Long Short-Term Memory (LSTM)

Output Gate



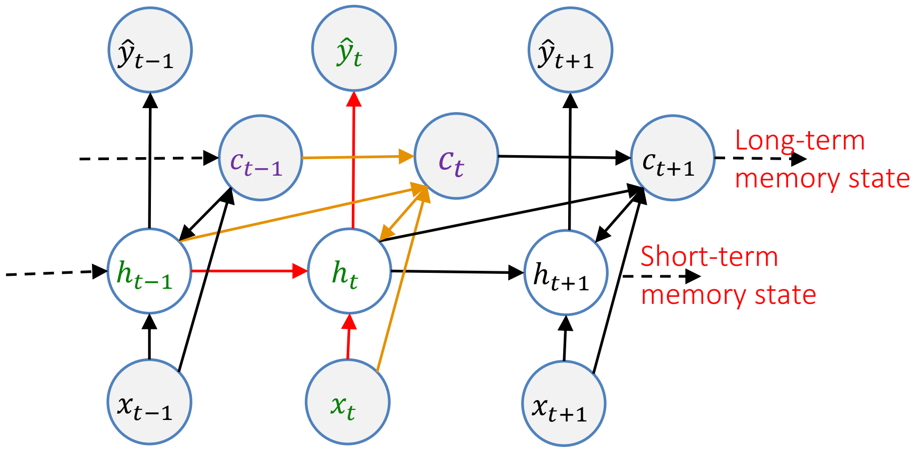


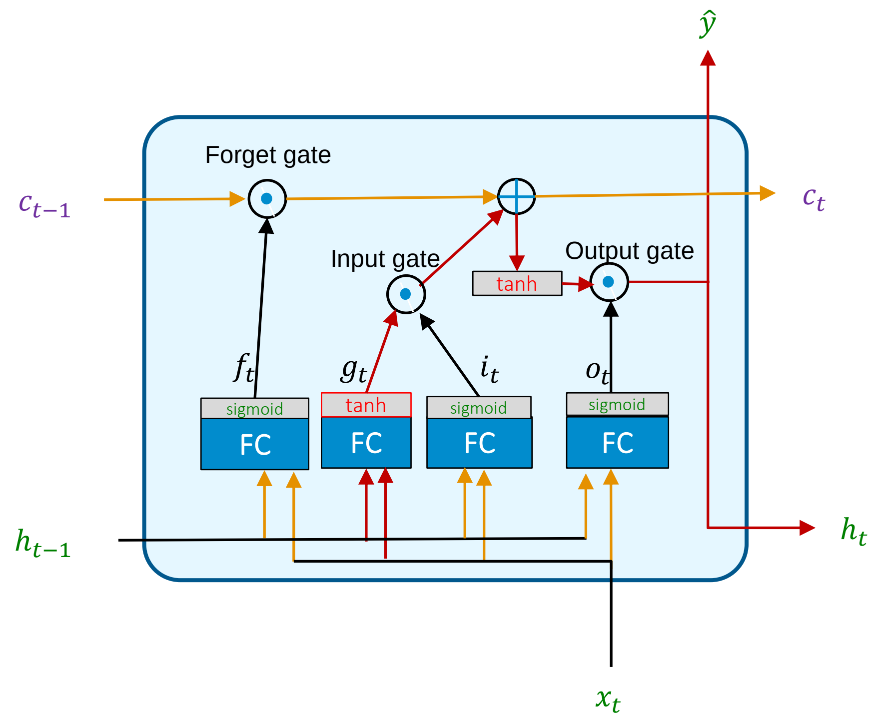
- Introduce three gate controllers:
  - Use sigmoid to ensure outputs' range between 0 and 1
  - Forget gate f_t controls how much c_{t-1} be 'forgotten'
  - Input gate i_t controls which how much g_t be remembered
  - Output gate o_t controls how much long-term c_t should be carried on to the next time slice:
    - to contribute to short-term state: h_t

- $g_t = \tanh(W h_{t-1} + U x_t + b)$
- Forget gate: $f_t = \sigma\left(U^f x_t + W^f h_{t-1} + b^f\right)$
- Input gate: $i_t = \sigma(U^i x_t + W^i h_{t-1} + b^i)$
- LSTM long-term state: $c_t = f_t \odot c_{t-1} + g_t \odot i_t$
- Output gate: $o_t = \sigma(U^o x_t + W^o h_{t-1} + b^o)$
- LSTM short-term state: $h_t = o_t \odot \tanh(c_t)$

Long Short-Term Memory (LSTM)

Output Gate



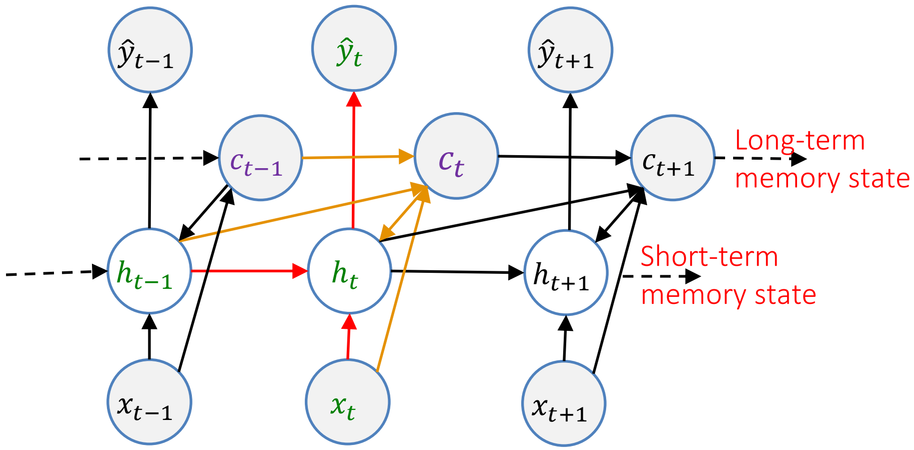


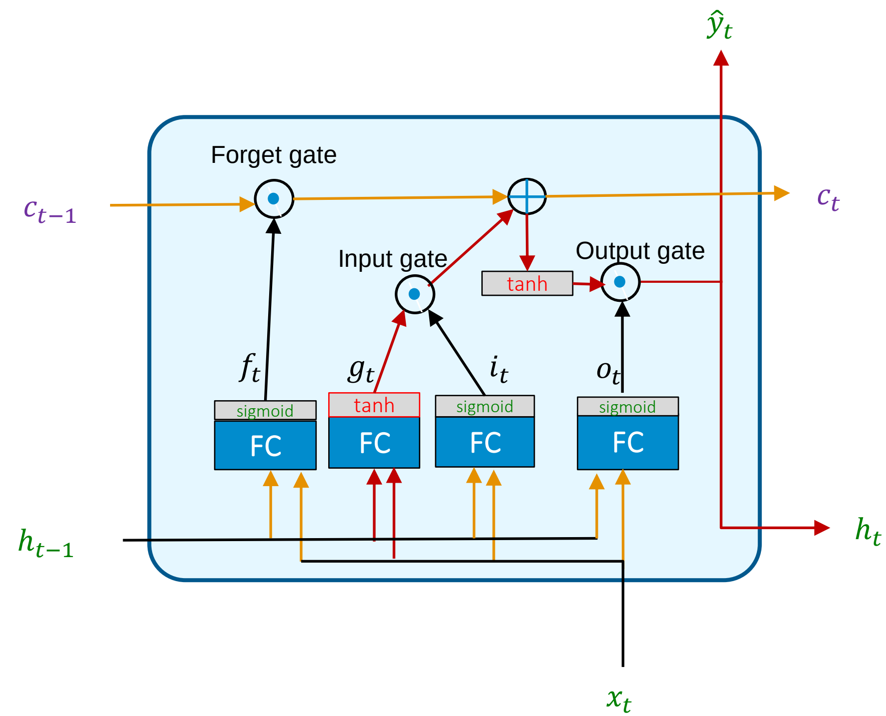
- Introduce three gate controllers:
  - Use sigmoid to ensure outputs' range between 0 and 1
  - Forget gate f_t controls how much c_{t-1} be 'forgotten'
  - Input gate i_t controls which how much g_t be remembered
  - Output gate o_t controls how much long-term c_t should be carried on to the next time slice:
    - to contribute to short-term state: h_t
    - to contribute to the output: $\hat{y}_t$

- $g_t = \tanh(Wh_{t-1} + Ux_t + b)$
- Forget gate: $f_t = \sigma\left(U^f x_t + W^f h_{t-1} + b^f\right)$
- Input gate: $i_t = \sigma(U^i x_t + W^i h_{t-1} + b^i)$
- LSTM long-term state: $c_t = f_t \odot c_{t-1} + g_t \odot i_t$
- Output gate: $o_t = \sigma(U^o x_t + W^o h_{t-1} + b^o)$
- LSTM short-term state: $h_t = o_t \odot \tanh(c_t)$
- LSTM output: $\hat{y}_t = Vh_t + c$

Long Short-Term Memory (LSTM)

Output Gate





- Output regression:

$$y_t = h_t$$

or $$y_t = Vh_t + c$$

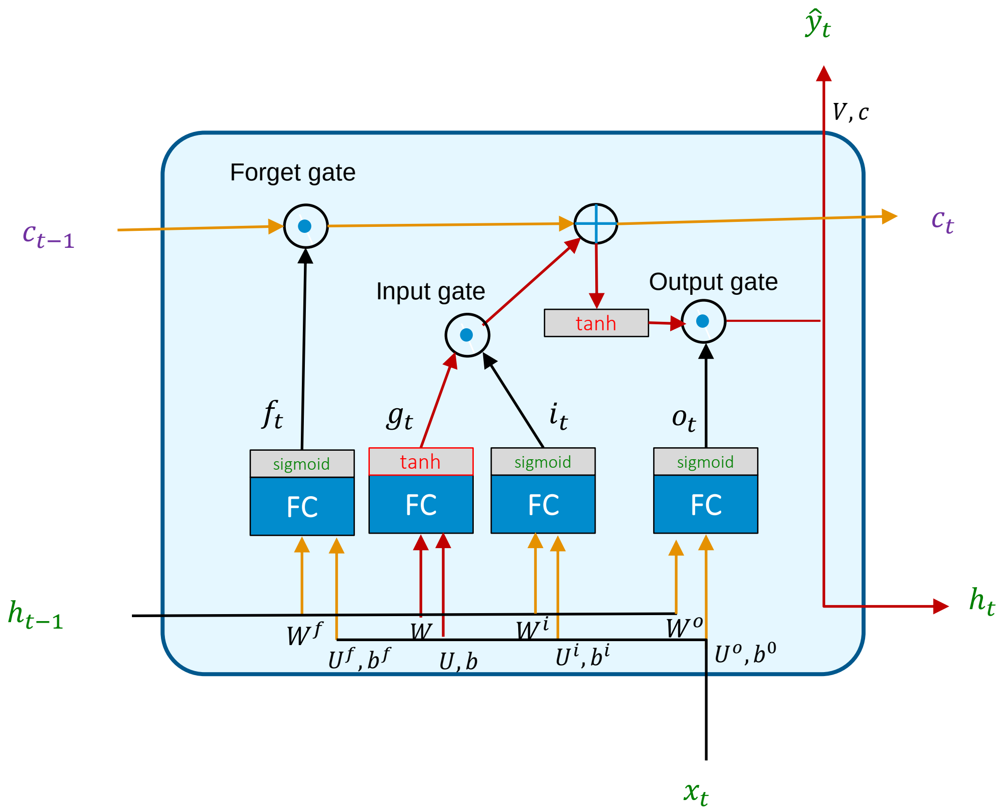
- Output classification

$$y_t = \text{softmax}(h_t)$$

- $g_t = \tanh(Wh_{t-1} + Ux_t + b)$
- Forget gate: $f_t = \sigma\left(U^f x_t + W^f h_{t-1} + b^f\right)$
- Input gate: $i_t = \sigma(U^i x_t + W^i h_{t-1} + b^i)$
- LSTM long-term state: $c_t = f_t \odot c_{t-1} + g_t \odot i_t$
- Output gate: $o_t = \sigma(U^o x_t + W^o h_{t-1} + b^o)$
- LSTM short-term state: $h_t = o_t \odot \tanh(c_t)$
- LSTM output: $y_t = Vh_t + c$

