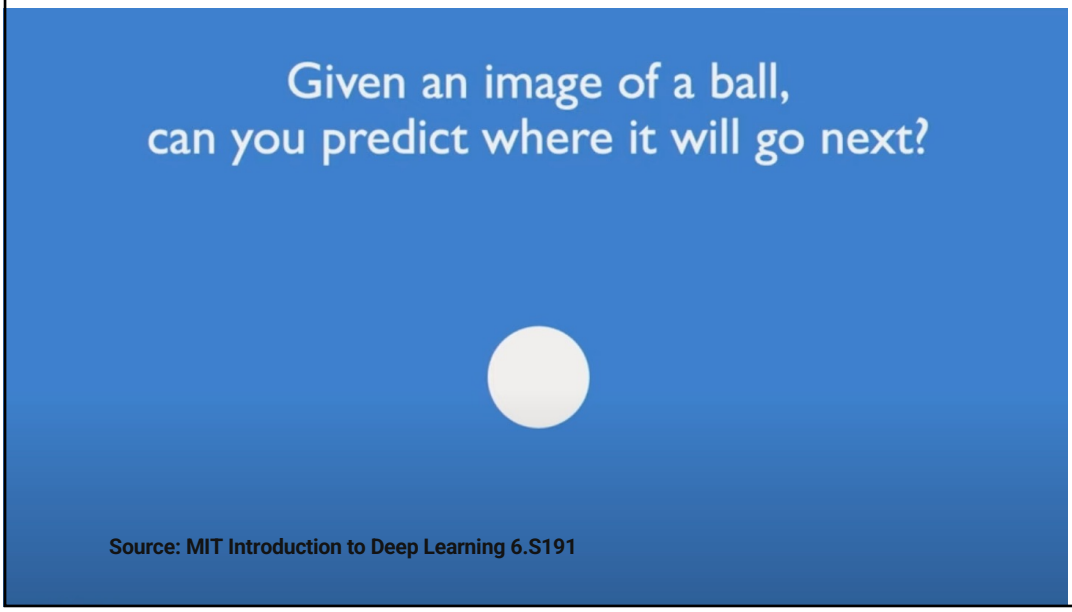
1

Outline

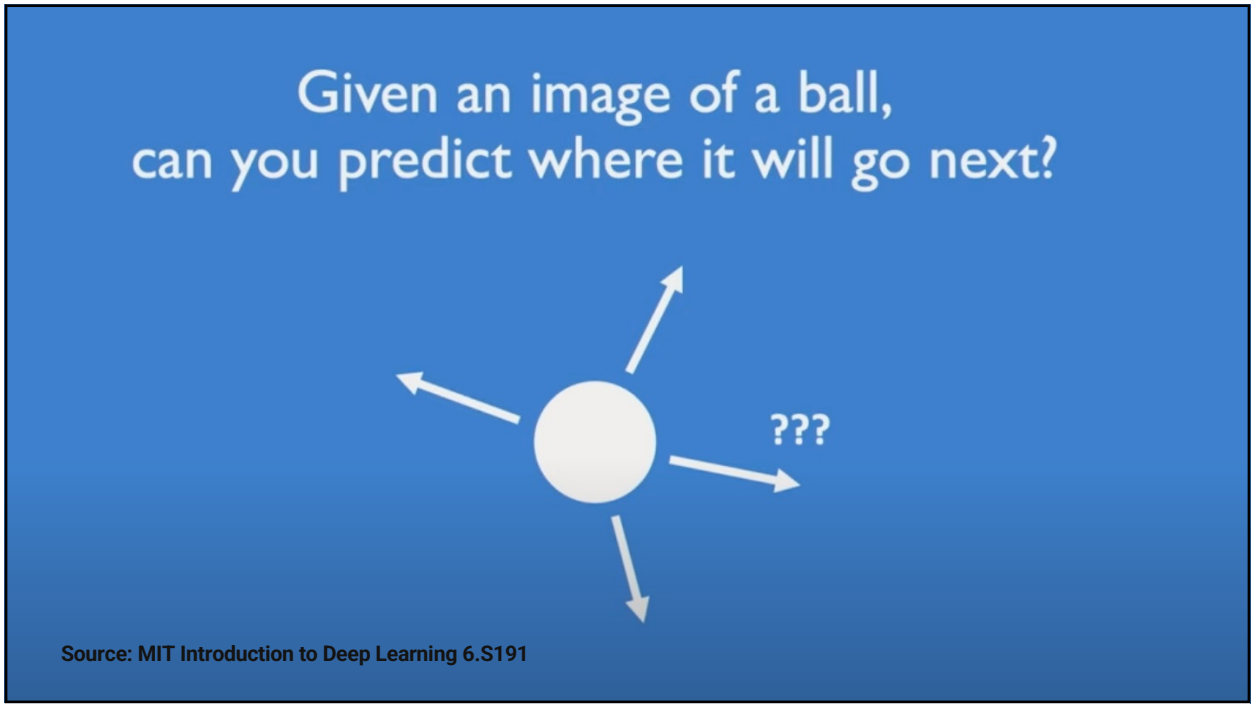
- Recurrent Neural Networks (RNNs)
  - Motivation, basic concepts, examples
- LSTM & GRU
- Transformer
  - Example applications

2

Sequences & Sequence modelling

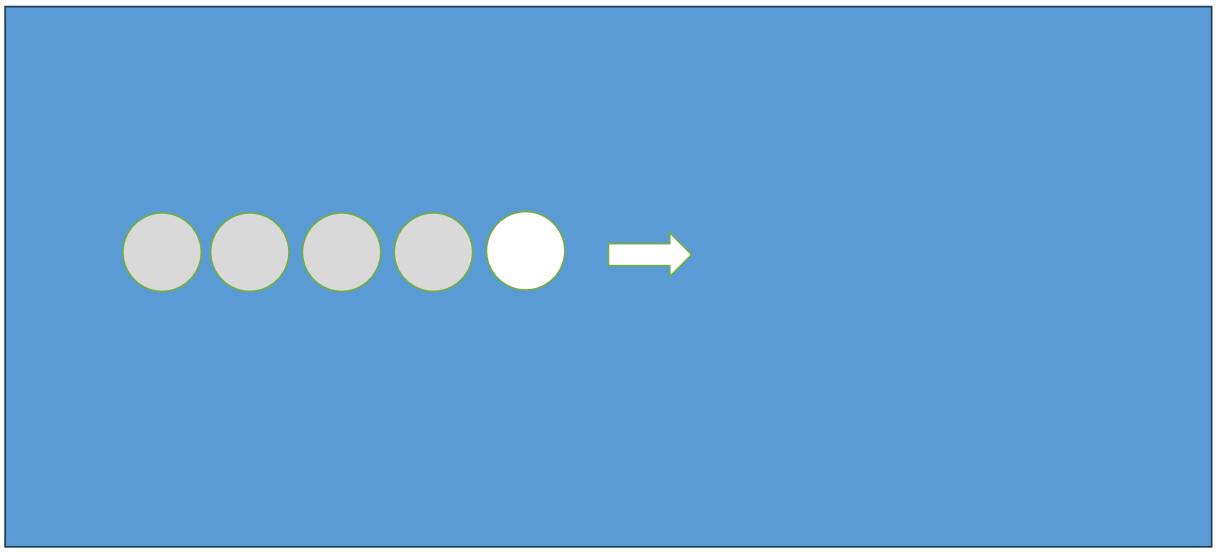


3



4

Now, given a history of the ball's positions



Source: MIT Introduction to Deep Learning 6.S191

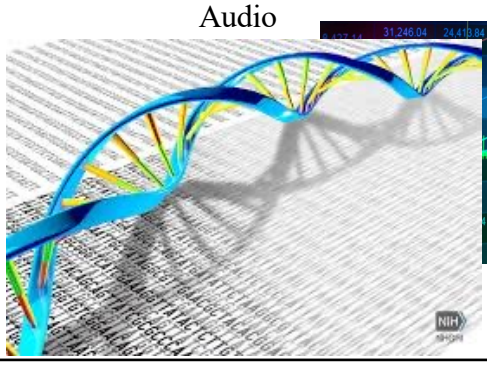
5

Sequences & Sequence modelling

Audio

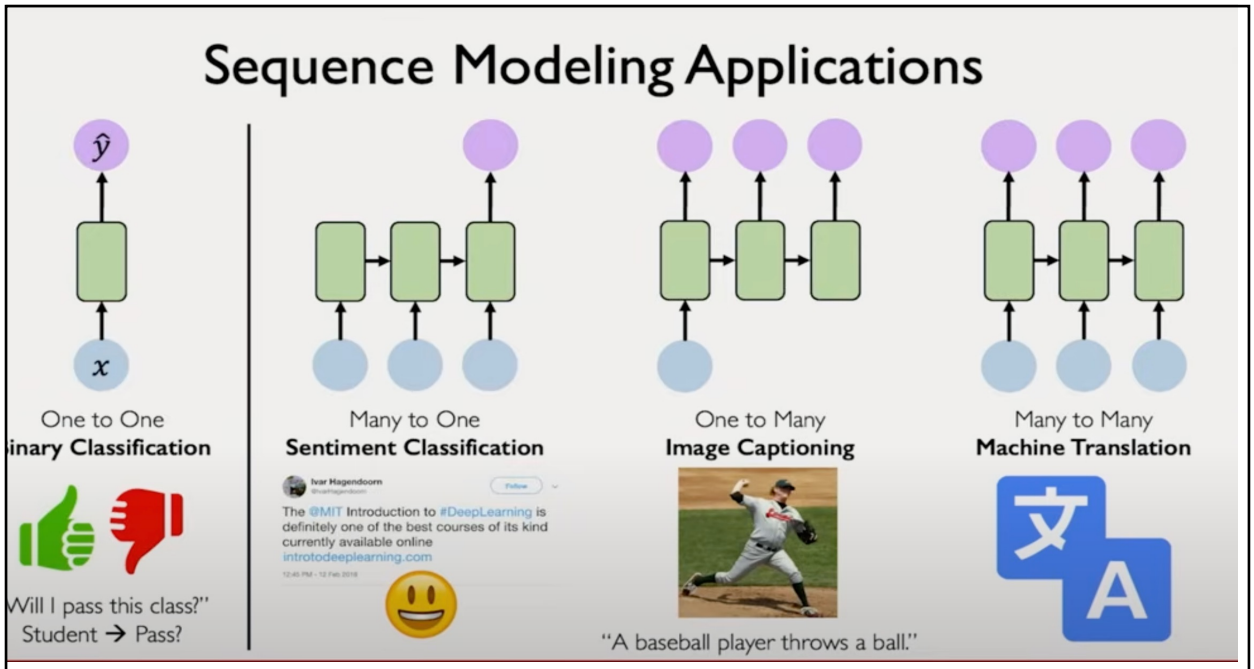
> Theo had been staring at the same sentence so long the letters bled together.
>
> (*A Doorway on Lenox* by Ovett Chapman)

