

Deep Generative Models

Prepared by Dr. Sherif Mostafa

AI314

Master of Science in Artificial Intelligence

Lecture 5



Autoregressive Models

Text Data

What is LSTM

The RNN Architecture

The LSTM Architecture

Application Of LSTM

Autoregressive Models

Autoregressive models are a class of machine learning (ML) models that automatically predict the next component in a sequence by taking measurements from previous inputs in the sequence.

Autoregression is a statistical technique used in time-series analysis that assumes that the current value of a time series is a function of its past values.

Autoregressive models use similar mathematical techniques to determine the probabilistic correlation between elements in a sequence. They then use the knowledge derived to guess the next element in an unknown sequence.

Dealing with sequential data is considered one of the hardest problems to solve in the Data science industry. Standard neural networks have paved a new path in the rising AI industry. It worked very well with tabular data but every new invention in technology comes with a disadvantage. Similarly, ANN also came up with some gaps which as not being able to deal with Sequential data.

These include a wide range of problems like predicting the next word on your mobile keyboard or finding the patterns in stock markets' data.

With the recent discoveries that have been happening in this field, it has been observed that for almost all of these sequence problems, Long Short-Term Memory (LSTMs) has given the most appropriate solution.

Understanding Text Data

Sequential nature:

Text data is fundamentally different from image data. While image can be processed as a whole, text data is inherently sequential, with each word building upon the previous ones to convey meaning.

Contextual relationship:

The meaning of a word in the text can depend heavily on the context in which it appears. LSTMs are well suited to capturing these contextual relationship and understand the nuances of language.

Challenging in text generation:

Generating coherent and meaningful text is a complex task that requires understanding the structure and flow of language. LSTMs can be trained to generate text by learning patterns and relationships from a given corpus of text data.

LSTM applications in text generation

Time series forecasting:

LSTM can be used to make predictions on time series data, such as stock prices or weather patterns, by learning patterns and relationships in the data over time.

Sentiment analysis:

LSTMs can be trained to analyze the sentiment expressed in the text, such as customer reviews or social media posts, by understanding the contextual meaning of the words and phrases used.

Audio classification:

LSTM can be applied to audio data, such as speech or music, by processing nature of the audio signal and learning to classify different types of audio contents.

Text generation:

LSTMs have the ability to generate new text such as creative writing or dialogue by learning the patterns and structures of the language from given corpus of text data.

LSTM applications in text generation

So, in order to train LSTM model to predict the next word in a sequence we need to create a training set that captures the relationships between words. This process involves shifting the input sequence by one token to generate the target variable, allowing the model to learn the patterns in the data.

What is LSTM?

Long short-term memory networks (LSTM) are a special variant of Recurrent Neural Network (RNN). Just like you remember what happened a few minutes ago, LSTM remembers things that happened in the past, even if it was a long time ago. It can understand the meaning of words in a sentence by looking at the words that came before. For example, if I say, "I love ice cream because it is delicious," LSTM remembers that "it" refers to "ice cream" and helps the computer understand the whole sentence.

What is LSTM?

LSTM is really good at understanding and remembering patterns in information. It can be used for various applications like predicting the next word in a sentence, translating languages, and even controlling robots. It enables computers to understand and analyze sequential data in a way that is similar to what we do.

LSTM has become a powerful tool in artificial intelligence and deep learning, enabling breakthroughs in various fields by uncovering valuable insights from sequential data.

What is LSTM?

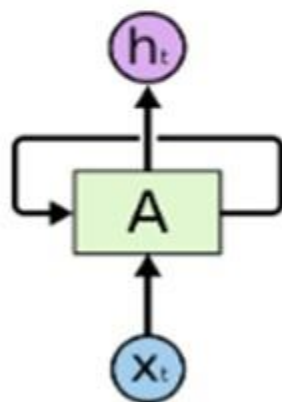
LSTM (Long Short-Term Memory) is a recurrent neural network (RNN) architecture widely used in Deep Learning. It excels at capturing long-term dependencies, making it ideal for sequence prediction tasks.

Unlike traditional neural networks, LSTM incorporates feedback connections, allowing it to process entire sequences of data, not just individual data points. This makes it highly effective in understanding and predicting patterns in sequential data like time series, text, and speech.

