University of Windsor

RECURRENT NEURAL NETWORKS (RNNs)

Team Members
Divykumar Sachapara
Khushbuben Ramoliya
Smit Sanjaykumar Patel

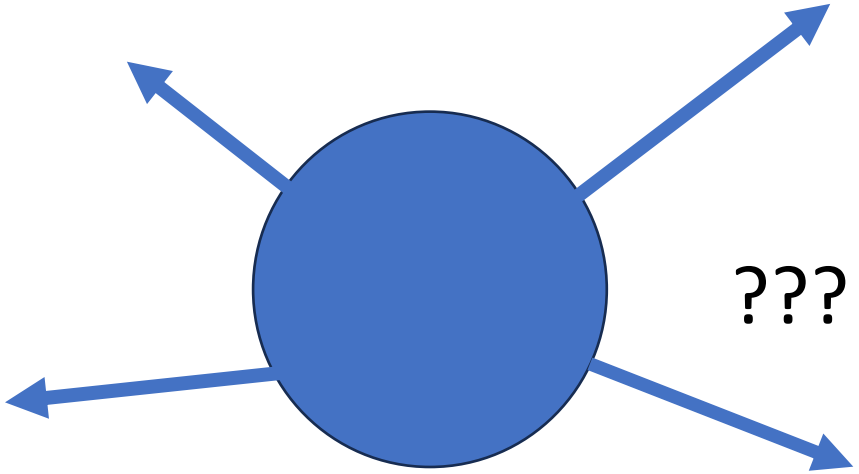
Instructor – Dr. Yasser Alginahi

October 27th , 2023

University of Windsor

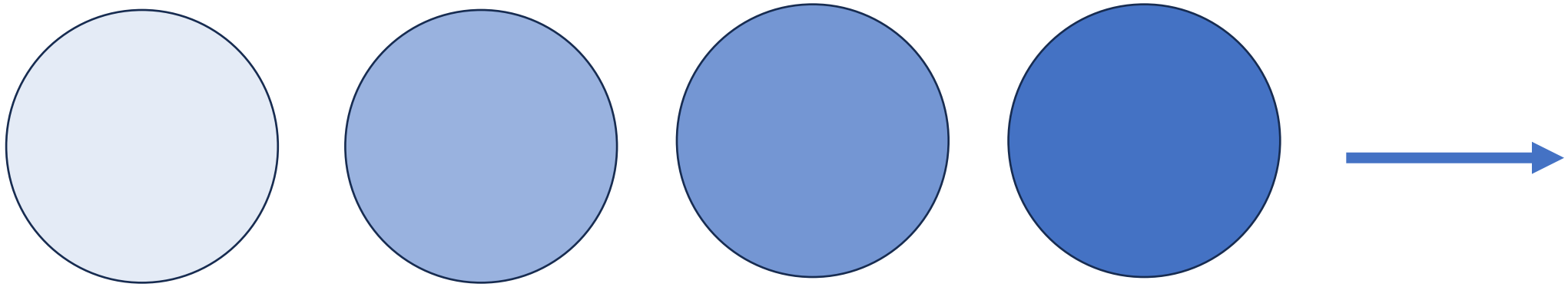
Table of Content

What is sequencing?



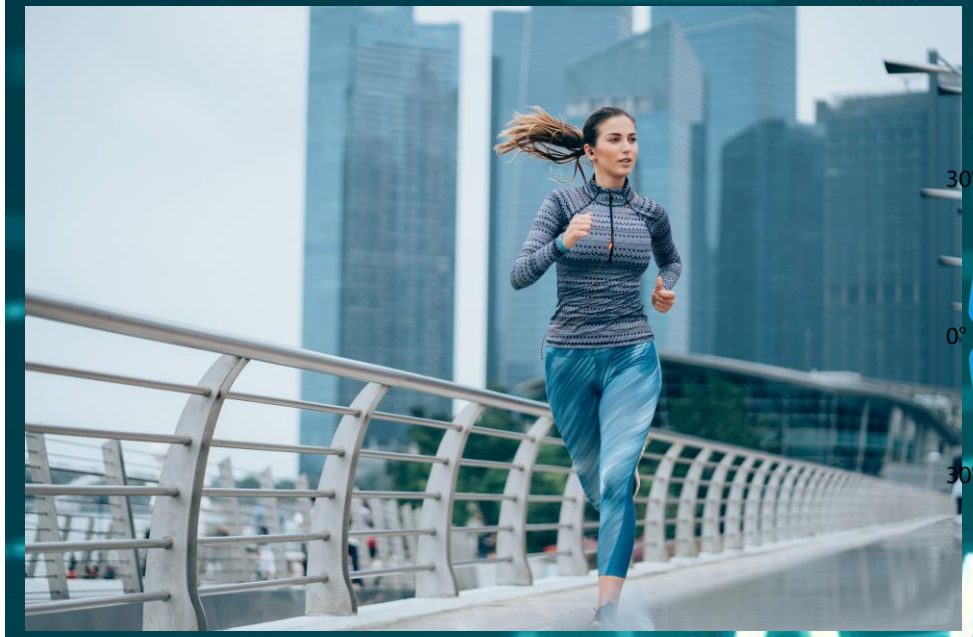
[Source: https://tinyurl.com/3pawnch9]

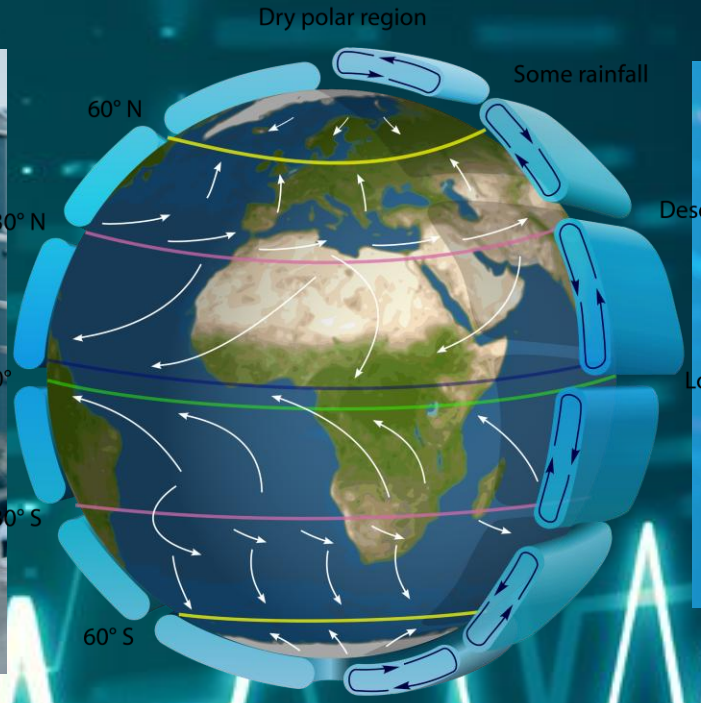
Given an image of a ball, can you predict where it will go next?



[Source: https://tinyurl.com/3pawnch9]

Given an image of a ball, can you predict where it will go next?





University of Windsor

What is RNN and how does it differ from CNN?

- Recurrent Neural Networks (RNNs) are a class of neural networks that are designed to handle sequential data by maintaining a hidden state that carries information from one time step to the next.[1]

- A convolutional neural network (CNN) is a powerful artificial neural network primarily used for image recognition and processing because it can recognize patterns in images, but it demands millions of labeled data points for training.[2]

- In a CNN, it takes in images of a fixed size and produces a predicted class label for each image, along with a confidence level. In contrast, in RNNs, the input size and resulting output can vary.[3]

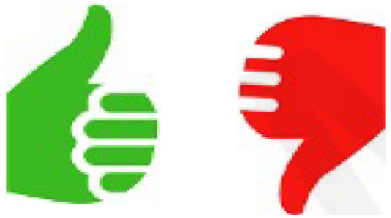
Sequential Model

- Sequential modeling in the context of Recurrent Neural Networks (RNNs) refers to the process of modeling and predicting sequential data, where the order of elements in the sequence matters.
- RNNs are a type of neural network designed specifically for sequential data.
- They are characterized by their ability to maintain a hidden state that captures information from previous elements in the sequence, which can be used to influence predictions for the next element.
- Sequential data includes any data where the order of elements is significant. This can include time series data, natural language text, audio signals, DNA sequences, etc.[4]

University of Windsor

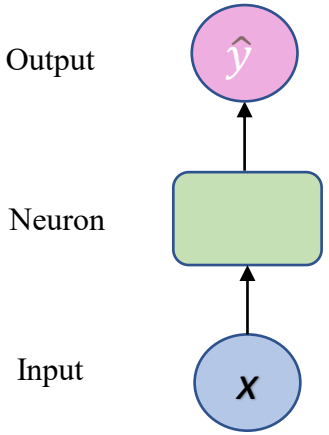
Sequence Modeling Applications and its Types

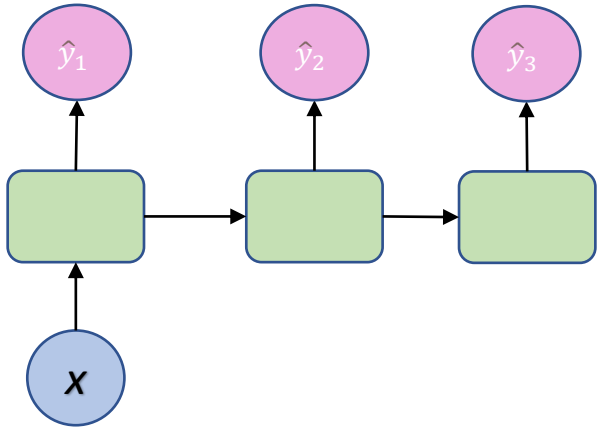
Will I pass this Class !!?



Binary Classification

One to one





One to many

Image Captioning

A Baseball player throws a ball !!