Text



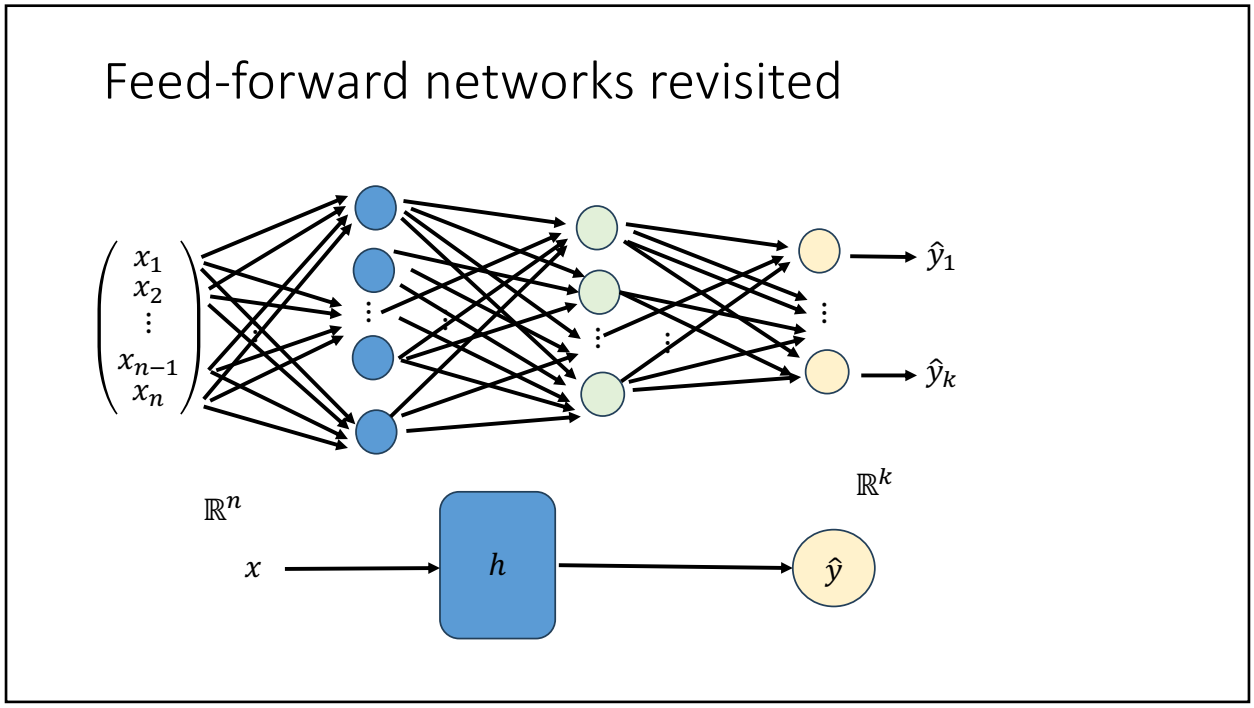
Video

6



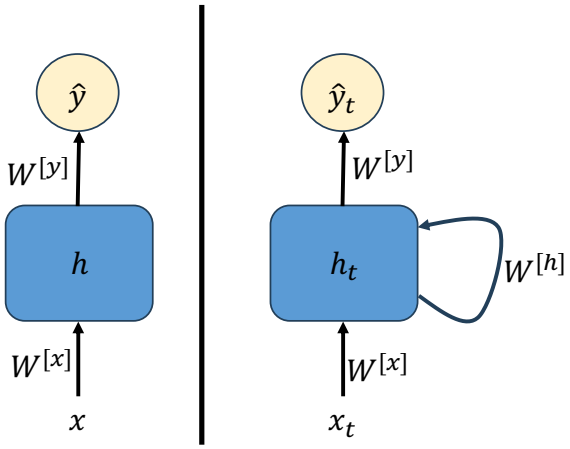
Source: MIT Introduction to Deep Learning 6.S191

7



8

Extending traditional FF networks to deal with sequences



$$h_t = G^{[h]}(W^{[x]}.\, x_t + W^{[h]}.\, h_{t-1})$$

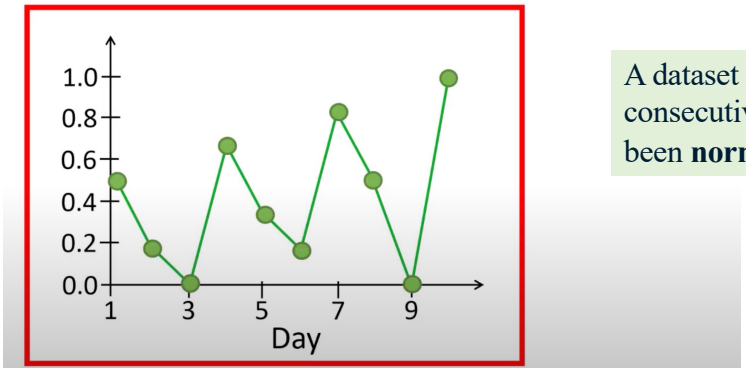
h_0 (the initial hidden state) is typically determined by initializing it to a zero vector.

x_t is the t^{th} element of the sequence (aka. the input at time step t)

9

Intuition of RNNs

	Mon	Tue	Wed	Thu	Fri	Mon	Tue	Wed	Thu	Fri
X	0.5	0.17	0.0	0.67	0.33	0.17	0.83	0.5	0.0	1.0



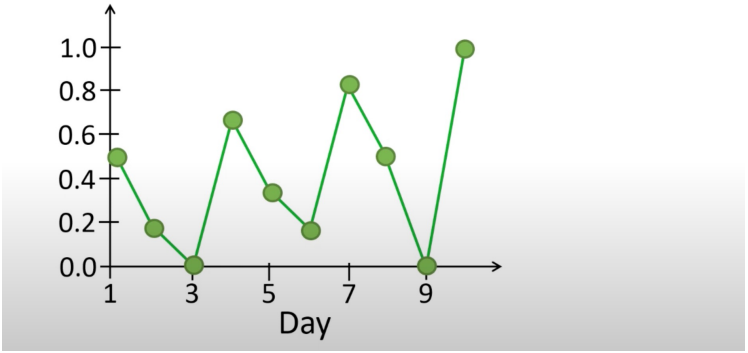
A dataset for stock prices (for 10 consecutive business days) that have been **normalized** into the [0,1] range.

Source: YouTube TileStats

10

Intuition of RNNs

	Mon	Tue	Wed	Thu	Fri	Mon	Tue	Wed	Thu	Fri
X	0.5	0.17	0.0	0.67	0.33	0.17	0.83	0.5	0.0	1.0

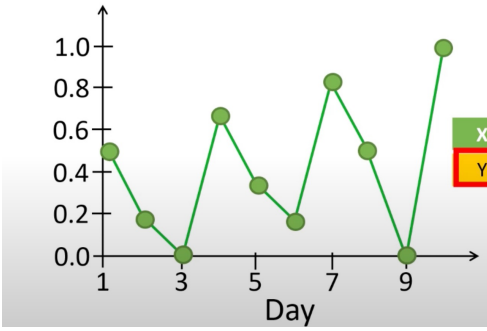


Source: YouTube TileStats

11

Intuition of RNNs

	Mon	Tue	Wed	Thu	Fri	Mon	Tue	Wed	Thu	Fri
X	0.5	0.17	0.0	0.67	0.33	0.17	0.83	0.5	0.0	1.0
Y		0.17	0.0	0.67	0.33	0.17	0.83	0.5	0.0	1.0



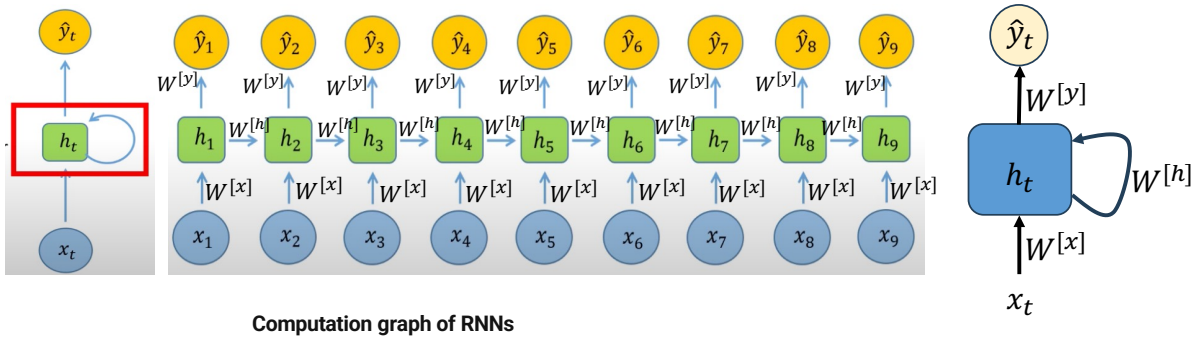
X	0.5	0.17	0.0	0.67	0.33	0.17	0.83	0.5	0.0	1.0
Y	0.17	0.0	0.67	0.33	0.17	0.83	0.5	0.0	1.0	

Source: YouTube TileStats

12

Intuition of RNNs

X	0.5	0.17	0.0	0.67	0.33	0.17	0.83	0.5	0.0	1.0
Y	0.17	0.0	0.67	0.33	0.17	0.83	0.5	0.0	1.0	

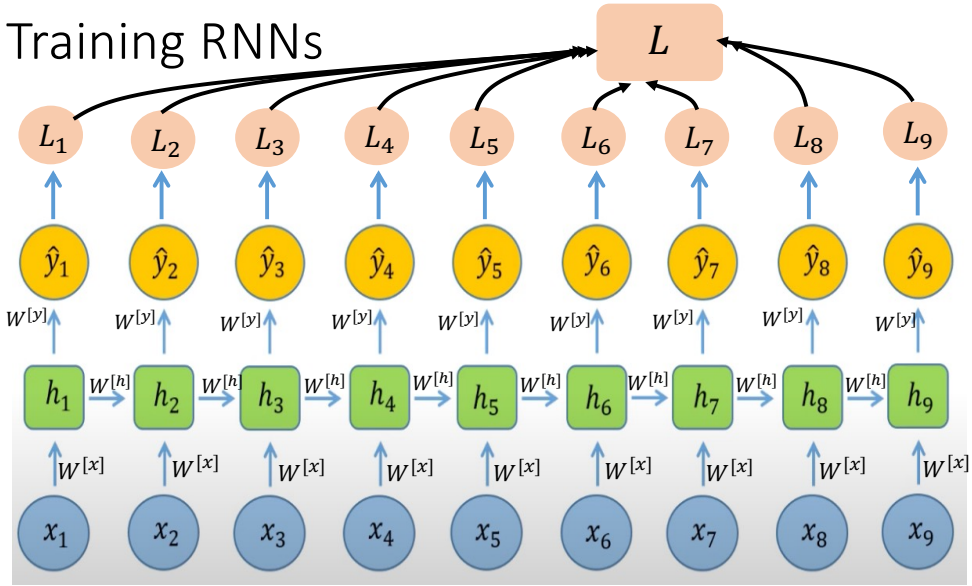


Computation graph of RNNs

Source: YouTube TileStats

13

Training RNNs

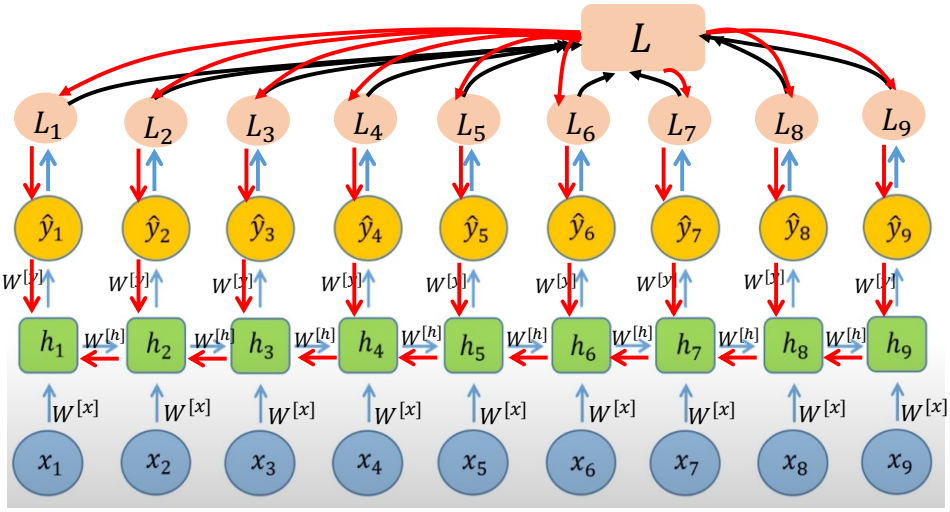


Computation graph of RNNs

forward pass

14

RNNs: Backpropagation through Time (BPTT)