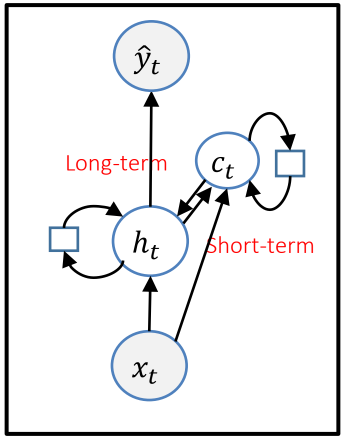
Long Short-Term Memory (LSTM)

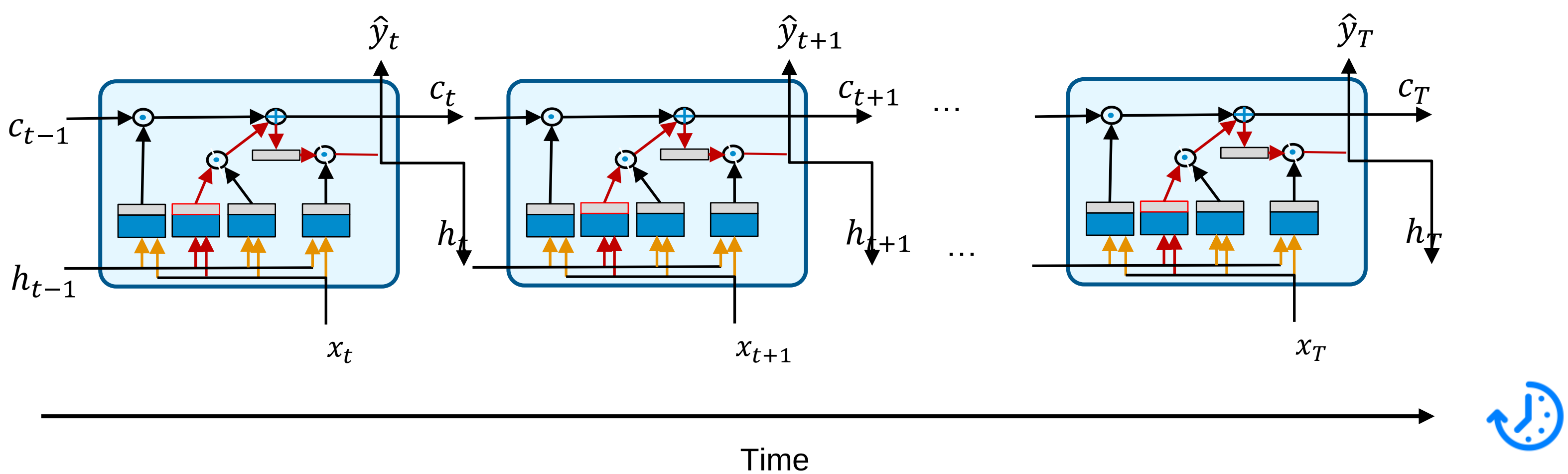
Output Gate



- $g_t = \tanh(W h_{t-1} + U x_t + b)$
- Forget gate: $f_t = \sigma\left(U^f x_t + W^f h_{t-1} + b^f\right)$
- Input gate: $i_t = \sigma(U^i x_t + W^i h_{t-1} + b^i)$
- LSTM long-term state: $c_t = f_t \odot c_{t-1} + g_t \odot i_t$
- Output gate: $o_t = \sigma(U^o x_t + W^o h_{t-1} + b^o)$
- LSTM short-term state: $h_t = o_t \odot \tanh(c_t)$
- LSTM output: $\hat{y}_t = V h_t + c$

Long Short-Term Memory (LSTM)

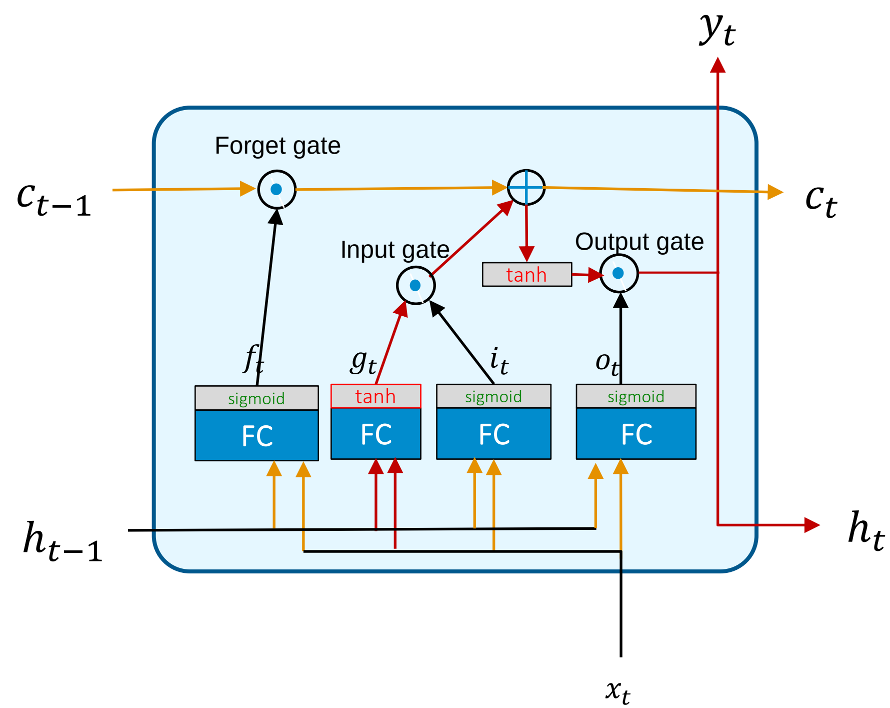




Long Short-Term Memory (LSTM)

Summary

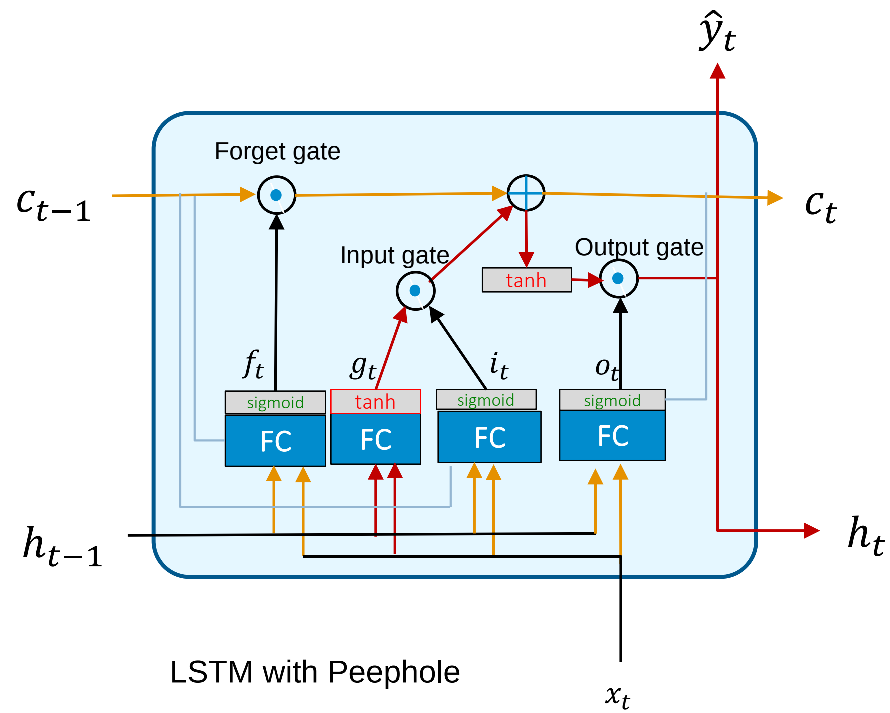
- LSTM belongs to a class of gated RNN models
- LSTM introduce self-loops to create paths where the gradient can flow for long durations
- Improve over basic RNN cell
  - Can capture long-term dependency
  - Faster and more robust to train, often with quicker convergence
- LSTM cells manage two state vectors, and for performance reasons they are kept separate by default
  - h_t as the short-term state
  - c_t as the long-term state
- Gates can remove or add information to the cell state: forget, input, output



Long Short-Term Memory (LSTM)

Peephole Connections

- LSTM: gate controllers only use information from x_t and h_{t-1}
- Peephole connections [Gers and Schimdhuber 2000]:
  - Long-term cell state c_{t-1} connects to f_t and i_t
  - c_t connects to o_t



LSTM with Peephole

Gated Recurrent Unit (GRU)

[Cho et. al, 2014]

Gated Recurrent Unit

[Cho et. al, 2014]

- Proposed by Cho et al. 2014
  - This was introduced the Encoder-Decoder network
- Can be viewed as a simplified version of the LSTM:
  - Both long term c_t and short-term h_t merged to a single state h_t
  - Single update gate controller z_t is used to control both forget and input gates
    - = 1: open input gate, close forget gate
    - = 0: close input gate, open forget gate
    - i.e., whenever a memory must be stored, the location to be stored will be erased first!
  - Output gate is removed, but an additional reset gate r_t controls how much previous state should be carried forward

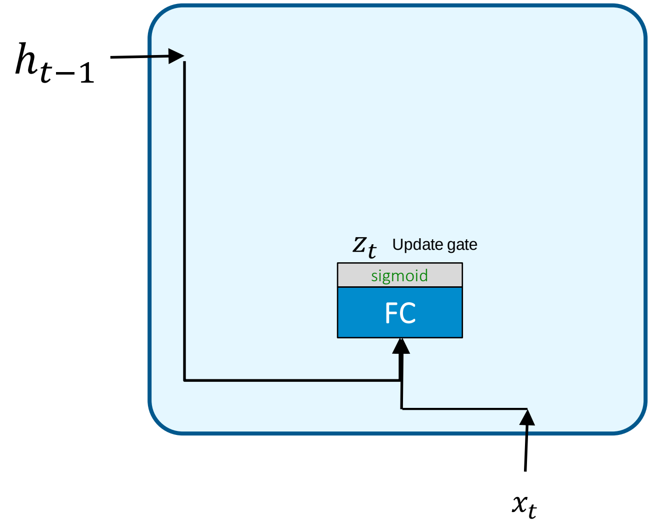
Cho et al., Learning Phrase Representation using RNN Encoder-Decoder for Statistical Machine Translation, *Proceedings of EMNLP*, 2014.

