Deep Learning and Applications

DSA 5204 • Lecture 5

Dr Low Yi Rui (Aaron)
Department of Mathematics

National University of Singapore

Last Time

We introduced CNN to handle image data. The main idea is to exploit the spatial structure of images

- Extract features using the convolution operation
- Weight sharing and sparse connectivity
- Equivariance and invariance

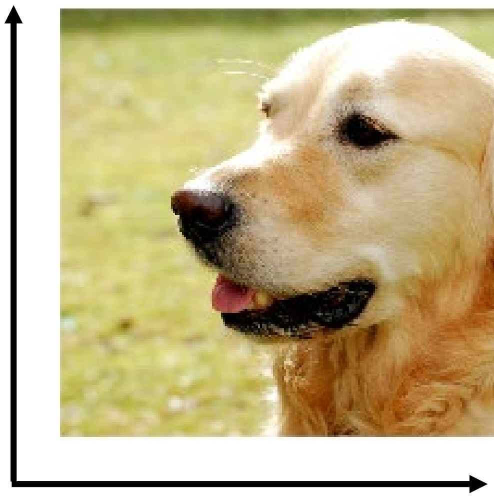
Today we will discuss another class of neural network to handle temporal data: recurrent neural networks

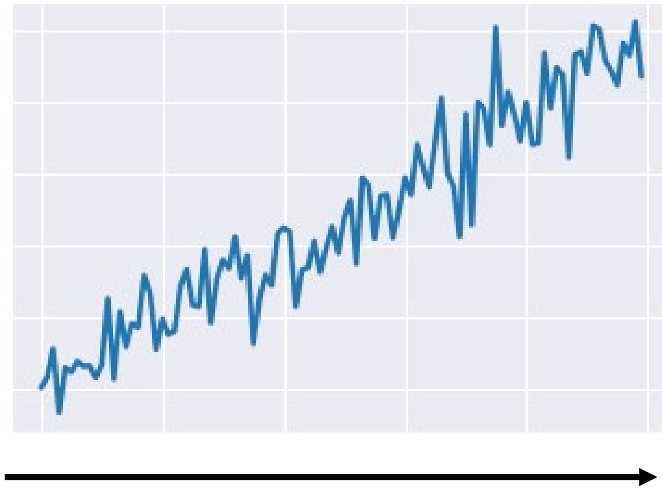
Time series modelling

What is a time series?

Like images, a time series is another type of data with some local structure.

Instead of spatial structure in images, we consider temporal structures.





We may represent a time series as either

- A discrete sequence

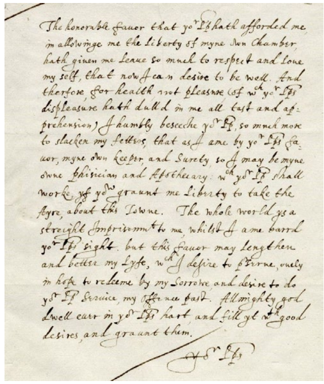
$$\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)}, \dots$$

- A continuous sequence

$$\left\{\boldsymbol{x}^{(t)} : t \geq 0\right\}$$

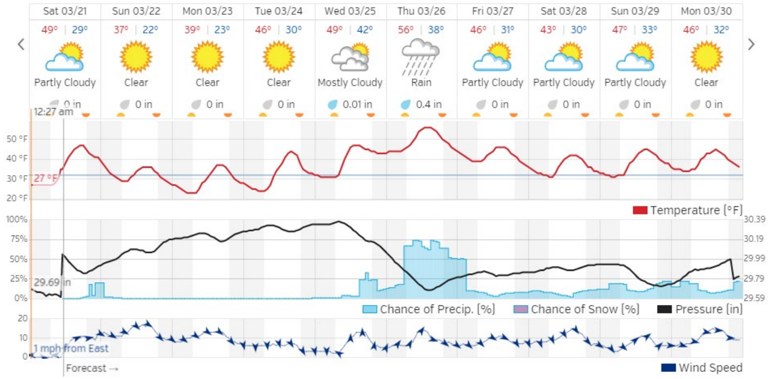
Each slice of the time series, $x^{(t)}$ is a vector, or something that can be represented as a vector

Examples of Time Series Data

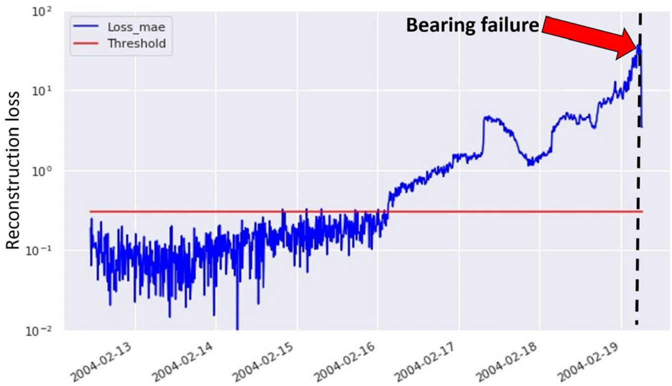


Text

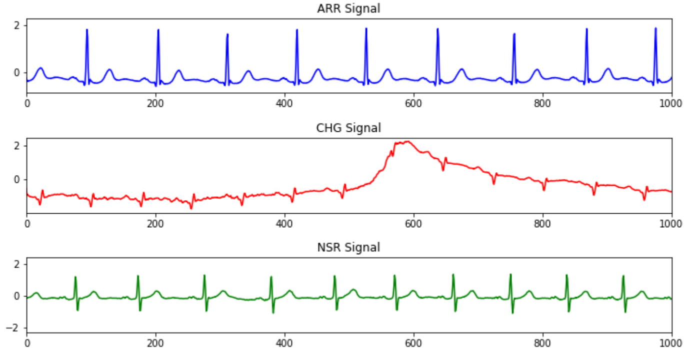
Stock Price



Weather



Machine Health



ECG Signals

Task I: Sequence Prediction

Given a historical time series data:

$$\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \dots \boldsymbol{x}^{(t)}$$

Task is to predict

- The next data point $\boldsymbol{x}^{(t+1)}$
- A sequence of next data points $\boldsymbol{x}^{(t+1)}, \boldsymbol{x}^{(t+2)}, \dots, \boldsymbol{x}^{(t+\tau)}$

Example applications

- Stock price prediction, weather forecasting

Task II: Sequence Classification/Regression

Given a time series data:

$$\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \dots \boldsymbol{x}^{(t)}$$

Task is to predict a label y of this sequence, which can be either discrete (classification) or continuous (regression)

Example Applications

- Credit card transaction fraud detection
- Heart arrhythmia detection

Task III: Sequence-to-sequence Modelling

Given a time series data:

$$\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \dots \boldsymbol{x}^{(t)}$$

Task is to predict a corresponding time series

$$\boldsymbol{y}^{(1)}, \boldsymbol{y}^{(2)}, \dots \boldsymbol{y}^{(t)}$$

Example Applications

- Machine translation
- Continuous health monitoring using wearable devices

Task IV: Sequence Generation

Given a seed sequence

$$\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \dots, \boldsymbol{x}^{(\tau)}$$

Task is to generate a longer sequence starting from this, i.e.

$$\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \dots, \boldsymbol{x}^{(\tau)}, \boldsymbol{x}^{(\tau+1)}, \boldsymbol{x}^{(\tau+2)}, \dots$$

according to some distribution

Example applications

- Writing poems
- Composing music

The Supervised Learning Problem

Recall that in supervised learning, we define the inputs, outputs and the target function that maps the former to the latter.