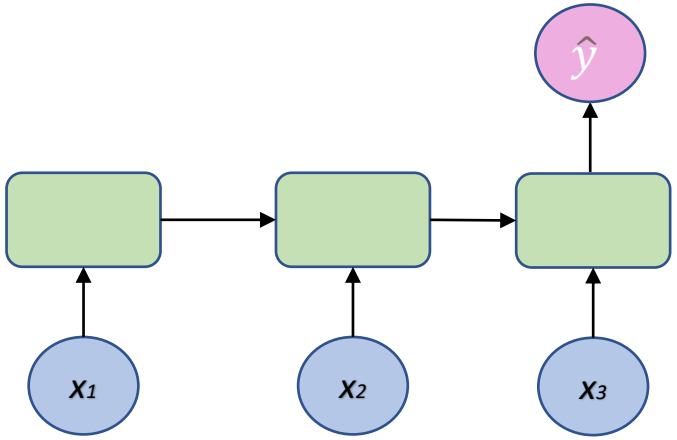
This stock image of a scientist is PRICELESS. I too often find myself inspecting each nugget of dry ice one by one. Can never be too careful 🤣

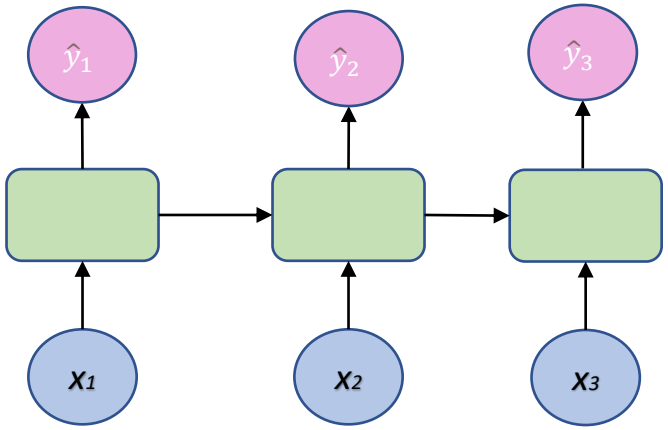
9:10 PM - Apr 24, 2018

♡2,857 💬688 people are talking about this

Sentiment Classification

Many to one



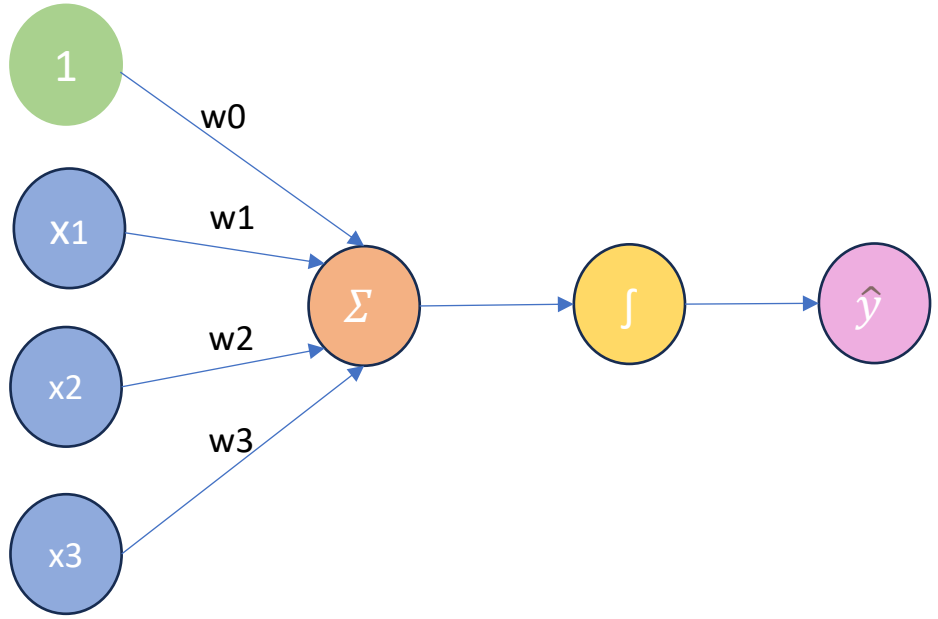


many to many

Google/Media Translator

[Source: https://tinyurl.com/3pawnch9]

The Perceptron



source : https://tinyurl.com/3pawnch9

Inputs Weights Sum Non-Linearity Output

- The structural building block of deep learning.

$$\hat{y} = g\left(\sum_{i=1}^{m} x_i w_i\right)$$

Output → ŷ

← linear combination of inputs

Non-linear activation function ↑

$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i w_i\right)$$

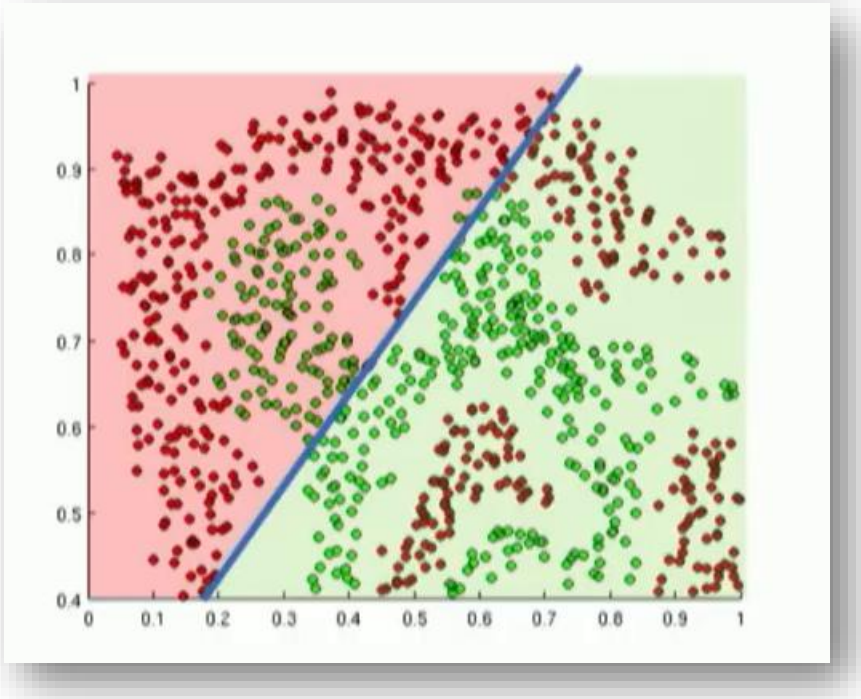
Bias →

$$\hat{y} = g(w_0 + X^T W)$$

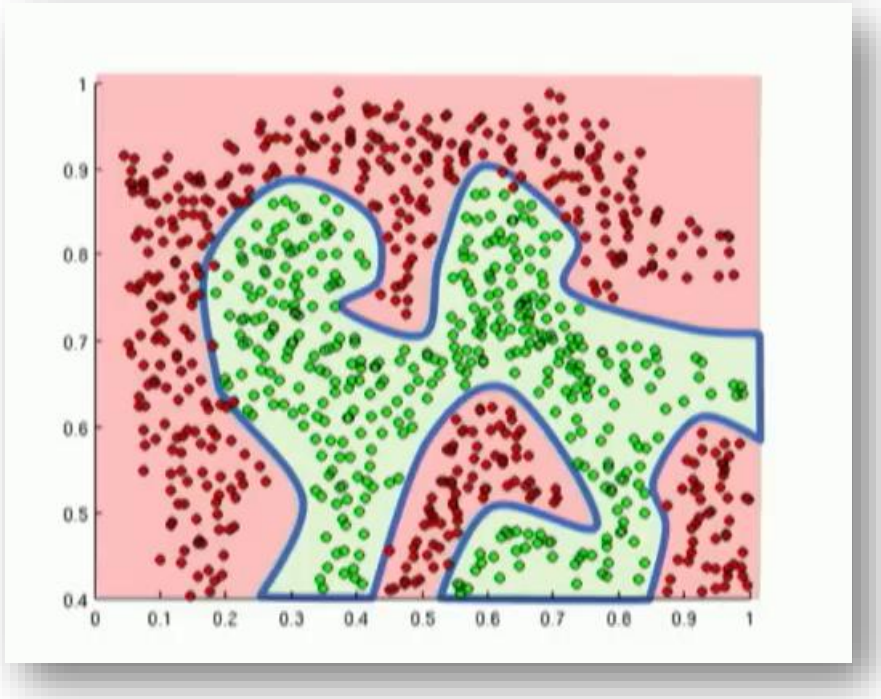
- Sigmoid and tanh are two of the most common non-linear functions in deep learning.

Importance of Activation Functions

The Purpose of activation function Is to introduce non-linearities into the network.



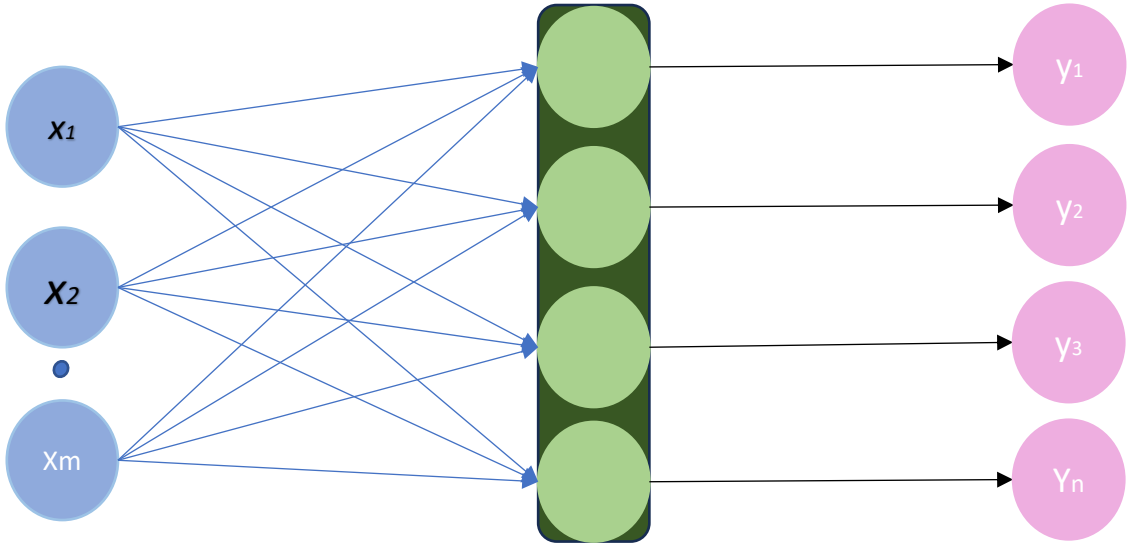
Linear activation function produce linear decisions no matter the network size.



