Sequence Models

DSE 3151 DEEP LEARNING

B.Tech Data Science & Engineering

August 2023

Rohini R Rao & Abhilash Pai

Department of Data Science and Computer Applications

MIT Manipal

Slide -2 of 5

Examples of Sequence Data

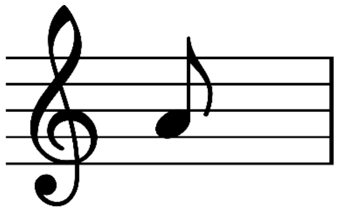
- Speech Recognition
- Music Generation
- Sentiment Classification
- DNA Sequence Analysis
- Machine Translation
- Video Activity Recognition
- Name Entity Recognition

Mary had a little lamb

Examples of Sequence Data

- Speech Recognition
- Music Generation
- Sentiment Classification
- DNA Sequence Analysis
- Machine Translation
- Video Activity Recognition
- Name Entity Recognition

La



Examples of Sequence Data

- Speech Recognition
- Music Generation
- Sentiment Classification

“Its an average movie”

- DNA Sequence Analysis
- Machine Translation
- Video Activity Recognition
- Name Entity Recognition

Examples of Sequence Data

- Speech Recognition
- Music Generation
- Sentiment Classification
- DNA Sequence Analysis   AGCCCCTGTGAGGAACTAG → AGCCCCTGTGAGGAACTAG
- Machine Translation
- Video Activity Recognition
- Name Entity Recognition

Examples of Sequence Data

- Speech Recognition
- Music Generation
- Sentiment Classification
- DNA Sequence Analysis
- Machine Translation

  ARE YOU FEELING SLEEPY ⟶ क्या आपको नींद आ रही है

- Video Activity Recognition
- Name Entity Recognition

Examples of Sequence Data

- Speech Recognition
- Music Generation
- Sentiment Classification
- DNA Sequence Analysis
- Machine Translation
- Video Activity Recognition
- Name Entity Recognition

WAVING

Examples of Sequence Data

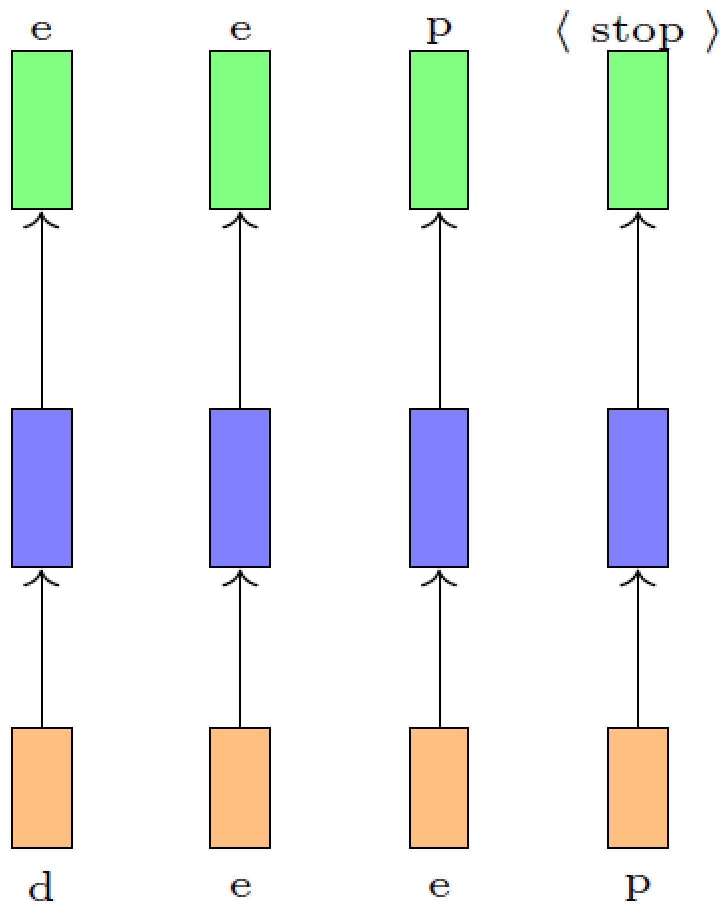
- Speech Recognition
- Music Generation
- Sentiment Classification
- DNA Sequence Analysis
- Machine Translation
- Video Activity Recognition
- Name Entity Recognition

"Alice wants to discuss about Deep Learning with Bob" → "Alice wants to discuss about Deep Learning with Bob"

Issues with using ANN/CNN on sequential data

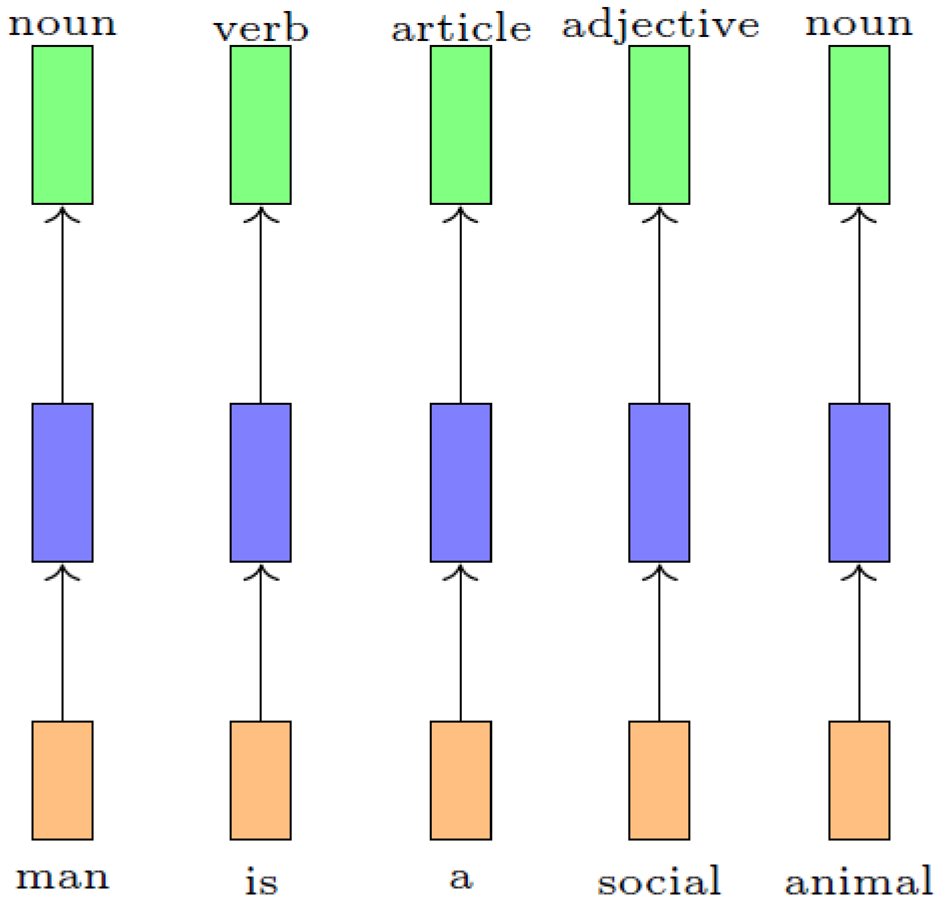
- In feedforward and convolutional neural networks the size of the input was always fixed.
- each input to the network was independent of the previous or future inputs.
- In many applications with sequence data, the input is not of a fixed size.
- Further successive inputs may not be independent of each other.

Examples of Sequence Learning Problems



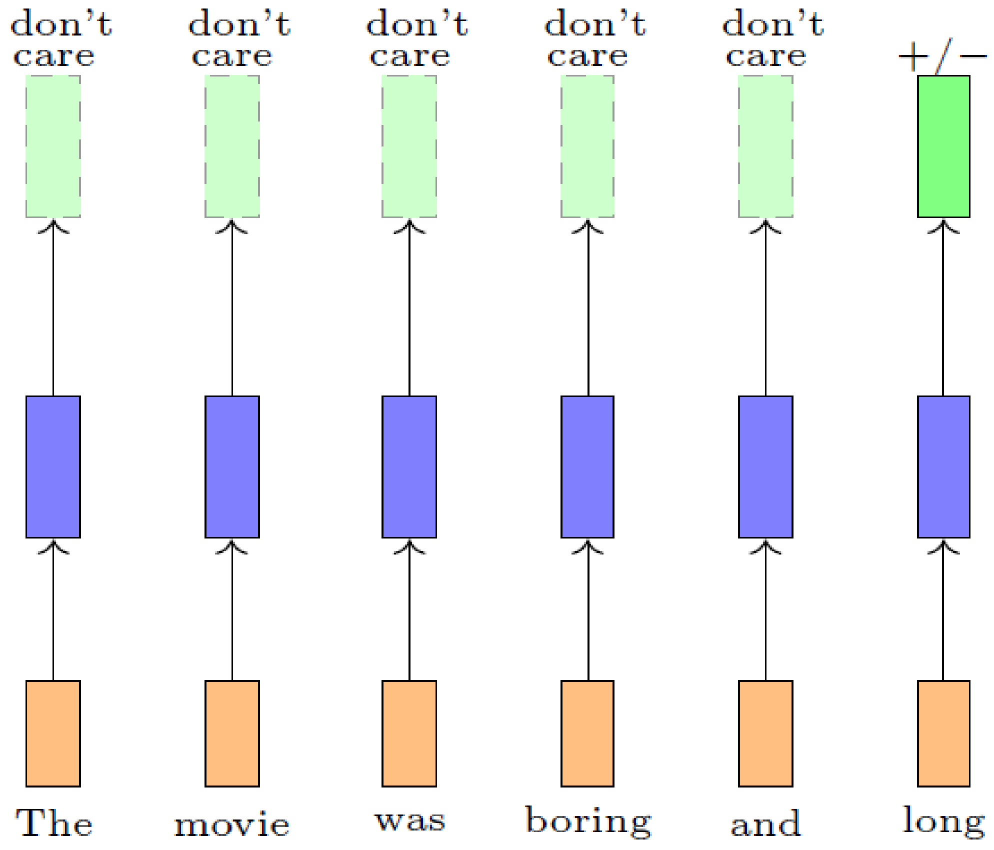
Task: Auto-complete

Examples of Sequence Learning Problems



Task: P-o-S tagging

Examples of Sequence Learning Problems

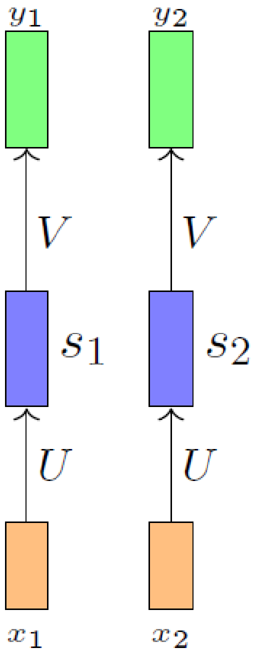


Task: Movie review

How to model such sequences?

- Account for dependence between inputs.
- Account for variable number of inputs.
- Make sure that the function executed at each time step is the same.

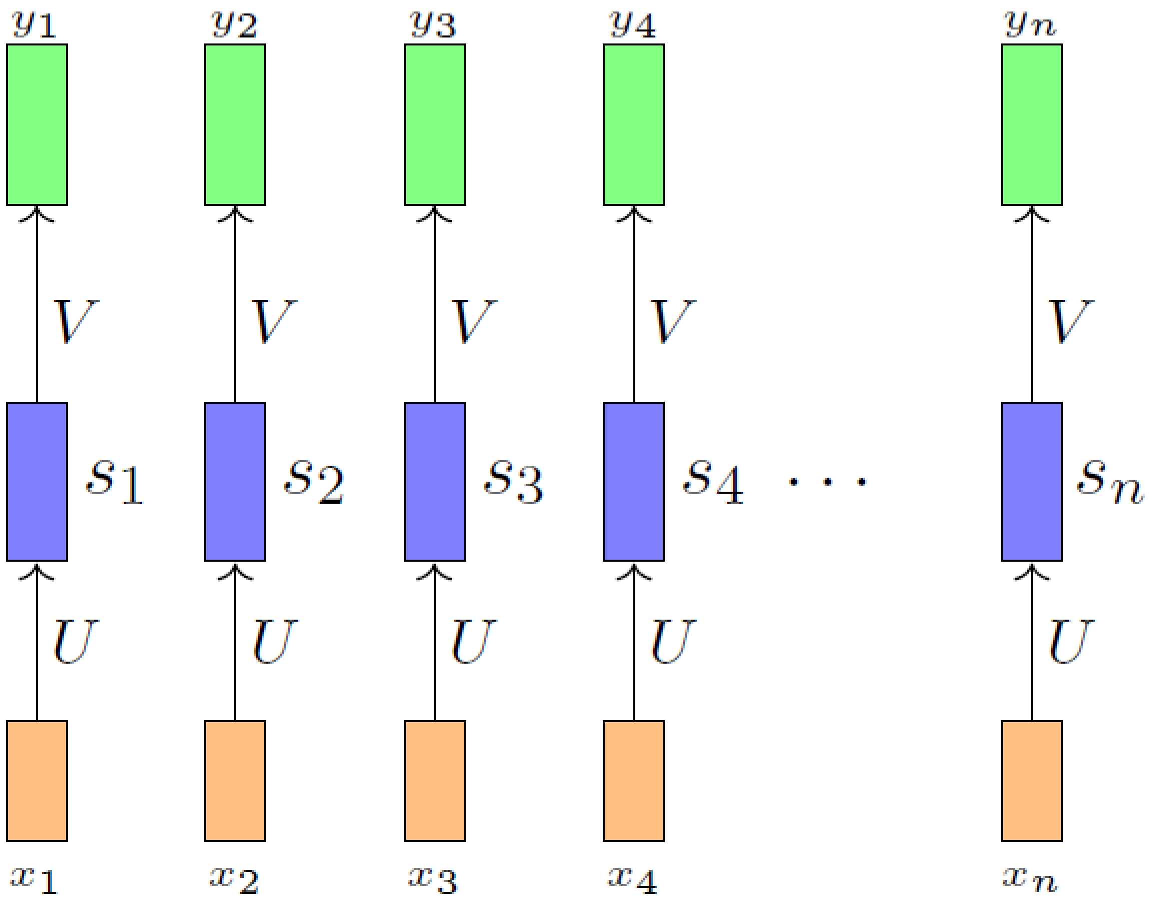
Recurrent Neural Networks (RNN)



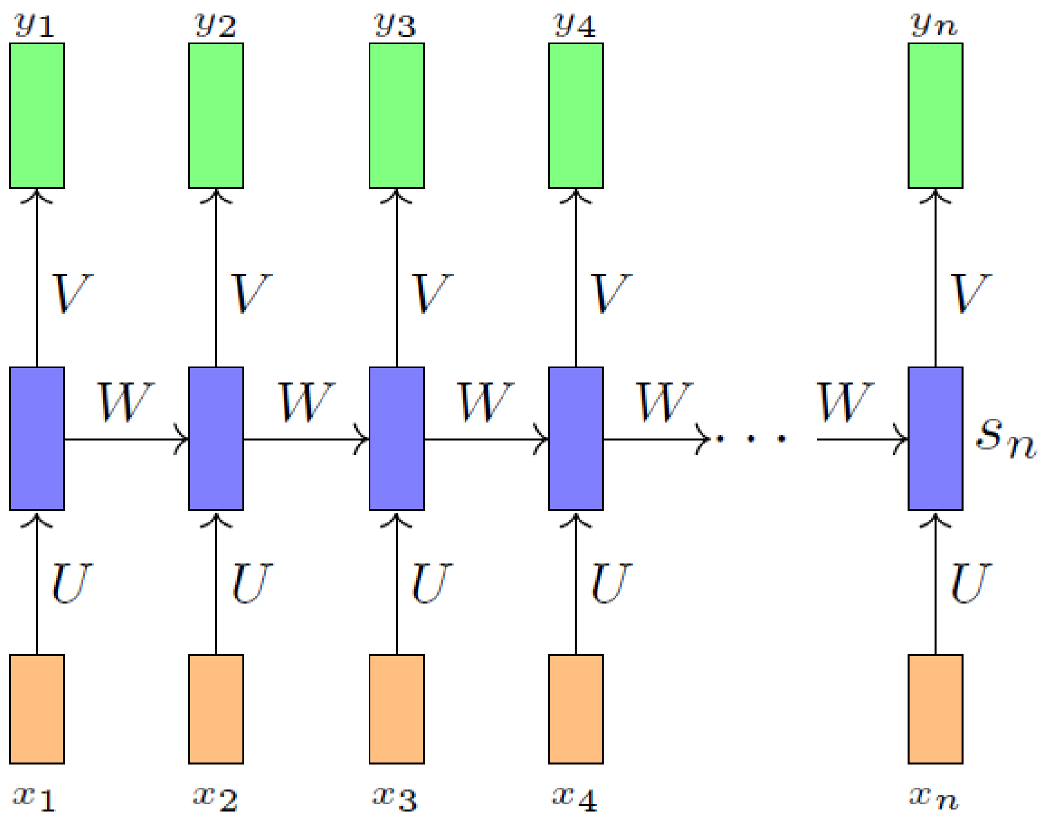
$$s_i = \sigma(Ux_i + b)$$
$$y_i = \mathcal{O}(Vs_i + c)$$
$$i = \text{timestep}$$

Since we want the same function to be executed at each timestep we should share the same network (i.e., same parameters at each timestep)

Recurrent Neural Networks (RNN)



Recurrent Neural Networks (RNN)



$$s_i = \sigma(Ux_i + Ws_{i-1} + b)$$

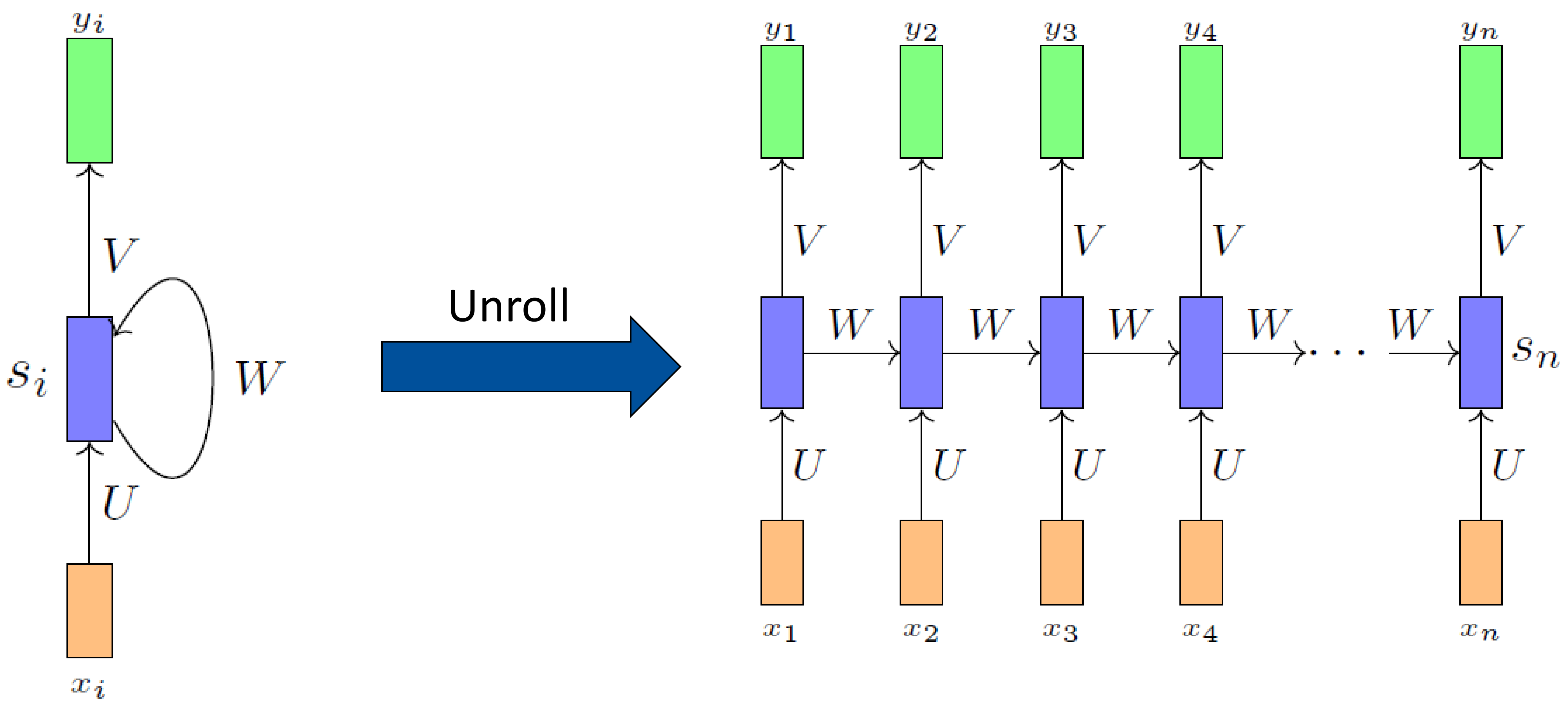
$$y_i = \mathcal{O}(Vs_i + c)$$

or

$$y_i = f(x_i, s_{i-1}, W, U, V, b, c)$$

Memory Cell-RNNs preserve some state across time steps

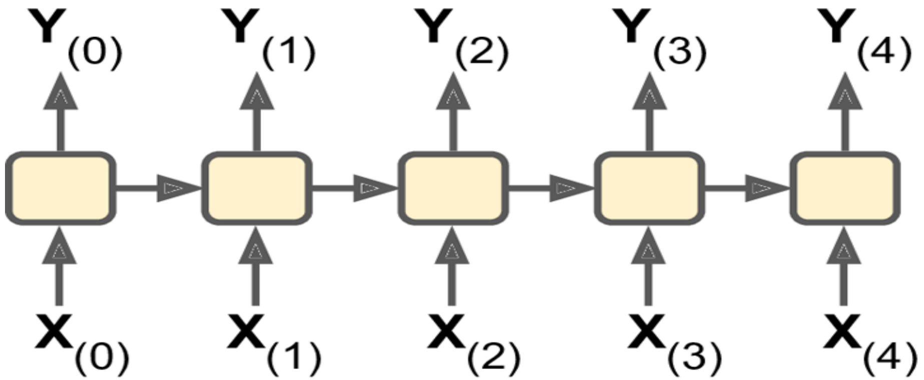
Recurrent Neural Networks (RNN)