Gated Recurrent Unit

[Cho et. al, 2014]

- Update gate z_t decides how much the unit updates its state:

$$z_t = \sigma(U^z x_t + W^z h_{t-1})$$



Gated Recurrent Unit

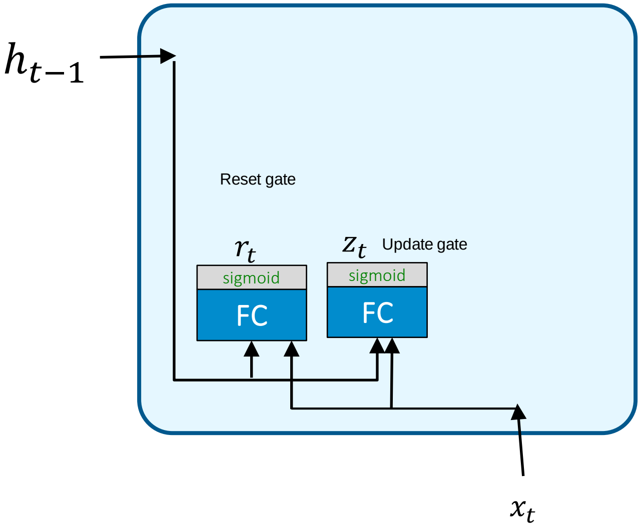
[Cho et. al, 2014]

- Update gate z_t decides how much the unit updates its state:

$$z_t = \sigma(U^z x_t + W^z h_{t-1})$$

- Reset gate controls which parts of the state get used to compute the next target state

$$r_t = \sigma(U^r x_t + W^r h_{t-1})$$



Gated Recurrent Unit

[Cho et. al, 2014]

- Update gate z_t decides how much the unit updates its state:

$$z_t = \sigma(U^z x_t + W^z h_{t-1})$$

- Reset gate controls which parts of the state get used to compute the next target state

$$r_t = \sigma(U^r x_t + W^r h_{t-1})$$

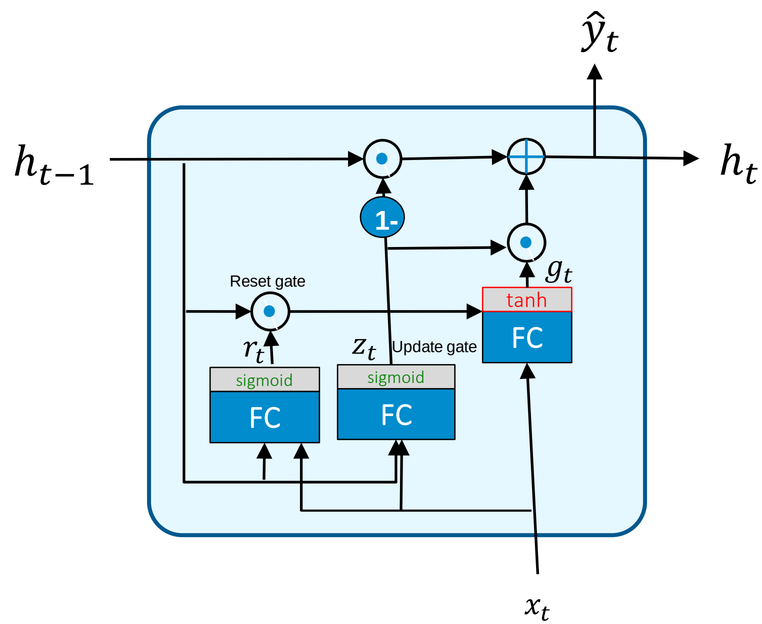
- The memory state h_t is a linear interpolation between h_{t-1} and g_t

$$h_t = (1 - z_t) h_{t-1} + z_t \odot g_t$$

where the candidate g_t is pre-computed

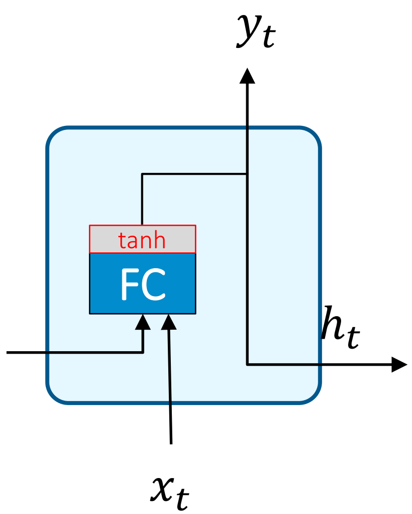
$$g_t = \tanh\left(U^g x_t + W^g \cdot (r_t \odot h_{t-1})\right)$$

- When z_t and r_t are close to 1, GRU will be reduced to Basic RNN

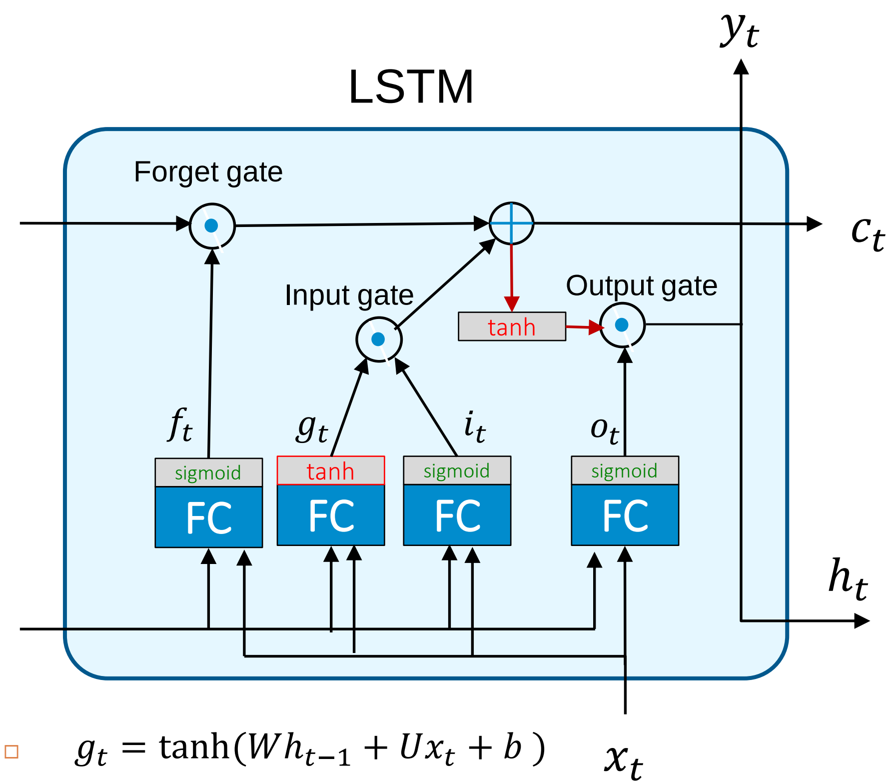


Summary: memory cells

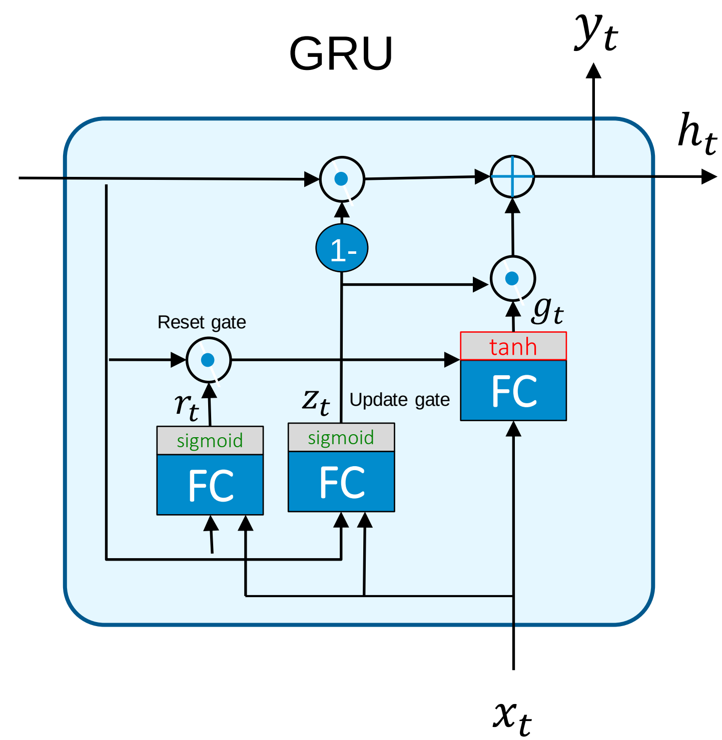
RNN



LSTM



GRU



Memory state:
$h_t = \tanh(Wh_{t-1} + Ux_t + b)$

- $g_t = \tanh(Wh_{t-1} + Ux_t + b\ )$
- Forget gate: $f_t = \sigma\left(U^f x_t + W^f h_{t-1} + b^f\right)$
- Input gate: $i_t = \sigma(U^i x_t + W^i h_{t-1} + b^i)$
- LSTM long-term state: $c_t = f_t \odot c_{t-1} + g_t \odot i_t$
- Output gate: $o_t = \sigma(U^o x_t + W^o h_{t-1} + b^o)$
- LSTM short-term state: $h_t = o_t \odot \tanh(c_t\ )$
- LSTM output: $\hat{y}_t = Vh_t + c$

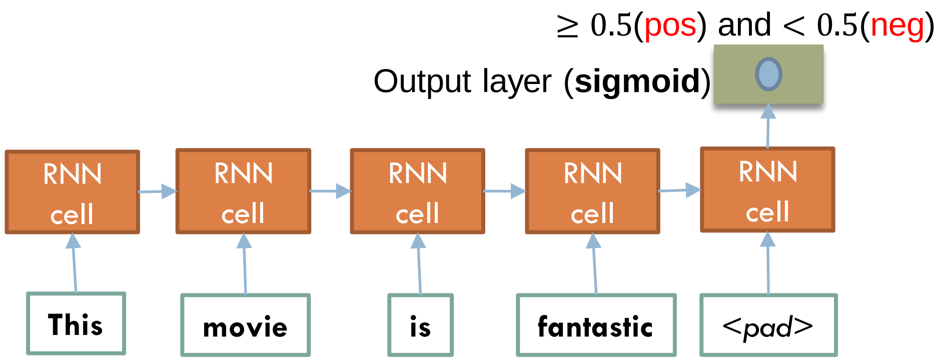
- Reset gate: $z_t = \sigma(U^z x_t + W^z h_{t-1})$
- Update gate:$r_t = \sigma(U^r x_t + W^r h_{t-1})$
- $g_t = \tanh\left(U^g x_t + W^g \cdot (r_t \odot h_{t-1})\right)$
- Memory state: $h_t = (1 - z_t)h_{t-1} + z_t \odot g_t$

Applications of Recurrent Neural Networks

- Sentiment Analysis
- Text generation

Sentiment analysis (Tutorial 7a)

- **Movie review dataset**
  1. I like that movie (**pos**:1).
  2. This is a bad movie to watch (**neg**:0)
  3. I love the movie (**pos**: 1)
  4. I do not recommend you to watch this movie (**neg**:0)
  5. This movie is fantastic (**pos**:1)



How to feed **symbolic words** to an RNN?

