Interactive Technologies Institute
LARSyS

arditi agência regional para o desenvolvimento da investigação tecnologia e inovação

UNIVERSIDADE da MADEIRA

MADEIRA INTERNATIONAL WORKSHOP IN MACHINE LEARNING

IKnowD

startup madeira

Premium sponsor:

asseco PST

Gold sponsor:

MadManTech

Bronze sponsor:

CÂMARA DE LOBOS

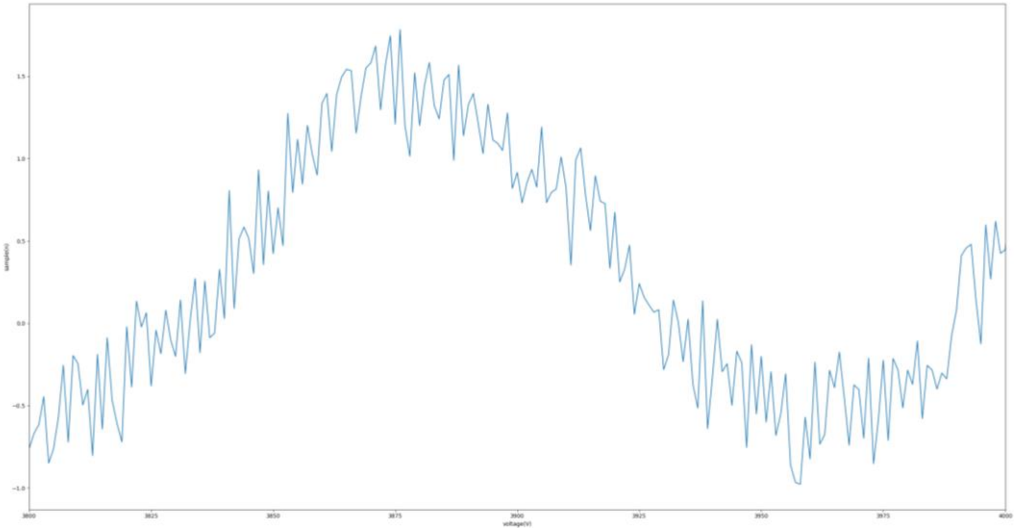
MEO

2022

MOTIVATION

MOTIVATION

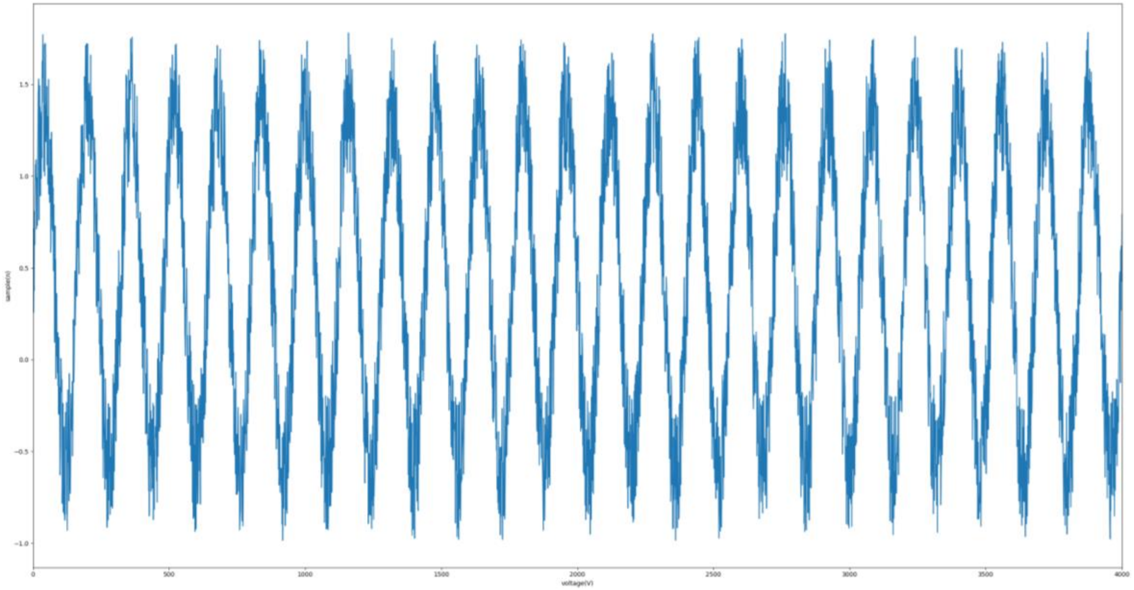
Where is the information?

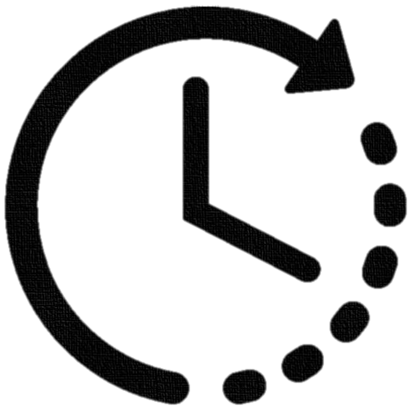


- Variation of the peeks amplitude?
- Frequency of the oscillations?
- Crossings of the trend line?

MOTIVATION

Where is the information?





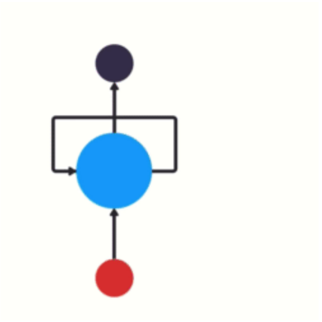
MOTIVATION

Humans don’t start their thinking from scratch every second

- You don’t throw everything away and start thinking from scratch
- Your thoughts have persistence
- You make use of context and previous knowledge to understand what is coming next

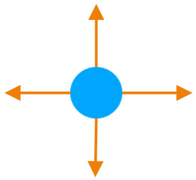
Recurrent Neural Networks (RNN) address this issue

- They are networks with loops, allowing information to persist



MOTIVATION

Predict the direction of a moving ball:



How would you do this by checking only the ball?

- Every guess is purely random without knowledge of where the ball has been
- You don't have enough data to predict where it's going

Record snapshots of the ball's position in succession

- you will have enough information to make a better prediction

MOTIVATION

RNN is good for processing a sequence data for predictions, but how?

- Make us of the sequence memory

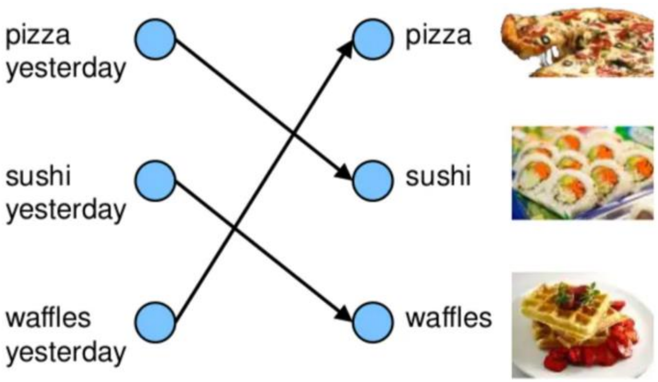
Try to say the alphabet in your head from A to Z

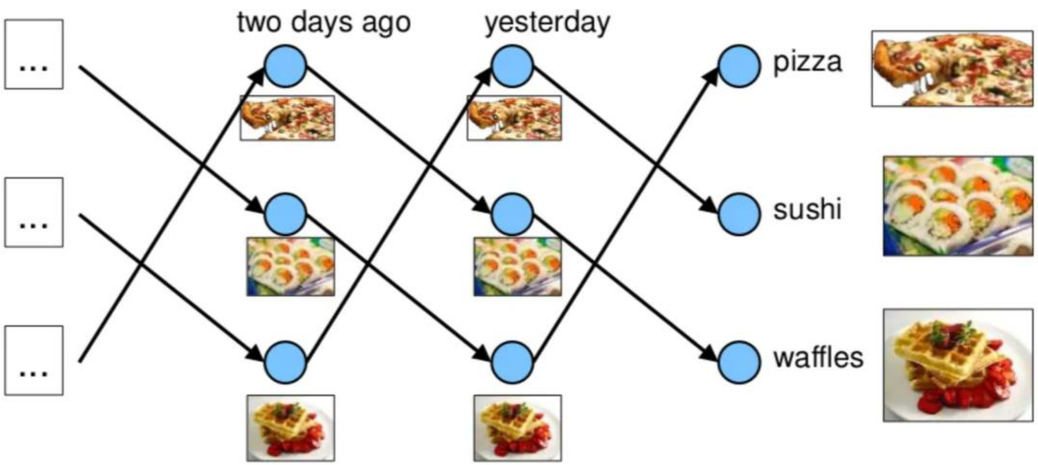
Now try to say from Z to A

- This can be difficult as you learn the alphabet as a sequence and your brain recognizes the sequential patterns

MOTIVATION

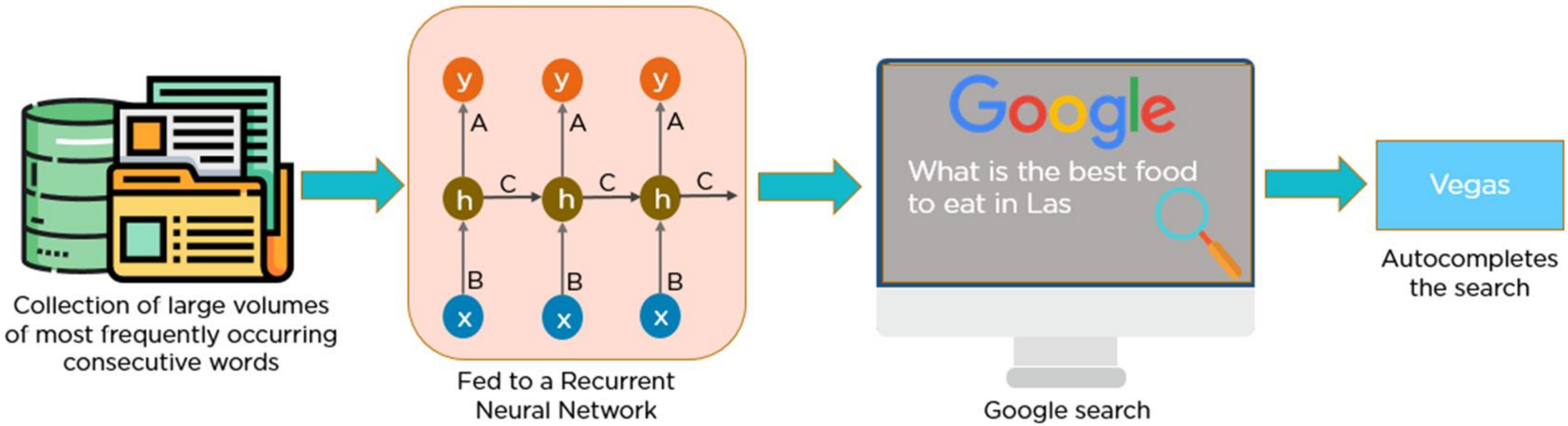
Lunch forecast:





MOTIVATION

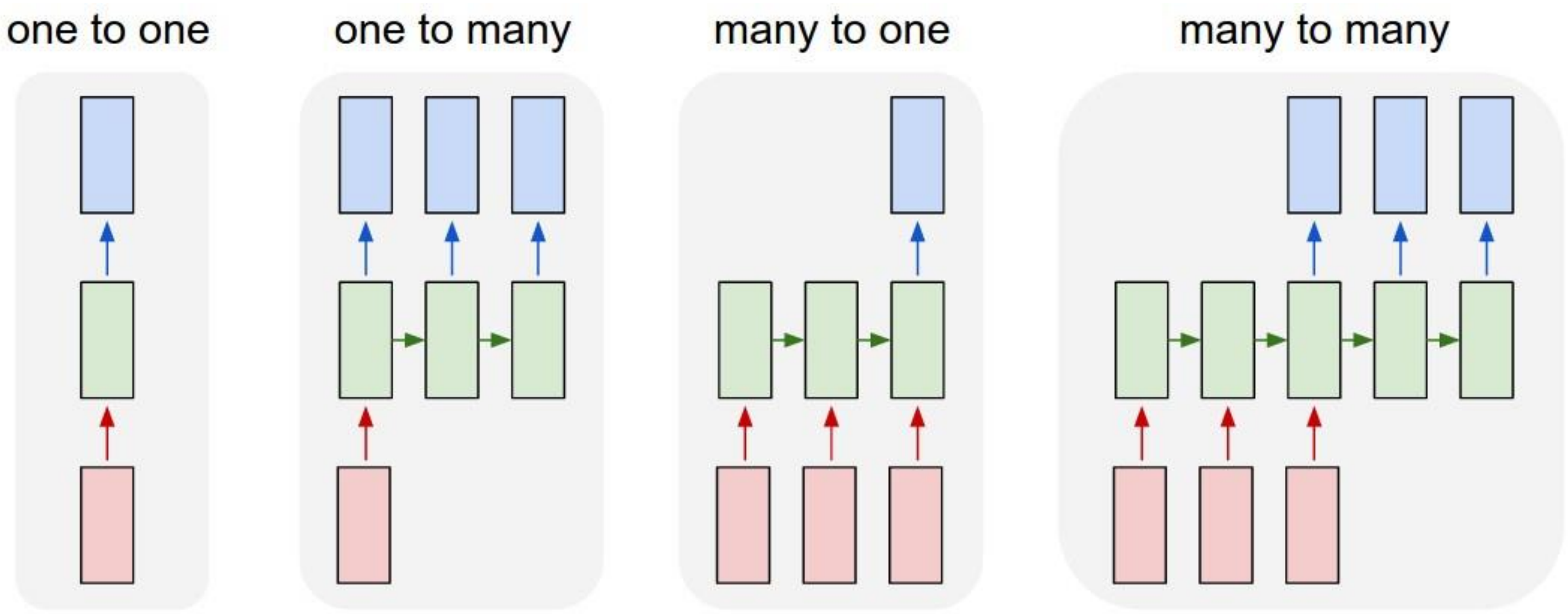
How autocomplete features predicts the rest of a sentence without the user typing?



