Long Short-Term Memory Networks

Dinesh Kumar Vishwakarma, Ph.D.

ASSOCIATE PROFESSOR, DEPARTMENT OF INFORMATION TECHNOLOGY

DELHI TECHNOLOGICAL UNIVERSITY, DELHI.

Webpage: http://www.dtu.ac.in/Web/Departments/InformationTechnology/faculty/dkvishwakarma.php

Email: dinesh@dtu.ac.in

Outlines

- Introduction of LSTM
- Structure of LSTM
- Training of LSTM
- Summary of LSTM
- Use cases

Applications of LSTM includes:

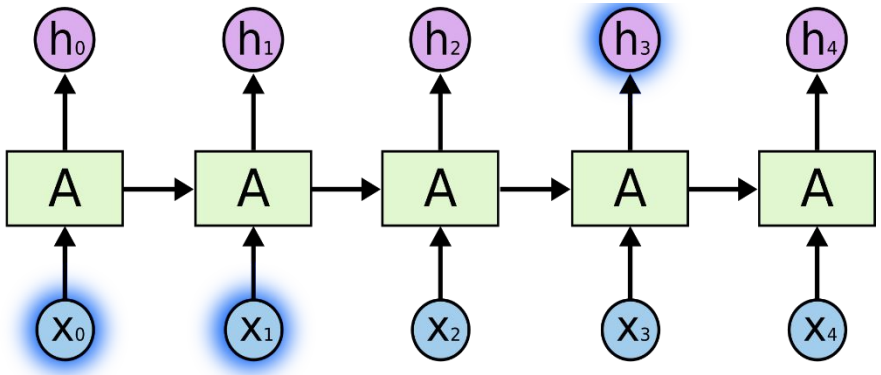
- Time series prediction
- Speech recognition
- Rhythm learning
- Music composition
- Grammar learning
- Handwriting recognition
- Human action recognition
- Sign language translation
- Protein homology detection
- Predicting subcellular localization of proteins
- Time series anomaly detection
- Several prediction tasks in the area of business process management
- Airport passenger management
- Short-term traffic forecast
- Market Prediction

What is LSTM?

- Long Short-Term Memory networks are usually called **“LSTMs”.** In 1997 by Sepp Hochreiter and Jürgen Schmidhube.
- A **special kind** of Recurrent Neural Networks which are **capable** of **learning long-term dependencies.**
- **Long-term dependencies** means there are some models that requires the **recent** and **past data** to perform operations such as a **language model** trying to predict the **next word** based on the previous ones. If we are trying to **predict** the **last word** in the sentence say **“The clouds are in the sky”**.
- Here, the last word ends up being **sky** all the time. In such cases, the gap between the **past information** and the **current requirement** can be **bridged** using **Recurrent Neural Networks.**
- **RNN** suffers from **Vanishing** and **Exploding Gradients** problems and this makes LSTM networks handle **long-term dependencies** easily.

Problem with RNN

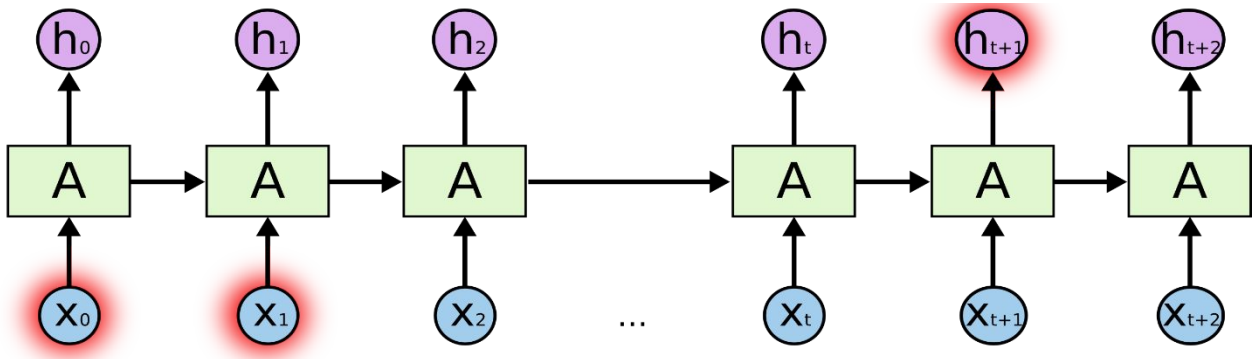
- The Problem of Long-Term Dependencies
  - RNN Suffers with Long-Term Dependencies such as:



The colour of tree is "Green"