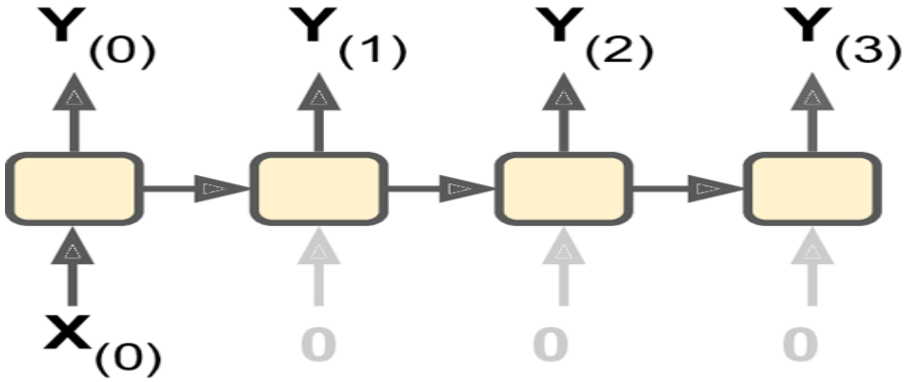
- Unrolling the network through time - representing network against time axis
- At each time step t (also called a frame) RNN receives inputs x_i as well as output from previous step y_{i-1}

Input and Output Sequences

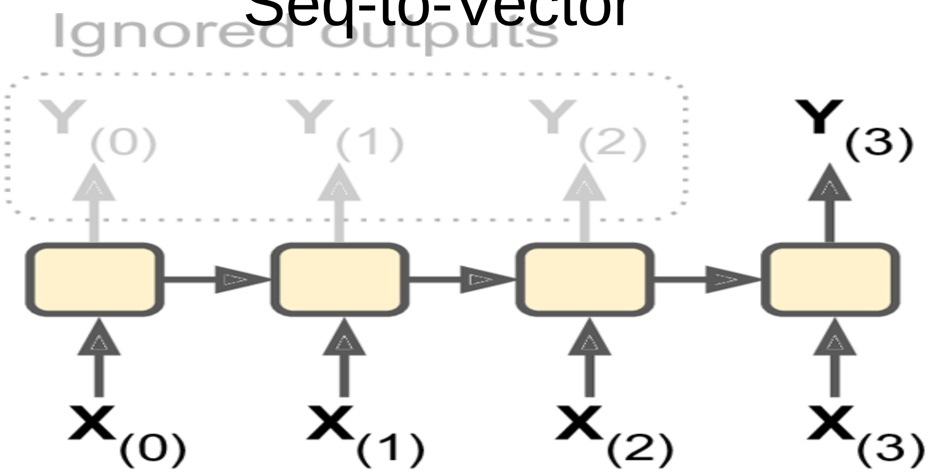
Seq-to-Seq

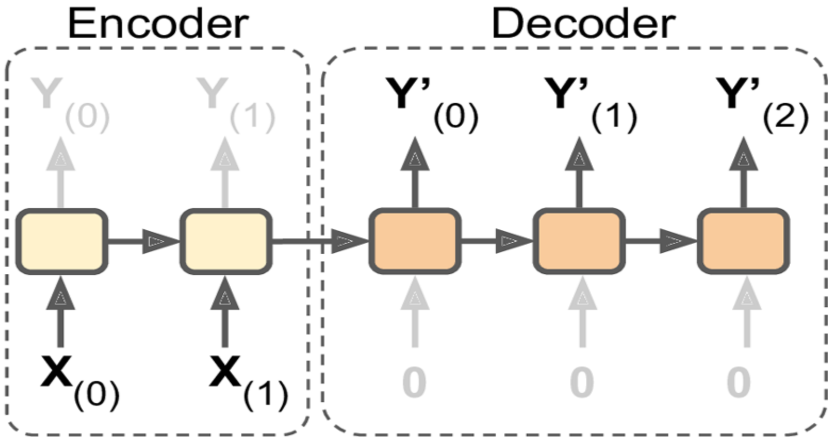


Vector-to-Seq

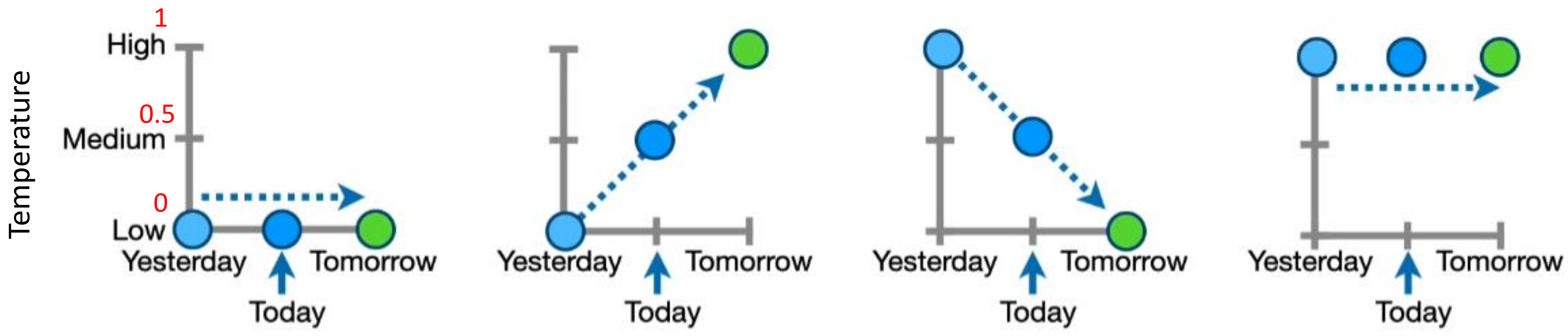


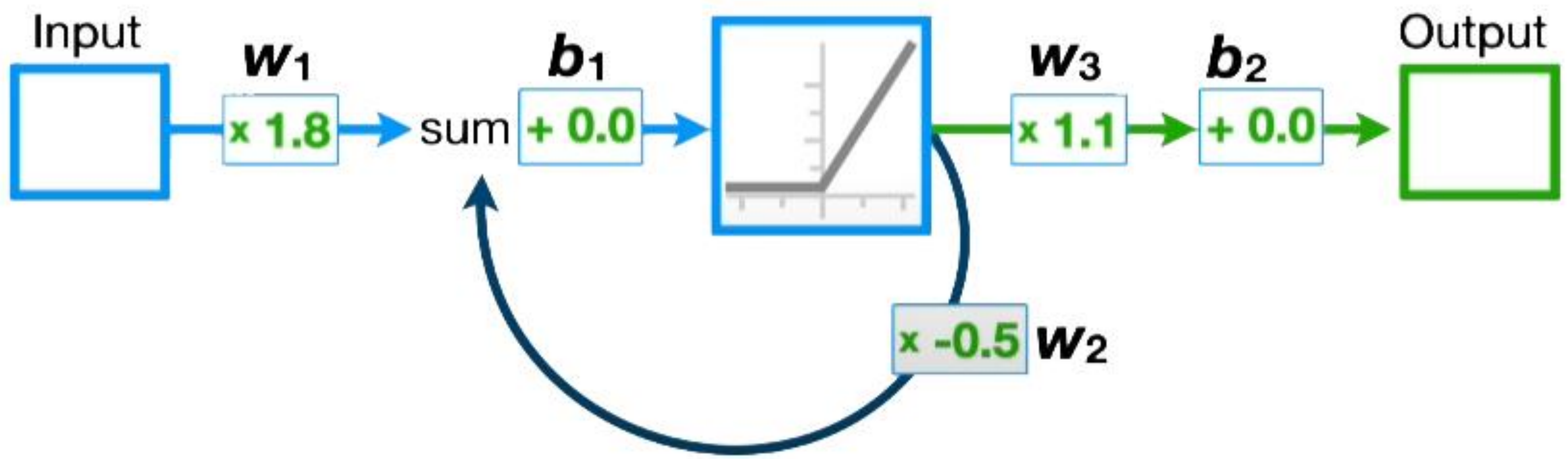
Seq-to-Vector





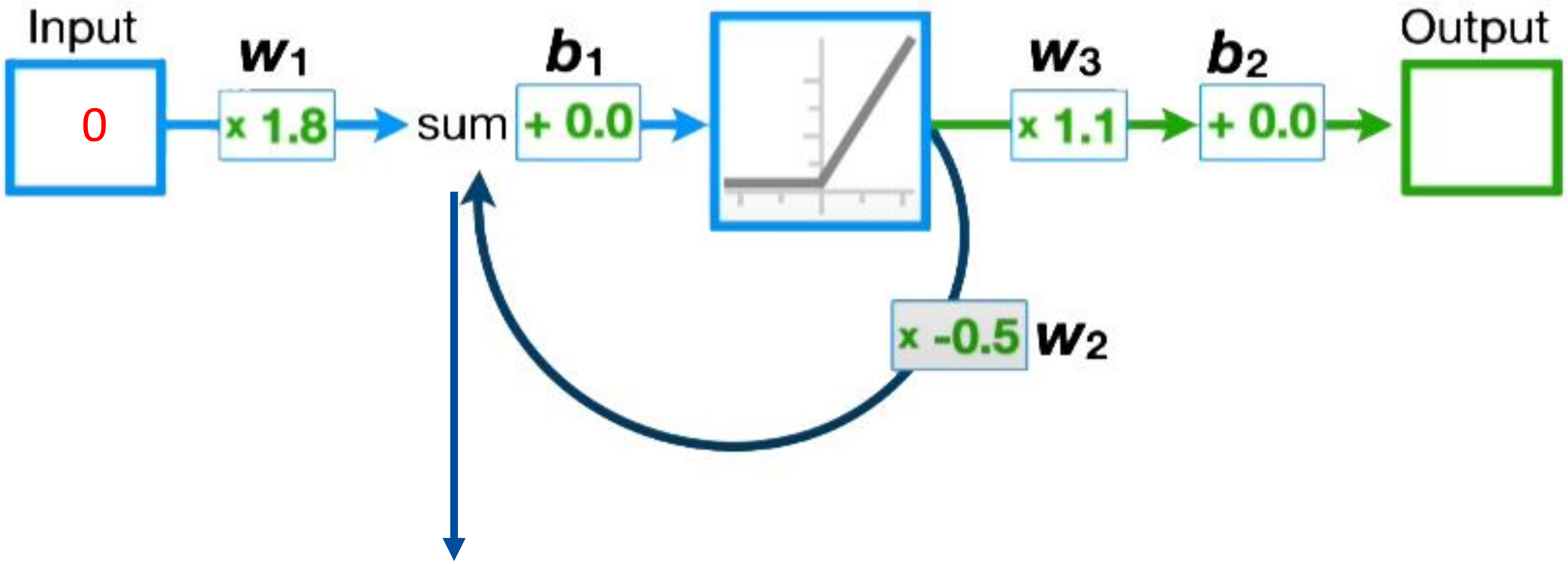
Recurrent Neural Networks (RNN) : Example





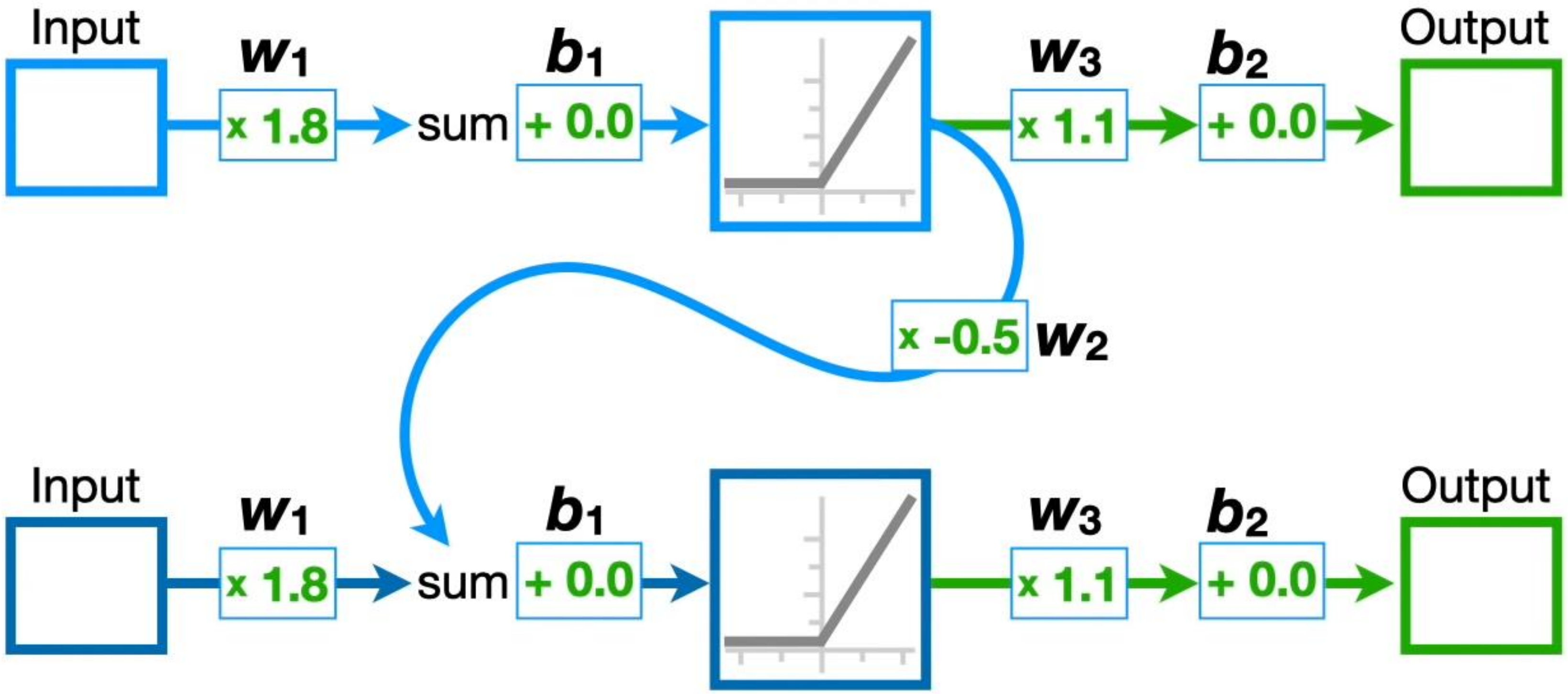
Source: https://www.youtube.com/c/joshstarmer

Recurrent Neural Networks (RNN) : Example



$$(\text{Today} \times w_1) + (y_1 \times w_2)$$

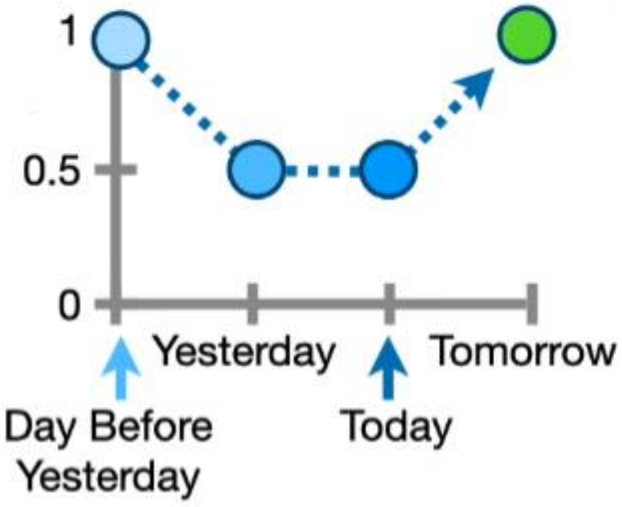
Source: https://www.youtube.com/c/joshstarmer