The RNN Architecture

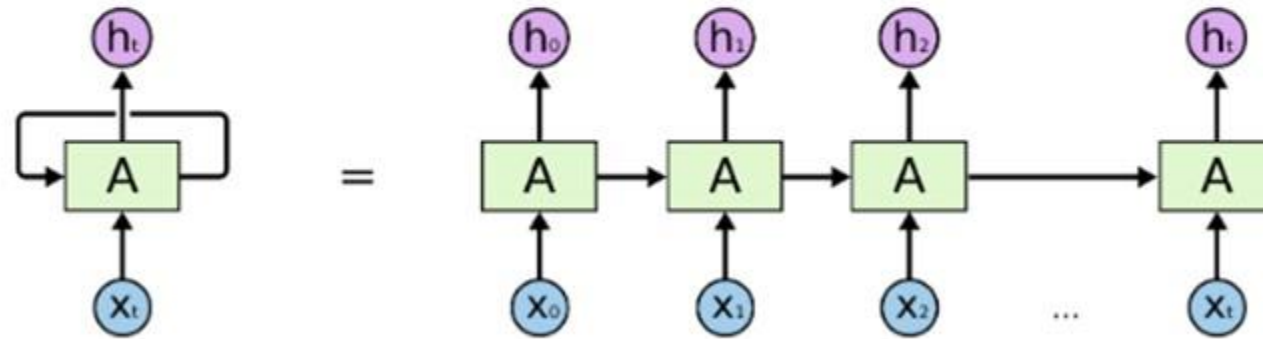
Recurrent neural networks address this issue. They are networks with loops in them, allowing information to persist.

In the following diagram, a chunk of neural network A looks at some input x_t and outputs a value h_t . A loop allows information to be passed from one step of the network to the next.



The RNN Architecture

These loops make recurrent neural networks seem kind of mysterious. However, if you think a bit more, it turns out that they aren't all that different than a normal neural network. A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor. Consider what happens if we unroll the loop:



Sometimes, we only need to look at recent information to perform the present task. For example, consider a language model trying to predict the next word based on the previous ones. If we are trying to predict the last word in “I grew up in France... I speak fluent French.” Recent information suggests that the next word is probably the name of a language, but if we want to narrow down which language, we need the context of France, from further back. It’s entirely possible for the gap between the relevant information and the point where it is needed to become very large.

Unfortunately, as that gap grows, RNNs become unable to learn to connect the information.