Sentiment analysis

Movie review dataset

1. I like this movie (**pos**:1).
2. This is a bad movie to watch (**neg**:0)
3. I love this movie (**pos**: 1)
4. I do not recommend you to watch this movie (**neg**:0)
5. This movie is fantastic (**pos**:1)

Build up vocabulary

1. Like (index: 1)
2. Love (index: 2)
3. Bad (index: 3)
4. Fantastic (index: 4)
5. Not (index: 5)
6. Recommend (index: 6)

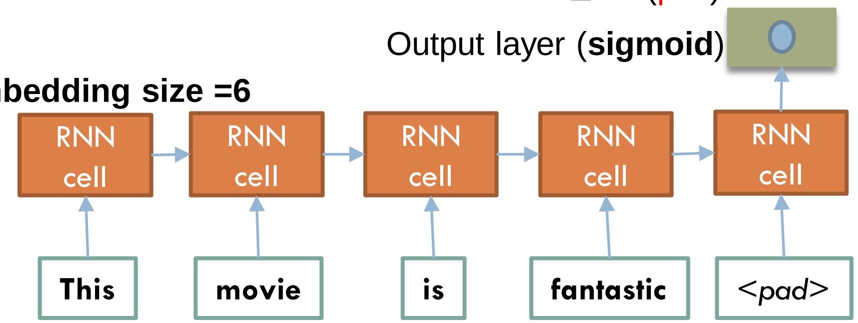
Not in vocabulary (out of vocabulary bucket: 2)

1. I, movie, to, pad (index: 7)
2. This, is, watch (index: 8)

≥ 0.5(pos) and < 0.5(neg)

Output layer (**sigmoid**)

embedding size =6



Embedding matrix (E [8×6])

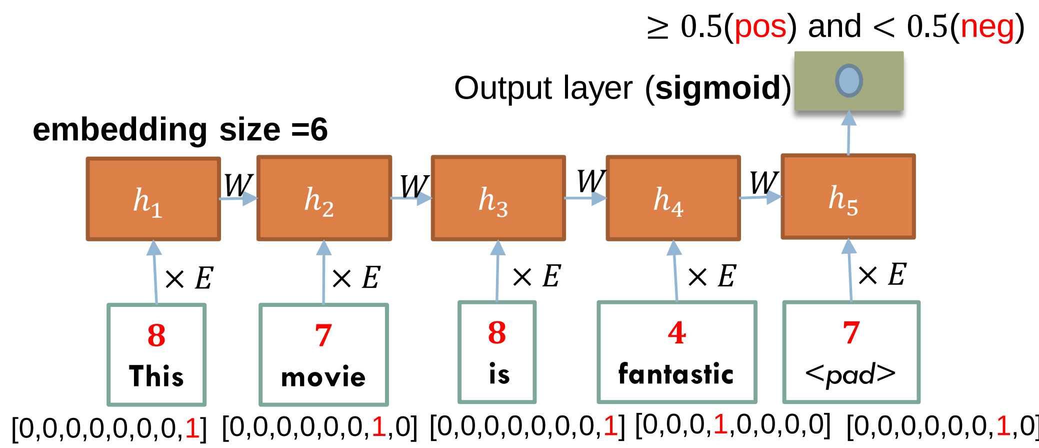
E_1	1	2	1.5	-1.2	1.3	1
E_2	-1	1.3	-2.5	-1.2	1.6	-1
E_3	1	3.3	-3.5	-1.0	2.6	1
E_4	-1	1.3	-2.5	-1.2	1.8	-1
E_5	-1.2	2.3	-2.5	-1.2	-1.6	1
E_6	-1.7	-1.3	-2.5	-1.2	3.6	1.2
E_7	-4.2	2.3	-3.5	4.3	1.8	-2
E_8	-1.7	-1.3	-4.5	-2.2	-3.6	1

Sentiment analysis

- **Movie review dataset**
  1. I like this movie (**pos**:1).
  2. This is a bad movie to watch (**neg**:0)
  3. I love this movie (**pos**: 1)
  4. I do not recommend you to watch this movie (**neg**:0)
  5. This movie is fantastic (**pos**:1)

- **Build up vocabulary**
  1. Like (index: 1)
  2. Love (index: 2)
  3. Bad (index: 3)
  4. Fantastic (index: 4)
  5. Not (index: 5)
  6. Recommend (index: 6)

- **Not in vocabulary** (out of vocabulary bucket: 2)
  1. I, movie, to, pad (index: 7)
  2. This, is, watch (index: 8)

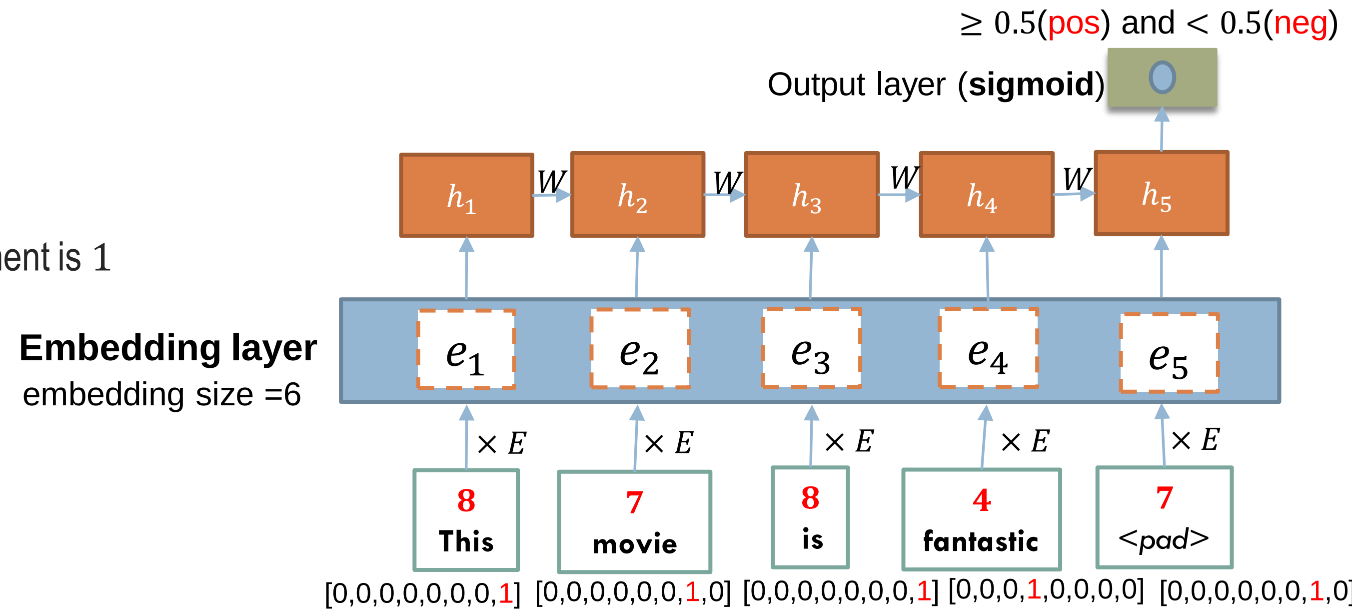


Embedding matrix (E [8×6])

E_1	1	2	1.5	-1.2	1.3	1
E_2	-1	1.3	-2.5	-1.2	1.6	-1
E_3	1	3.3	-3.5	-1.0	2.6	1
E_4	-1	1.3	-2.5	-1.2	1.8	-1
E_5	-1.2	2.3	-2.5	-1.2	-1.6	1
E_6	-1.7	-1.3	-2.5	-1.2	3.6	1.2
E_7	-4.2	2.3	-3.5	4.3	1.8	-2
E_8	-1.7	-1.3	-4.5	-2.2	-3.6	1

Sentiment analysis

- The word/item embedding
  - $e_1 = 1_8 E \in \mathbb{R}^{1\times 6}$
    - $[1\times 8]\times[8\times 6] = [1\times 6]$
    - 1_i is one-hot vector in which the $i-th$ element is 1 and the rest are 0.
  - $e_2 = 1_7 E \in \mathbb{R}^{1\times 6}$
  - $e_3 = 1_8 E \in \mathbb{R}^{1\times 6}$
  - $e_4 = 1_4 E \in \mathbb{R}^{1\times 6}$
  - $e_5 = 1_7 E \in \mathbb{R}^{1\times 6}$