Performs Good

Distance between actual context and prediction is very **less**

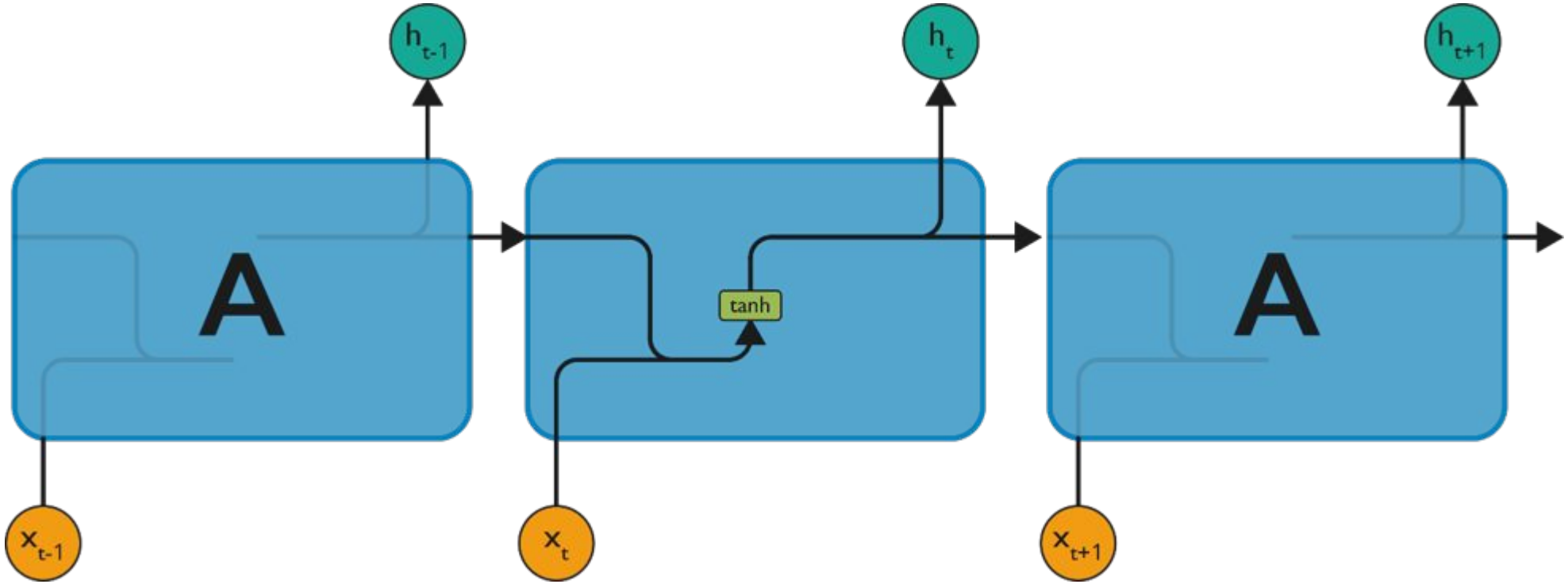


I live in Kashmir, India….where I like the weather and generally it is "cold"

Performs Poor

LSTM Structure

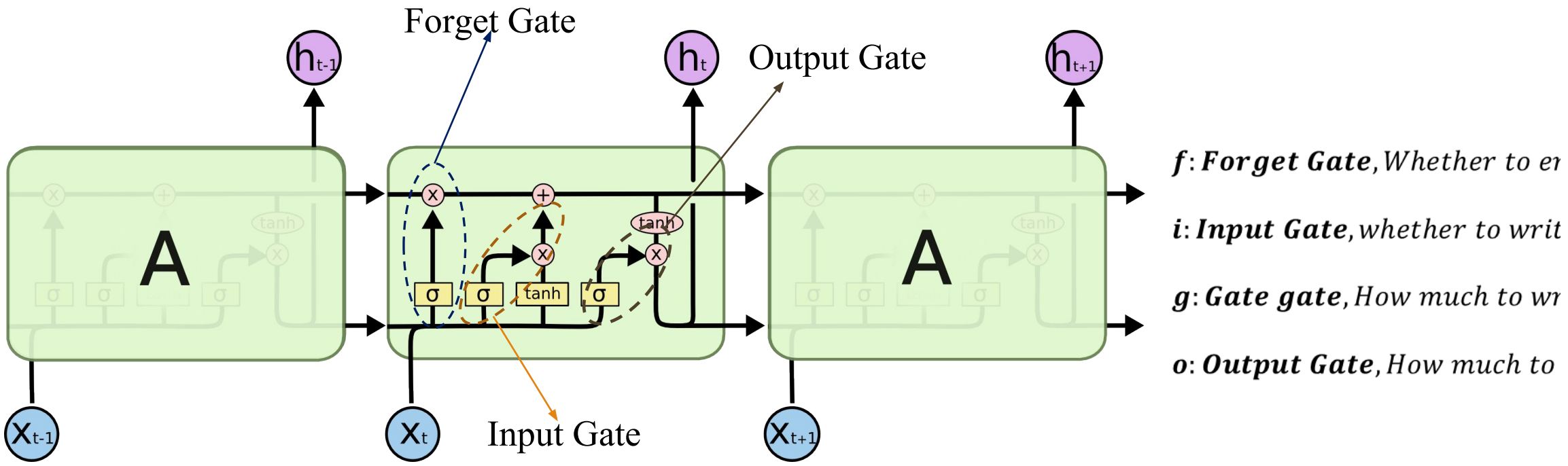
- LSTM have **chain-like** neural network layer. In a standard ***RNN***, the repeating module consists of one **single function** as shown in the below figure.



- ***tanh*** is squashing function, which makes values between -1 to 1.

