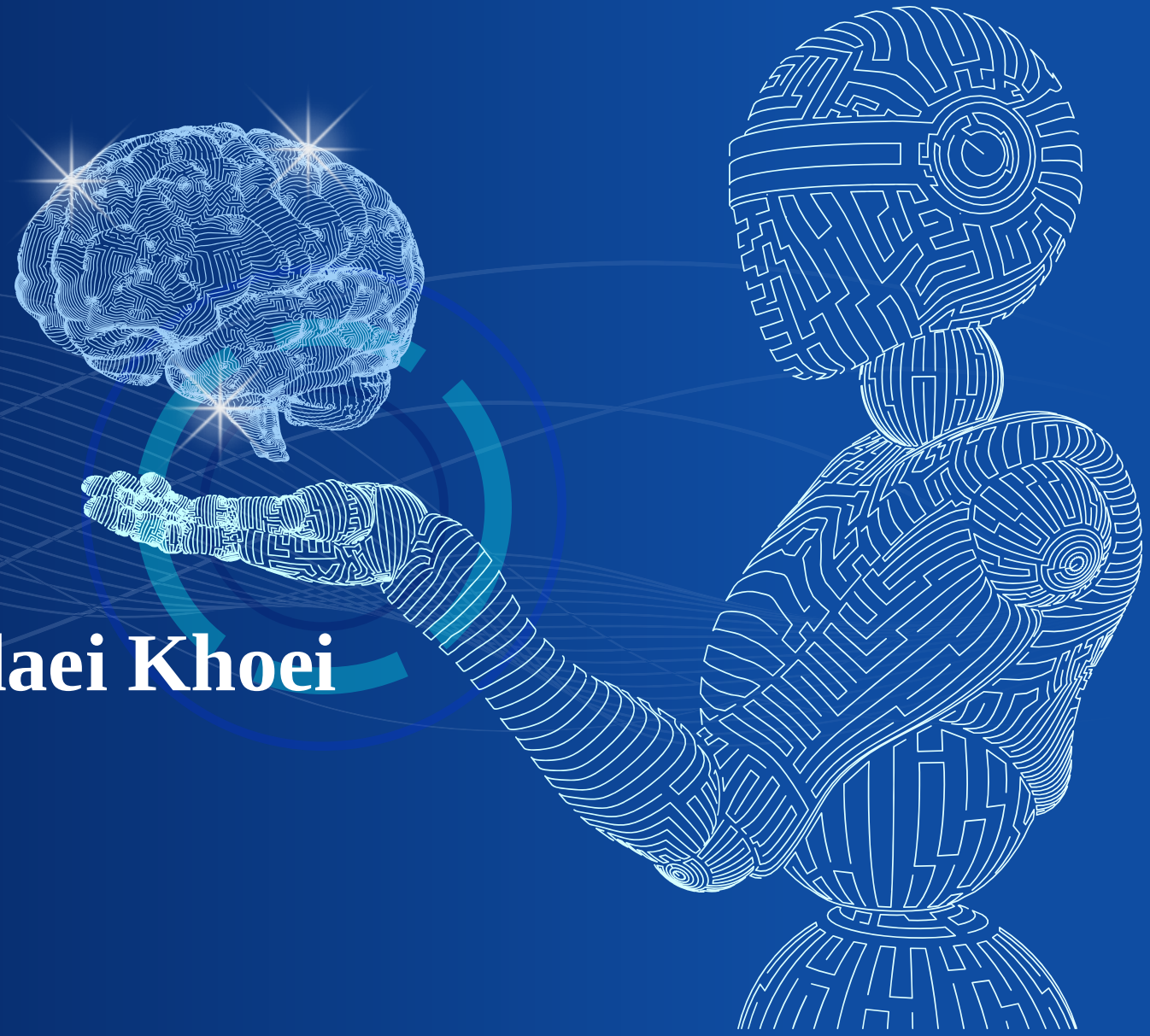
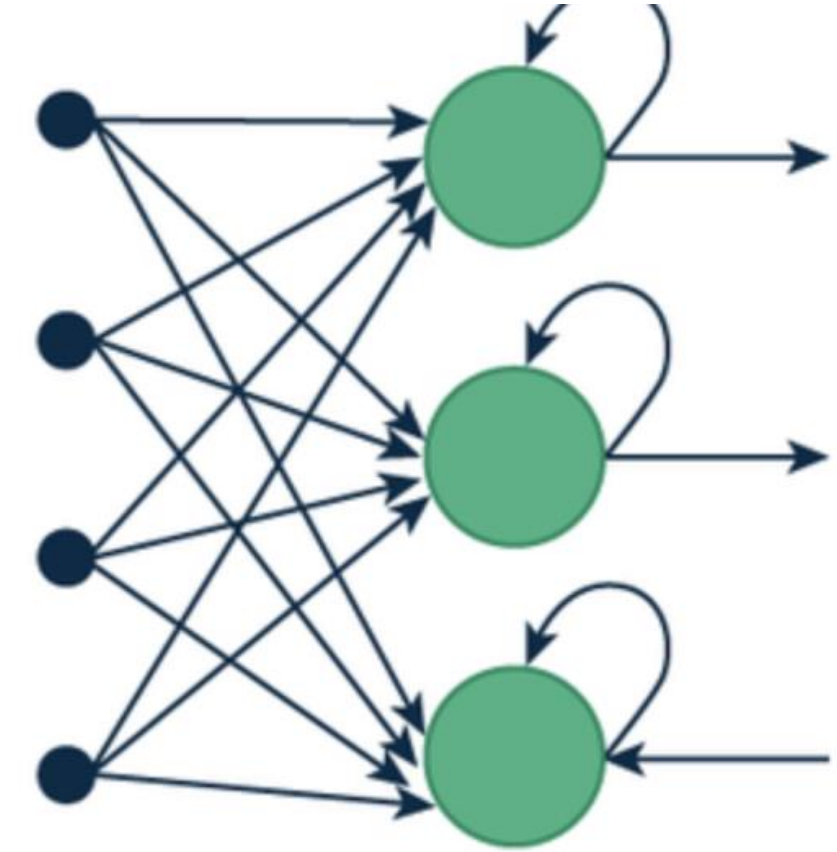
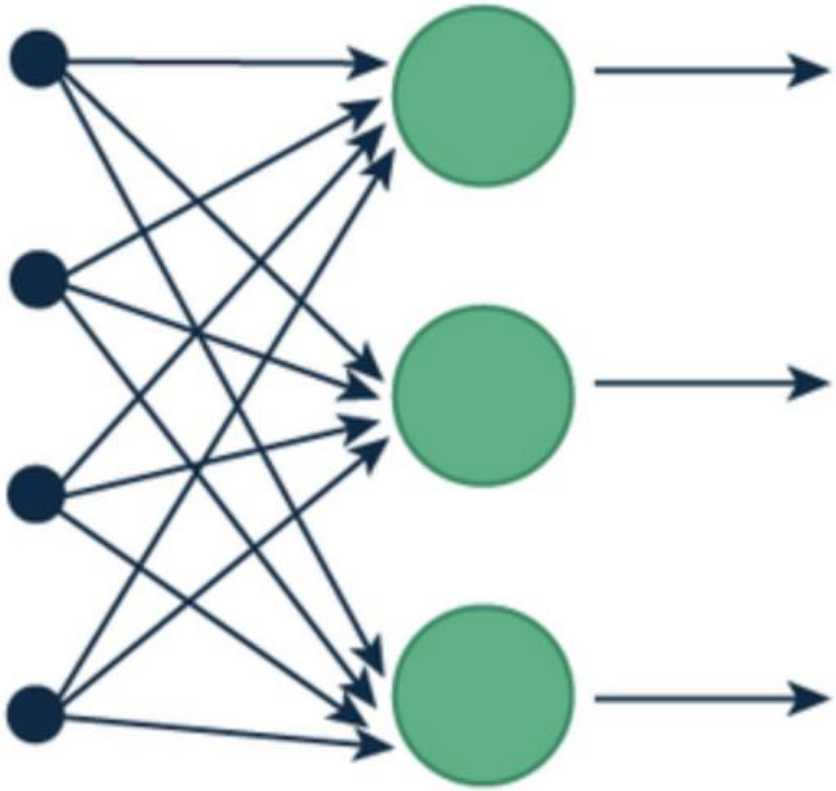
# CS 7150: Deep Learning



