Introduction to Deep Learning

Recurrent Neural Networks for Sequential Data (Time Series)

MATH 370: Machine Learning

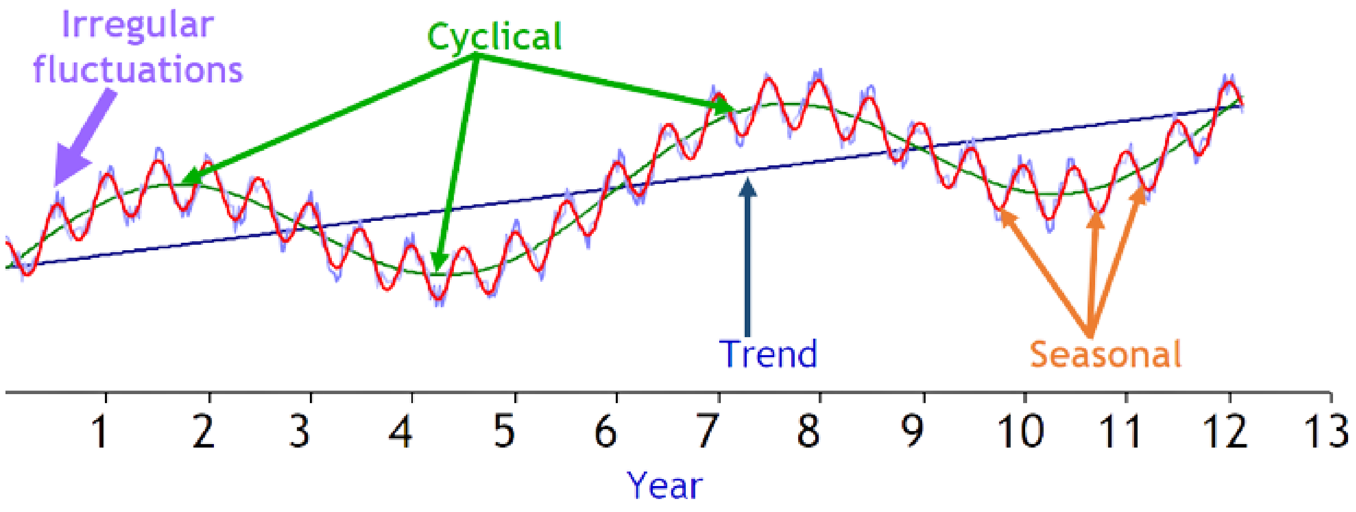
Tanujit Chakraborty

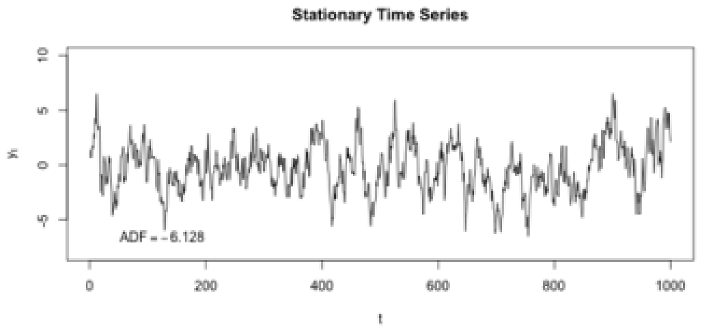
@ Sorbonne

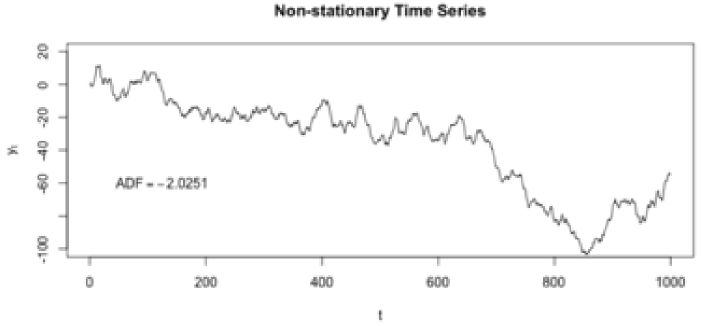
ctanujit@gmail.com

Learning from Time-Series Data

- The input is a sequence of (non-i.i.d.) examples $y_1, y_2, \ldots, y_t$.
- The problem may be supervised or unsupervised, e.g.,
  - Forecasting: Predict y_{t+1} using $y_1, y_2, \ldots, y_t$
  - Cluster the examples or perform dimensionality reduction / Anomaly detection
- Evolution of time-series data can be attributed to several factors





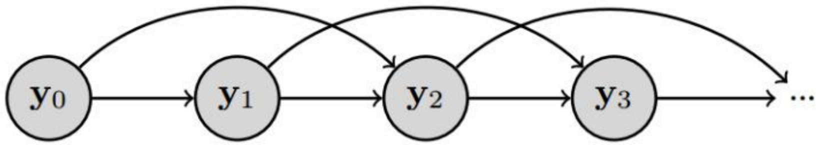


- Teasing apart these factors of variation is also an important problem.

Auto-regressive Models

- Auto-regressive (AR): Regress each example on p previous lagged values - AR(p) model

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \varepsilon_t,$$



Auto-regressive Model (shown above: 2nd order AR)

- Moving Average (MA): Regress each example on q previous stochastic errors - MA(q) model

$$y_t = c + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \cdots + \theta_q \varepsilon_{t-q},$$

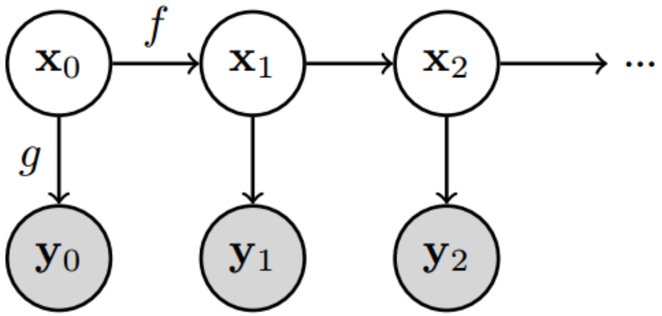
- Auto-regressive Integrated Moving Average (ARMA): Regress each example of p previous lagged values and q previous stochastic errors

$$y'_t = c + \phi_1 y'_{t-1} + \cdots + \phi_p y'_{t-p} + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q} + \varepsilon_t,$$

where y_t' is the differenced series (if the data is nonstationary and differencing is applied). We call this an **ARIMA**(p, d, q) model.

State-Space Models

- Assume that each observation y_t in the time-series is generated by a low-dimensional latent factor x_t (one-hot or continuous)



State-Space Model (shown above: 1st order SSM)

- Basically, a generative latent factor model: $y_t = g(x_t)$ and $x_t = f(x_{t-1})$, where g and f are probability distributions.

- Some popular SSMs: Hidden Markov Models (one-hot latent factor x_t), Kalman Filters (real-valued latent factor x_t)

- Note: Models like RNN/LSTM are also similar, except that these are not generative (but can be made generative)

Long Memory and ARFIMA Process

- Definition (Long Memory). Let $\{X_t, t \in \mathbb{Z}\}$ be a weakly stationary univariate process with auto-covariance function $\gamma_X(k)$ and spectral density function

$$f_X(\lambda) = (2\pi)^{-1} \sum_{k=-\infty}^{\infty} \gamma_X(k) \exp(-ik\lambda) \text{ for } k \in \mathbb{Z} \text{ and } \lambda \in [-\pi, \pi].$$

Then, $\{X_t\}$ has long memory if $\sum_{k=-\infty}^{\infty} |\gamma_X(k)| = \infty$, and short memory otherwise.

Equivalently, as $|\lambda| \to 0$, $f_X(\lambda) \to \infty$ for long memory,

- The most popular long-memory model for level data y_t is the ARFIMA(p, d, q) model introduced by Granger and Joyeux (1980) and Hosking (1981). Specifically, an ARFIMA(p, d, q) process y_t is defined by

$$(1 - B)^d y_t = x_t$$

- B is the one-dimensional backshift operator, x_t is an ARMA(p, q) process that captures short-range dependence, and d is a fractional differencing parameter.

- Typically, d is chosen such that $-1/2 < d < 1/2$ to ensure that y_t is stationary and invertible.

Fractional Differencing

Biometrika (1981), **68**, 1, *pp.* 165–76
Printed in Great Britain

Fractional differencing

BY J. R. M. HOSKING
Institute of Hydrology, Wallingford, Oxfordshire

SUMMARY

The family of autoregressive integrated moving-average processes, widely used in time series analysis, is generalized by permitting the degree of differencing to take fractional values. The fractional differencing operator is defined as an infinite binomial series expansion in powers of the backward-shift operator. Fractionally differenced processes exhibit long-term persistence and antipersistence; the dependence between observations a long time span apart decays much more slowly with time span than is the case with the more commonly studied time series models. Long-term persistent processes have applications in economics and hydrology; compared to existing models of long-term persistence, the family of models introduced here offers much greater flexibility in the simultaneous modelling of the short-term and long-term behaviour of a time series..