For time series modelling, we can define these similarly

- Inputs: $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)}, \dots, \boldsymbol{x}^{(\tau)}$
- Outputs: $\boldsymbol{y}^{(1)}, \boldsymbol{y}^{(2)}, \boldsymbol{y}^{(3)}, \dots, \boldsymbol{y}^{(\tau)}$
- Target: $\{\boldsymbol{F}_t^*\}$ with
$$\boldsymbol{y}^{(t)} = \boldsymbol{F}_t^*\left(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \dots, \boldsymbol{x}^{(\tau)}\right)$$
- Goal: Learn some $\{\widehat{\boldsymbol{F}}_t\}$ to approximate $\{\boldsymbol{F}_t^*\}$

Examples

- **Task I: Sequence Prediction**

$$\boldsymbol{F}_t^*\left(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \dots, \boldsymbol{x}^{(\tau)}\right) = \boldsymbol{F}_t^*\left(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \dots, \boldsymbol{x}^{(t)}\right) = \boldsymbol{x}^{(t+1)}$$

- **Task II: Sequence Classification/Regression**

$$\boldsymbol{F}_\tau^*\left(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \dots, \boldsymbol{x}^{(\tau)}\right) = \boldsymbol{y}^{(\tau)} = \boldsymbol{y}$$

- **Task III: Sequence-to-sequence**

$$\boldsymbol{F}_t^*\left(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \dots, \boldsymbol{x}^{(\tau)}\right) = \boldsymbol{y}^{(t)}, \qquad t = 1,2,\dots$$

or

$$\boldsymbol{F}_t^*\left(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \dots, \boldsymbol{x}^{(t)}\right) = \boldsymbol{y}^{(t)}, \qquad t = 1,2,\dots$$

Recurrent Neural Networks

Sharing Parameters in Time

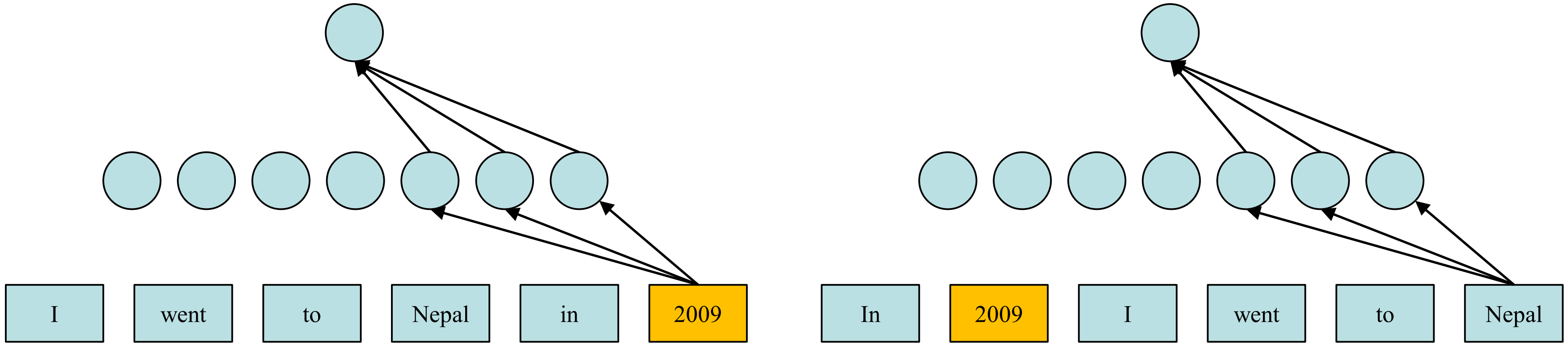
Suppose we want to extract information from two sentences

"I went to Nepal in 2009"

"In 2009 I went to Nepal"

We want a model to behave similarly for these inputs.

Suppose we use a FCNN for this...



Dynamical System

A natural way to share parameters in time is to define a dynamical system depending on some parameter:

$$\boldsymbol{s}^{(t+1)} = \boldsymbol{f}\left(\boldsymbol{s}^{(t)}; \boldsymbol{\theta}\right)$$

- **The vector $s^{(t)}$ is the state of the dynamical system at time t**
- **$\boldsymbol{\theta}$ is a vector of parameters**
- **This is different from "feed-forward" dynamics**

  $$\boldsymbol{h}^{(t+1)} = \boldsymbol{f}\left(\boldsymbol{h}^{(t)}; \boldsymbol{\theta}^{(t)}\right)$$

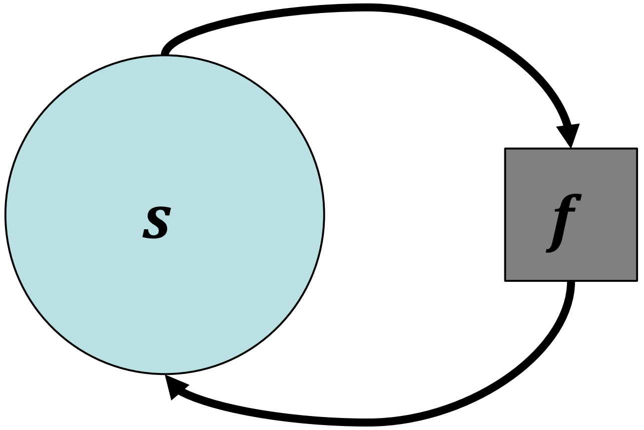
  i.e. the parameter θ is shared in time.

Computational Graph Representation

We can represent the dynamics

$$\boldsymbol{s}^{(t+1)} = \boldsymbol{f}\left(\boldsymbol{s}^{(t)}; \boldsymbol{\theta}\right)$$

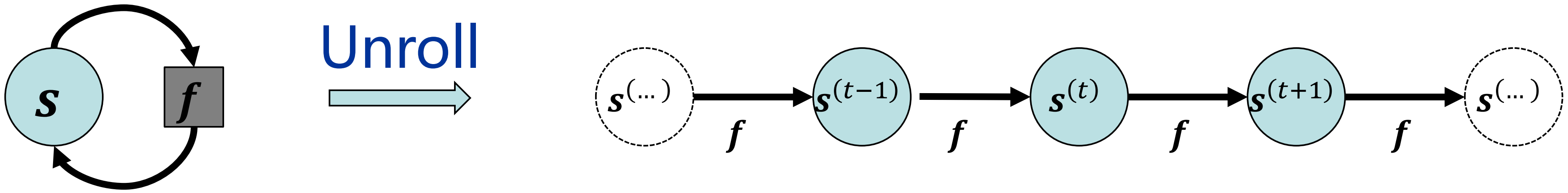
using the usual computational graph approach.