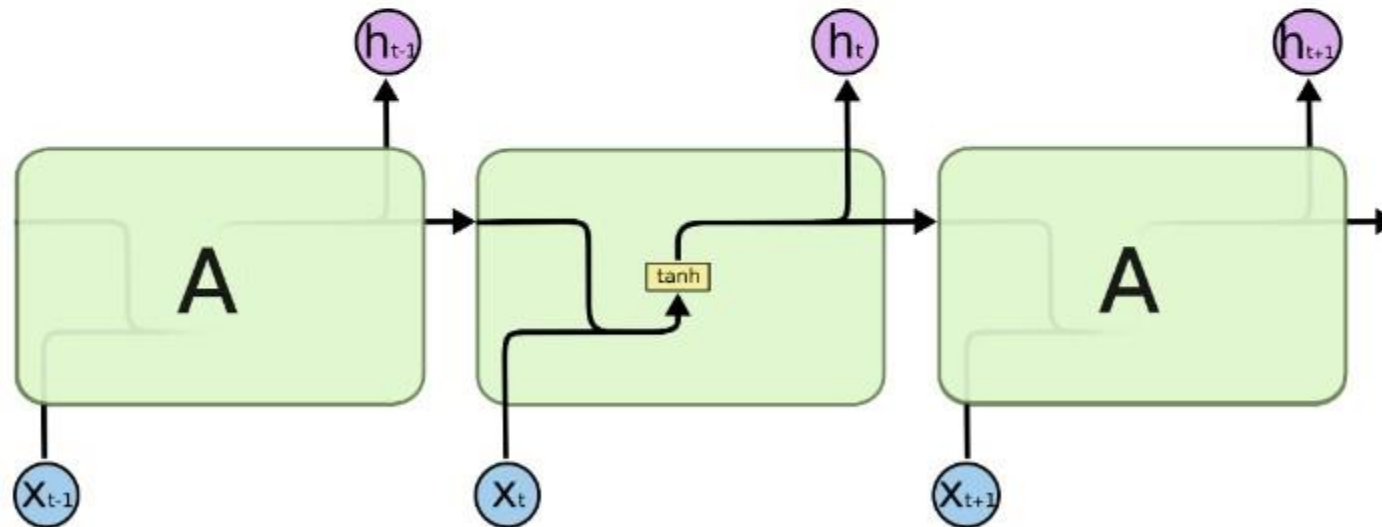
The LSTM Architecture

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)

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!

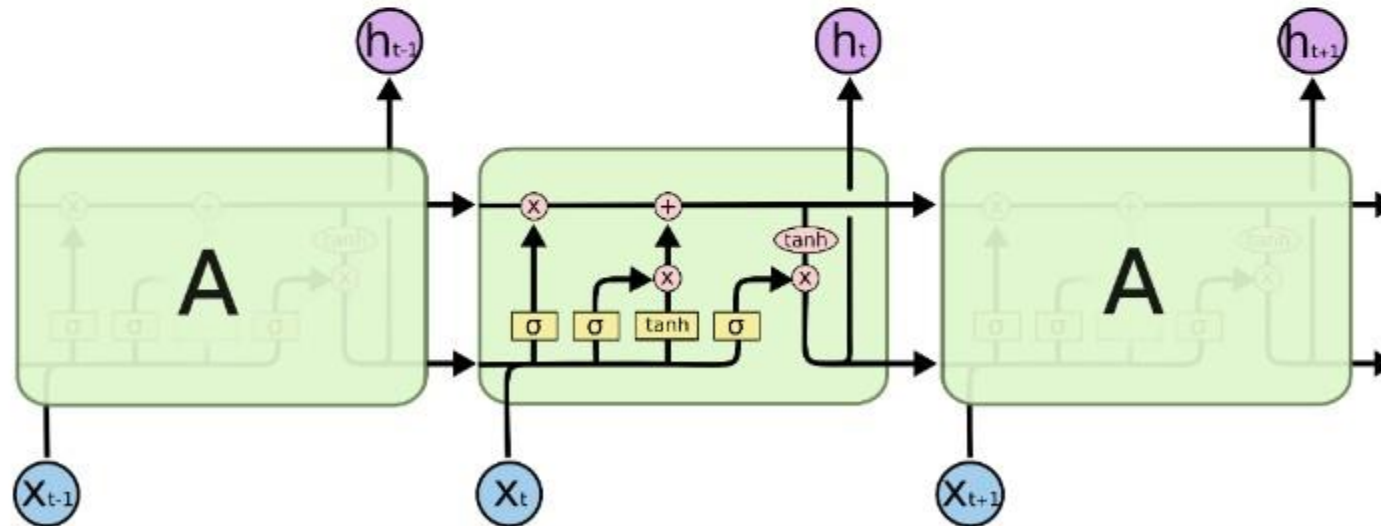
The LSTM Architecture

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.

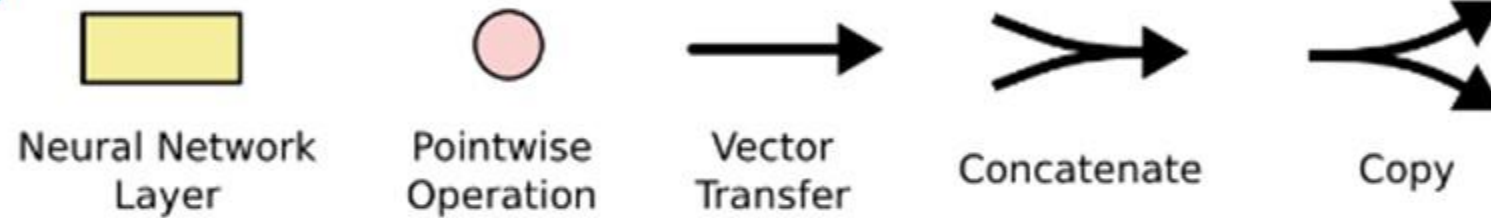


The LSTM Architecture

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.