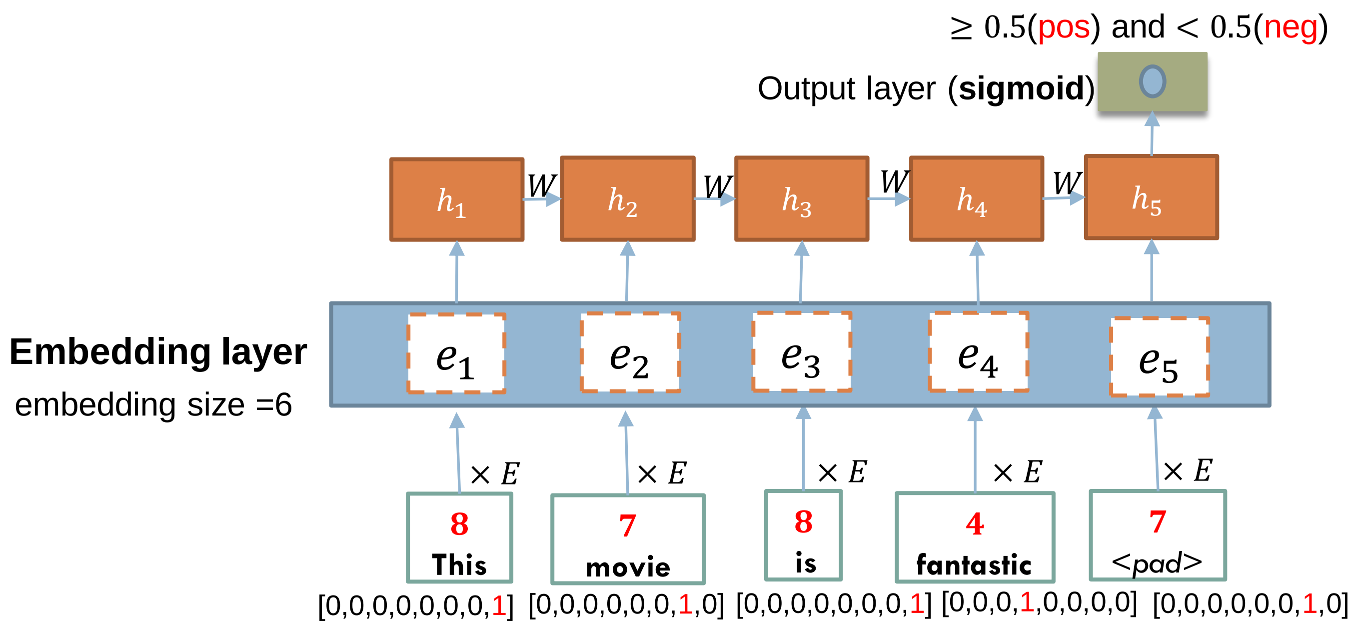
- The sequence embedding
  - $e = \begin{bmatrix} e_1 \\ e_2 \\ e_3 \\ e_4 \\ e_5 \end{bmatrix} \in \mathbb{R}^{5\times 6} = \mathbb{R}^{seq\_len \times embed\_size}$
  - **Embedding lookup** operation
    - Pick rows with indices **8, 7, 8, 4, 7**

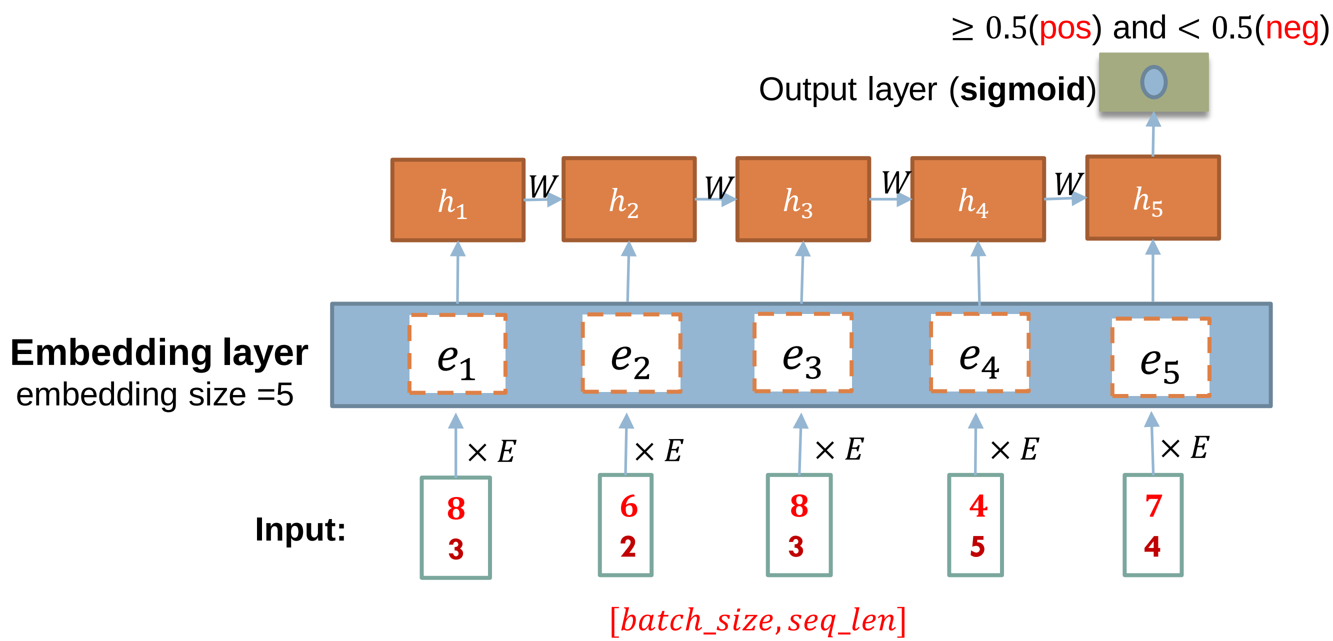
Embedding matrix (E [8×6])

E_1	1	2	1.5	-1.2	1.3	1
E_2	-1	1.3	-2.5	-1.2	1.6	-1
E_3	1	3.3	-3.5	-1.0	2.6	1
E_4	-1	1.3	-2.5	-1.2	1.8	-1
E_5	-1.2	2.3	-2.5	-1.2	-1.6	1
E_6	-1.7	-1.3	-2.5	-1.2	3.6	1.2
E_7	-4.2	2.3	-3.5	4.3	1.8	-2
E_8	-1.7	-1.3	-4.5	-2.2	-3.6	1

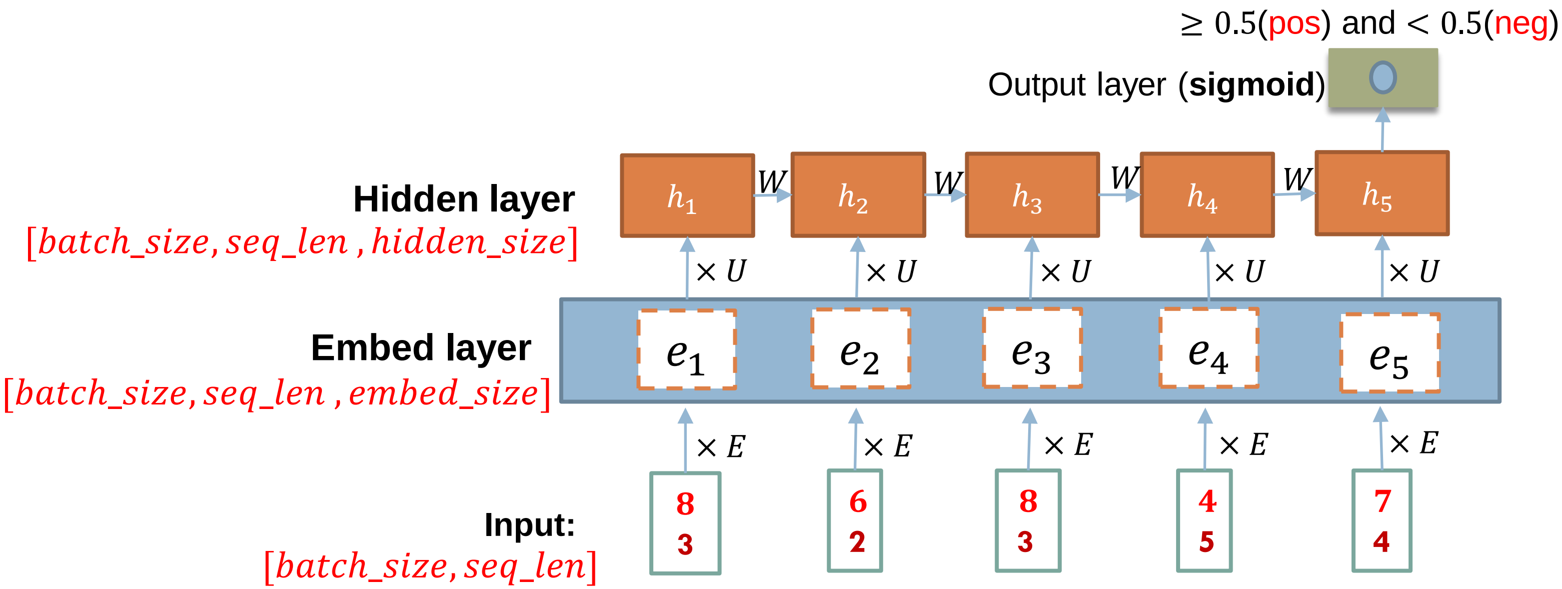
Sentiment analysis

- The word/item embedding
  - $e_1 = 1_8 E \in \mathbb{R}^{1\times 6}$
  - $e_2 = 1_7 E \in \mathbb{R}^{1\times 6}$
  - $e_3 = 1_8 E \in \mathbb{R}^{1\times 6}$
  - $e_4 = 1_4 E \in \mathbb{R}^{1\times 6}$
  - $e_5 = 1_7 E \in \mathbb{R}^{1\times 6}$
- The sequence embedding
  - $e = \begin{bmatrix} e_1 \\ e_2 \\ e_3 \\ e_4 \\ e_5 \end{bmatrix} \in \mathbb{R}^{5\times 6} = \mathbb{R}^{seq\_len \times embed\_size}$
- The sequence batch embedding
  - The embedding for one sequence: $\mathbb{R}^{seq\_len \times embed\_size}$.
  - The embedding for entire batch with $batch\_size$ sequences: $\mathbb{R}^{batch\_size \times seq\_len \times embed\_size}$