Unrolling the Dynamics

Of course, we can also unroll the dynamics

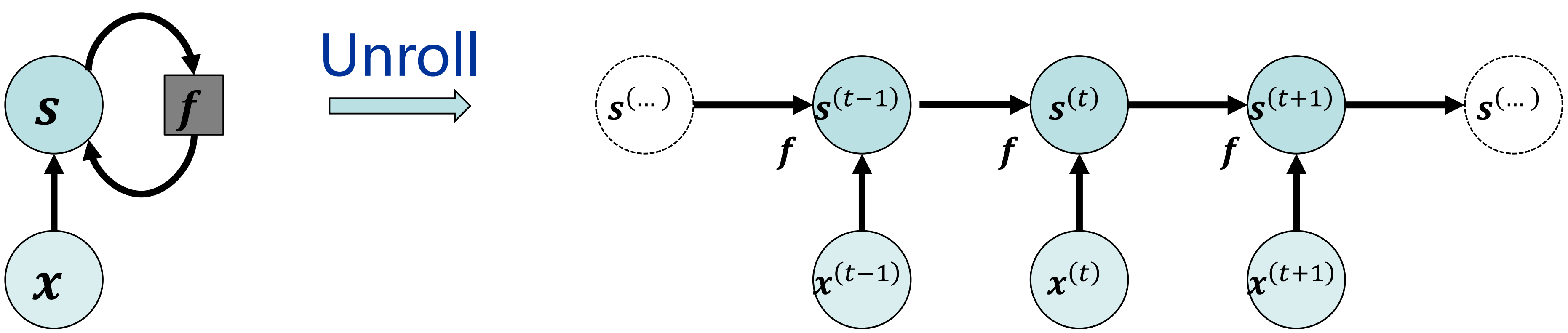
$$\boldsymbol{s}^{(t)} = \boldsymbol{f}\left(\boldsymbol{s}^{(t-1)}; \boldsymbol{\theta}\right) = \boldsymbol{f}\left(\boldsymbol{f}\left(\boldsymbol{s}^{(t-2)}; \boldsymbol{\theta}\right); \boldsymbol{\theta}\right)$$
$$= \cdots = \boldsymbol{f}\left(\boldsymbol{f}\left(\cdots \boldsymbol{f}\left(\boldsymbol{s}^{(0)}; \boldsymbol{\theta}\right) \cdots ; \boldsymbol{\theta}\right); \boldsymbol{\theta}\right)$$



Dynamical Systems with Inputs

Now, we consider a dynamical system with inputs or external forcing

$$\boldsymbol{s}^{(t)} = \boldsymbol{f}\left(\boldsymbol{s}^{(t-1)}, \boldsymbol{x}^{(t)}, \boldsymbol{\theta}\right)$$



Recurrent Neural Networks

The basic architecture of recurrent neural networks uses a forced, hidden dynamical system as a basic hypothesis space

$$\boldsymbol{h}^{(t)} = \boldsymbol{f}\left(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}, \boldsymbol{\theta}\right)$$
$$\widehat{\boldsymbol{y}}^{(t)} = \boldsymbol{g}(\boldsymbol{h}^{(t)}, \boldsymbol{\phi})$$

This implicitly parameterizes

$$\widehat{\boldsymbol{F}}_t\left(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \dots, \boldsymbol{x}^{(t)}; \boldsymbol{\theta}, \boldsymbol{\phi}\right) = \widehat{\boldsymbol{y}}^{(t)}$$

Simple RNN

To be explicit, we consider the following simple RNN structure:

$$\boldsymbol{h}^{(t)} = \sigma_r\left(W\boldsymbol{h}^{(t-1)} + U\boldsymbol{x}^{(t)} + \boldsymbol{b}\right)$$

$$\widehat{\boldsymbol{y}}^{(t)} = o^{(\boldsymbol{t})} = \sigma_o(V\boldsymbol{h}^{(t)} + \boldsymbol{c})$$

- **The trainable parameters are** $(\boldsymbol{\theta}, \boldsymbol{\phi}) = (W, U, \boldsymbol{b}, V, \boldsymbol{c})$
- **The recurrent activation** σ_r **is usually taken to be tanh**
- **The output activation function** σ_o **depends on application**
- **This is the Elman variant of RNN. The Jordan variant replaces** h^{t-1} **by** o^{t-1} **in the first equation**

Loss Functions

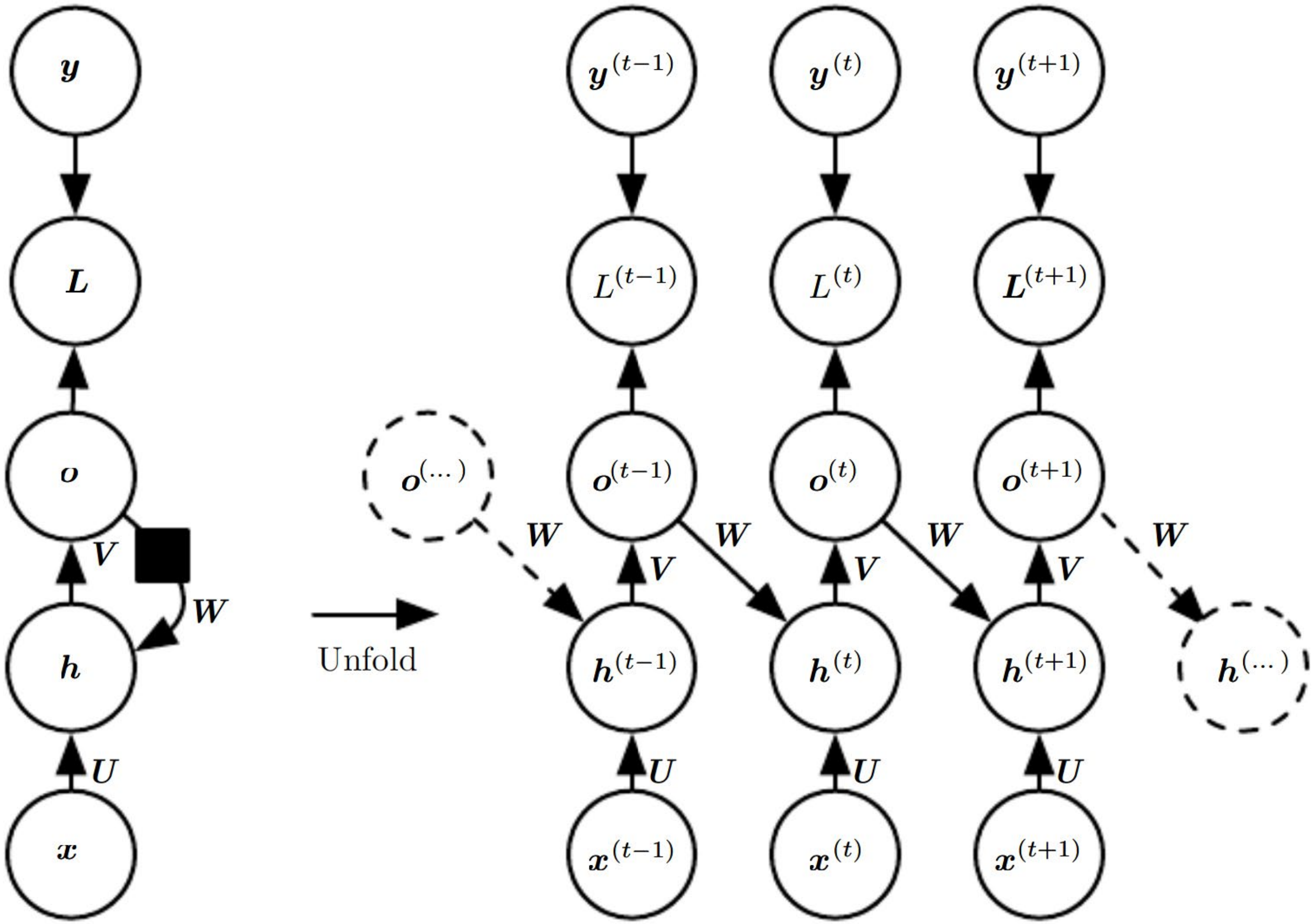
For single-prediction tasks (Tasks I and II), the loss function may be defined only at the end of the sequence, e.g.