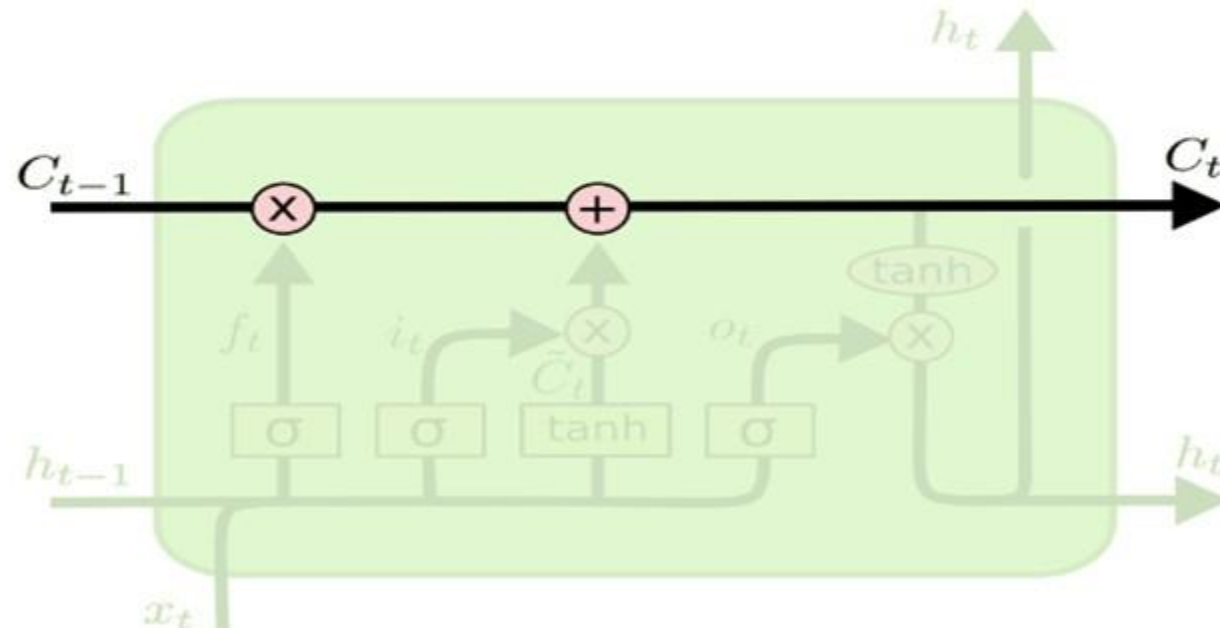
The LSTM Architecture



In the above diagram, each line carries an entire vector, from the output of one node to the inputs of others. The pink circles represent pointwise operations, like vector addition, while the yellow boxes are learned neural network layers. Lines merging denote concatenation, while a line forking denote its content being copied and the copies going to different locations.

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.

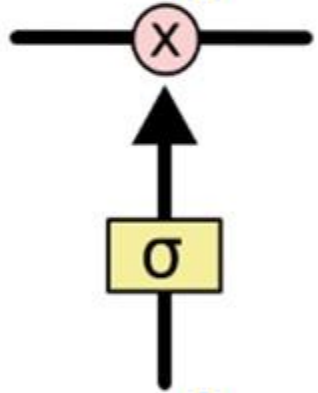


The Core Idea Behind LSTMs

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.