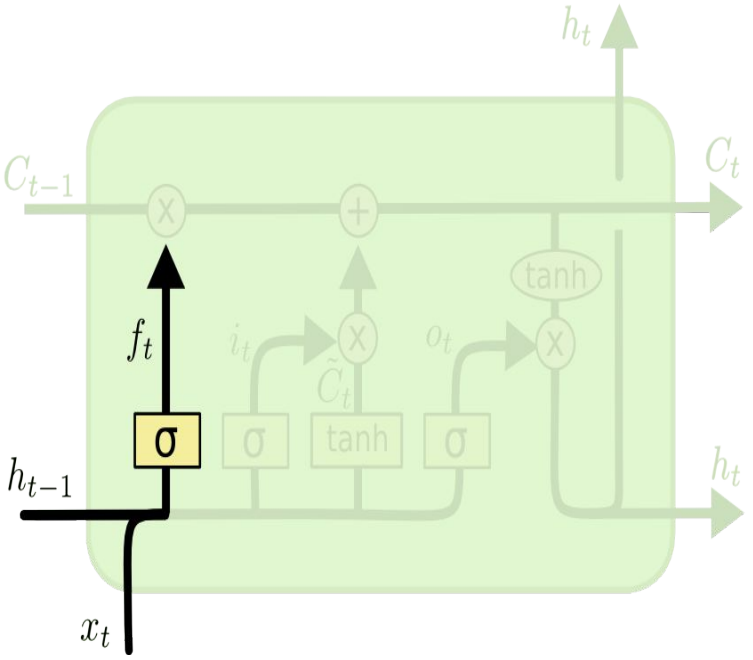
LSTM Structure…

- Each of the functions in the layers has their own structures. The **cell state** is the ***horizontal line*** and it acts like a conveyer belt carrying certain data linearly across the data channel. A simple LSTM consist of 4-gates.



LSTM Structure…

- The **1st step** is to **identify,** the information which is **not required** and will be **thrown away** from the **cell state.** It is done by a **sigmoid layer** called as **forget gate layer**.
- **Forget Gate**: After getting the output of **previous state**, **h(t-1)**, Forget gate helps us to take decisions about what must be removed from h(t-1) state and thus keeping only relevant stuff. It is surrounded by a sigmoid function which helps to crush the input between [0,1].
- The **calculation** is **done** by considering the **new input** and the **previous timestamp** which eventually **leads** to the **output** of a number **between 0 and 1** for **each** number in that **cell state.** As typical binary, **1** represents to **kee**p the cell state while **0** represents to **trash** it.

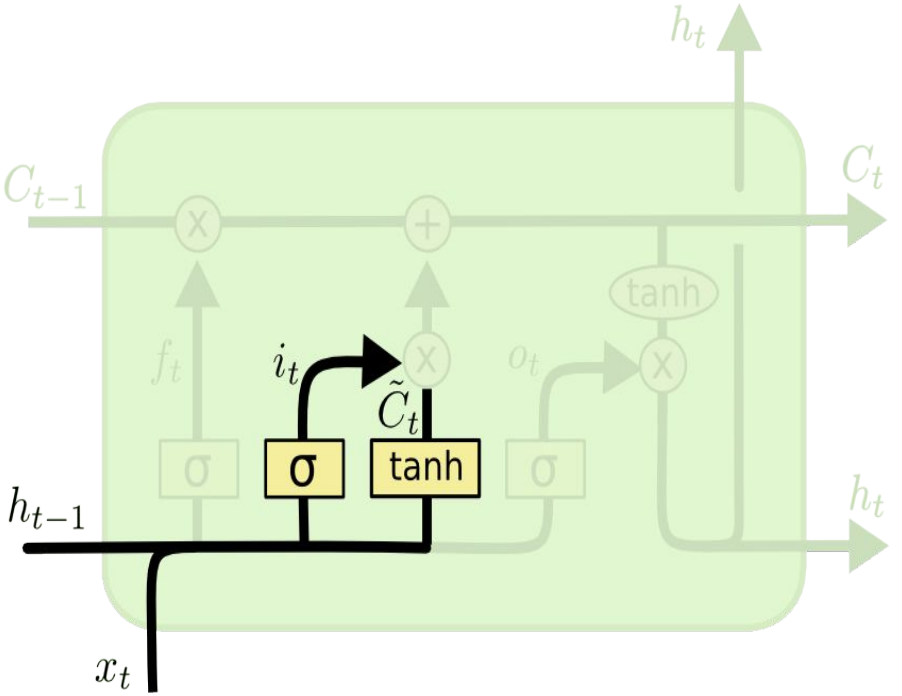


$$f_t = \sigma(w_f[h_{t-1}, x_t] + b_f)$$

$w_f = Weight$
$h_{t-1} = Output\ from\ previous\ timestamp$
$x_t = New\ input$
$b_f = Bias$

LSTM Structure…

- The **2nd** step is to **decide,** what **new information** we're going to **store** in the cell state. This whole process comprises of following steps:
  - A **sigmoid layer** called the "input gate layer" decides **which values** will be **updated.**
  - The **tanh layer** creates a **vector** of **new candidate** values, that could be **added** to the state.
  - The input from the **previous timestamp** and the new input are **passed** through a **sigmoid function** which gives the value $\boldsymbol{i_t}$**.** This value is then **multiplied by** $\boldsymbol{c_t}$ and then added to the **cell state.**