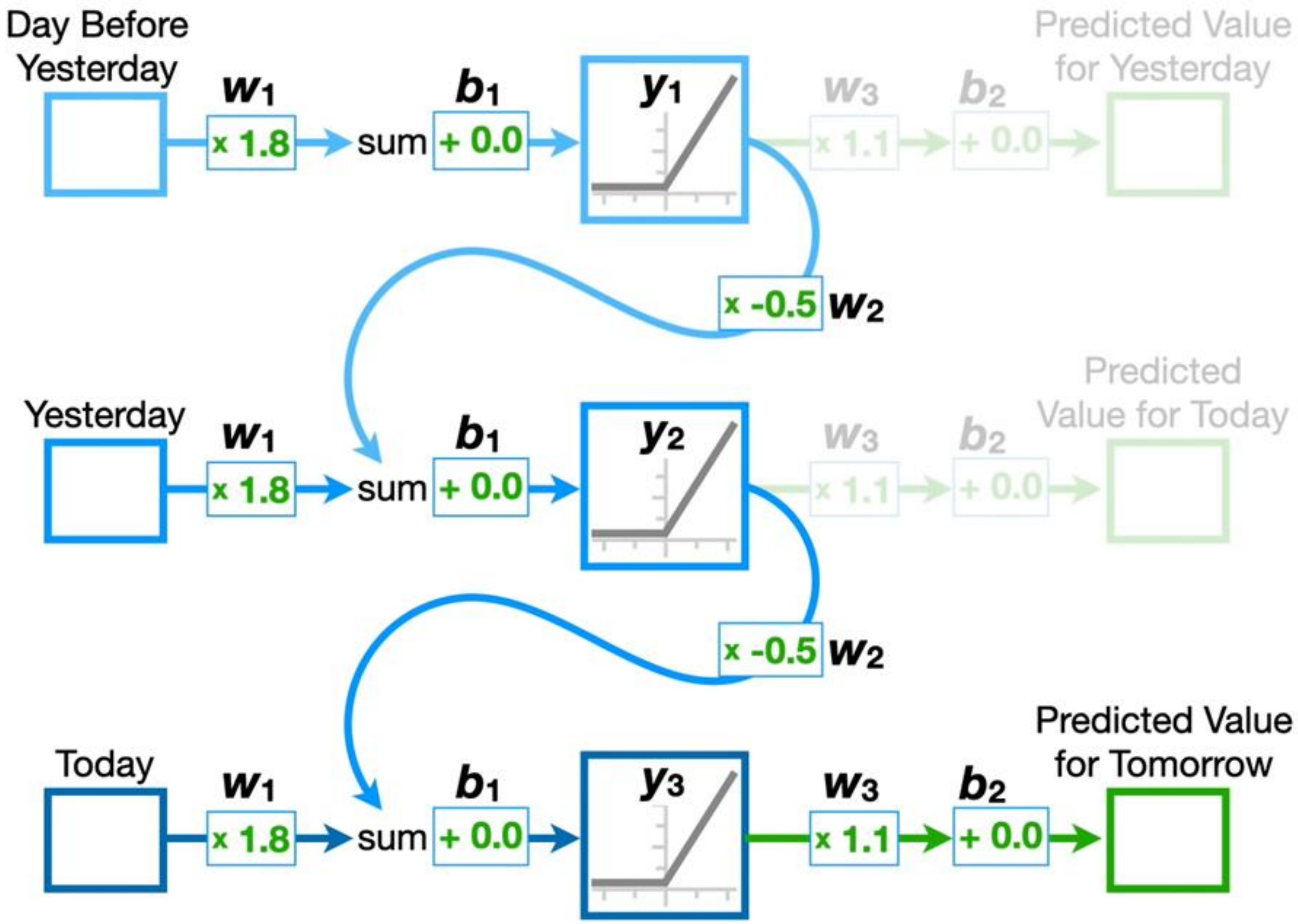


Unrolling the feedback loop by making a copy of NN for each input value

Source: https://www.youtube.com/c/joshstarmer

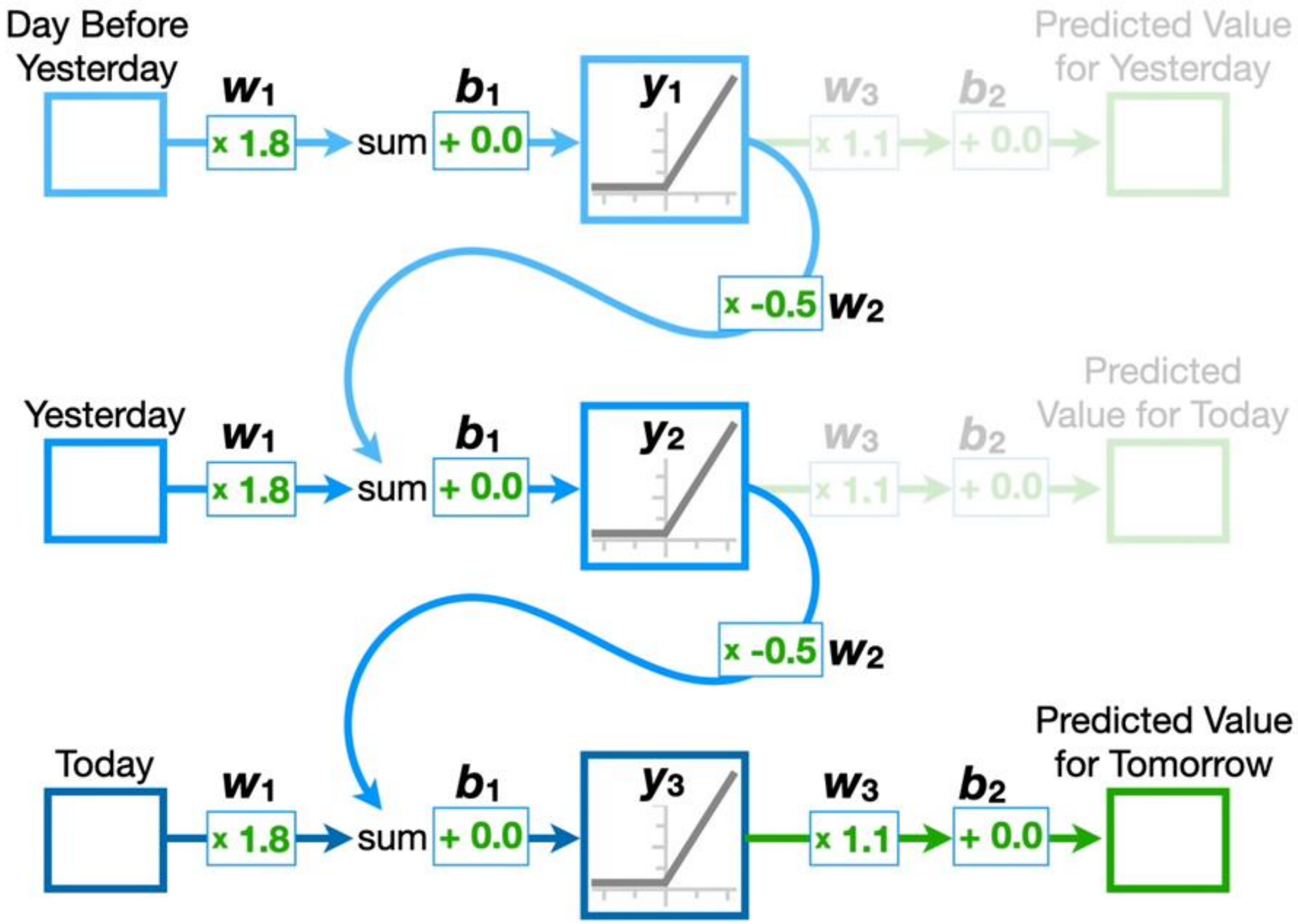
Recurrent Neural Networks (RNN) : Example





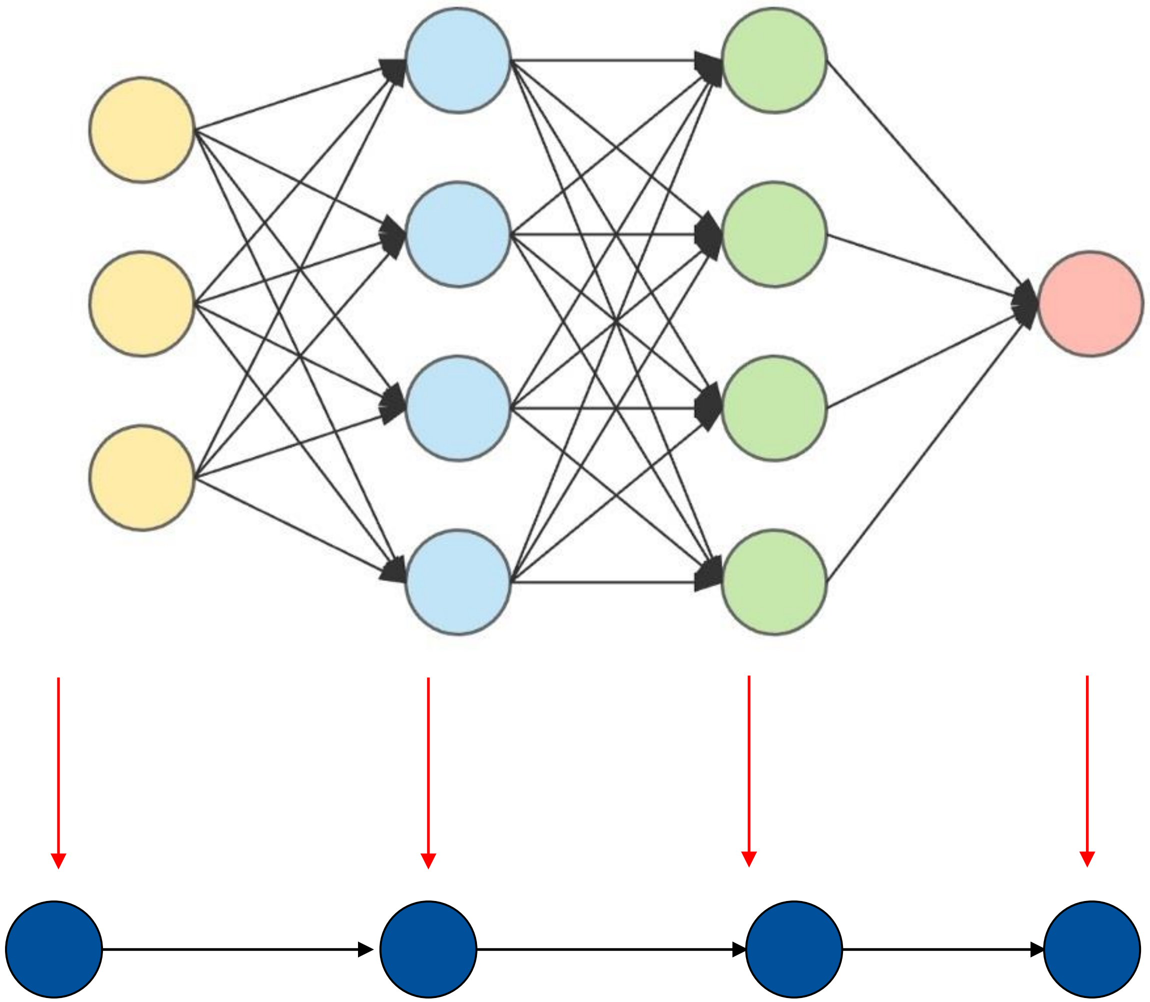
Source: https://www.youtube.com/c/joshstarmer

Recurrent Neural Networks (RNN) : : Example



Source: https://www.youtube.com/c/joshstarmer

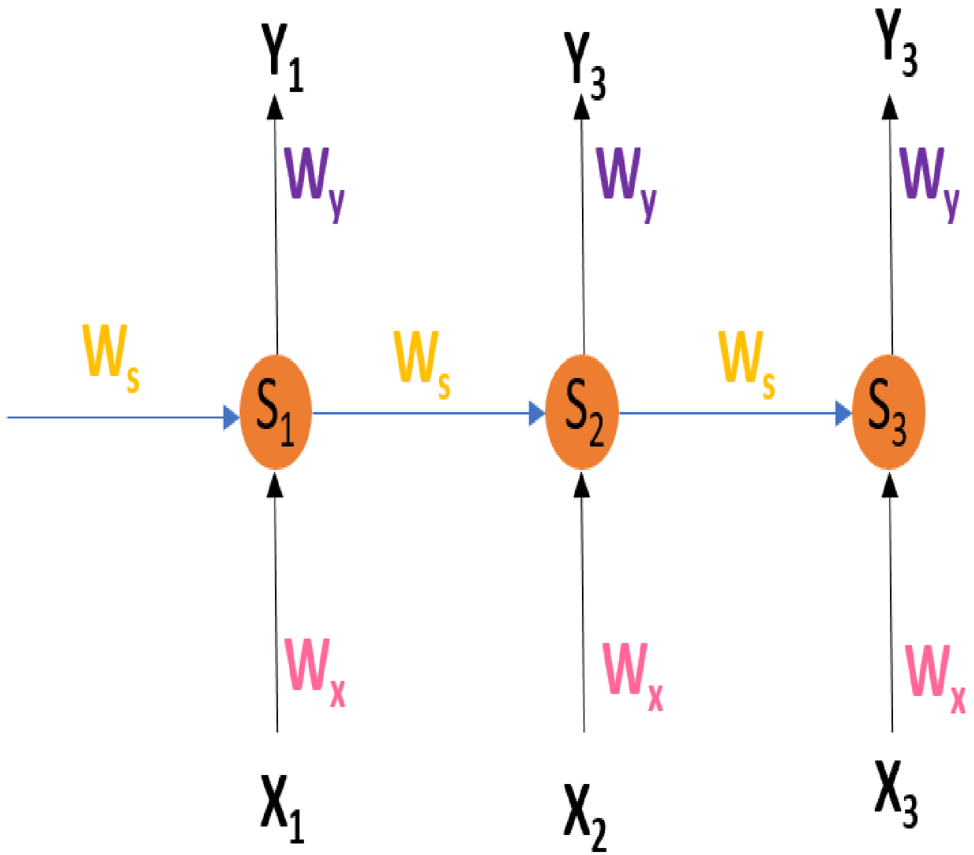
Backpropagation in ANN : Recap



Training RNNS

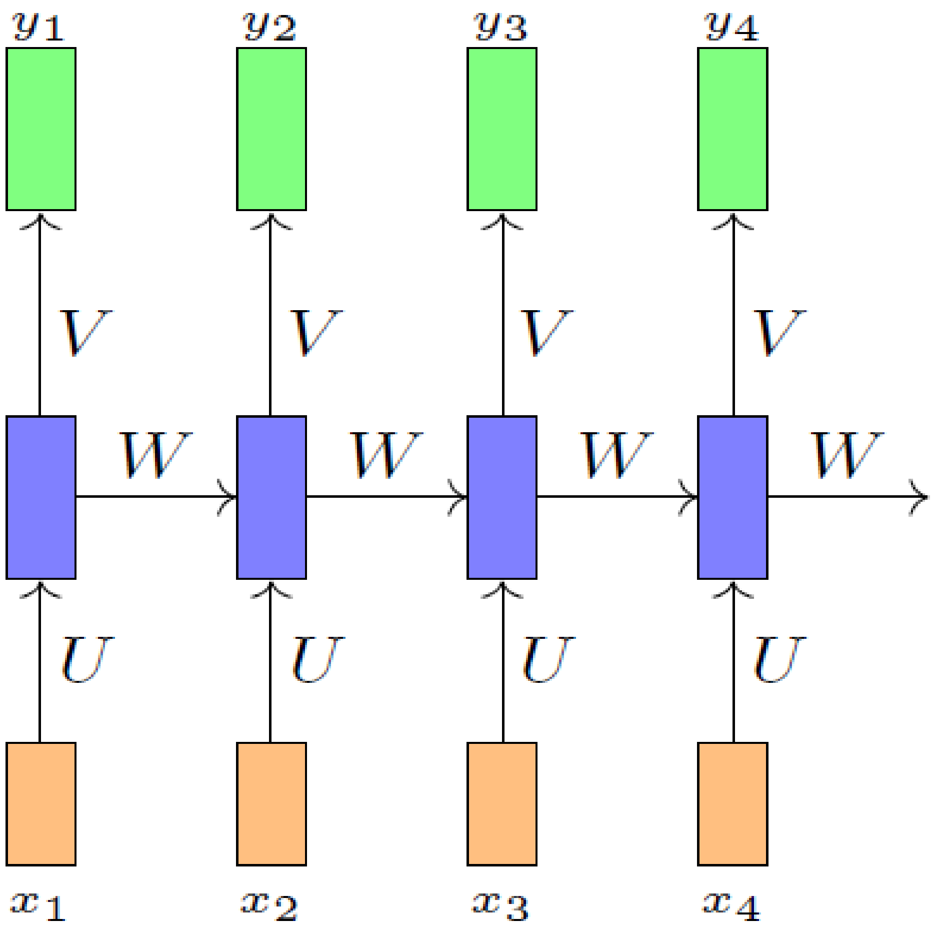
- Backpropagation through time (BPTT)
- Step 1 – Forward pass through the network
- Step 2 – Output Sequence is evaluated using a cost function $C(Y_{(0)},Y_{(1)},\ldots..Y_{(T)})$ where T is max time step
- Step 3 –
  - Gradient of cost function is then propagated backward.
  - Gradients flow backward through all the outputs used by the cost function
- Step 4- Model parameters are updated using the gradients computed during BPTT

Back Propagation through time



- *X1, X2, X3* are the inputs at time *t1, t2, t3*
- *Wx* is the weight matrix associated with it.
- *S1, S2, S3* are the hidden states or memory units at time *t1, t2, t3*
- *Ws* is the weight matrix associated with it.
  *Y1, Y2, Y*3 are the outputs at time *t1, t2, t3* respectively
- *Wy* is the weight matrix associated with it.

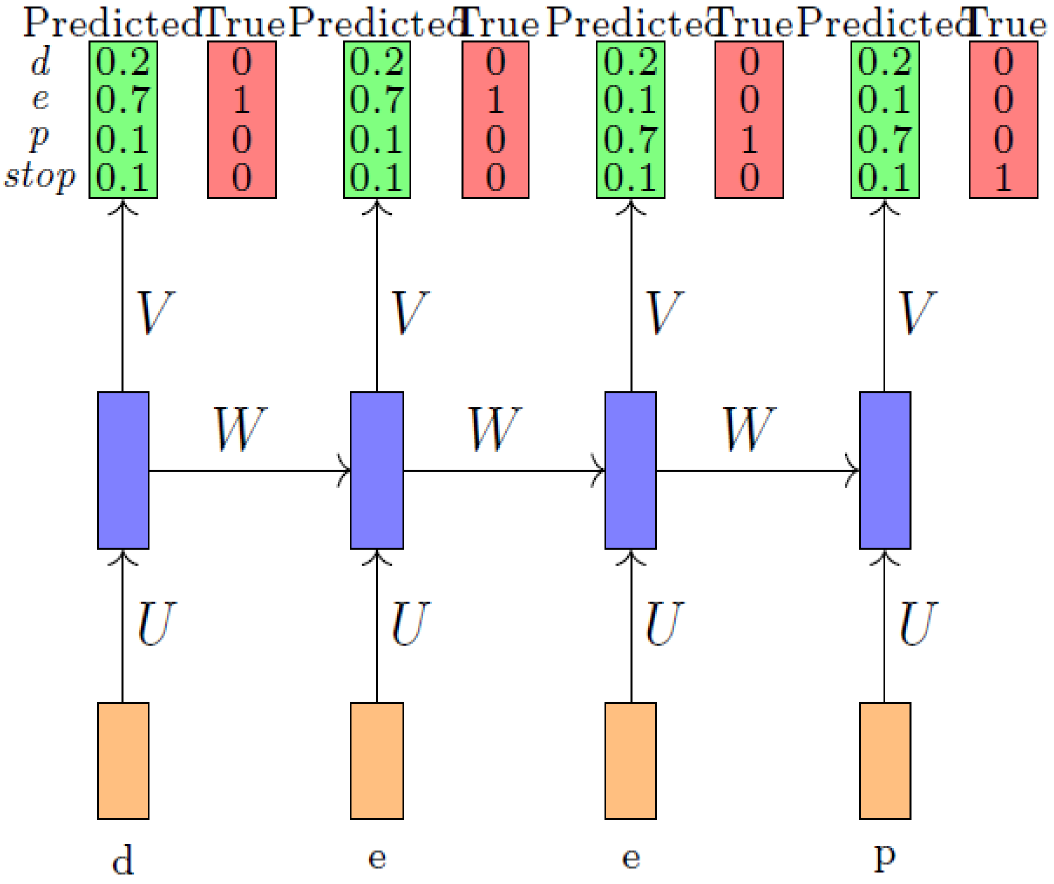
Back Propagation Through Time



dimensions of the parameters carefully

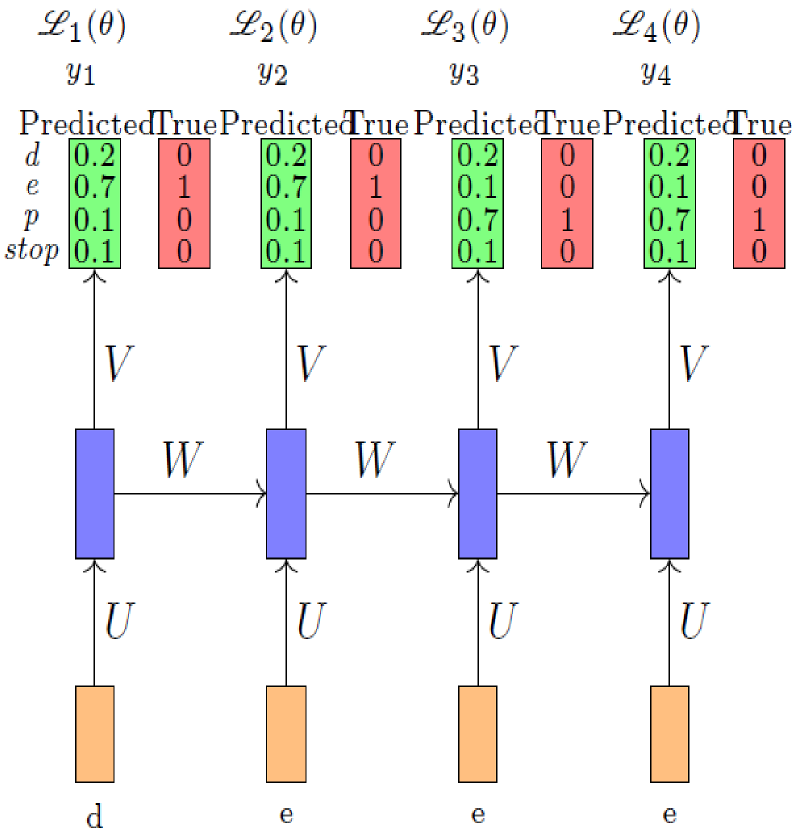
$$x_i \in \mathbb{R}^n \quad \text{(n-dimensional input)}$$
$$s_i \in \mathbb{R}^d \quad \text{(d-dimensional state)}$$
$$y_i \in \mathbb{R}^k \quad \text{(say k classes)}$$
$$U \in \mathbb{R}^{n \times d}$$
$$V \in \mathbb{R}^{d \times k}$$
$$W \in \mathbb{R}^{d \times d}$$

Back Propagation Through Time



- Suppose we initialize U, V, W randomly and the network predicts the probabilities as shown
- And the true probabilities are as shown
- We need to answer two questions
- What is the total loss made by the model ?
- How do we backpropagate this loss and update the parameters ($\theta = \{U, V, W, b, c\}$) of the network ?