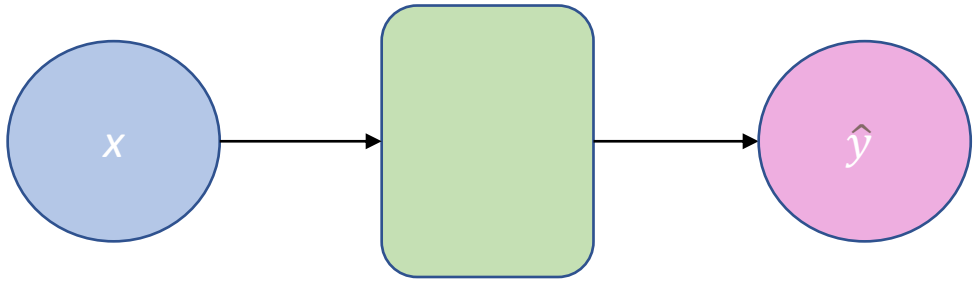
Non-linearities allow us to approximate arbitrarily complex functions.

[Source: https://tinyurl.com/3pawnch9]

Feed-Forward Networks



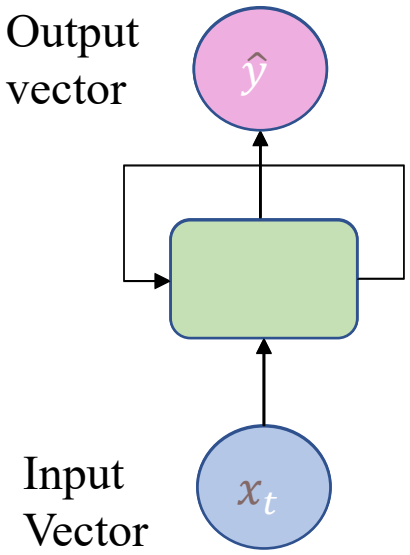
$x \in \mathbb{R}^m$ $y \in \mathbb{R}^n$



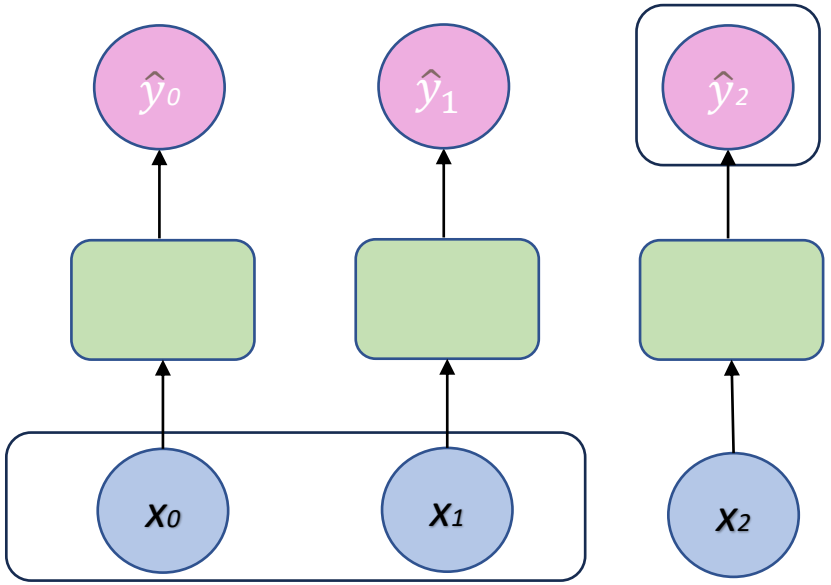
$x_t \in \mathbb{R}^n$ $y_t \in \mathbb{R}^n$

[Source: https://tinyurl.com/3pawnch9]

Handling Individual Time Steps

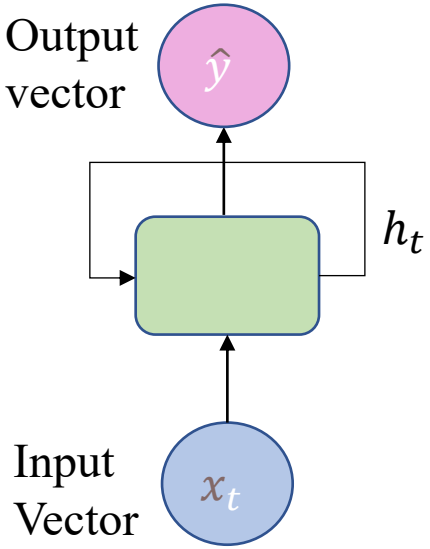


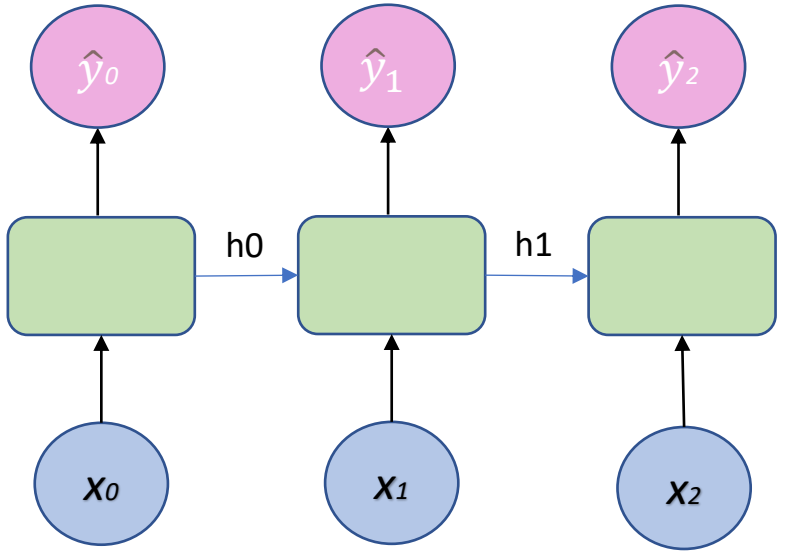
[Source: https://tinyurl.com/3pawnch9]



$$\hat{\boldsymbol{y}}(\boldsymbol{t}) = \boldsymbol{f}(\boldsymbol{x_t})$$

Neurons with Reccurence



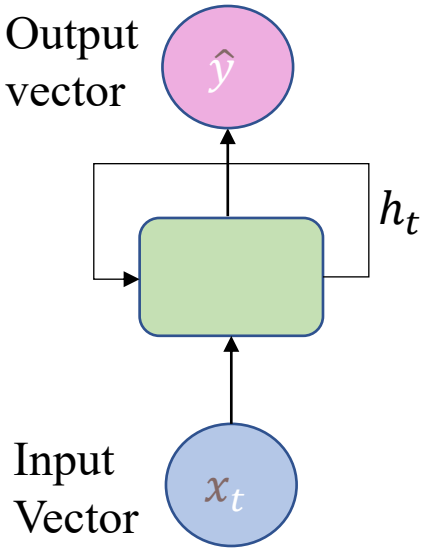


$$\hat{y}(t) = f(x_t, h_{t-1})$$

OUTPUT INPUT PAST MEMORY

[Source: https://tinyurl.com/3pawnch9]

Recurrent Neural Networks(RNNs)



- Apply a recurrence relation at every time step to process a sequence.

Cell State $\boldsymbol{h(t)} = \boldsymbol{f_w}\left(\boldsymbol{x_t}, \boldsymbol{h_{t-1}}\right)$

Function with weights w — Input — Old state

- Note: The same function and set of parameters are used at every time step.

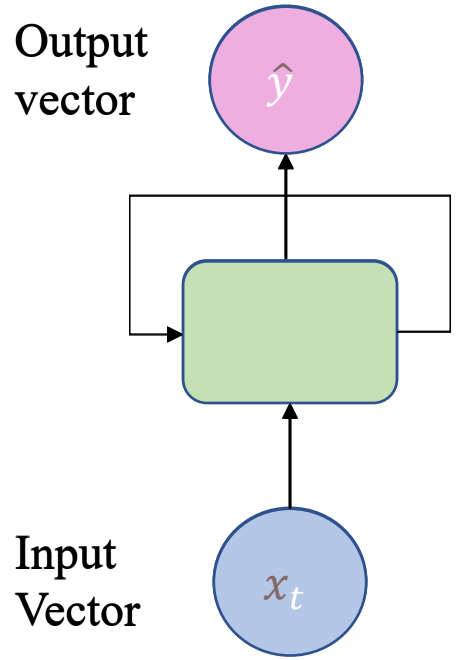
RNNs have a state $\boldsymbol{h_t}$, that is updates at each time step as a sequence is processed.

RNN INTUITION

```
my_rnn = RNN() (#Define RNN)
hidden_state = [0, 0, 0, 0]  (#Initiate hidden state)

sentence = ["I","love","recurrent", "neural"] (#Input)
```

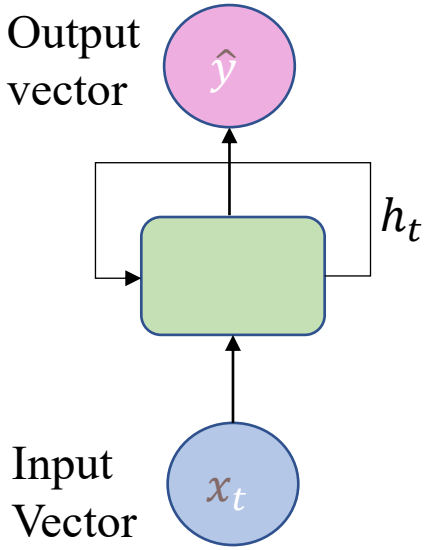
```
for word in sentence: (#For loop for sequencing through I/P)
    prediction, hidden_state = my_rnn(word, hidden_state)
```

```
next_word_prediction = prediction (#Prediction for last word)
# >> "networks!"
```



GitHub link: https://github.com/KhushbuKathrotiya/RNN_Next_Sequence_Word

RNN State Update and Output



[Source: https://tinyurl.com/3pawnch9]