Presented by: Dr. Tala Talaei Khoei



(a) Recurrent Neural Network

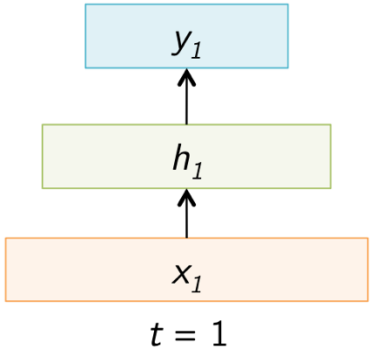


(b) Feed-Forward Neural Network

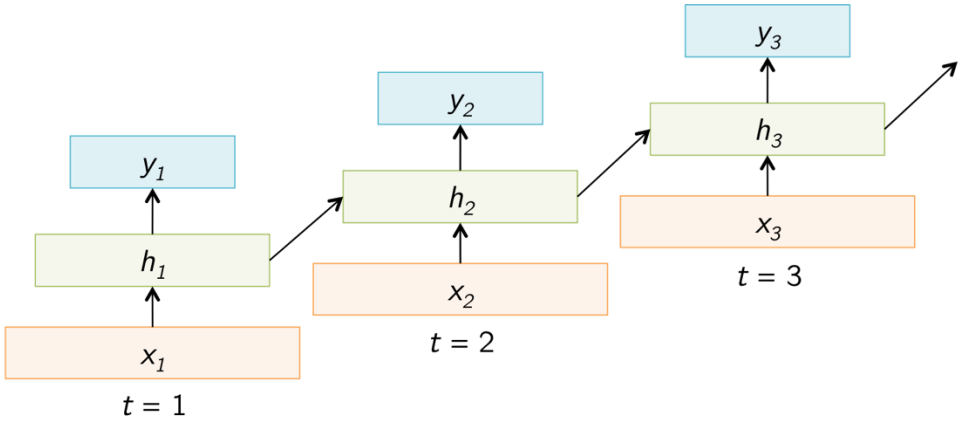
# Recurrent Neural Network (RNN)

- If you want to make predictions on sequential or time series data (e.g., text, audio, etc.) traditional neural networks are a bad choice.

- In time series data, the current observation depends on previous observations, and thus observations are not independent from each other. Traditional neural networks, however, view each observation as independent as the networks are not able to retain past or historical information. Basically, they have no memory of what happened in the past.

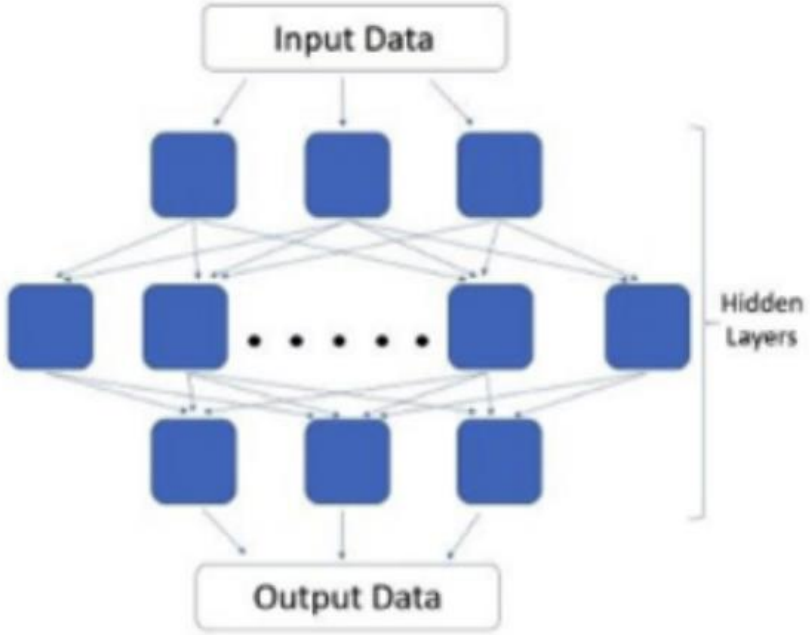



A Simple DNN



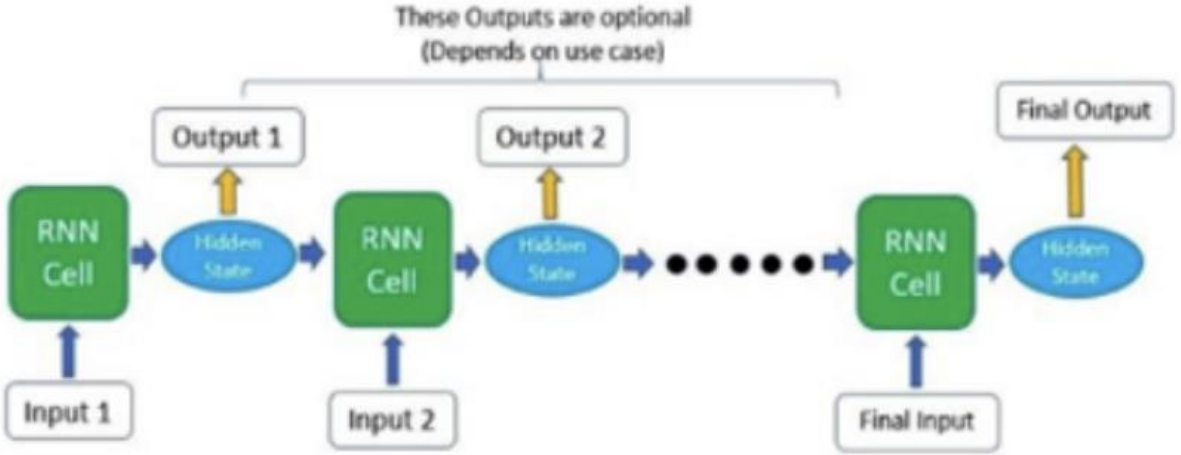

A Simple RNN

## Traditional Feed-Forward Network



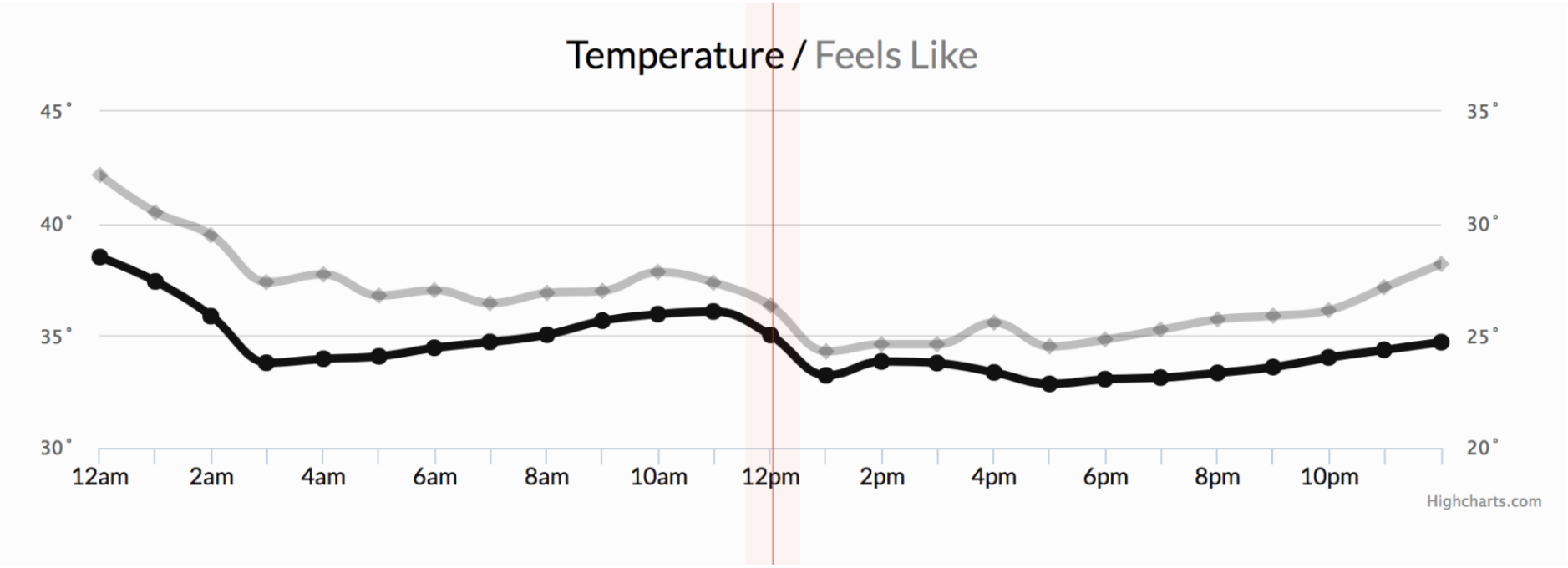

VS

## Recurrent Neural Network



**What is Time Series Data?**

A time series is a collection of data points gathered over a period of time and ordered chronologically. The primary characteristic of a time series is that it's indexed or listed in time order, which is a critical distinction from other types of data sets. If you were to plot the points of time series data on a graph, and one of your axes would always be time. Time series metrics refer to a piece of data that is tracked at an increment in time. For instance, a metric could refer to how much inventory was sold in a store from one day to the next.