Back Propagation Through Time



- The total loss is simply the sum of the loss over all time-steps

$$\mathscr{L}(\theta) = \sum_{t=1}^{T} \mathscr{L}_t(\theta)$$

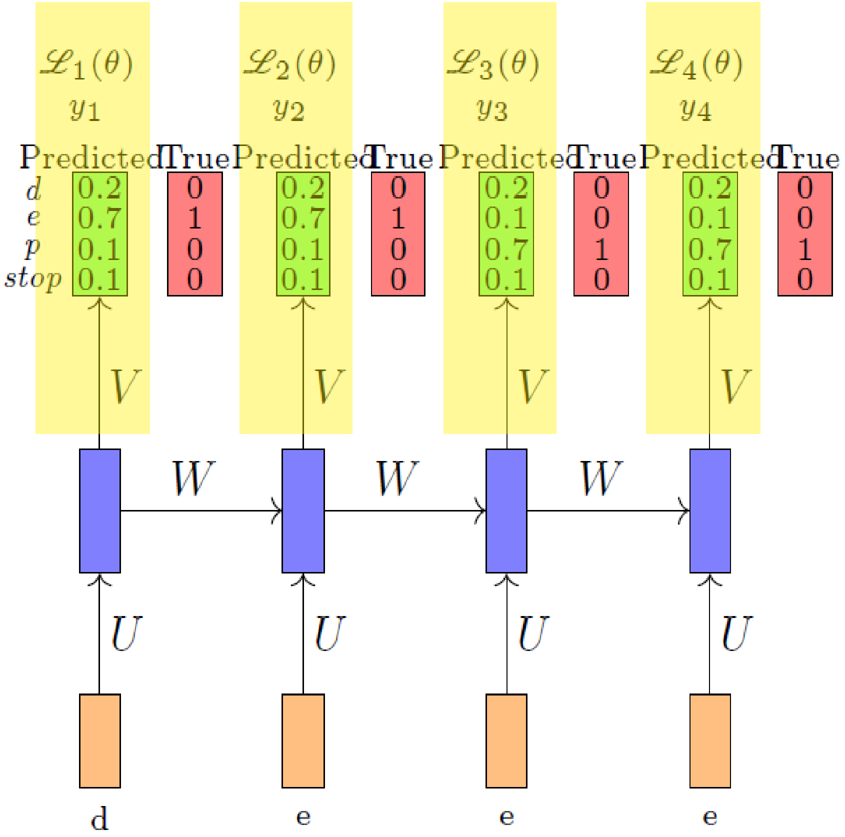
$$\mathscr{L}_t(\theta) = -log(y_{tc})$$

$$y_{tc} = \text{predicted probability of true character at time-step } t$$

$$T = \text{number of timesteps}$$

- For backpropagation we need to compute the gradients w.r.t. W, U, V, b, c

Back Propagation Through Time

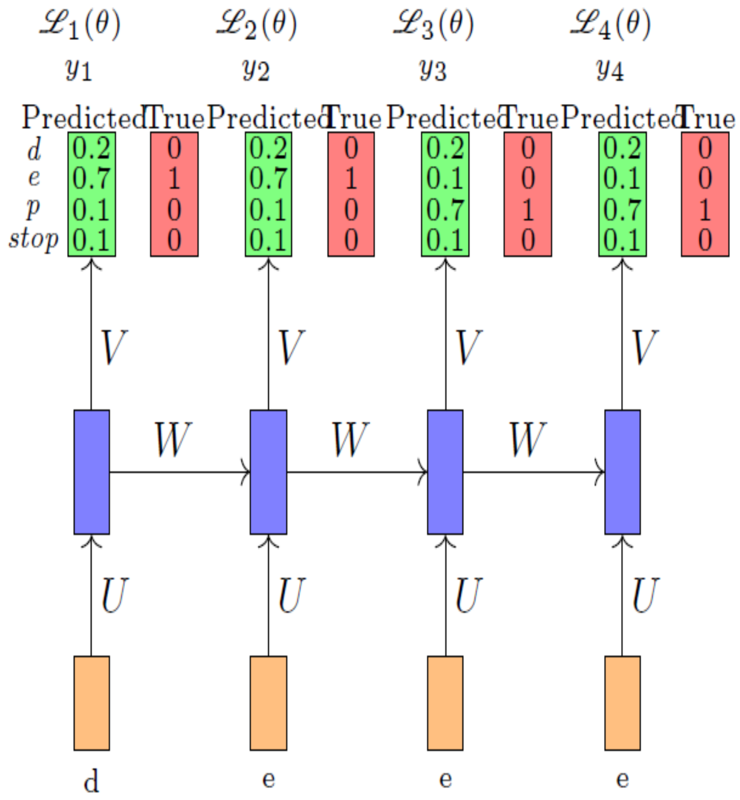


- Let us consider $\frac{\partial \mathscr{L}(\theta)}{\partial V}$ (V is a matrix so ideally we should write $\nabla_v \mathscr{L}(\theta)$)

$$\frac{\partial \mathscr{L}(\theta)}{\partial V} = \sum_{t=1}^{T} \frac{\partial \mathscr{L}_t(\theta)}{\partial V}$$

- Each term is the summation is simply the derivative of the loss w.r.t. the weights in the output layer
- We have already seen how to do this when we studied backpropagation

Back Propagation Through Time

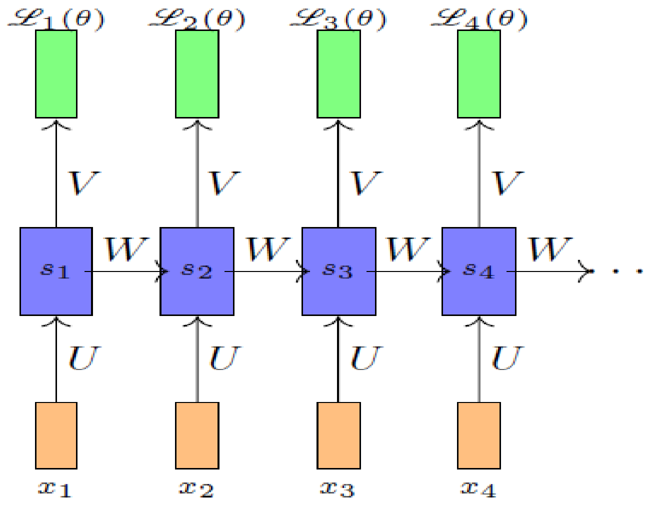


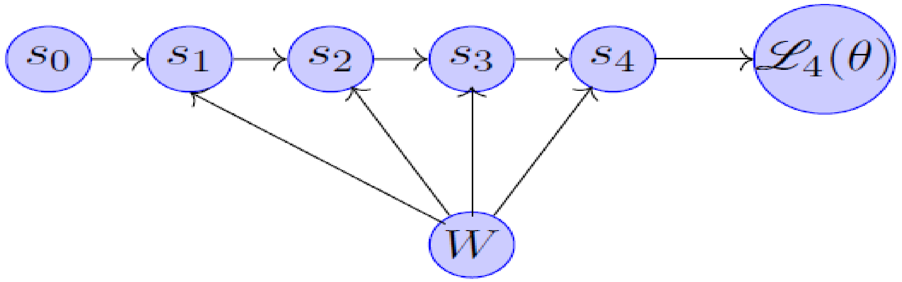
- Let us consider the derivative $\frac{\partial \mathscr{L}(\theta)}{\partial W}$

$$\frac{\partial \mathscr{L}(\theta)}{\partial W} = \sum_{t=1}^{T} \frac{\partial \mathscr{L}_t(\theta)}{\partial W}$$

- By the chain rule of derivatives we know that $\frac{\partial \mathscr{L}_t(\theta)}{\partial W}$ is obtained by summing gradients along all the paths from $\mathscr{L}_t(\theta)$ to W
- What are the paths connecting $\mathscr{L}_t(\theta)$ to W ?
- Let us see this by considering $\mathscr{L}_4(\theta)$

Back Propagation Through Time





- $\mathscr{L}_4(\theta)$ depends on s_4
- s_4 in turn depends on s_3 and W
- s_3 in turn depends on s_2 and W
- s_2 in turn depends on s_1 and W
- s_1 in turn depends on s_0 and W

  where s_0 is a constant starting state.

Back Propagation Through Time

$$\frac{\partial s_4}{\partial W} = \underbrace{\frac{\partial^+ s_4}{\partial W}}_{\text{explicit}} + \underbrace{\frac{\partial s_4}{\partial s_3}\frac{\partial s_3}{\partial W}}_{\text{implicit}}$$

$$= \frac{\partial^+ s_4}{\partial W} + \frac{\partial s_4}{\partial s_3}\left[\underbrace{\frac{\partial^+ s_3}{\partial W}}_{\text{explicit}} + \underbrace{\frac{\partial s_3}{\partial s_2}\frac{\partial s_2}{\partial W}}_{\text{implicit}}\right]$$

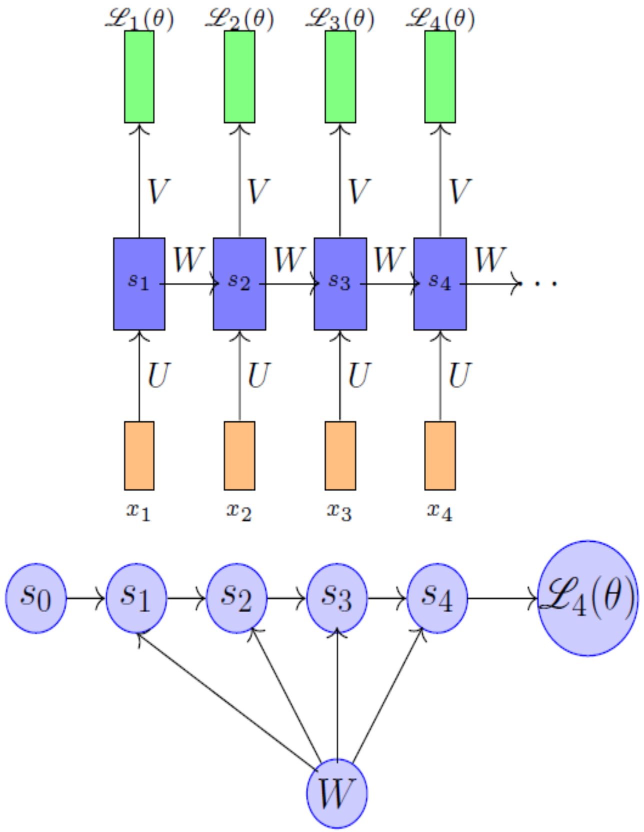
$$= \frac{\partial^+ s_4}{\partial W} + \frac{\partial s_4}{\partial s_3}\frac{\partial^+ s_3}{\partial W} + \frac{\partial s_4}{\partial s_3}\frac{\partial s_3}{\partial s_2}\left[\frac{\partial^+ s_2}{\partial W} + \frac{\partial s_2}{\partial s_1}\frac{\partial s_1}{\partial W}\right]$$

$$= \frac{\partial^+ s_4}{\partial W} + \frac{\partial s_4}{\partial s_3}\frac{\partial^+ s_3}{\partial W} + \frac{\partial s_4}{\partial s_3}\frac{\partial s_3}{\partial s_2}\frac{\partial^+ s_2}{\partial W} + \frac{\partial s_4}{\partial s_3}\frac{\partial s_3}{\partial s_2}\frac{\partial s_2}{\partial s_1}\left[\frac{\partial^+ s_1}{\partial W}\right]$$

For simplicity we will short-circuit some of the paths

$$\frac{\partial s_4}{\partial W} = \frac{\partial s_4}{\partial s_4}\frac{\partial^+ s_4}{\partial W} + \frac{\partial s_4}{\partial s_3}\frac{\partial^+ s_3}{\partial W} + \frac{\partial s_4}{\partial s_2}\frac{\partial^+ s_2}{\partial W} + \frac{\partial s_4}{\partial s_1}\frac{\partial^+ s_1}{\partial W} = \sum_{k=1}^{4}\frac{\partial s_4}{\partial s_k}\frac{\partial^+ s_k}{\partial W}$$

Back Propagation Through Time



- Finally we have

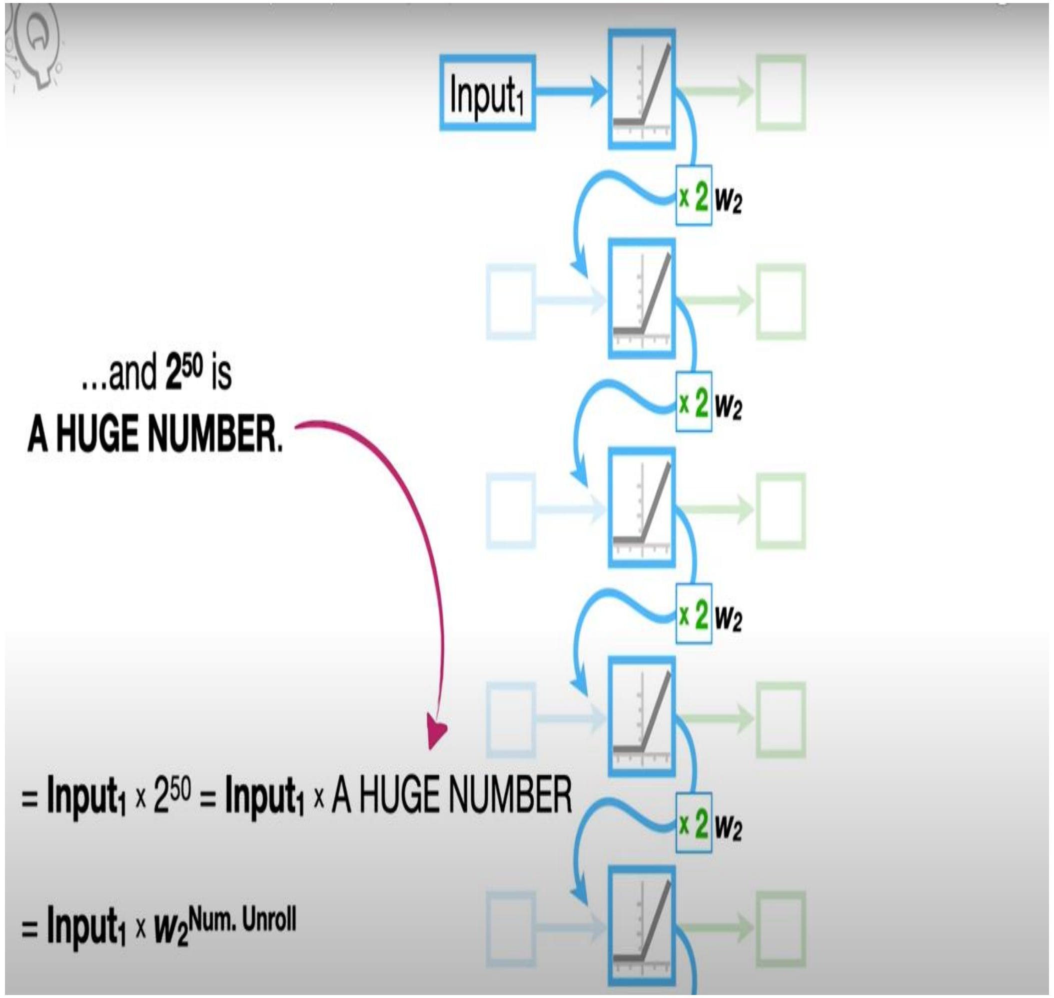
$$\frac{\partial \mathscr{L}_4(\theta)}{\partial W} = \frac{\partial \mathscr{L}_4(\theta)}{\partial s_4}\frac{\partial s_4}{\partial W}$$

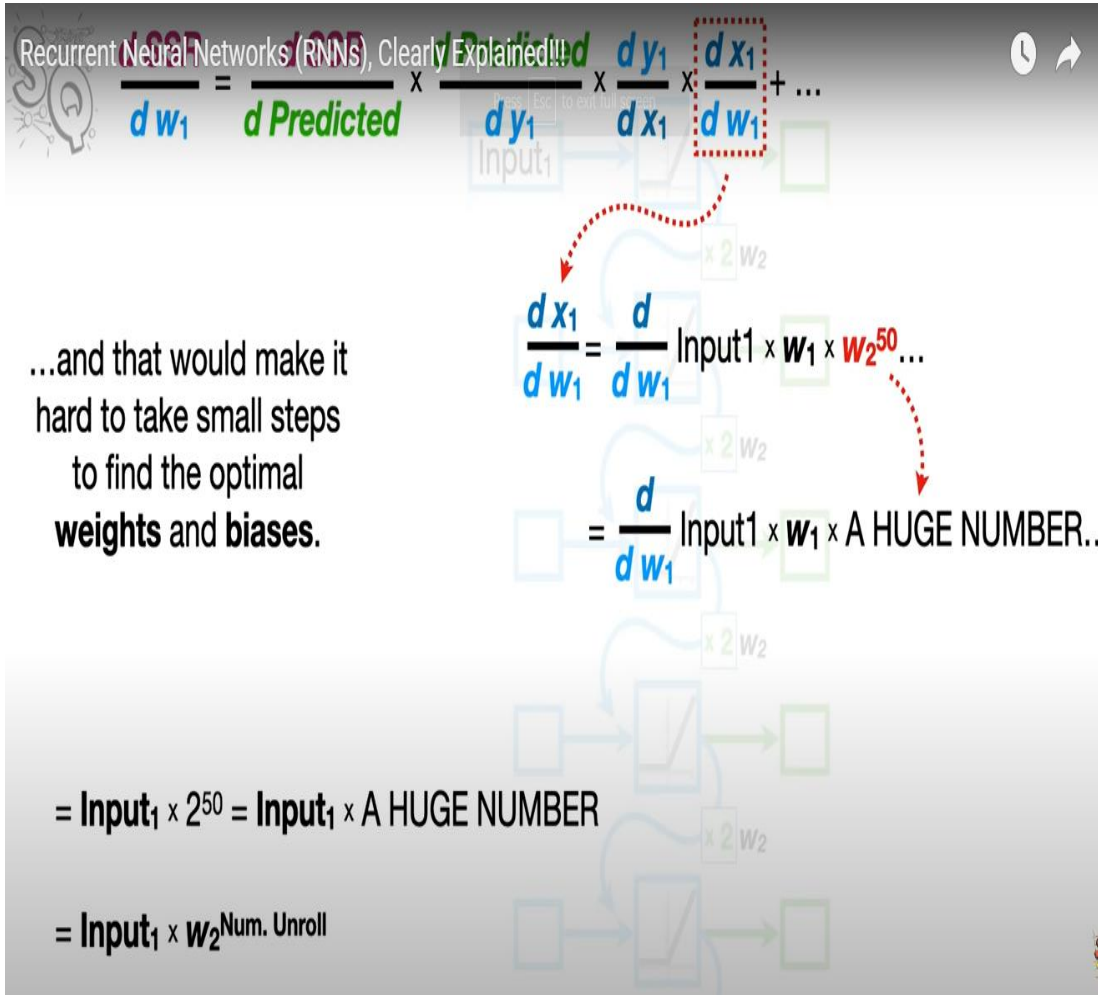
$$\frac{\partial s_4}{\partial W} = \sum_{k=1}^{4} \frac{\partial s_4}{\partial s_k}\frac{\partial^+ s_k}{\partial W}$$

$$\therefore \frac{\partial \mathscr{L}_t(\theta)}{\partial W} = \frac{\partial \mathscr{L}_t(\theta)}{\partial s_t}\sum_{k=1}^{t} \frac{\partial s_t}{\partial s_k}\frac{\partial^+ s_k}{\partial W}$$