$$L\left(\boldsymbol{y}, \widehat{\boldsymbol{y}}^{(\tau)}\right) = \frac{1}{2}\left\|\boldsymbol{y} - \widehat{\boldsymbol{y}}^{(\tau)}\right\|^2$$

For sequence-prediction tasks (Task III), we may take the sum

$$\sum_{t=1}^{\tau} L\left(\boldsymbol{y}^{(t)}, \widehat{\boldsymbol{y}}^{(t)}\right) = \frac{1}{2}\sum_{t=1}^{\tau} \|\boldsymbol{y}^{(t)} - \widehat{\boldsymbol{y}}^{(t)}\|^2$$

Computational Graphs (Jordan Variant)



Why a hidden dynamical system?

Consider instead the naive approach of just using FCNN to map each input to each output

$$\widehat{\boldsymbol{y}}^{(t)} = \mathrm{FCNN}\left(\boldsymbol{x}^{(t)}\right)$$

What is wrong?

- The prediction at time t only depends on $\boldsymbol{x}^{(t)}$
- This cannot model systems with memory, no matter how complex FCNN is

Example: Hidden States

Consider generating from $\{x^{(t)}, t \geq 1\}$ the outputs

$$y^{(t)} = x^{(t)} + x^{(t-1)} + x^{(t-2)}, \qquad t \geq 1$$

where we define $x^{(0)} = x^{(-1)} = 0$.

Then, it is obvious that we cannot predict the value of $y^{(t)}$ from just $x^{(t)}$

To account for memory, we can form the linear model

$$\hat{y}^{(t)} = \sum_{s=1}^{t} a^{(s)} x^{(s)}$$

which can learn our system with $a^{(t)} = a^{(t-1)} = a^{(t-2)} = 1$ and $a^{(s)} = 0$ for $s \leq t - 3$

However, there are issues with this approach. The model

$$\hat{y}^{(t)} = \sum_{s=1}^{t} a^{(s)} x^{(s)}$$

operates on variable-length inputs. Also, it is hard to generalize to non-linear models.

Alternative approach: using hidden states

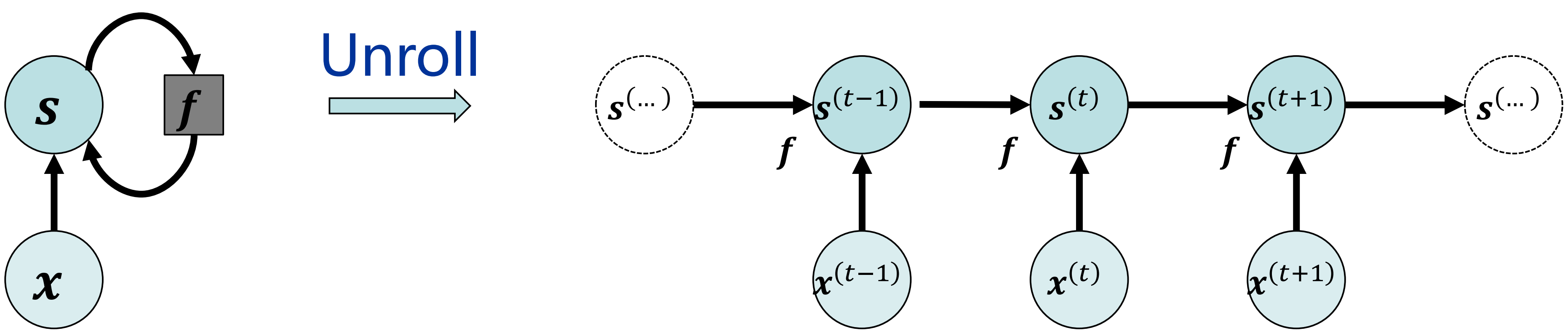
$$\begin{aligned} h_1^{(t)} &= x^{(t)} \\ h_2^{(t)} &= h_1^{(t-1)} \\ h_3^{(t)} &= h_2^{(t-1)} \\ \hat{y}^{(t)} &= h_1^{(t)} + h_2^{(t)} + h_3^{(t)} \end{aligned}$$

Note: this is simply a linear RNN!

Training Recurrent Neural Networks

Unrolling Computational Graph

Once we unroll the computational graph, then RNN is just a parameter-tied feed-forward NN, thus we can apply back-propagation algorithm



Example: Regression on Linear RNN

Let us take a linear RNN in 1D with no bias

$$h^{(t)} = wh^{(t-1)} + x^{(t)}, \qquad h^{(0)} = 0$$
$$\hat{y}^{(t)} = h^{(t)}$$

and terminal loss $L\left(y, \hat{y}^{(\tau)}\right)$.

Observe we can write

$$h^{(t)} = H\left(h^{(t-1)}, x^{(t)}; w\right), \qquad H(h, x, w) = wh + x$$

Then, we can write

$$\frac{dL}{dw} = \sum_{t=1}^{\tau} \frac{dL}{dh^{(t)}} \frac{\partial H}{\partial w}\left(h^{(t-1)}, x^{(t)}, w\right) = \sum_{t=1}^{\tau} \frac{dL}{dh^{(t)}} h^{(t-1)}$$

Then, it remains to compute

$$p^{(t)} \coloneqq \frac{dL}{dh^{(t)}}$$

which is easily shown to obey the backward recursion

$$p^{(t-1)} = wp^{(t)}, \qquad p^{(\tau)} = \frac{\partial L}{\partial \hat{y}}\left(y, \hat{y}^{(\tau)}\right)$$

This gives

$$\frac{dL}{dw} = \frac{\partial L}{\partial \hat{y}}\left(y, \hat{y}^{(\tau)}\right) \sum_{t=1}^{\tau} w^{\tau - t}\, h^{(t-1)}$$

Gradient Explosion and Vanishing

A big problem that plagues RNNs is the difficulty to train them.

