Lecture 3 Long Short-Term Memory Model

1 Introduction to Deep Learning

In a neural network, layers are the building blocks that organize and process data. Each layer consists of a set of interconnected nodes, also called neurons or units. These layers are arranged sequentially, forming a network architecture. In the Recurrent Neural Network (RNN) structure, we've discussed 3 types of layers: Input Layer, Hidden Layer and Output Layer.

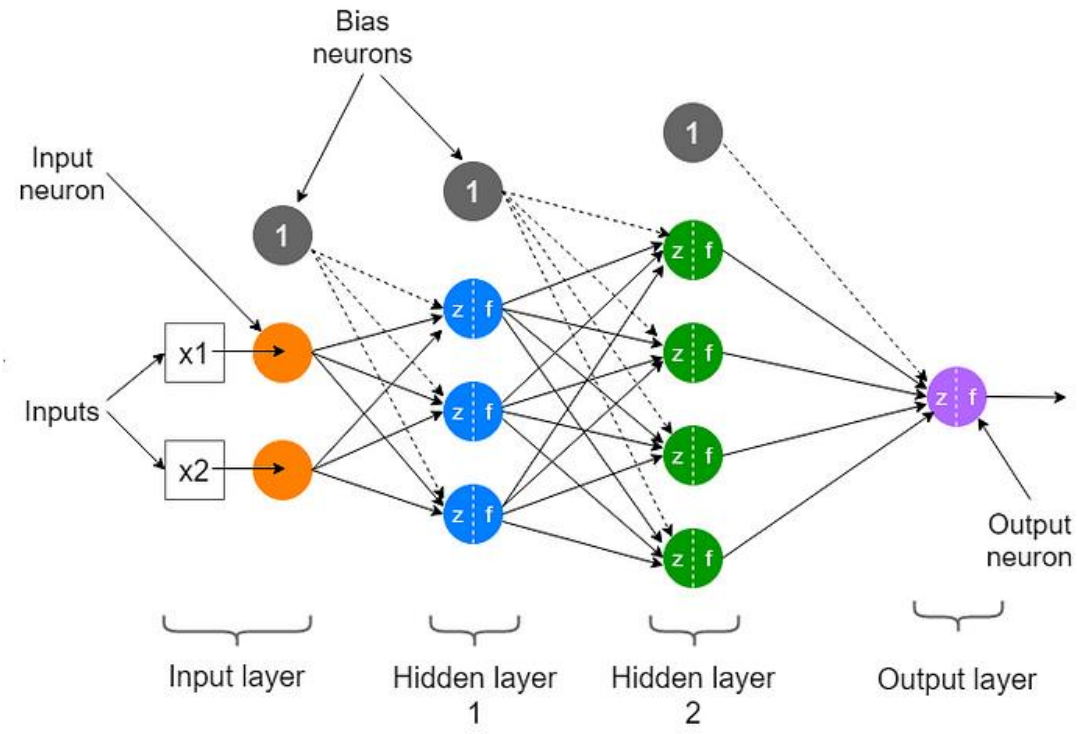
- Input Layer: This is the first layer of the neural network and is responsible for receiving the raw input data. Each node in the input layer represents a feature or attribute of the input data. The number of nodes in the input layer corresponds to the dimensionality of the input data.
- Hidden Layers: Hidden layers are intermediate layers between the input and output layers. They perform various transformations on the input data through weighted connections between nodes. Each node in a hidden layer computes a weighted sum of its inputs and applies an activation function to produce an output.
- Output Layer: The output layer is the final layer of the neural network. It produces the network's predictions or outputs based on the transformations performed by the hidden layers. The number of nodes in the output layer depends on the type of task the neural network is designed for. For example, in a binary classification task, there may be one node representing each class with a sigmoid activation function, while in a multi-class classification task, there may be multiple nodes with a softmax activation function.

Each layer in a neural network can have its own unique architecture, including the number of nodes and the choice of activation function. The connections between nodes in adjacent layers are weighted, and these weights are adjusted during the training process through methods like backpropagation and gradient descent in order to minimize the difference between the predicted outputs and the actual targets.

Those layers also exist in **deep neural networks**. The key difference is that deep neural networks have two or more hidden layers instead of just one.

An Artificial Neural Network (ANN) with two or more hidden layers is known as a **Deep Neural Network**. The process of training deep neural networks is called *deep learning*. The term **"deep"** in deep learning refers to the number of hidden layers (also called *depth*) of a neural network.