Input Vector

$$\boldsymbol{x_t}$$

Updated Hidden State

$$\boldsymbol{h_t = \tanh\left(W_{hh}^T h_{t-1} + W_{xh}^T x_t\right)}$$

Output Vector

$$\boldsymbol{\hat{y}_t = w_{hy}^T h_t}$$

$\boldsymbol{W_{hh}^T}$, $\boldsymbol{W_{xh}^T}$, $\boldsymbol{w_{hy}^T}$ = Weight matrices

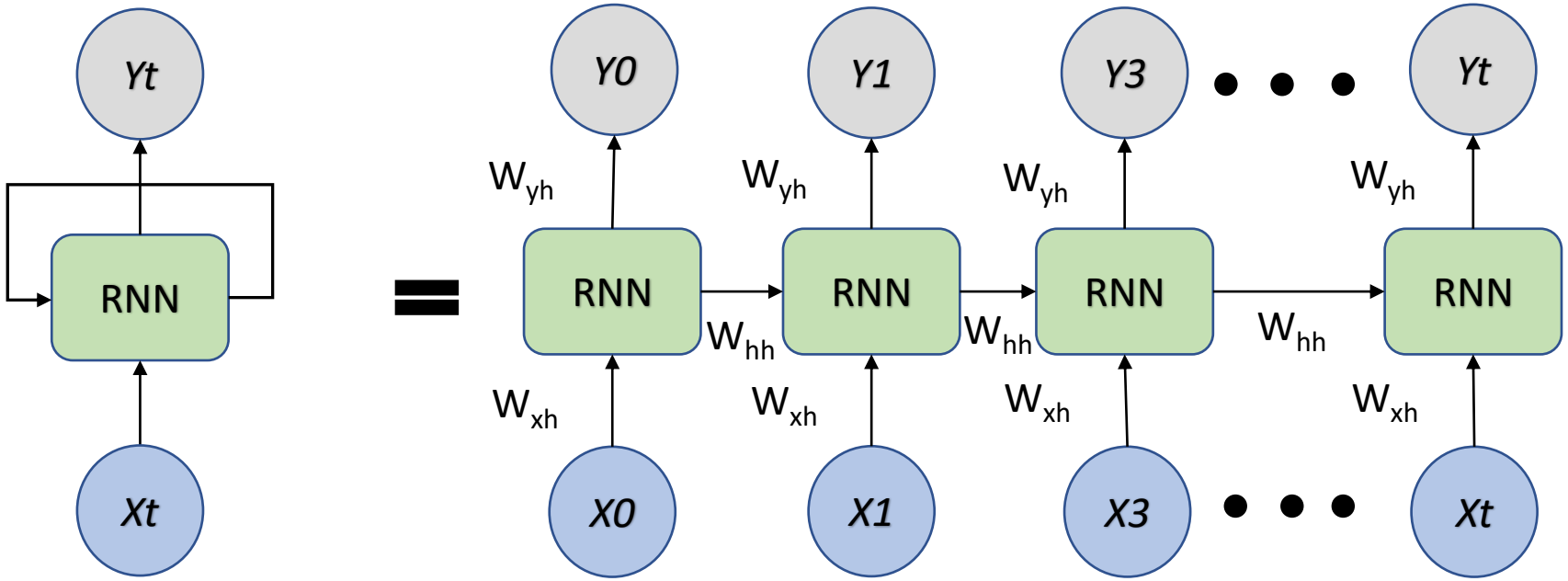
RNNs: Computational Graph Across Time:

- Represent as a computational graph unrolled across time
- Re-use the same **weight matrices** at every time step

W_{yh} = Weight matrices define to update to generate predicted output

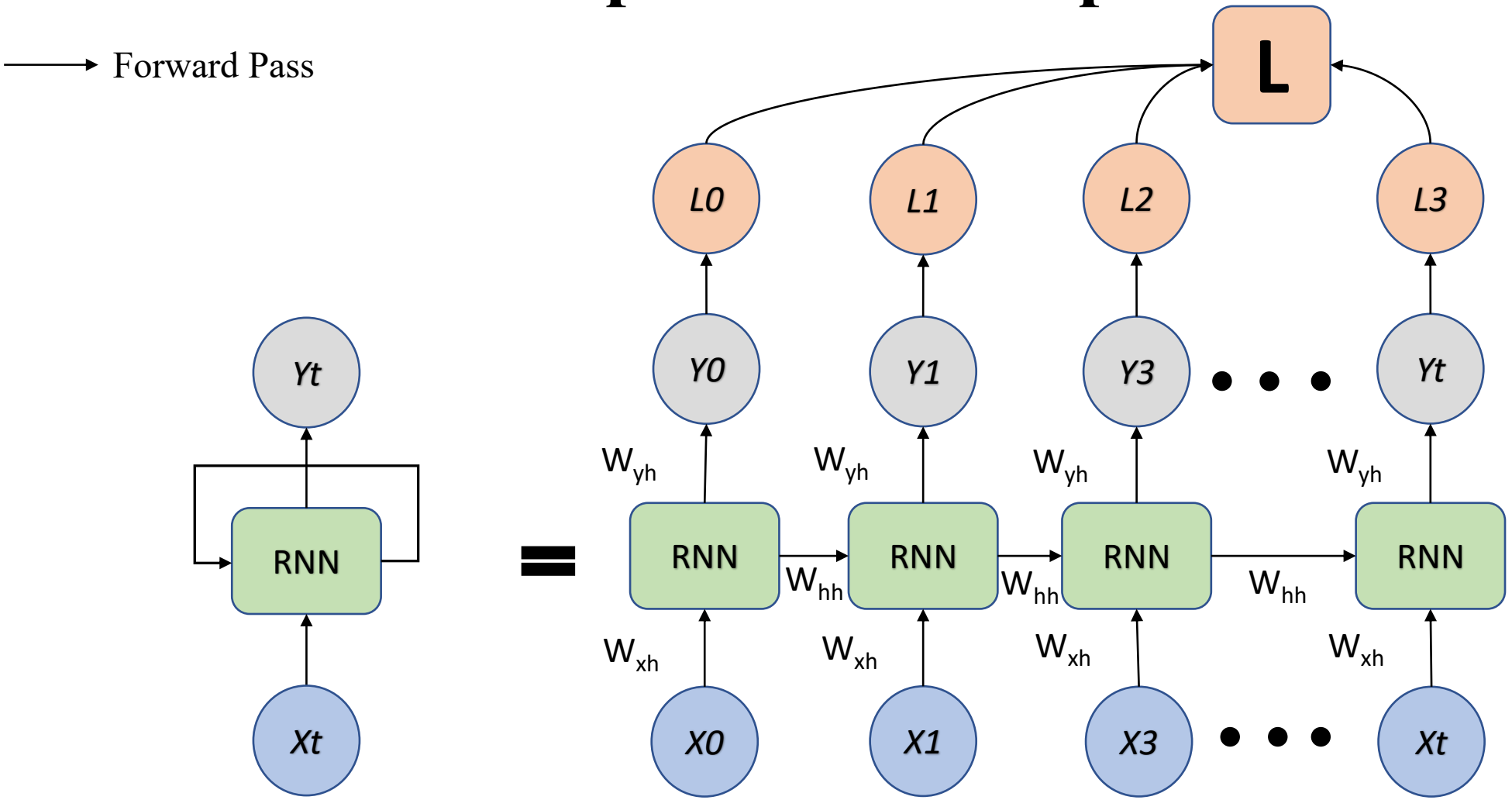
W_{hh} = Weight matrices that is used to update hidden state

W_{xh} = Weight matrices connecting the input to hidden state layer



[Source: https://tinyurl.com/3pawnch9]

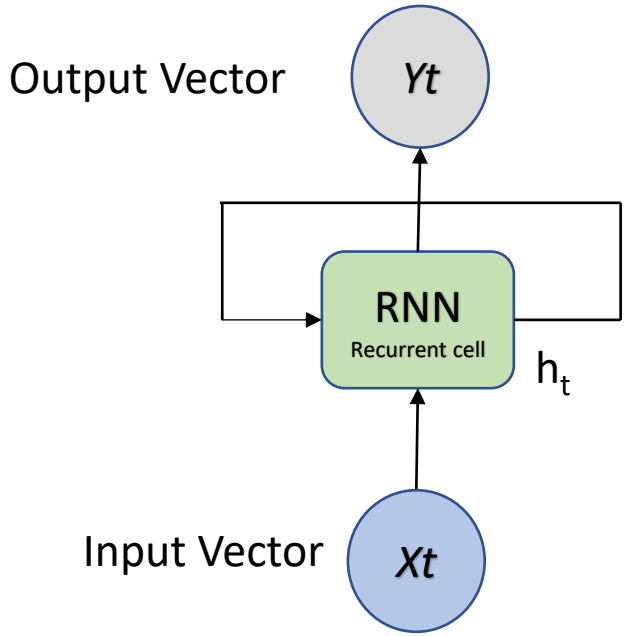
RNNs: Computational Graph Across Time:



[Source: https://tinyurl.com/3pawnch9]

RNNs from Scratch

```python
class MyRNNCell(tf.keras.layers.Layer):
  def __init__(self, rnn_units, input_dim, output_dim):
    super(MyRNNCell, self).__init__()

    # Initialize weight matrices
    self.W_xh = self.add_weight([rnn_units, input_dim])
    self.W_hh = self.add_weight([rnn_units, rnn_units])
    self.W_hy = self.add_weight([output_dim, rnn_units])

    # Initialize hidden state to zeros
    self.h = tf.zeros([rnn_units, 1])

  def call(self, x):
    # Update the hidden state
    self.h = tf.math.tanh( self.W_hh * self.h + self.W_xh * x )

    # Compute the output
    output = self.W_hy * self.h

    # Return the current output and hidden state
    return output, self.h
```



[Source: https://tinyurl.com/3pawnch9]

RNN Implementation in TensorFlow:

tf.keras.layers.SimpleRNN(rnn_units)