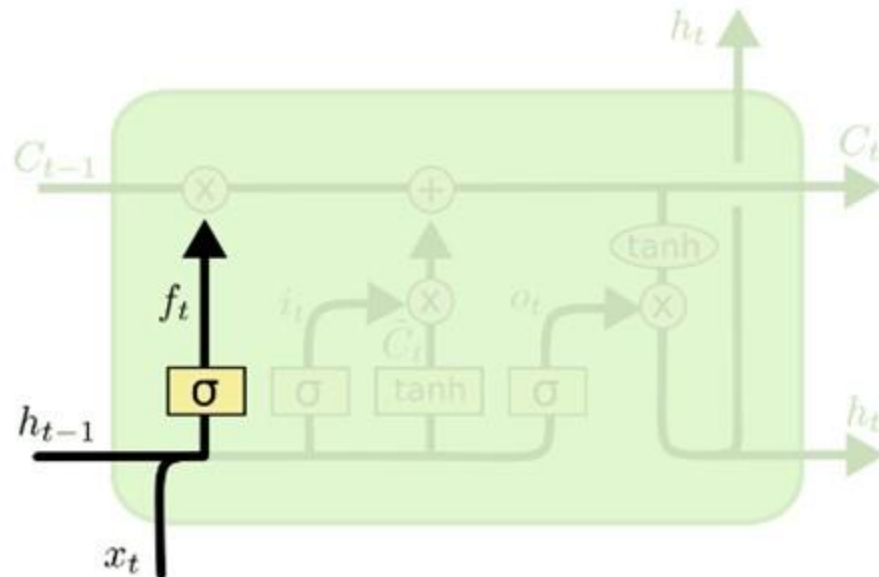
Step-by-Step LSTM Walk Through

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 h_{t-1} and x_t , and outputs a number between 0 and 1. for each number in the cell state C_{t-1} .

1 represents "completely keep this" while a 0 represents "completely get rid of this."

Step-by-Step LSTM Walk Through

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.



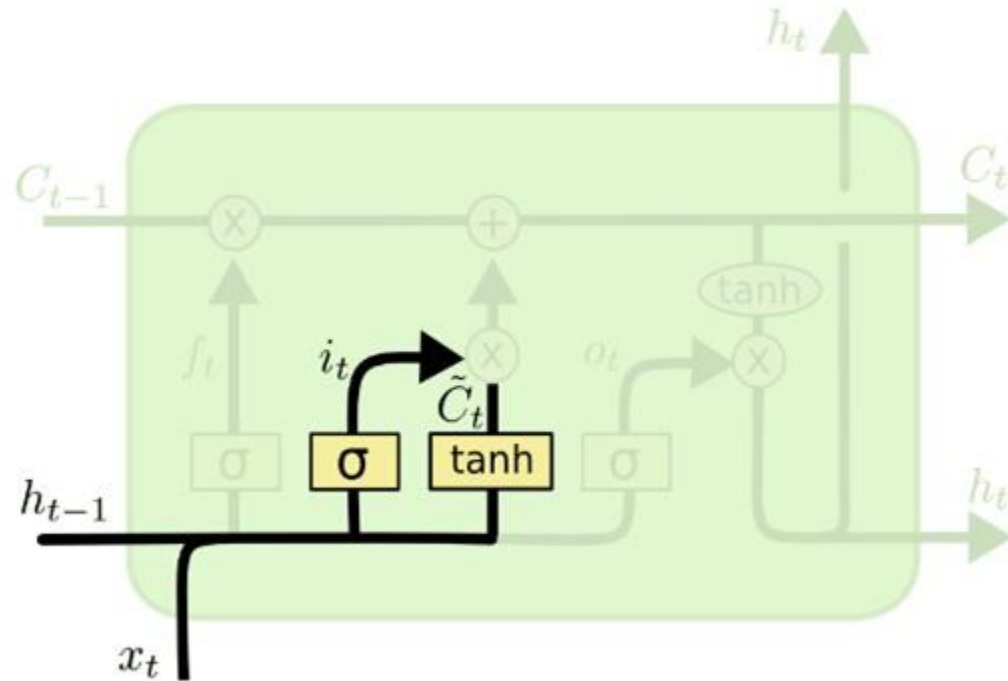
$$f_t = \sigma (W_f \cdot [h_{t-1}, x_t] + b_f)$$

The Core Idea Behind LSTMs

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, $\tilde{C}_t$, 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.