Figure 1. Deep Neural Network Architecture with two hidden layers

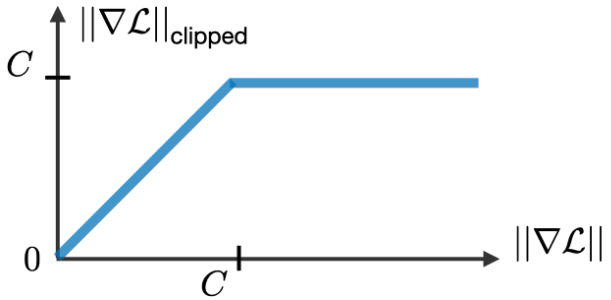


The same matrix representation of inputs and parameters used in shallow neural network architectures can also be used in deep neural architectures. All we have to do is to increase the number of weight matrices and bias vectors depending on the number of hidden layers we use.

2 Motivation of GRU and LSTM

The vanishing and exploding gradient phenomena are often encountered in the context of RNNs. The reason why they happen is that it is difficult to capture long term dependencies because of multiplicative gradient that can be exponentially decreasing/increasing with respect to the number of layers.

Gradient clipping is a technique used to cope with the exploding gradient problem sometimes encountered when performing backpropagation. By capping the maximum value for the gradient, this phenomenon is controlled in practice.



Gated Recurrent Unit (GRU) and Long Short-Term Memory units (LSTM) deal with the vanishing gradient problem encountered by traditional RNNs, with LSTM being a generalization of GRU.

2 GRU Anatomy

Beyond the extensions discussed so far, RNNs have been found to perform better with the use of more complex units for activation. So far, we have discussed methods that transition from hidden state $h(t-1)$ to $h(t)$ using an affine transformation and a point-wise nonlinearity. Here, we discuss the use of a gated activation function thereby modifying the RNN architecture. Gated recurrent units are designed in a manner to have more persistent memory thereby making it easier for RNNs to capture long-term dependencies. Let us see mathematically how a GRU uses $h(t-1)$ and $x(t)$ to generate the next hidden state $h(t)$. We will then dive into the intuition of this architecture.

Update Gate

$$z(t) = \sigma(U_z \cdot x(t) + W_z \cdot h(t-1) + b_z)$$

Reset Gate

$$r(t) = \sigma(U_r \cdot x(t) + W_r \cdot h(t-1) + b_r)$$

New Memory

$$\tilde{h}(t) = tanh(U \cdot x(t) + r(t) \circ Wh(t-1))$$

Hidden State

$$h(t) = \big(1 - z(t)\big) \circ \tilde{h}(t) + z(t) \circ h(t-1))$$

The above equations can be thought of a GRU's four fundamental operational stages, and they have intuitive interpretations that make this model much more intellectually satisfying:

1. New memory generation: A new memory $\tilde{h}(t)$ is the consolidation of a new input word $x(t)$ with the past hidden state $h(t-1)$. Anthropomorphically, this stage is the one who knows the recipe of combining a newly observed word with the past hidden state $h(t-1)$ to summarize this new word in light of the contextual past as the vector $\tilde{h}(t)$. The white bullet sign ° represents the dot product between two vectors.

2. Reset Gate: The reset signal $r(t)$ is responsible for determining how important $h(t-1)$ is to the new memory $\tilde{h}(t)$. The reset gate has the ability to completely diminish past hidden state if it finds that $h(t-1)$ is irrelevant to the computation of the new memory.

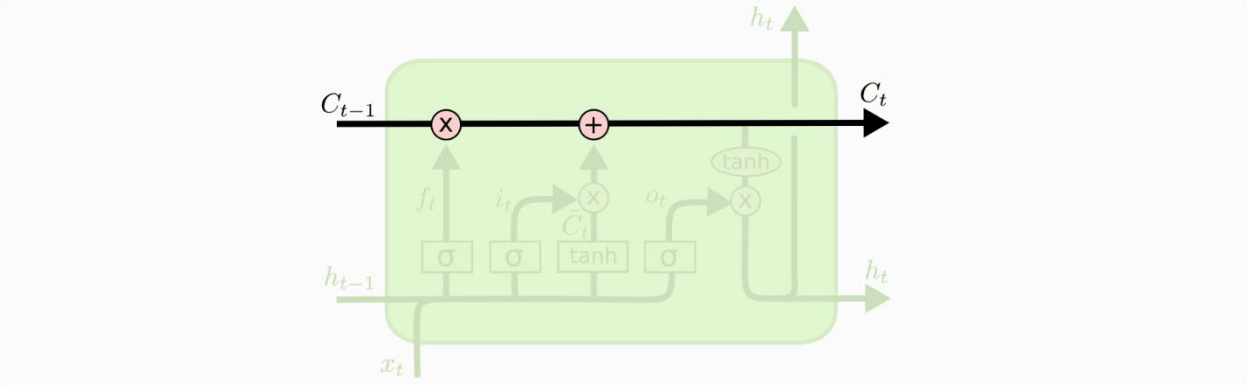
3. Update Gate: The update signal $z(t)$ is responsible for determining how much of $h(t-1)$ should be carried forward to the next state. For instance, if $z(t) \rightarrow 1$, then $h(t-1)$ is almost entirely copied out to $h(t)$. Conversely, if $z(t) \rightarrow 0$, then mostly the new memory $h(t)$ is forwarded to the next hidden state.