- This algorithm is called backpropagation through time (BPTT) as we backpropagate over all previous time steps

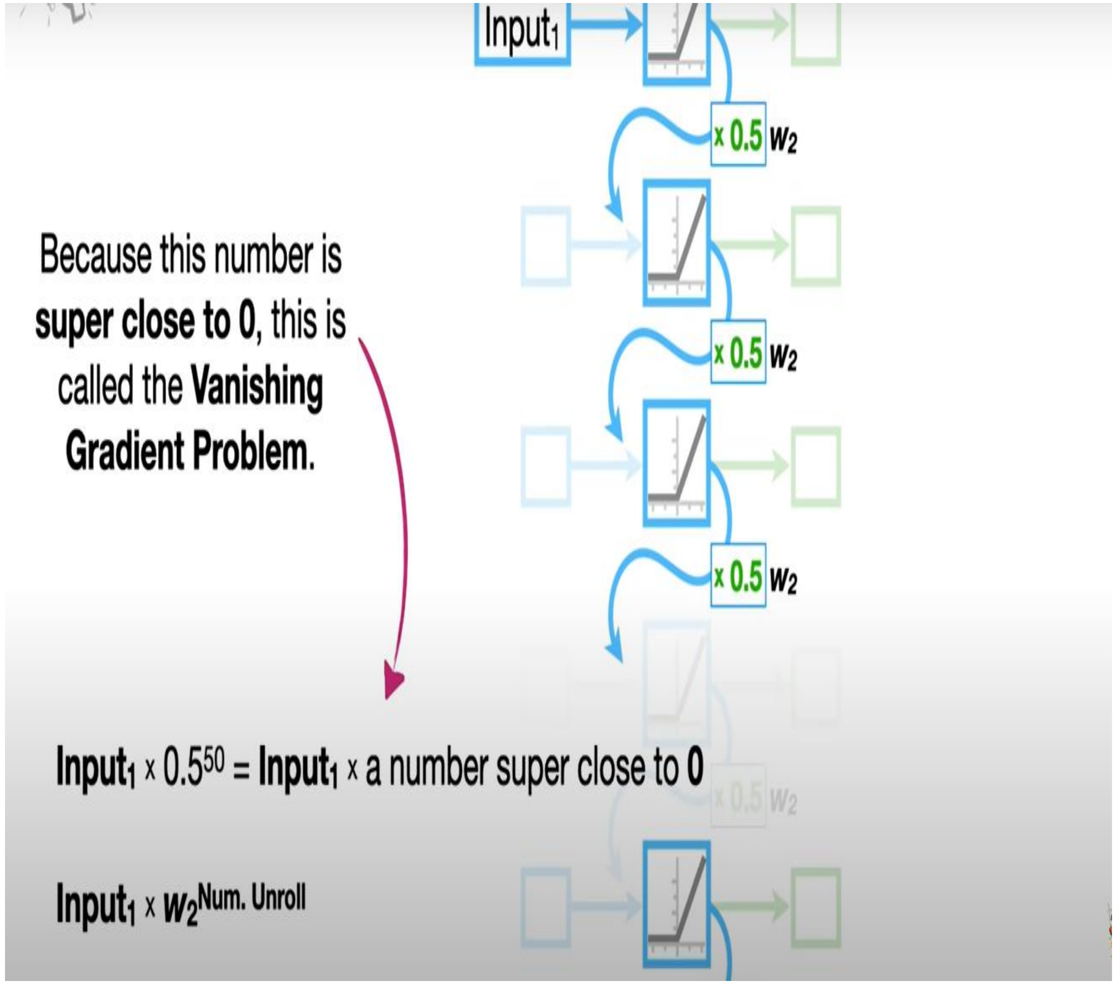
Back Propagation – Exploding gradient





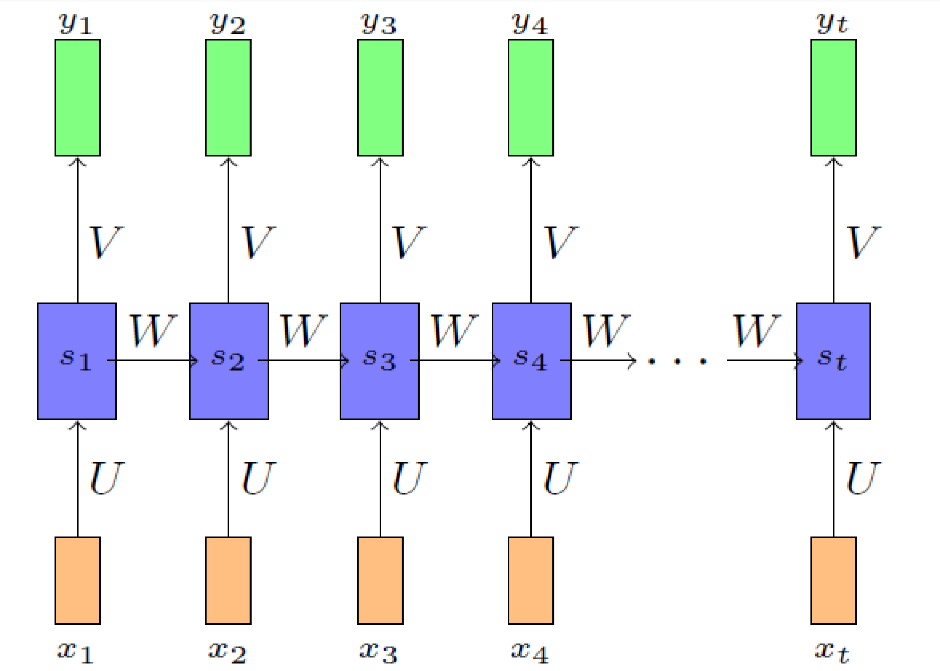
Solution - Can specify a cap on the upper limit of gradient called ***Gradient Clipping***

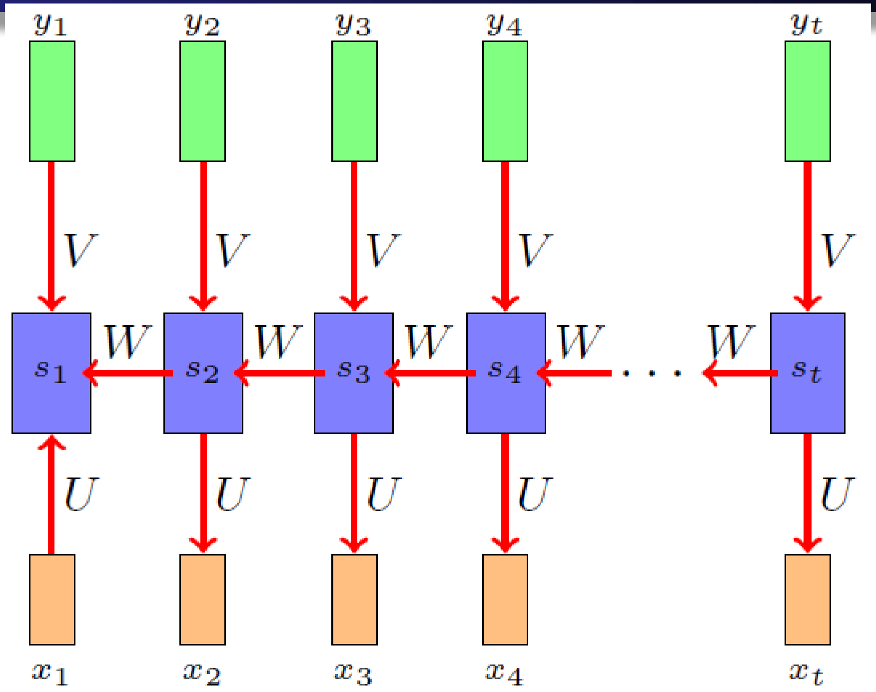
Back Propagation- Vanishing Gradients



- BPTT can be used up to a limited number of time steps like 8 or 10.
- If we back propagate further, the gradient becomes too small.
- The problem is that the contribution of information decays geometrically over time.
- if the number of time steps is >10 ,that information will effectively be discarded.
- Solution:
- Can use advanced RNN techniques like ***LSTM or GRU***

Pros and Cons for RNN



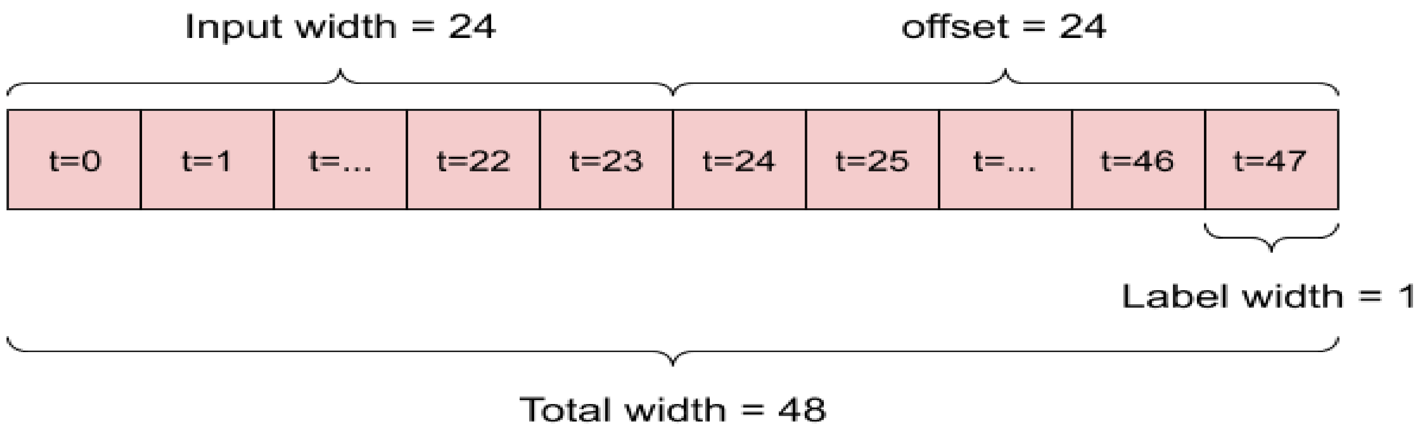


- The state (s_i) of an RNN records information from all previous time steps.
- At each new timestep the old information gets morphed by the current input.
- After ‘t’ steps the information stored at time step t-k (for some k < t) gets completely morphed.
- It would be impossible to extract the original information stored at time step t – k.

Also, during back propagation there is the problem of Vanishing or Exploding gradients

Applications – Forecasting Time Series

- **Time series-** Data is a sequence of one or more values per time step
- Can be univariate or multivariate
- **Strategies to Split the data – Split across time**
  - (70%, 20%, 10%) split for the training, validation, and test sets. Data is not being randomly shuffled before splitting. This is for two reasons:
    - It ensures that chopping the data into windows of consecutive samples is still possible.
    - It ensures that the validation/test results are more realistic, being evaluated on the data collected after the model was trained
- **The main features of the input windows are:**
  - The width (number of time steps) of the input and label windows.
  - The time offset between them.
  - Which features are used as inputs, labels, or both.

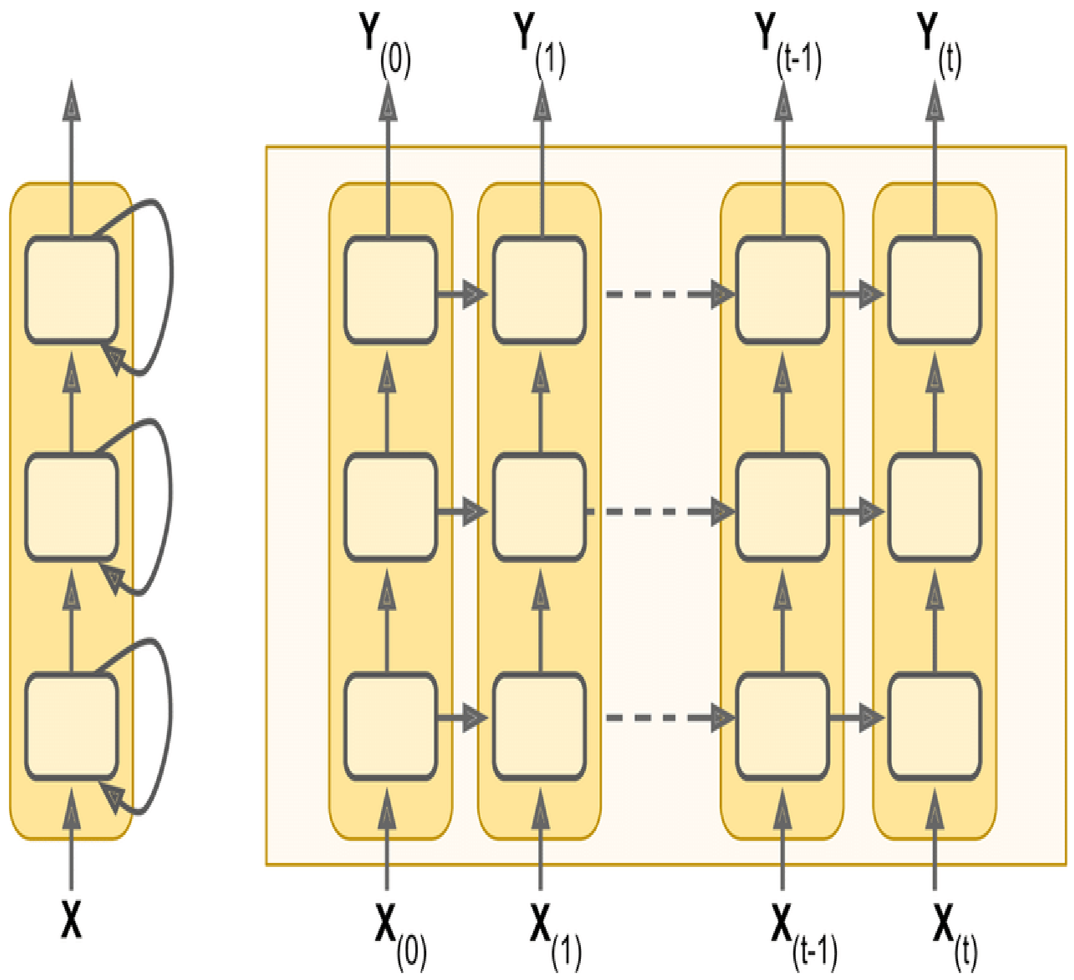


Applications- Forecasting Time Series

```
Model = keras.models.Sequential([
    Keras.layers.SimpleRNN(20,return_sequences=True,
    input_shape[None,1]),
    Keras.layers.SimpleRNN(20),
    Keras.layers.Dense(1)
    ])
```

- **Forecasting Several Time Steps ahead**

```
Model = keras.models.Sequential([
    Keras.layers.SimpleRNN(20,return_sequences=True,
    input_shape[None,1]),
    Keras.layers.SimpleRNN(20, return_sequences=True),
    Keras.layers.TimeDistributed(keras.layers.Dense(10))
    ])
     # wrapper applies a layer to every temporal slice of an input
```



Natural Language Processing with RNNS and Attention

- **Turing test(1950) objective:**
  - to evaluate a machine's ability to match human intelligence
  - Devised chatbot capable of fooling interlocutor into thinking it is human
- Mastering language is Homo Sapien's greatest cognitive ability
- Can we build a machine that can read and write natural language?
- Common approach is RNN
  - Character RNN – trained to predict the next character in a sentence
  - Stateless RNN – learns on random portions of text in each iteration , without any information on the rest of the text
  - Stateful RNN- preserves the hidden state between training iterations and continues reading where it left off, thereby learning longer patterns

Generating Shakespearan Text using a Character RNN

- "The unreasonable Effectiveness of Recurrent Neural Networks" – Andrej Karpathy (2015)
- 3-layer RNN with 512 hidden nodes on each layer
- Char-RNN was trained on Shakpeare's work used to generate novel text- one character at a time
- Model able to learn words, grammar, proper punctuation
- PANDARUS:

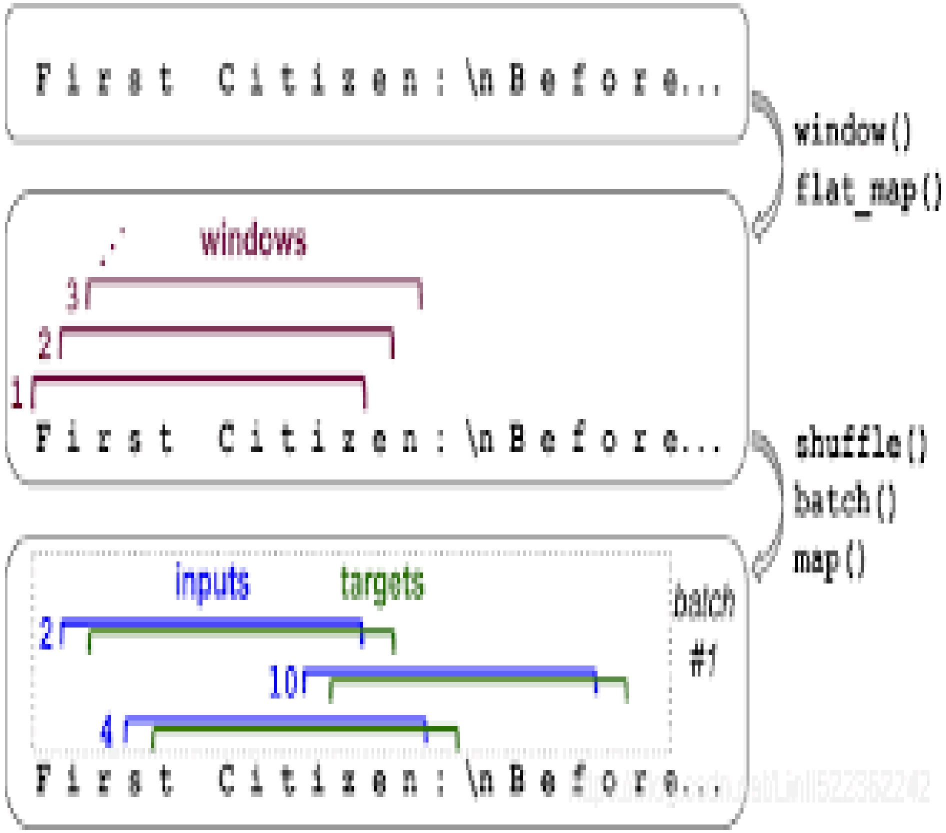
  Alas, I think he shall be come approached and the day
  When little srain would be attain'd into being never fed,
  And who is but a chain and subjects of his death,
  I should not sleep.