One is often faced with a dilemma for large τ:

- Gradient explosion: for some parameter values, the gradients computed using the RNN diverges
- Gradient vanishing: for other parameter values, the gradients computed using the RNN gives vanishing weight to faraway inputs

Example: Gradient Explosion/Vanishing

Let us go back to our previous example, where we computed

$$\frac{dL}{dw} = \sum_{t=1}^{\tau} w^{\tau-t} h^{(t-1)}$$

With some simplifications, we can show that

$$\frac{dL}{dw} = \sum_{t=1}^{\tau} (\tau - t) w^{\tau-t-1} x^{(t)}$$

- Now, if $w > 1$, gradients explode for large τ
- If $w < 1$, gradients do not explode, but attaches vanishing weights to earlier inputs,

  i.e. $(\tau - 1)w^{\tau-2}x^{(1)} \ll 1$ for large τ

Theoretical analysis in linear case: https://openreview.net/pdf?id=8Sqhl-nF50

Gated Recurrent Neural Networks

Gating and long-term dependencies

One of the most effective ways to overcome the inability for RNNs to learn long-term dependencies is their gated extensions

The key ideas

- Design RNN variants where gradients neither vanish nor explode
- Achieve this by the construction of “trainable gates”, which controls the flow of information
    - **Gates control the accumulation of information**
    - **When accumulated information is no-longer required, we just forget them**

Long Short Term Memory (LSTM)

LSTM [Hochreiter and Schmidhuber, 1997] is one of the most commonly used gated RNNs.

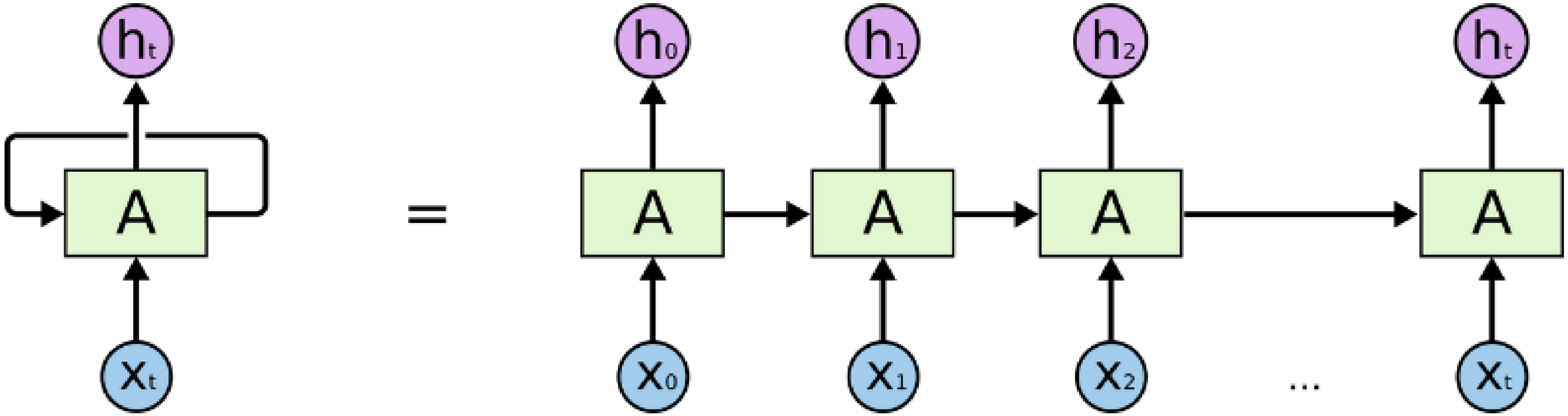
We will use a series of illustrations taken from https://colah.github.io/posts/2015-08-Understanding-LSTMs/ to illustrate the main innovations of LSTM, as well as introducing the abstraction of RNN cells

RNN Cells

RNNs operate by feeding the input and previous hidden states to a function to give the next hidden state, e.g.

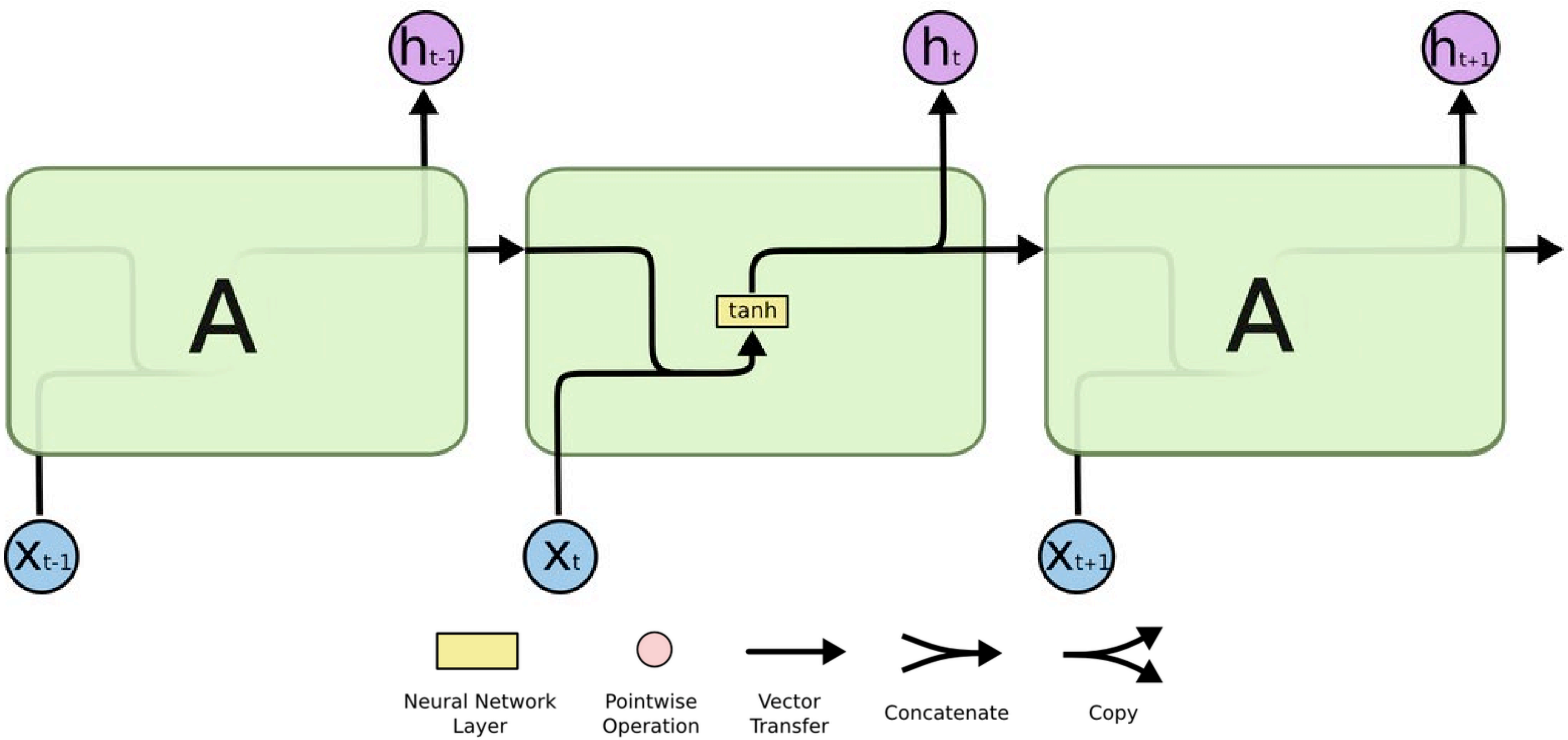
$$\boldsymbol{h}^{(t)} = \tanh\left(W\boldsymbol{h}^{(t-1)} + U\boldsymbol{x}^{(t)} + \boldsymbol{b}\right)$$