RNN Example : Predict Next Word in Sequence

1] Import the libraries and to train our model we need to add some data or Source text.To encode text to an array of numbers we use Tokenizer from keras: This class allows to vectorize a text corpus, by turning each text into either a sequence of integers (each integer being the index of a token in a dictionary) or into a vector where the coefficient for each token could be binary, based on word count, based on tf-idf.

```
In [3]: import numpy as np
        from numpy import array
        from keras.preprocessing.text import Tokenizer
        from keras.utils import to_categorical
        from keras.models import Sequent
        from keras.layers import Dense
        from keras.layers import LSTM
        from keras.layers import Embedding

In [21]: # source text
         data = """ I love recurrent nueral network\n
         and convolution nueral network"""

         # integer encode text
         tokenizer = Tokenizer()
         tokenizer.fit_on_texts([data])
         encoded_data= tokenizer.texts_to_sequences([data])[0]
         encoded_data

Out[21]: [3, 4, 5, 1, 2, 6, 7, 1, 2]
```

GitHub link: https://github.com/KhushbuKathrotiya/RNN_Next_Sequence_Word

Predict Sequence Next Word

2] Creating encoded sequences to get the data of specified length and the vocabulary. We use one-hot encoding to map output where the number of classes are number of given words in the vocabulary.

```
In [24]: # create word -> word sequences
         sequences = list()
         for i in range(1, len(encoded_data)):
         ----sequence = encoded_data[i-1:i+1]
         ----sequences.append(sequence)
         print('Total Sequences: %d' % len(sequences))
         # split into X and y elements

         Total Sequences: 8

In [25]: sequences
         #sequences[:5] # [input, output]

Out[25]: [[3, 4], [4, 5], [5, 1], [1, 2], [2, 6], [6, 7], [7, 1], [1, 2]]

In [26]: sequences = array(sequences)
         X, y = sequences[:,0],sequences[:,1]

In [27]: X[:5]

Out[27]: array([3, 4, 5, 1, 2])

In [28]: y[:5]

Out[28]: array([4, 5, 1, 2, 6])

In [29]: # one hot encode outputs
         y = to_categorical(y, num_classes=vocab_size)
         # define model
         y[:5]

Out[29]: array([[0., 0., 0., 0., 1., 0., 0., 0.],
                [0., 0., 0., 0., 0., 1., 0., 0.],
                [0., 1., 0., 0., 0., 0., 0., 0.],
                [0., 0., 1., 0., 0., 0., 0., 0.],
                [0., 0., 0., 0., 0., 0., 1., 0.]], dtype=float32)
```

GitHub link: https://github.com/KhushbuKathrotiya/RNN_Next_Sequence_Word

Predict Sequence Next Word

3] Creating embeddings and LSTM model and training model using categorical crossentropy as loss function, where accuracy is used as an evaluation metric.

```
In [30]: model = Sequential()
         model.add(Embedding(vocab_size, 10, input_length=1))
         model.add(LSTM(50))
         model.add(Dense(vocab_size, activation='softmax'))
         print(model.summary())
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 1, 10)             80

 lstm_1 (LSTM)               (None, 50)                12200

 dense_1 (Dense)             (None, 8)                 408

=================================================================
Total params: 12688 (49.56 KB)
Trainable params: 12688 (49.56 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

```
In [31]: # compile network
         model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
In [32]: # fit network
         model.fit(X, y, epochs=100)
```

```
Epoch 92/100
1/1 [==============================] - 0s 2ms/step - loss: 1.6687 - accuracy: 0.8750
Epoch 93/100
1/1 [==============================] - 0s 2ms/step - loss: 1.6578 - accuracy: 0.8750
Epoch 94/100
1/1 [==============================] - 0s 2ms/step - loss: 1.6467 - accuracy: 0.8750
Epoch 95/100
1/1 [==============================] - 0s 2ms/step - loss: 1.6355 - accuracy: 0.8750
Epoch 96/100
```

GitHub link: https://github.com/KhushbuKathrotiya/RNN_Next_Sequence_Word

University of Windsor

Predict Sequence Next Word

4] Inference code is given in below figure: Once the model is trained, the below function is used to predict the next word. First, create encoded sequences that are from words to array of numbers which is then fed to the trained model. Once the word with maximum probability is obtained, it is mapped back to the word from the given vocabulary.

In [33]:
```
# generate a sequence from the model
def generate_seq(model, tokenizer, enter_text, n_pred):
    in_text, result = enter_text, enter_text  #
    # generate a fixed number of words
    for _ in range(n_pred):

        # encode the text as integer
        encoded = tokenizer.texts_to_sequences([in_text])[0]
        encoded = array(encoded)