https://github.com/karpathy/char-rnn

Stateless RNN

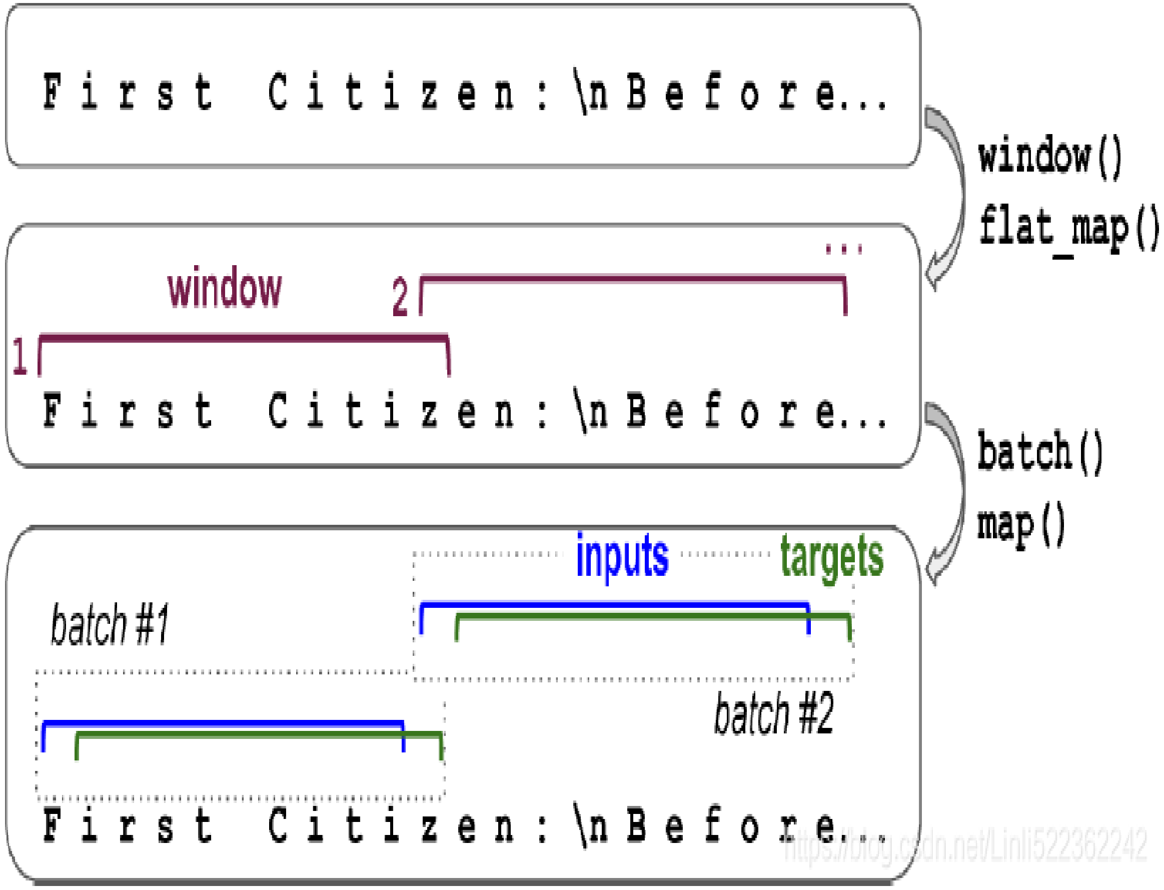
- Window() method creates a nested dataset (list of lists)
- Flat_map() converts nested dataset into a flat dataset
- Batch() to to batch windows and separate the input from the target.
- Stateless RNNs
  - at each training iteration the model starts with a hidden state full of 0s
  - Update this state at each time step

- Chop the Sequential dataset into multiple windows



Stateful RNN

- Stateful RNN
  - Uses sequential nonoverlapping input sequences
  - Preserves the final state after processing one training batch
  - use it as initial state for next training batch
  - Model will learn long-term patterns despite only backpropagating through short sequences

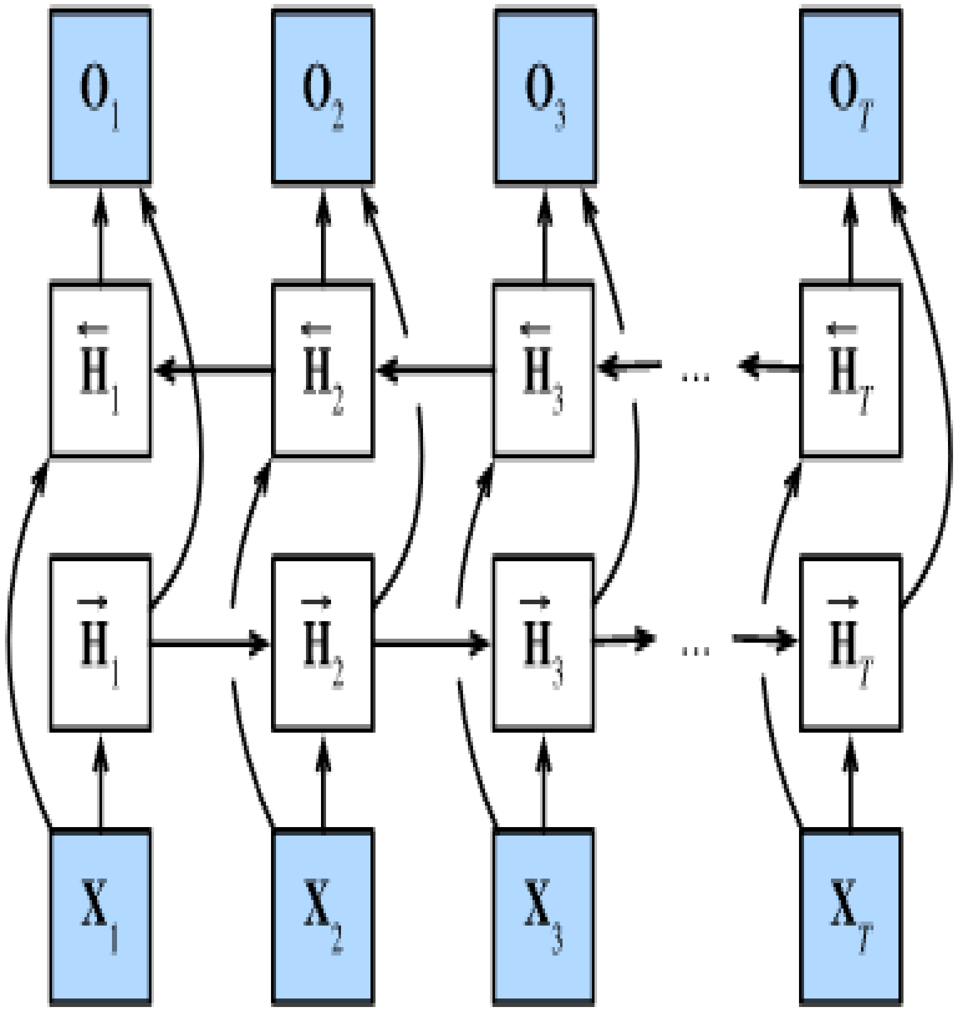


Sentiment Analysis

- Pre-processing steps include:
  - **Tokenizing sentences** to break text down into sentences, words, or other units
  - **Removing stop words** like “if,” “but,” “or,” and so on
  - **Normalizing words** by condensing all forms of a word into a single form
    - Stemming - the process of reducing a word to its word stem
    - Lemmatization- the process of grouping together the inflected forms of a word so they can be analysed as a single item, identified by the word's lemma, or dictionary form
  - **Vectorizing text** by turning the text into a numerical representation for consumption by your classifier
- Example : Sentiment analysis of a movie review :
  - rates positive or negative movie reviews for overall rating for a movie.

Bi-directional RNNs

- Example - speech detection
- I am ___.
- I am ___ hungry.
- I am ___ hungry, and I can eat half a cake.
- Regular RNNs are causal
  - look at past and present inputs to generate output.
- Use 2 recurrent layers on the same inputs
  - One reading words from left to right
  - Another reading words from right to left
- Combine their outputs at each time step



Bi-directional RNN computation

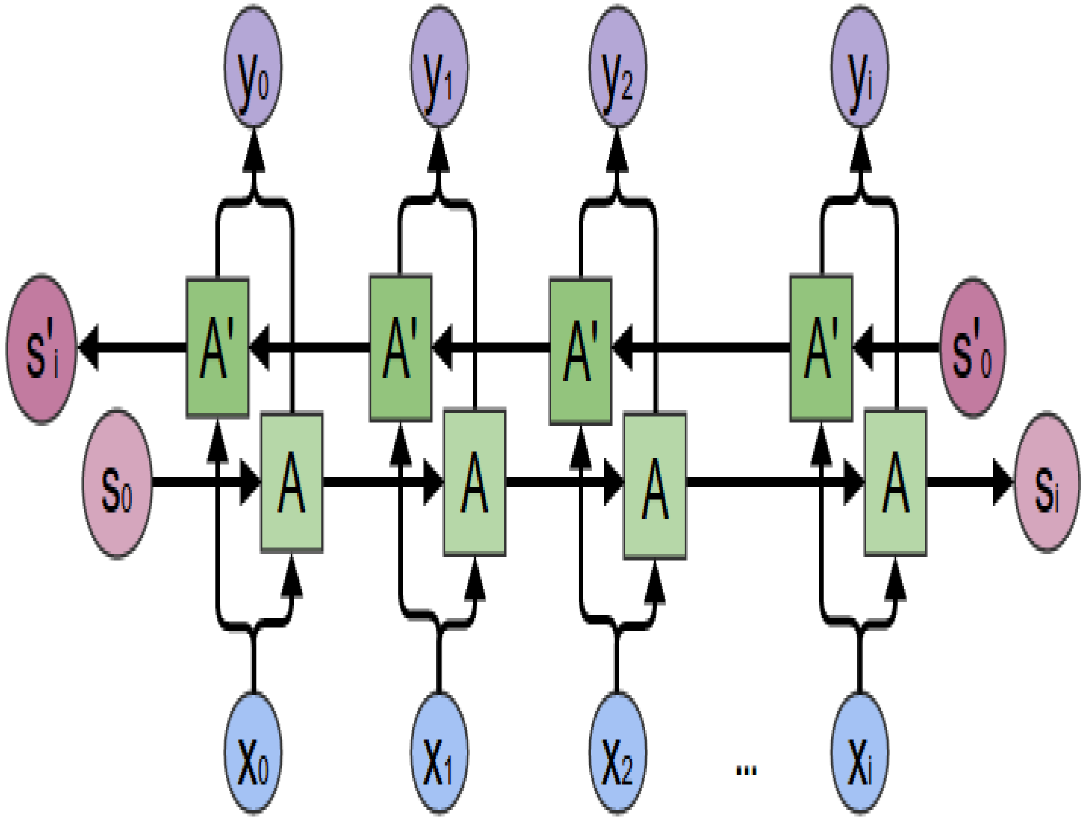
$$A_t(Forward) = \phi(X_t * W_{XA}^{forward} + A_{t-1}(Forward) * W_{AA}^{forward} + b_A^{forward})$$

$$A_t(Backward) = \phi(X_t * W_{XA}^{backward} + A_{t+1}(Backward) * W_{AA}^{backward} + b_A^{backward})$$

- ϕ is the activation function
- W the weight matrix
- b the bias.
- The hidden state at time t is given by a combination of A_t(Forward) and A_t(Backward).
- The output at any given hidden state is:

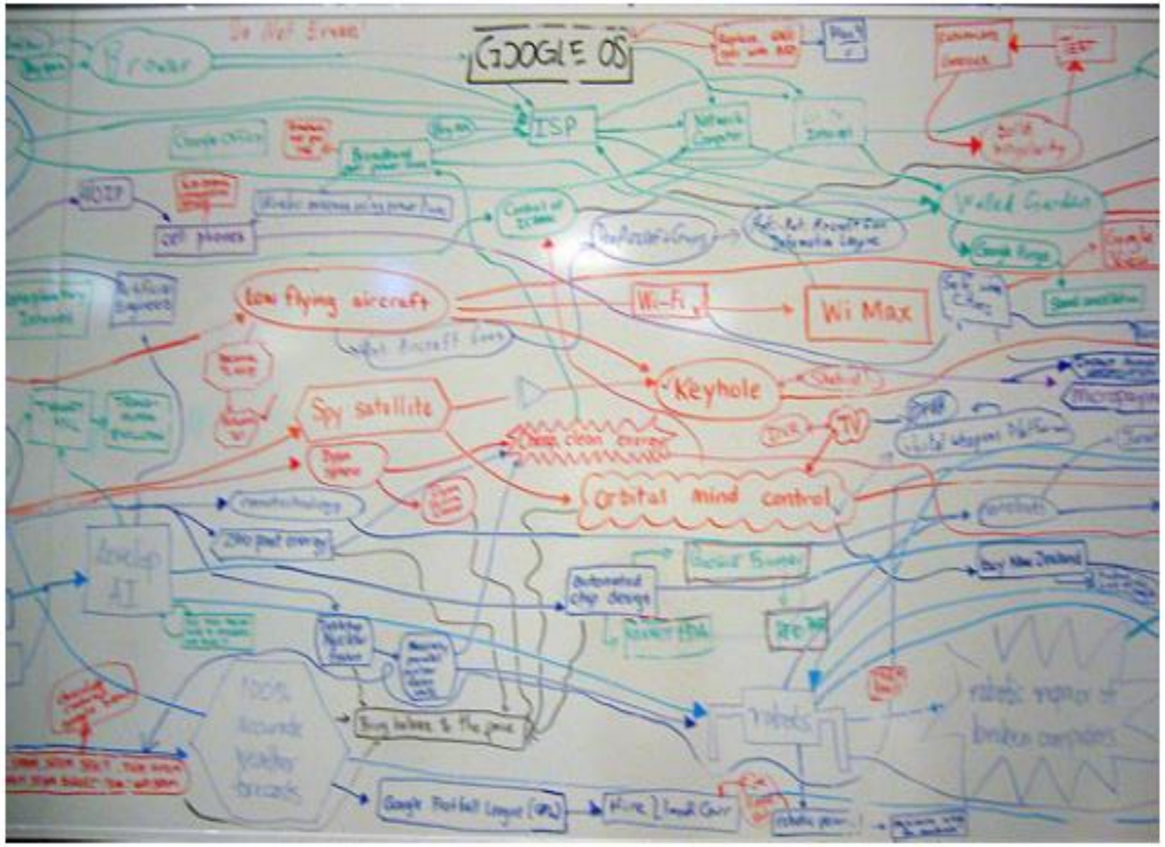
$$O_t = H_t * W_{AY} + b_Y$$

Keras.layers.Bidirectional(
keras.layers.SimpleRNN(10,return_sequences=True))



LSTM and GRU : Introduction

<u>The white board analogy:</u>



- Consider a scenario where we have to evaluate the expression on a whiteboard:

  a= 1, b= 3, c= 5, d=11
  Evaluate ac(bd+a) + ad

- White board would look like:
  - ac=5
  - bd=33
  - bd + a = 34
- Now, if the white board has space to accommodate only 3 steps, the next step cannot fit in the required space and would lead to loss of information.
- So forget ‘bd’ and store ad
  - ac=5
  - bd + a = 34
  - ac(bd+a)= 170

LSTM and GRU : Introduction

a= 1, b= 3, c= 5, d=11
Evaluate ac(bd+a) + ad

- A solution is to do the following:

  - Selectively write:

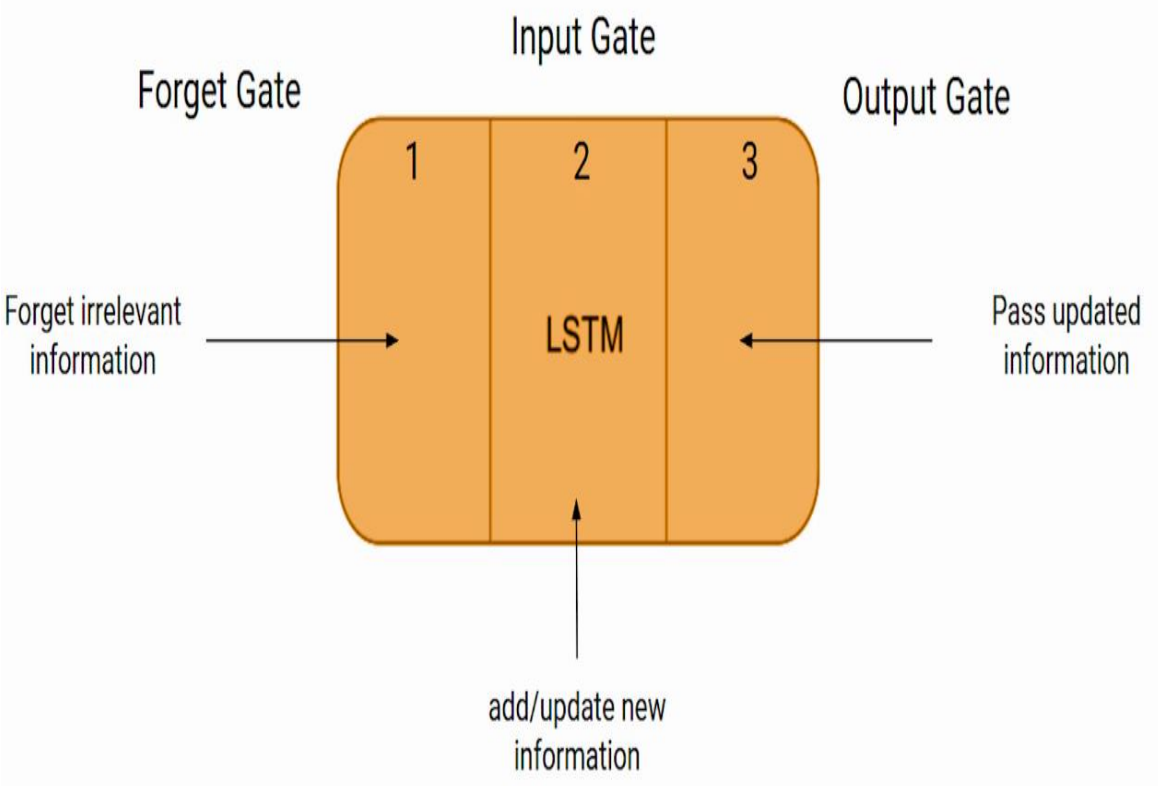
    ac=5

    bd=33

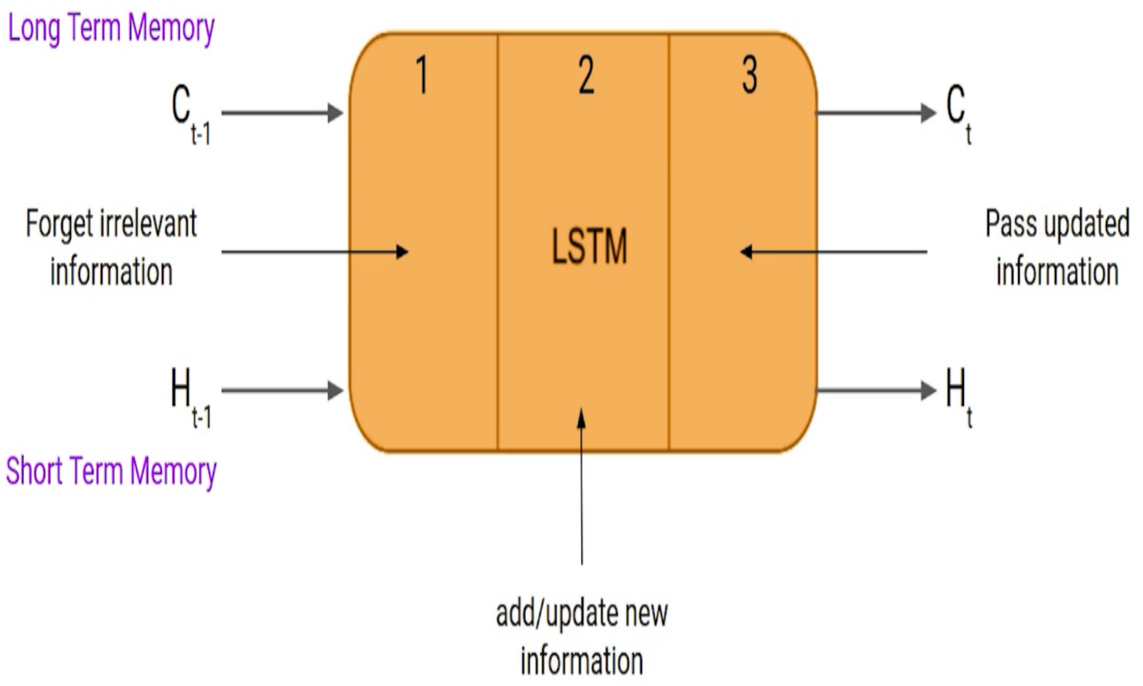
    bd + a = 34
  - Selectively read:
  - Selectively forget: Forget ‘bd’

Since the RNN also has a finite state size, we need to figure out a way to allow it to selectively read, write and forget

Long Short Term Memory (LSTM) cell
(Sepp Hochreiter and Jürgen Schmidhuber 1997)