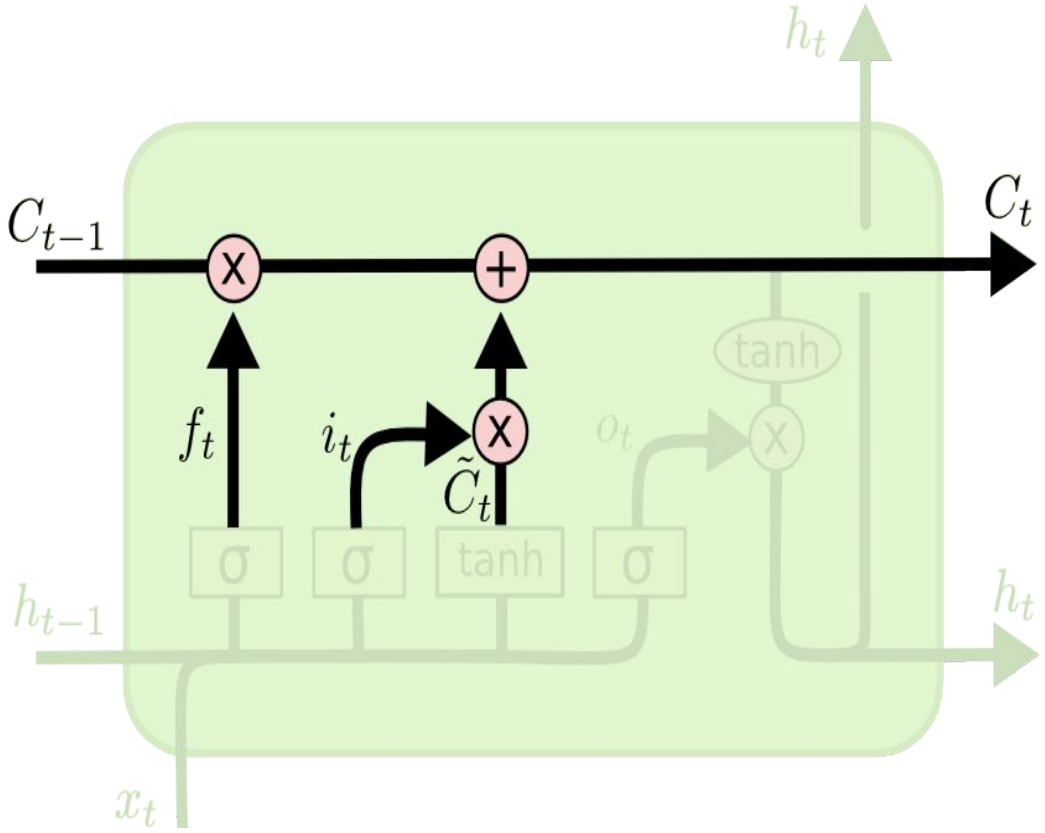


$$i_t = \sigma(w_i[h_{t-1}, x_t] + b_i)$$

$$\tilde{c}_t = tanh(w_c[h_{t-1}, x_t] + b_c)$$

LSTM Structure…

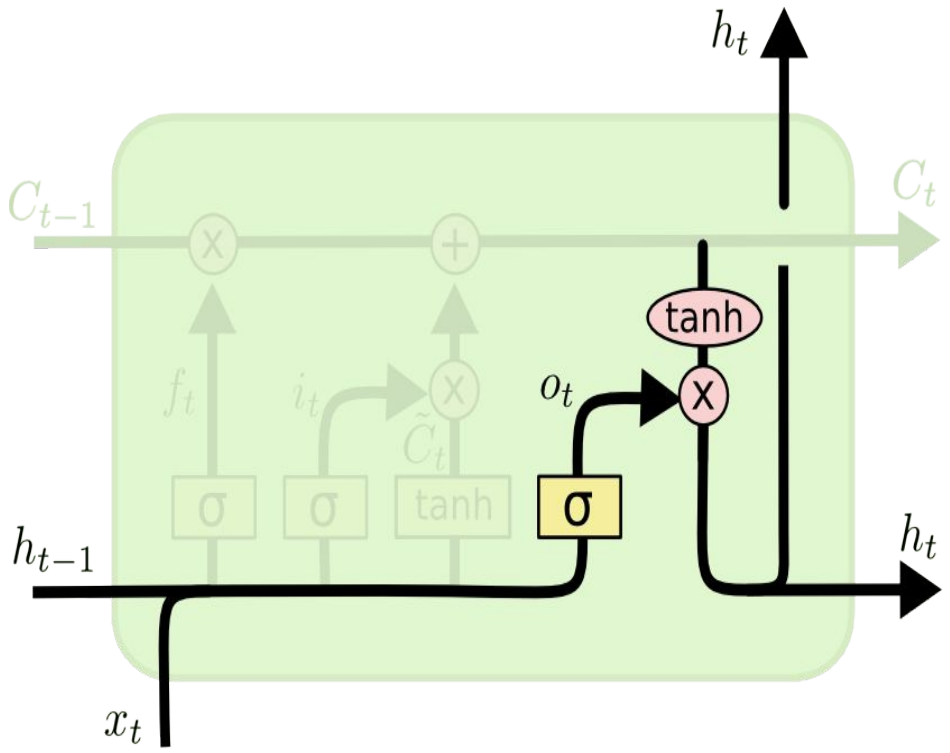
- In **3rd** step, **update** the **old cell state Ct−1,** into the **new** cell state **Ct.**
- *First*, **multiply** the **old** state c_{t-1} by f_t , **forgetting** the things we **decided** to **leave behind** earlier.
- Then, we **add** $i_t * \tilde{c_t}$. This is the **new candidate** values, **scaled** by how much we decided to **update each state** value.
- In the *Second* step, we decided to do **make use** of the **data** which is only required at that **stage.**
- In the *third* step, **implement** it.