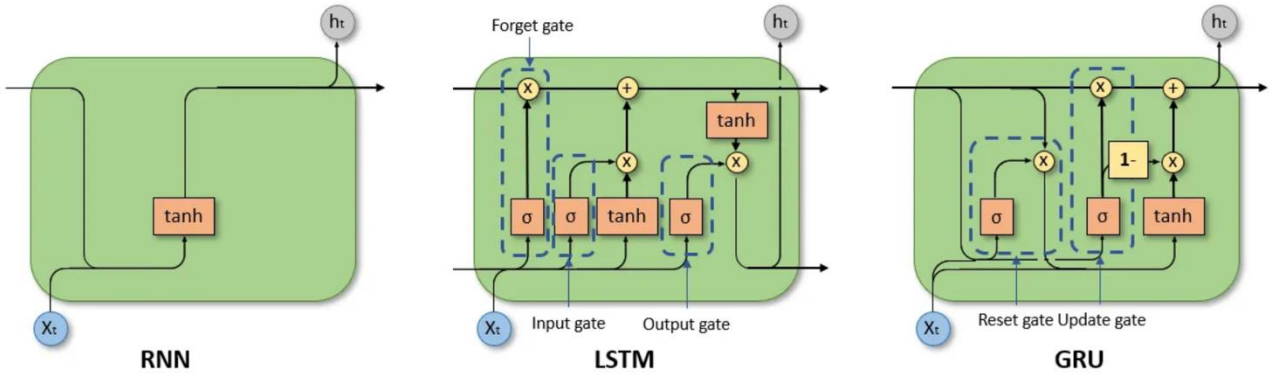

Example 1: Weather conditions

- This led to the rise of Recurrent Neural Networks (RNNs), which introduce the concept of memory to neural networks by including the dependency between data points. With this, RNNs can be trained to remember concepts based on context, i.e., learn repeated patterns.

RNNs achieve a memory through a feedback loop in the cell. And this is the main difference between a RNN and a traditional neural network. The feed-back loop allows information to be passed within a layer in contrast to feed-forward neural networks in which information is only passed between layers.

RNNs must then define what information is relevant enough to be kept in the memory. For this, different types of RNN evolved:

- Traditional Recurrent Neural Network (RNN)
- Long-Short-term-Memory Recurrent Neural Network (LSTM)
- Gated Recurrent Unit Recurrent Neural Network (GRU)

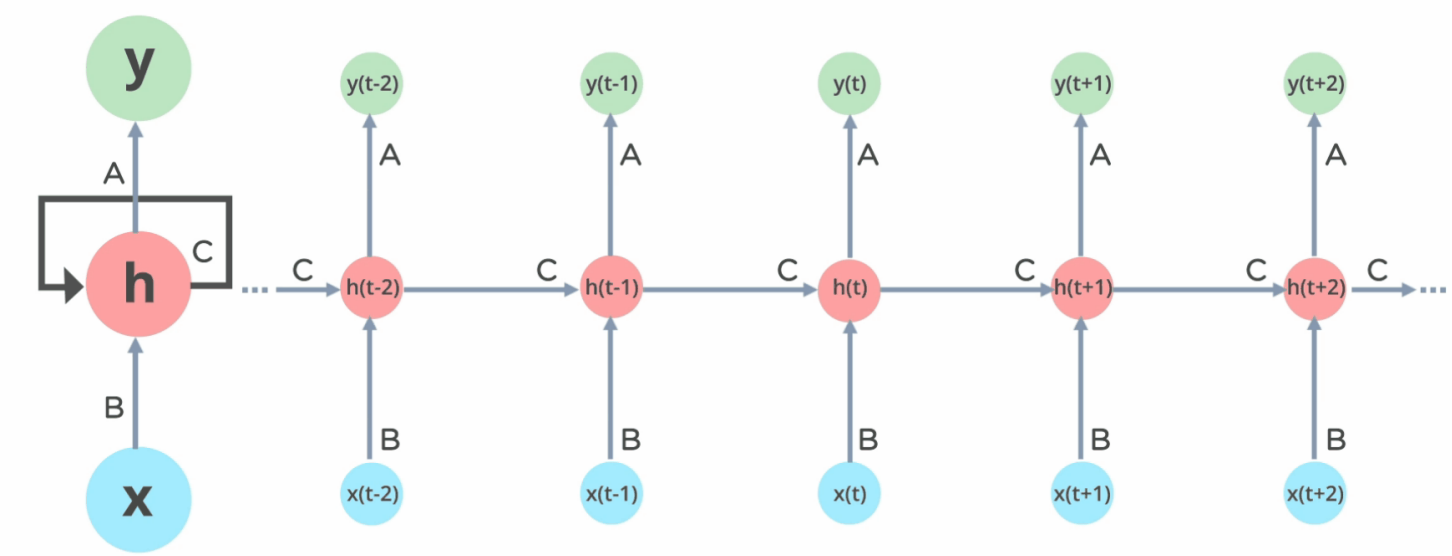


# Recurrent Neural Network

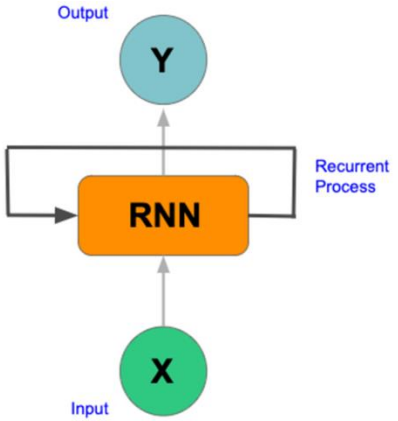
- Recurrent Neural Network remembers the past and its decisions are influenced by what it has learned from the past. Note: Basic feedforward networks “remember” things too, but they remember things they learned during training. For example, an image classifier learns what a “1” looks like during training and then uses that knowledge to classify things in production.

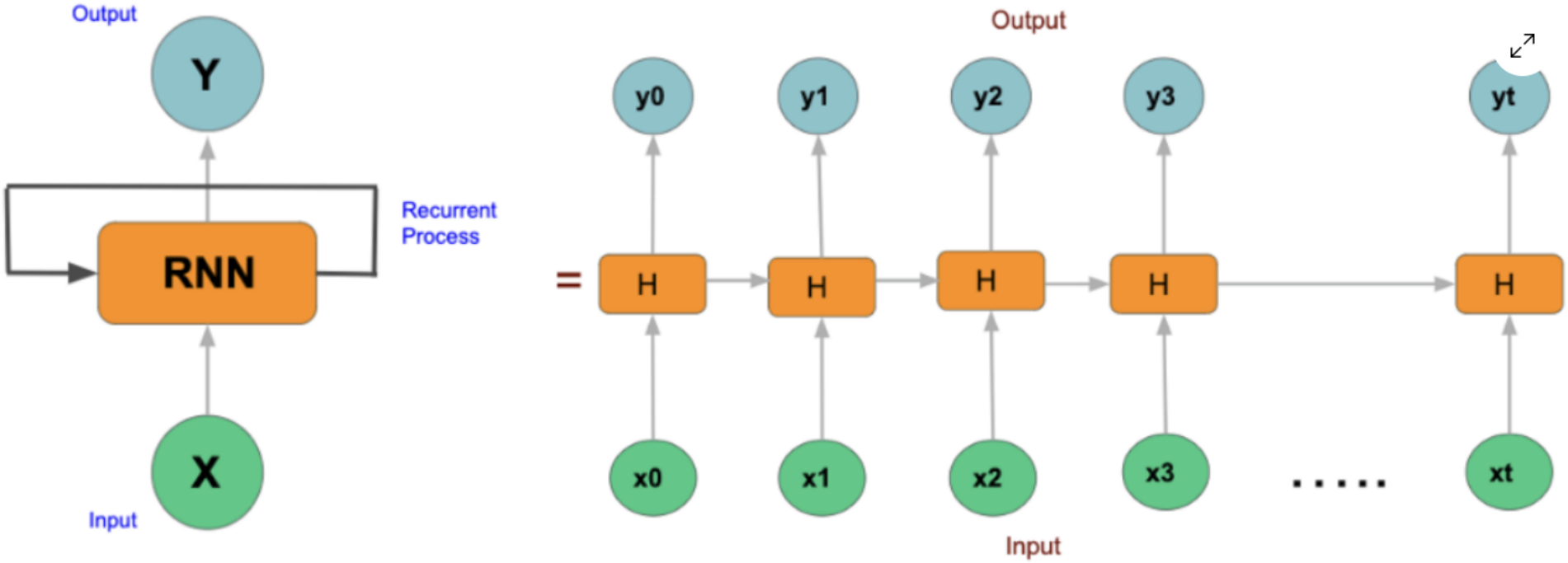
- While RNNs learn similarly while training, besides, they remember things learned from prior input(s) while generating output(s). It’s part of the network. RNNs can take one or more input vectors and produce one or more output vectors and the output(s) are influenced not just by weights applied on inputs like a regular NN, but also by a “hidden” state vector representing the context based on prior input(s)/output(s). So, the same input could produce a different output depending on previous inputs in the series.