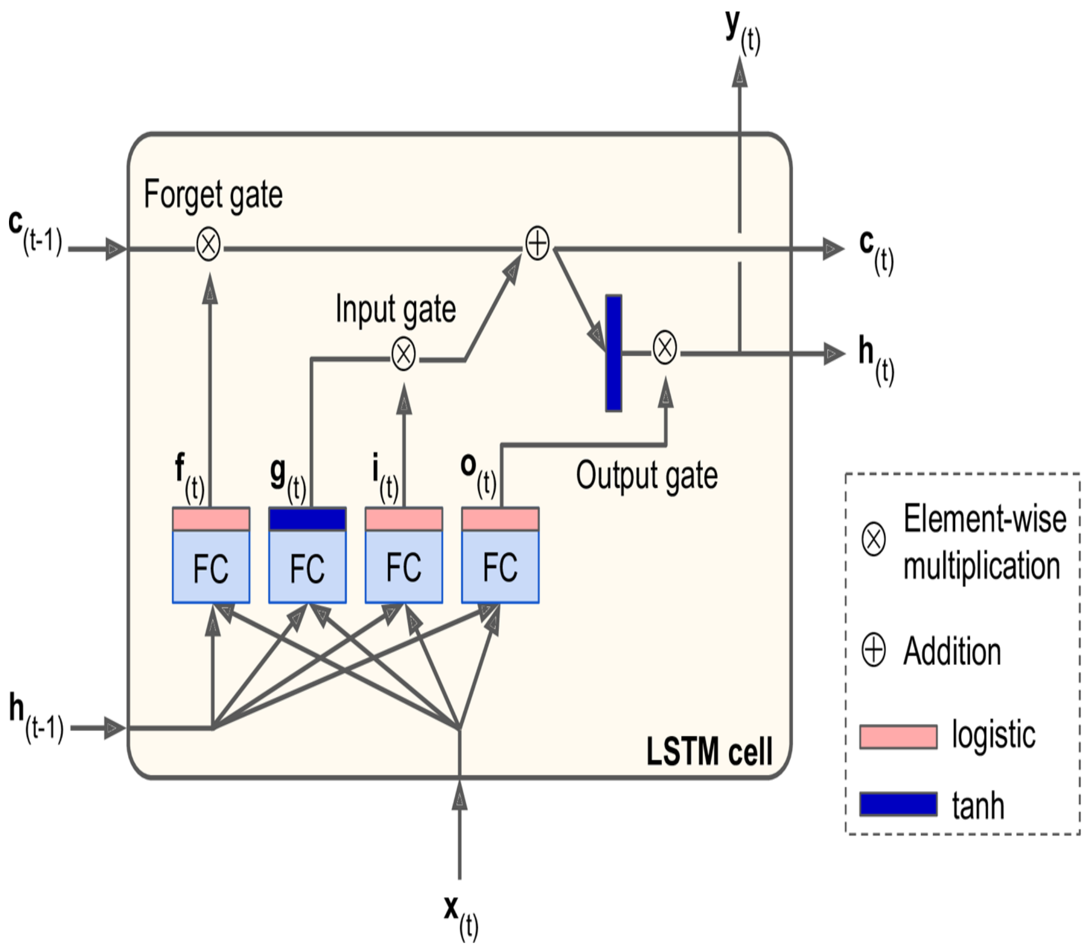


Bob is a nice person. Dan on the other hand is evil.



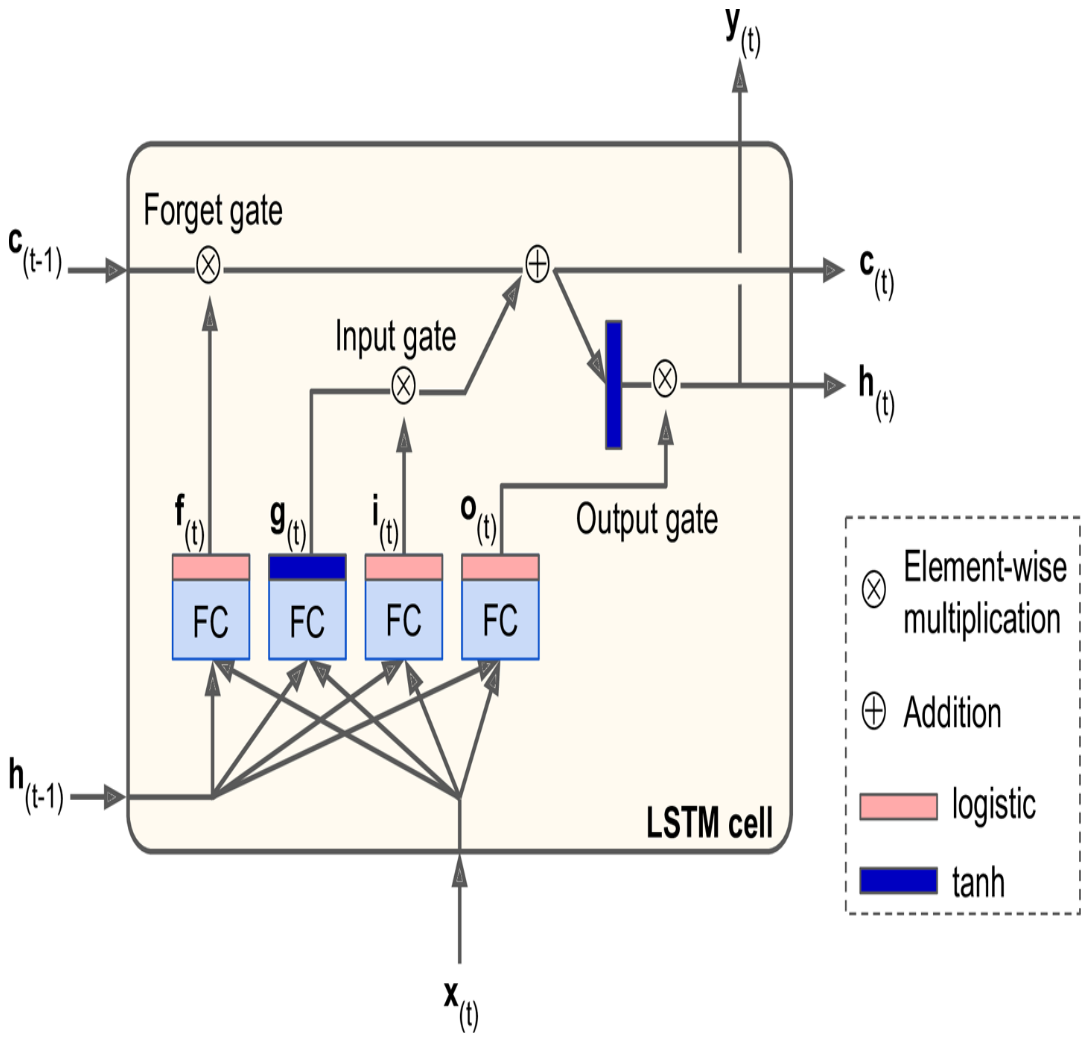
- $H_{(t-1)}$ represents the hidden state of the previous timestamp
- H_t is the hidden state of the current timestamp
- Cell state represented by $C_{(t-1)}$ and $C_{(t)}$ for previous and current timestamp respectively

LSTM Cell



- Neuron called a Cell
- FC are fully connected layers
- Long Term state $c_{(t-1)}$ traverses through forget gate forgetting some memories and adding some new memories
- Long term state $c_{(t-1)}$ is passed through tanh and then filtered by an output gate, which produces short term state $h_{(t)}$
- Update gate- $g_{(t)}$ takes current input $x_{(t)}$ and previous short term state $h_{(t-1)}$
- Important parts of output $g_{(t)}$ goes to long term state

LSTM Cell



- ***Gating Mechanism-*** regulates information that the network stores
- Other 3 layers are gate controllers
- Use logistic activation
- If output is
  - 0- close the gate
  - 1- open the gate
- Forget gate $f_{(t)}$ controls which part of the long—term state should be erased
- Input gate $i_{(t)}$ controls which part of $g_{(t)}$ should be added to long term state
- Output gate $o_{(t)}$ controls which parts of long term state should be read and output at this time state
  - both to $h_{(t)}$ and to $y_{(t)}$

LSTM computations

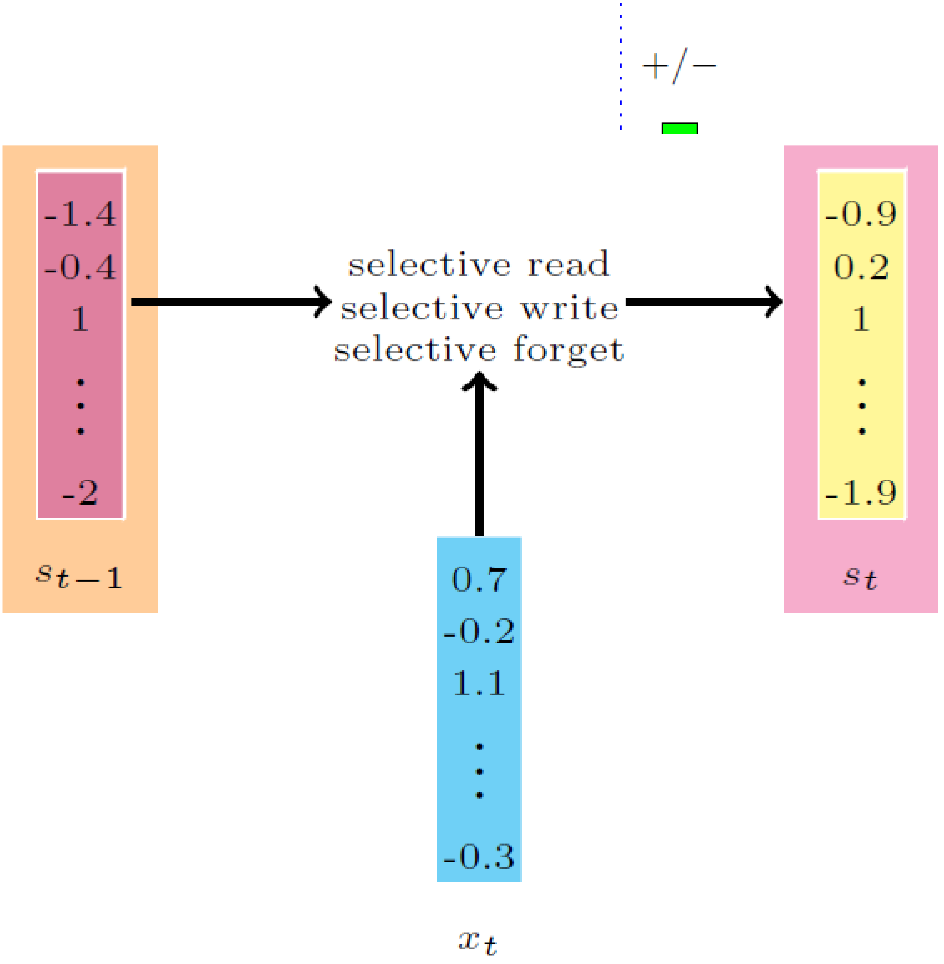
$$
\begin{aligned}
\mathbf{i}_{(t)} &= \sigma(\mathbf{W}_{xi}{}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hi}{}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_{i}) \\
\mathbf{f}_{(t)} &= \sigma(\mathbf{W}_{xf}{}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hf}{}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_{f}) \\
\mathbf{o}_{(t)} &= \sigma(\mathbf{W}_{xo}{}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{ho}{}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_{o}) \\
\mathbf{g}_{(t)} &= \tanh(\mathbf{W}_{xg}{}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hg}{}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_{g}) \\
\mathbf{c}_{(t)} &= \mathbf{f}_{(t)} \otimes \mathbf{c}_{(t-1)} + \mathbf{i}_{(t)} \otimes \mathbf{g}_{(t)} \\
\mathbf{y}_{(t)} &= \mathbf{h}_{(t)} = \mathbf{o}_{(t)} \otimes \tanh(\mathbf{c}_{(t)})
\end{aligned}
$$

$\mathbf{W}_{xi}$, $\mathbf{W}_{xf}$, $\mathbf{W}_{xo}$, $\mathbf{W}_{xg}$ are the weight matrices of each of the four layers for their connection to the input vector $\mathbf{x}_{(t)}$.

$\mathbf{W}_{hi}$, $\mathbf{W}_{hf}$, $\mathbf{W}_{ho}$, and $\mathbf{W}_{hg}$ are the weight matrices of each of the four layers for their connection to the previous short-term state $\mathbf{h}_{(t-1)}$.

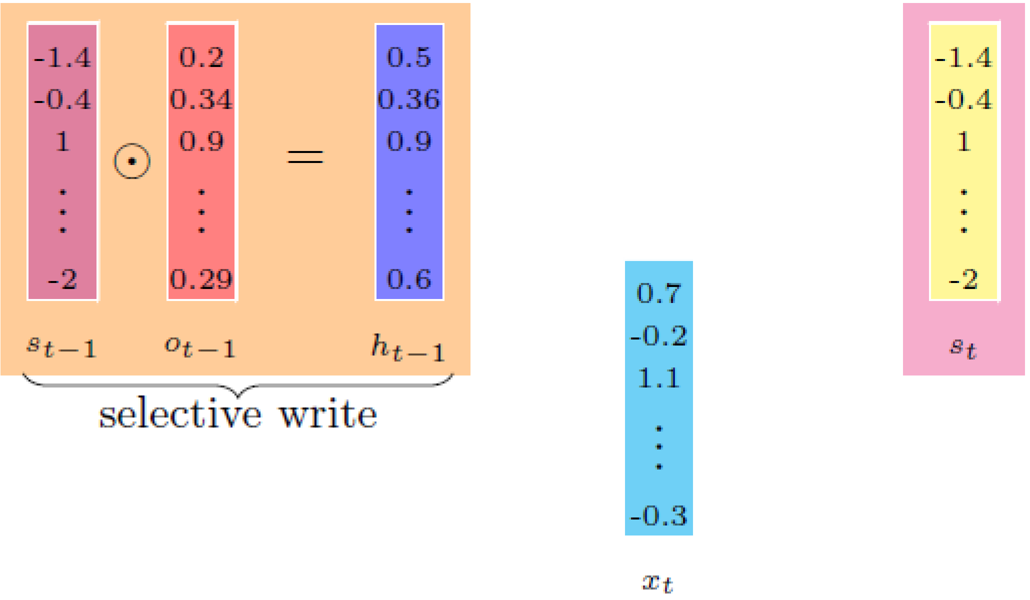
$\mathbf{b}_{i}$, $\mathbf{b}_{f}$, $\mathbf{b}_{o}$, and $\mathbf{b}_{g}$ are the bias terms for each of the four layers.

LSTM and GRU : Introduction



- RNN reads the document from left to right and after every word updates the state.
- By the time we reach the end of the document the information obtained from the first few words is completely lost.
- In our improvised network, ideally, we would like to:
  - **Forget** the information added by stop words (a, the, etc.)
  - **Selectively read** the information added by previous sentiment bearing words (awesome, amazing, etc.)
  - **Selectively write** new information from the current word to the state.

LSTM and GRU : Introduction



The RNN has to learn o_{t-1} along with other parameters (W,U,V)

$$o_{t-1} = \sigma(W_o h_{t-2} + U_o x_{t-1} + b_o)$$
$$h_{t-1} = o_{t-1} \odot \sigma(s_{t-1})$$

New parameters to be learned are: **W_o, U_o, b_o**

O_t is called the output gate.

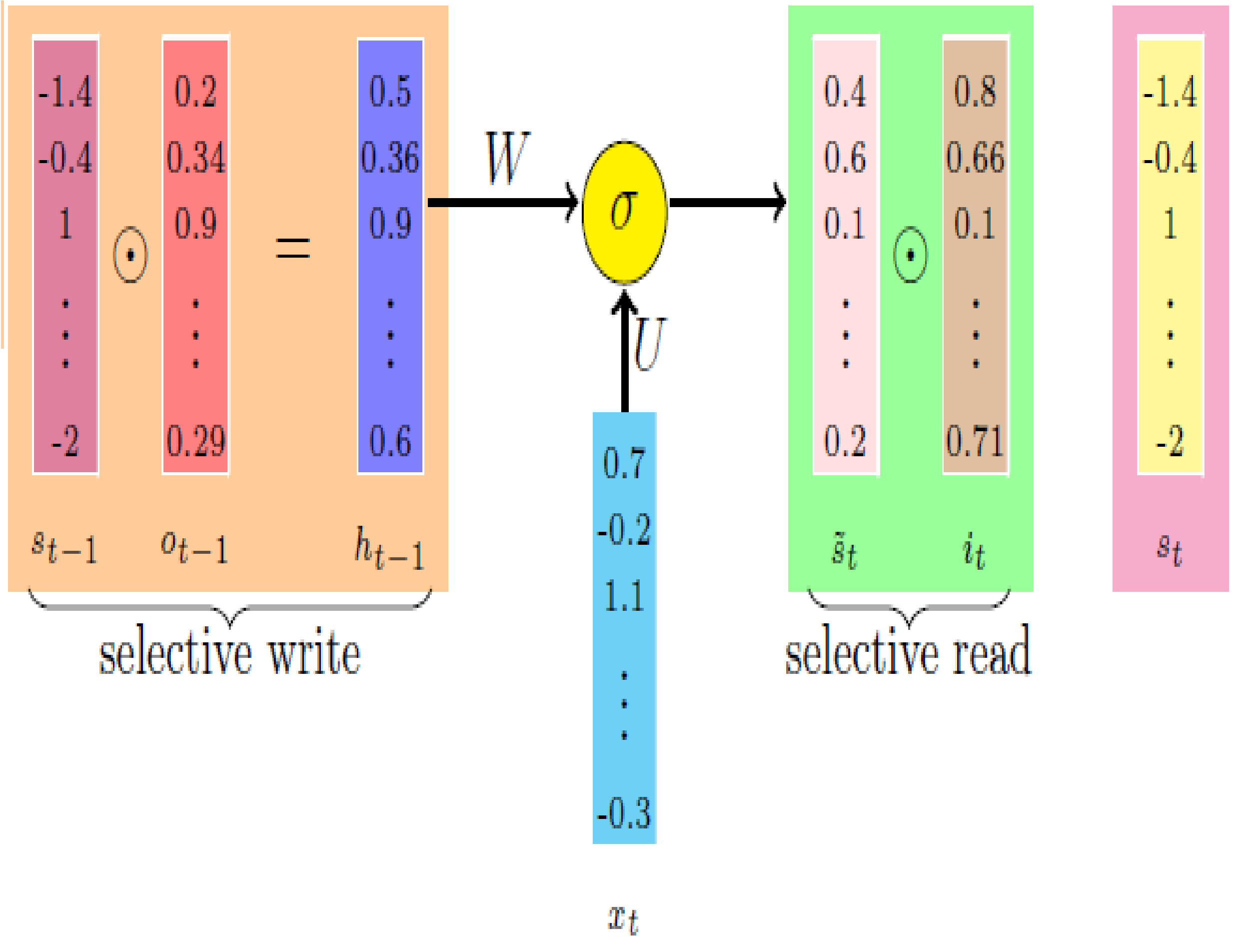
Selectively write:

- In an RNN, the state s_t is defined as follows:

$$s_t = \sigma(W s_{t-1} + U x_t) \;\; (ignoring\ bias)$$

- Instead of passing s_{t-1} as it is, we need to pass (write) only some portions of it.
- To do this, we introduce a vector o_{t-1} which decides what fraction of each element of s_{t-1} should be passed to the next state.
- Each element of o_{t-1} (restricted to be between 0 and 1) gets multiplied with s_{t-1}
- How does RNN know what fraction of the state to pass on?