$$c_t = f_t * c_{t-1} + i_t^* \tilde{c}_t$$

LSTM Structure…

- In 4th Step, a **sigmoid layer** is used which decides what **parts** of the **cell state** is going to **output.** It is also called **Output Gate**.
- Then, the **cell state** is kept through ***tanh*** (push the values to be between −1 and 1).
- Later, we **multiply** it by the **output** of the **sigmoid gate,** so that we only output the **parts** we decided to.
- The **output** consists of only the **outputs** there were decided to be **carry forwarded** in the **previous** steps and not all the **outputs** at once.



$$o_t = \sigma(w_o[h_{t-1}, x_t] + b_o)$$

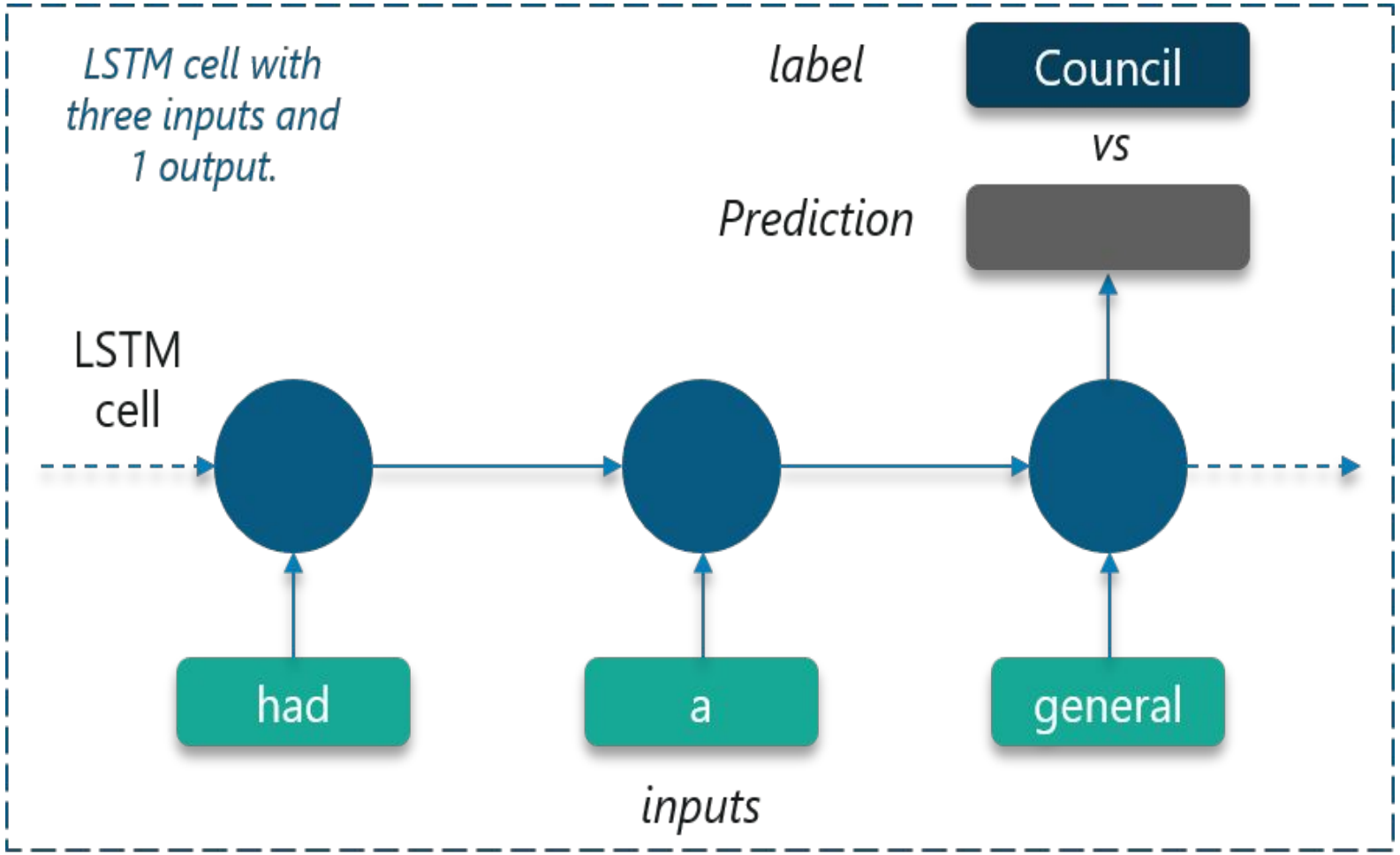
$$h_t = o_t * \tanh(c_t)$$

Summary of LSTM

- During **backpropagation** through each LSTM cell, it's multiplied by different values of forget gate, which makes it **less prone** to **vanishing/exploding** gradient.
- Though, if values of all forget gates are less than 1, it may **suffer** from **vanishing gradient** but in practice people tend to initialise the bias terms with some positive number so in the beginning of training ***f*** (forget gate) is very close to 1 and as time passes the model can learn these bias terms.
- In the **first** step, **we found out what was needed to be dropped.**
- **The second** step consisted of w**hat new inputs are added to the network.**
- **The third** step was to **combine the previously obtained inputs to generate the new cell states.**
- **Lastly,** we arrived at the **output as per requirement.**

A case study: LSTM

- Consider to **predict** the **next word** in a sample short story.
- Start by **feeding** an **LSTM** Network with **correct sequences** from the text of 3 **symbols** as **inputs** a nd 1 labeled symbol.