- RNNs are called recurrent because they perform the same task for every element of a sequence, with the output being dependent on the previous computations and you already know that they have a “memory” which captures information about what has been calculated so far.



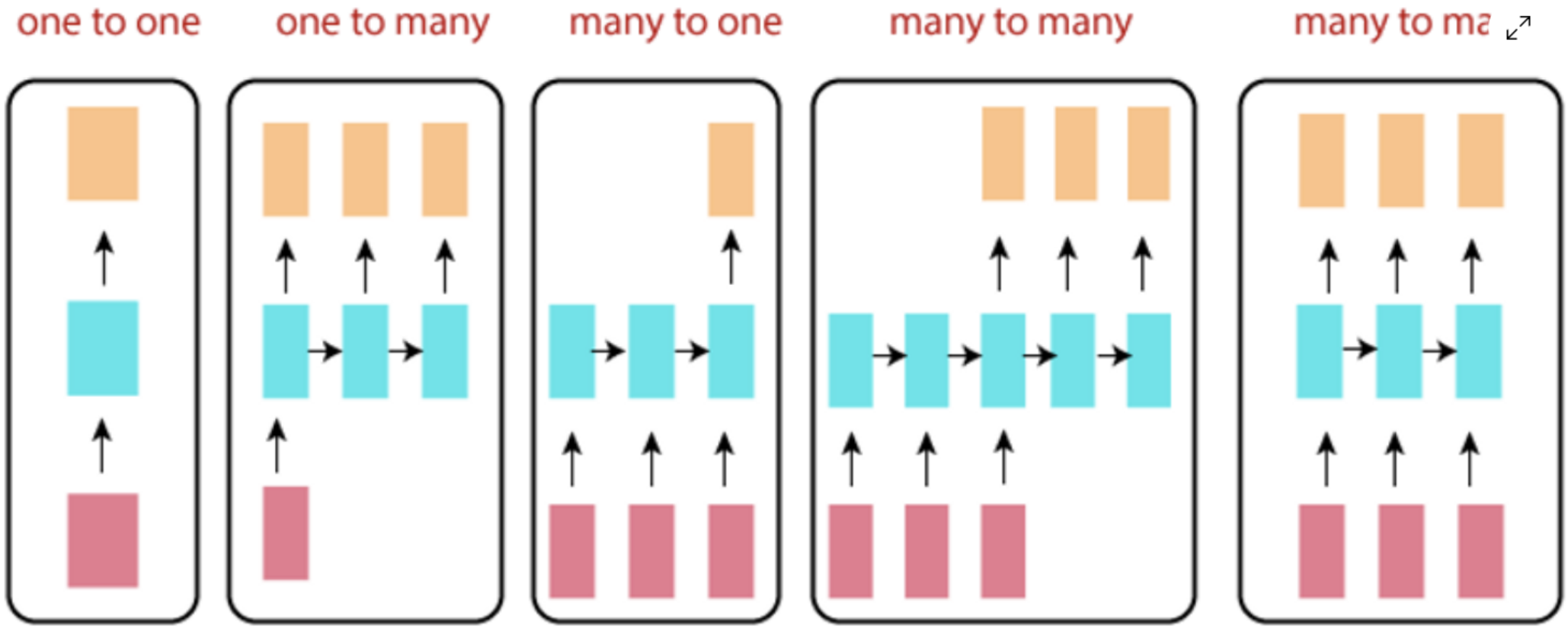
- A Recurrent Neural Network with input X and output Y with multiple recurrent steps and a hidden unit.
- This may be browbeating at first sight, but once unfolded, into a full network it looks a lot simpler.
- The below diagram shows an RNN being unrolled (or unfolded). By unrolling we simply mean that we write out the network for the complete sequence. For example, if the sequence we care about is a sentence of 5 words, the network would be unrolled into a 5-layer neural network, one layer for each word.



# Types of RNN

RNN models are mostly used in the fields of natural language processing(NLP) and speech recognition. The main reason that the recurrent nets are more exciting is that they allow us to operate over sequences of vectors: Sequence in the input, the output, or in the most general case, both.




Different types of RNN

# Definitions

- **One-to-one:** This is also called **Plain Neural networks**. It deals with a fixed size of the input to the fixed size of output, where they are independent of previous information/output.

**Example**: Image classification.

- **One-to-Many:** It deals with a fixed size of information as input that gives a sequence of data as output.

**Example:** Image Captioning takes the image as input and outputs a sentence of words.

- **Many-to-One:** It takes a sequence of information as input and outputs a fixed size of the output.

**Example:** sentiment analysis where any sentence is classified as expressing the positive or negative sentiment.

- **Many-to-Many:** It takes a Sequence of information as input and processes the recurrently outputs as a Sequence of data.

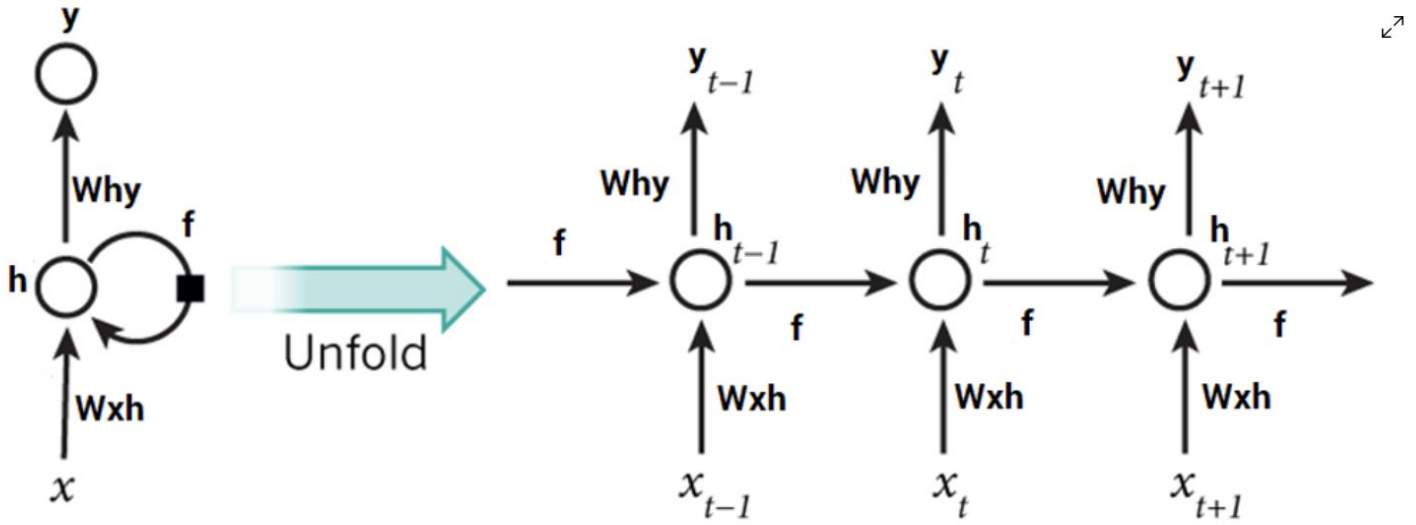
**Example:** Machine Translation, where the RNN reads any sentence in English and then outputs the sentence in French.

- **Bidirectional Many-to-Many:** Synced sequence input and output. Notice that in every case are no pre-specified constraints on the lengths sequences because the recurrent transformation (green) is fixed and can be applied as many times as we like.

**Example:** Video classification where we wish to label every frame of the video.

# How RNN Work?

- Consider an unfolded RNN:




- The formula for the current state can be written as:

$$h_t = f(h_{t-1}, x_t)$$

New state

Previous state

Current input

- Taking the simplest form of a recurrent neural network, let's say that the activation function is **tanh**, the weight at the recurrent neuron is **Whh**, and the weight at the input the neuron is **Wxh**, we can write the equation for the state at the time **t** as:

$$h_t = \tanh (W_{hh}h_{t-1} + W_{xh}x_t)$$

- The Recurrent neuron, in this case, is just considering the immediately previous state. For longer sequences, the equation can involve multiple such states. Once the final state is calculated we can go on to produce the output.