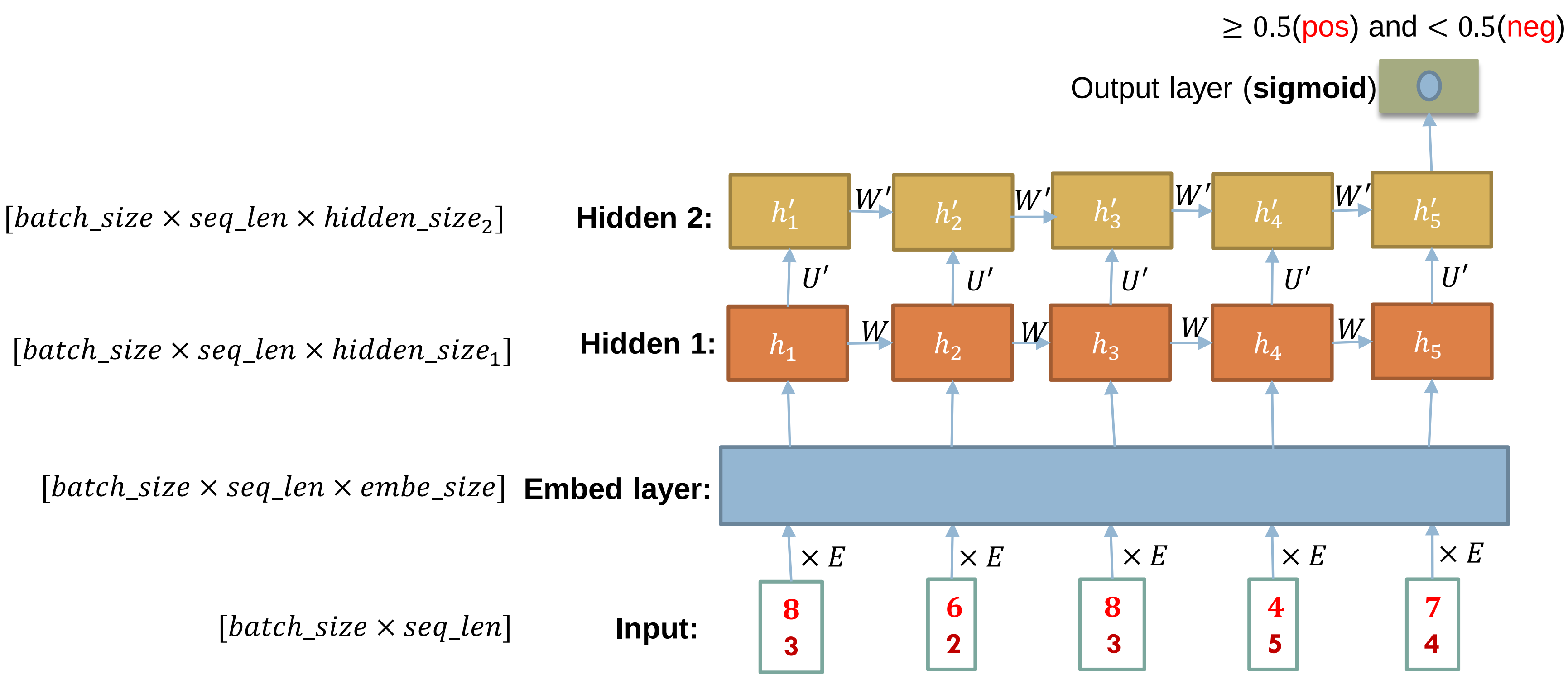
Sentiment analysis



- The sequence batch embedding
  - The embedding for one sequence: $\mathbb{R}^{seq\_len\ \times embed\_size}$.
  - The embedding for entire batch with $batch\_size$ sequences: $\mathbb{R}^{batch\_size \times seq\_len\ \times embed\_size}$
- For one sequence, the hidden value is
  - $h = eU + b$ has form of $[seq\_len, embed\_size] \times [hidden\_size, embed\_size] = [seq\_len, hidden\_size]$ and plus a bias with an appropriate shape (e.g., $[1 \times hidden\_size]$) to gain **hidden value** with shape $[seq\_len, hidden\_size]$.
- For a batch of sequences, the hidden values have shape
  - $[batch\_size, seq\_len, hidden\_size]$

Sentiment analysis

- Stack one more hidden layer of memory cells



Sentiment analysis (Tutorial 8b)

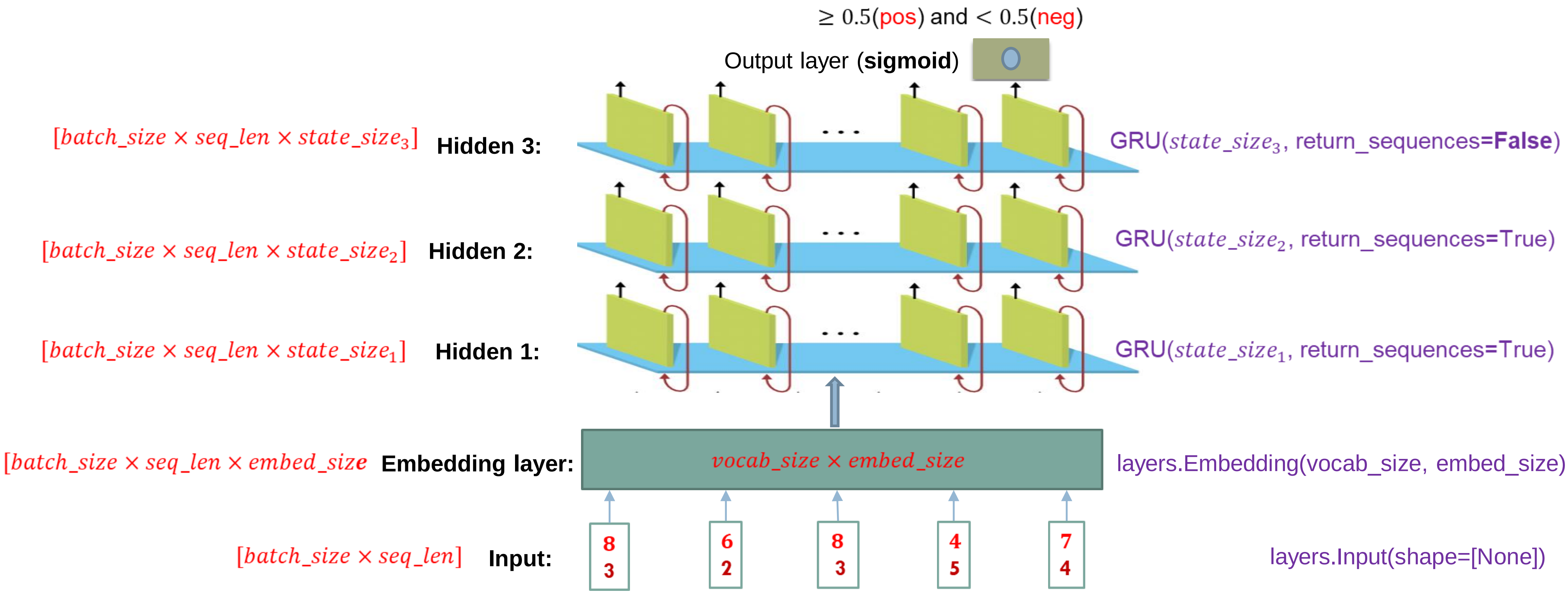
```
embed_size = 128
x = tf.keras.Input(shape=[None], dtype="int64")
print(x.shape)
h = tf.keras.layers.Embedding(vocab_size + num_oov_buckets, embed_size)(x)
print(h.shape)
h = tf.keras.layers.GRU(64, return_sequences=True)(h)
print(h.shape)
h = tf.keras.layers.GRU(64)(h)
print(h.shape)
h = tf.keras.layers.Dense(1, activation="sigmoid")(h)
rnn_model = tf.keras.models.Model(inputs = x, outputs= h)
```

```
(None, None)
(None, None, 128)
(None, None, 64)
(None, 64)
```

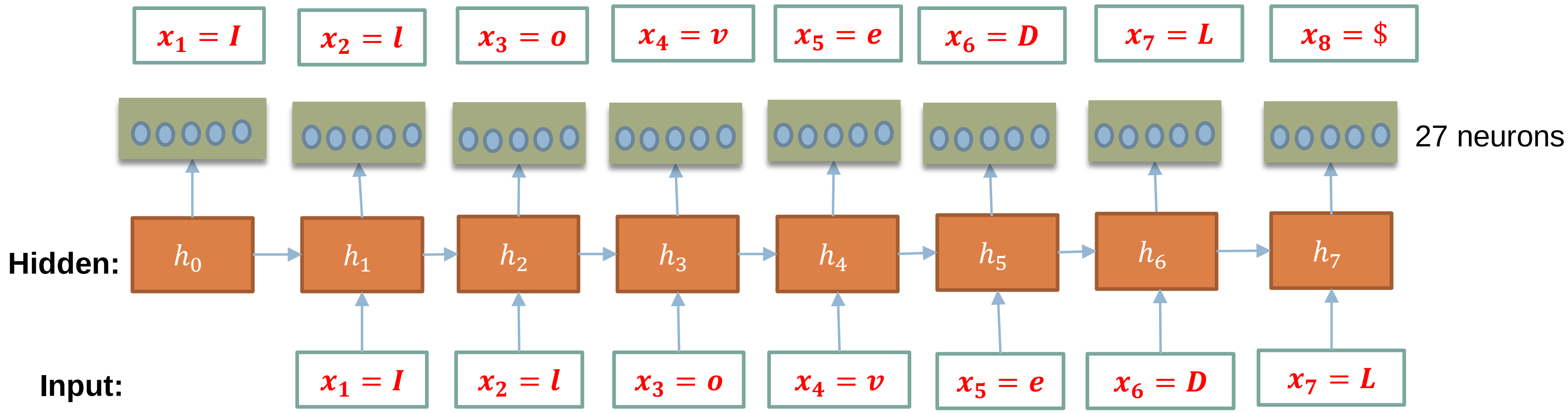
```
embed_size = 128
x = tf.keras.Input(shape=[None], dtype="int64")
print(x.shape)
h = tf.keras.layers.Embedding(vocab_size + num_oov_buckets, embed_size)(x)
print(h.shape)
h = tf.keras.layers.GRU(64, return_sequences=False)(h)
print(h.shape)
h = tf.keras.layers.GRU(64)(h)
print(h.shape)
h = tf.keras.layers.Dense(1, activation="sigmoid")(h)
rnn_model = tf.keras.models.Model(inputs = x, outputs= h)
```

```
(None, None)
(None, None, 128)
(None, 64)
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-22-696dcaff996e> in <module>
      6 h = tf.keras.layers.GRU(64, return_sequences=False)(h)
      7 print(h.shape)
----> 8 h = tf.keras.layers.GRU(64)(h)
      9 print(h.shape)
     10 h = tf.keras.layers.Dense(1, activation="sigmoid")(h)
```

Implementation of RNNs



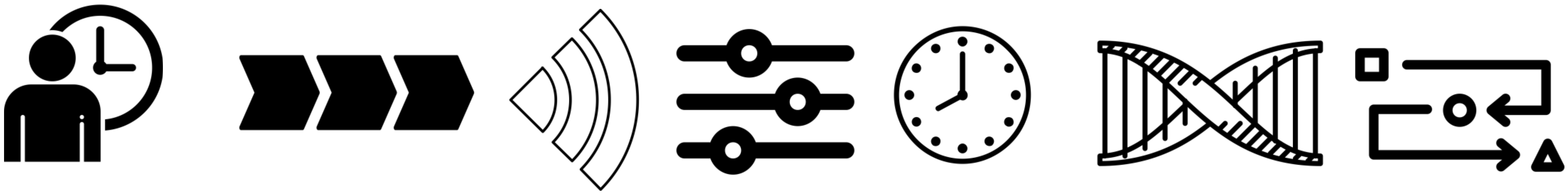
Text generation at the character level (Tutorial 8c)



- Consider a sentence: x="I love DL" in **our dataset**
  - $x_1 = I, x_2 = l, x_3 = o, x_4 = v, x_5 = e, x_6 = D, x_7 = L, x_8 = \$$
- The joint distribution
  - $p(x_1, x_2, \ldots, x_7, x_8) = p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_{1:2}) \ldots p(x_7 \mid x_{1:6})p(x_8 \mid x_{1:7})$
  - $p(x_1, x_2, \ldots, x_7, x_8) = p(x_1 \mid h_0)p(x_2 \mid h_1)p(x_3 \mid h_2) \ldots p(x_7 \mid h_6)p(x_8 \mid h_7)$
  - $\mathbf{max} \log p(x_1, x_2, \ldots, x_7, x_8)$ is equivalent to $\mathbf{min}[-[\log p(x_1 \mid h_0) + \log p(x_2 \mid h_1) + \cdots + \log p(x_8 \mid h_7)]]$
  - Given h_1, we need to maximize the probability to predict x_2 and so on.
  - Cast to the problem of prediction to train our RNN
    - The loss is sum of loss at each time step

Thanks for your attention!