4. Hidden state: The hidden state $h(t)$ is finally generated using the past hidden input $h(t-1)$ and the new memory generated $\tilde{h}(t)$ with the advice of the update gate.

It is important to note that to train a GRU, we need to learn parameters: W, U, W_r, U_r, W_z , U_z. These follow the same backpropagation procedure we have seen in the RNN lecture.

3 LSTM Anatomy

Now let's get into the details of the architecture of LSTM network. The key to LSTMs is the cell state, the horizontal line running through the top of the diagram. The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions between input $x(t)$ and hidden state $h(t)$.

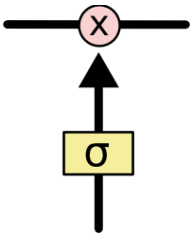


A typical LSTM network is comprised of different memory blocks called cells (the rectangles that we see in the image). The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates.

There are two states that are being transferred to the next cell; the cell state and the hidden state. The memory blocks are responsible for remembering things and manipulations to this memory is done through three major mechanisms, called gates. Each of them is listed below.

- Forget Gate
- Input Gate
- Output Gate

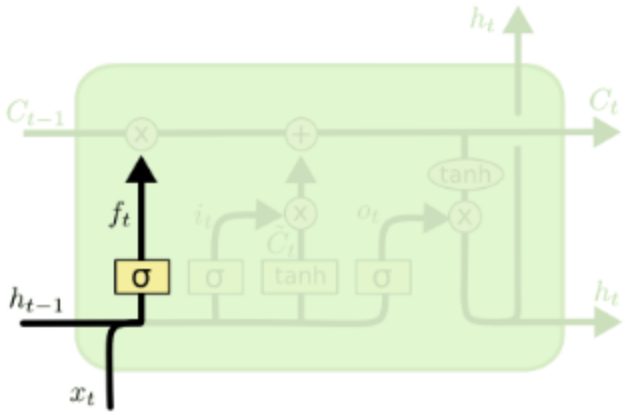
Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.



The sigmoid layer $\sigma(\cdot)$ outputs numbers between zero and one, describing how much of each component should be let through. A value of zero means "let nothing through," while a value of one means "let everything through!"

2.1 Forget Gate

The first step in our LSTM is to decide what information we're going to throw away from the cell state. This decision is made by a sigmoid layer called the "forget gate layer."



A forget gate is responsible for removing information from the cell state. The information that is no longer required for the LSTM to understand things or the information that is of less importance is removed via multiplication of a filter.

$$a_f(t) = U_f \cdot x(t) + W_f \cdot h(t-1) + b_f$$

$$f(t) = \sigma(a_f(t))$$

Let's try to understand the equation, here

$x(t)$: input to the current timestamp.

U_f: weight associated with the input

$h(t-1)$: The hidden state of the previous timestamp

W_f: It is the weight matrix associated with the hidden state

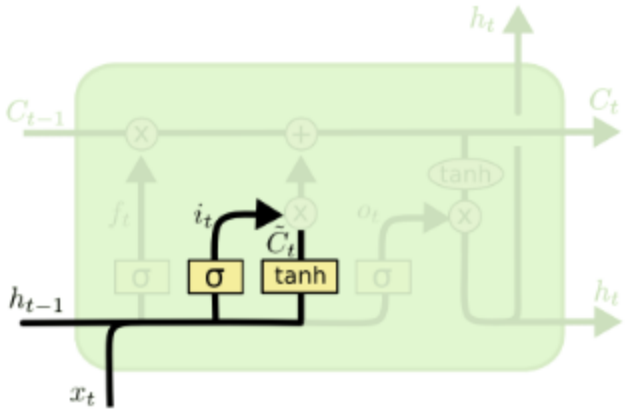
b_f: bias term at the Forget Gate

Later, a sigmoid function $\sigma(\cdot)$ is applied to it. That will make $f(t)$ a number between 0 and 1. This $f(t)$ is later multiplied with the cell state of the previous timestamp to update the cell state at t. The information kept from Forget Gate for cell state $c(t)$ is:

$$f(t) \cdot c(t-1) = \begin{cases} 0, & if\ f(t) \to 0 \\ c(t-1), & if\ f(t) \to 1 \end{cases}$$

2.2 Input Gate

The next step is to decide what new information we're going to store in the cell state. This has two parts. First, a sigmoid layer called the "input gate layer" decides which values we'll update. Next, a tanh layer creates a vector of new information, $\tilde{C}(t)$, to be added to the state.



The input gate is used to quantify the impact carried by the input at timestamp t, i.e. $x(t)$. Here is the equation of the input gate:

$$a_i(t) = U_i \cdot x(t) + W_i \cdot h(t-1) + b_i$$

$$i(t) = \sigma(a_i(t))$$

$x(t)$: input to the current timestamp.

U_i: weight associated with the input $x(t)$

$h(t-1)$: The hidden state of the previous timestamp

W_i: It is the weight matrix associated with the hidden state $h(t-1)$

b_i: bias term at the Input Gate

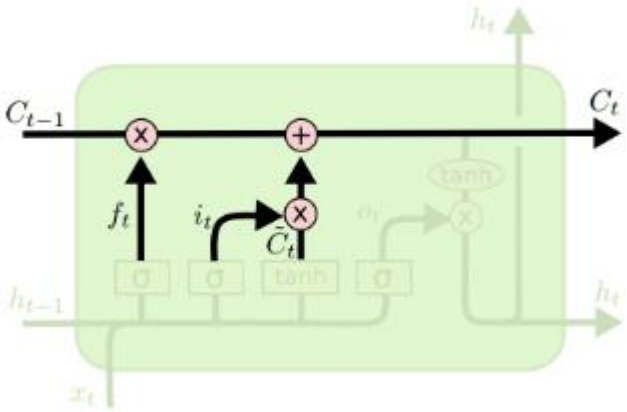
Again, we have applied the sigmoid function over it. As a result, the value of $i(t)$ at timestamp t will be between 0 and 1.

2.3 New Information

$$a_c(t) = U_c \cdot x(t) + W_c \cdot h(t-1)$$

$$\tilde{C}(t) = tanh(a_c(t))$$

Now the new information that needed to be passed to the cell state is a function of a hidden state at the previous timestamp $h(t-1)$ and input x at the current timestamp $x(t)$. The activation function here is $\tanh(\cdot)$. Due to the tanh function, the value of new information will be between -1 and 1. If the value of $\tilde{C}(t)$ is negative, the information is subtracted from the cell state, and if the value is positive, the information is added to the cell state at the current timestamp.



Here $\tilde{C}(t)$ is essentially used to update the cell state. Here comes the updated equation:

$$c(t) = f(t) \circ c(t-1) + i(t) \circ \tilde{C}(t)$$

Here $i(t)$ is the output from the input gate. The value of $i(t)$ will be between 0 and 1. It is used to multiply the new information $\tilde{C}(t)$. $c(t)$ is the cell state at the current timestamp t. The white bullet sign ° represents the dot product.

Roughly, you can regard each cell state is updated as

$$cell\ state\ c(t) = \ what\ is\ kept\ from\ (t-1)\ by\ Forget\ Gate + \ news\ brought\ by\ Input\ Gate\ at\ t$$

2.4 Output Gate

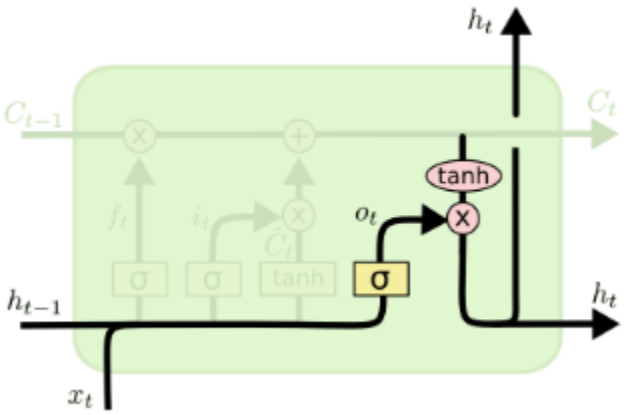
Finally, we need to decide what we're going to output. This output will be based on our cell state but will be a filtered version. First, we run a sigmoid layer which decides what parts of the cell state we're going to output.