- Now, once the current state is calculated we can calculate the output state as:

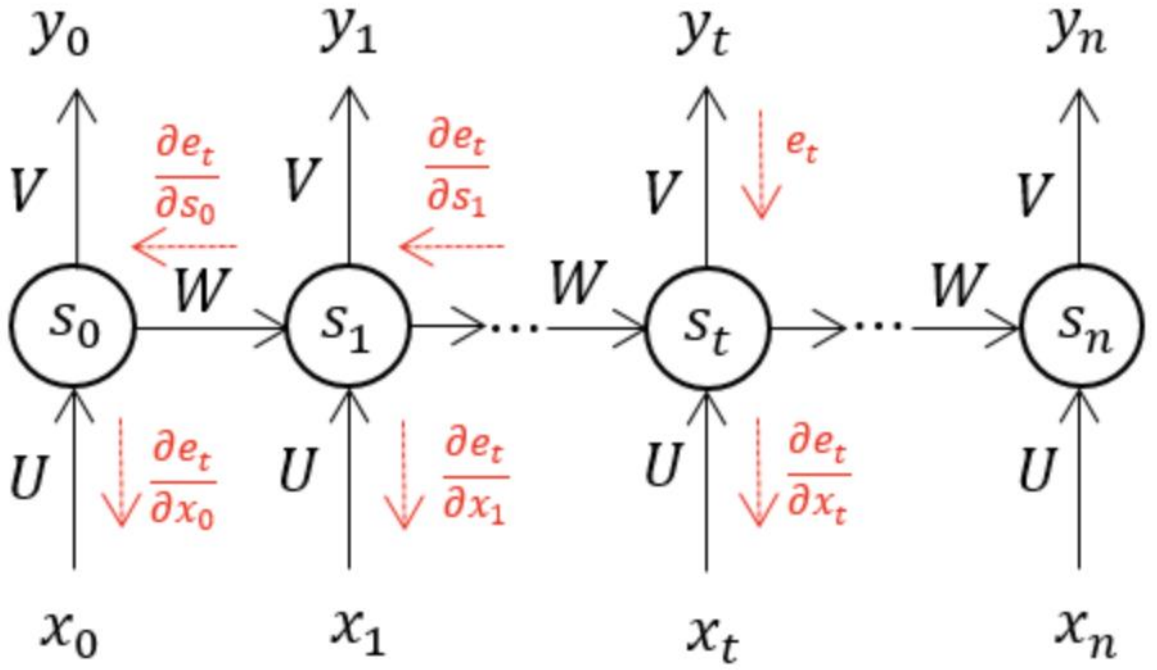
$$y_t = W_{hy}h_t$$

## Steps

1. A single time step of the input is supplied to the network i.e. **xt** is supplied to the network
2. We then calculate its current state using a combination of the current input and the previous state i.e. we calculate **ht**
3. The current ht becomes **ht-1** for the next time step
4. We can go as many time steps as the problem demands and combine the information from all the previous states
5. Once all the time steps are completed the final current state is used to calculate the output **yt**
6. The output is then compared to the actual output and the error is generated
7. The error is then backpropagated to the network to update the weights(we shall go into the details of backpropagation in further sections) and the network is trained.

# Backpropagation Through Time(BPTT)

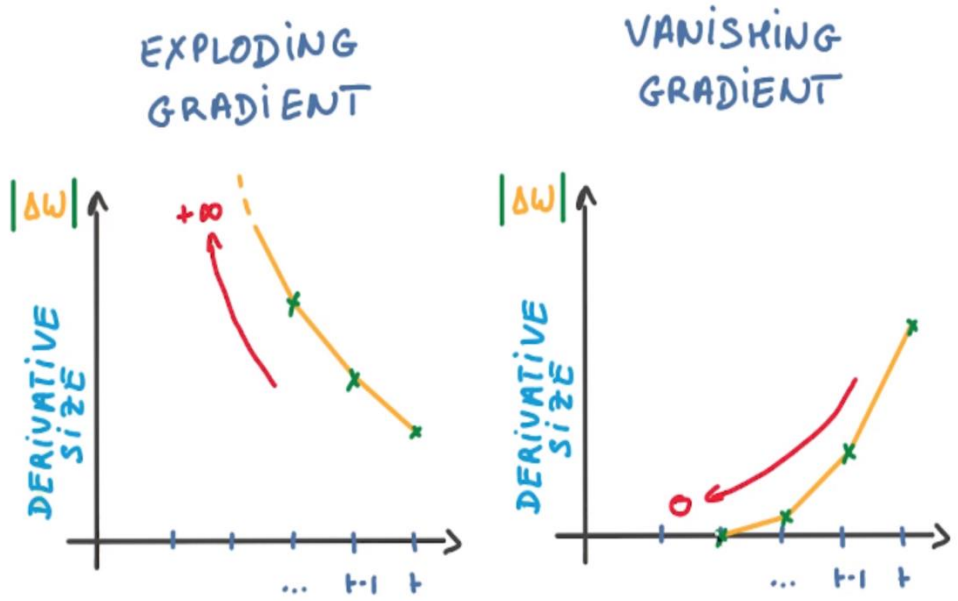
- Training an RNN is similar to training a traditional Neural Network. We also use the backpropagation algorithm, but with a little twist. Because the parameters are shared by all-time steps in the network, the gradient at each output depends not only on the calculations of the current time step but also on the previous time steps.
- For example, to calculate the gradient at t = 4, we would need to backpropagate 3 steps and sum up the gradients. This is called Backpropagation Through Time (BPTT). The RNNs trained with BPTT have difficulties learning long-term dependencies (e.g. dependencies between steps that are far apart) due to what is called the vanishing/exploding gradient problem. There exists some machinery to deal with these problems, and certain types of RNNs (like LSTMs) were specifically designed to get around them.



# Problem of Long-Term Dependencies

Unfortunately, if you implement the above steps, you won't be so delighted with the results. That is because the simplest RNN model has three major drawbacks, called vanishing gradient, gradient exploding, and slow training time which prevents it from being accurate.

- **Exploding Gradients:** Exploding gradients occur when the algorithm gives the weights an absurdly high priority for no apparent reason. Fortunately, truncating or squashing the gradients is a simple solution to this problem.
- **Vanishing Gradients:** Vanishing gradients occur when the gradient values are too small, causing the model to stop learning or take far too long. This was a big issue in the 1990s, and it was far more difficult to address than the exploding gradients. Fortunately, Sepp Hochreiter and Juergen Schmidhuber's LSTM concept solved the problem.



## RNN Applications

Recurrent Neural Networks are used to tackle a variety of problems involving sequence data. There are many different types of sequence data, but the following are the most common: Audio, Text, Video, Biological sequences. Using RNN models and sequence datasets, you may tackle a variety of problems, including :

- Speech recognition
- Generation of music
- Automated Translations
- Analysis of video action
- Sequence study of the genome and DNA

**Advantages of RNNs:**

- Handle sequential data effectively, including text, speech, and time series.
- Process inputs of any length, unlike feedforward neural networks.
- Share weights across time steps, enhancing training efficiency.