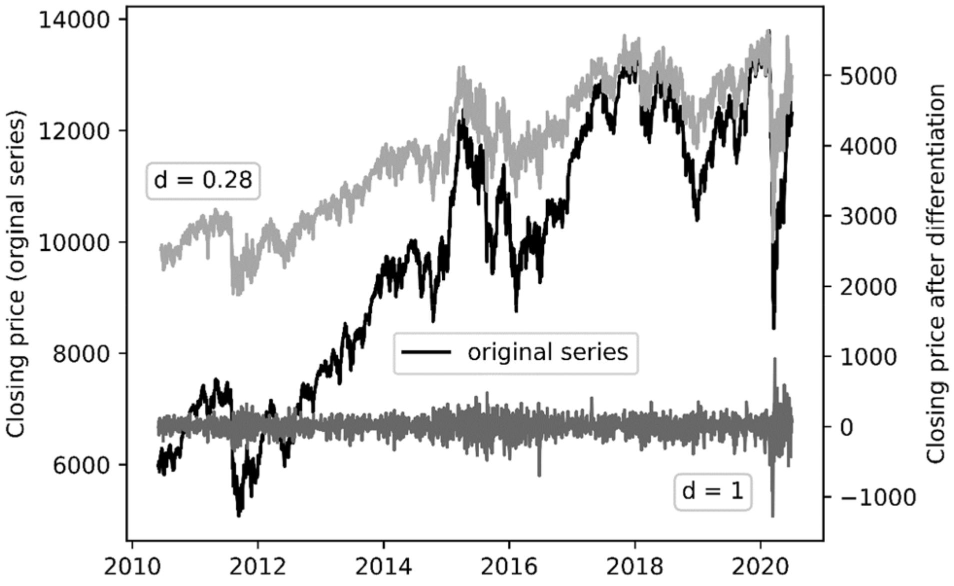
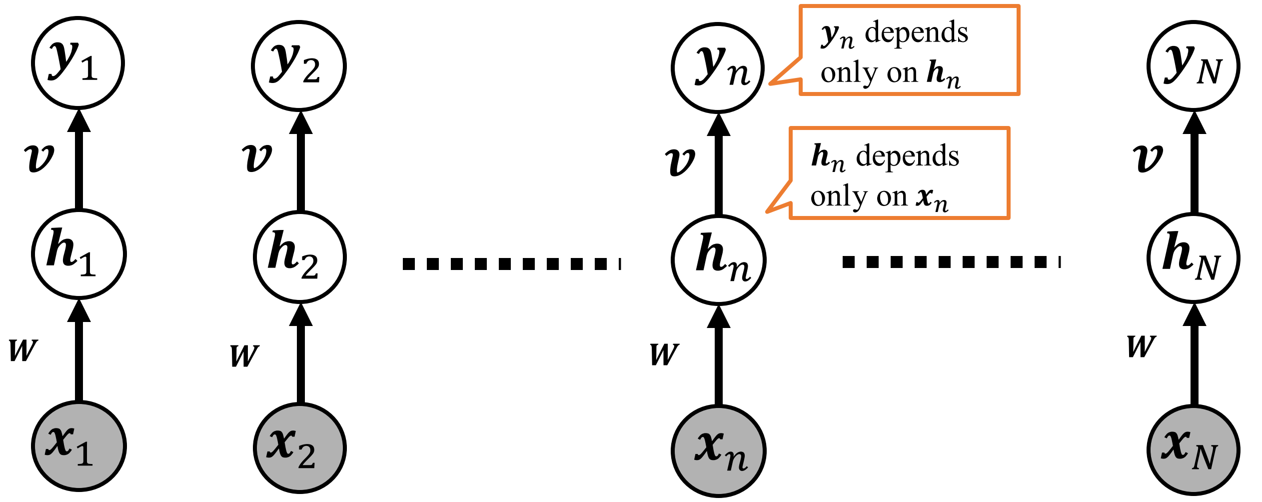


Figure: Fractional Differencing applied to DAX index

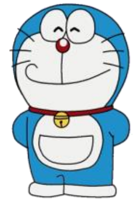
https://www.ma.imperial.ac.uk/~ejm/M3S8/Problems/hosking81.pdf

Recurrent Connections in Deep Neural Networks

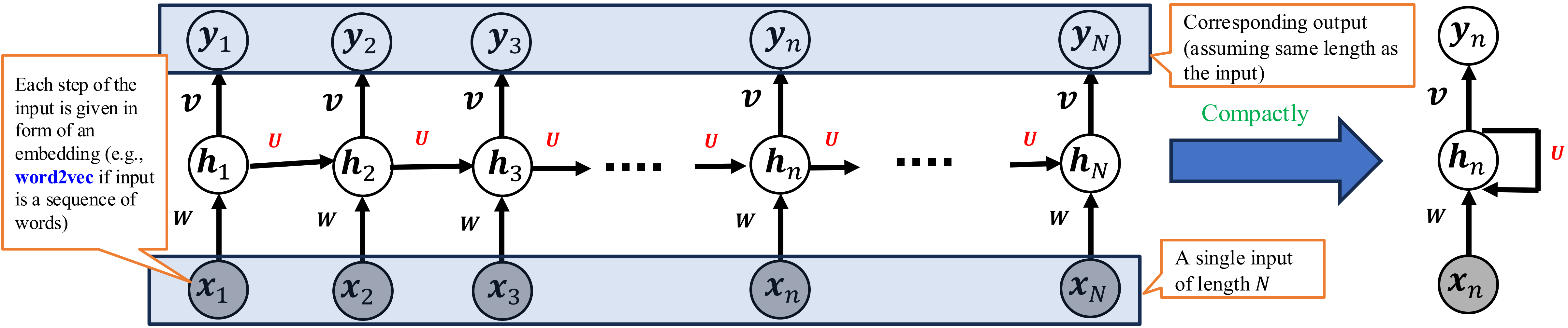
- Feedforward nets such as MLP assume independent observations



Feedforward neural networks are not ideal when inputs $[\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_N]$ and/or outputs $[\boldsymbol{y}_1, \boldsymbol{y}_2, \dots, \boldsymbol{y}_N]$ represent sequential data (e.g., sequence of words, video (sequence of frames), etc.

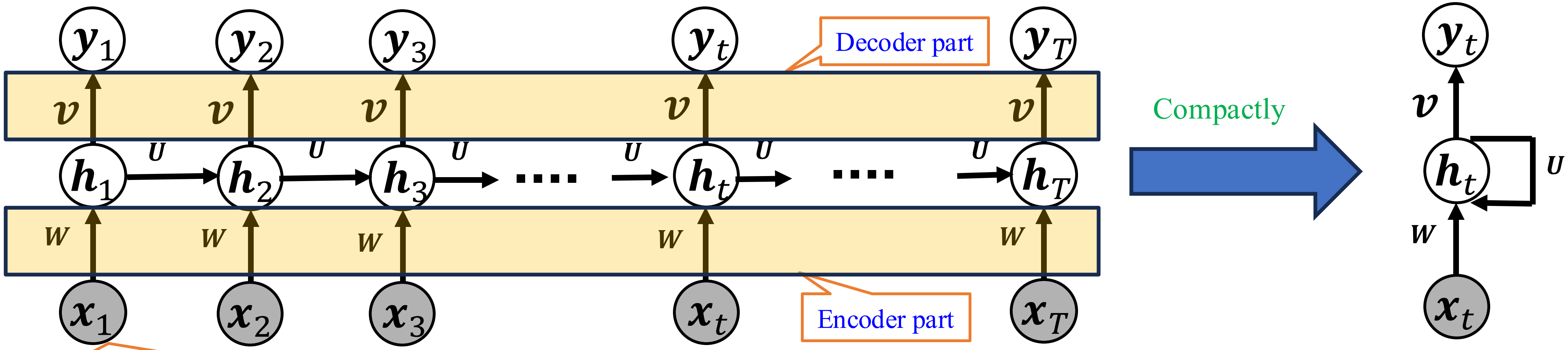


- A recurrent structure can be helpful if each input and/or output is a sequence



RNNs

- RNNs are used when each input or output or both are sequences of tokens

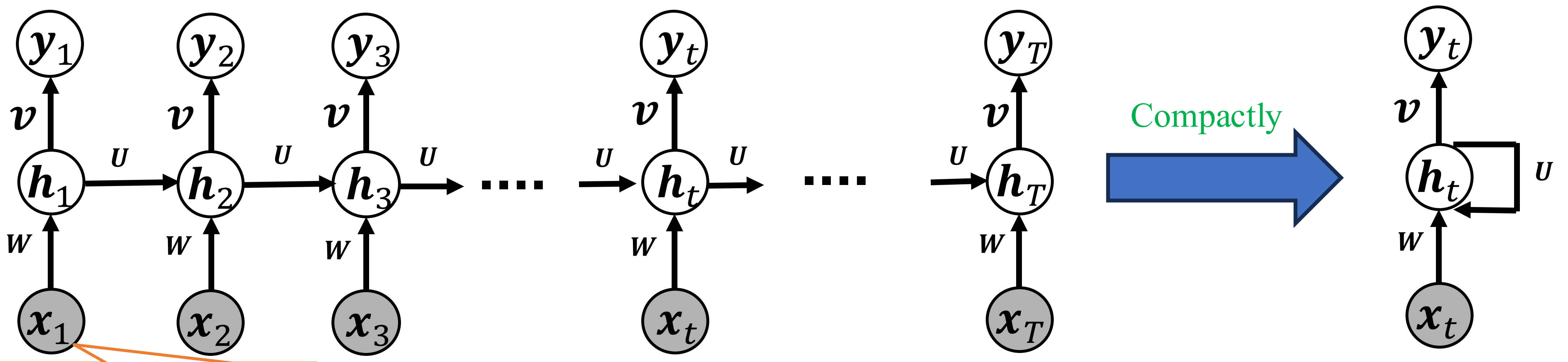


If the input is a word sequence, then each $\boldsymbol{x}_n$ represent the corresponding word's embedding (either a pre-computed word embedding like word2vec or a learned word embedding)

- Hidden state $\boldsymbol{h}_t$ is supposed to remember everything up to time $t-1$. However, in practice, RNNs have difficulties remembering the distant past
  - Variants such as LSTM, GRU, etc mitigate this issue to some extent
- Slow processing is another major issue (e.g., can't compute $\boldsymbol{h}_t$ before computing $\boldsymbol{h}_{t-1}$)