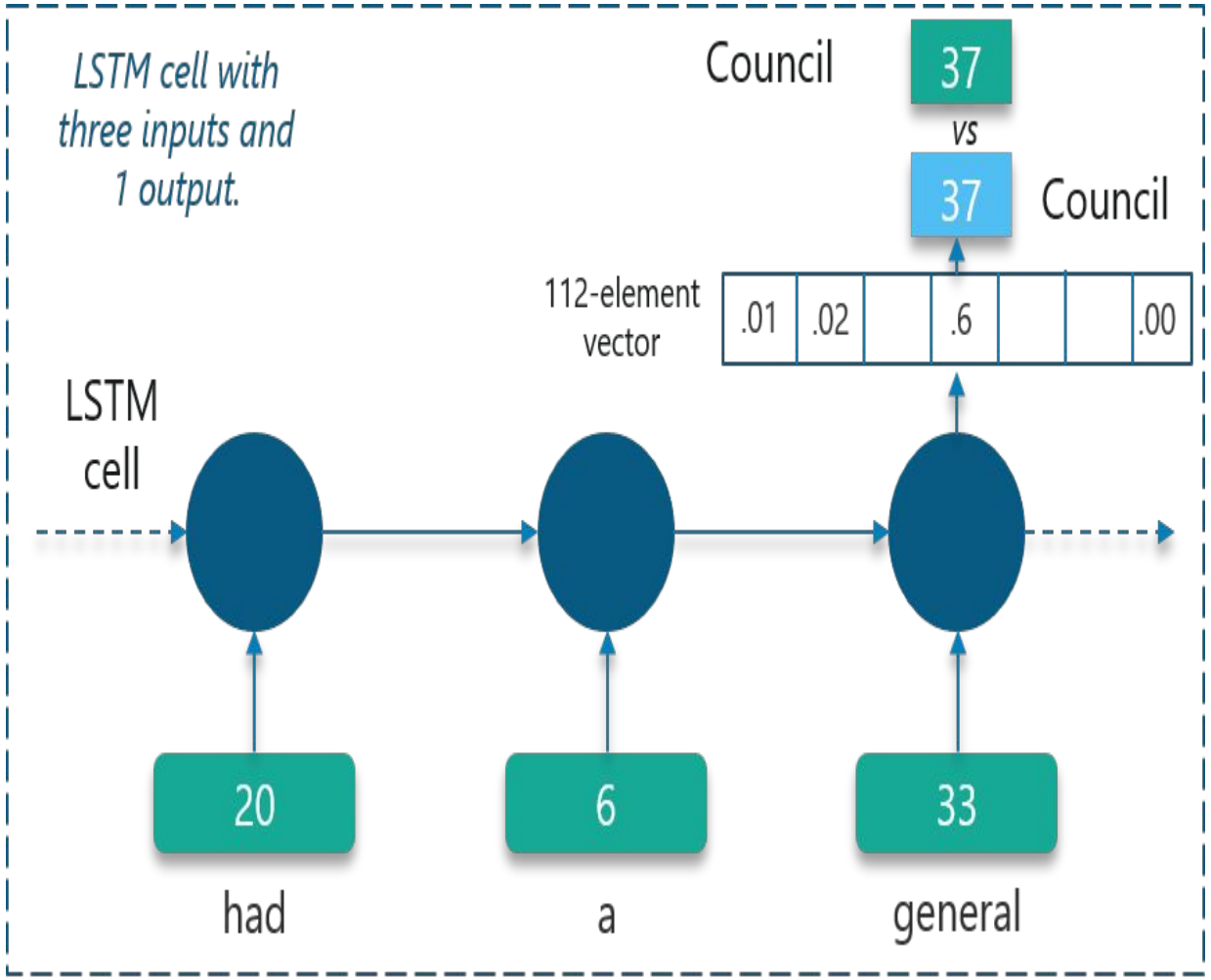


Dataset

- The LSTM is trained using a **sample short story** which consists of **112 unique symbols. Comma** and **period** are also **considered** as **unique symbols** in this **case.**
- *“long ago , the mice had a general council to consider what measures they could take to outwit their common enemy , the cat . some said this , and some said that but at last a young mouse got up and said he had a proposal to make , which he thought would meet the case . you will all agree , said he , that our chief danger consists in the sly and treacherous manner in which the enemy approaches us . now , if we could receive some signal of her approach , we could easily escape from her . i venture , therefore , to propose that a small bell be procured , and attached by a ribbon round the neck of the cat . by this means we should always know when she was about , and could easily retire while she was in the neighborhood . this proposal met with general applause , until an old mouse got up and said that is all very well , but who is to bell the cat ? the mice looked at one another and nobody spoke . then the old mouse said it is easy to propose impossible remedies .”*