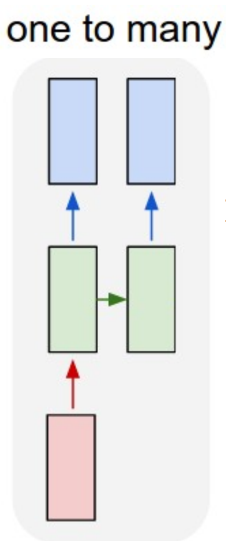
Analyses the data by finding the sequence of words occurring
frequently and builds a model to predict the next word in the sentence

MOTIVATION

Usual **sequence prediction problems** for Recurrent Neural Networks (RNN)



MOTIVATION



Example: Input is a sequence of numbers, while the output is the sequence of the next two numbers

1, 4, 7, 10, 13, 16, 19, 22, 25, 28, 31, 34, 37, 40, 43] ← Input

[2, 3], [5, 6], [8, 9], [11, 12], [14, 15], [17, 18], [20, 21], [23, 24], [26, 27], [29, 30], [32, 33], [35, 36], [38, 39], [41, 42], [44, 45]]

Output

`inputs`: A 3D tensor with shape `[batch, timesteps, feature]`

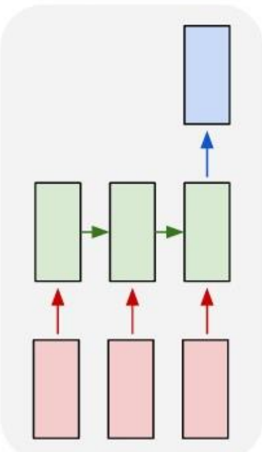
```
model = Sequential()
model.add(LSTM(50, activation='relu', input_shape=(1, 1)))
model.add(Dense(2))

model.compile(optimizer='adam', loss='mse')
model.fit(X, Y, epochs=1000, validation_split=0.2, batch_size=3, callbacks=[WandbCallback()])
```

```
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 50)                10400
_________________________________________________________________
dense (Dense)                (None, 2)                 102
=================================================================
Total params: 10,502
Trainable params: 10,502
Non-trainable params: 0
```

Predicting with an input value of 10, we expect the sequence [11, 12]
The model predicted the sequence [11.01, 12.14]

MOTIVATION

many to one



Example: Input has samples with three time steps, and the output is the sum of the values in each step

```
[[[ 1]     [[ 4]     [[ 7]
  [ 2]      [ 5]      [ 8]     . . .
  [ 3]] ,   [ 6]] ,   [ 9]] ,
```

Input

Output → [6 15 24 33 42 . . .

inputs: A 3D tensor with shape [batch, timesteps, feature]

```
model = Sequential()
model.add(LSTM(50, activation='relu', input_shape=(3, 1)))
model.add(Dense(1))

model.compile(optimizer='adam', loss='mse')
history = model.fit(X, Y, epochs=1000, validation_split=0.2, verbose=1, callbacks=[WandbCallback()])
```

Layer (type)	Output Shape	Param #
lstm_1 (LSTM)	(None, 50)	10400
dense_1 (Dense)	(None, 1)	51

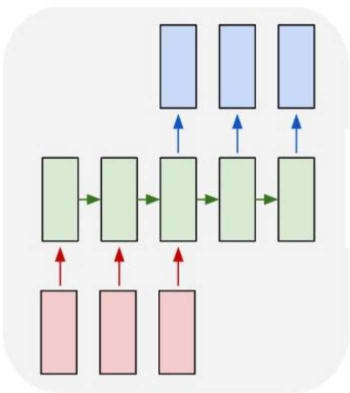
Total params: 10,451
Trainable params: 10,451
Non-trainable params: 0

Predicting with an input sequence of three time steps [50, 51, 52] we expect an output value of 153

The model predicted the value 152.93

MOTIVATION

many to many



Example: Input has samples with three time steps, and the output has the next three consecutive multiples of 5

```
[[[  5]     [[ 20]     [[ 35]
  [ 10]      [ 25]      [ 40]     . . .  <— Input
  [ 15]],    [ 30]],    [ 45]],
```

```
Output  —>  [[[ 20]     [[ 35]     [[ 50]
              [ 25]      [ 40]      [ 55]     . . .
              [ 30]],    [ 45]],    [ 60]],
```

inputs: A 3D tensor with shape [batch, timesteps, feature]

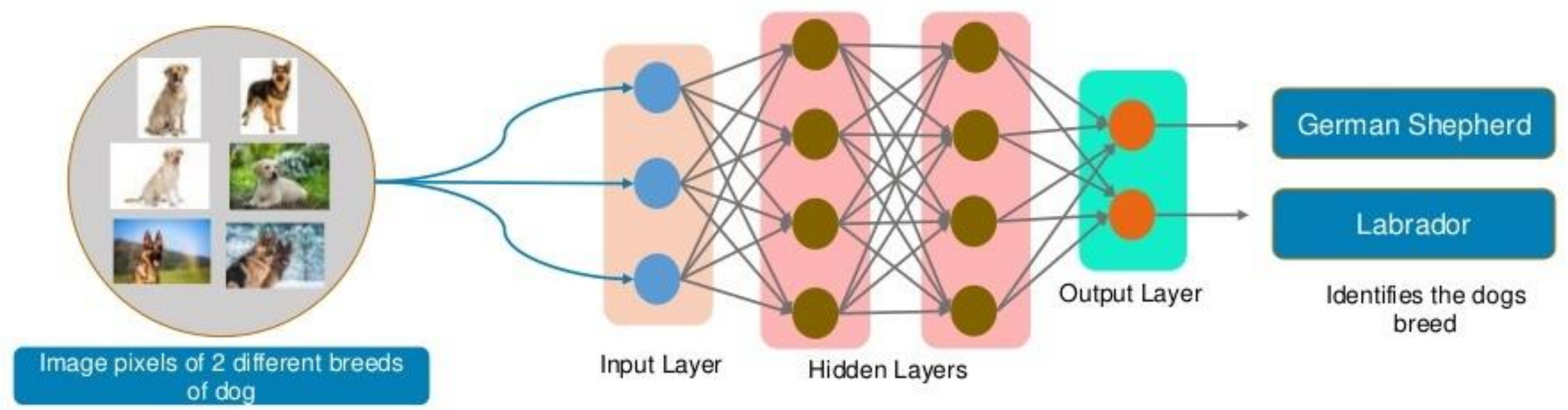
```
model = Sequential()
model.add(LSTM(50, activation='relu', input_shape=(3, 1)))
model.add(Dense(3))

model.compile(optimizer='adam', loss='mse')
model.fit(X, Y, epochs=1000, validation_split=0.2, batch_size=3, callbacks=[WandbCallback()])
```

```
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 50)                10400
_________________________________________________________________
dense (Dense)                (None, 2)                 102
=================================================================
Total params: 10,502
Trainable params: 10,502
Non-trainable params: 0
```

Predicting with an input a sequence of three time steps: [300, 305, 310] we expect an output sequence of [315, 320, 325]
The model predicted the sequence [315.30, 321.04, 327.00]

MOTIVATION

How can a neural network identify a dog's breed based on its features?

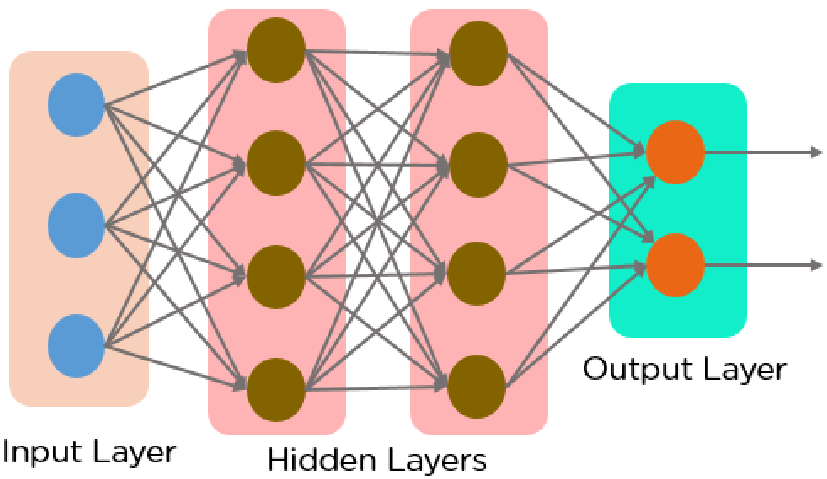


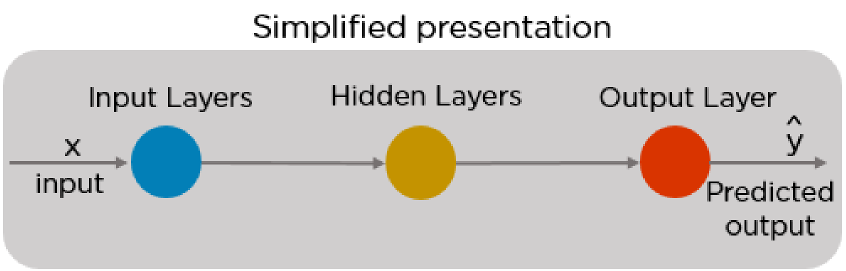
-Images of two different breeds of dogs are fed to the input layer
-The image pixels are then processed in the hidden layers for feature extraction
-The output layer produces the result to identify the breed

Such networks do not require memorizing the past output

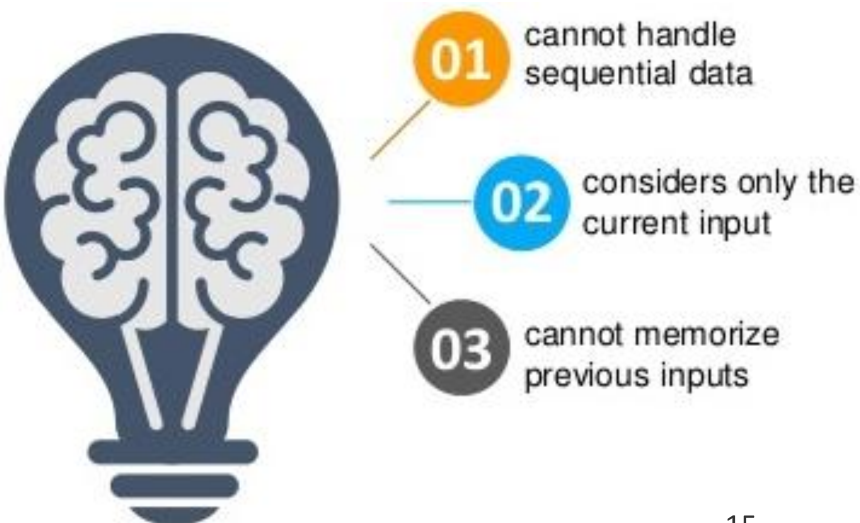
MOTIVATION

How can a neural network identify a dog's breed based on its features?





-Decisions are based on current input
-No memory about the past
-No future scope



WHAT ARE RECURRENT NEURAL NETWORKS

WHAT ARE RECURRENT NEURAL NETWORKS

The aim of RNNs is to detect dependencies in sequential data

- Find correlations between different points within a sequence

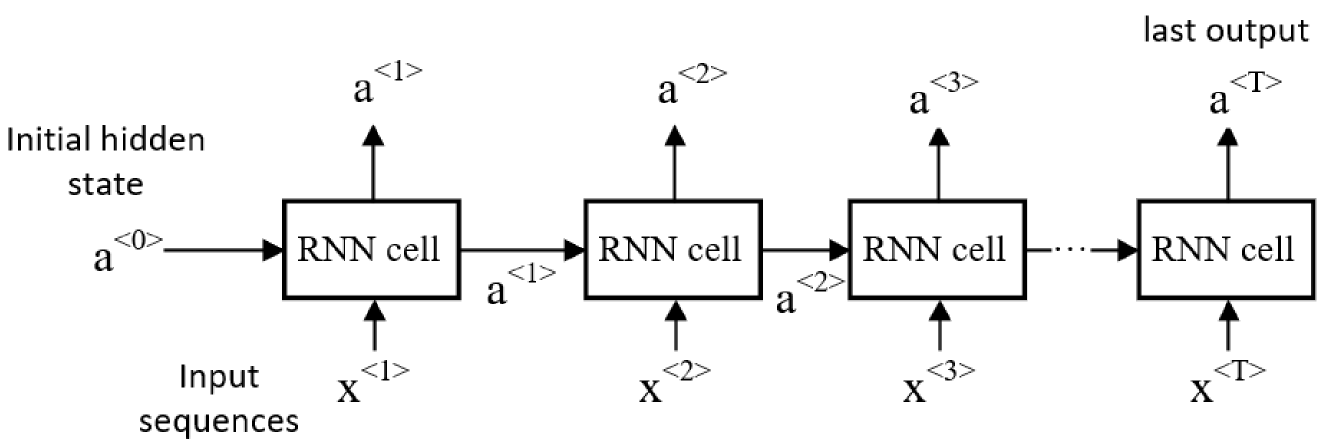
Two kinds of dependencies:

- Short-term dependencies are associated with the recent past
- Long-term dependencies are far away from each other in time

WHAT ARE RECURRENT NEURAL NETWORKS

Key terms:

- An **input** in a **sequence** is a **time step**
- The **number of time steps** defines the **sequence length**
- Every **time step** in the sequence has **associated** a **feature vector** as input with the **values we want to track**



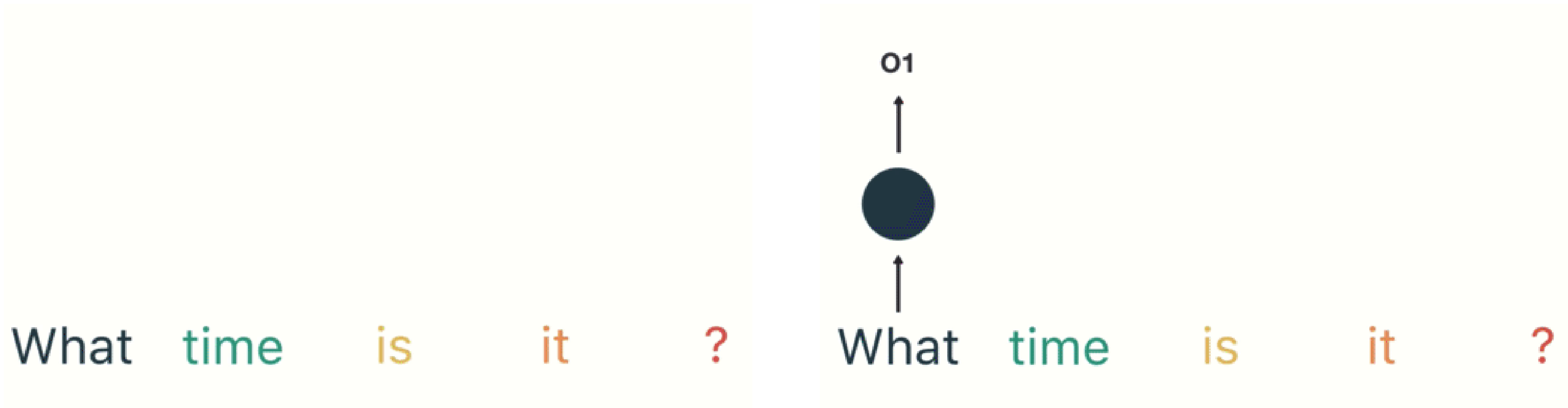
WHAT ARE RECURRENT NEURAL NETWORKS

Example: Classifying intents from users' inputs

What time is it?