We can abstract this operation as a black-box **cell** A

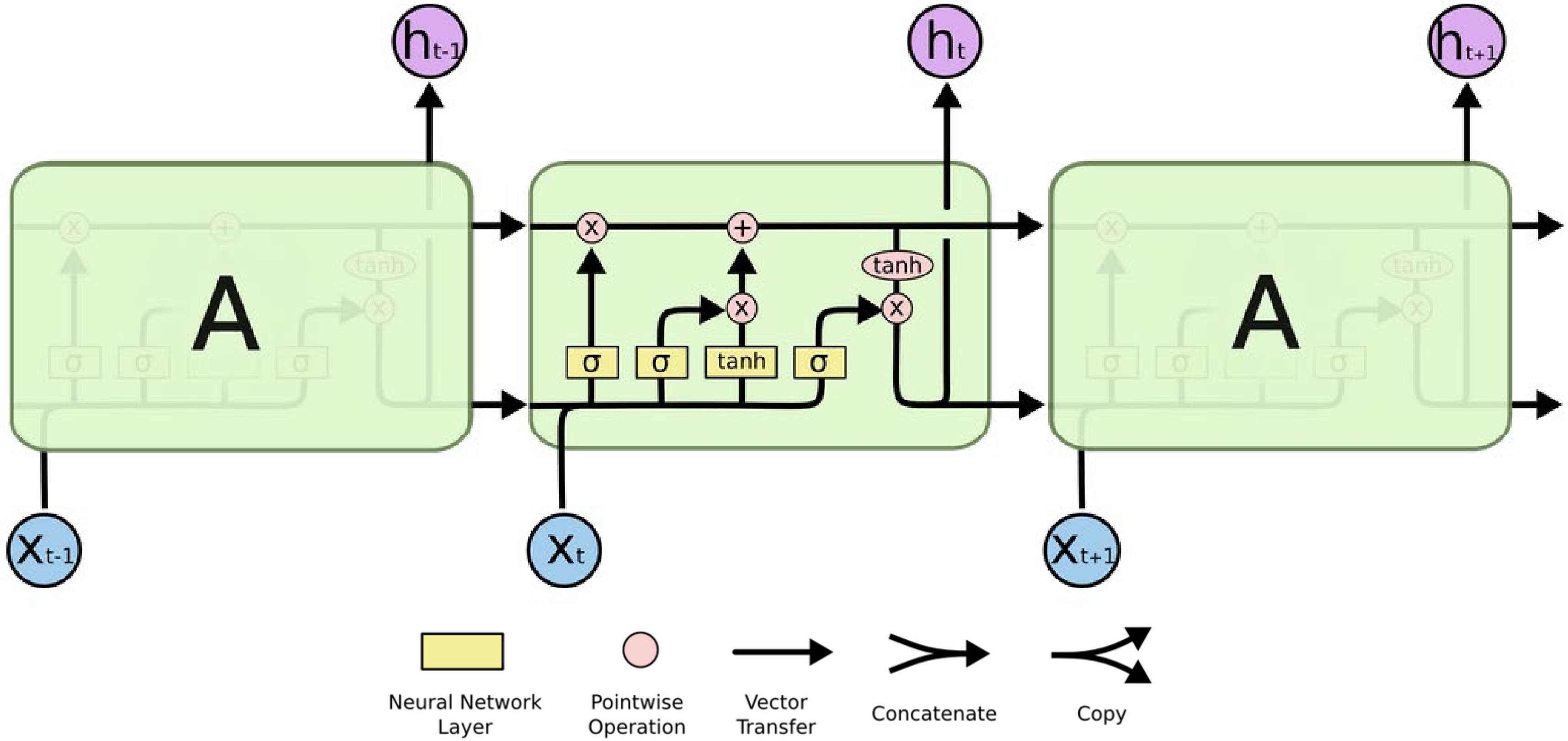


Inside a Simple RNN Cell

$$\boldsymbol{h}^{(t)} = \tanh\left(W\boldsymbol{h}^{(t-1)} + U\boldsymbol{x}^{(t)} + \boldsymbol{b}\right)$$

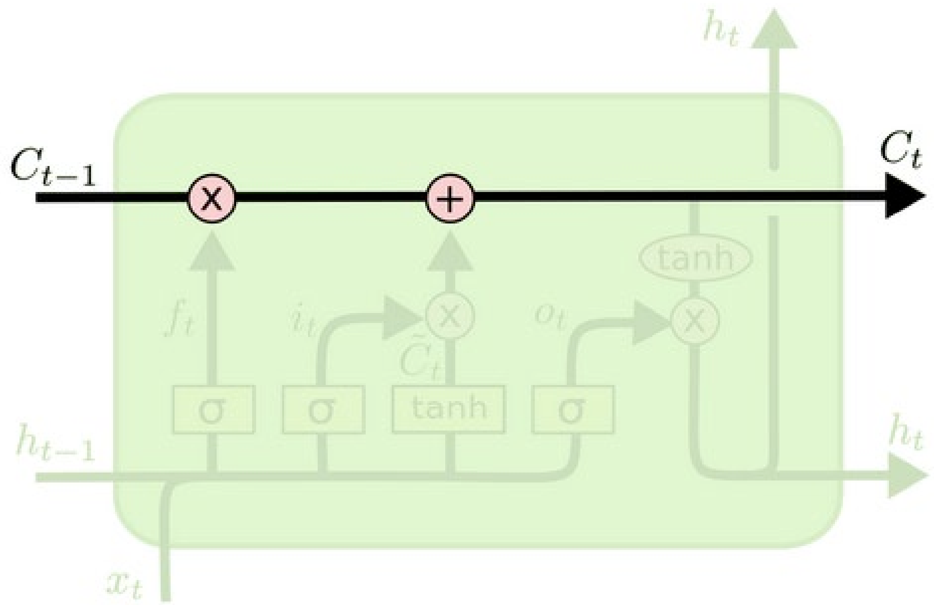


Inside a LSTM Cell



The Cell State

The cell state C_t is another hidden variable that is designed to flow through time with minimal interruptions