In particular, the Output gate is pretty similar to the two previous gates: Forget Gate and Input Gate.

$$a_o(t) = U_o \cdot x(t) + W_o \cdot h(t-1) + b_o$$

$$o(t) = \sigma(a_o(t))$$

Its value will also lie between 0 and 1 by using the sigmoid function.



Then, we put the cell state through tanh (to push the values to be between −1 and 1) and multiply it by the output of the sigmoid function from Output Gate, so that we only output the parts we decided to.

To update the hidden state, we will use $o(t)$ and tanh of the updated cell state $c(t)$:

$$h(t) = o(t) \circ tanh\big(c(t)\big)$$

It turns out that the hidden state is a function of long-term memory $c(t)$ and the current output $o(t)$. For the language model, assume our model has taken a subject as input $x(t)$, it might want to output information relevant to a verb in case that's what is coming next. For example, it might output whether the subject is singular or plural (hidden state $h(t)$), so that we know what form a verb should be conjugated into if that's what follows next. Again, the white bullet sign ° represents the dot product.

If you need to take the output of the current timestamp, just apply the SoftMax activation on hidden state:

$$\hat{y}(t) = \text{softmax}(\, h(t))$$

4 Model

Input gate:

$$i(t) = \sigma(U_i \cdot x(t) + W_i \cdot h(t-1) + b_i)$$

We see that the new memory generation stage doesn't check if the new word is even important before generating the new memory – this is exactly the input gate's function. The input gate uses the input word and the past hidden state to determine whether or not the input is worth preserving and thus is used to gate the new memory. It thus produces $i(t)$ as an indicator of this information.

Forget gate:

$$f(t) = \sigma(U_f \cdot x(t) + W_f \cdot h(t-1) + b_f)$$

This gate is similar to the input gate except that it does not make a determination of usefulness of the input word – instead it makes an assessment on whether the past memory cell is useful for the

computation of the current memory cell. Thus, the forget gate looks at the input word and the past hidden state and produces $f(t)$.

Output / Exposure gate:

$$o(t) = \sigma(U_o \cdot x(t) + W_o \cdot h(t-1) + b_o)$$

This is a gate that does not explicitly exist in GRUs. Its purpose is to separate the final memory from the hidden state. The final memory $c(t)$ contains a lot of information that is not necessarily required to be saved in the hidden state. Hidden states are used in every single gate of an LSTM and thus, this gate makes the assessment regarding what parts of the memory $c(t)$ needs to be exposed/present in the hidden state $h(t)$. The signal it produces to indicate this is $o(t)$ and this is used to gate the point-wise $\tanh(\cdot)$ of the memory.

New memory generation

$$\tilde{C}(t) = tanh(U_c \cdot x(t) + W_c \cdot h(t-1))$$

This stage is analogous to the new memory generation stage we saw in GRUs. We essentially use the input word $x(t)$ and the past hidden state $h(t-1))$ to generate a new memory $\tilde{C}(t)$ which includes aspects of the new word $x(t)$.

Final memory generation

$$c(t) = f(t) \circ c(t-1) + i(t) \circ \tilde{C}(t)$$

This stage first takes the advice of the forget gate $f(t)$ and accordingly forgets (or remember) the past memory $c(t-1)$. Similarly, it takes the advice of the input gate $i(t)$ and accordingly gates the new memory $\tilde{C}(t)$. It then sums these two results to produce the final memory $c(t)$.