Computation graph of RNNs

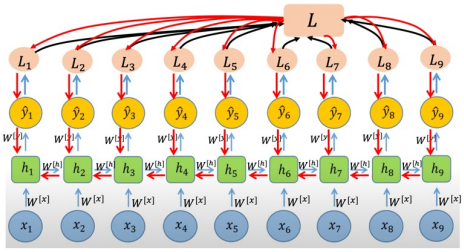
forward pass
backward pass

15

RNNs: Backpropagation through Time (BPTT)

The loss function:

$$L = \frac{1}{T}\sum_{t=1}^{T} L_t = \frac{1}{T}\sum_{t=1}^{T} loss(y_t, \hat{y}_t)$$



We'll show the hardest derivative:

$$\frac{\partial L}{\partial W^{[h]}} = \frac{1}{T}\sum_{t=1}^{T} \frac{\partial loss(y_t,\hat{y}_t)}{\partial W^{[h]}} = \frac{1}{T}\sum_{t=1}^{T} \frac{\partial loss(y_t,\hat{y}_t)}{\partial \hat{y}_t} \cdot \frac{\partial \hat{y}_t}{\partial h_t} \cdot \frac{\partial h_t}{\partial W^{[h]}}$$

The thi… ute the effect o… $_{t-1}))$

Very long recursive calculations of $\frac{\partial h_t}{\partial W^{[h]}}$ for all $t = 1, .., T$

$$\frac{\partial h_t}{\partial W^{[h]}} = \frac{\partial f(W^{[h]}, W^{[x]}, x_t, h_{t-1})}{\partial W^{[h]}} + \frac{\partial f(W^{[h]}, W^{[x]}, x_t, h_{t-1})}{\partial h_{t-1}} \cdot \frac{\partial h_{t-1}}{\partial W^{[h]}}$$

16

RNNs: Issues and Challenges

- Intensive computation (esp. for long sequences)
  - Truncation: Approximating the computation of $\frac{\partial h_t}{\partial W^{[h]}}$ by going back only τ steps.
- Numerical instability:
  - High powers of matrices can lead to divergent or vanishing eigenvalues.

  → exploding or vanishing gradients

Many values > 1:
Exploding gradients
→ Gradient clipping

Many values < 1:
Vanishing gradients
- Activation function
- Parameter initialization
- Network architecture

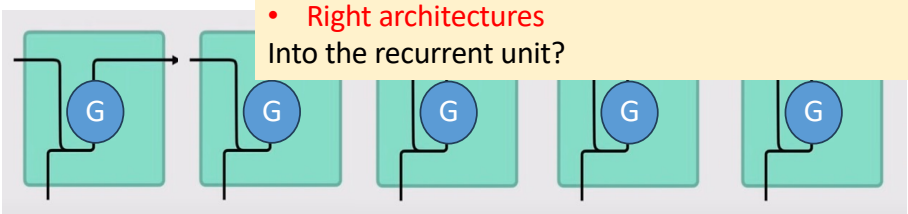
17

RNNs: Limitations

- Encoding bottleneck (not all data are numerical, e.g., text, audio, video, etc.)
- Slow computation (due to no parallelization)
- No long memory

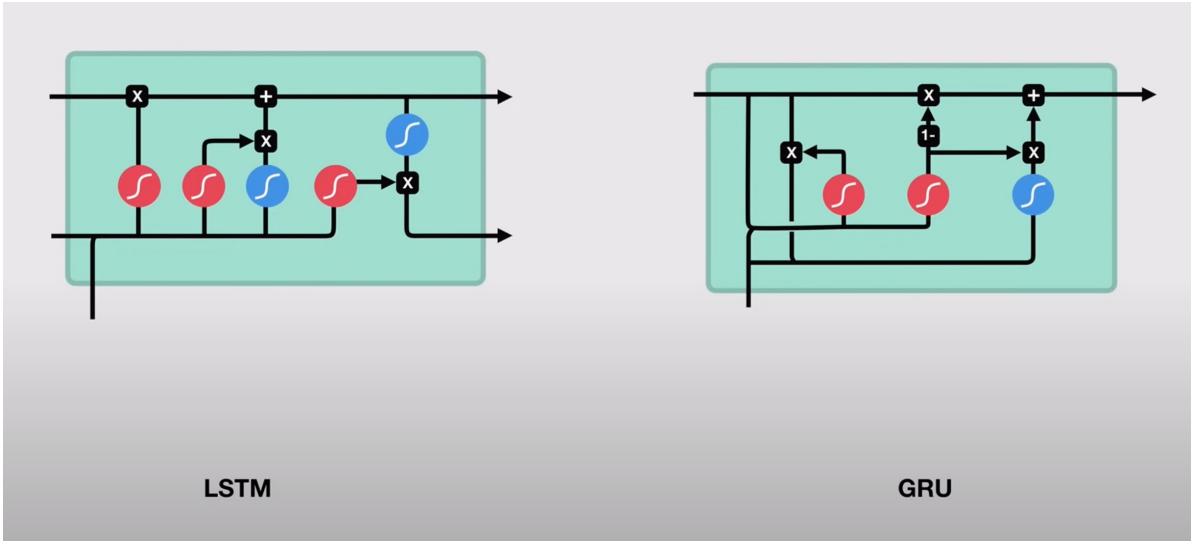
Question: Can we overcome (some of) these limitations by introducing the:
- Right activation functions
- Right architectures

Into the recurrent unit?