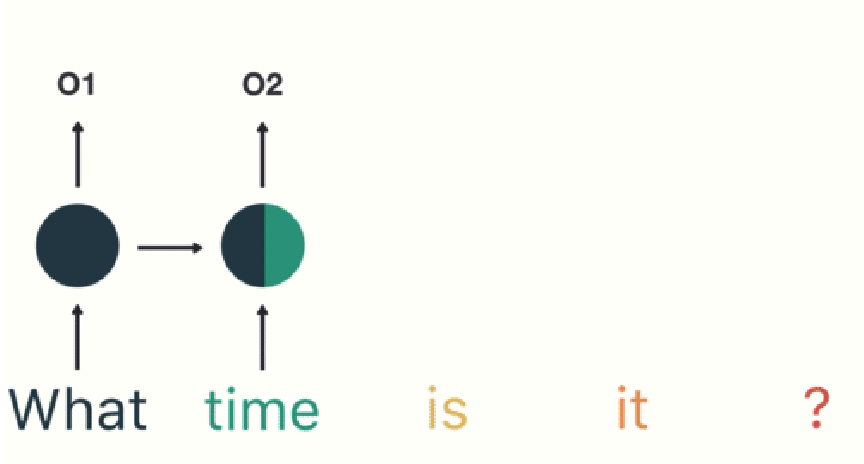
WHAT ARE RECURRENT NEURAL NETWORKS

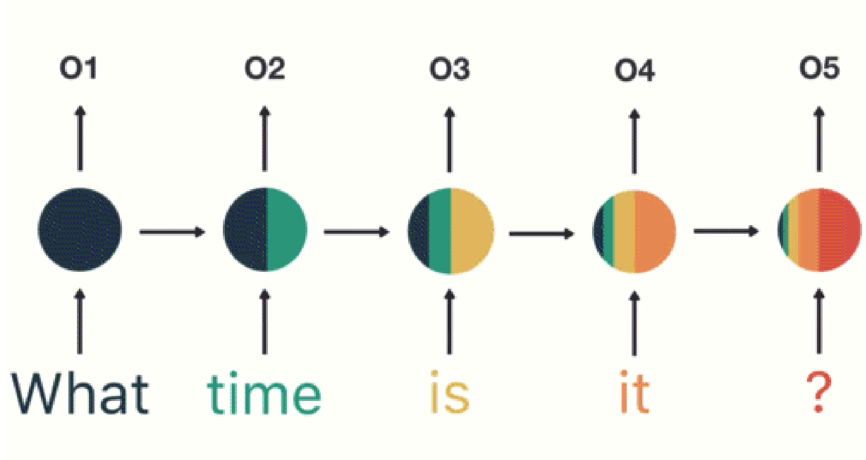
Example: Classifying intents from users' inputs



WHAT ARE RECURRENT NEURAL NETWORKS

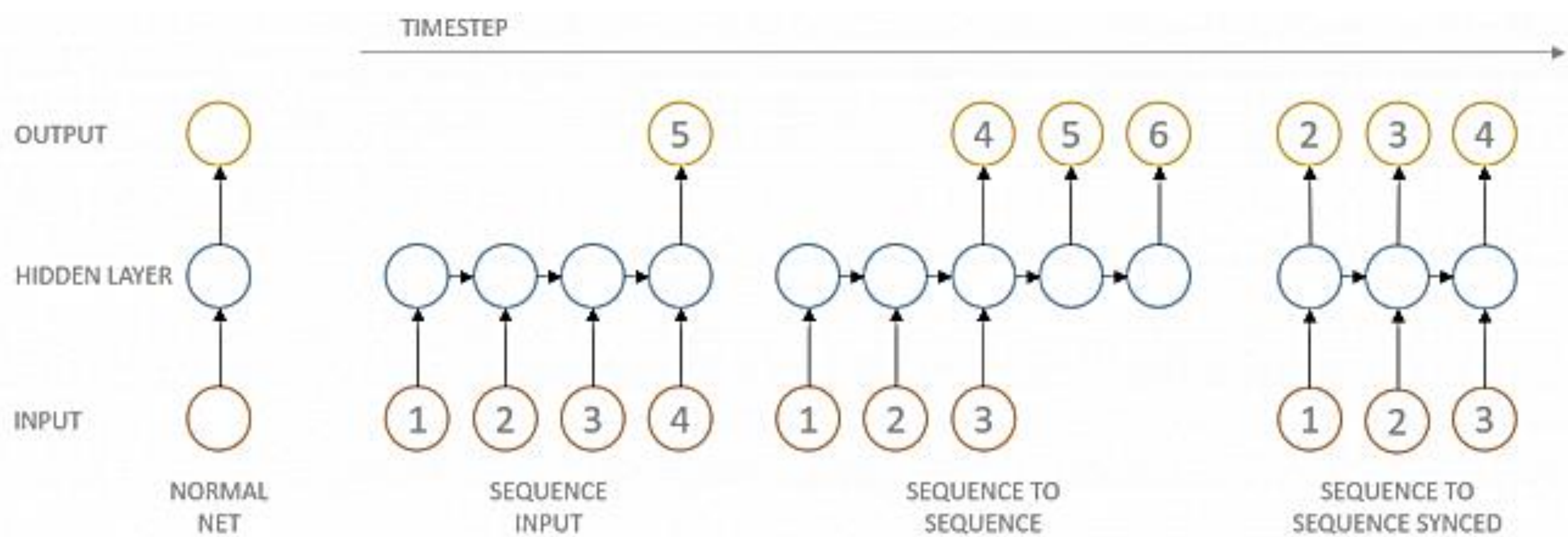
Example: Classifying intents from users' inputs





WHAT ARE RECURRENT NEURAL NETWORKS

Sequence prediction problems



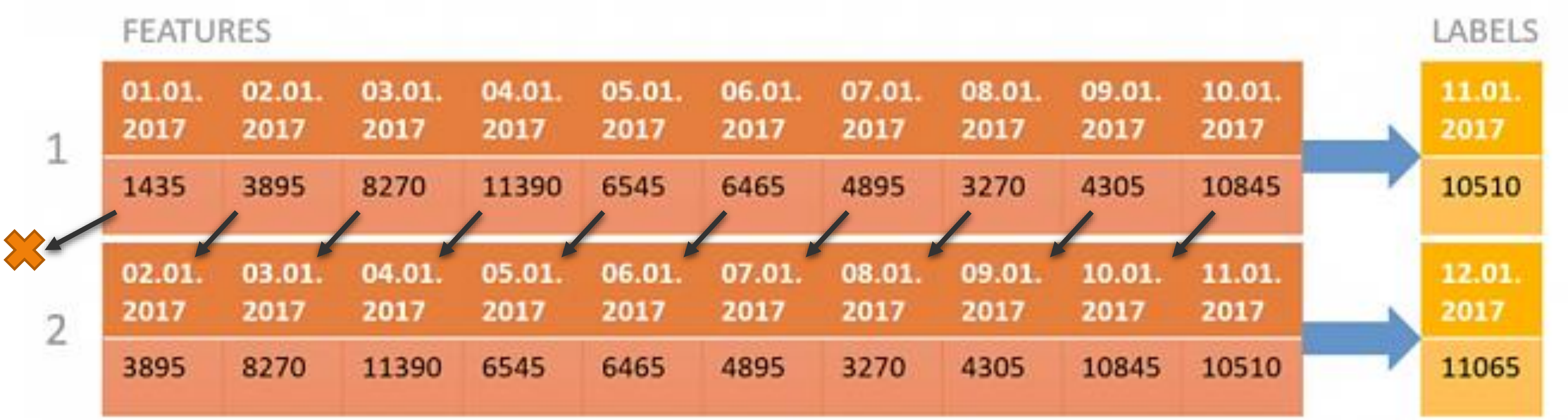
WHAT ARE RECURRENT NEURAL NETWORKS

Sequence creation

	FEATURES										LABELS
1	01.01. 2017	02.01. 2017	03.01. 2017	04.01. 2017	05.01. 2017	06.01. 2017	07.01. 2017	08.01. 2017	09.01. 2017	10.01. 2017	11.01. 2017
	1435	3895	8270	11390	6545	6465	4895	3270	4305	10845	10510

WHAT ARE RECURRENT NEURAL NETWORKS

Sequence creation



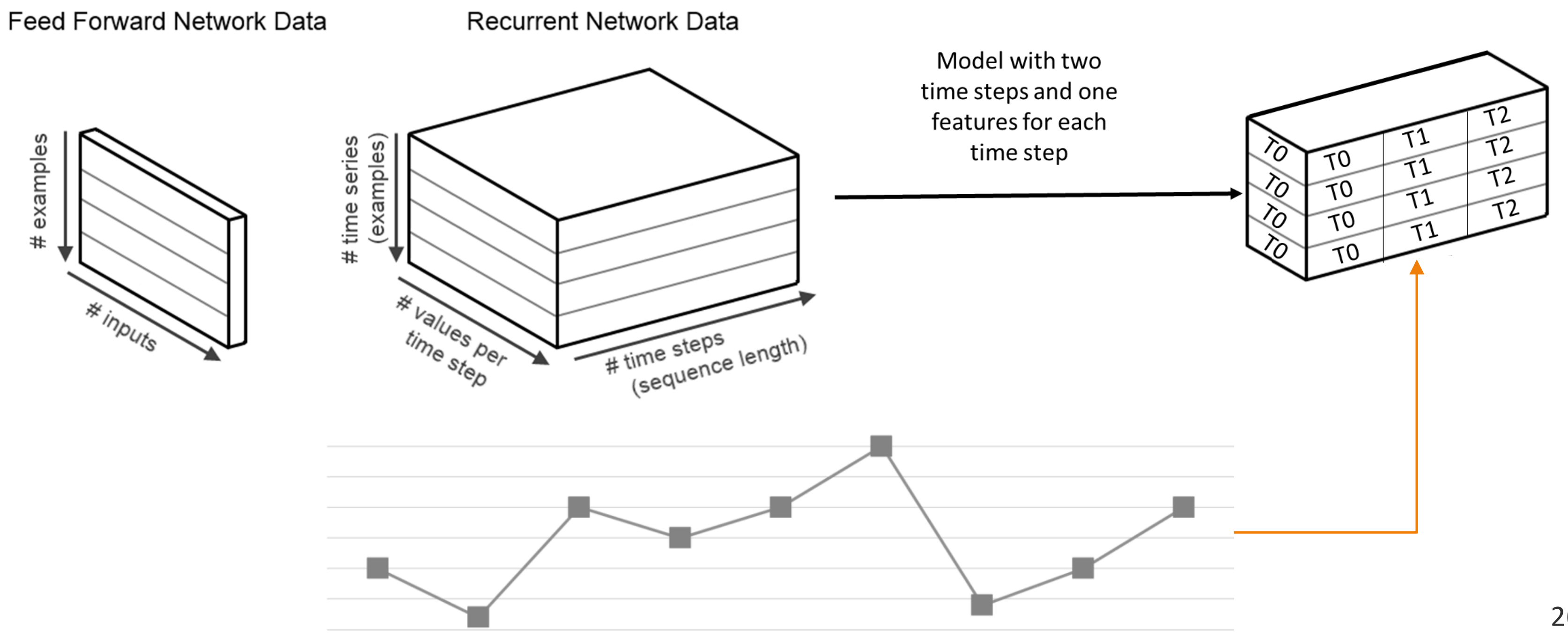
WHAT ARE RECURRENT NEURAL NETWORKS

Sequence creation

	FEATURES										LABELS
1	01.01.2017	02.01.2017	03.01.2017	04.01.2017	05.01.2017	06.01.2017	07.01.2017	08.01.2017	09.01.2017	10.01.2017	11.01.2017
	1435	3895	8270	11390	6545	6465	4895	3270	4305	10845	10510
2	02.01.2017	03.01.2017	04.01.2017	05.01.2017	06.01.2017	07.01.2017	08.01.2017	09.01.2017	10.01.2017	11.01.2017	12.01.2017
	3895	8270	11390	6545	6465	4895	3270	4305	10845	10510	11065
3	03.01.2017	04.01.2017	05.01.2017	06.01.2017	07.01.2017	08.01.2017	09.01.2017	10.01.2017	11.01.2017	12.01.2017	13.01.2017
	8270	11390	6545	6465	4895	3270	4305	10845	10510	11065	50525