The Core Idea Behind LSTMs



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

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

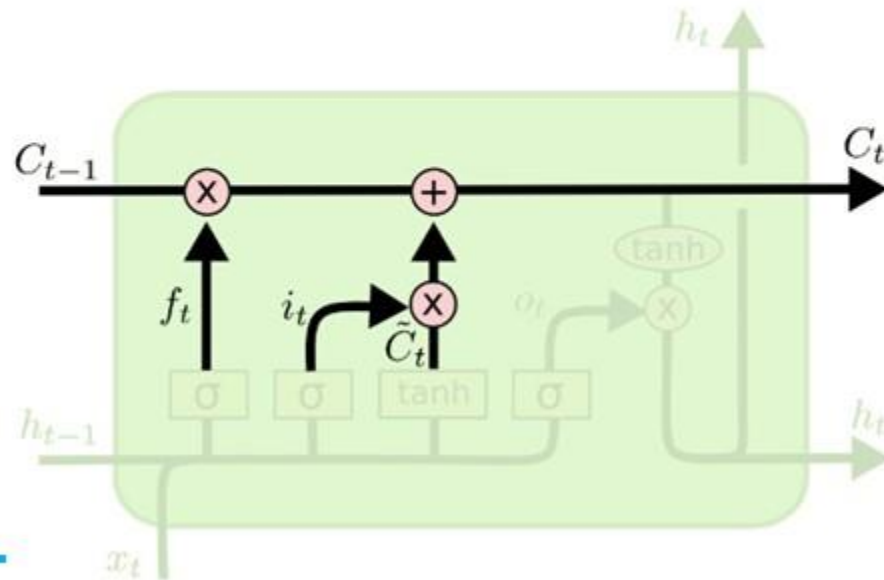
The Core Idea Behind LSTMs

It's now time to update the old cell state, C_{t-1} , into the new cell state C_t . The previous steps already decided what to do, we just need to actually do it.

We multiply the old state by f_t , forgetting the things we decided to forget 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.

The Core Idea Behind LSTMs

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.



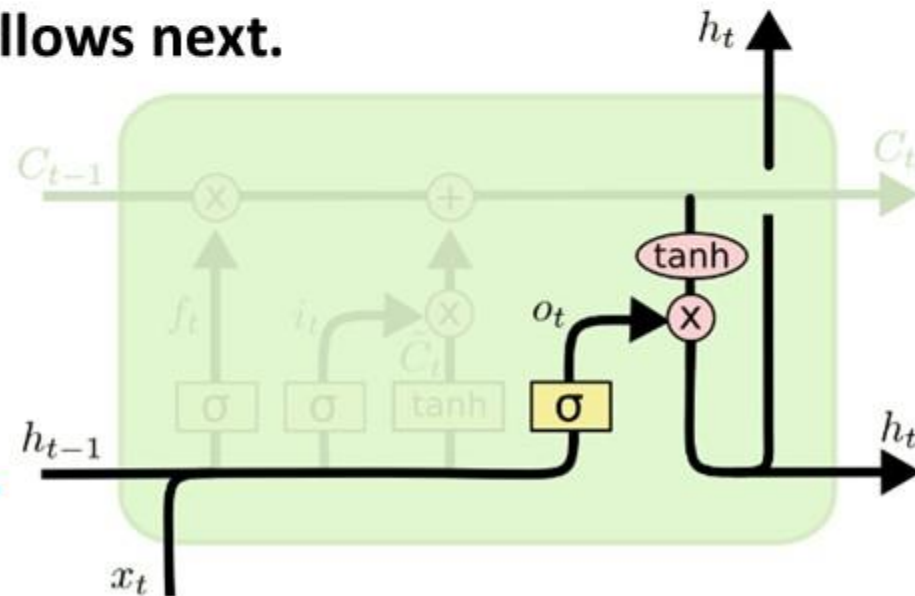
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

The Core Idea Behind LSTMs

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 tanh (to push the values to be between -1 and 1) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.