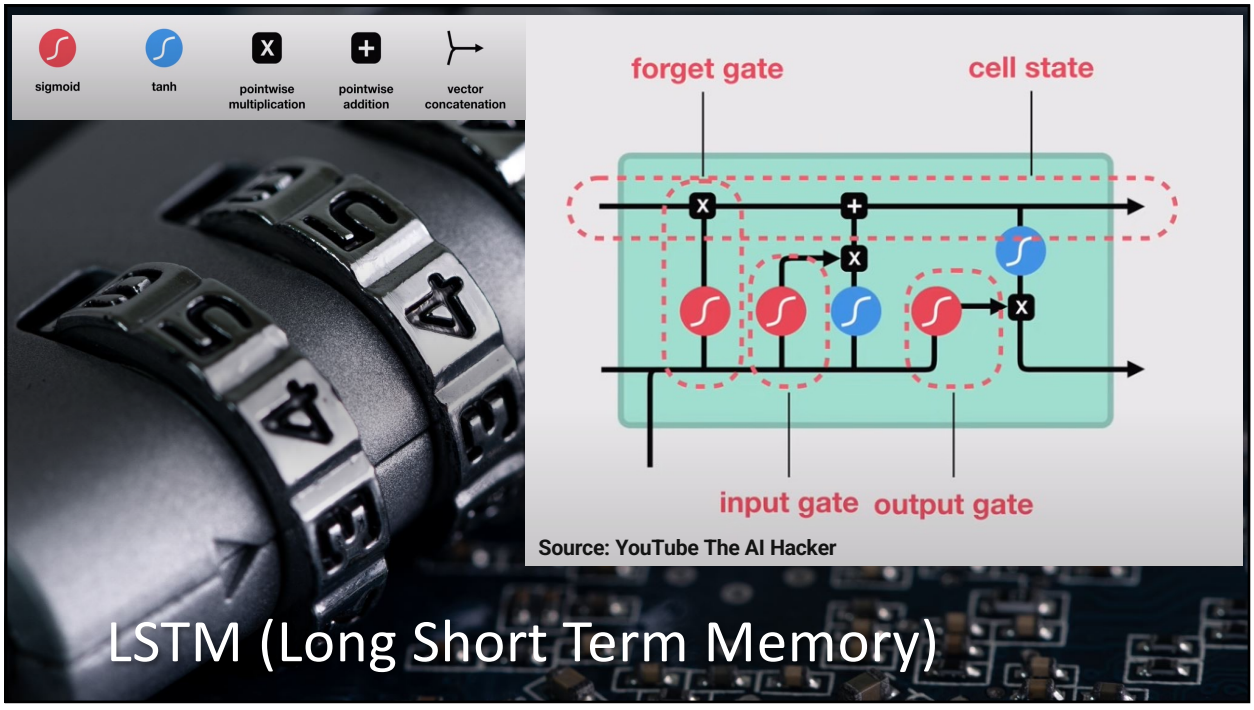
18

LSTM & GRU

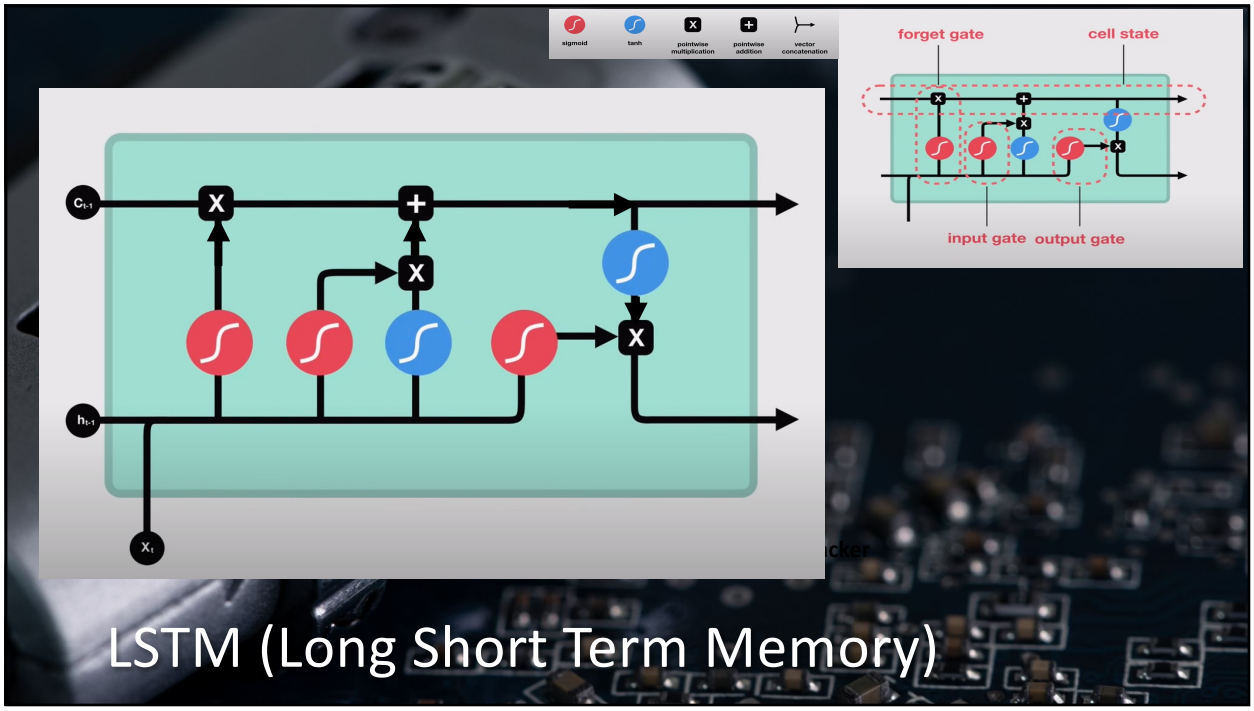


Source: YouTube The AI Hacker

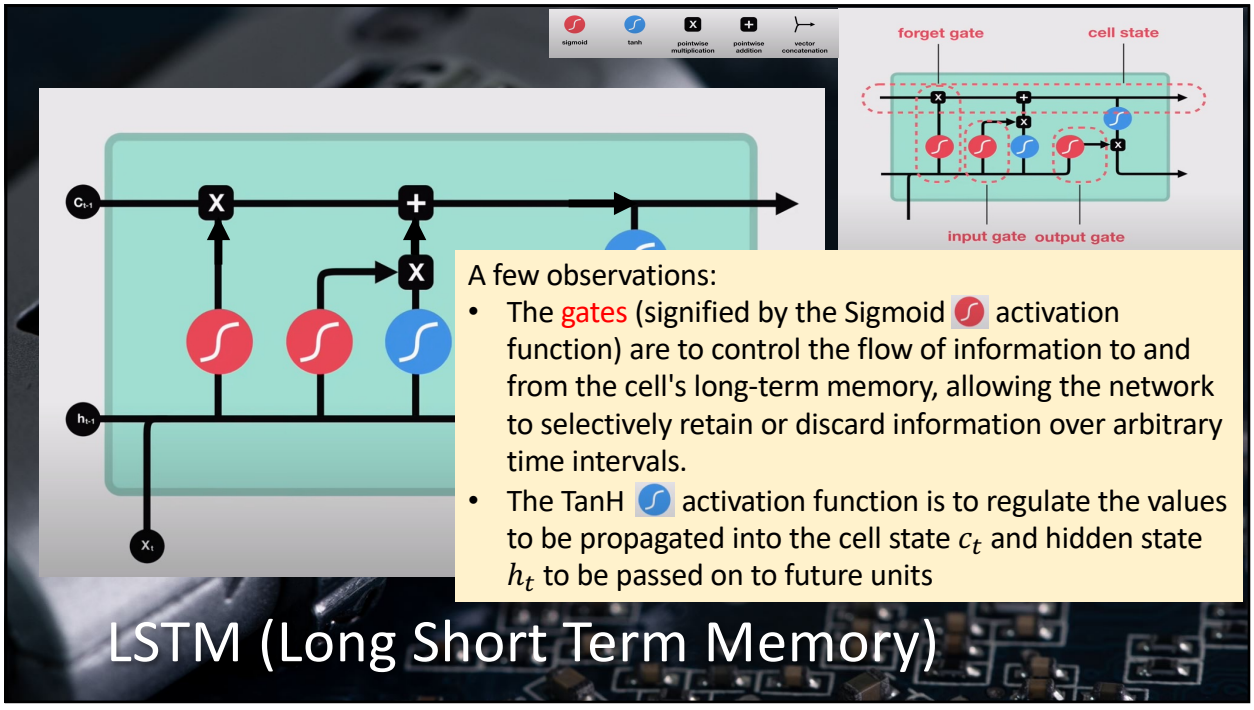
19



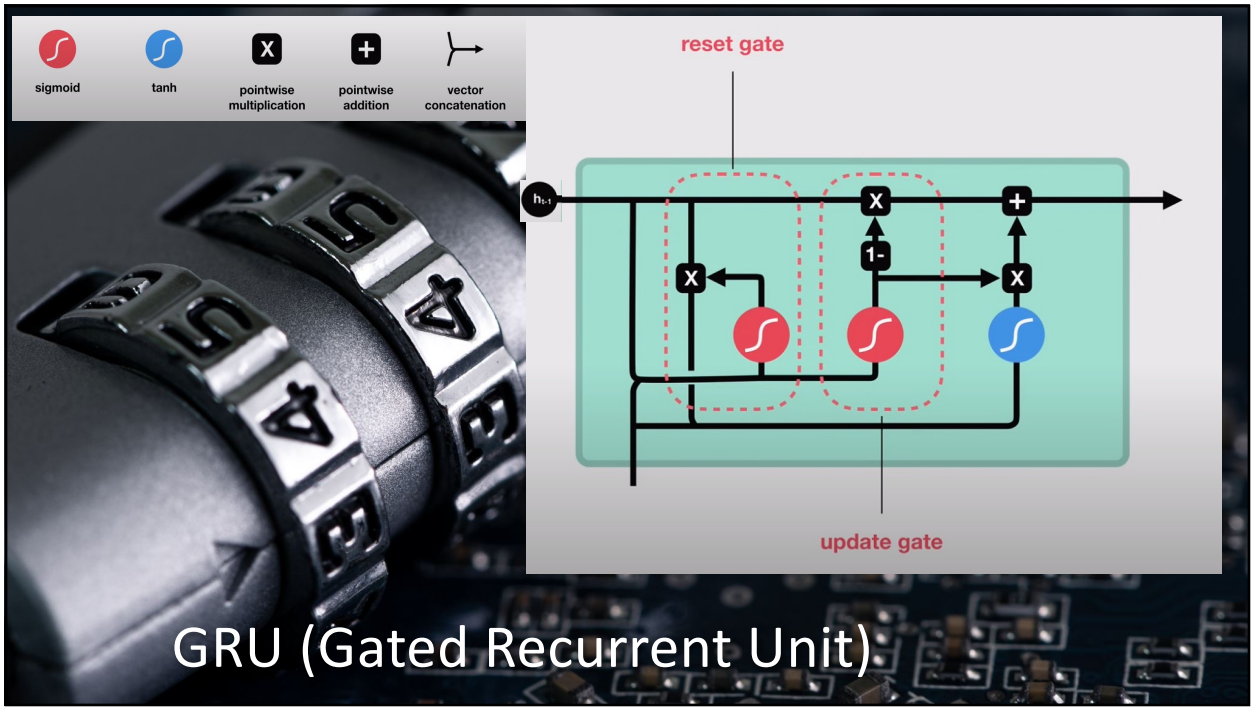
20


sigmoid
tanh
pointwise multiplication
pointwise addition
vector concatenation
forget gate
cell state
input gate
output gate
LSTM (Long Short Term Memory)

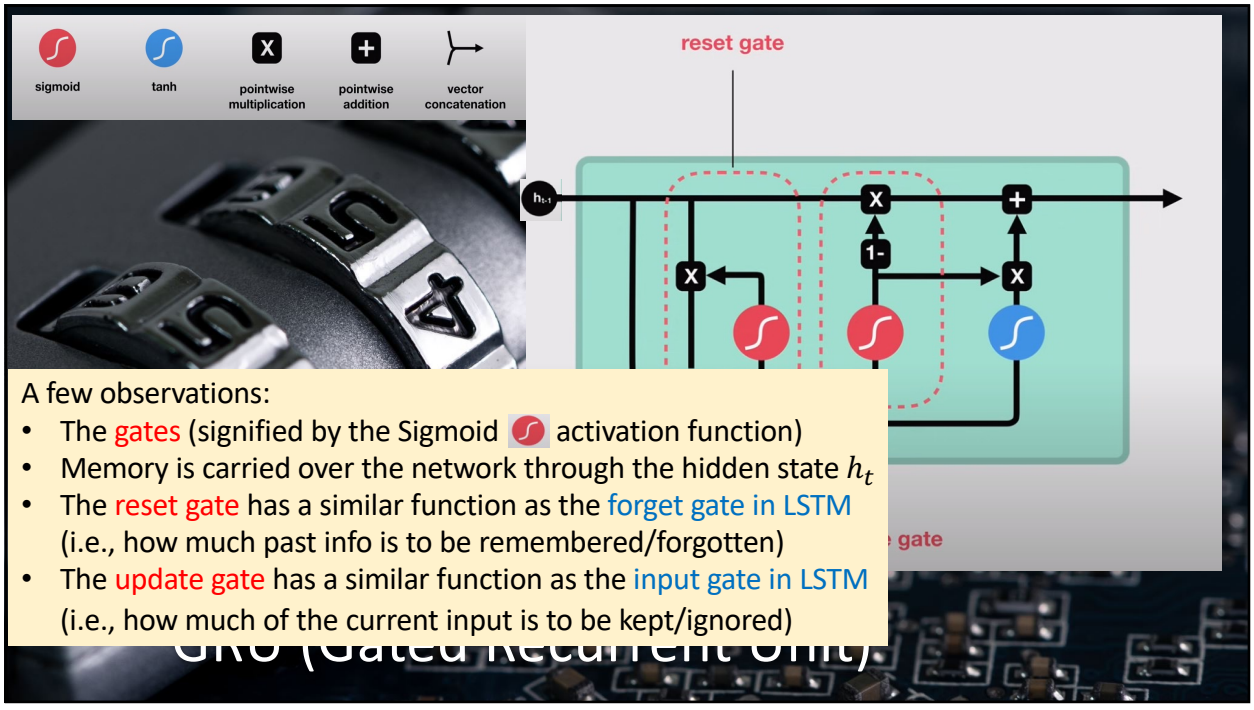
21


sigmoid
tanh
pointwise multiplication
pointwise addition
vector concatenation
forget gate
cell state
input gate
output gate
A few observations:
• The gates (signified by the Sigmoid activation function) are to control the flow of information to and from the cell's long-term memory, allowing the network to selectively retain or discard information over arbitrary time intervals.
• The TanH activation function is to regulate the values to be propagated into the cell state c_t and hidden state h_t to be passed on to future units
LSTM (Long Short Term Memory)