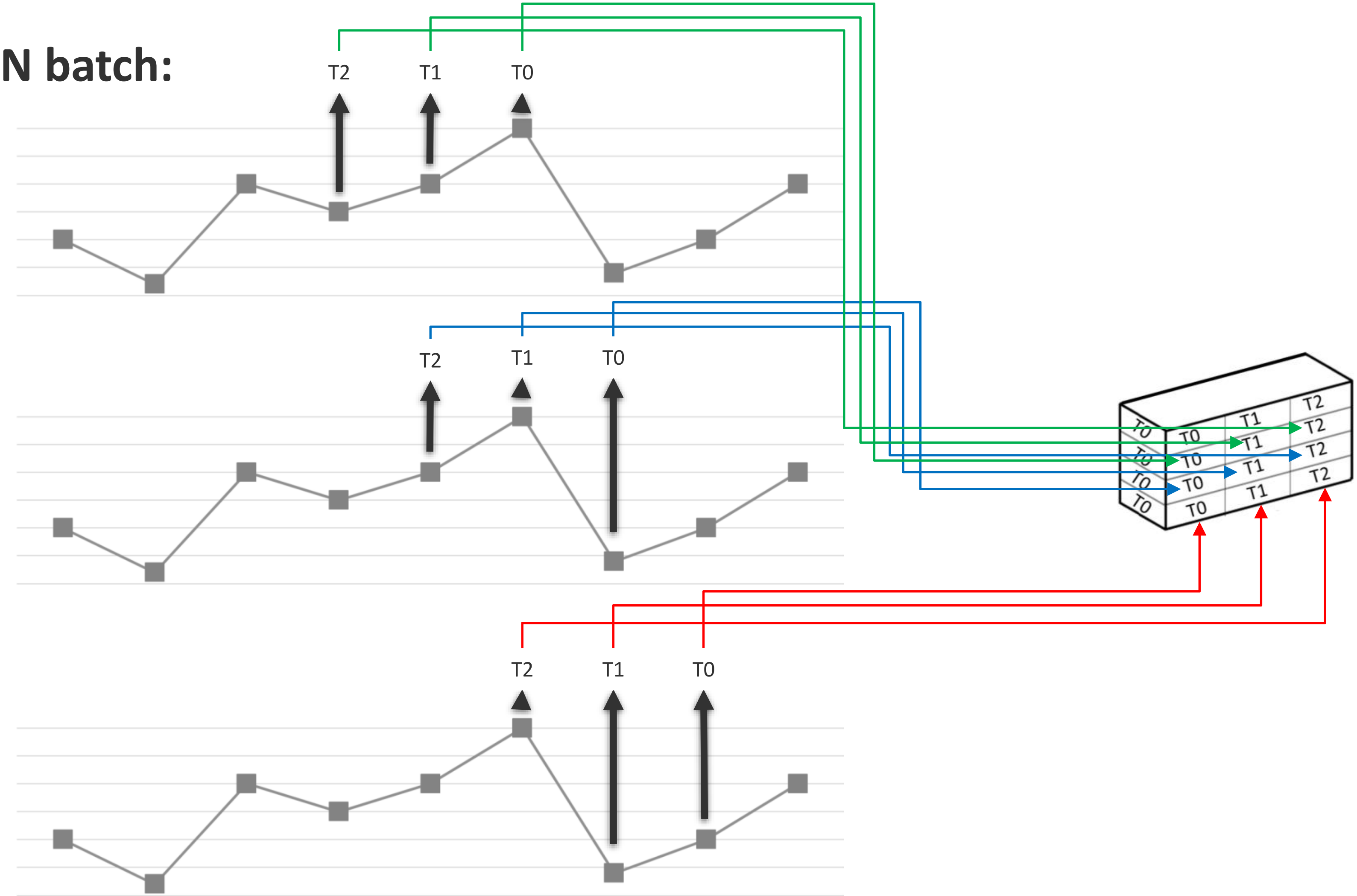
WHAT ARE RECURRENT NEURAL NETWORKS

RNN batch:



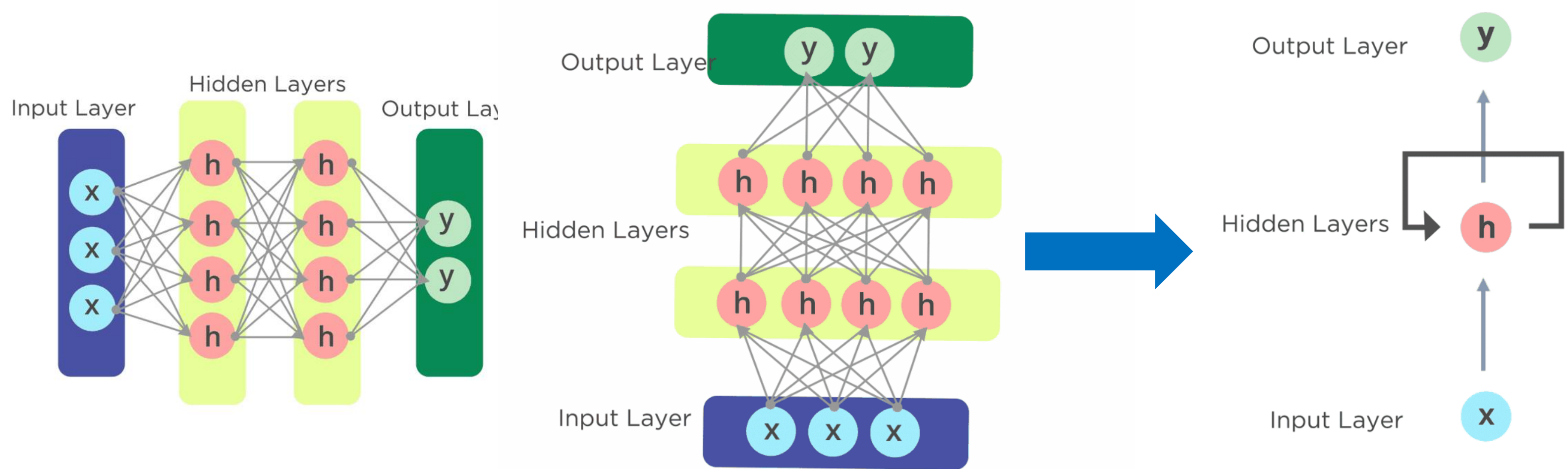
WHAT ARE RECURRENT NEURAL NETWORKS

RNN batch:



WHAT ARE RECURRENT NEURAL NETWORKS

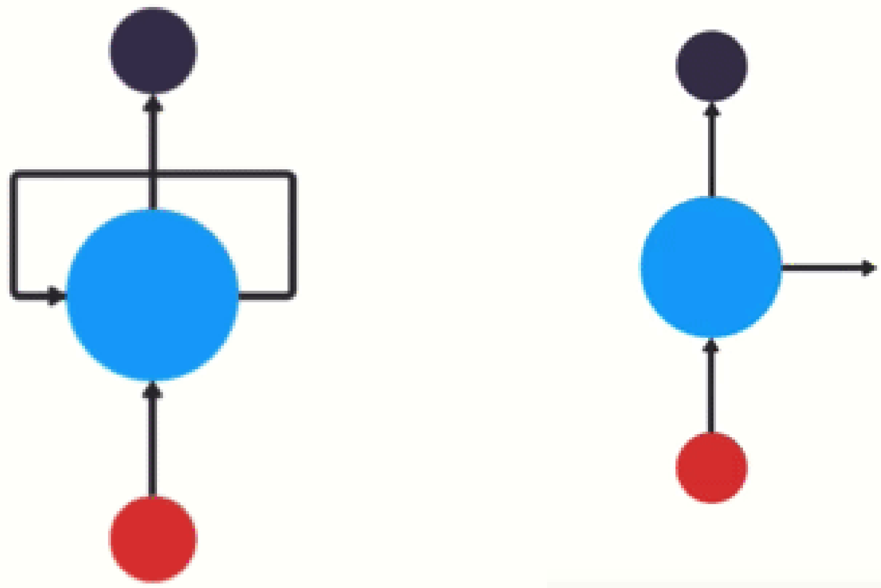
RNN structure:



Key idea: RNNs have an “internal state” that is updated as a sequence is processed

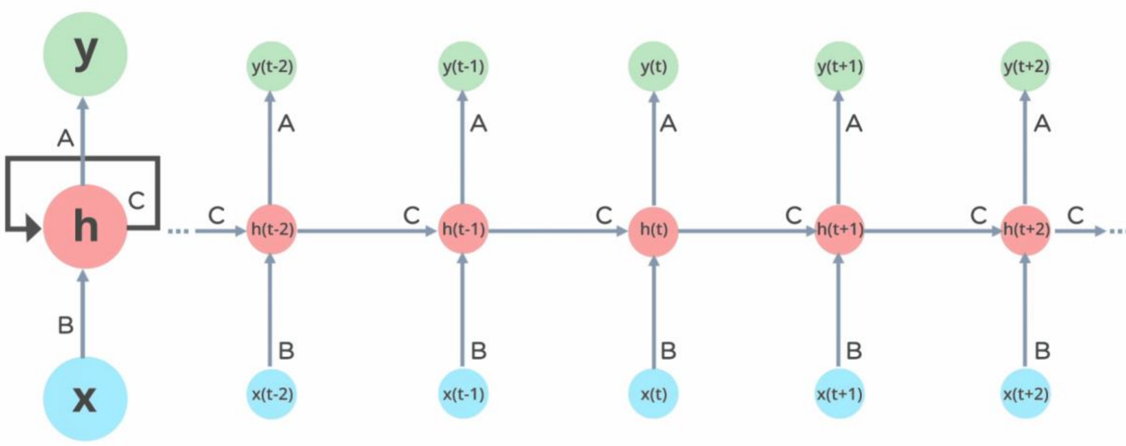
WHAT ARE RECURRENT NEURAL NETWORKS

RNN structure:



WHAT ARE RECURRENT NEURAL NETWORKS

RNN structure:



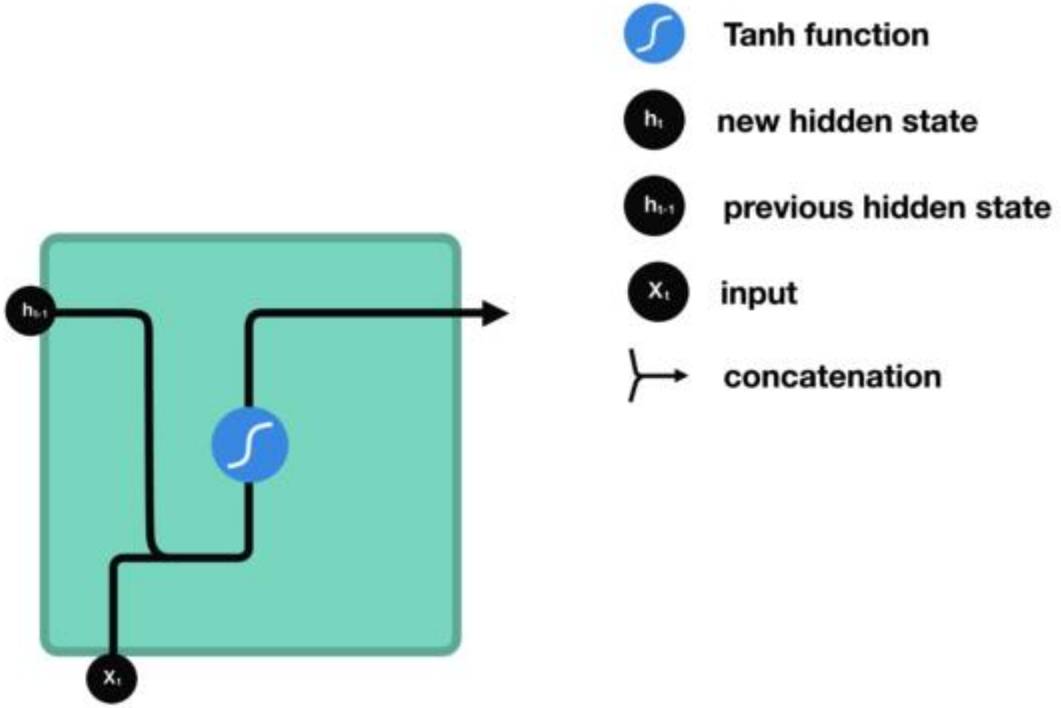
-"x" is the input layer, "h" is the hidden state, and "y" is the output layer

-A, B, and C are the network **parameters**

-At any given time t, the hidden state is a **combination** of input at x(t) and information from previous hidden state by h(t) = f (h(t-1), x(t)), where f is the activation function

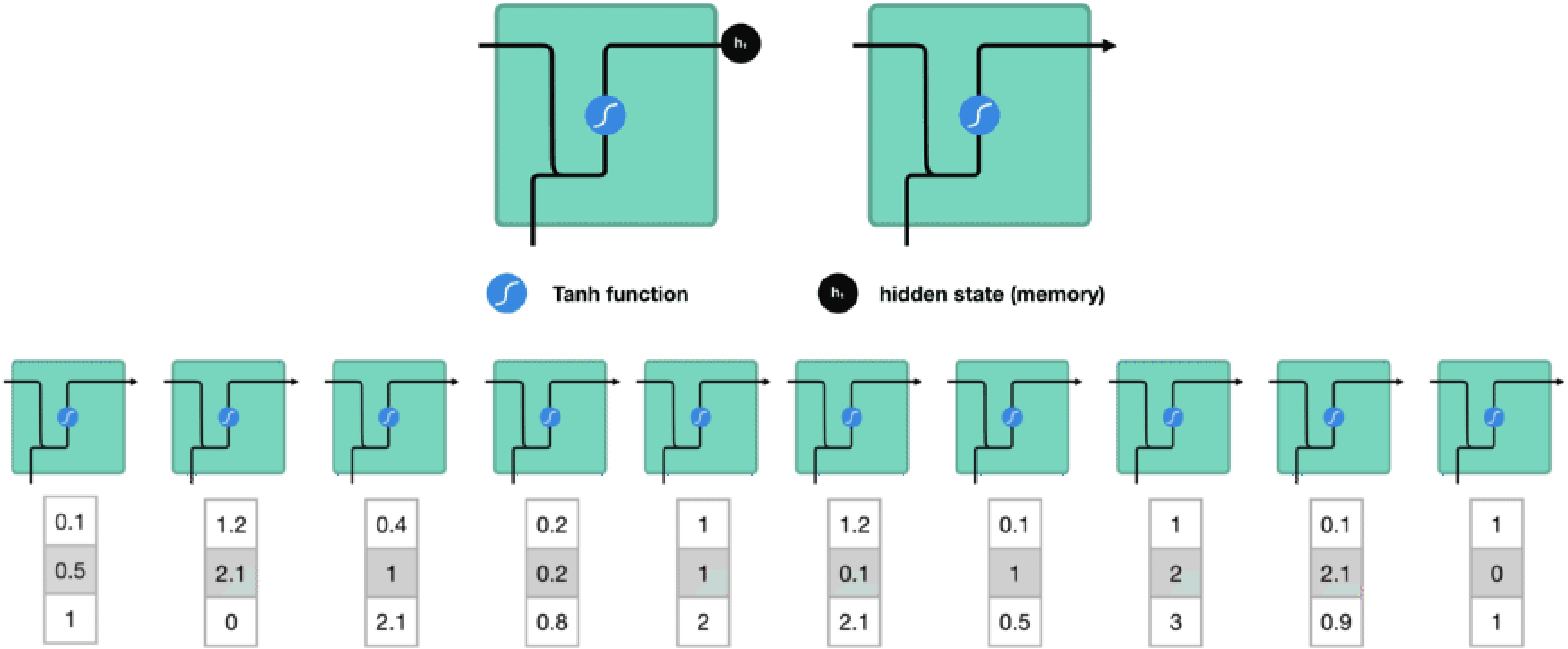
WHAT ARE RECURRENT NEURAL NETWORKS

RNN structure:



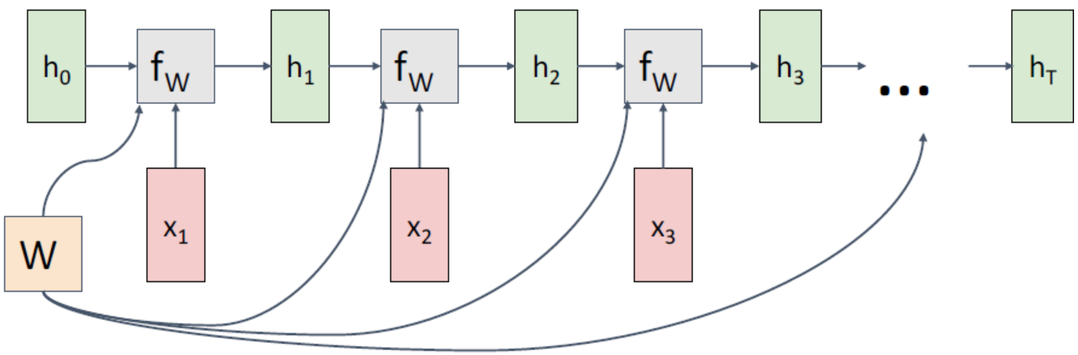
WHAT ARE RECURRENT NEURAL NETWORKS

RNN structure:



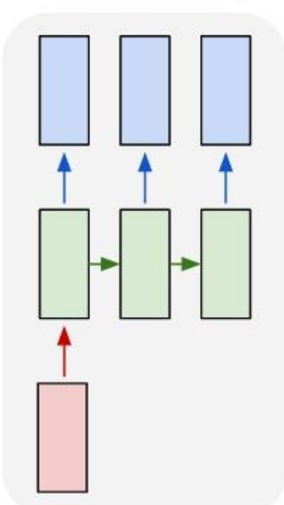
WHAT ARE RECURRENT NEURAL NETWORKS

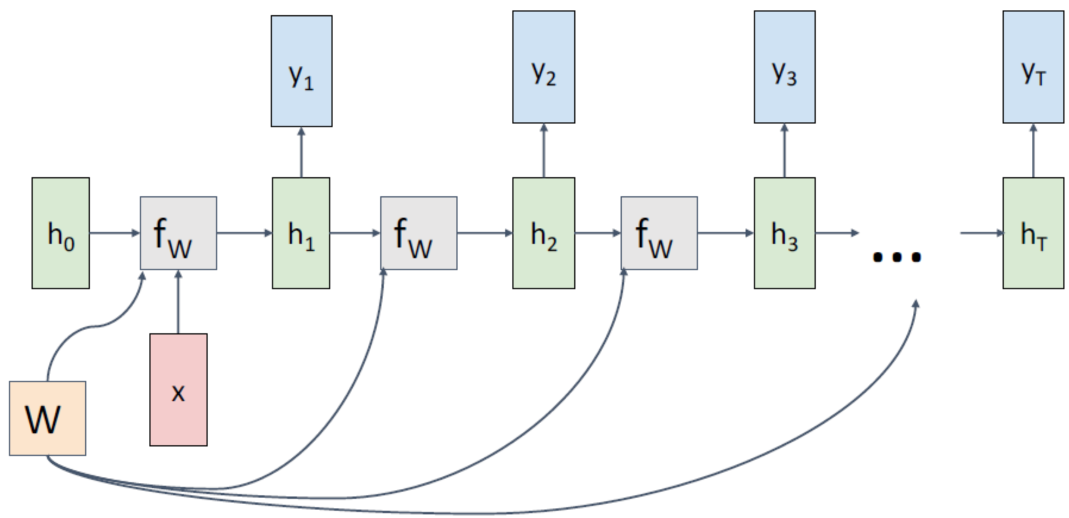
RNN **computational graph**:



Re-use the same **weight matrix** at **every time-step**

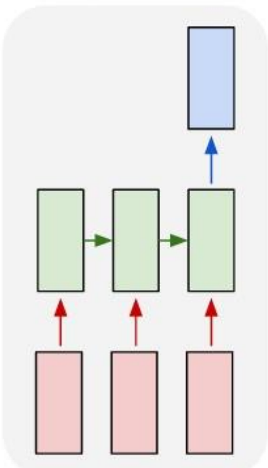
one to many

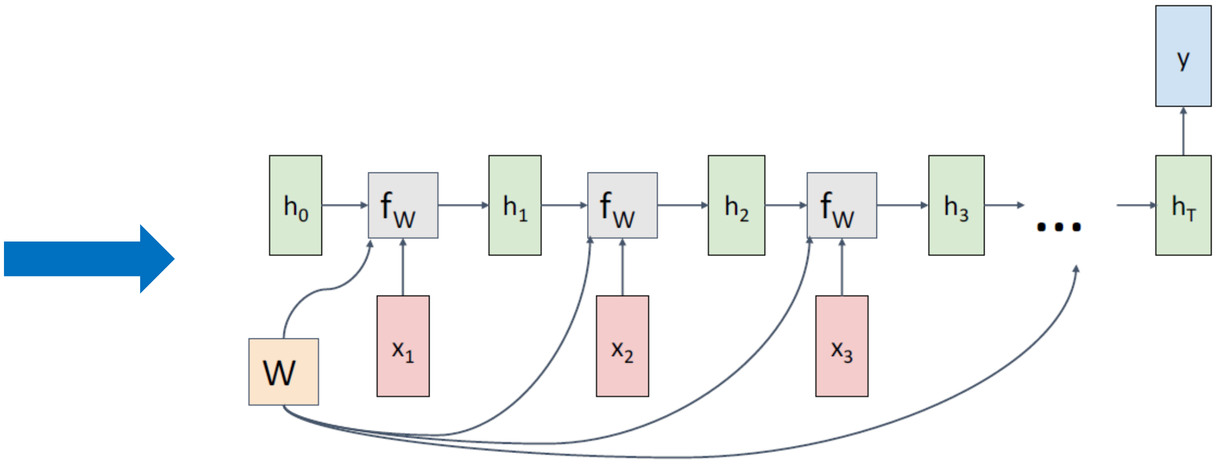




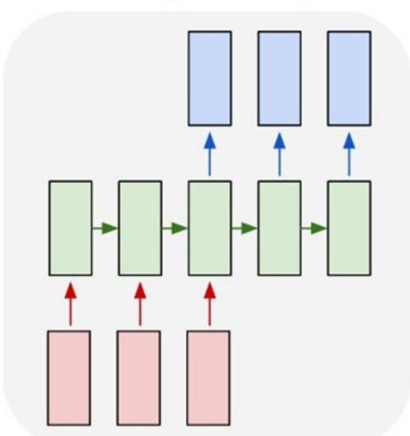
WHAT ARE RECURRENT NEURAL NETWORKS

many to one

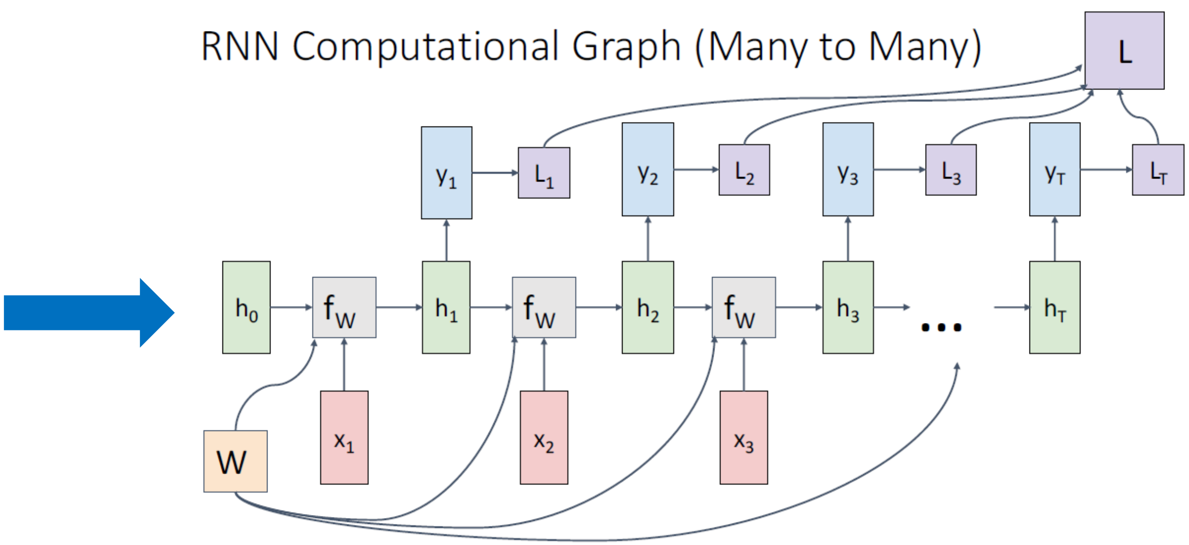




many to many

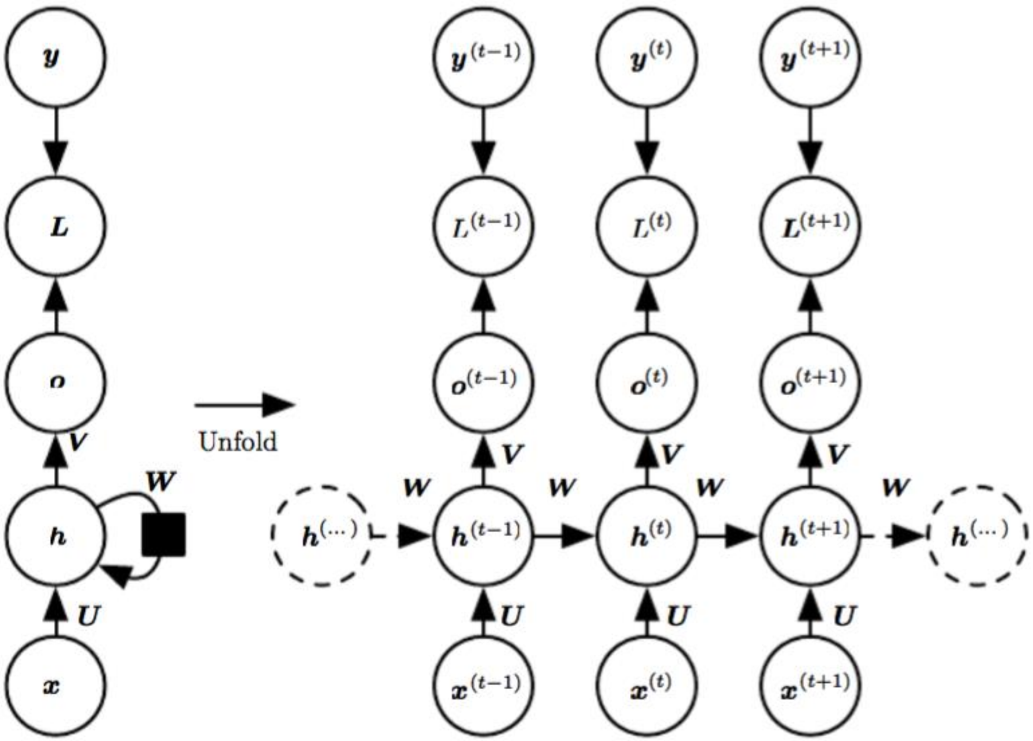


RNN Computational Graph (Many to Many)



WHAT ARE RECURRENT NEURAL NETWORKS

RNN structure:

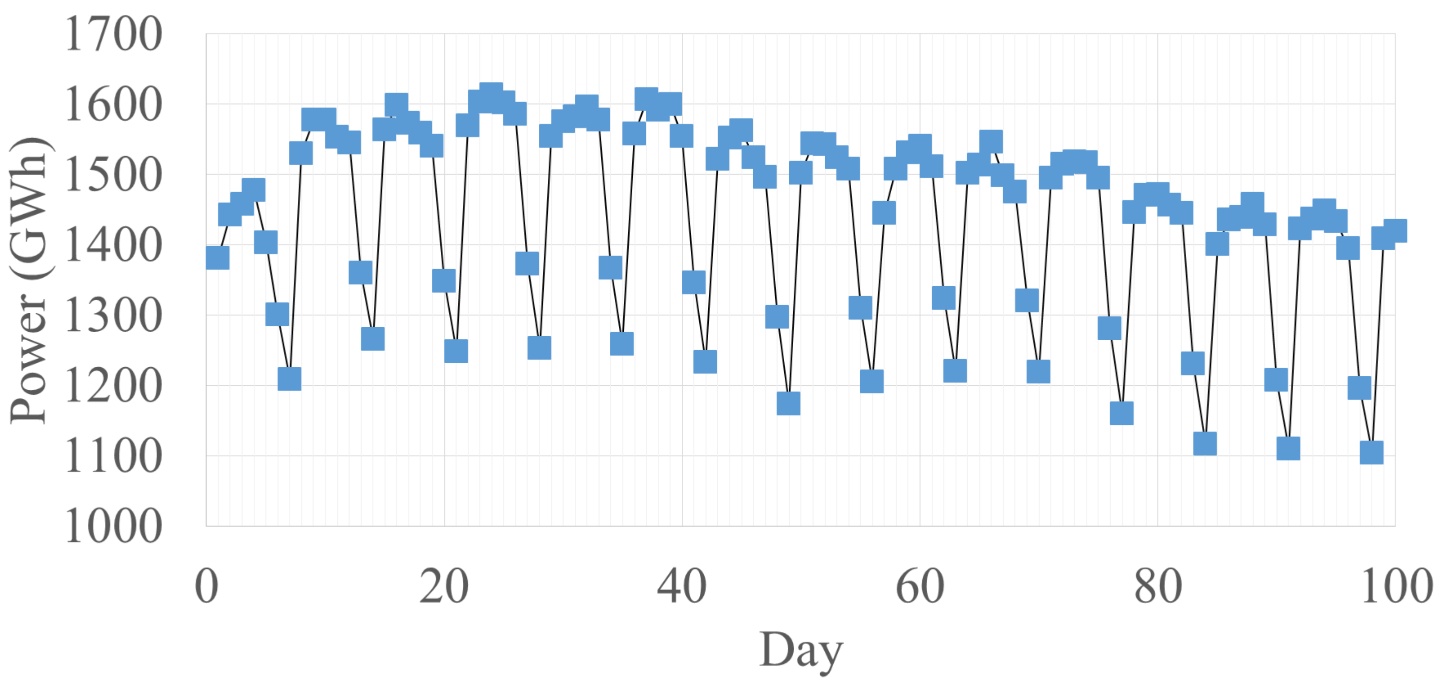


$$
\begin{aligned}
\boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)} \\
\boldsymbol{h}^{(t)} &= \tanh(\boldsymbol{a}^{(t)}) \\
\boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)} \\
\hat{\boldsymbol{y}}^{(t)} &= \mathrm{softmax}(\boldsymbol{o}^{(t)})
\end{aligned}
$$