        # predict a word in the vocabulary
        yhat = model.predict(encoded)
        pc = np.argmax(yhat, axis=1)

        # map predicted word index to word
        out_word = ''
        for word, index in tokenizer.word_index.items():
            if index == pc:
                out_word = word
                break
        # append to input
        in_text, result = out_word, result + ' ' + out_word
    return result
```

In [38]:
```
# evaluate
print(generate_seq(model, tokenizer, 'convolution', 2))
```

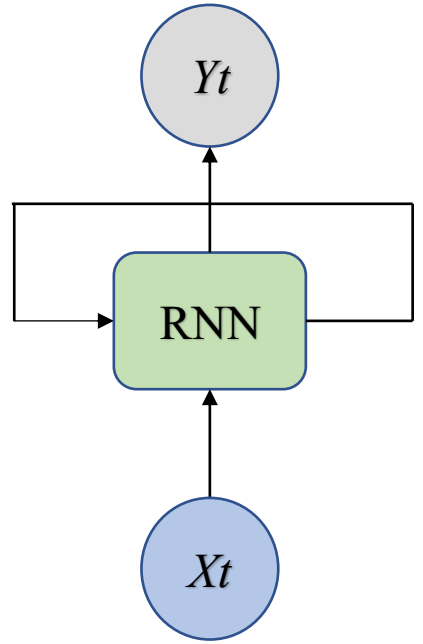
```
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 8ms/step
convolution nueral network
```

GitHub link: https://github.com/KhushbuKathrotiya/RNN_Next_Sequence_Word

Sequence Modeling: Design Criteria

To model sequences, we need to:

- Handle variable length sequence
- Track long-term dependencies
- Maintain information about order
- Share parameters across the sequence



[Source: https://tinyurl.com/3pawnch9]

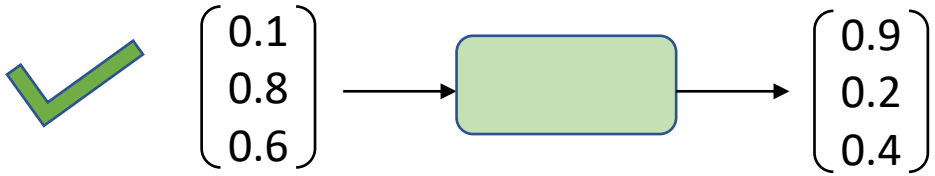
Recurrent Neural Networks(RNNs) meets these sequence modeling design criteria

A Sequence Modeling Problem: Predict the Next Word

"This morning I took my cat for a walk"

Representing language to a Neural Network

Neural networks cannot interpret words

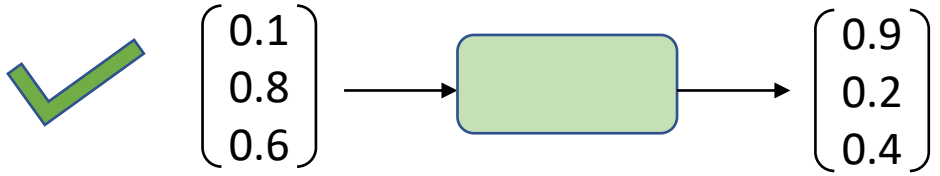


Neural networks require numerical inputs

[Source: https://tinyurl.com/3pawnch9]

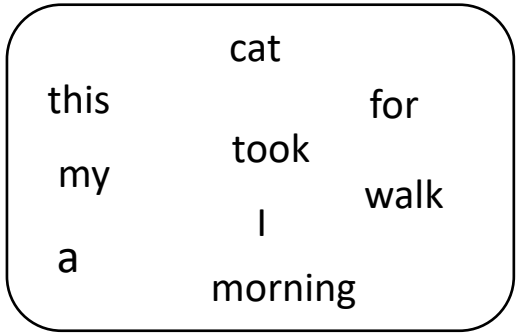
Encoding Language for a Neural Network

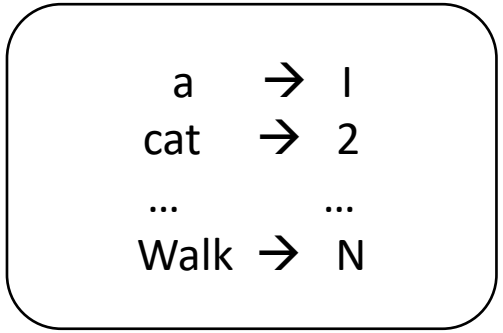
Neural networks cannot interpret words

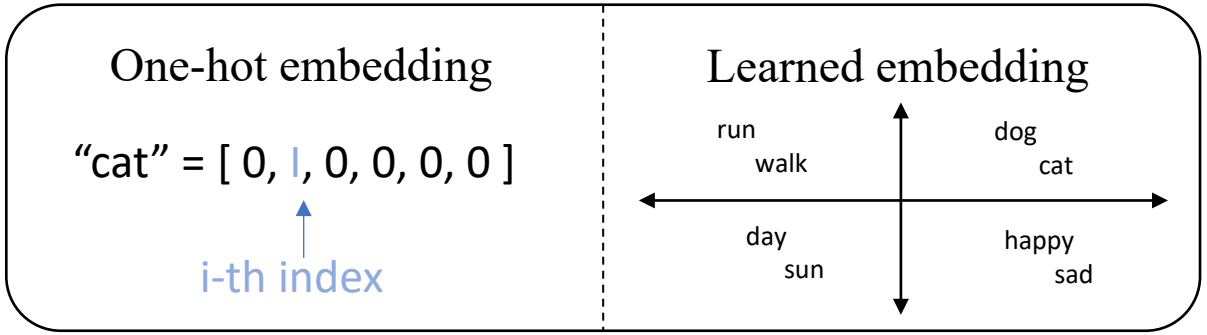


Neural networks require numerical inputs

Embedding: transform indexes into a vector of fixed size.







1. Vocabulary:
Corpus of words

2. Iendexing:
Word to index

3. Embedding:
Index to fixed-sized vector

[Source: https://tinyurl.com/3pawnch9]

Handle Variable Sequence Lengths

The food was great

vs.

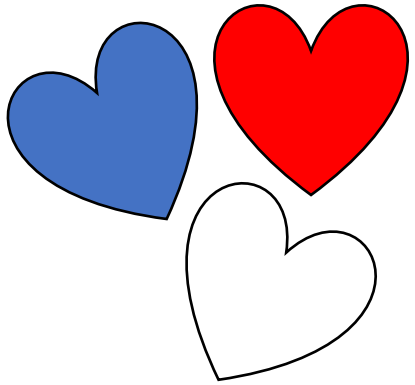
We visited a restaurant for lunch

vs.

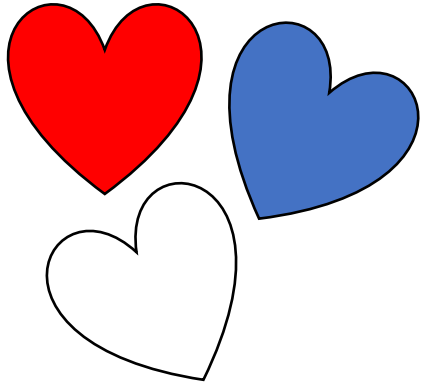
We were hungry but cleaned the house before eating

Model Long-Term Dependencies

"**France** is where I grew up, but I now live in Boston. I speak fluent ___."



J'aime 6.SI9I!



We need information from the distant past to accurately predict the correct word.

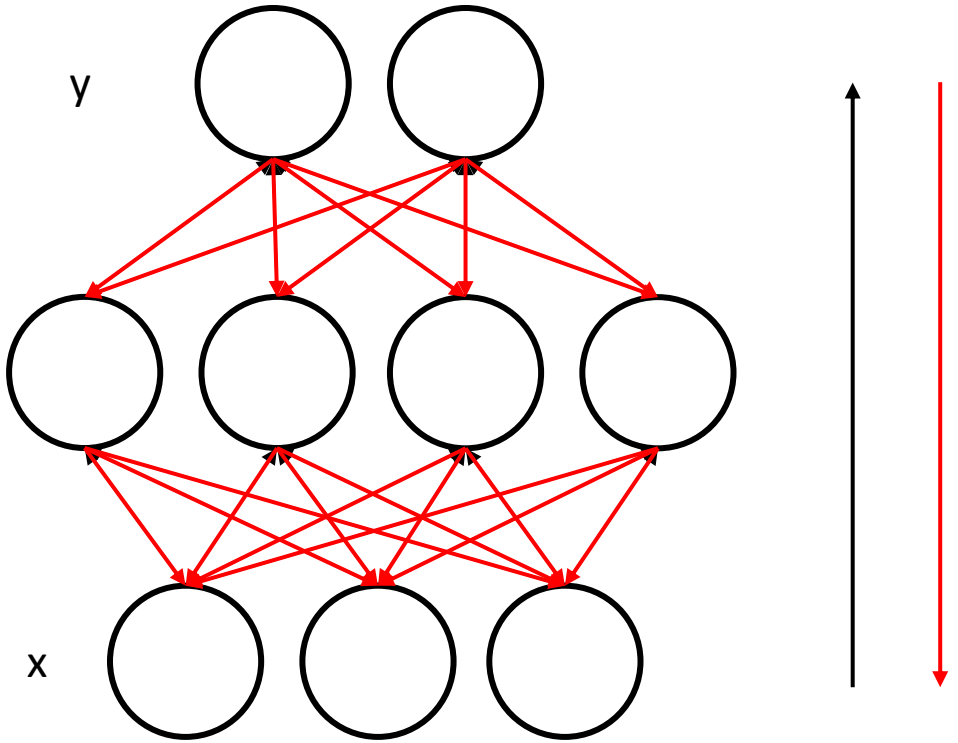
Capture Differences in Sequence Order

The food was good, not bad at all.

vs.

The food was bad, not good at all.

Recall: Backpropagation in Feed Forward Models

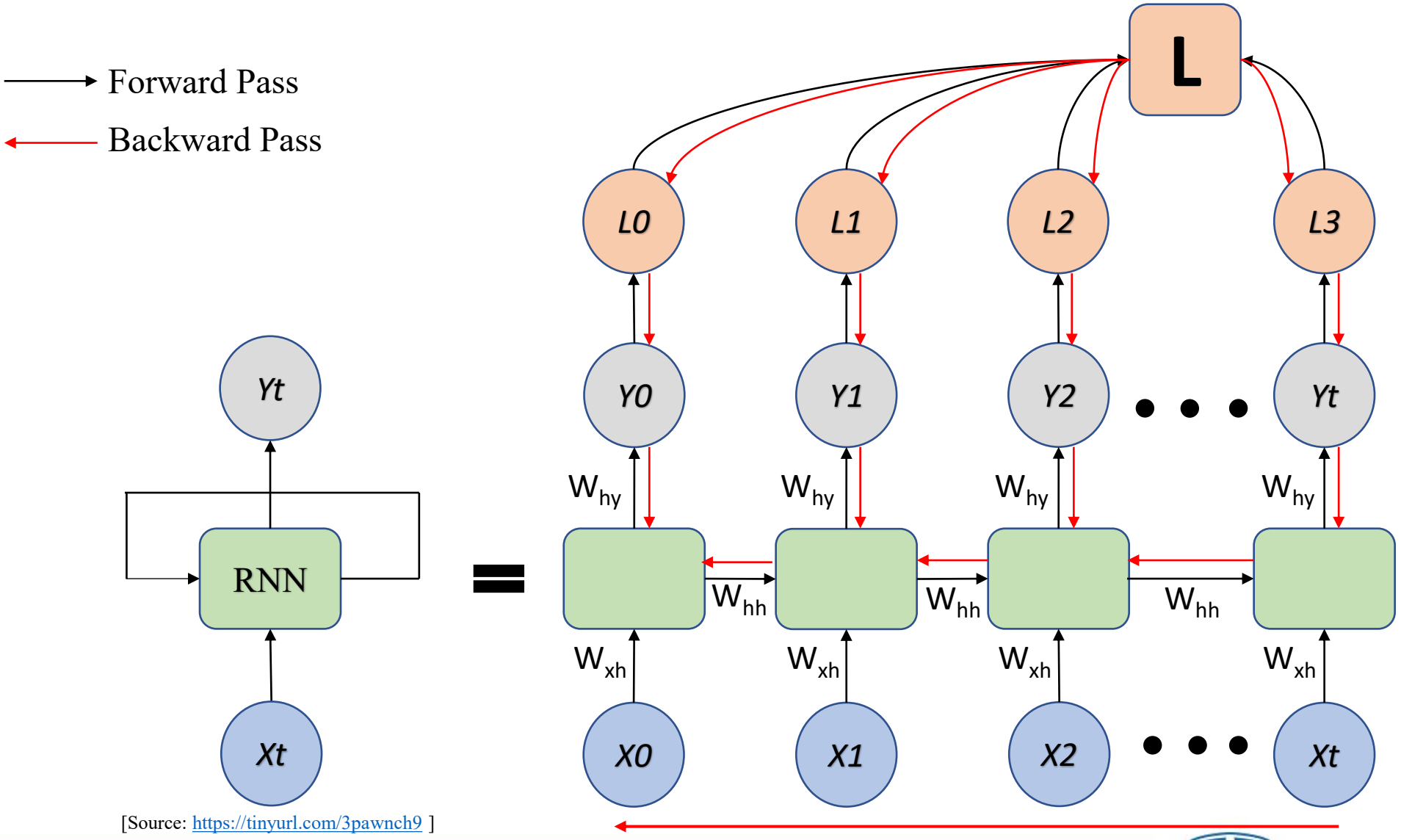


[Source: https://tinyurl.com/3pawnch9]

Backpropagation algorithm:

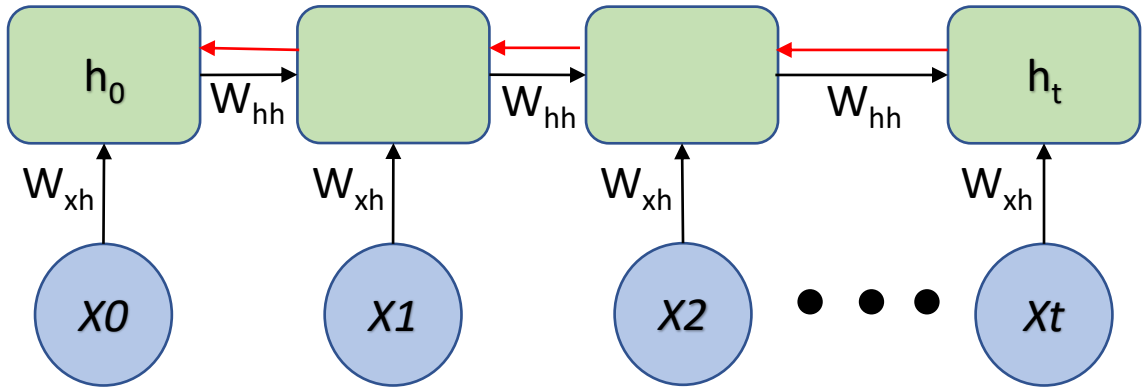
1. Take the derivative(gradient) of the loss with respect to each parameter.
2. Shift parameters in order to minimize loss

RNNs: Backpropagation Through Time



[Source: https://tinyurl.com/3pawnch9]

Standard RNN Gradient Flow



[Source: https://tinyurl.com/3pawnch9]

Computing the gradient w.r.t h_0 involves many factors of W_{hh} + repeated gradient computation!

Many values > 1
exploding gradients

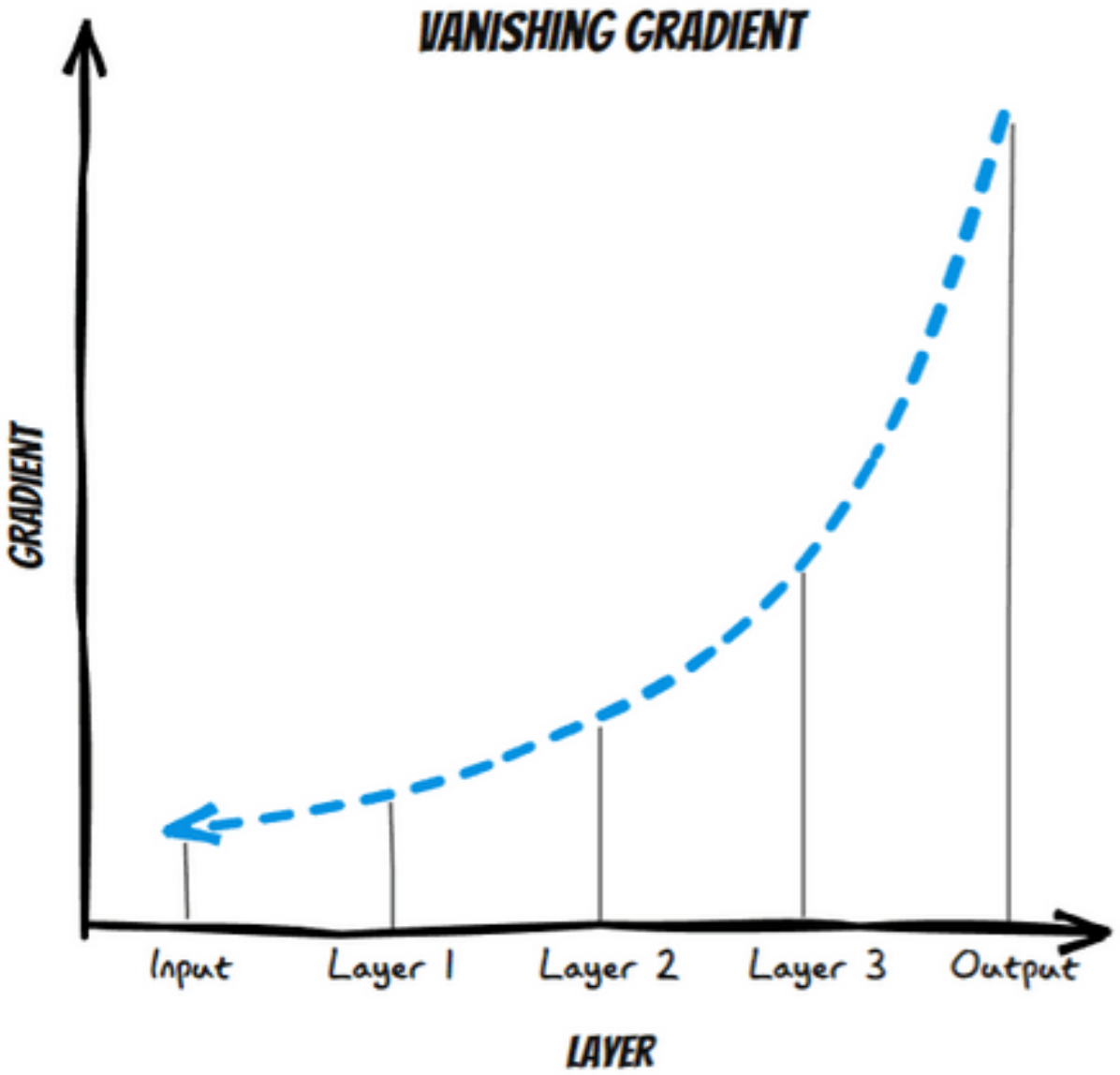
Gradient clipping to
scale big gradients

Many values < 1:
vanishing gradients

1. Activation function
2. Weight initialization
3. Network architecture

Problem of long-term dependencies

- Recurrent Neural Network (RNN) is used for modeling sequential information and is a state-of-the-art deep learning algorithm.
- Recurrent neural networks struggle to learn dependencies between inputs that are separated by many time steps.
- As information passes through the RNN, the signal gets diluted over time, making it challenging to capture long-term dependencies.