22


sigmoid
tanh
pointwise multiplication
pointwise addition
vector concatenation
reset gate
update gate
GRU (Gated Recurrent Unit)

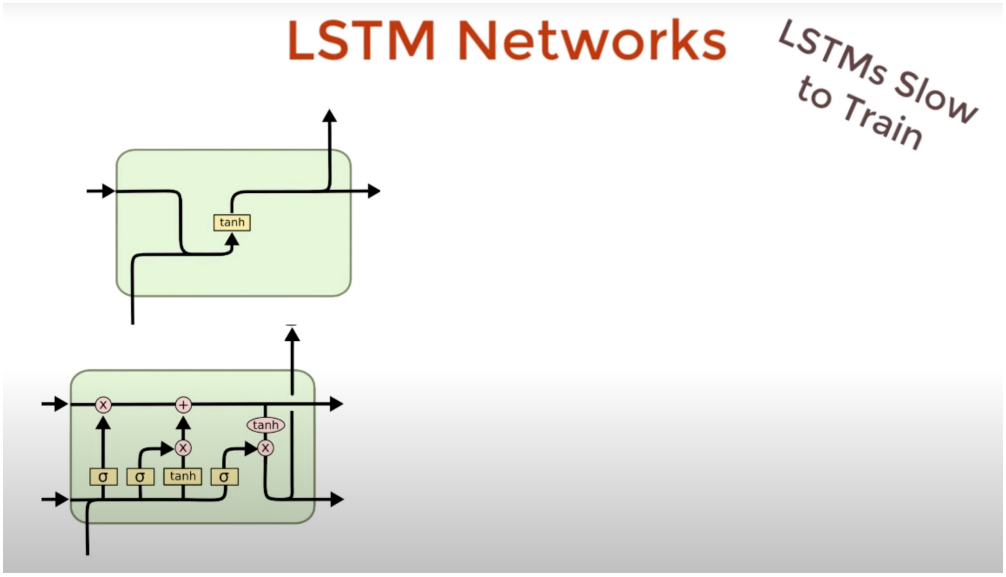
23


sigmoid
tanh
pointwise multiplication
pointwise addition
vector concatenation
reset gate
A few observations:
• The gates (signified by the Sigmoid activation function)
• Memory is carried over the network through the hidden state h_t
• The reset gate has a similar function as the forget gate in LSTM (i.e., how much past info is to be remembered/forgotten)
• The update gate has a similar function as the input gate in LSTM (i.e., how much of the current input is to be kept/ignored)

24

LSTM drawbacks

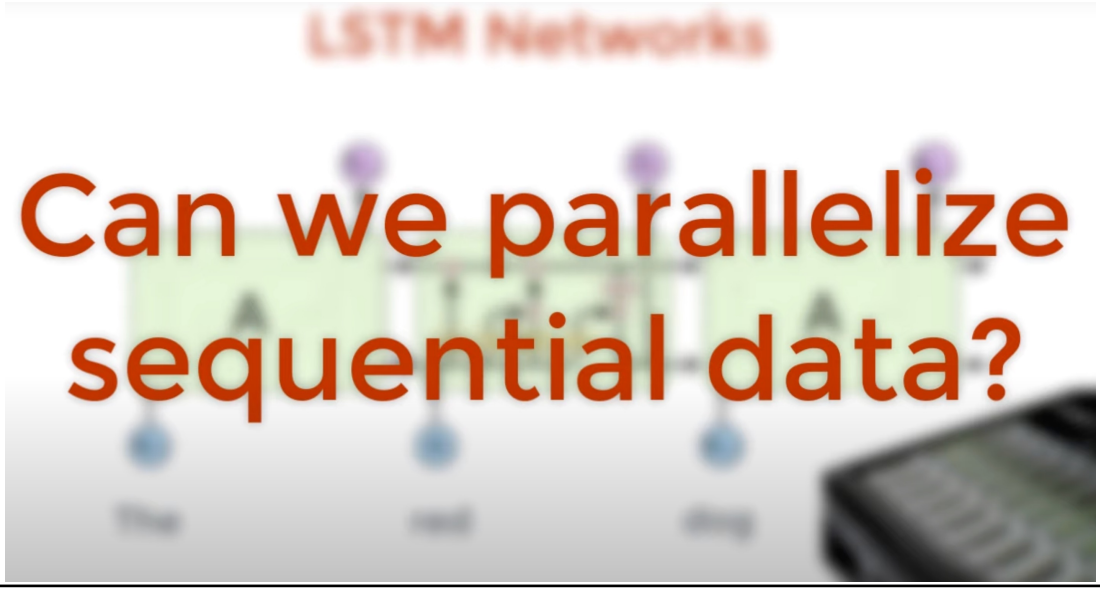
Source: YouTube CodeEmporium



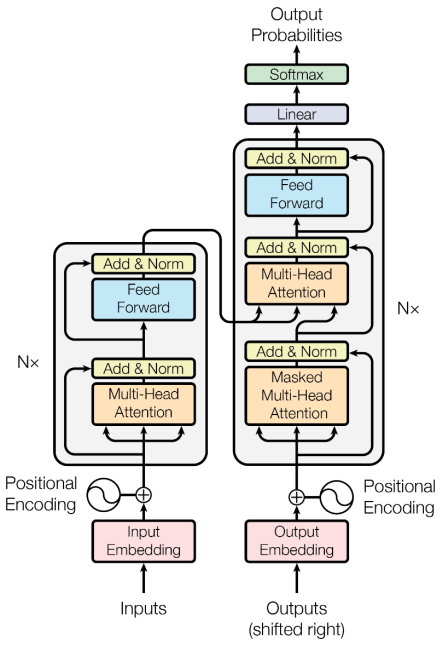
25

LSTM drawbacks

Source: YouTube CodeEmporium



26



Enter the Transformer (Vaswani et al., 2017)

"Attention Is All You Need"

Figure 1: The Transformer - model architectu ء

27

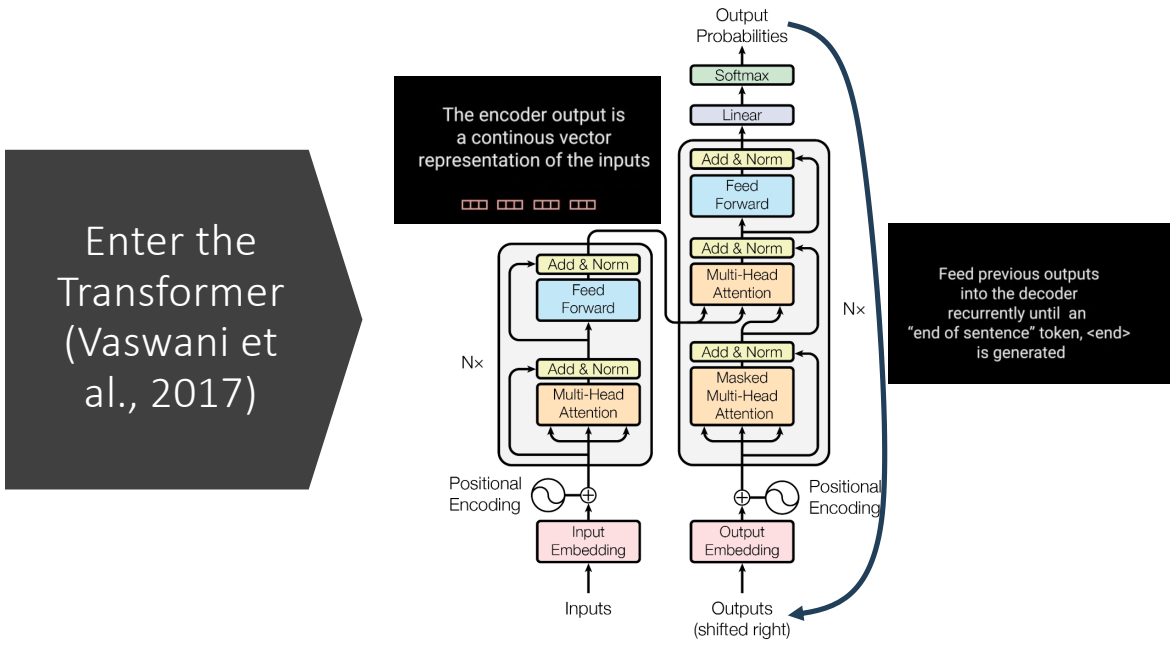


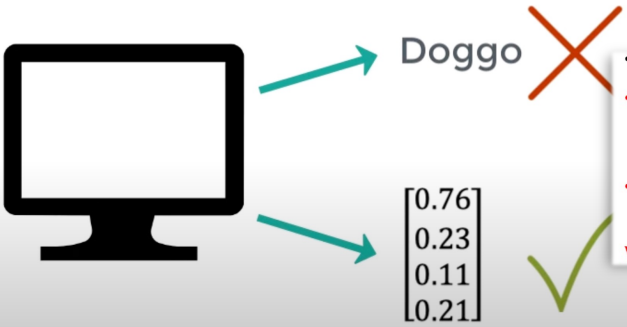
Figure 1: The Transformer - model architecture.

28

Transformer Components

Input Embedding

Remember our one-hot encoding in Week 5?



- Given a 10,000-word English dictionary (e.g., the top-10,000 English words seen in emails)
- The feature vector:

$$X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{10,000} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = 1\{ \begin{bmatrix} a \\ aardvark \\ aardwolf \\ \vdots \\ zymurgy \end{bmatrix} \}$$

where 1{} is again the indicator function.