Recurrent Neural Networks

- A basic RNN's architecture (assuming input and output sequence have same lengths)



Given in form of an embedding (e.g., word embedding if x_1 is a word)

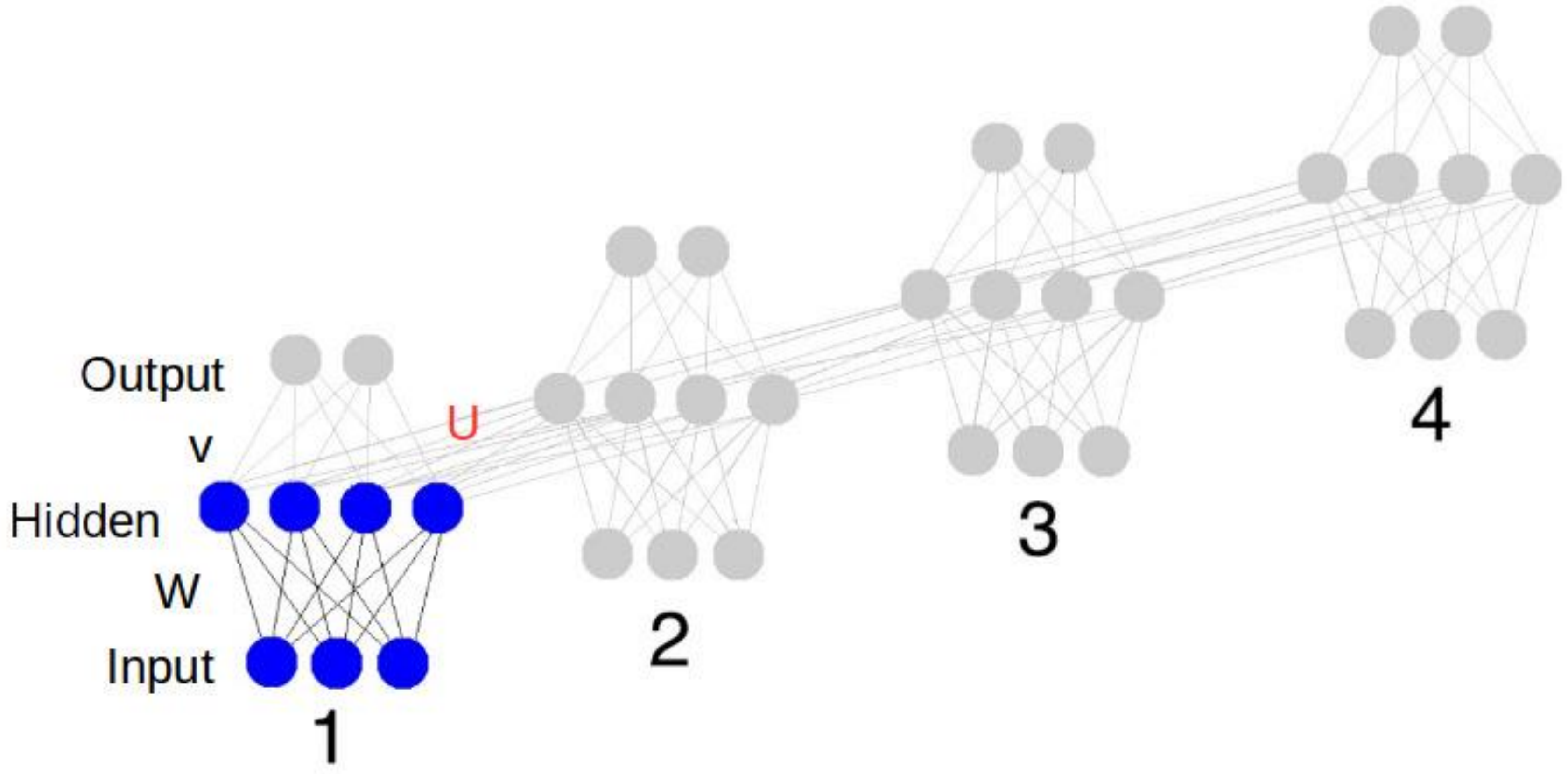
g is some activation function like ReLU

- RNN has three sets of weights $\boldsymbol{W}, \boldsymbol{U}, \boldsymbol{v}$
- $\boldsymbol{W}$ and $\boldsymbol{U}$ model how h_t at step t is computed: $\boldsymbol{h}_t = g(\boldsymbol{W}\boldsymbol{x}_t + \boldsymbol{U}\boldsymbol{h}_{t-1})$
- $\boldsymbol{v}$ models the hidden layer to output mapping, e.g., $\boldsymbol{y}_t = o(\boldsymbol{v}\boldsymbol{h}_t)$
- Important: Same $\boldsymbol{W}, \boldsymbol{U}, \boldsymbol{v}$ are used at all steps of the sequence (weight sharing)

o depends on the nature of y_t. If it is categorical then o can be softmax

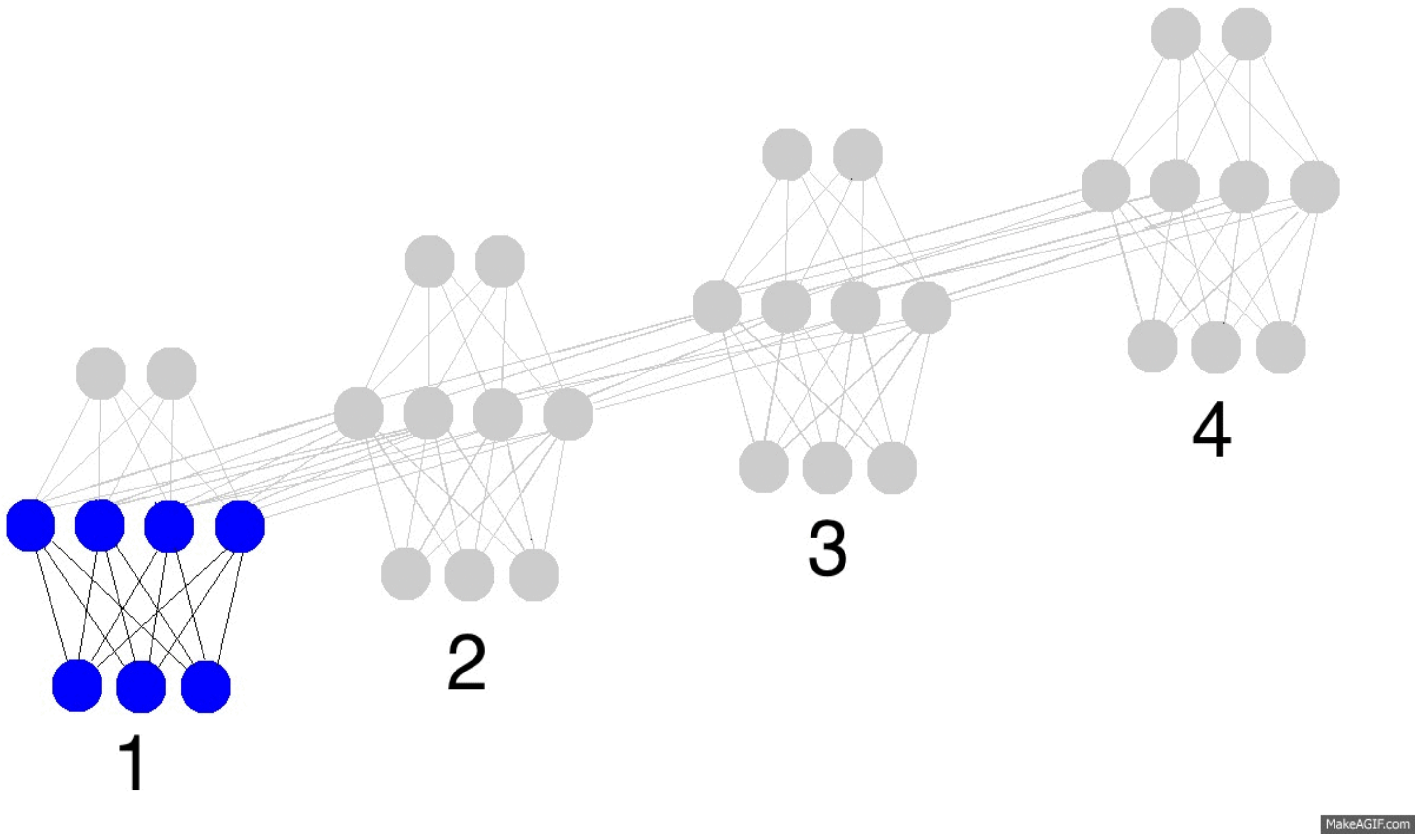
Recurrent Neural Nets (RNN)

- A more "micro" view of RNN (the transition matrix U connects the hidden states across observations, propagating information along the sequence)