The Core Idea Behind LSTMs

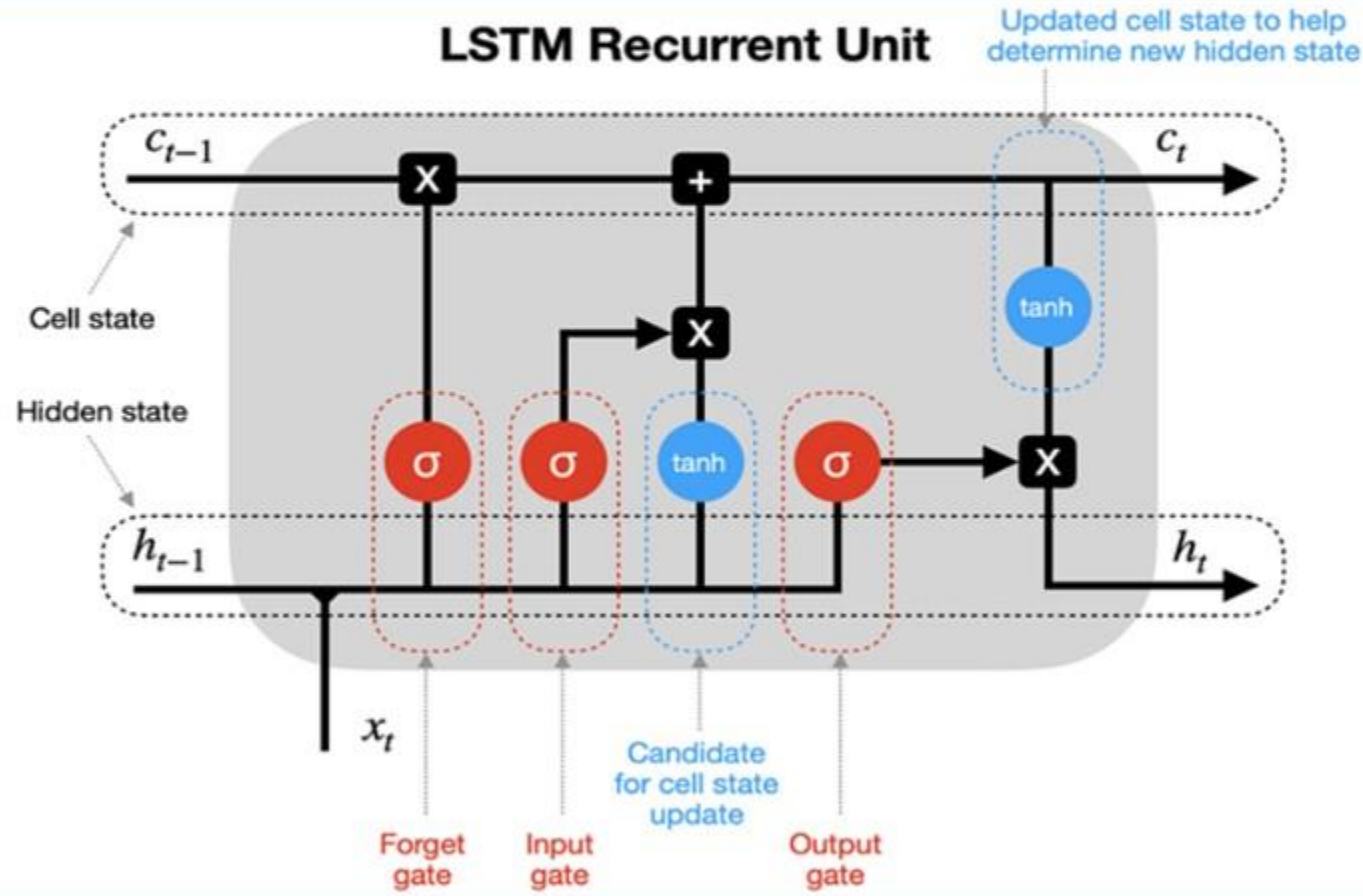
For the language model example, since it just saw a subject, it might want to output information relevant to a verb, in case that's what is coming next. For example, it might output whether the subject is singular or plural, so that we know what form a verb should be conjugated into if that's what follows next.



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

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

The Core Idea Behind LSTMs



To summarize**Input and Output**

The input to the LSTM model is a sequence of integer tokens, and the output is the probability of each word in the word's vocabulary appearing next in the sequence.

Embedding layer

The embedding layer is used to convert the integer tokens into a dense vector representation, which the LSTM can then process more effectively.

The LSTM Architecture

LSTM layer

The LSTM layer is a specialized type of the recurrent neural network that can learn long short dependencies in sequential data, making it well suited for language modeling tasks.

The LSTM Architecture (Embedding layer)

Word embeddings

The embedding layer converts each integer token into a dense vector representation, capturing the semantic and syntactic relationships between words. Which allowing for better understanding the context and meaning of the input sequence.