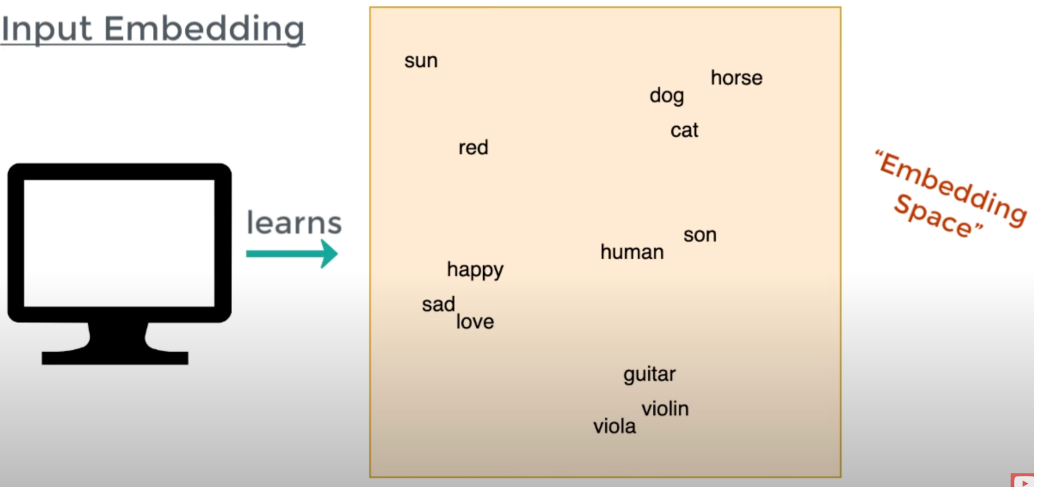
- (Very) high dimension
- Sparse matrix

Source: YouTube CodeEmporium

29

Transformer Components

Input Embedding

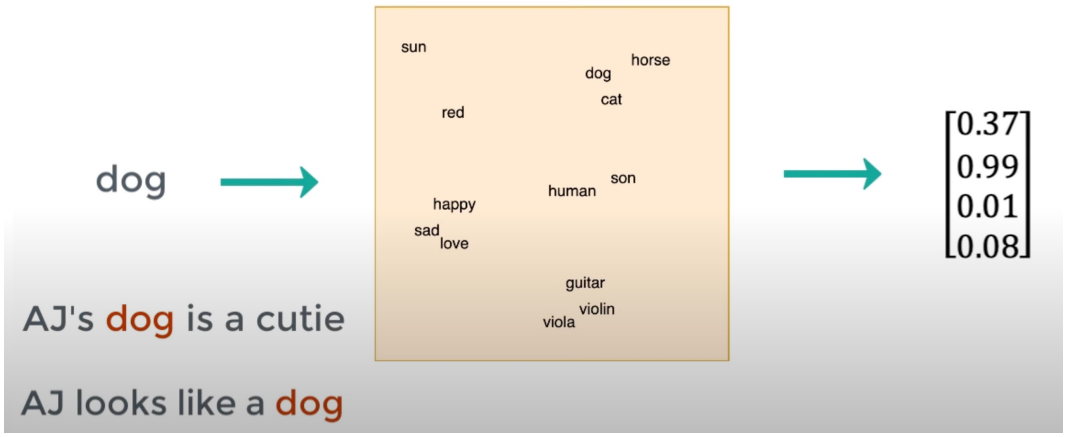


Source: YouTube CodeEmporium

30

Transformer Components

Input Embedding



Source: YouTube CodeEmporium

31

Transformer Components

Positional Encoder

:vector that gives context based on position of word in sentence

AJ's dog is a cutie → Position 2

AJ looks like a dog → Position 5

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

Source: YouTube CodeEmporium

32

Transformer Components

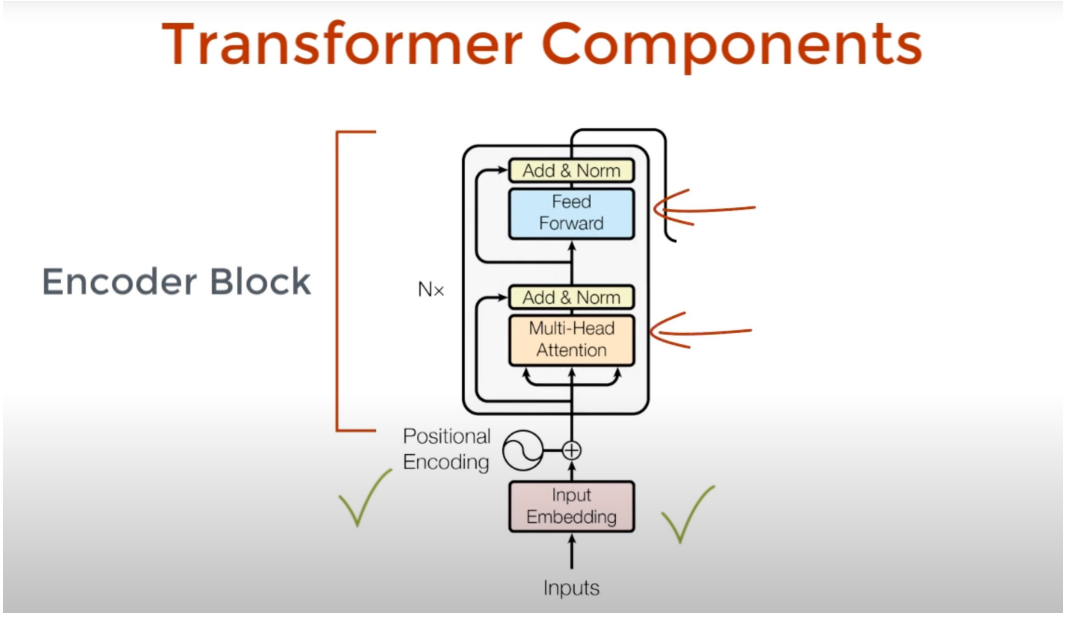
Positional Encoder :vector that gives context based on position of word in sentence

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

Source: YouTube CodeEmporium

33

Transformer Components

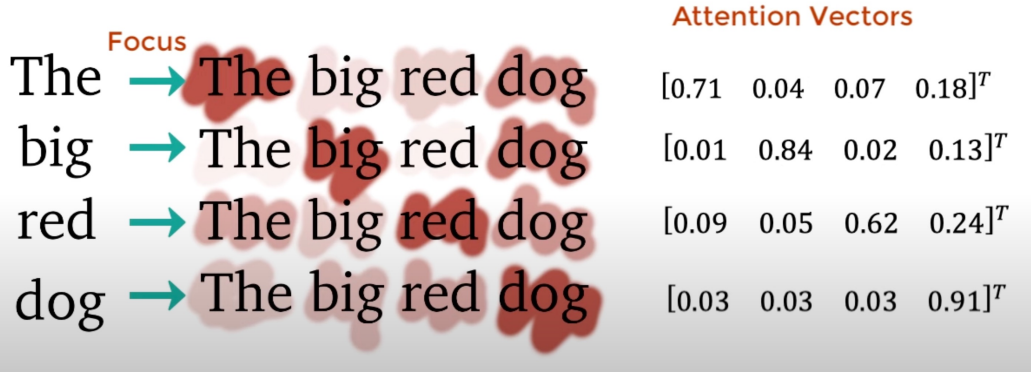


Source: YouTube CodeEmporium

34

Transformer Components

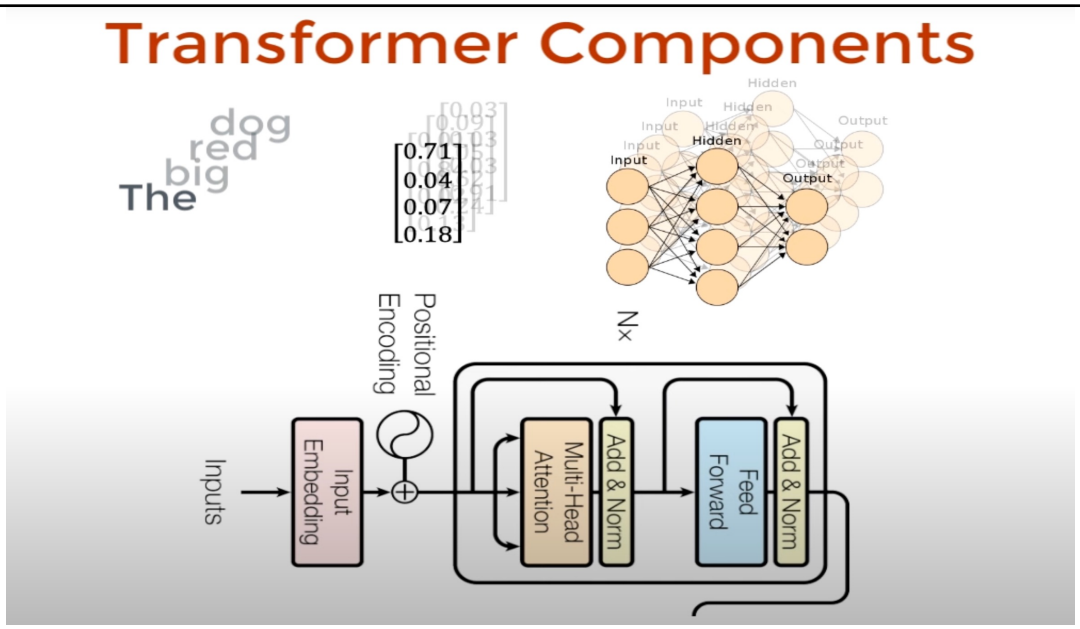
Attention : What part of the input should we focus?