Pic source: https://iamtrask.github.io/

RNN in Action



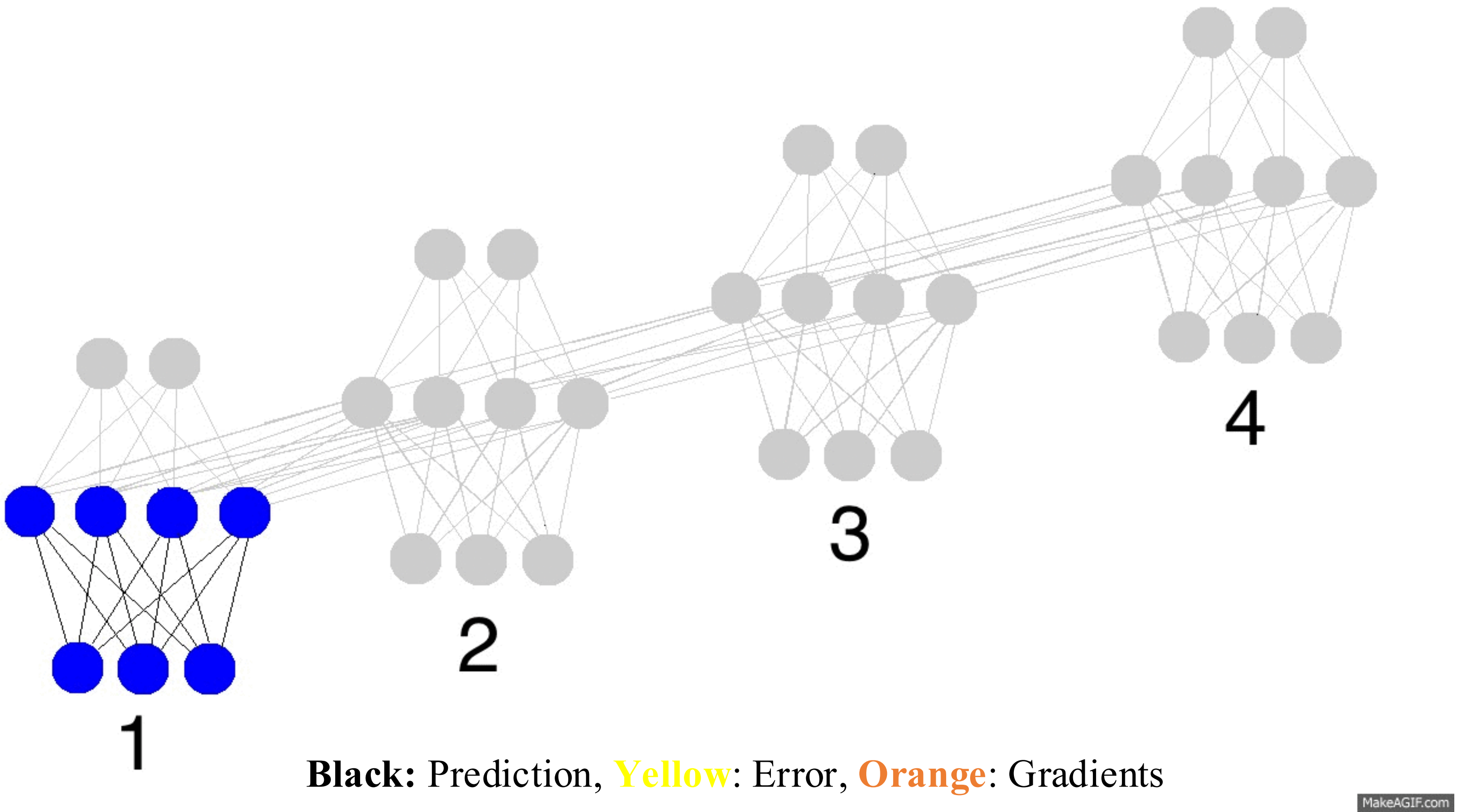
Pic source: https://iamtrask.github.io/

Workflow of RNN:

- The gif above reflects the magic of recurrent networks.
- It depicts 4 timesteps. The first is exclusively influenced by the input data.
- The second one is a mixture of the first and second inputs. This continues on.
- You should recognize that, in some way, network 4 is "full".
- Presumably, timestep 5 would have to choose which memories to keep and which ones to overwrite.
- This is very real. It's the notion of memory "capacity".
- As you might expect, bigger layers can hold more memories for a longer period of time.

Training RNN

- Trained using Backpropagation Through Time (forward propagate from step 1 to end, and then backward propagate from end to step)
- Think of the time-dimension as another hidden layer and then it is just like standard backpropagation for feedforward neural nets



- They learn by fully propagating forward from 1 to 4 (through an entire sequence of arbitrary length), and then backpropagating all the derivatives from 4 back to 1.
- You can also pretend that it's just a funny shaped normal neural network, except that we're re-using the same weights (synapses 0,1,and h) in their respective places.
- Other than that, it's normal backpropagation.

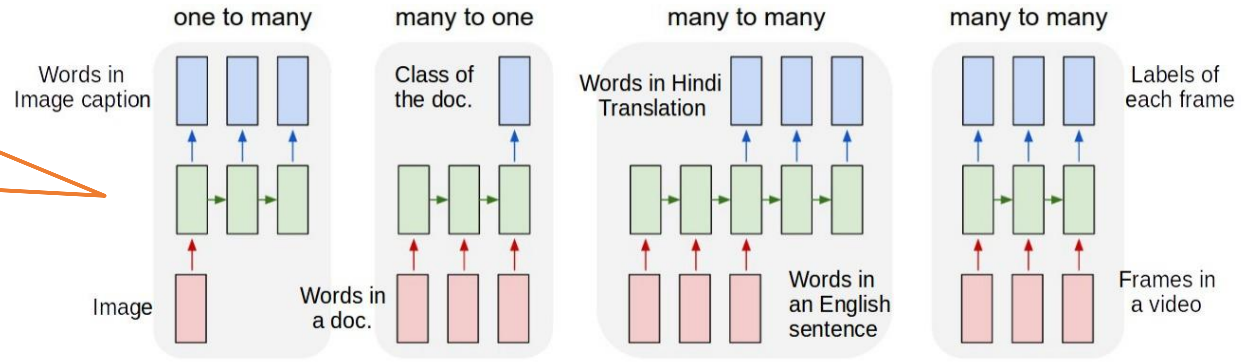
Black: Prediction, Yellow: Error, Orange: Gradients

Pic source: https://iamtrask.github.io/

RNN Applications

- In many problems, each input, each output, or both may be in form of sequences

These green cells are some feature representation (e.g., hidden layer of a deep neural network) of the inputs



- Different inputs or outputs need not have the same length
- Some examples of prediction tasks in such problems
  - Image captioning: Input is image (not a sequence), output is the caption (word sequence)
  - Document classification: Input is a word sequence, output is a categorical label
  - Machine translation: Input is a word sequence, output is a word sequence (in different language)
  - Stock price prediction: Input is a sequence of stock prices, output is its predicted price tomorrow
  - No input – just output (e.g., generation of random but plausible-looking text)

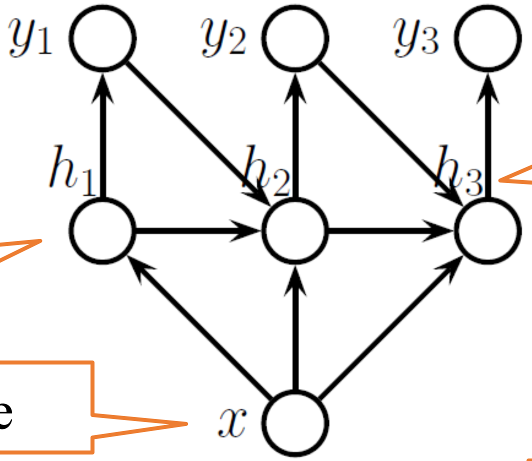
Recurrent Neural Networks: Some Examples

- Consider generating a sequence $y_1, y_2, \dots, y_T$ given an input x

Words in the generated caption

Hidden states at each step of the sequence

An image



Predicted y_{t-1} also fed into h_t

At test time, we can only feed the predicted y_{t-1}

During training, if the true y_{t-1} is fed, we call it "teacher forcing"

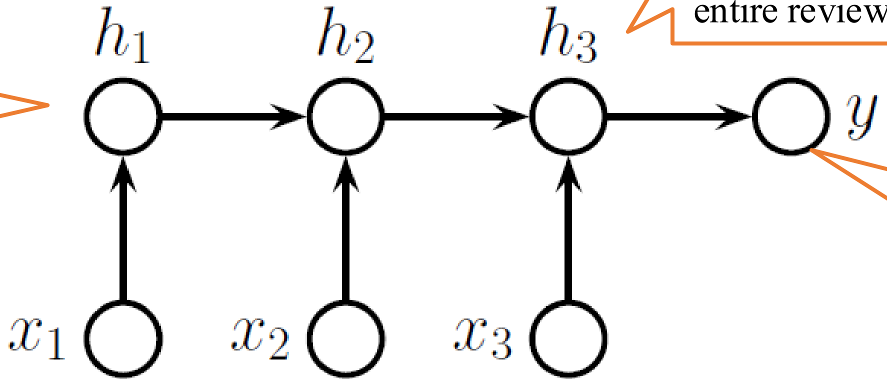
- Predicting the sentiment of a movie review

This final hidden state is supposed to contain the information about the entire review

Isn't this too much to expect?? ☺

Indeed; this can be an issue with RNNs

Hidden states



Predicted sentiment

Each node denotes an embedding of the word

Words in the review

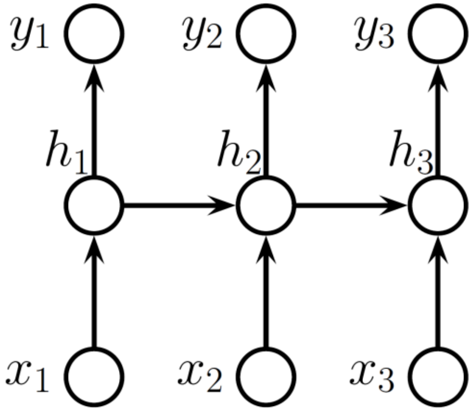
Recurrent Neural Networks: Some Examples

- Parts of speech tagging (or "aligned" translation; input and output have same length)