WHAT ARE RECURRENT NEURAL NETWORKS

RNN Colab example: RNN – Energy

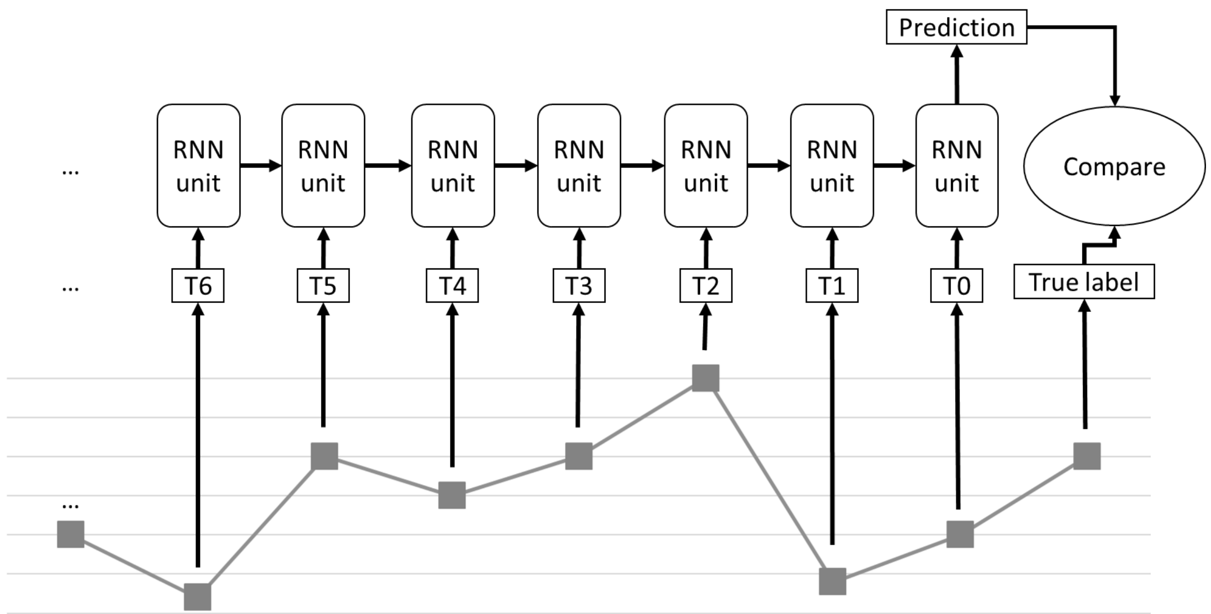
- Forecast the electricity consumption (in GWh) in Germany



WHAT ARE RECURRENT NEURAL NETWORKS

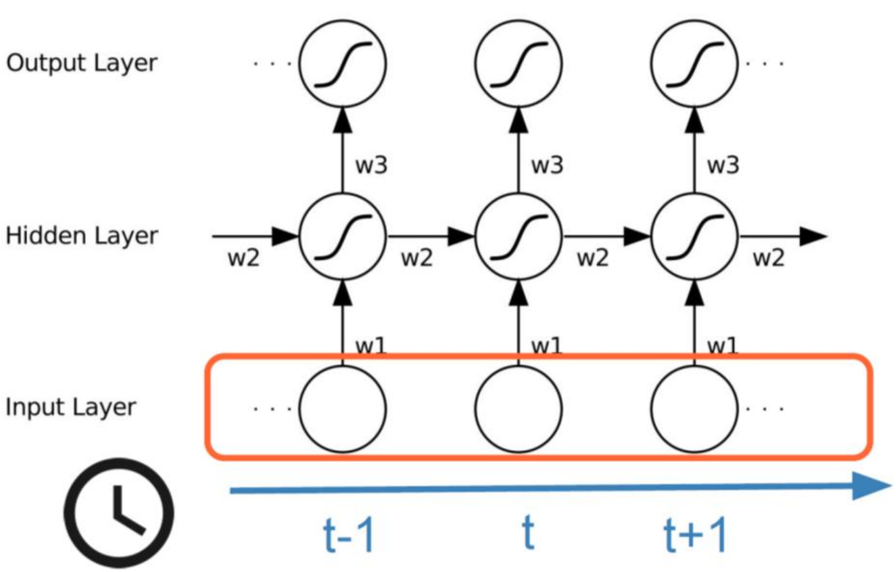
RNN Colab example: RNN – Energy

- Use 33 time steps (33 days)
- Estimate the power consumption for next day

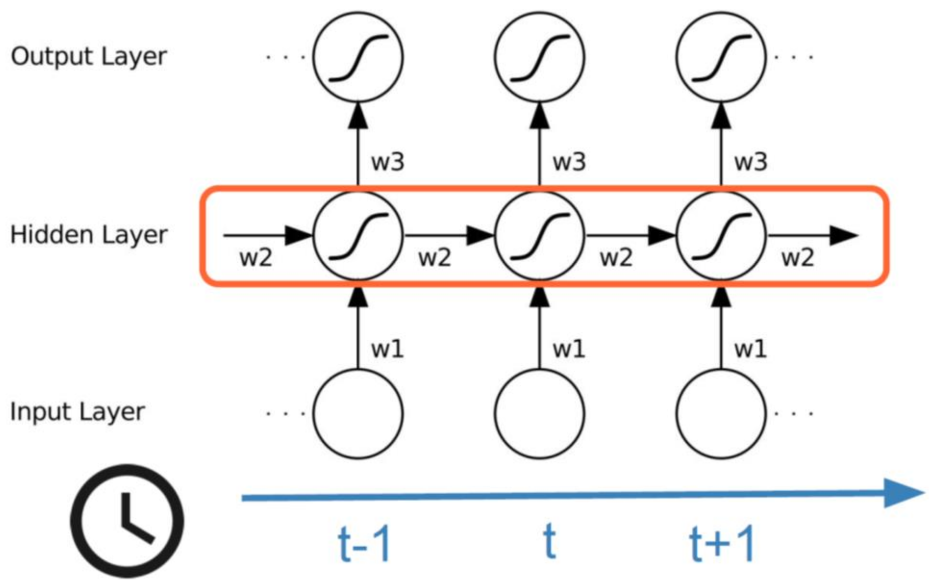


WHEN THE RNN BREAKS

WHEN THE RNN BREAKS

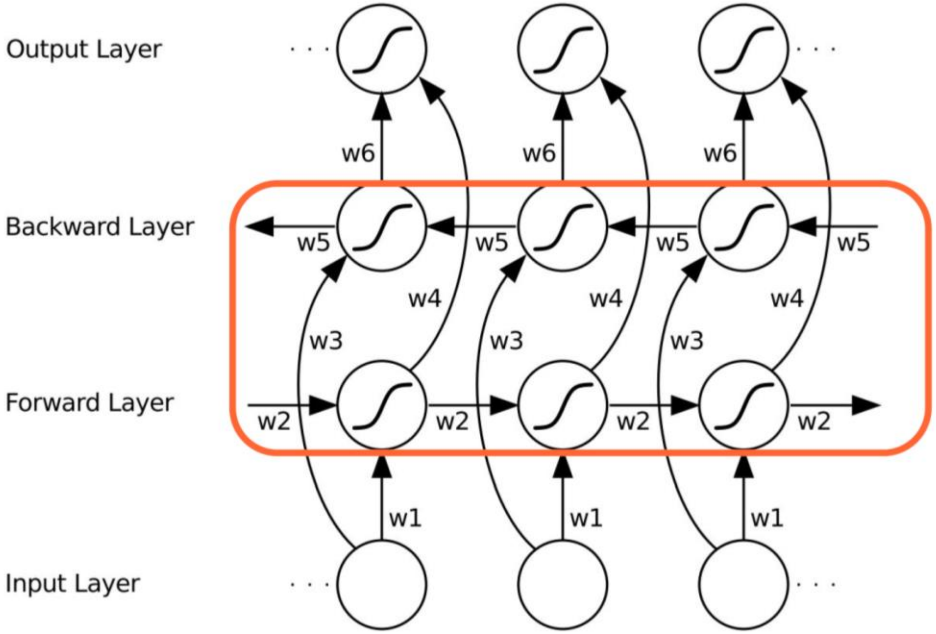


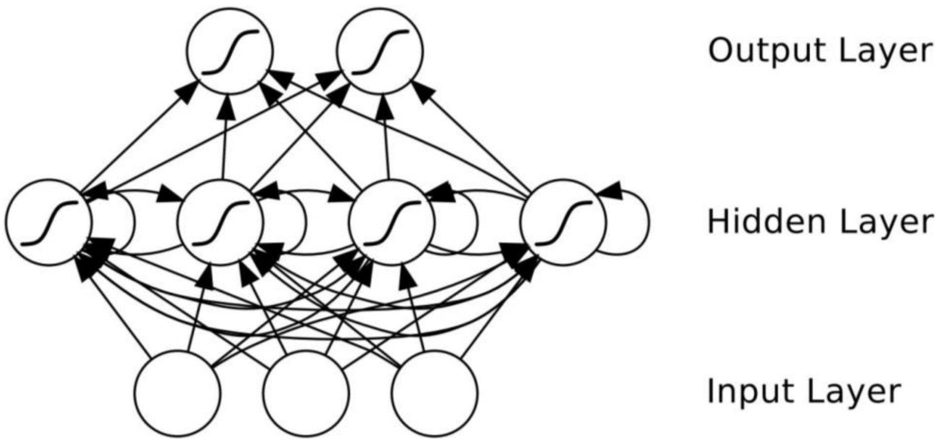
The input is a sequence x(t) of any length



Must learn temporally shared weights w2 in addition to w1 and w3

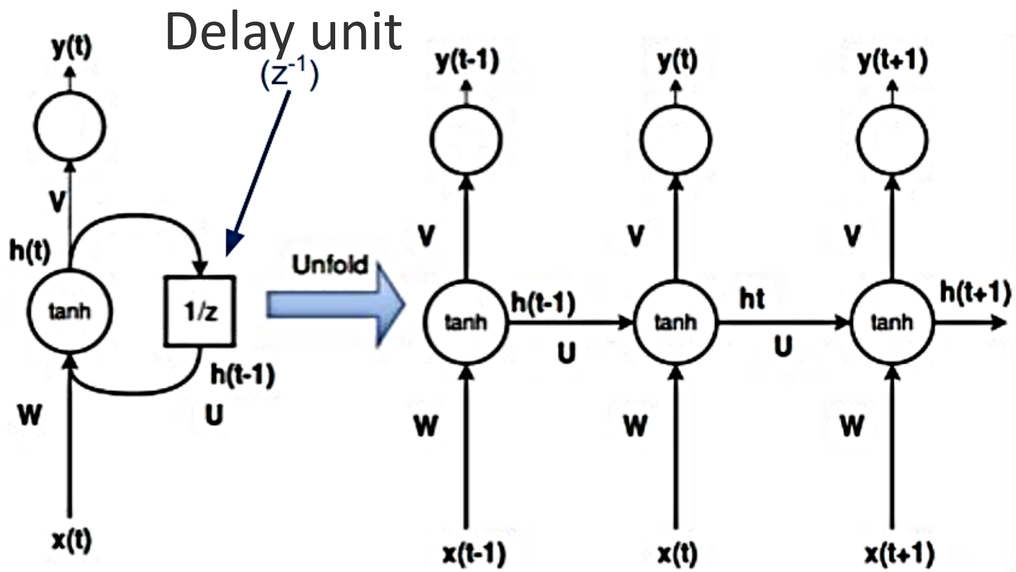
WHEN THE RNN BREAKS





For bidirectional RNN, must learn weights w2, w3, w4, and w5, in addition to w1 and w6

WHEN THE RNN BREAKS



RNN has two data flows: forward in space + time propagation

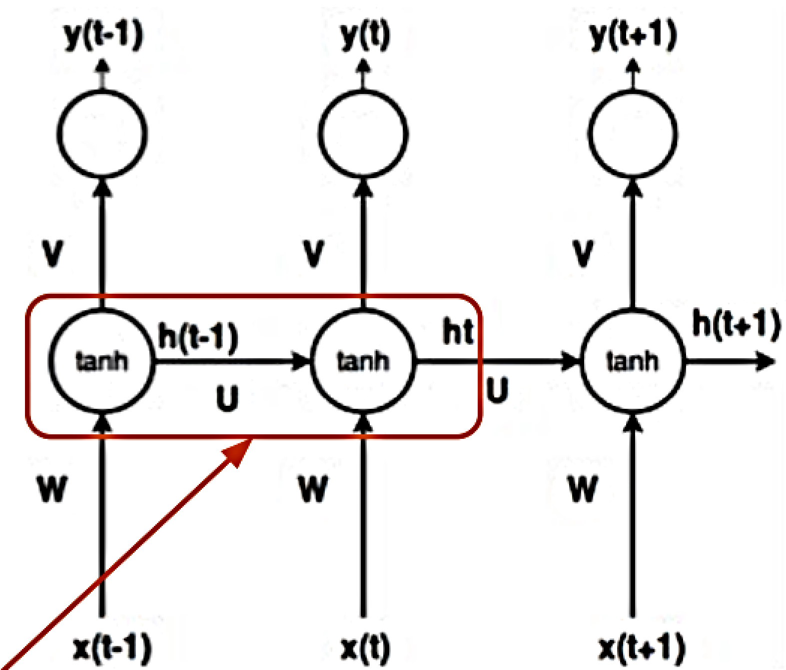
Beware: We have extra depth now! Every time-step is an extra level of depth (as a deeper stack of layers in a feed-forward fashion!)