Source: YouTube CodeEmporium

35

Transformer Components

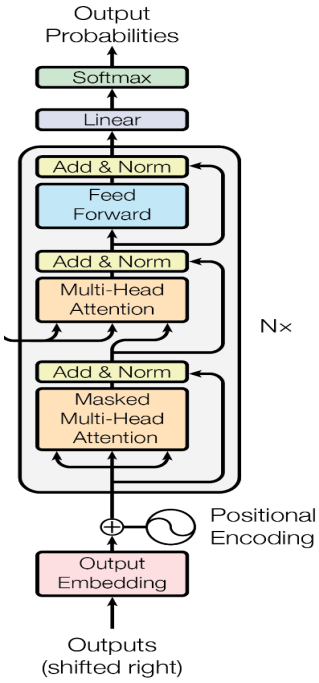


Source: YouTube CodeEmporium

36

Transformer Components

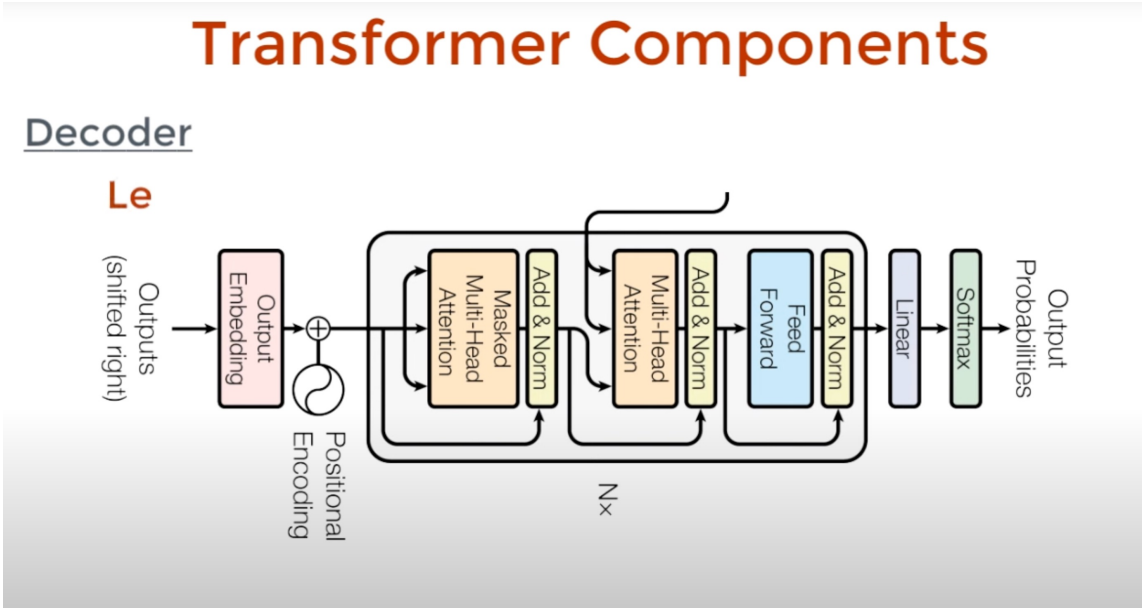
Decoder



Le gros chien rouge

37

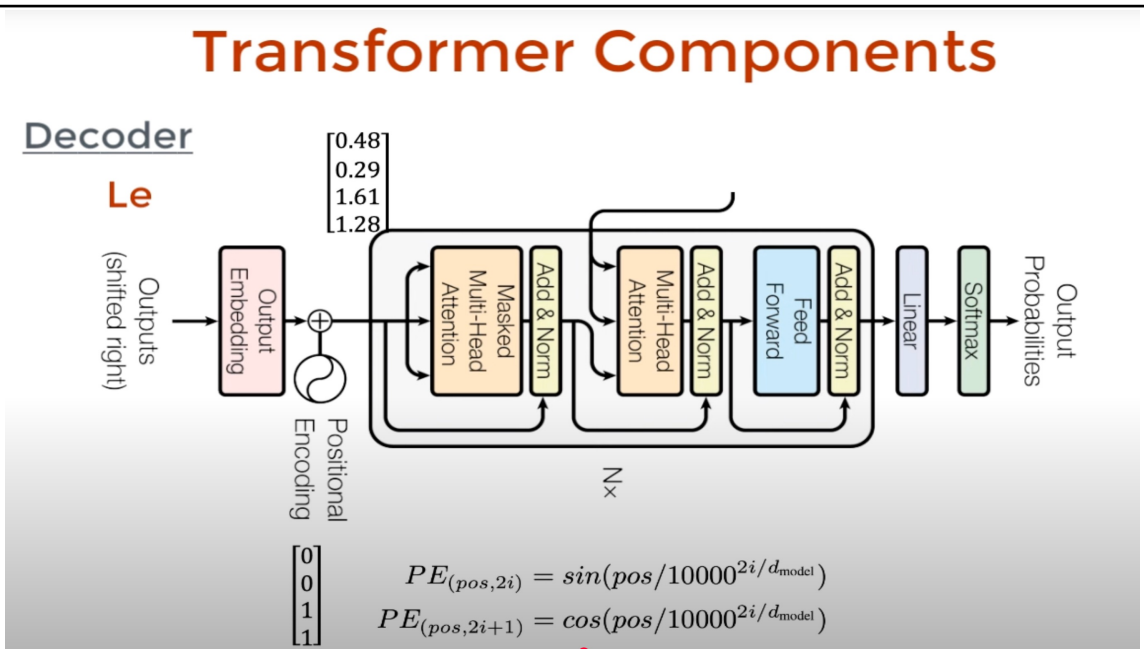
Transformer Components



Source: YouTube CodeEmporium

38

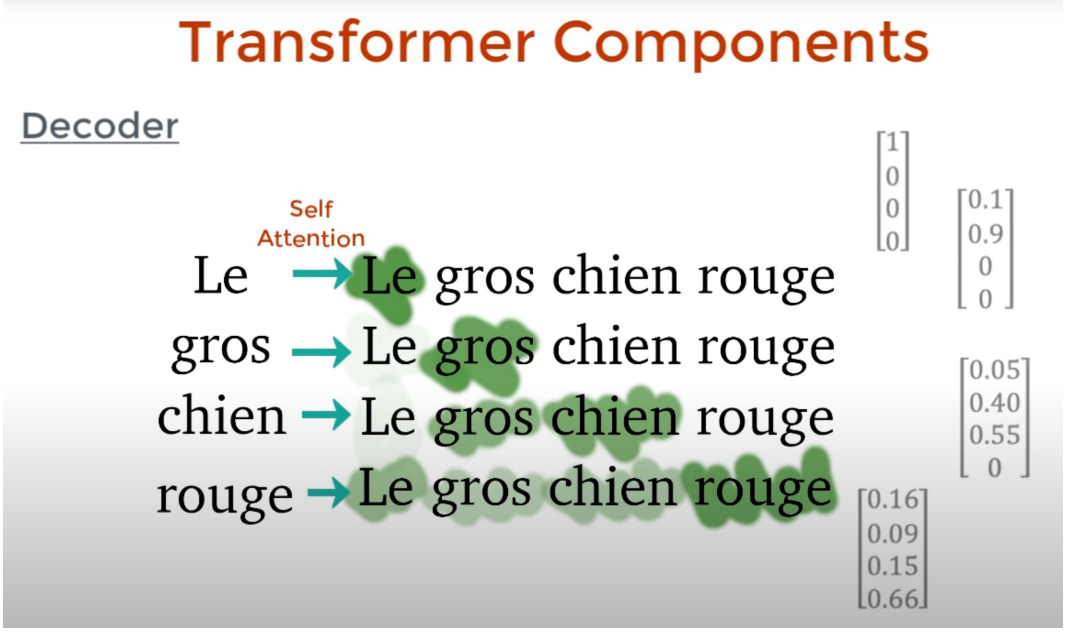
Transformer Components



Source: YouTube CodeEmporium

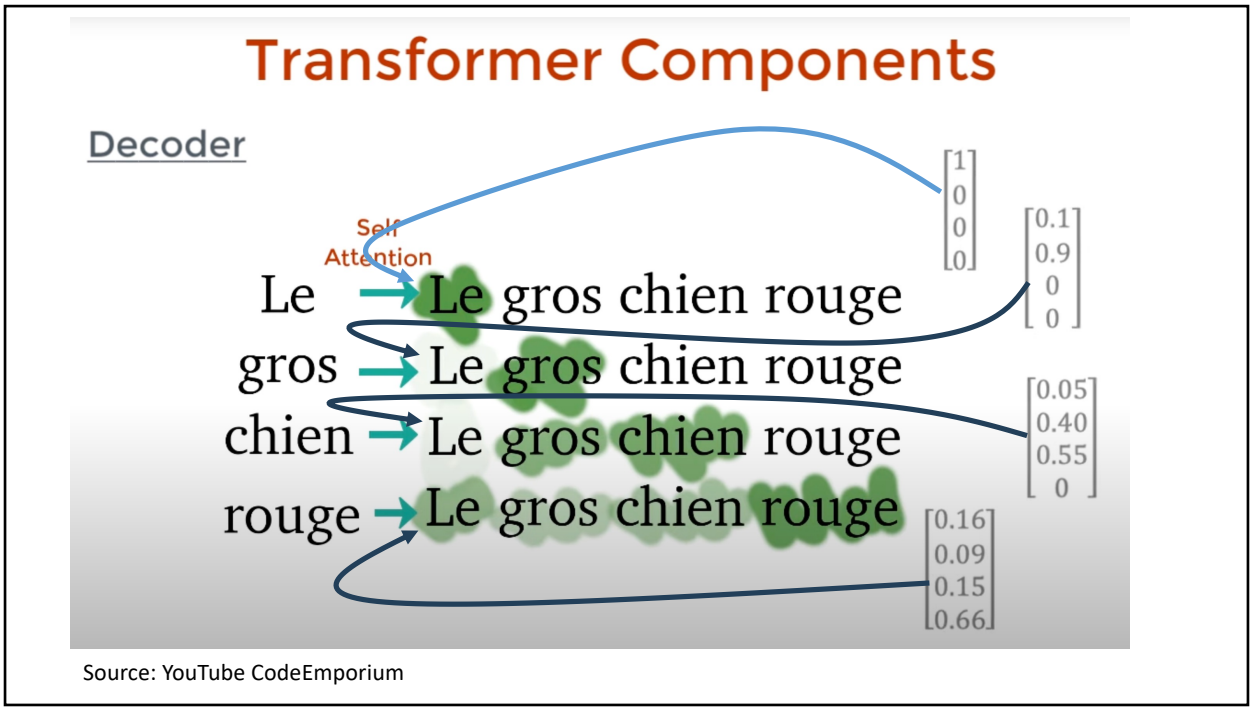
39

Transformer Components



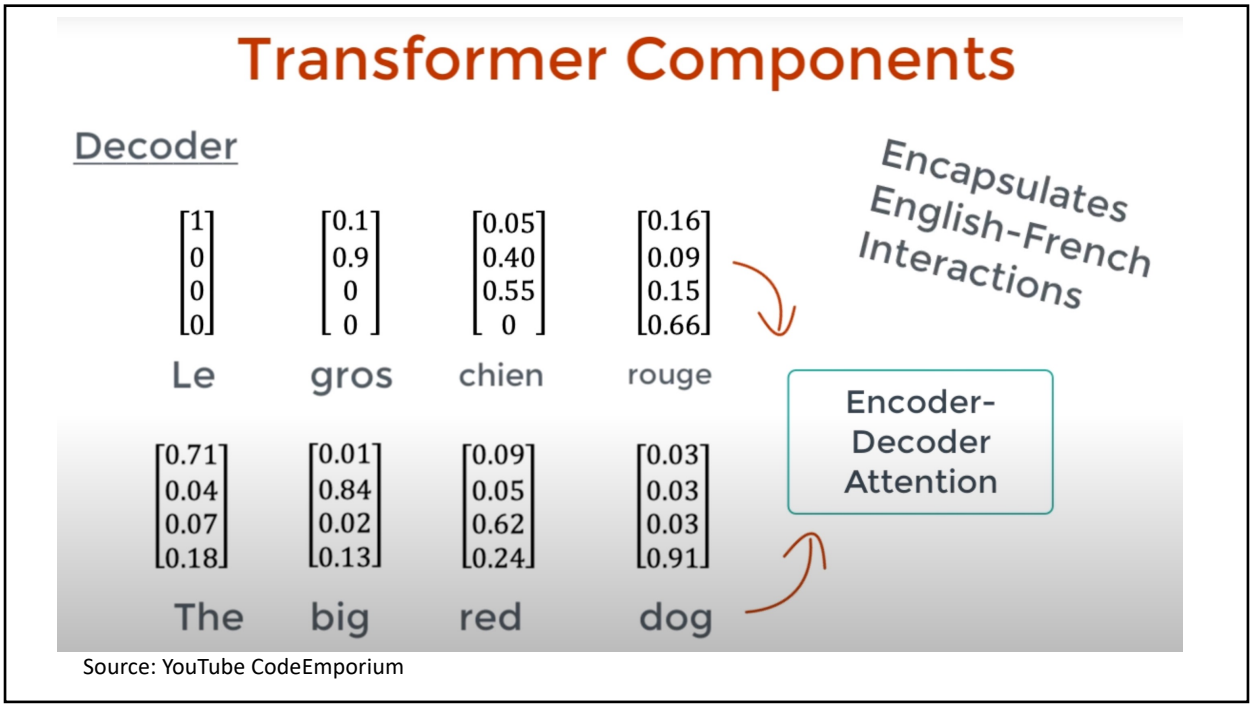
Source: YouTube CodeEmporium

40



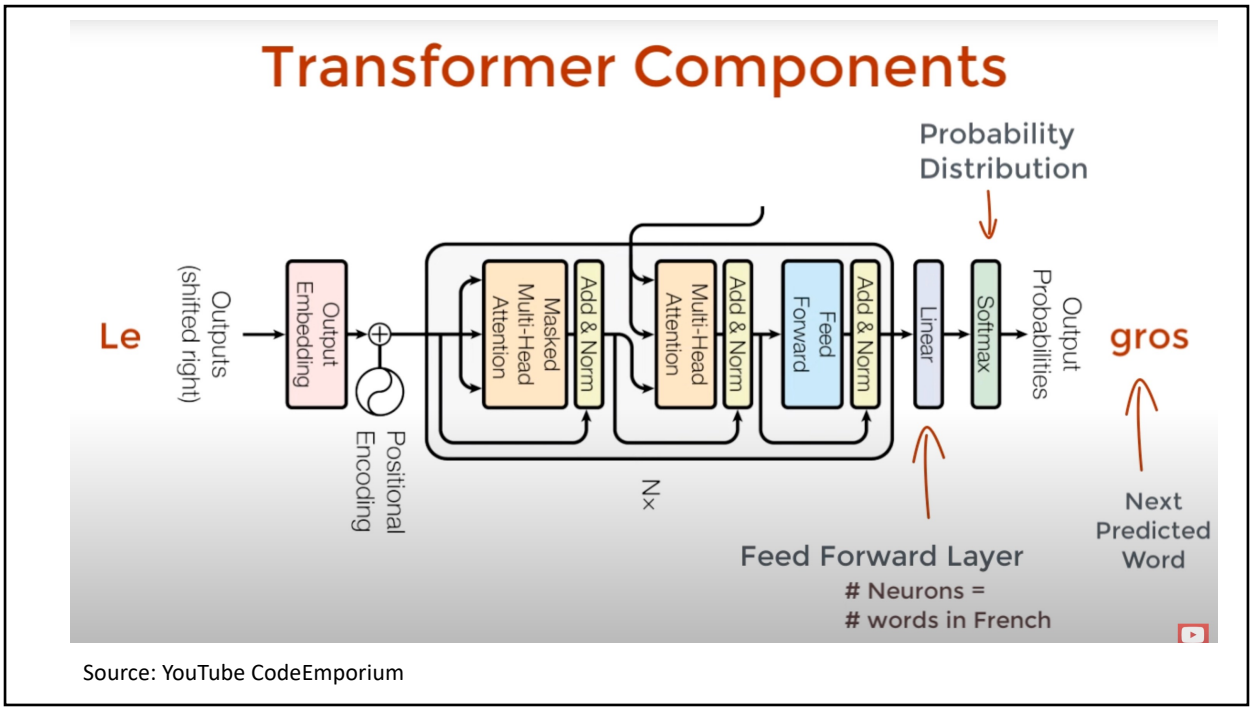
Source: YouTube CodeEmporium

41



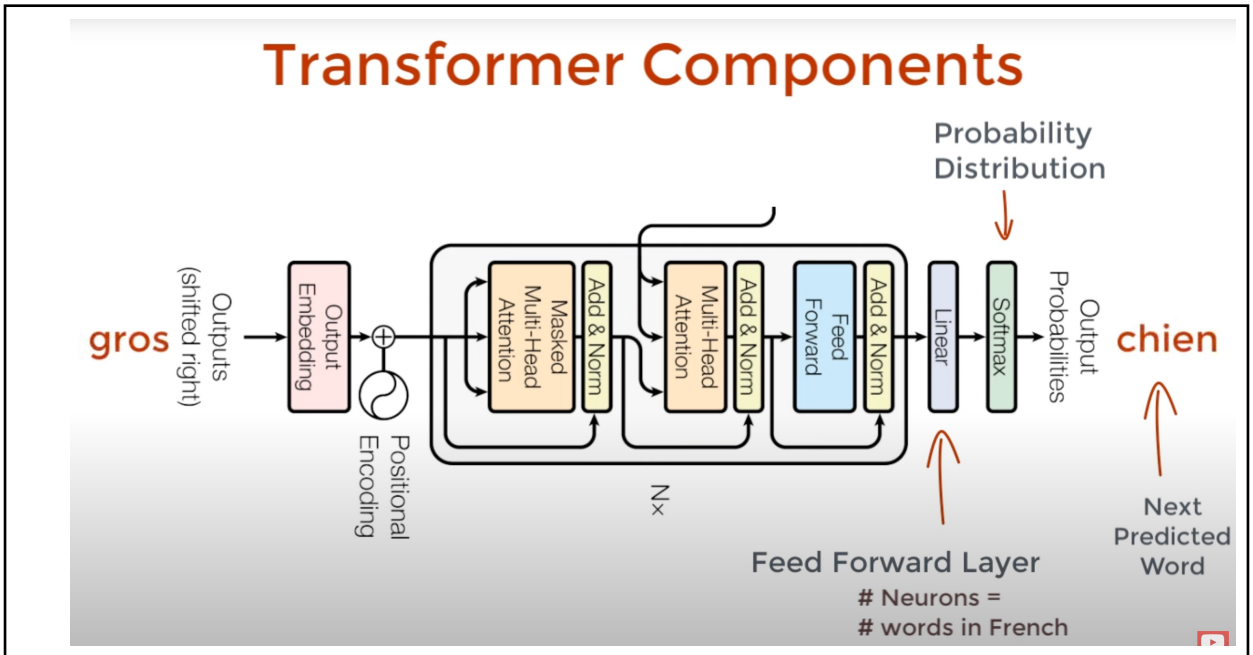
Source: YouTube CodeEmporium