- This limitation is known as the vanishing gradient problem and hampers the ability of vanilla RNNs to model sequences with long-range temporal structures.
- Capturing long-term dependencies in sequential data is a basic challenge in many fields, such as NLP, speech recognition, and time series analysis. [5]



[Source: https://tinyurl.com/y4x357kk]

Introduction to LSTMs

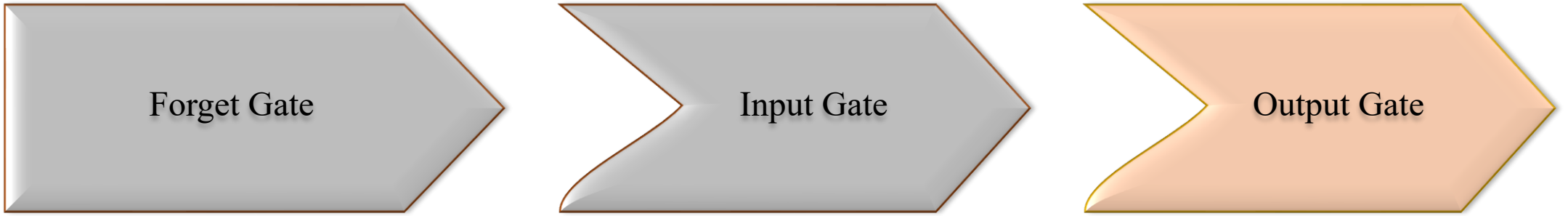
- LSTMs (Long-Short Term Memory) are a type of recurrent neural network that contains memory cells, allowing them to remember values over long periods of time.

- They have gates that regulate the flow of information, allowing them to selectively remember or forget past information.[6]

Short video explaining LSTM

[Source: https://www.youtube.com/watch?v=5dMXyiWddYs&ab_channel=SkillC]

LSTM Architecture

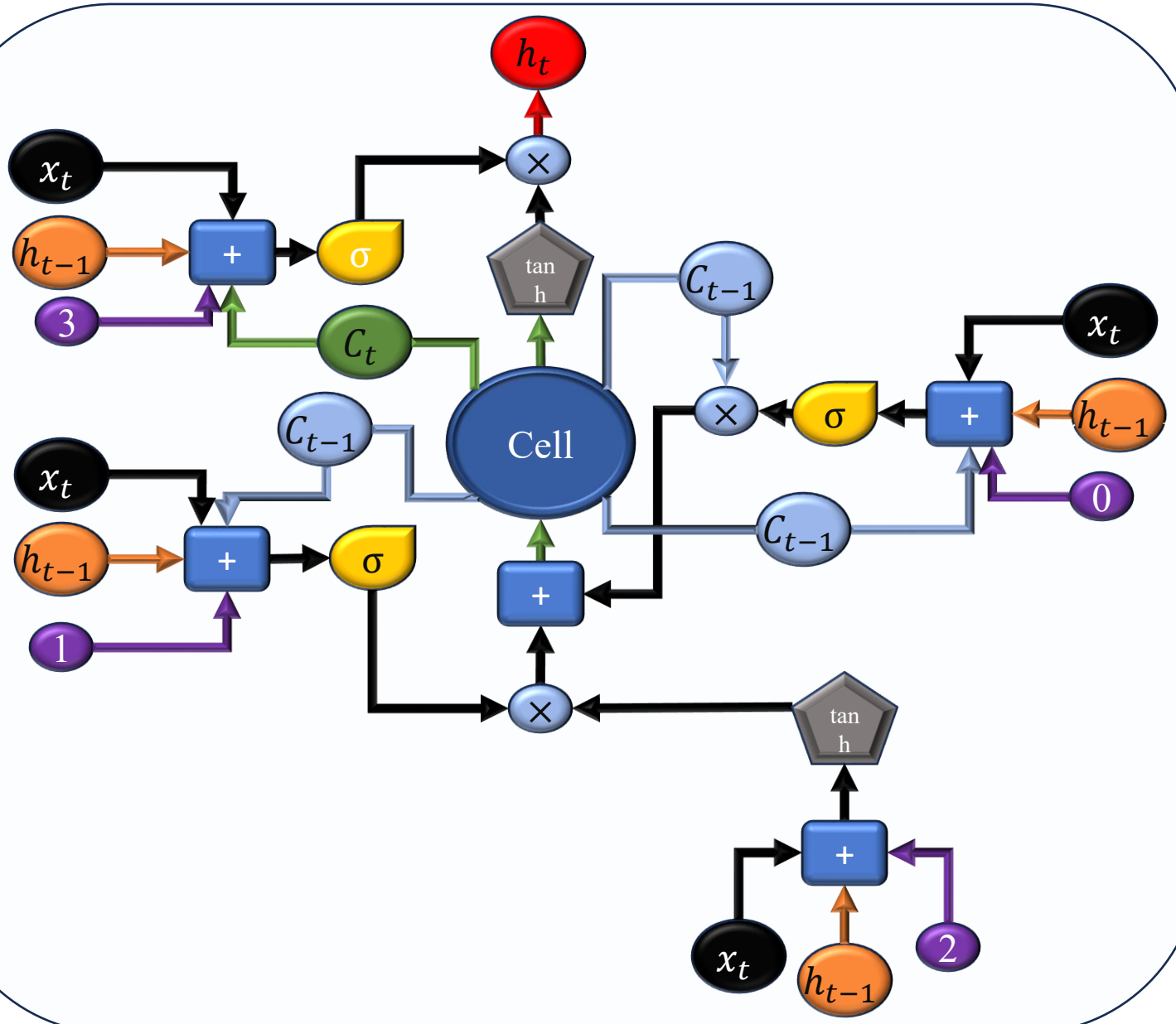


- Decides what information to remove from the cell state based on the previous hidden state and current input. Outputs number between 0-1 indicating importance.

- Decide what new information to add to the cell state.
- Create new candidate values to add.

- Decides what parts of the cell state to output. Scales cell state between -1 and 1.

[Source: https://tinyurl.com/mrycxxp5]



[Source: https://tinyurl.com/mrycxxp5]

Inputs:

x_t Input Vector

h_{t-1} Output of previous block

C_{t-1} Memory from previous block

Non-linearities:

tan h Hyperbolic tangent

σ Sigmoid

0 Bias

Outputs:

h_t Output of current block

C_t Memory from current block

Vector operations:

\+ Element-wise summation/Concatenation

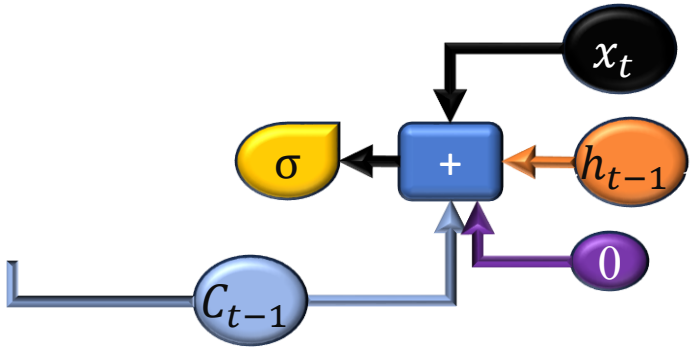
× Element-wise multiplication

Forget Gate

Decides Information to Discard	Inputs	Forgetting	Optimization	Cell State
• Uses a sigmoid layer to output values between 0 and 1 indicating which parts of the cell state to discard.	• Takes the hidden state (ht-1) from the previous cell and the current input (xt).	• Values close to 0 means discard that part of the cell state; values close to 1 mean keep that part.	• Forgetting irrelevant information optimizes the LSTM network's performance.	• Multiplies output of sigmoid layer with cell state to selectively forget information.

[Source: https://tinyurl.com/mrycxxp5]

This is the forget gate that shuts the irrelevant memory.



[Source: https://tinyurl.com/mrycxxp5]

Inputs:

x_t Input Vector

h_{t-1} Output of previous block

C_{t-1} Memory from previous block

Non-linearities:

tanh Hyperbolic tangent

σ Sigmoid

0 Bias

Outputs:

h_t Output of current block

C_t Memory from current block

Vector operations:

\+ Element-wise summation/Concatenation

× Element-wise multiplication

Input Gate

Introduction

- The input gate regulates what new information to add to the cell state. It decides which values should be updated in the cell state.

Optimizes Performance

- By preventing redundant information, the input gate optimizes the performance of the LSTM network.

Working

The input gate works in three steps:

1. A sigmoid layer filters candidate values.
2. A tanh layer creates candidate values.
3. The sigmoid output is multiplied by the tanh output to get the updates.

Provides Control

- The input gate provides precise control over what enters the cell state.

Role

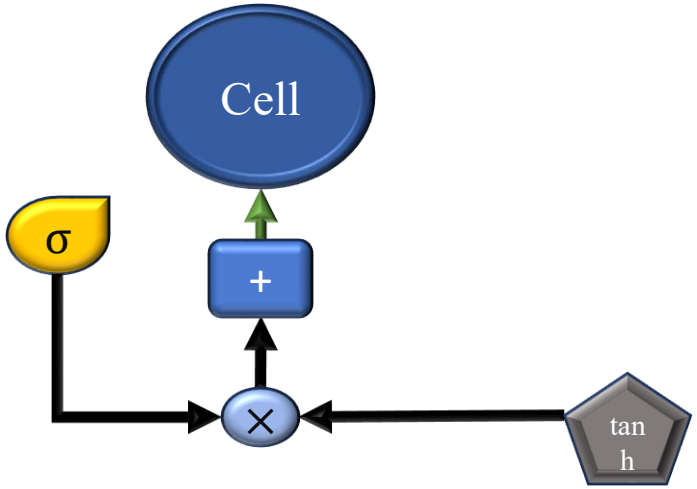
- The input gate helps the LSTM selectively add only relevant new information to the cell state.

Key Component

- The input gate is a key component of the LSTM architecture.

[Source: https://tinyurl.com/mrycxxp5]