Parts of speech tag for each word

Each node denotes an embedding of the word

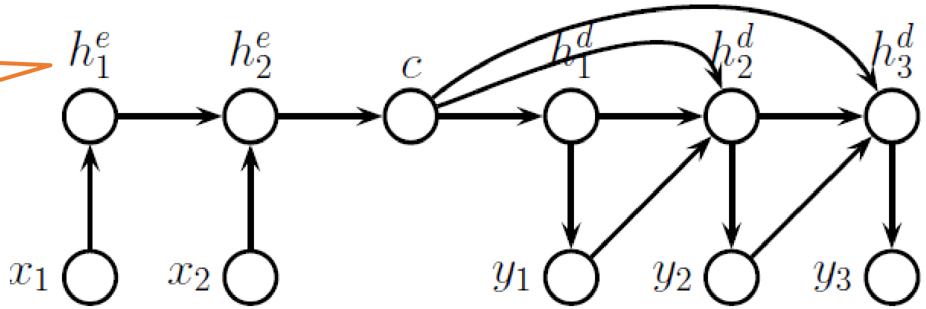
Words in a sentence



- "Unaligned" translation (input and output can have different lengths)

Such problems usually require a sequence encoder- sequence decoder architecture

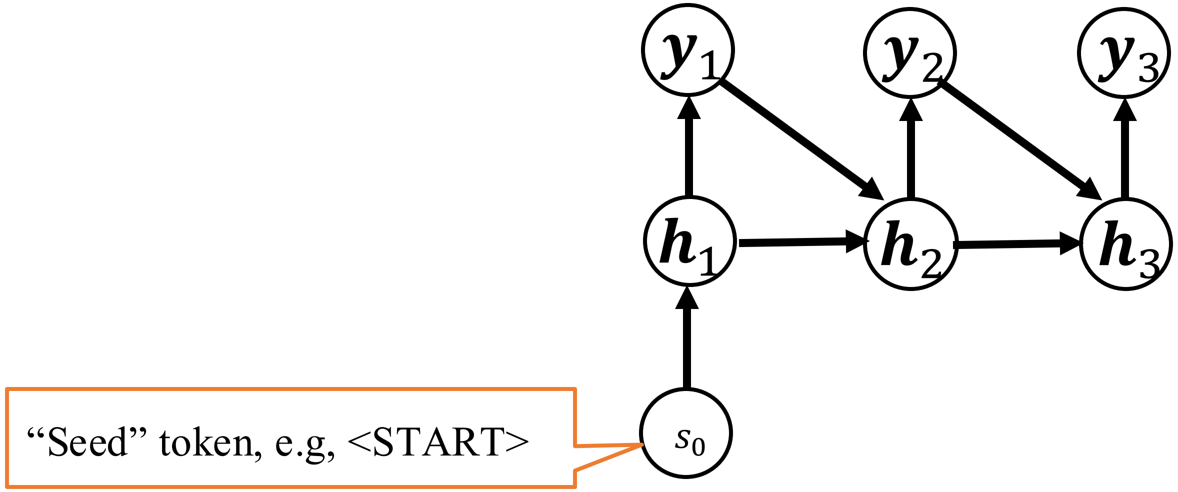
Encode the input sequence (embeddings of tokens) into a single embedding vector c and then decode this embedding one output token at a time



- In the unaligned case, generation stops when an "end" token (e.g., <END>) is generated on the output side

Recurrent Neural Networks: Some Examples

- Unconditional generation (no input, only an output sequence is generated given a RNN that was trained using some training data containing several sequences)

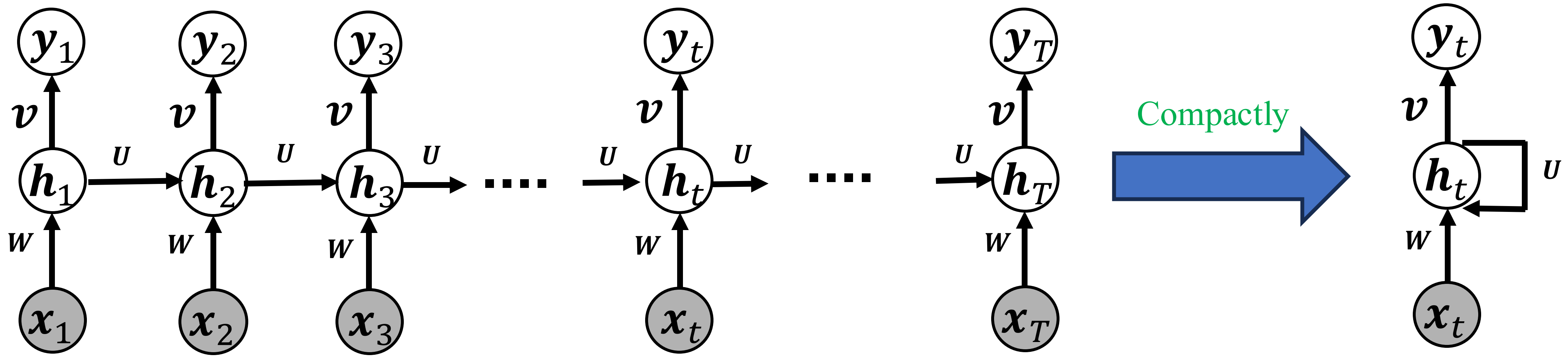


- Each generate word/token is fed to the next step’s hidden state

- Generation stops when an “end” token (e.g., <END>) is generated

For RNNs, Long Distant Past is Hard to Remember

- The hidden layer nodes h_t are supposed to summarize the past up to time $t-1$



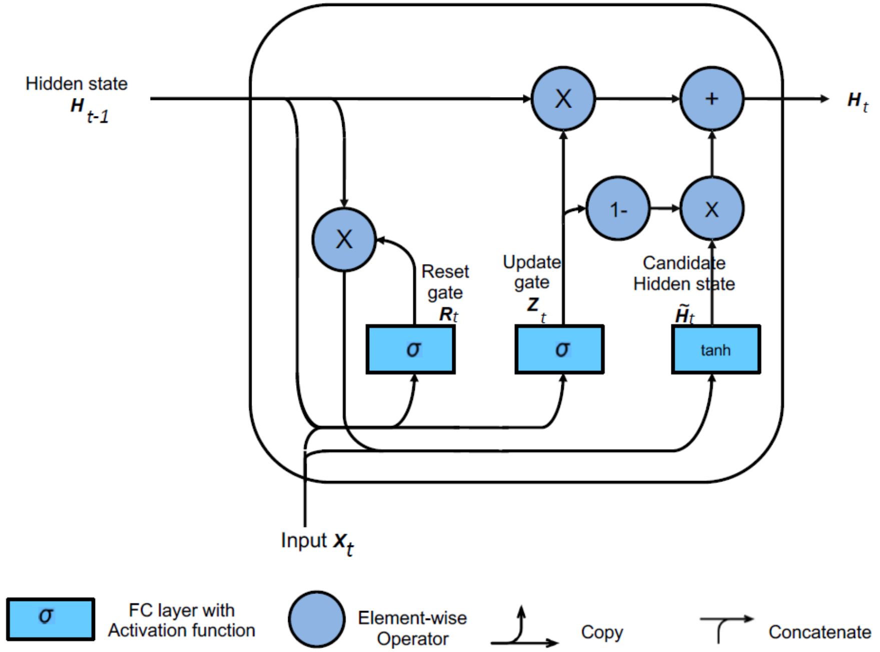
- In theory, they should. In practice, they can't. Some reasons
  - Vanishing gradients along the sequence too (due to repeated multiplications) – past knowledge gets "diluted"
  - Hidden nodes also have limited capacity because of their finite dimensionality
- Various extensions of RNNs have been proposed to address forgetting
  - Gated Recurrent Units (GRU), Long Short Term Memory (LSTM) (Hochreiter and Schmidhuber, mid-90s)

GRU and LSTM

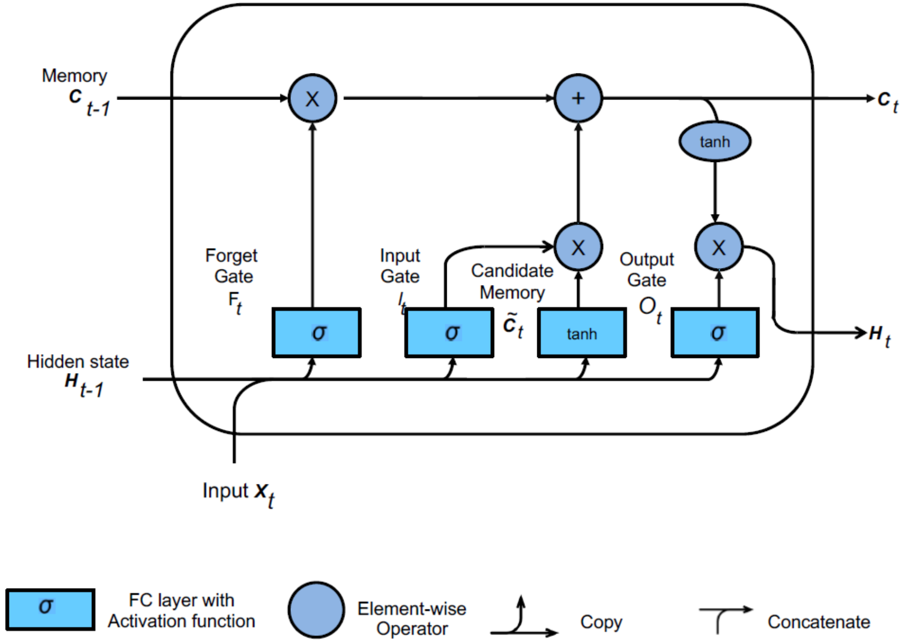
- Essentially an RNN, except that the hidden states are computed differently
- Recall that RNN computes the hidden states as

$$\boldsymbol{h}_t = tanh(\boldsymbol{W}\boldsymbol{x}_t + \boldsymbol{U}\boldsymbol{h}_{t-1})$$