WHEN THE RNN BREAKS

$$\mathbf{y}_t = f(\mathbf{V} \cdot \mathbf{h}_t + \mathbf{b}_v)$$

$$\mathbf{h}_t = g(\mathbf{W} \cdot \mathbf{x}_t + \boxed{\mathbf{U} \cdot \mathbf{h}_{t-1}} + \mathbf{b}_h)$$

$$\mathbf{x}_t \in \mathbb{R}^n$$



One-time
Recurrence

WHEN THE RNN BREAKS

Back Propagation Through Time (**BPTT**): The **training** method has to **take into account** the **time operations** → a **cost function E** is defined to train our RNN, and in this case, the **total error** at the output of the network is the **sum of the errors at each time-step**:

$$E(\mathbf{y},\hat{\mathbf{y}}) = \sum_{t=1}^{T} E_t(\mathbf{y}_t,\hat{\mathbf{y}}_t) \qquad\qquad \frac{\partial E}{\partial \mathbf{W}} = \sum_{t=0}^{T-1} \frac{\partial E_t}{\partial \mathbf{W}}$$

Long-term memory (remembering quite far time-steps) **vanishes** quickly because of the recursive operation with U (**temporal depth**), thus, during training **gradients explode/vanish** easily because of depth-in-time

One time-step recurrence

$$\mathbf{h}_t = g(\mathbf{W}\cdot\mathbf{x}_t + \boxed{\mathbf{U}\cdot\mathbf{h}_{t-1}} + \mathbf{b}_h)$$

Recurrence

T time steps recurrences

$$\mathbf{h}_t = g(\mathbf{W}\cdot\mathbf{x}_t + \boxed{\mathbf{U}\cdot g(\cdots g(\mathbf{W}\cdot\mathbf{x}_{t-T} + \mathbf{U}\cdot\mathbf{h}_{t-T} + \mathbf{b}_h)\cdots)} + \mathbf{b}_h)$$

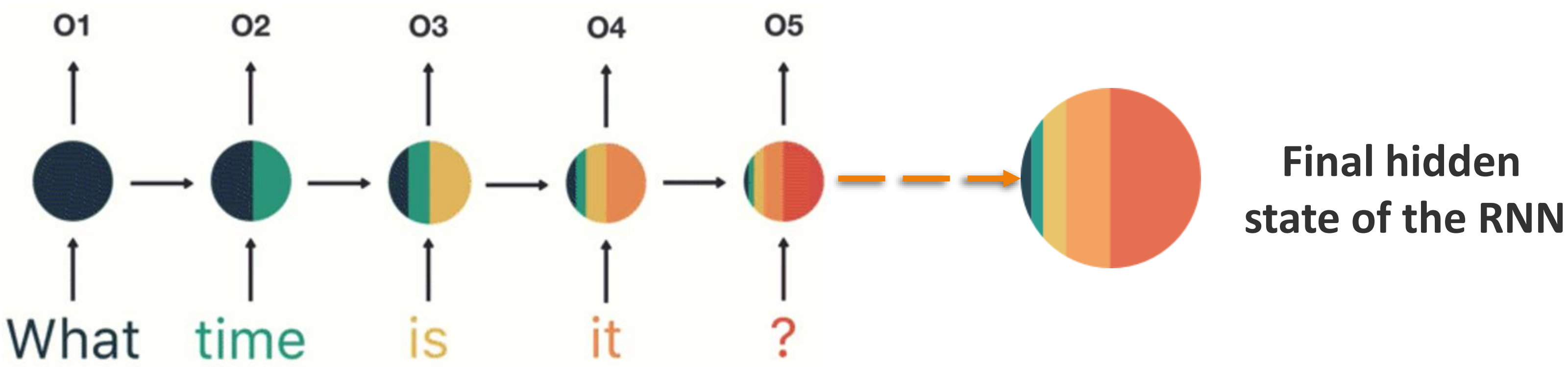
WHEN THE RNN BREAKS

Vanishing gradient:

- Gradient allows the network to learn by adjusting the weights
- The higher the gradient, the higher the adjustments
- Each neuron estimates it’s gradient with respect to the gradient of the layer before it
- If the layers before have small adjustments, then adjustments to the current layer will be even smaller
- Gradients exponentially shrink as it back propagates

WHEN THE RNN BREAKS

As the RNN processes more steps, it has troubles retaining information from previous steps

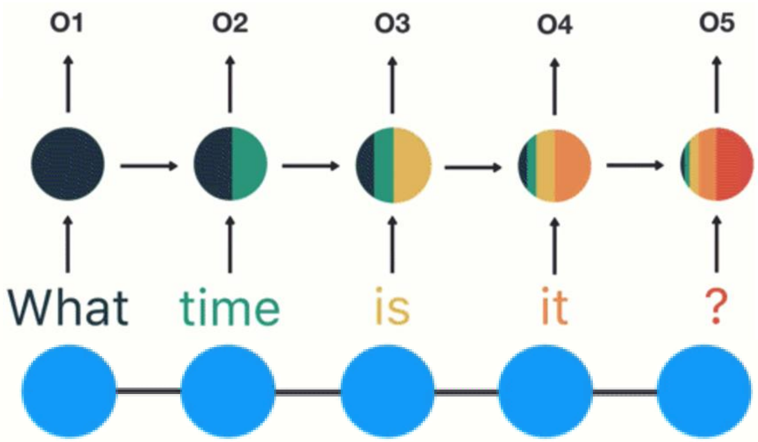


- Information from the words “what” and “time” is nearly extinct at the final time step
- This short-term memory problem is caused by the vanishing gradient during back-propagation

WHEN THE RNN BREAKS

Vanishing gradient:

- Think of each time step of the RNN as a layer
- Use back-propagation through time to train
- The gradient values will exponentially shrink as it propagates through each time step



WHEN THE RNN BREAKS

Solution (Gating method):

1. **Change** the **way** in which **past information is kept** → create the notion of **cell state**, a memory unit that **keeps long-term information** in a **safer** way by protecting it from recursive operations

2. Make every **RNN unit** able to **decide** whether the **current time-step information matters** or not, to accept or discard (optimized reading mechanism)

3. Make every **RNN unit able to forget** whatever may not be useful anymore by clearing that info from the cell state (optimized clearing mechanism)

4. Make every **RNN unit** able to **output** the **decisions whenever** it is **ready** to do so (optimized output mechanism)

LONG SHORT-TERM MEMORY

LONG SHORT-TERM MEMORY

Intuition:

- Read a review to decide if you want to buy a cereal
- Determine if someone thought it was good or bad

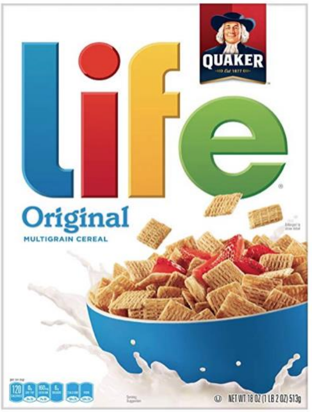
Customers Review 2,491

Thanos

September 2018

Verified Purchase

Amazing! This box of cereal gave me a perfectly balanced breakfast, as all things should be. I only ate half of it but will definitely be buying again!



A Box of Cereal
$3.99

LONG SHORT-TERM MEMORY

Intuition:

- Your brain will only remember the important keywords such as“amazing” and “perfectly balanced breakfast”
- The irrelevant words will be ignored

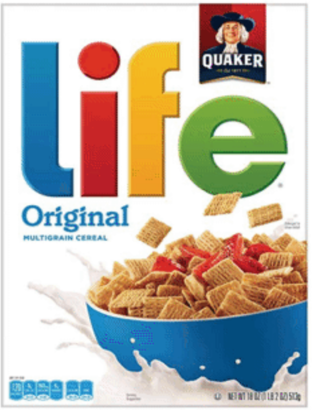
Customers Review 2,491

Thanos

September 2018

Verified Purchase

Amazing! This box of cereal gave me a perfectly balanced breakfast, as all things should be. I only ate half of it but will definitely be buying again!

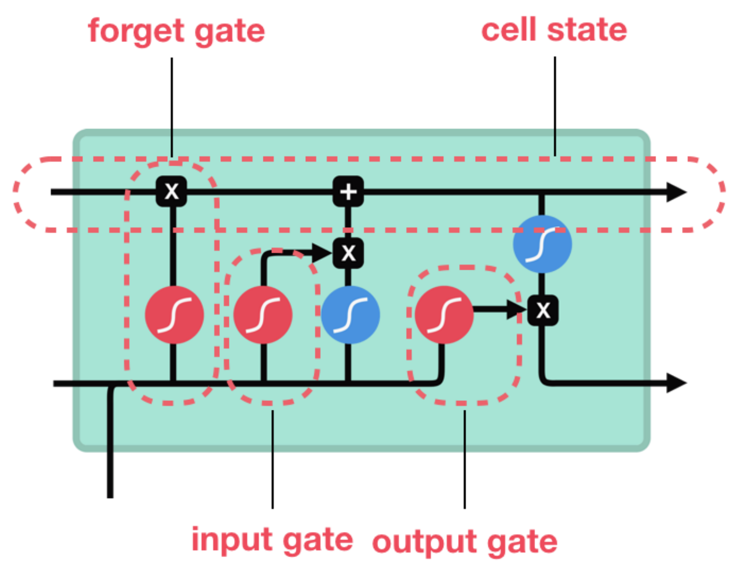


A Box of Cereal
$3.99

LONG SHORT-TERM MEMORY

How to address the problem:

- The Long Short-Term Memory (LSTM) keeps only relevant information to make predictions
- Use gate mechanism to learn long-term dependencies
- These gates are trained to identify what information should be added or removed to the hidden state



LONG SHORT-TERM MEMORY

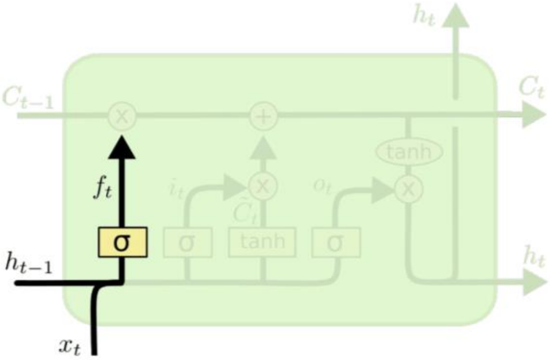
How LSTM works:

- The LSTM is a combination of gates and a cell state
- The cell state acts as the network's memory and transfers information across the sequence chain
- Information from all time steps can reach the output cell, reducing the short-term memory effects

LONG SHORT-TERM MEMORY

How LSTM works:

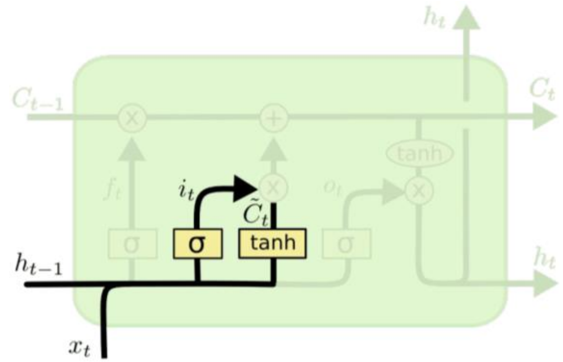
- Forget gate decides what information should be kept or thrown away
- The information from the previous hidden state and current input is transformed by the sigmoid (0 to 1)
- Values closer to 1 means to keep while closer to 0 is to forget



LONG SHORT-TERM MEMORY

How LSTM works:

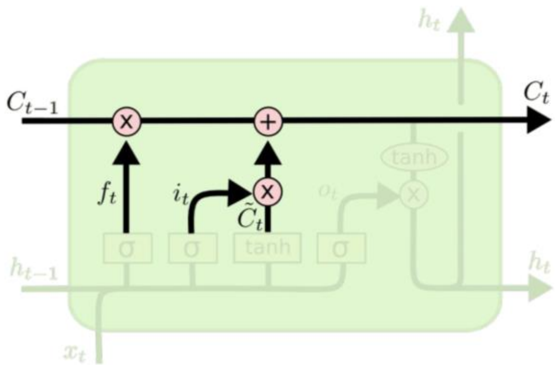
- Input gate allows to update the cell state, according to the output of the sigmoid function (0 to 1)
- If 0 then is irrelevant (skipping the time step) while 1 is very important
- The information from the previous hidden state and current input is multiplied by the sigmoid output to update the cell state



LONG SHORT-TERM MEMORY

How LSTM works:

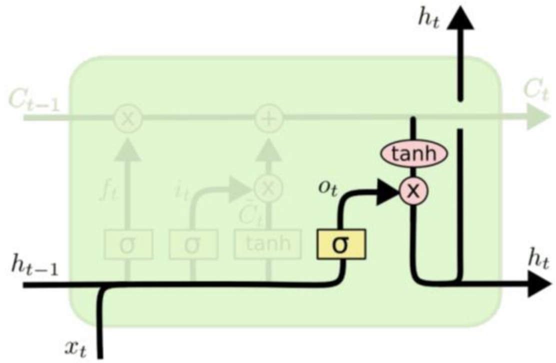
- The previous cell state is multiplied by the forget gate's output
- Then the input gate's output is added, producing the new cell state



LONG SHORT-TERM MEMORY

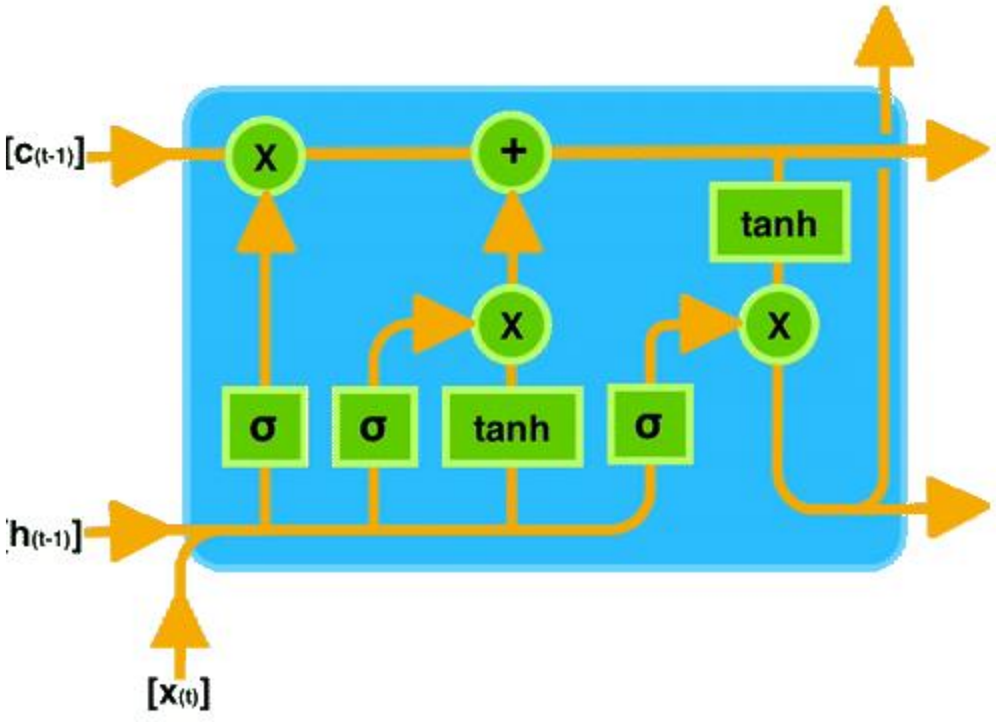
How LSTM works:

- The output gate selected the relevant information to be used as the next hidden.
- This decision is taken according to the output of the sigmoid function
- The output is the hidden state of the current cell