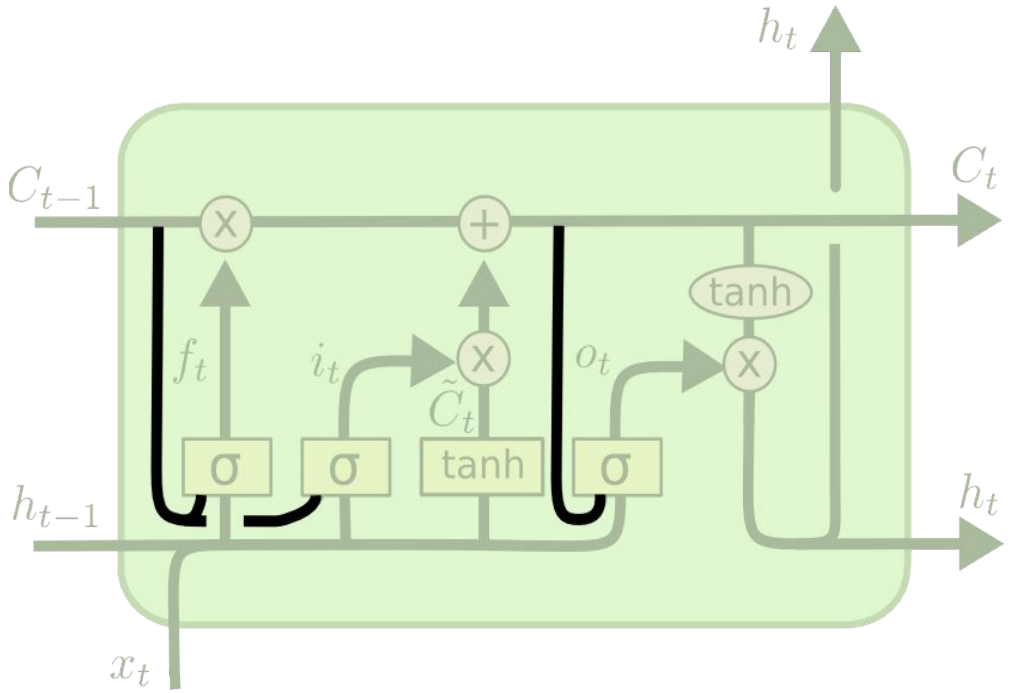
Training

- **LSTMs** can only **understand real numbers.** So, the first **requirement** is to **convert** the unique symbols into **unique integer** values based on the **frequency** of **occurrence.**
- Doing this will create a **customized dictionary** that we can **make use** of later on to **map** the values.
- The network will create a **112-element vector** consisting of the **probability of occurrence** of each of these unique integer values.



Variants of LSTM

- A normal LSTM is discussed in previous topic. But not all LSTMs are the same as we discussed.
- In fact, most of the paper uses slightly different version of LSTM. The differences are minor, but it's worth mentioning some of them.
- One popular LSTM variant, introduced by Gers & Schmidhuber (2000), is adding "peephole connections." This means that we let the gate layers look at the cell state.
- The diagram adds peepholes to all the gates, but many papers will give some peepholes and not others.



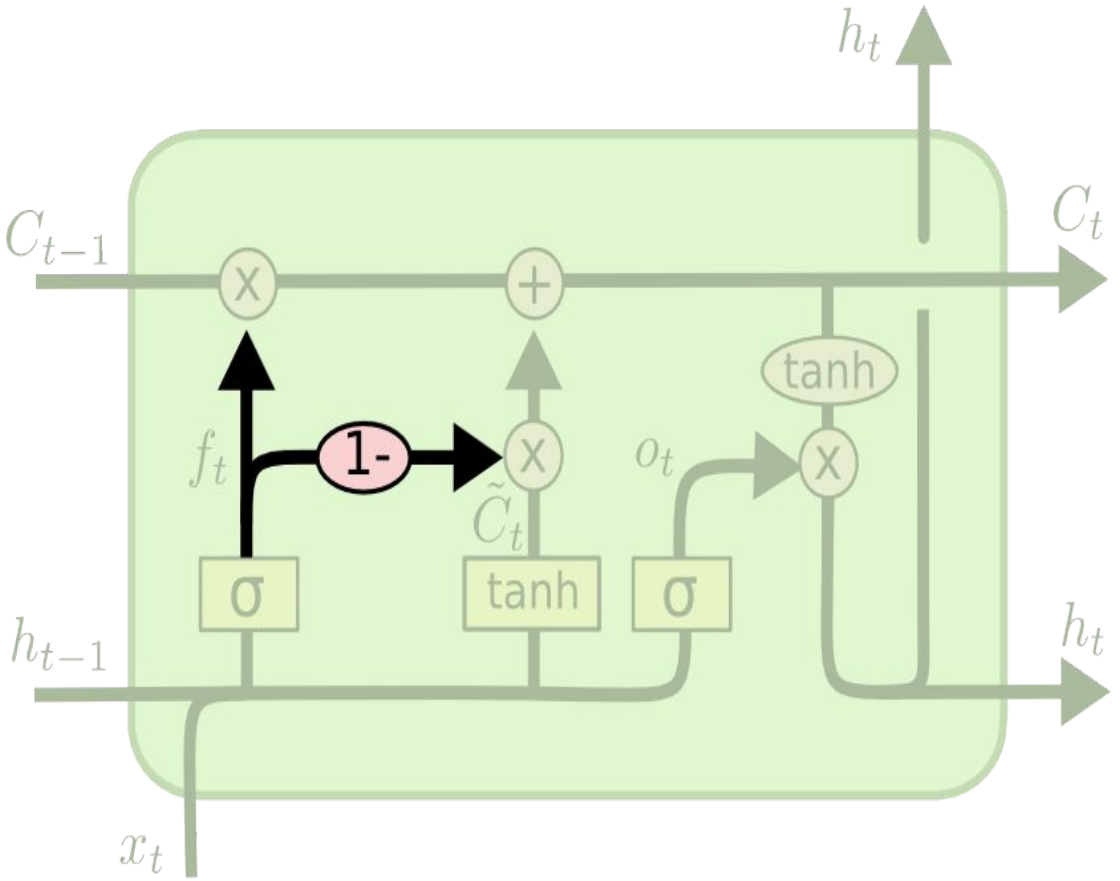
$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_f\right)$$