The sigmoid output is multiplied by the tanh output to get the updates



[Source: https://tinyurl.com/mrycxxp5]

Inputs:

x_t Input Vector

h_{t-1} Output of previous block

C_{t-1} Memory from previous block

Non-linearities:

tan h Hyperbolic tangent

σ Sigmoid

0 Bias

Outputs:

h_t Output of current block

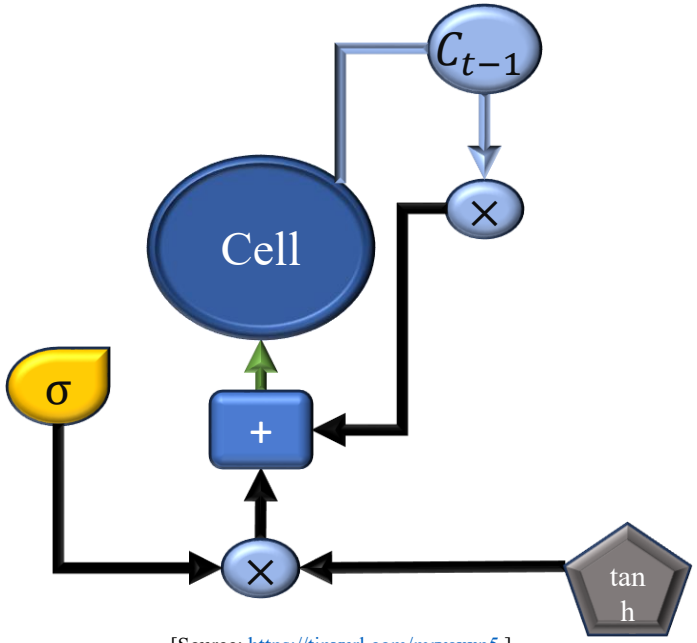
C_t Memory from current block

Vector operations:

\+ Element-wise summation/Concatenation

× Element-wise multiplication

The two gates here are responsible for actively merging the old memory and the new memory to create Ct (green arrow), which is subsequently fed back into the main "Cell."



[Source: https://tinyurl.com/mrycxxp5]

Inputs:

- x_t Input Vector
- h_{t-1} Output of previous block
- C_{t-1} Memory from previous block

Non-linearities:

- tan h Hyperbolic tangent
- σ Sigmoid
- 0 Bias

Outputs:

- h_t Output of current block
- C_t Memory from current block

Vector operations:

- \+ Element-wise summation/Concatenation
- × Element-wise multiplication

Output Gate

Sigmoid Layer

- Outputs numbers between 0 and 1 to select which parts of the cell state to keep.

tanh Layer

- Transforms the cell state into the -1 to 1 range.

Vector Multiplication

- The sigmoid and tanh vectors are multiplied to yield the relevant parts of the cell state.

Pass to Hidden State

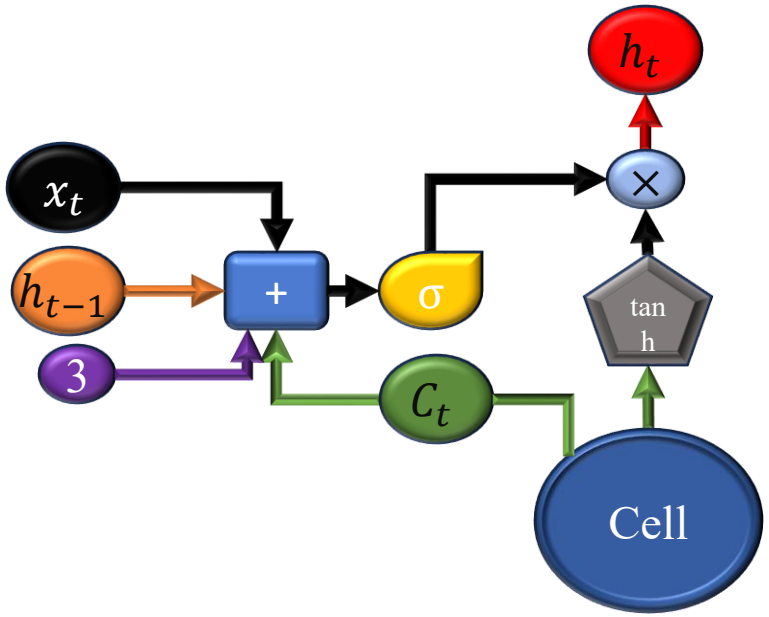
- The output of the vector multiplication becomes the next hidden state.

Select Relevant Information

- The output gate selects the relevant parts of the cell state to pass to the hidden state.

[Source: https://tinyurl.com/mrycxxp5]

Output gate and output unit of LSTM



[Source: https://tinyurl.com/mrycxxp5]

Inputs:

x_t Input Vector

h_{t-1} Output of previous block

C_{t-1} Memory from previous block

Non-linearities:

tan h Hyperbolic tangent

σ Sigmoid

0 Bias

Outputs:

h_t Output of current block

C_t Memory from current block

Vector operations:

\+ Element-wise summation/Concatenation

× Element-wise multiplication

RNN Advantages and Disadvantages

Advantages

- Possibility of processing input of any length
- Model size not increasing with the size of the input
- Computation considers historical information
- Weights are shared across time[7]

Disadvantages

- Computation is slow
- Difficulty accessing information from a long time ago
- Cannot consider any future input for the current state[7]

YouTube Video links:

1. https://youtu.be/ySEx_Bqxvvo?si=jHxqk5DrS8hoTnh7
2. https://youtu.be/NmLK_WQBxB4?si=kvUWczk7PkJ_xFlc
3. https://youtu.be/JgnbwKnHMZQ?si=hsVA6WVX0l37-Hdu
4. https://www.youtube.com/watch?v=5dMXyiWddYs&ab_channel=SkillC

University of Windsor

References

[1] “What is a convolutional neural network?: 3 things you need to know,” What Is a Convolutional Neural Network? | 3 things you need to know - MATLAB & Simulink, https://tinyurl.com/pk7jrwnm (accessed Oct. 27, 2023).

[2] Group, “Recurrent Neural Network and Long Term Dependencies,” Medium, Mar. 09, 2020. https://tinyurl.com/m9cxvuj (accessed Oct. 18, 2023).

[3] L. Craig, “CNN vs. RNN: How are they different?: TechTarget,” Enterprise AI, https://tinyurl.com/3ukwrsjy (accessed Oct. 27, 2023).

[4] MIT Deep Learning 6.S191, https://tinyurl.com/3pawnch9 (accessed Oct. 1, 2023).

[5] Deep learning, https://www.deeplearningbook.org/contents/rnn.html (accessed Oct. 13, 2023).

[6] S. Yan, “Understanding LSTM and its diagrams,” Medium, https://tinyurl.com/mrycxxp5 (accessed Oct. 27, 2023).

[7] “Recurrent neural networks cheatsheet star,” CS 230 - Recurrent Neural Networks Cheatsheet, https://tinyurl.com/yc5t3xy3 (accessed Oct. 27, 2023).