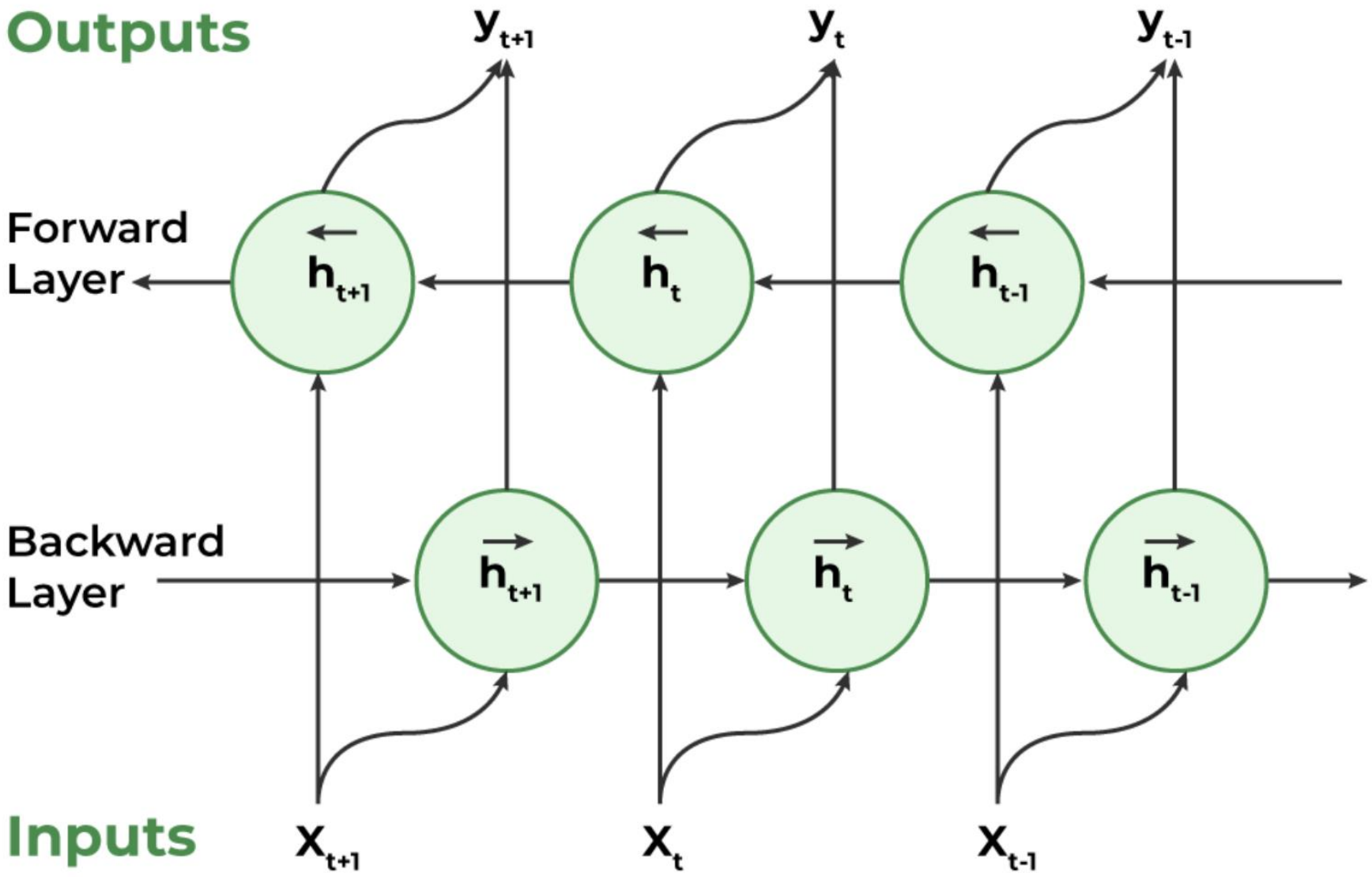
**Disadvantages of RNNs:**

- Prone to vanishing and exploding gradient problems, hindering learning.
- Training can be challenging, especially for long sequences.
- Computationally slower than other neural network architectures.

# Please refer to Codes on Canvas

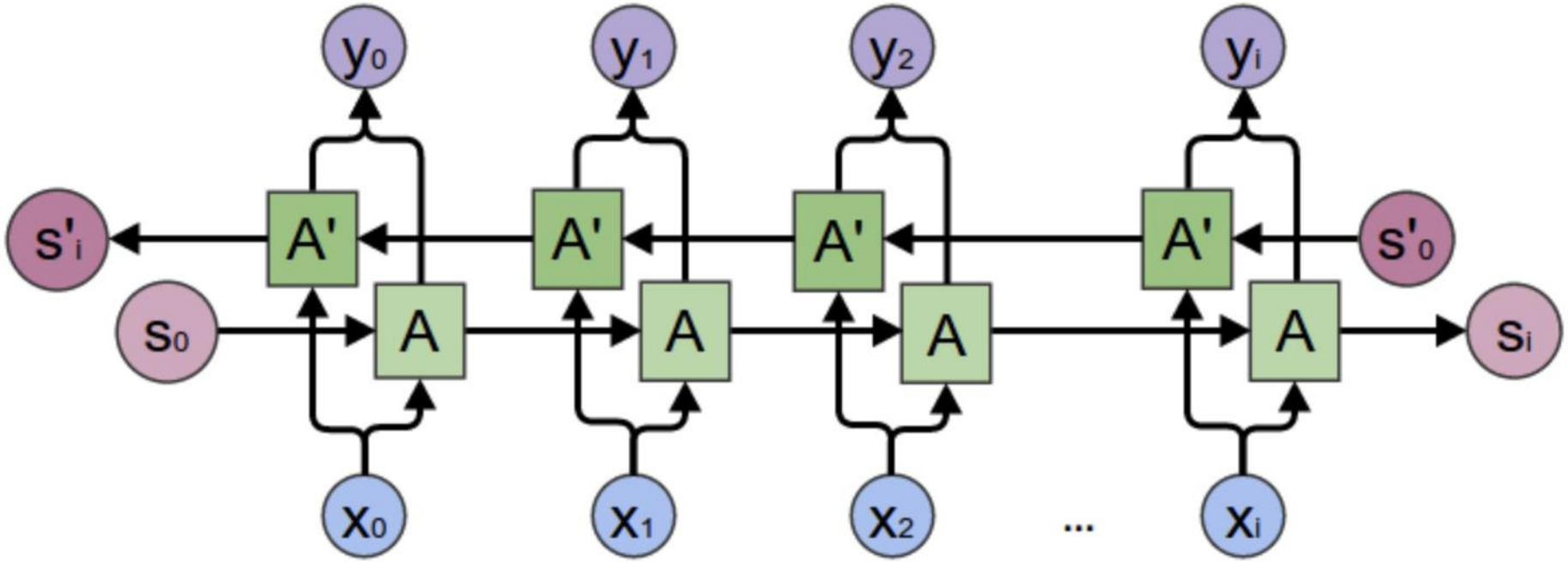
# Bidirectional Recurrent Neural Network (RNN)

- An architecture of a neural network called a bidirectional recurrent neural network (BRNN) is made to process sequential data. In order for the network to use information from both the past and future context in its predictions, BRNNs process input sequences in both the forward and backward directions. This is the main distinction between BRNNs and conventional recurrent neural networks.
- A BRNN has two distinct recurrent hidden layers, one of which processes the input sequence forward and the other of which processes it backward. After that, the results from these hidden layers are collected and input into a prediction-making final layer. Any recurrent neural network cell, such as Long Short-Term Memory (LSTM) or Gated Recurrent Unit, can be used to create the recurrent hidden layers.
- The BRNN functions similarly to conventional recurrent neural networks in the forward direction, updating the hidden state depending on the current input and the prior hidden state at each time step. The backward hidden layer, on the other hand, analyses the input sequence in the opposite manner, updating the hidden state based on the current input and the hidden state of the next time step.
- Compared to conventional unidirectional recurrent neural networks, the accuracy of the BRNN is improved since it can process information in both directions and account for both past and future contexts. Because the two hidden layers can complement one another and give the final prediction layer more data, using two distinct hidden layers also offers a type of model regularization.




*Bi-directional Recurrent Neural Network*

To enable straight (past) and reverse traversal of input (future), Bidirectional RNNs, or BRNNs, are used. A BRNN is a combination of two RNNs - one RNN moves forward, beginning from the start of the data sequence, and the other, moves backward, beginning from the end of the data sequence. The network blocks in a BRNN can either be simple RNNs, GRUs, or LSTMs.



## Training Bidirectional Recurrent Neural Networks

- Training a BRNN is similar to training a traditional RNN, but it involves updating two sets of weights simultaneously—one for the forward direction and one for the backward direction. The training process involves forward propagation, where input data is passed through both RNNs, and backward propagation, where gradients from the output layer are passed back through the network to update the weights. Both forward and backward passes are necessary to capture the dependencies in both directions.
- One of the challenges with training BRNNs, similar to other RNNs, is the issue of vanishing and exploding gradients. However, this can be mitigated by using gated units such as Long Short-Term Memory (LSTM) units or Gated Recurrent Units (GRUs) in the architecture.

## Application

BRNN is useful for the following applications:

- Handwriting Recognition
- Speech Recognition
- Dependency Parsing
- Natural Language Processing

## Limitations of Bidirectional Recurrent Neural Networks

Despite their advantages, BRNNs also have some limitations. They require the complete sequence to be known before processing, which makes them unsuitable for real-time processing tasks where the entire sequence is not available upfront. Additionally, BRNNs can be more computationally intensive to train due to the doubled number of calculations for the forward and backward passes.

## Advantages of Bidirectional Recurrent Neural Networks

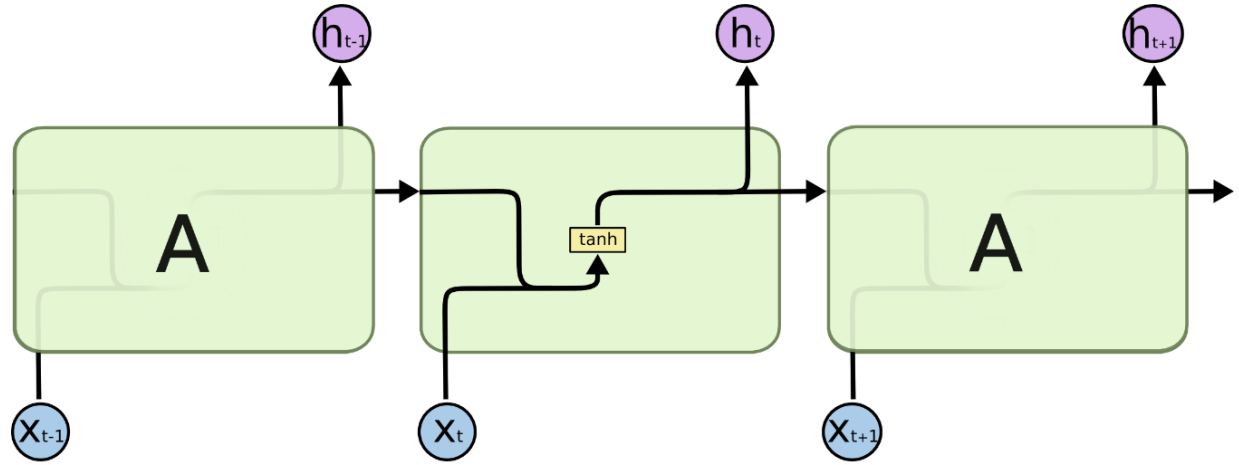
The primary advantage of BRNNs is their ability to take into account both past and future data, which can lead to more accurate models for certain types of problems. This bidirectional context is particularly beneficial in tasks where the complete sequence is known and the context from both directions is necessary to understand the content.