Time-series and sequential data

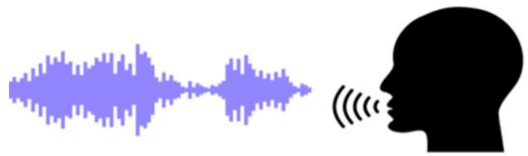


- We live in a time-space universe
- All data collected has a timestamp
- Time-series/sequential data

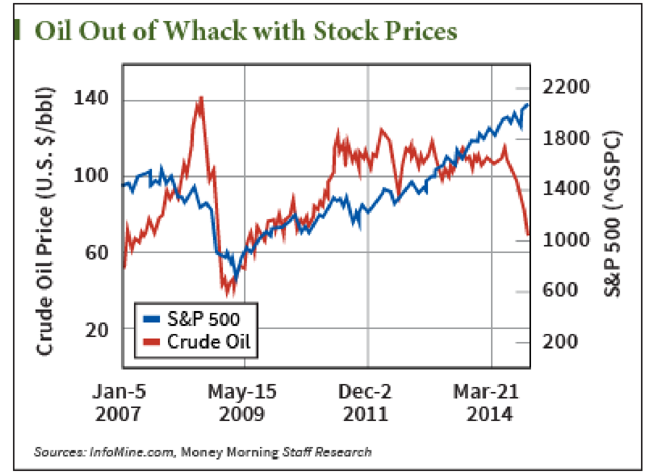
  = collection of sequential data points indexed by time order!

What are time-series and sequential data?

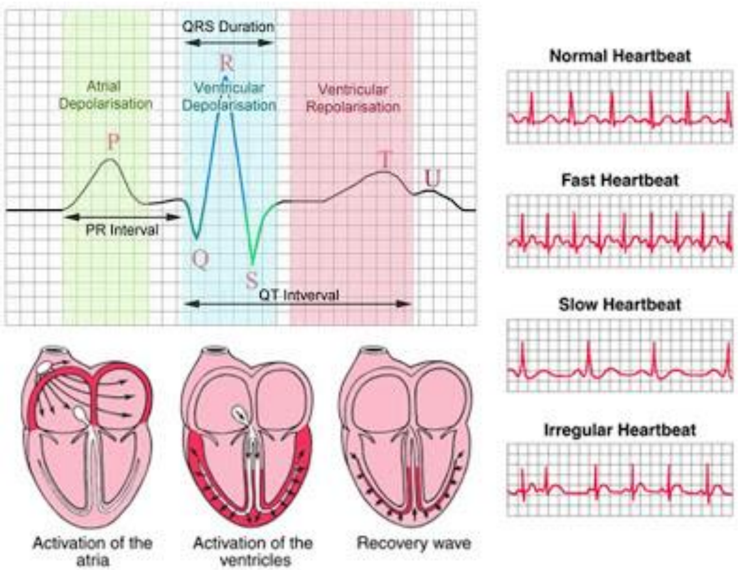
Video surveillance

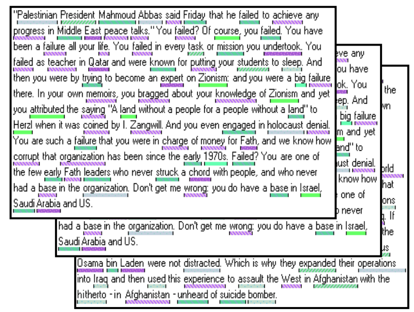


Speech



Stock market

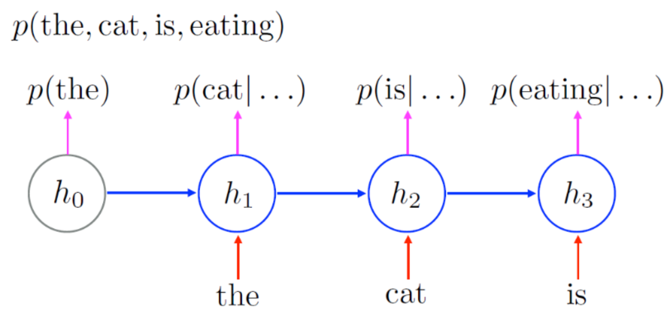




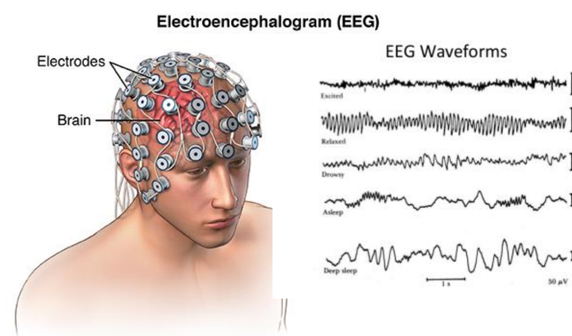
"Palestinian President Mahmoud Abbas said Friday that he failed to achieve any progress in Middle East peace talks." You failed? Of course, you failed. You have been a failure all your life. You failed in every task or mission you undertook. You failed as teacher in Qatar and were known for putting your students to sleep. And then you were by trying to become an expert on Zionism: and you were a big failure there. In your own memoirs, you bragged about your knowledge of Zionism and yet you attributed the saying "A land without a people for a people without a land" to Herzl when it was coined by I. Zangwill. And you even engaged in holocaust denial. You are such a failure that you were in charge of money for Fath, and we know how corrupt that organization has been since the early 1970s. Failed? You are one of the few early Fath leaders who never struck a chord with people, and who never had a base in the organization. Don't get me wrong: you do have a base in Israel, Saudi Arabia and US.

had a base in the organization. Don't get me wrong: you do have a base in Israel, Saudi Arabia and US.

Osama bin Laden were not distracted. Which is why they expanded their operations into Iraq and then used this experience to assault the West in Afghanistan with the hitherto - in Afghanistan - unheard of suicide bomber.



Language modelling/natural language processing tasks and data

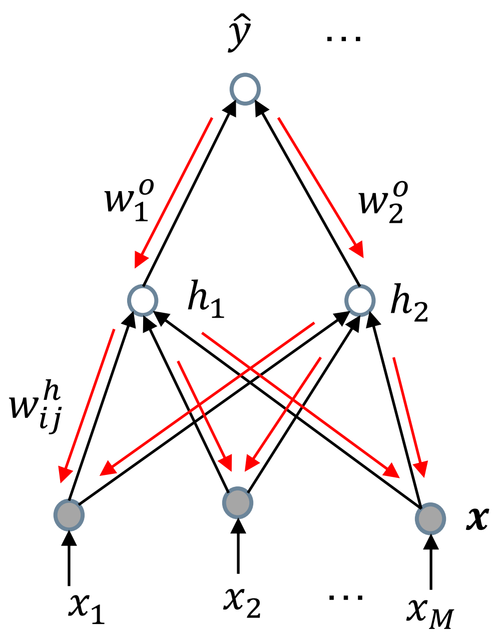


EEG brain

Electrocardiography signals = electrical activity of the heart over time

Training RNN

- Use back propagation through time (BPTT)