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

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

Variants of LSTM…

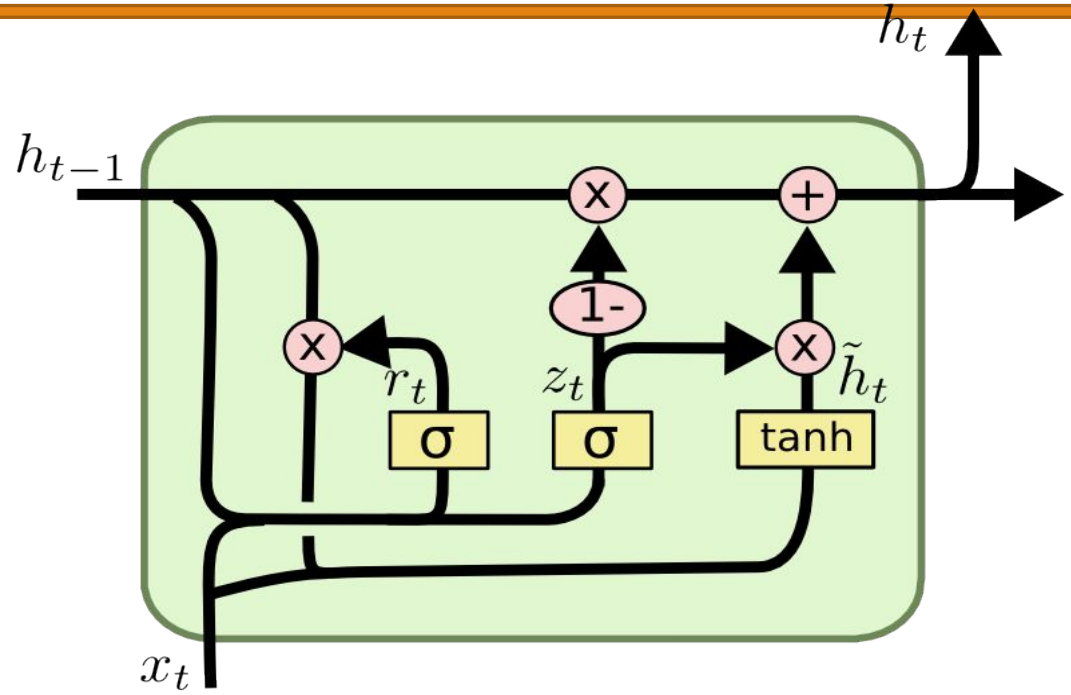
- Another variation is to use coupled forget and input gates.
- Instead of separately deciding what to forget and what we should add new information to, we make those decisions together.
- We only forget when we're going to input something in its place.
- We only input new values to the state when we forget something older.



$$C_t = f_t * C_{t-1} + (1 - f_t) * \tilde{C}_t$$

Variants of LSTM…

- A slightly more dramatic variation on the LSTM is the Gated Recurrent Unit, or GRU, introduced by Cho, et al. (2014).
- It combines the forget and input gates into a single “update gate.”
- It also merges the cell state and hidden state, and makes some other changes.
- The resulting model is simpler than standard LSTM models, and has been growing increasingly popular.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Summary of LSTM Variants

- These are only a few of the most notable LSTM variants. There are lots of others, like Depth Gated RNNs by Yao, et al. (2015) and Clockwork RNNs by Koutnik, et al. (2014).
- A nice comparison is made by Greff, et al. (2015) of popular variants, finding that they're all about the same. Jozefowicz, et al. (2015) tested more than ten thousand RNN architectures, finding some that worked better than LSTMs on certain tasks.
- The next step in LSTMs are attention. The idea is to let every step of an RNN pick information to look at from some larger collection of information. For example, if you are using an RNN to create a caption describing an image, it might pick a part of the image to look at for every word it outputs.
- Attention isn't the only exciting thread in RNN research. For example, Grid LSTMs by Kalchbrenner, *et al.* (2015) seem extremely promising. Work using RNNs in generative models – such as Gregor, *et al.* (2015), Chung, *et al.* (2015), or Bayer & Osendorfer (2015) – also seems very interesting.
- The last few years have been an exciting time for recurrent neural networks, and the coming ones promise to only be more so!

References

- https://www.youtube.com/watch?v=UNmqTiOnRfg
- https://www.youtube.com/watch?v=WCUNPb-5EYI
- https://www.youtube.com/watch?v=iX5V1WpxxkY
- https://www.edureka.co/blog/recurrent-neural-networks/

https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d#e276

Thank You

Contact: dinesh@dtu.ac.in
Mobile: +91-9971339840