Learned representations

The embedding layer's weights are learned during the training process allowing the model to develop its own internal representations of the input words.

The LSTM Architecture (LSTM layer)

Memory Cell

The LSTM layer maintains a memory cell that can selectively remember and forget information, allowing it to capture long term dependences in the input sequence.

Gating Mechanism

The LSTM layer uses a gating mechanism to control the flow of information into and out of the memory cell, enabling it to learn patterns in the data.

The LSTM Architecture (LSTM layer)

Sequential processing:

The LSTM layer process the input sequence one token at a time, updating its internal state and producing an output at each step, which is then used to predict the next word in the sequence.

Forget Gate:

The forget gate determine which information from the previous time step should be retained in the memory cell, allowing the LSTM to selectively remember or forget relevant information.

Cell State:

The cell state is the long-term memory of the LSTM, which is updated at each time step based on the input, forget and output gates enabling the model to maintain and update its understanding of the sequence.

Input Gate:

The input gate controls which new information from the current input and previous hidden state should be added to the memory cell, enabling the LSTM to learn and update its internal state.

Output Gate:

The output gate decides which information from the memory cell and current input should be used to produce the output for the current time step, allowing the LSTM to generate relevant predictions.

Working of a LSTM (Output Gate)

The output gate uses a sigmoid activation function to determine the relevance of the information stored in the memory cell. It outputs a value between 0 and 1, where 0 means the information is not relevant for the output and 1 means it is entirely relevant. Then we pass this information through the tanh function to compress the value between 1 and -1.

Backpropagation

The LSTM model is trained using backpropagation through time, which allows the gradients of the loss function to be computed with respect to the model's parameters, enabling the model to learn from the training data.

Optimization

The model's parameters are updated using an optimization algorithm, such as stochastic gradient descent (SGD) or Adam, to minimize the loss function and improve the model's performance on the training data.

Validation

The model's performance is evaluated on a separate set during training, allowing the model to be tuned and preventing overfitting to the training data.

Metric used for evaluating the text generation like:

Perplexity: a measure of how well the model predicts the next word in a sequence, with lower values indicating better performance.

BLEU Score: a metric that compares the generated text to a reference corpus, evaluating the similarity of the generated text to the reference.

RNN vs LSTM

The key difference between RNN and LSTM is that it has additional signal information that is given from one cell to the other which is known as cell state. The cell state is capable of passing the information without getting diminished and this is controlled using gates.

RNN vs LSTM

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

RNN vs LSTM

Feature	LSTM (Long Short-term Memory)	RNN (Recurrent Neural Network)
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

RNN vs LSTM

Recurrent Neural Networks (RNNs) are designed to handle sequential data by maintaining a hidden state that captures information from previous time steps. However, they often face challenges in learning long-term dependencies, where information from distant time steps becomes crucial for making accurate predictions. This problem is known as the vanishing gradient or exploding gradient problem.

Vanishing Gradient

During backpropagation through time, gradients can become extremely small as they are multiplied through the chain of recurrent connections, causing the model to have difficulty learning dependencies that are separated by many time steps.

Exploding Gradient

Conversely, gradients can explode during backpropagation, leading to numerical instability and making it difficult for the model to converge.

Different Variants on Long Short-Term Memory

Vanilla LSTM

This is the original LSTM architecture proposed by Hochreiter and Schmidhuber. It includes memory cells with input, forget, and output gates to control the flow of information. The key idea is to allow the network to selectively update and forget information from the memory cell.

Peephole Connections

In the peephole LSTM, the gates are allowed to look at the cell state in addition to the hidden state. This allows the gates to consider the cell state when making decisions, providing more context information.

Different Variants on Long Short-Term Memory

Gated Recurrent Unit (GRU)

GRU is an alternative to LSTM, designed to be simpler and computationally more efficient. It combines the input and forget gates into a single “update” gate and merges the cell state and hidden state. While GRUs have fewer parameters than LSTMs, they have been shown to perform similarly in practice.

Applications of LSTM

- **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.

Applications of LSTM

- **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.

Applications of LSTM

- **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.



Thank You