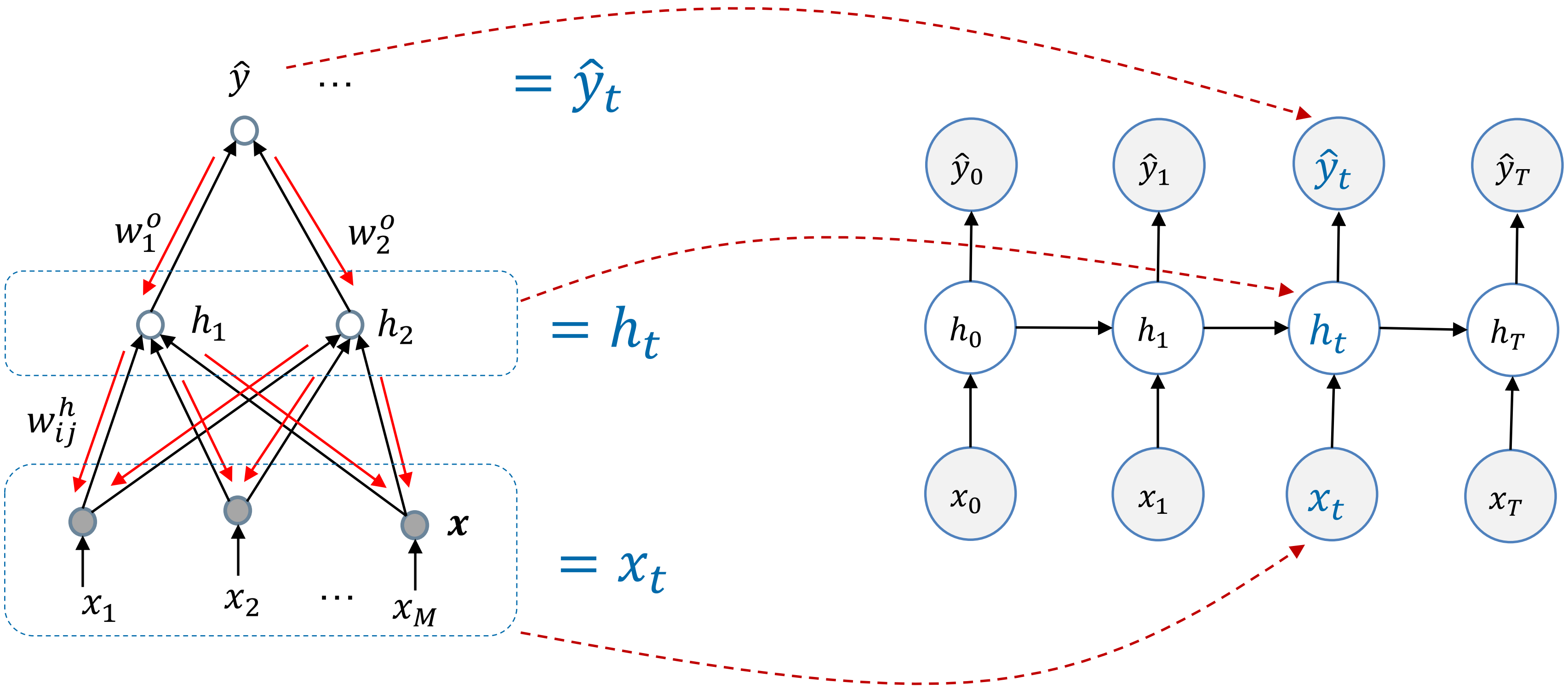
Input: $(\boldsymbol{x}, y)$

$\hat{y} = \text{forward}(x)$

Goal: minimize $J(\boldsymbol{w}) = \frac{1}{2}(\hat{y} - y)^2$

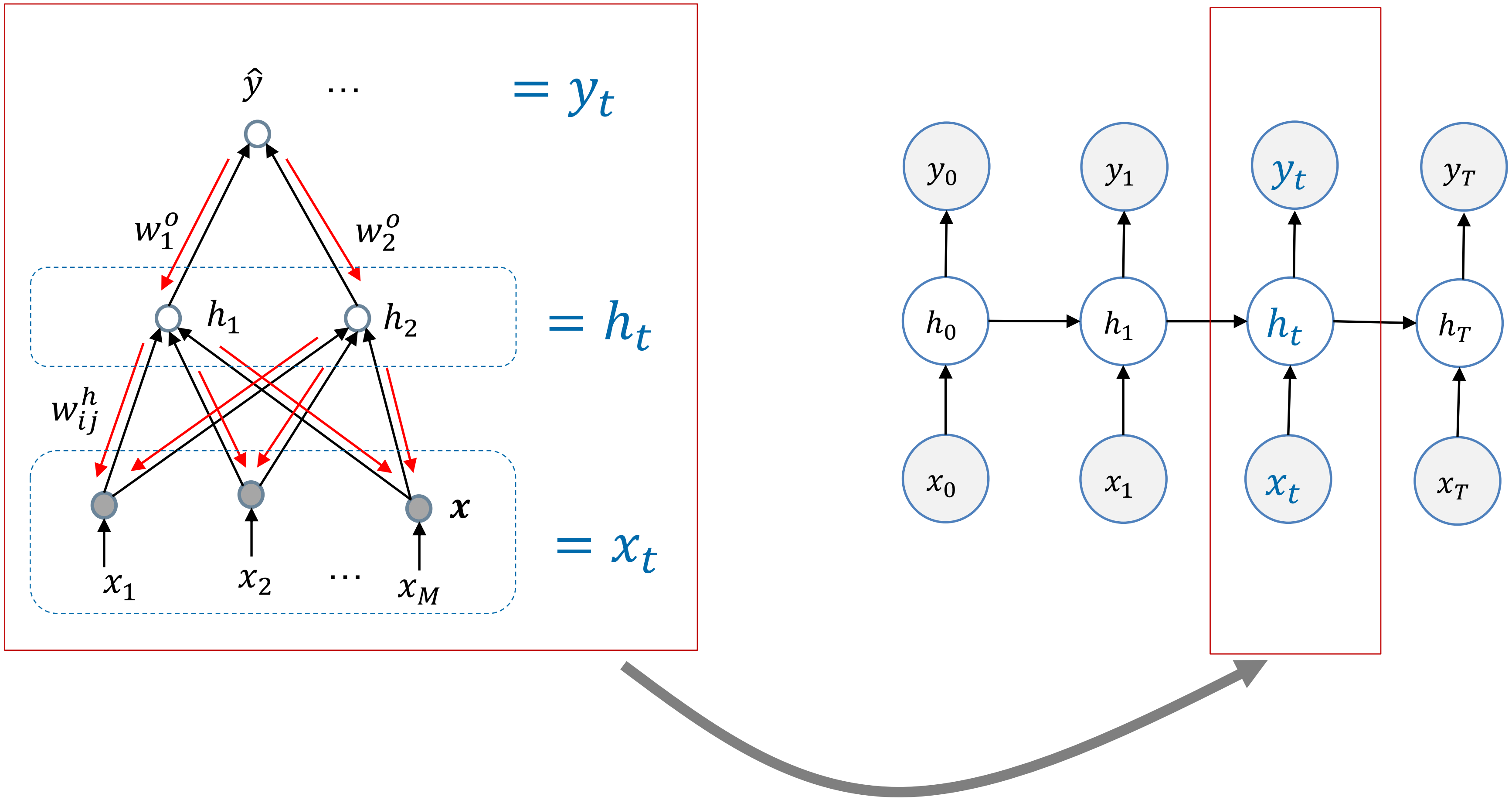
Training RNN

- Use back propagation through time (BPTT)



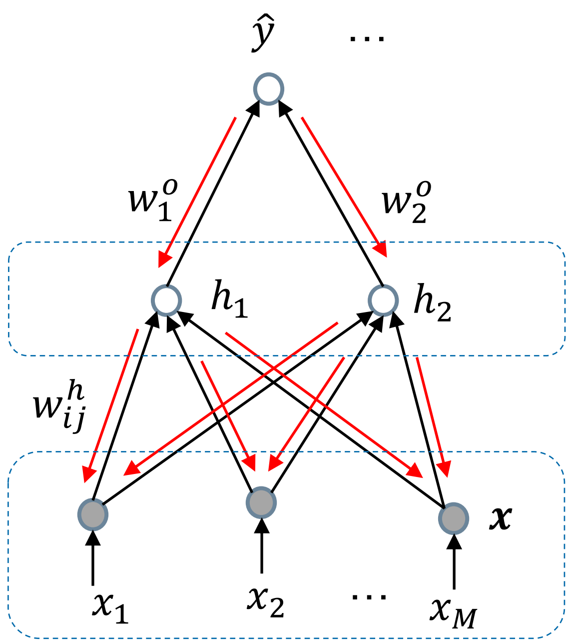
Training RNN

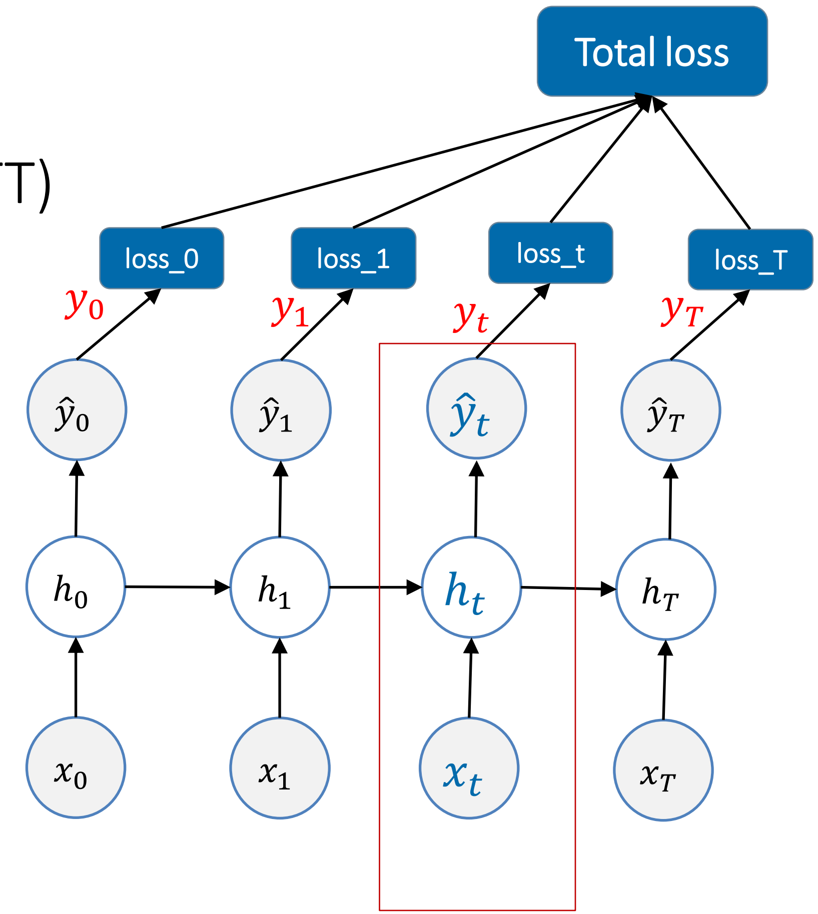
- Use back propagation through time (BPTT)



Training RNN

- Use back propagation through time (BPTT)





Input: $(\boldsymbol{x}, y)$

$\hat{y} = \text{forward}(x)$

Goal: minimize $J(\boldsymbol{w}) = \frac{1}{2}(\hat{y} - y)^2$

Replicated over multiple time slices

Input: sequence $x_{1:T}$, sequence $y_{1:T}$

$\hat{y}_{1:T} = \text{forward}(x_{1:T})$

Goal: minimize $J(U, W, V) = \frac{1}{2}\sum_t (\hat{y}_t - y_t)^2$

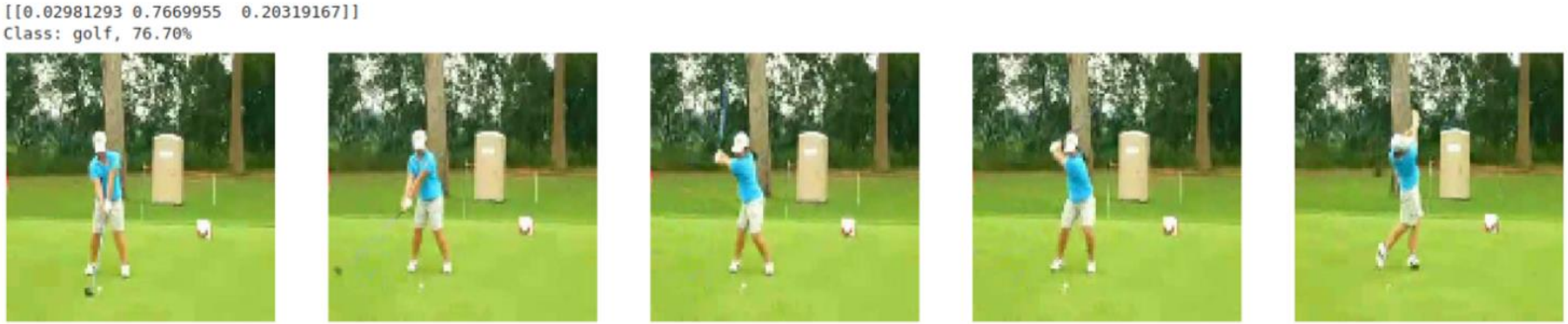
Sequential data examples

- Data can also be viewed from different, more subtle angles

I love deep learning

- Word level
  - I, love, deep, learning
- Character level
  - I, l, o, v, e, d, e, e, p, l, e, a, r, n, i, n, g

- **Sequence of pixels or rows of pixels**



- Video as a **sequence of images** (Source: medium.com)