42



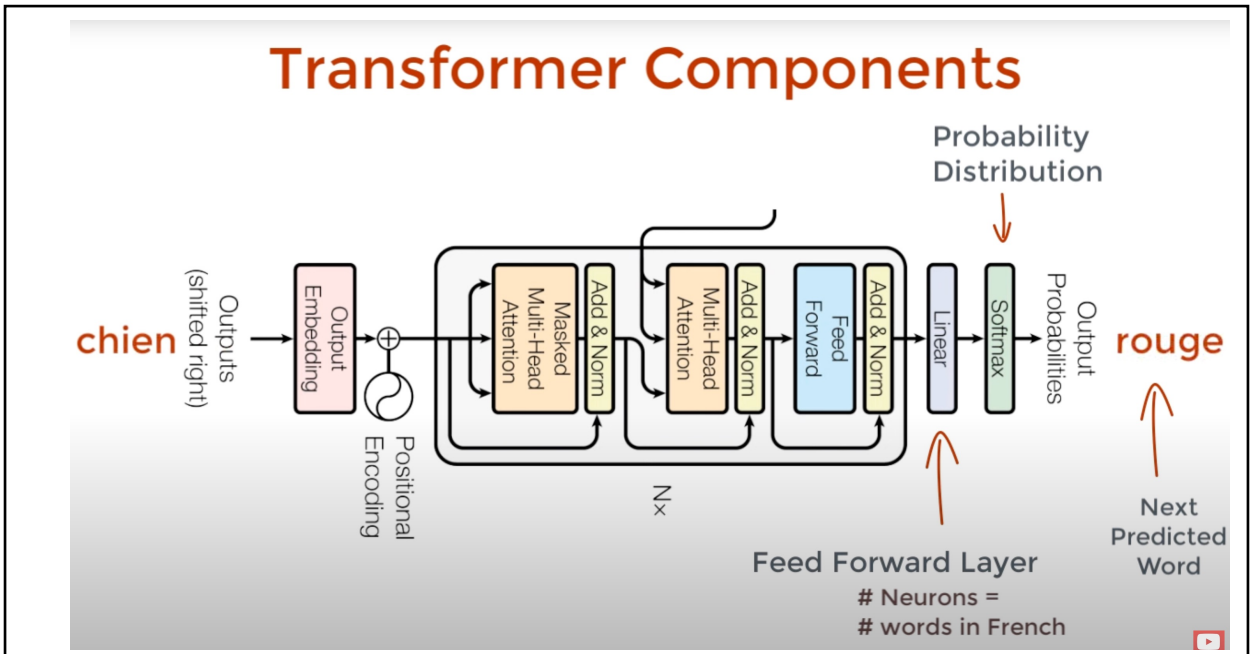
Source: YouTube CodeEmporium

43



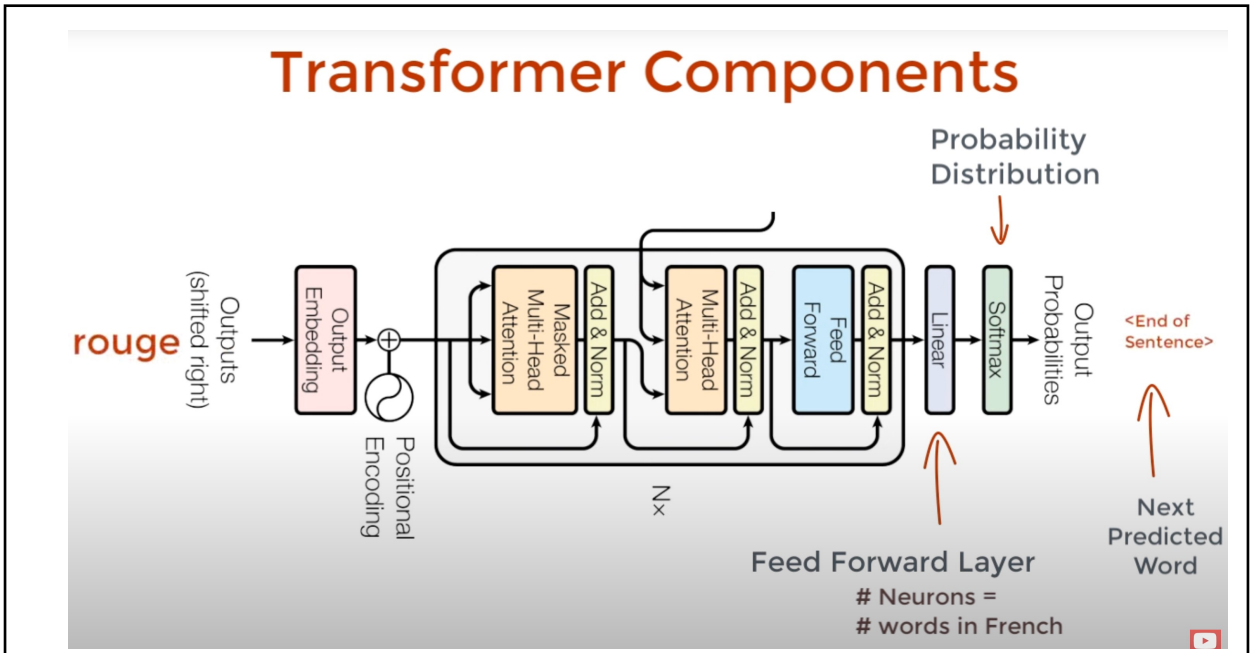
Source: YouTube CodeEmporium

44



Source: YouTube CodeEmporium

45



Source: YouTube CodeEmporium

46

Summary

- DL architectures for sequence data
- RNN
  - Encoding bottleneck
  - Slow computation (due to no parallelization)
  - No long memory
- LSTM & GRU
  - Advanced architectures for RNN
  - Address the issues of exploding/vanishing gradients
- Transformer
  - Foundation of most advanced DL systems
  - Architectural foundation for LLMs/LFMs
  - The combination of self-supervised learning (SSL) and transformers allows models to learn from large amounts of unlabeled data.

47