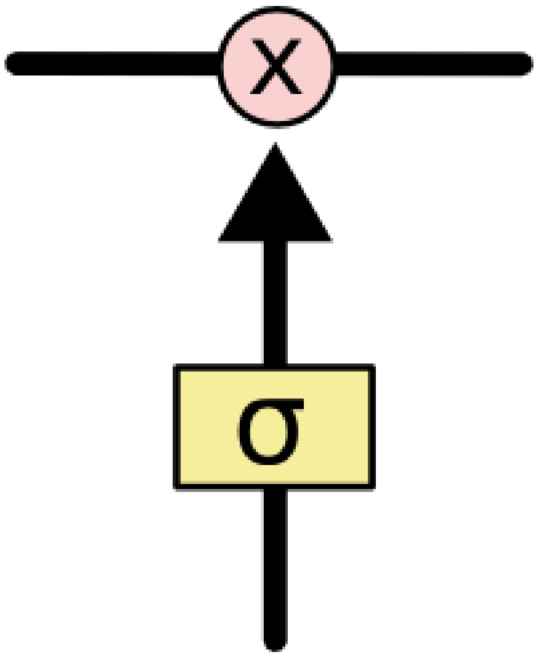
Gates

The LSTM interacts with the cell state using gates, there are just layers with sigmoid outputs

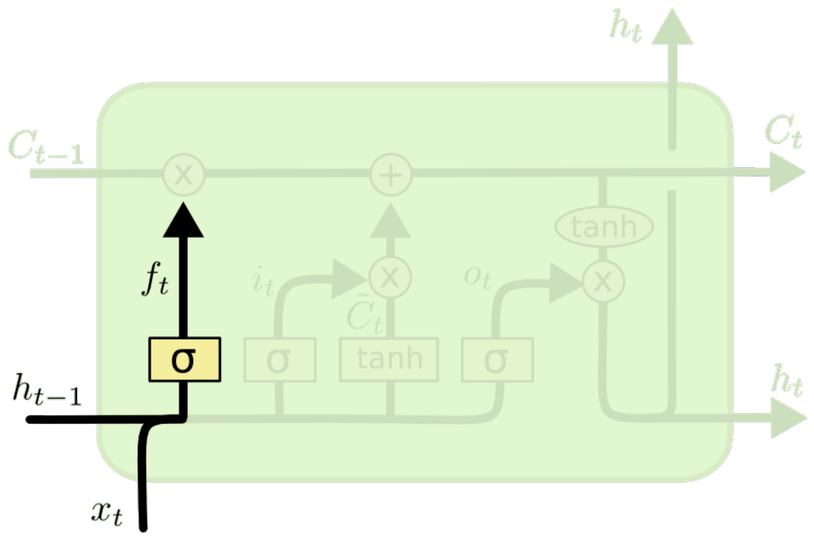
$$\boldsymbol{u} \mapsto \sigma(W\boldsymbol{u} + \boldsymbol{b}),$$

$$\sigma(z) = \frac{1}{1 + e^{-z}} \in (0,1)$$

Multiplying the cell state with the above makes a modification to it

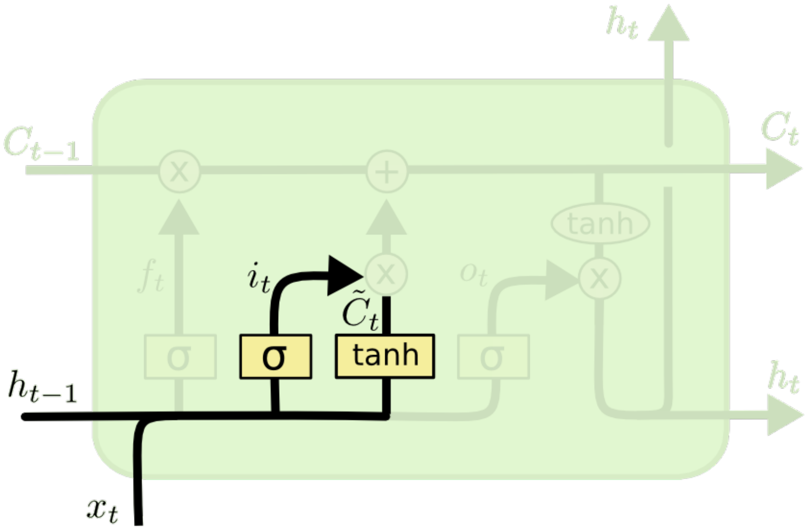


LSTM Step 1



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

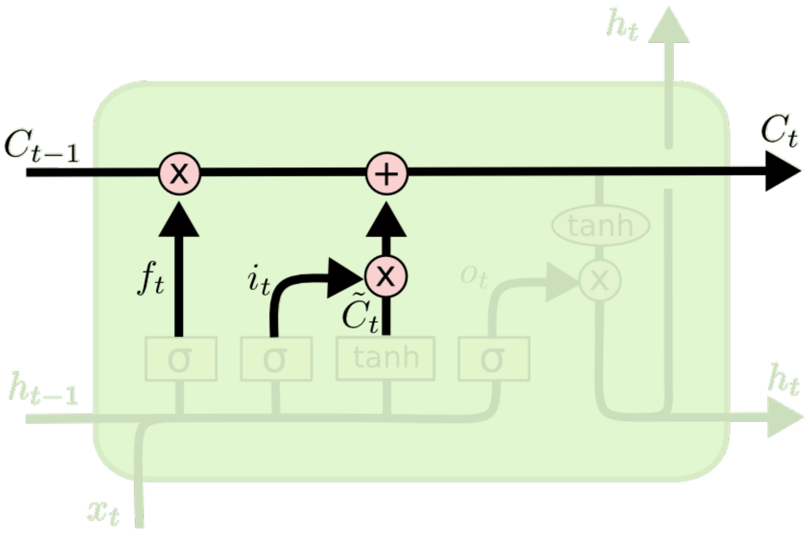
LSTM Step 2



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

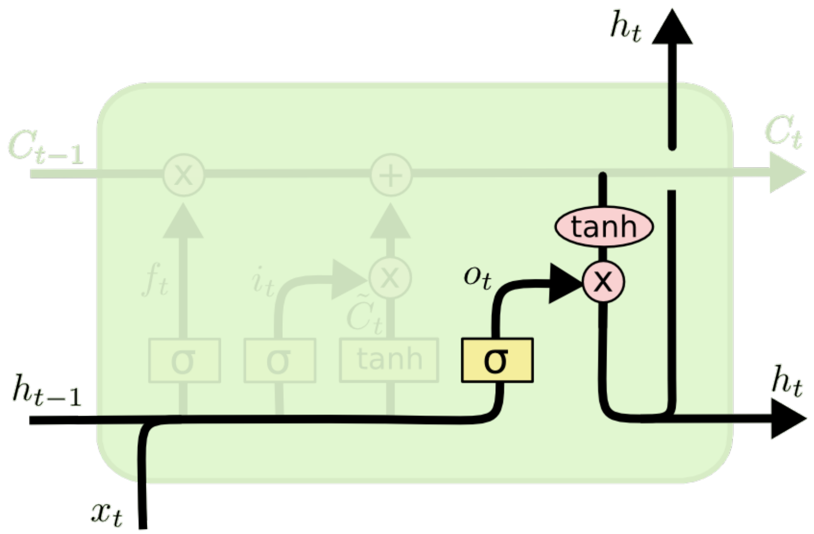
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