- For RNN: State update is multiplicative (weak memory and gradient issues)
- GRU and LSTM contain specialized units and “memory” which modulate what/how much information from the past to retain/forget

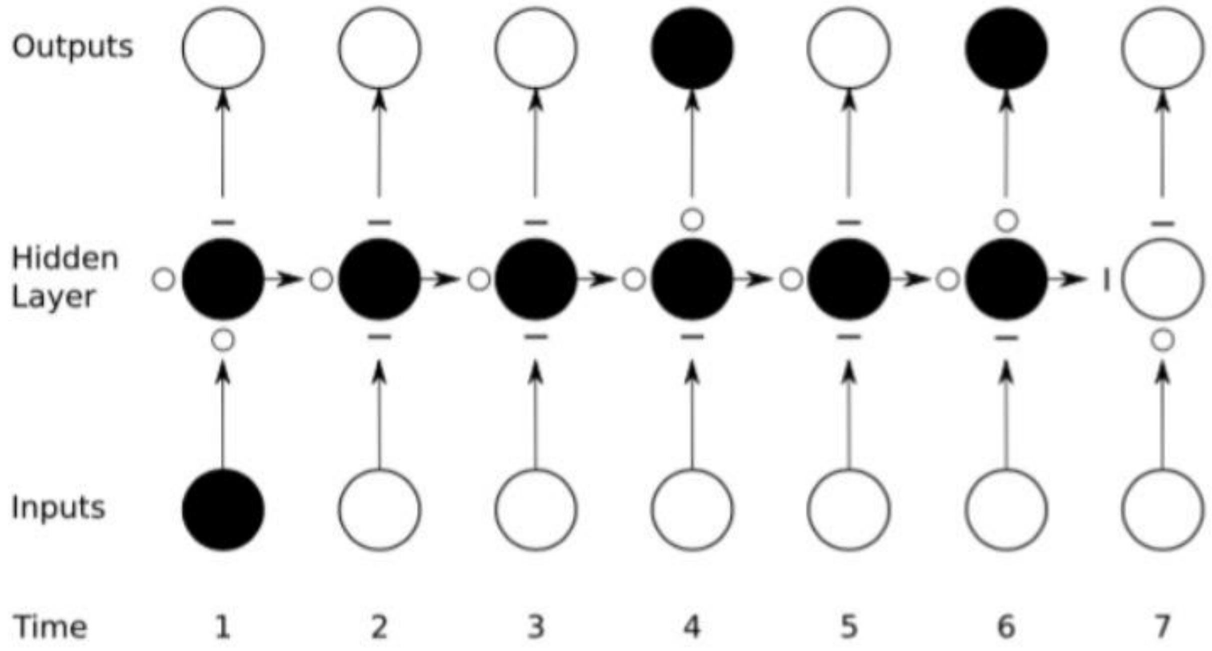


Pic source: https://d2l.ai/



Capturing Long-Range Dependencies

- Idea: Augment the hidden states with gates (with parameters to be learned)
- These gates can help us remember and forget information "selectively"



Pic source: www.deeplearning4j.org

- The hidden states have 3 type of gates: Input (bottom), Forget (left), Output (top)
- Open gate denoted by 'o' closed gate denoted by '-'

LSTM

- In contrast, LSTM maintains a "context" $\boldsymbol{C}_t$ and computes hidden states as

$$
\begin{aligned}
\hat{C}_t &= \tanh(\mathbf{W}^c \boldsymbol{x}_t + \mathbf{U}^c \boldsymbol{h}_{t-1}) && \text{(“local” context, only up to immediately preceding state)} \\
i_t &= \sigma(\mathbf{W}^i \boldsymbol{x}_t + \mathbf{U}^i \boldsymbol{h}_{t-1}) && \text{(how much to take in the local context)} \\
f_t &= \sigma(\mathbf{W}^f \boldsymbol{x}_t + \mathbf{U}^f \boldsymbol{h}_{t-1}) && \text{(how much to forget the previous context)} \\
o_t &= \sigma(\mathbf{W}^o \boldsymbol{x}_t + \mathbf{U}^o \boldsymbol{h}_{t-1}) && \text{(how much to output)} \\
C_t &= C_{t-1} \odot f_t + \hat{C}_t \odot i_t && \text{(a modulated additive update for context)} \\
h_t &= \tanh(C_t) \odot o_t && \text{(transform context into state and selectively output)}
\end{aligned}
$$

- Note: $\odot$ represents elementwise vector product. Also, state updates now additive, not multiplicative. Training using backpropagation through time.

- Many variants of LSTM exists, e.g., using $\boldsymbol{C}_t$ in local computations, Gated Recurrent Units (GRU), etc. Mostly minor variations of basic LSTM above.

Do LSTM really have long memory? (ICML'2020)

Do RNN and LSTM have Long Memory?

Jingyu Zhao [1] Feiqing Huang [1] Jia Lv [2] Yanjie Duan [2] Zhen Qin [2] Guodong Li [1] Guangjian Tian [2]

Abstract

The LSTM network was proposed to overcome the difficulty in learning long-term dependence, and has made significant advancements in applications. With its success and drawbacks in mind, this paper raises the question - do RNN and LSTM have long memory? We answer it partially by proving that RNN and LSTM do not have long memory from a statistical perspective. A new definition for long memory networks is further introduced, and it requires the model weights to decay at a polynomial rate. To verify our theory, we convert RNN and LSTM into long memory networks by making a minimal modification, and their superiority is illustrated in modeling long-term dependence of various datasets.

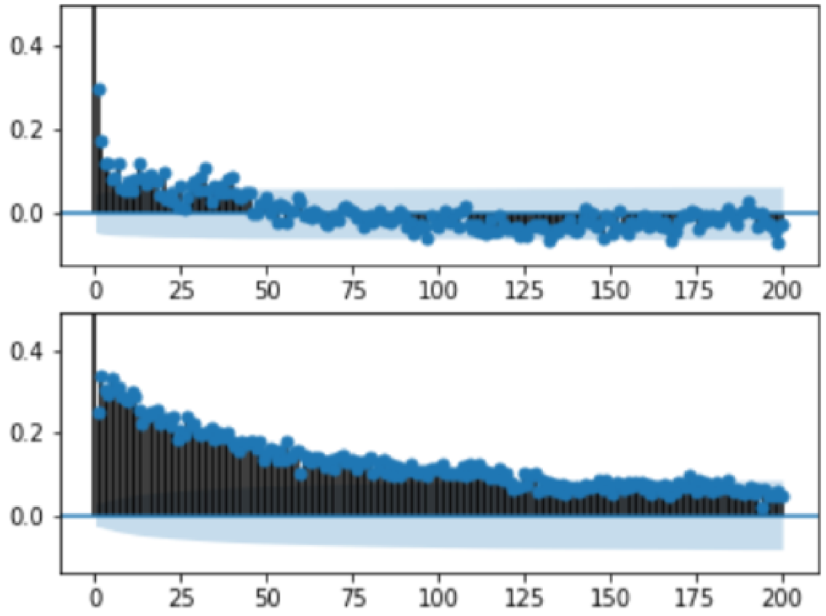


Figure: Autocorrelation plot of traffic and DJI datasets
(To visualize the long memory in the dataset)

Ref: https://proceedings.mlr.press/v119/zhao20c

Memory RNN and Bidirectional RNN

- RNNs and GRU and LSTM only remember the information from the previous tokens
- Memory RNN and Bidirectional RNN can remember information from the past and future tokens