# Please refer to Codes on Canvas

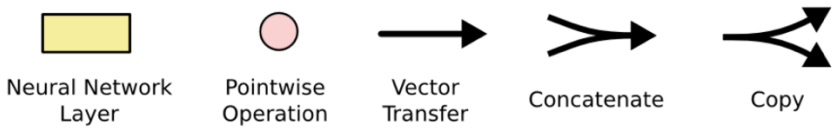
# Long Short-Term Memory

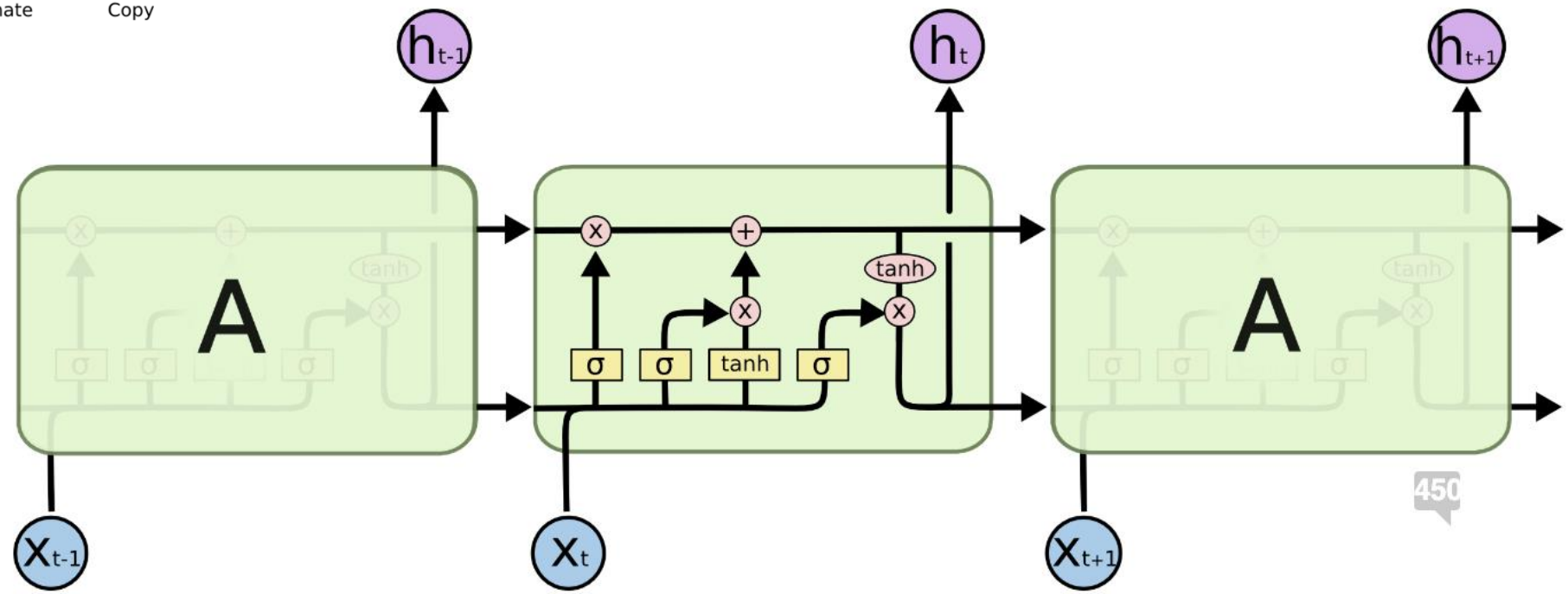
**Long Short-Term Memory** is an improved version of recurrent neural network designed by Hochreiter & Schmidhuber. **LSTM** is well-suited for sequence prediction tasks and excels in capturing long-term dependencies. Its applications extend to tasks involving time series and sequences. LSTM's strength lies in its ability to grasp the order dependence crucial for solving intricate problems, such as machine translation and speech recognition.

- Long Short Term Memory networks – usually just called "LSTMs" – are a special kind of RNN, capable of learning long-term dependencies. They were introduced by Hochreiter & Schmidhuber (1997), and were refined and popularized by many people in following work.[1] They work tremendously well on a large variety of problems, and are now widely used.
- LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn!
- All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.
- LSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way.




The repeating module in a standard RNN contains a single layer.



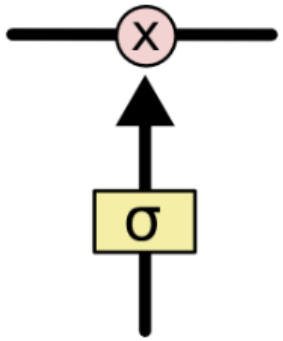

The repeating module in an LSTM contains four interacting layers.

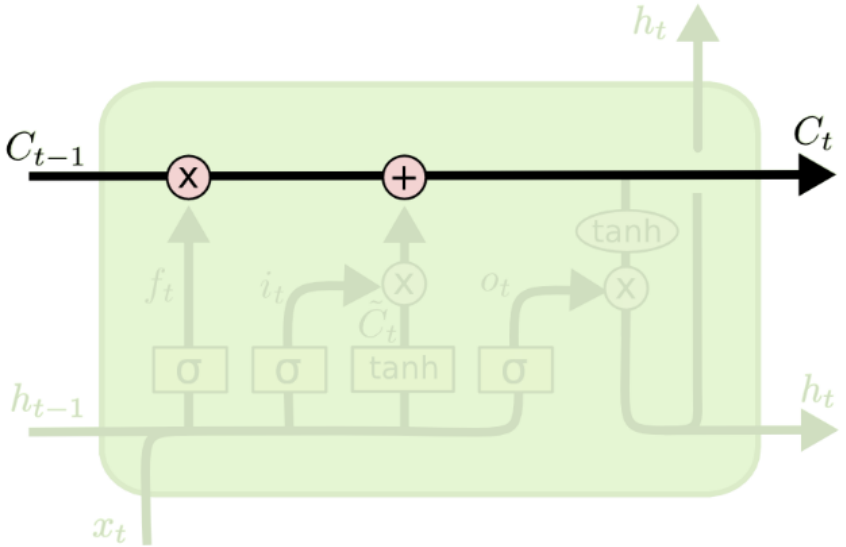
**The Core Idea Behind LSTMs**

- 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. It's very easy for information to just flow along it unchanged.

- The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates.
- Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.

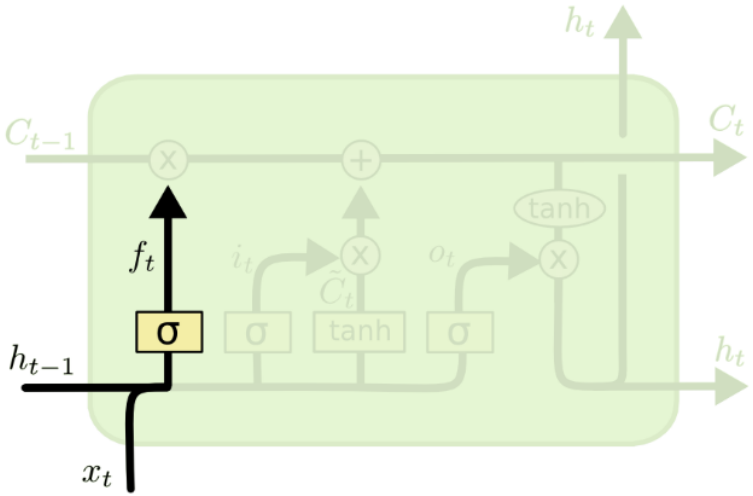
The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. A value of zero means "let nothing through," while a value of one means "let everything through!"
An LSTM has three of these gates, to protect and control the cell state.