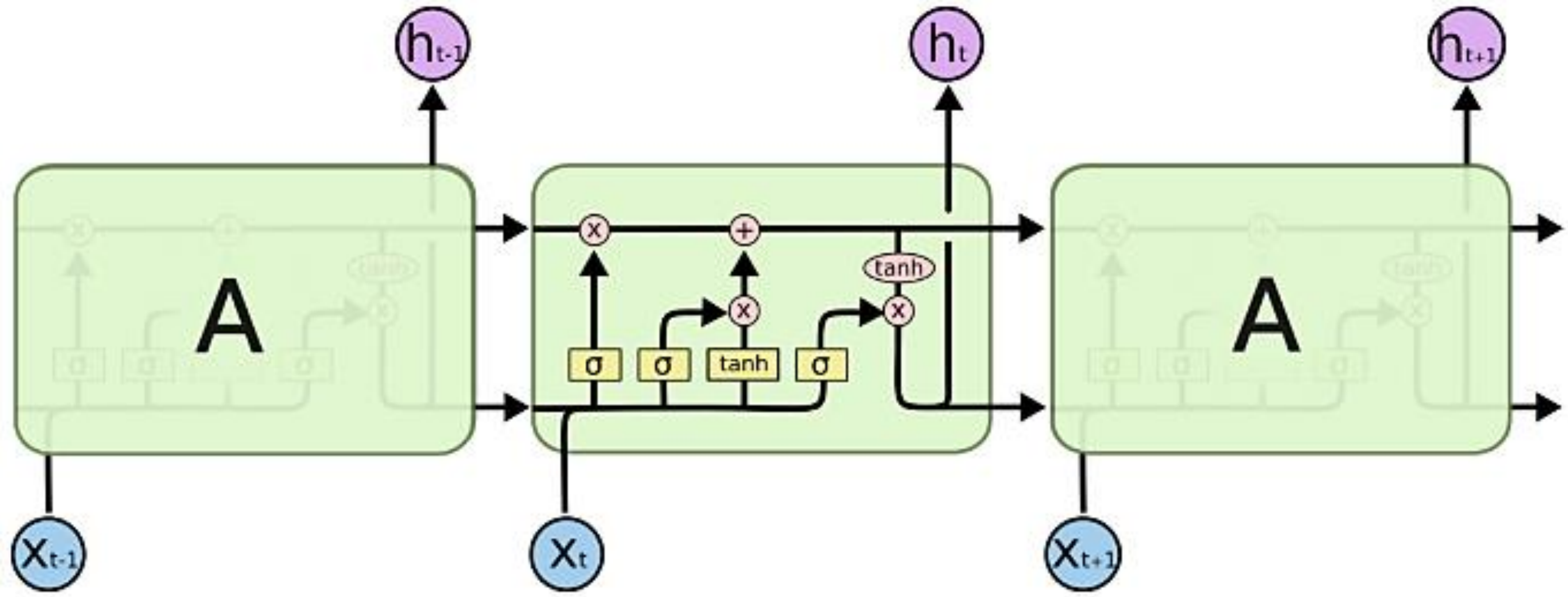
LONG SHORT-TERM MEMORY

How LSTM works:



LONG SHORT-TERM MEMORY

How LSTM works:



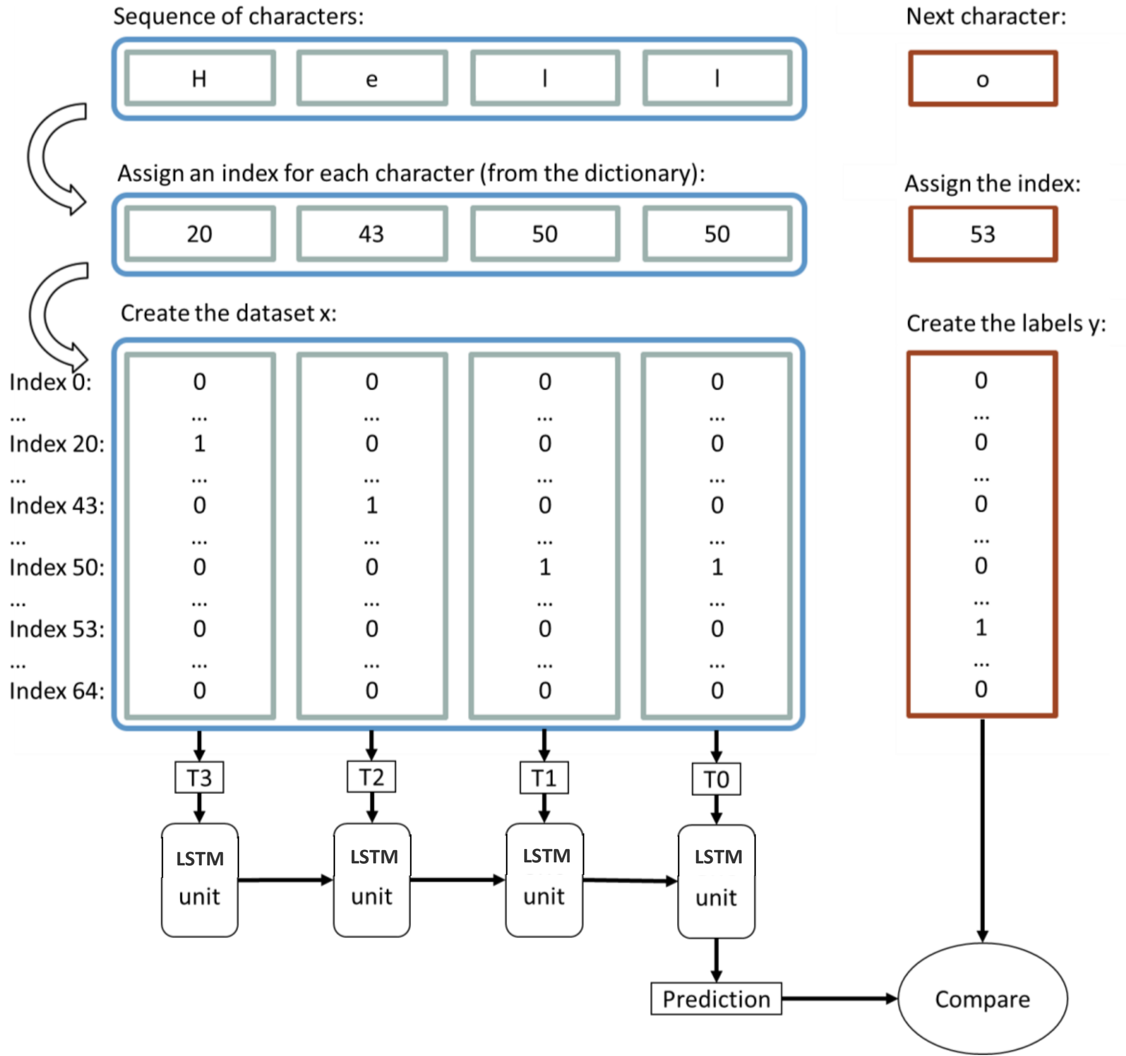
LONG SHORT-TERM MEMORY

LSTM Colab example: LSTM – Shakespeare

- Use character-based RNN to generate a Shakespeare's-like text based on the Shakespeare dataset
- All of Shakespeare's plays, characters, lines, and acts
- Total of 1,115,394 characters where 65 are different
- All unique characters: \n, , !, $, &, ', ,, -, ., 3, :, ;, ?, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z

LONG SHORT-TERM MEMORY

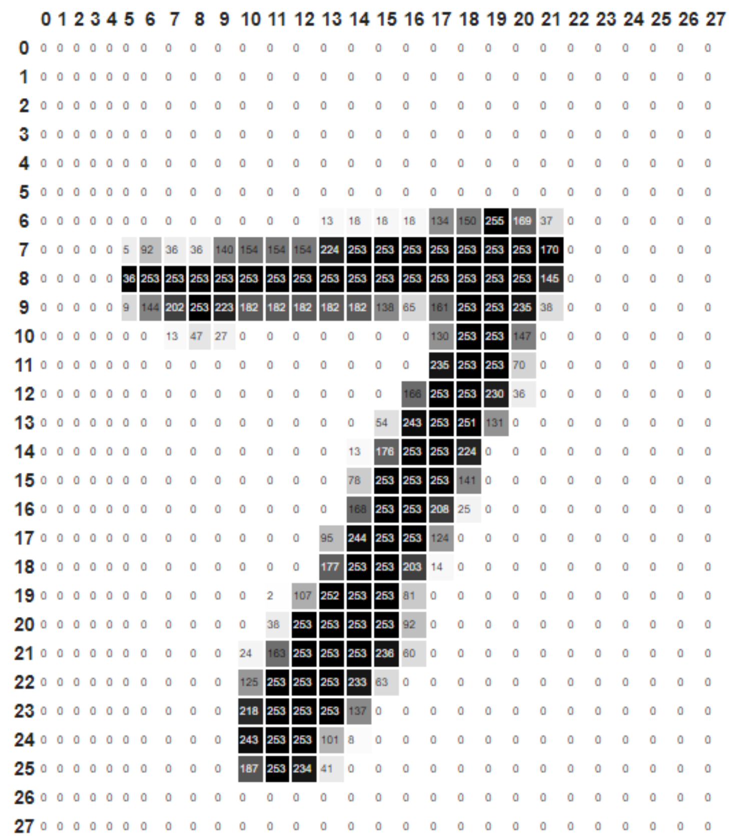
LSTM Colab example:
LSTM – Shakespeare



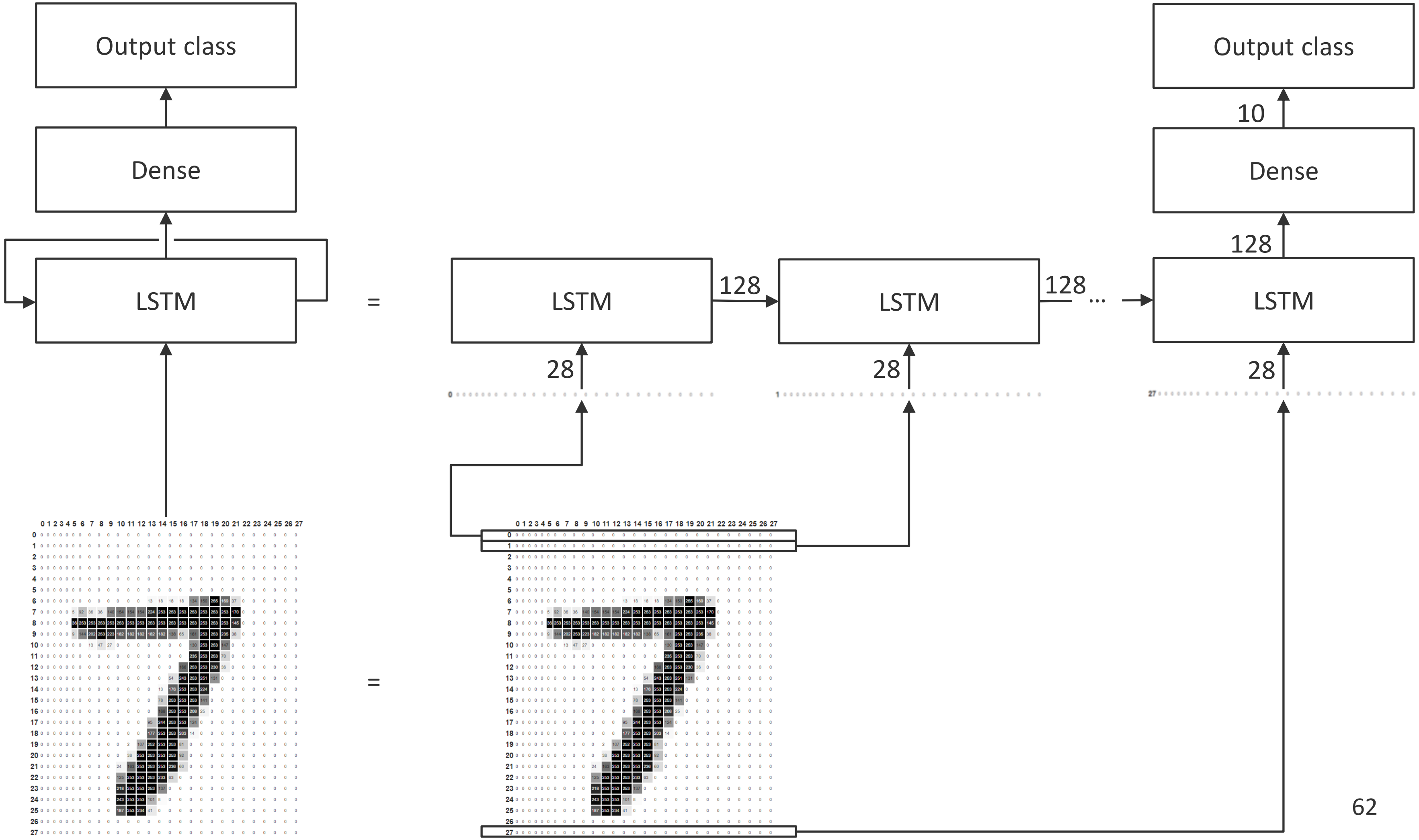
LONG SHORT-TERM MEMORY

LSTM Colab example: LSTM – MNIST

- Handwritten dataset
- 70000 images
- All are 28x28
- 784 pixels in total



LONG SHORT-TERM MEMORY



LONG SHORT-TERM MEMORY

LSTM advantages:

- Are usually the most accurate among the RNN
- As the best when the problem involves longer sequences

LSTM issues:

- Are slow to train
- As the complexity of the problem increases, it also increases the amount of data required to properly train
- Requires hardware with large memory

SOURCES

SOURCES

- https://towardsdatascience.com/illustrated-guide-to-recurrent-neural-networks-79e5eb8049c9
- https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21
- https://www.novatec-gmbh.de/en/blog/recurrent-neural-networks-for-time-series-forecasting/
- http://clipart-library.com/
- https://www.simplilearn.com/tutorials/deep-learning-tutorial/rnn
- https://www.dlology.com/blog/how-to-use-return_state-or-return_sequences-in-keras/

SOURCES

- https://colah.github.io/posts/2015-08-Understanding-LSTMs/
- https://towardsdatascience.com/ghost-writing-with-tensorflow-49e77e26978f
- https://hackernoon.com/understanding-architecture-of-lstm-cell-from-scratch-with-code-8da40f0b71f4
- http://imatge-upc.github.io/telecombcn-2016-dlcv/slides/D2L6-recurrent.pdf
- https://d2l.ai/chapter_recurrent-modern/gru.html
- https://www.kdnuggets.com/2020/06/introduction-convolutional-neural-networks.html
- https://mgubaidullin.github.io/deeplearning4j-docs/usingrnns.html