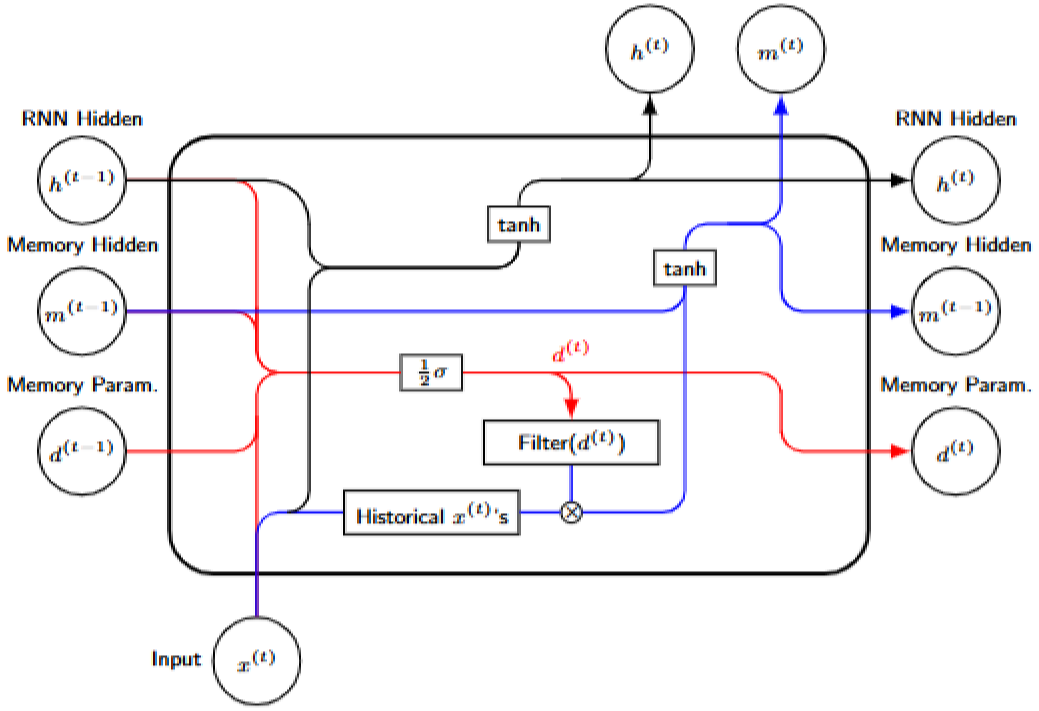
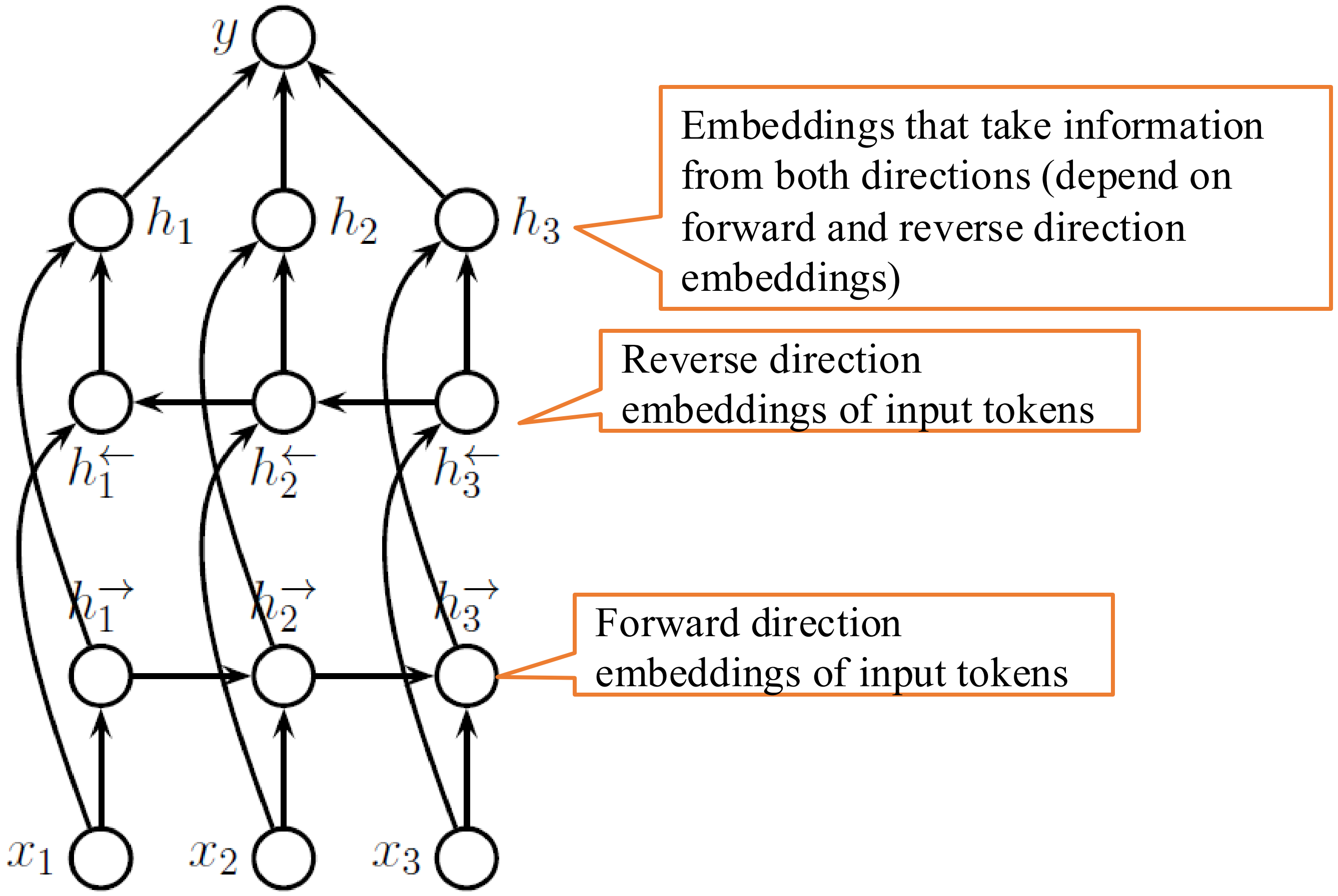


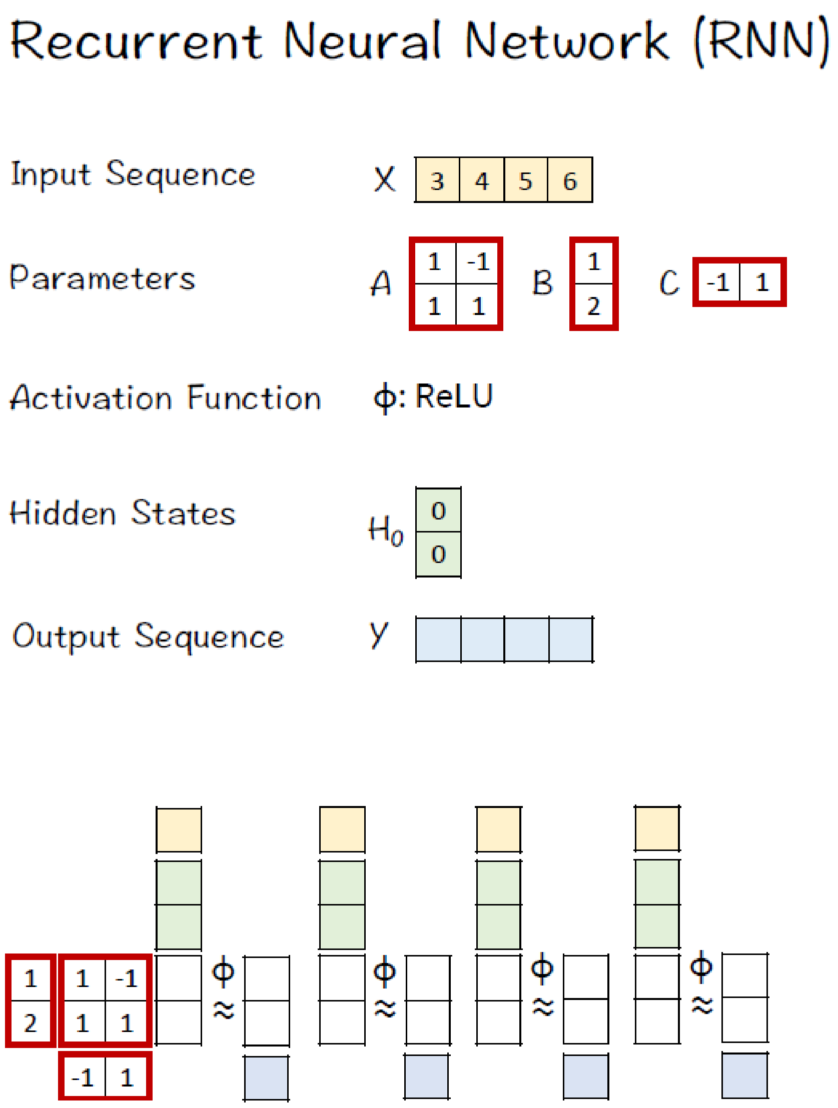
Figure 1. The MRNN cell structure.

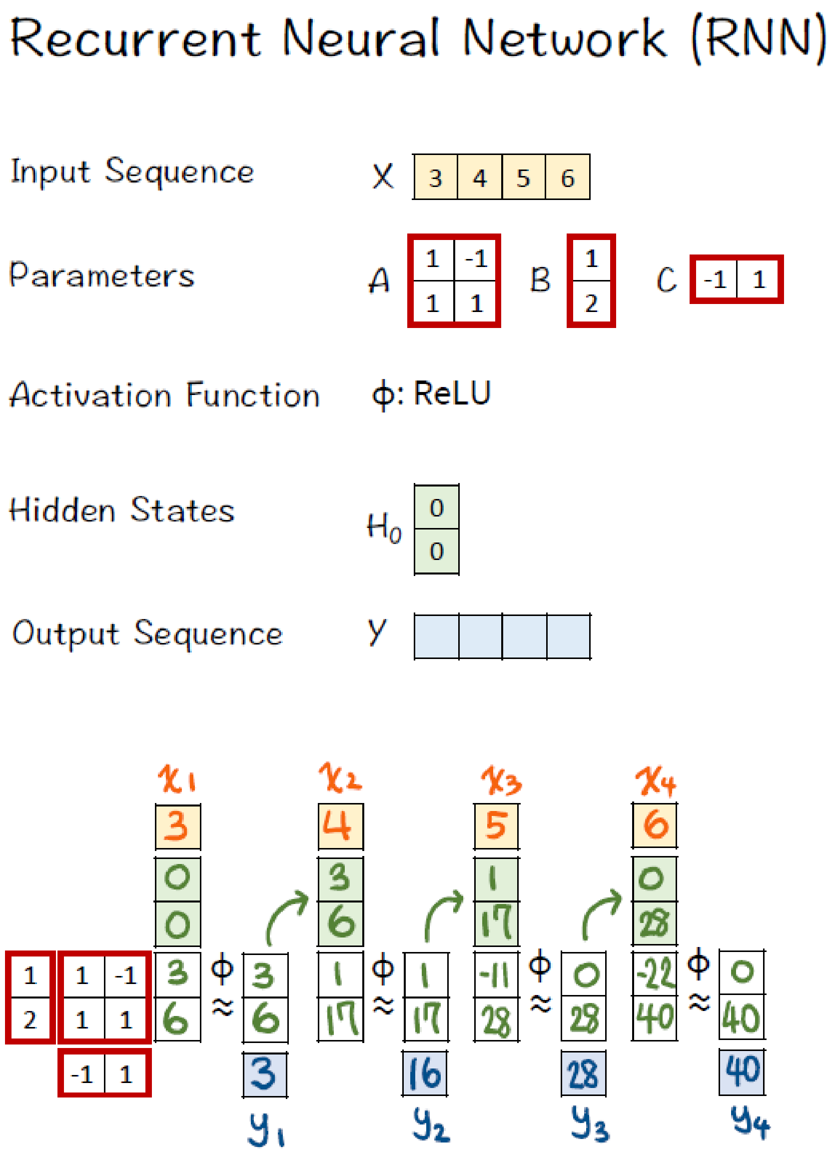
Ref: https://proceedings.mlr.press/v119/zhao20c