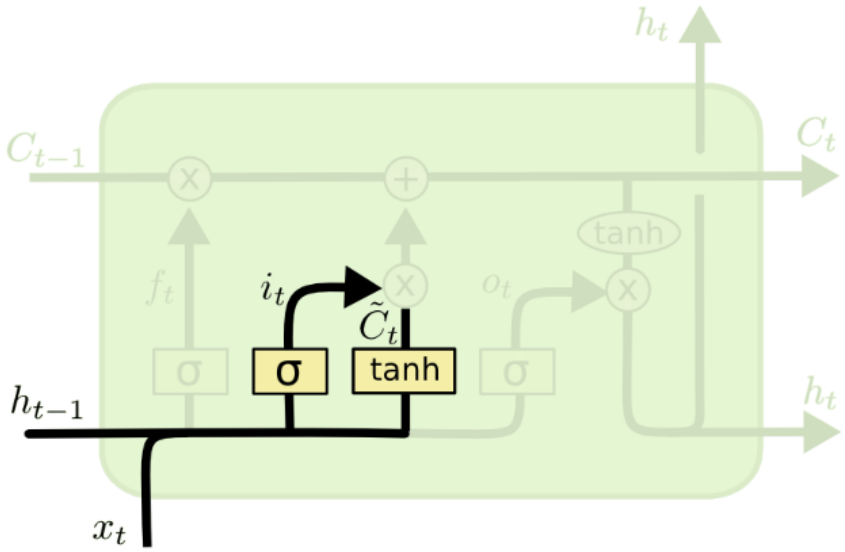
# Step-by-Step LSTM Walk Through

- 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." It looks at ht−1 and xt, and outputs a number between 00 and 11 for each number in the cell state Ct−1. A 1 represents "completely keep this" while a 00 represents "completely get rid of this."
- Let's go back to our example of a language model trying to predict the next word based on all the previous ones. In such a problem, the cell state might include the gender of the present subject, so that the correct pronouns can be used. When we see a new subject, we want to forget the gender of the old subject.




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

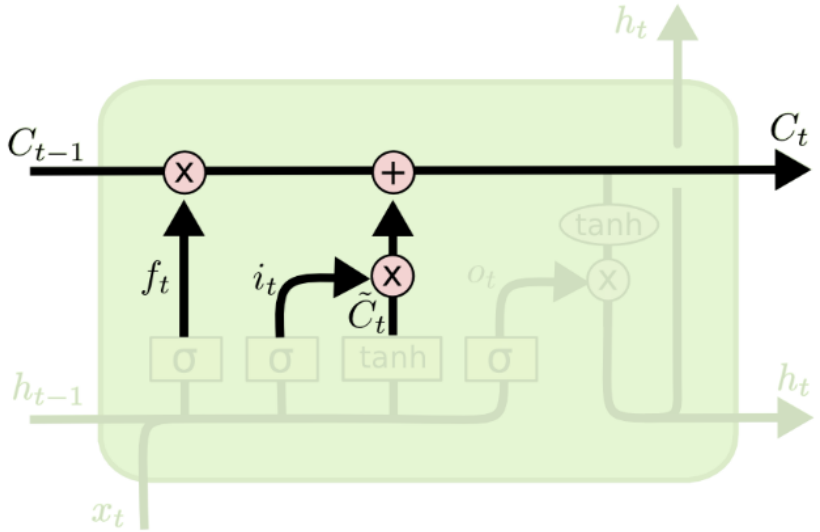
The next step is to decide what new information we're going to store in the cell state. This has two parts. First, a sigmoid layer called the "input gate layer" decides which values we'll update. Next, a tanh layer creates a vector of new candidate values, C̃ t, that could be added to the state. In the next step, we'll combine these two to create an update to the state.
In the example of our language model, we'd want to add the gender of the new subject to the cell state, to replace the old one we're forgetting.




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

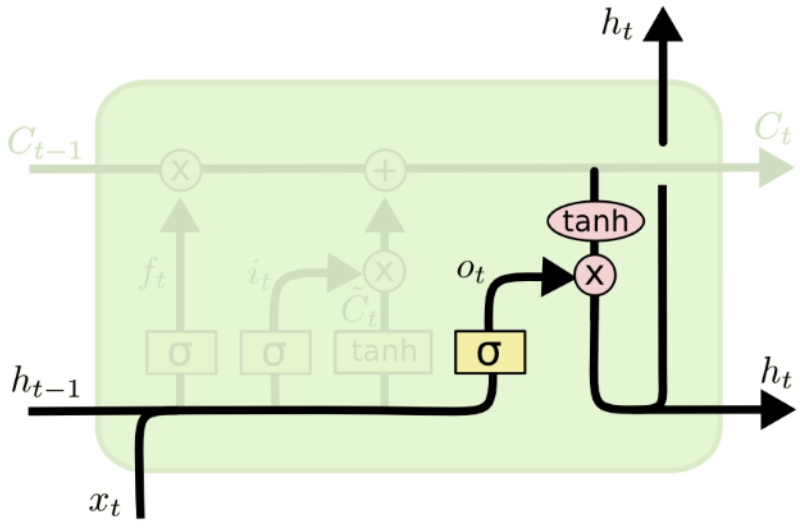
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

It’s now time to update the old cell state, Ct−1, into the new cell state Ct The previous steps already decided what to do, we just need to actually do it.
We multiply the old state by ft, forgetting the things we decided to forget earlier. Then we add it∗C̃ t. This is the new candidate values, scaled by how much we decided to update each state value.
In the case of the language model, this is where we’d actually drop the information about the old subject’s gender and add the new information, as we decided in the previous steps.




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

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. Then, we put the cell state through tanhtanh (to push the values to be between −1−1 and 11) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.
For the language model example, since it just saw a subject, 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, so that we know what form a verb should be conjugated into if that's what follows next.




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

$$h_t = o_t * \tanh\left(C_t\right)$$

**The advantages of LSTM (Long-Short Term Memory) are as follows:**

- Long-term dependencies can be captured by LSTM networks. They have a memory cell that is capable of long-term information storage.
- In traditional RNNs, there is a problem of vanishing and exploding gradients when models are trained over long sequences. By using a gating mechanism that selectively recalls or forgets information, LSTM networks deal with this problem.
- LSTM enables the model to capture and remember the important context, even when there is a significant time gap between relevant events in the sequence. So where understanding context is important, LSTMS are used. eg. machine translation.

**The disadvantages of LSTM (Long-Short Term Memory) are as follows:**

- Compared to simpler architectures like feed-forward neural networks LSTM networks are computationally more expensive. This can limit their scalability for large-scale datasets or constrained environments.
- Training LSTM networks can be more time-consuming compared to simpler models due to their computational complexity. So training LSTMs often requires more data and longer training times to achieve high performance.
- Since it is processed word by word in a sequential manner, it is hard to parallelize the work of processing the sentences.

# Applications of LSTM

Some of the famous applications of LSTM includes:

- **Language Modeling:** LSTMs have been used for natural language processing tasks such as language modeling, machine translation, and text summarization. They can be trained to generate coherent and grammatically correct sentences by learning the dependencies between words in a sentence.
- **Speech Recognition:** LSTMs have been used for speech recognition tasks such as transcribing speech to text and recognizing spoken commands. They can be trained to recognize patterns in speech and match them to the corresponding text.
- **Time Series Forecasting:** LSTMs have been used for time series forecasting tasks such as predicting stock prices, weather, and energy consumption. They can learn patterns in time series data and use them to make predictions about future events.
- **Anomaly Detection:** LSTMs have been used for anomaly detection tasks such as detecting fraud and network intrusion. They can be trained to identify patterns in data that deviate from the norm and flag them as potential anomalies.
- **Recommender Systems:** LSTMs have been used for recommendation tasks such as recommending movies, music, and books. They can learn patterns in user behavior and use them to make personalized recommendations.
- **Video Analysis:** LSTMs have been used for video analysis tasks such as object detection, activity recognition, and action classification. They can be used in combination with other neural network architectures, such as Convolutional Neural Networks (CNNs), to analyze video data and extract useful information.

## LTSM vs RNN

| Feature | LSTM (Long Short-term Memory) | RNN (Recurrent Neural Network) |
|---|---|---|
| **Memory** | Has a special memory unit that allows it to learn long-term dependencies in sequential data | Does not have a memory unit |
| **Directionality** | Can be trained to process sequential data in both forward and backward directions | Can only be trained to process sequential data in one direction |
| **Training** | More difficult to train than RNN due to the complexity of the gates and memory unit | Easier to train than LSTM |
| **Long-term dependency learning** | Yes | Limited |
| **Ability to learn sequential data** | Yes | Yes |
| **Applications** | Machine translation, speech recognition, text summarization, natural language processing, time series forecasting | Natural language processing, machine translation, speech recognition, image processing, video processing |

# Please refer to Codes on Canvas


THANK YOU
Any Questions?