LSTM and GRU : Introduction



Selectively read:

- We will now use h_{t-1} and x_t to compute the new state at the time step t :

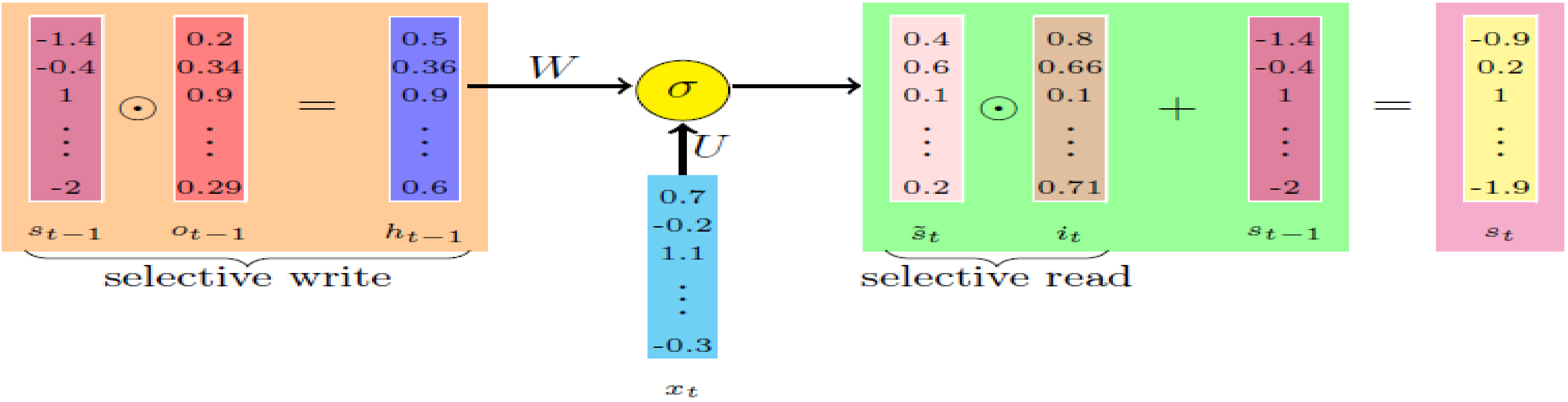
$$\tilde{s}_t = \sigma(Wh_{t-1} + Ux_t + b)$$

- Again, to pass only useful information from to s_t, we selectively read from it before constructing the new cell state.

- To do this we introduce another gate called as the input gate:

$$i_t = \sigma(W_i h_{t-1} + U_i x_t + b_i)$$

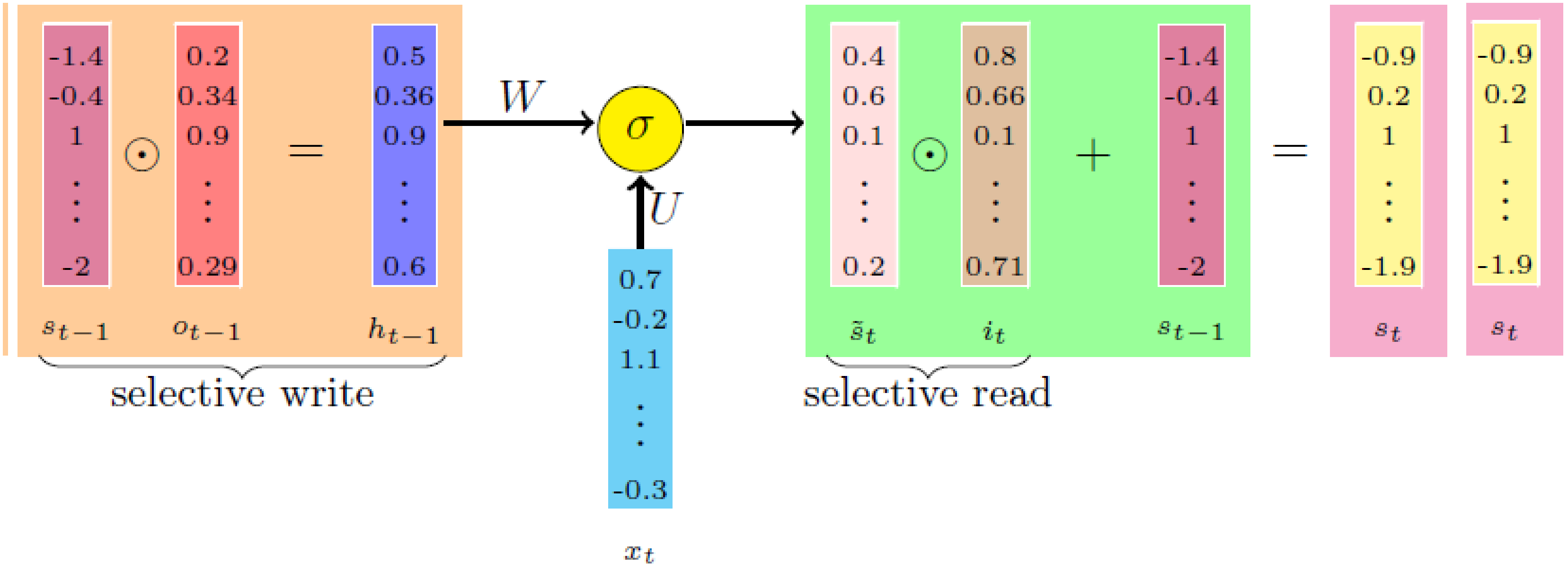
- And use $i_t \odot \tilde{s}_t$ selectively read the information.



Selectively forget

- How do we combine s_{t-1} and $\tilde{s}_t$ to get the new state?

$$s_t = s_{t-1} + i_t \odot \tilde{s}_t$$



Selectively forget

- How do we combine s_{t-1} and $\tilde{s}_t$, get the new state?

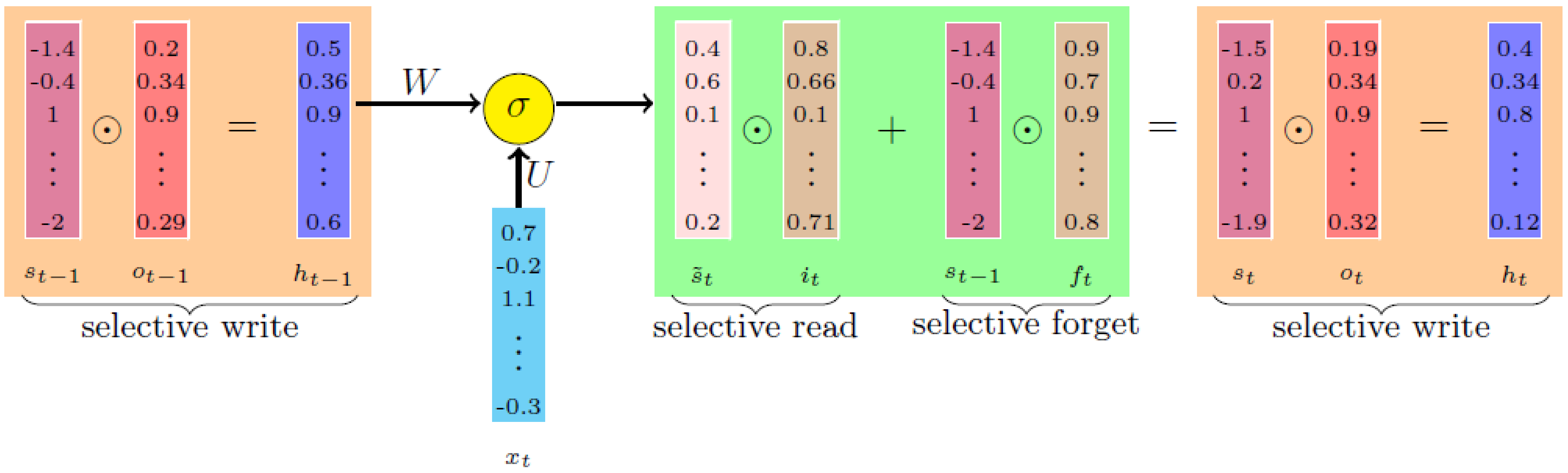
$$s_t = s_{t-1} + i_t \odot \tilde{s}_t$$

$$s_t = f_t \odot s_{t-1} + i_t \odot \tilde{s}_t$$

- But we may not want to use the whole of s_{t-1} but forget some parts of it.
- To do this a forget gate is introduced:

$$f_t = \sigma(W_f h_{t-1} + U_f x_t + b_f)$$

LSTM (Long Short-Term Memory)



Gates:

$$o_t = \sigma(W_o h_{t-1} + U_o x_t + b_o)$$

$$i_t = \sigma(W_i h_{t-1} + U_i x_t + b_i)$$

$$f_t = \sigma(W_f h_{t-1} + U_f x_t + b_f)$$

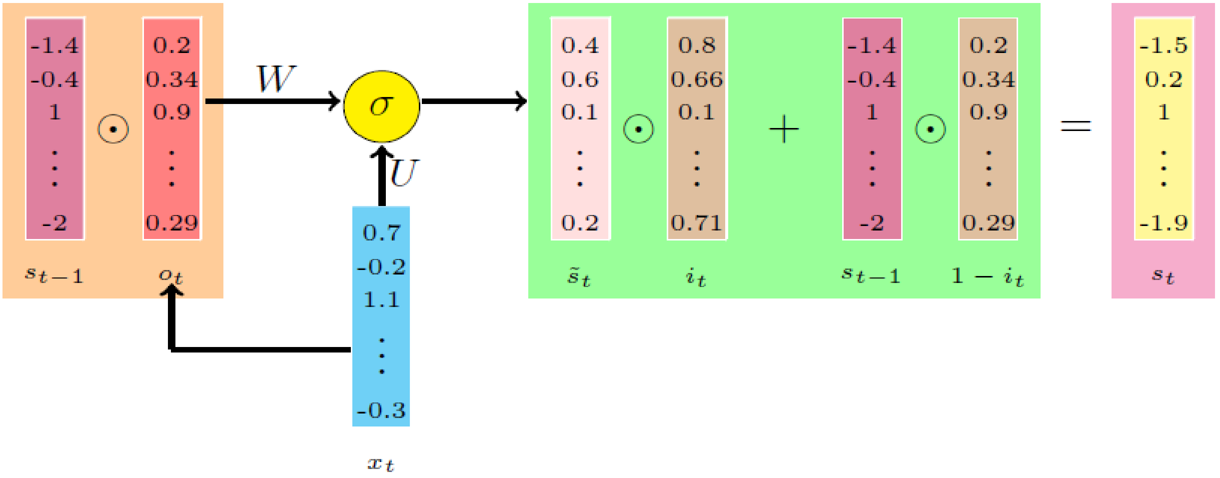
States:

$$\tilde{s}_t = \sigma(W h_{t-1} + U x_t + b)$$

$$s_t = f_t \odot s_{t-1} + i_t \odot \tilde{s}_t$$

$$h_t = o_t \odot \sigma(s_t)$$

Gated Recurrent Unit



The full set of equations for GRUs

Gates:

$$o_t = \sigma(W_o s_{t-1} + U_o x_t + b_o)$$

$$i_t = \sigma(W_i s_{t-1} + U_i x_t + b_i)$$

States:

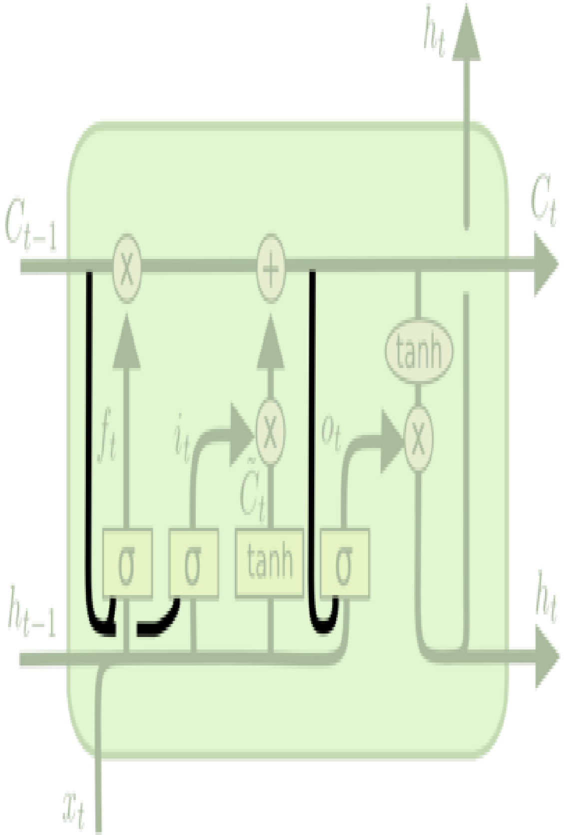
$$\tilde{s}_t = \sigma(W(o_t \odot s_{t-1}) + U x_t + b)$$

$$s_t = (1 - i_t) \odot s_{t-1} + i_t \odot \tilde{s}_t$$

- No explicit forget gate (the forget gate and input gates are tied)
- The gates depend directly on s_{t-1} and not the intermediate h_{t-1} as in the case of LSTMs

LSTM with Peephole connections (Felix Gers and Jürgen Schmidhuber in 2000)

- In LSTM cell, the gate controllers get input $\mathbf{x}_{(t)}$ and $\mathbf{h}_{(t-1)}$.
- Can be given more context by letting them peek at the long-term state as well
- LSTM variant with extra connections called *peephole connections*
  - previous long-term state $\mathbf{c}_{(t-1)}$ is added as an input to the controllers of the forget gate and the input gate
  - current long-term state $\mathbf{c}_{(t)}$ is added as input to the controller of the output gate

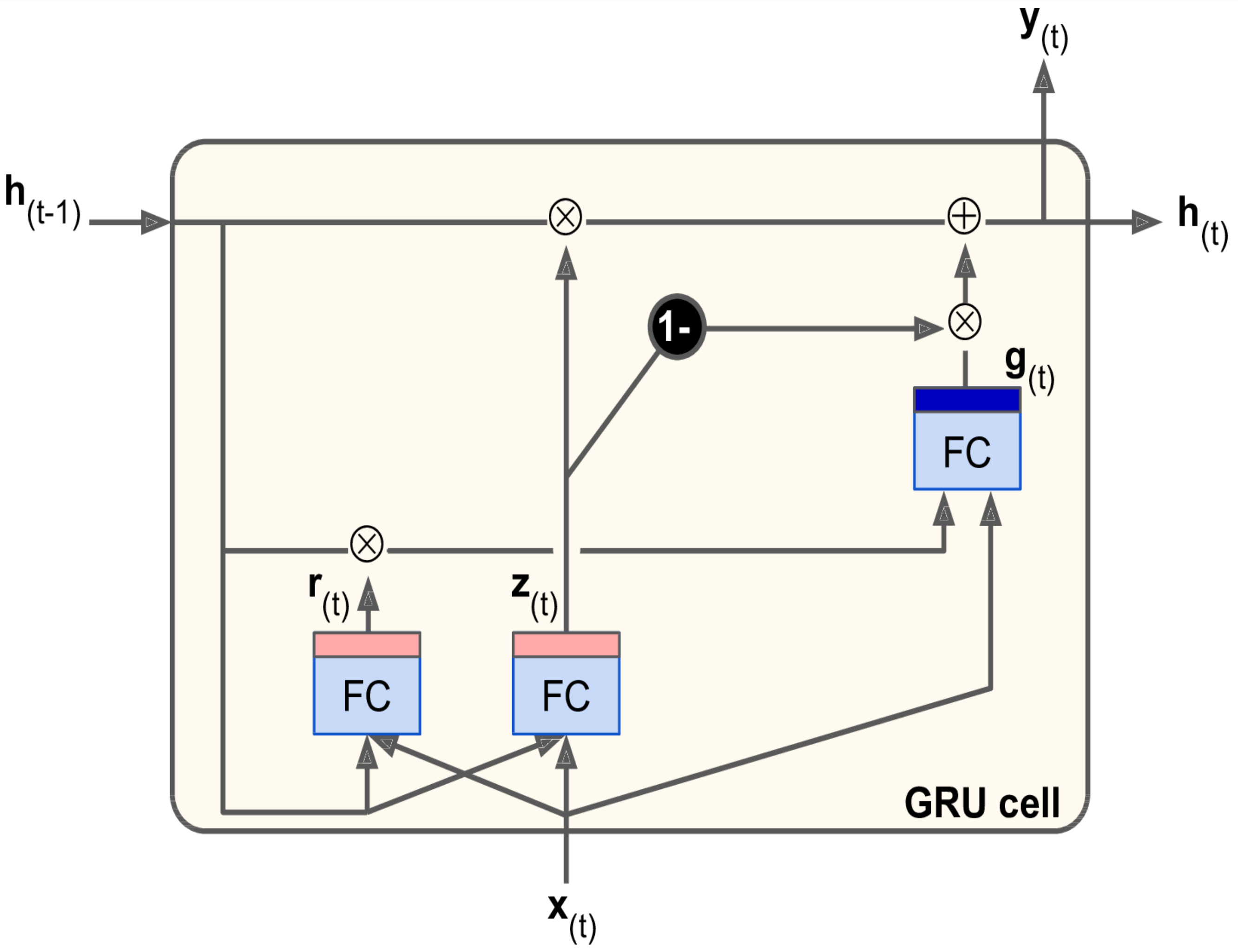


$$f_t = \sigma\left(W_f \cdot [C_{t-1}, h_{t-1}, x_t] + b_f\right)$$

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

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

Gated Recurrent Unit CELL (Kyunghyun Cho et al, 2014)



- No explicit memory unit

The main simplifications of LSTM are:

- Both state vectors are merged into a single vector $\mathbf{h}_{(t)}$.
- **Gate controller** $\mathbf{z}_{(t)}$ controller controls both the forget gate and the input gate.
- If the gate controller outputs
  - 1, the forget gate is open and the input gate is closed.
  - 0, the opposite happens
  - whenever a memory must be written, the location where it will be stored is erased first.
- No output gate, the full state vector is output at every time step.
- **Reset gate controller** $\mathbf{r}_{(t)}$ that controls which part of the previous state will be shown to the main layer $\mathbf{g}_{(t)}$.

LSTM vs GRU computation

$$\mathbf{i}_{(t)} = \sigma({\mathbf{W}_{xi}}^T \cdot \mathbf{x}_{(t)} + {\mathbf{W}_{hi}}^T \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_i)$$
$$\mathbf{f}_{(t)} = \sigma({\mathbf{W}_{xf}}^T \cdot \mathbf{x}_{(t)} + {\mathbf{W}_{hf}}^T \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_f)$$
$$\mathbf{o}_{(t)} = \sigma({\mathbf{W}_{xo}}^T \cdot \mathbf{x}_{(t)} + {\mathbf{W}_{ho}}^T \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_o)$$
$$\mathbf{g}_{(t)} = \tanh({\mathbf{W}_{xg}}^T \cdot \mathbf{x}_{(t)} + {\mathbf{W}_{hg}}^T \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_g)$$
$$\mathbf{c}_{(t)} = \mathbf{f}_{(t)} \otimes \mathbf{c}_{(t-1)} + \mathbf{i}_{(t)} \otimes \mathbf{g}_{(t)}$$
$$\mathbf{y}_{(t)} = \mathbf{h}_{(t)} = \mathbf{o}_{(t)} \otimes \tanh(\mathbf{c}_{(t)})$$

$$\mathbf{z}_{(t)} = \sigma({\mathbf{W}_{xz}}^T \cdot \mathbf{x}_{(t)} + {\mathbf{W}_{hz}}^T \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_z)$$
$$\mathbf{r}_{(t)} = \sigma({\mathbf{W}_{xr}}^T \cdot \mathbf{x}_{(t)} + {\mathbf{W}_{hr}}^T \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_r)$$
$$\mathbf{g}_{(t)} = \tanh({\mathbf{W}_{xg}}^T \cdot \mathbf{x}_{(t)} + {\mathbf{W}_{hg}}^T \cdot (\mathbf{r}_{(t)} \otimes \mathbf{h}_{(t-1)}) + \mathbf{b}_g)$$
$$\mathbf{h}_{(t)} = \mathbf{z}_{(t)} \otimes \mathbf{h}_{(t-1)} + (1 - \mathbf{z}_{(t)}) \otimes \mathbf{g}_{(t)}$$

- GRU Performance is good but may have a slight dip in the accuracy
- But lesser number of trainable parameters which makes it advantageous to use

References

- Ian Goodfellow, Yoshua Bengio and Aaron Courville, “Deep Learning”, MIT Press 2016
- NPTEL Notes from CS6910 Deep Learning , Mitesh Khapra,
- Coursera Notes from Neural Networks and Deep Learning , Andrew NG
- Aurelien Geron, “Hands-On Machine Learning with Scikit-Learn , Keras & Tensorflow, OReilly Publications