Ref: https://ieeexplore.ieee.org/document/650093

Exercise: RNN

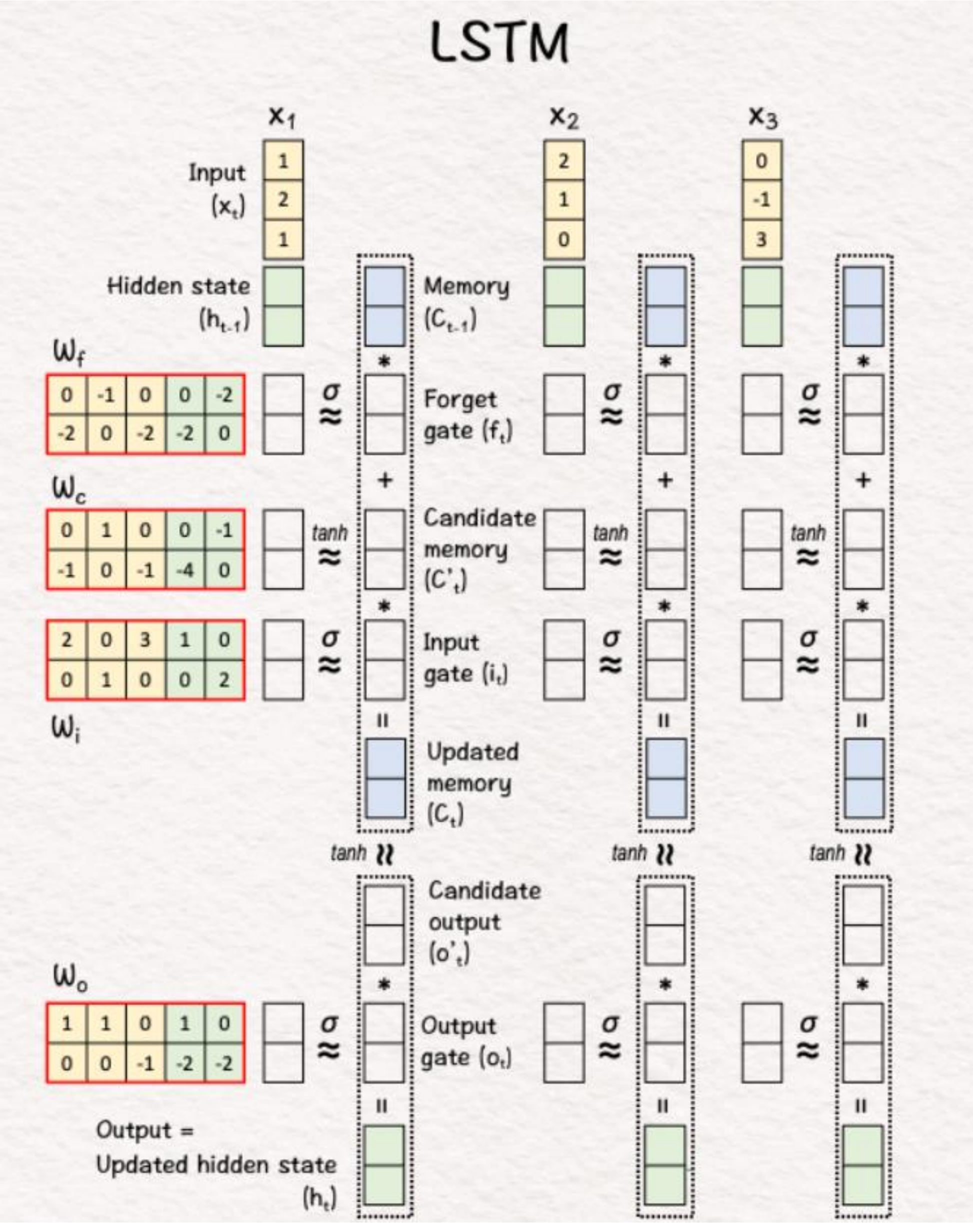


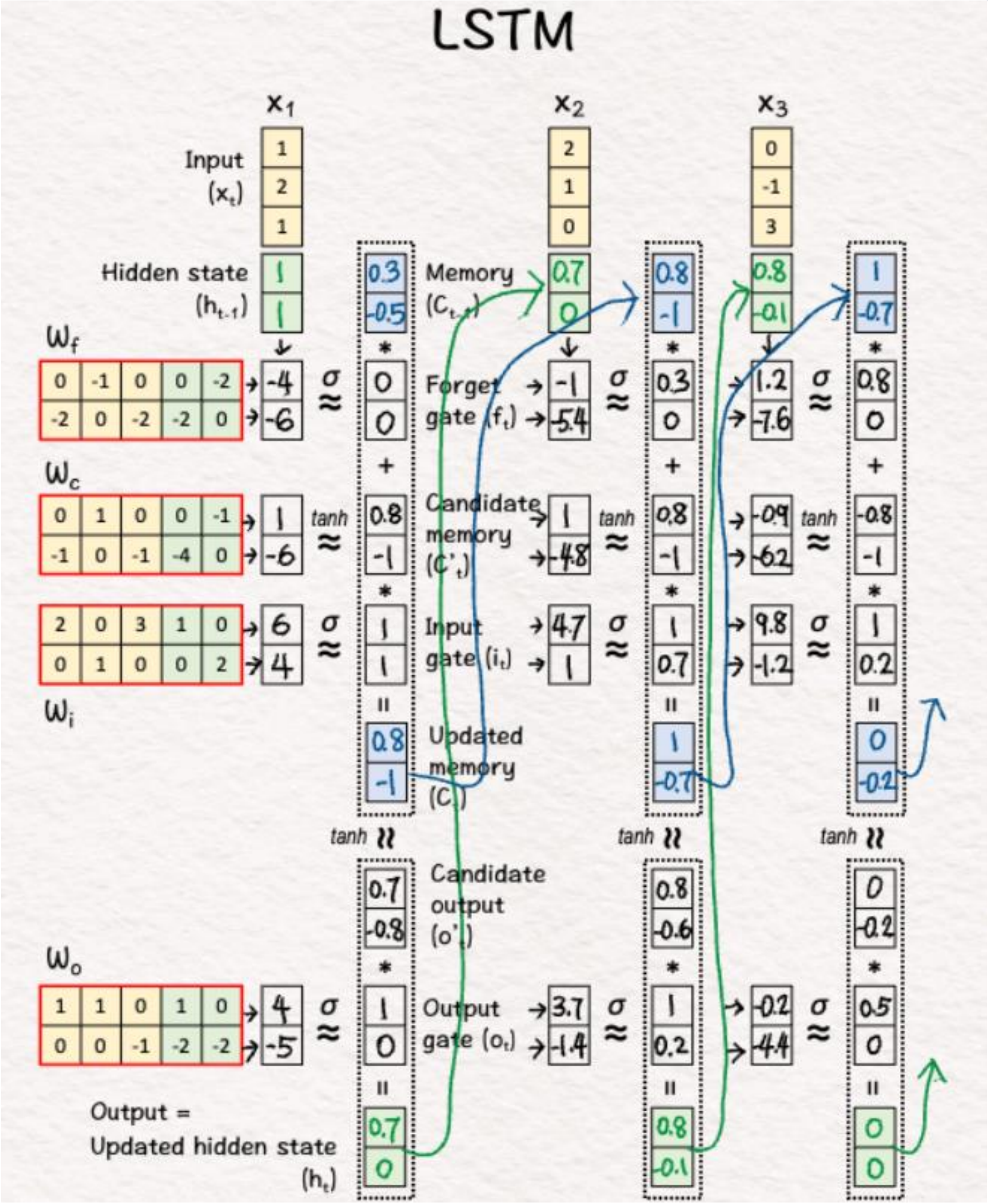


Pic source: https://www.byhand.ai/

Exercise: LSTM

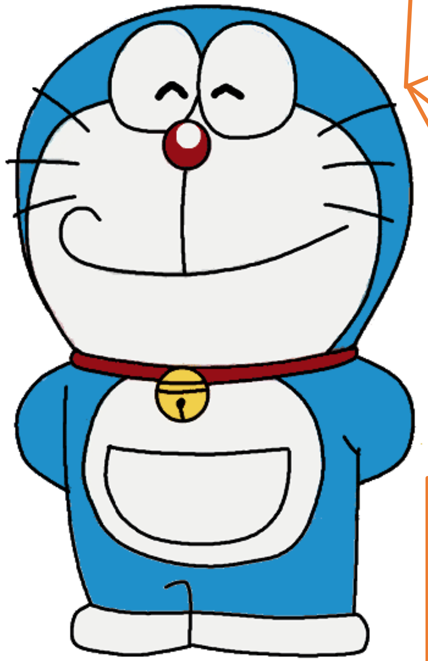
Initialize: Randomly set the previous hidden state h0 to [1, 1] and memory cells C0 to [0.3, -0.5]





Pic source: https://www.byhand.ai/

Any question?



Readings for you:

- [Deep Learning book]
- [Forecasting (FPP) Book using Python]
- [AI by Hand by Tom Yeh]
- Special thanks to *Piyush Rai and Jay Alammar*
  - – I adopted some of their slides available online.