Update hidden-state:

To update the hidden state, we will use $o(t)$ and tanh of the updated cell state $c(t)$:

$$h(t) = o(t) \circ tanh\big(c(t)\big)$$

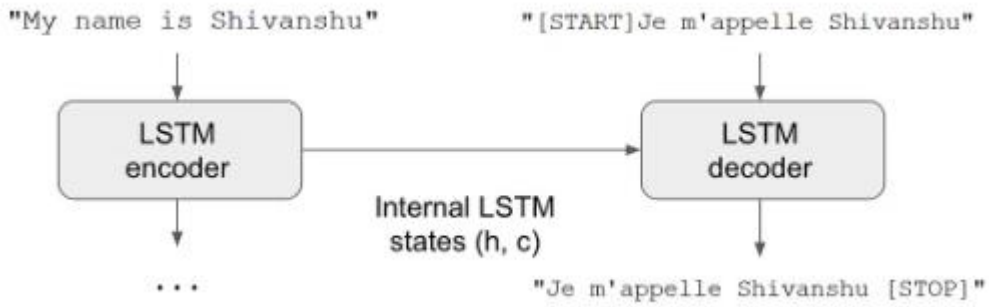
5 Applications of LSTM Neural Networks

5.1 Sequence to Sequence (Seq2Seq) Models

LSTM is widely used in Sequence to Sequence (Seq2Seq) models, a type of neural network architecture used for many sequence-based tasks such as machine translation, speech recognition, and text summarization.

In Seq2Seq models, the input sequence is fed into an encoder LSTM layer, which produces a hidden state that summarizes the input sequence. This hidden state is then used as the initial state for a decoder LSTM layer, which generates the output sequence one token at a time.

For example, the following diagram shows the utilization of the LSTM model.



The input sequence of the model would be the sentence in the source language (e.g. English), and the output sequence would be the sentence in the target language (e.g. French).

The model would use an encoder LSTM to encode the input sentence into a fixed-length vector, which would then be fed into a decoder LSTM to generate the output sentence.

5.2 Time Series Forecasting

LSTMs are popular for time series forecasting due to their ability to model complex temporal dependencies and handle long-term memory.

Time series datasets often exhibit different types of recurring patterns known as seasonalities. These seasonalities can occur over long periods, such as every year, or over shorter time frames, such as weekly cycles. LSTMs can identify and model both long and short-term seasonal patterns within the data.

The occurrence of events can impact demand not only on the day of the event but also on the days preceding and following the event. For instance, people may book more accommodations to attend a sports event. The LSTM model can distinguish and analyze the effect of different types of events on demand patterns.

The flexibility of LSTM allows it to handle input sequences of varying lengths. It becomes especially useful when building custom forecasting models for specific industries or clients.

To improve its ability to capture non-linear relationships for forecasting, LSTM has several gates. Causal factors tend to have a non-linear impact on demand. LSTM can learn this relationship for forecasting when these factors are included as part of the input variable.

5.3 Natural Language Processing (NLP)

NLP involves the processing and analysis of natural language data, such as text, speech, and conversation. Using LSTMs in NLP tasks enables the modeling of sequential data, such as a sentence or document text, focusing on retaining long-term dependencies and relationships.

One of the key challenges in NLP is the modeling of sequences with varying lengths. LSTMs can handle this challenge by allowing for variable-length input sequences as well as variable-length output sequences. In text-based NLP, LSTMs can be used for a wide range of tasks, including language translation, sentiment analysis, speech recognition, and text summarization.

In addition to their ability to model variable-length sequences, LSTMs can also capture contextual information over time, making them well-suited for tasks that require an understanding of the context or the meaning of the text.

Reference

https://www.deeplearningbook.org/

https://web.stanford.edu/class/cs224n/readings/cs224n-2019-notes05-LM_RNN.pdf

https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/?social=google&next=https%3A%2F%2Fwww.analyticsvidhya.com%2Fblog%2F2017%2F12%2Ffundamentals-of-deep-learning-introduction-to-lstm%2F

https://www.analyticsvidhya.com/blog/2021/03/introduction-to-long-short-term-memory-lstm/

https://www.analyticsvidhya.com/blog/2022/01/the-complete-lstm-tutorial-with-implementation/

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

https://www.geeksforgeeks.org/deep-learning-introduction-to-long-short-term-memory/

https://d2l.ai/chapter_recurrent-modern/lstm.html

https://medium.com/data-science-365/two-or-more-hidden-layers-deep-neural-network-architecture-9824523ab903#:~:text=The%20process%20of%20training%20deep,depth)%20of%20a%20neural%20network.&text=In%20the%20shallow%20neural%20network,Hidden%20Layer%20and%20Output%20Layer

https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/

https://www.projectpro.io/article/lstm-model/832

https://machinelearningmastery.com/how-to-develop-lstm-models-for-time-series-forecasting/

https://colab.research.google.com/gist/ayulockin/c53e9b3c1e804a05c05360807d88220a/how-to-use-lstm-for-one-to-many-many-to-one-and-many-to-many-sequences.ipynb#scrollTo=lovky1ZMviH_