LSTM Step 3



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

LSTM Step 4



$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

Summary: LSTM Model (In our Notation)

$$\boldsymbol{f}^{(t)} = \sigma\left(W_f \boldsymbol{h}^{(t-1)} + U_f \boldsymbol{x}^{(t)} + \boldsymbol{b}_f\right)$$
$$\boldsymbol{i}^{(t)} = \sigma\left(W_i \boldsymbol{h}^{(t-1)} + U_i \boldsymbol{x}^{(t)} + \boldsymbol{b}_i\right)$$
$$\boldsymbol{o}^{(t)} = \sigma\left(W_o \boldsymbol{h}^{(t-1)} + U_o \boldsymbol{x}^{(t)} + \boldsymbol{b}_o\right)$$
$$\boldsymbol{C}^{(t)} = \boldsymbol{f}_t \cdot \boldsymbol{C}_{t-1} + \boldsymbol{i}_t \cdot \sigma_c\left(W_c \boldsymbol{h}^{(t-1)} + U_c \boldsymbol{x}^{(t)} + \boldsymbol{b}_c\right)$$
$$\boldsymbol{h}^{(t)} = \boldsymbol{o}^{(t)} \cdot \sigma_h\left(\boldsymbol{C}^{(t)}\right)$$

Gates: $\boldsymbol{f}^{(t)}, \boldsymbol{i}^{(t)}, \boldsymbol{o}^{(t)}$

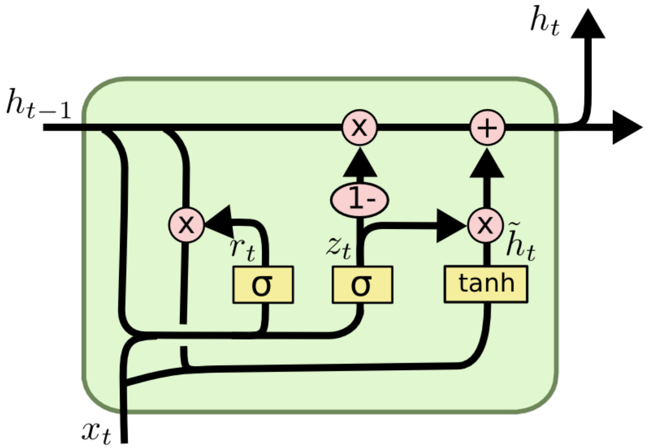
Cell state: $\boldsymbol{C}^{(t)}$

Hidden/RNN state: $\boldsymbol{h}^{(t)}$

Variants of the LSTM architecture

There are many variants

- Peephole connections
- Coupled input/forget gates
- Gated Recurrent Unit (GRU)



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Deep RNNs and Other Architectures

Deep Recurrent Neural Networks

One may observe that the RNN, even in its simplest form, is already deep in the time direction

However, each time-step relied on a single layer of FCNN-like structure

Deep RNNs generalize this by using a deep NN for the recurrent step

The Basic Architecture

Shallow/Simple RNN:

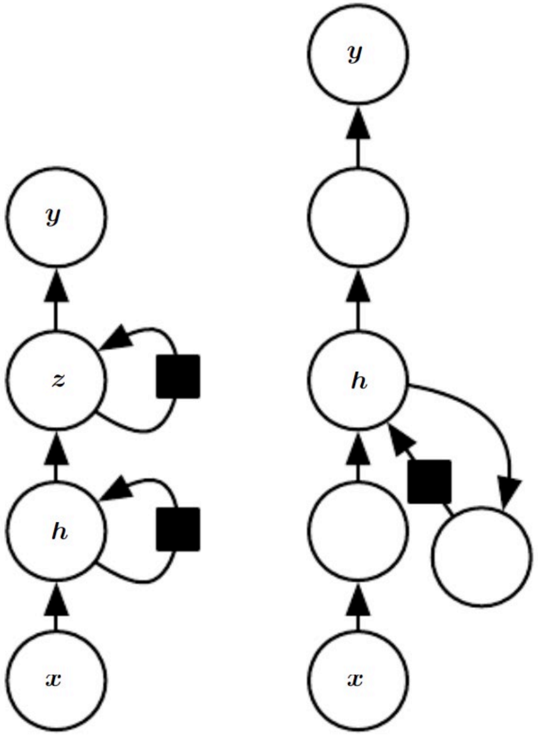
$$\boldsymbol{h}^{(t)} = \sigma_r\left(W\boldsymbol{h}^{(t-1)} + U\boldsymbol{x}^{(t)} + \boldsymbol{b}\right)$$
$$\widehat{\boldsymbol{y}}^{(t)} = \sigma_o(V\boldsymbol{h}^{(t)} + \boldsymbol{c})$$

Deep RNN (One variant):

$$\boldsymbol{h}^{(t)} = \sigma_r\left(W\boldsymbol{h}^{(t-1)} + U\boldsymbol{x}^{(t)} + \boldsymbol{b}\right)$$
$$\boldsymbol{z}^{(t)} = \sigma_r\left(W\boldsymbol{z}^{(t-1)} + U\boldsymbol{h}^{(t)} + \boldsymbol{b}\right)$$
$$\widehat{\boldsymbol{y}}^{(t)} = \sigma_o(V\boldsymbol{z}^{(t)} + \boldsymbol{c})$$

Other Ways to obtain Deep RNN models

(a) Using Stacked Hidden/Cell States



(b) Using MLPs for Connections

Other Important Architectures For Sequence Modelling

There are a number of other architectures for time-series modelling that are useful to know

Look at future, e.g translation.
Non-eg price prediction

- Bi-directional RNNs
- Sequence-to-sequence autoencoders
- Attention mechanisms and the transformer network

More information:

- Deep learning book, Chapter 10
- Graves, A. (2012). Supervised Sequence Labelling with Recurrent Neural Networks . Studies in Computational Intelligence.

Demo: Modelling Time-series with Recurrent Neural Networks

Summary

- **We introduced recurrent neural networks as another type of parameter-sharing network for handling time series data**
- **Primary idea: incorporate memory into a hidden, forced dynamical system**
- **Extensions:**
  - Gated RNNs
  - Deep RNNs