Deep Learning

BCSE-332L

Module 5:

Recursive Neural Networks

Dr . Saurabh Agrawal

Faculty Id: 20165

School of Computer Science and Engineering

VIT, Vellore-632014

Tamil Nadu, India

Outline

- Recursive Neural Networks
- Long-Term Dependencies
- Long Short-Term Memory
- Other Gated RNNs
- Optimization for Long-Term Dependencies
- Echo State Networks
- Explicit Memory

Recursive Neural Networks

- Recursive Neural Networks (RvNNs) are deep neural networks used for natural language processing.
- We get a Recursive Neural Network when the same weights are applied recursively on a structured input to obtain a structured prediction.

Recursive Neural Networks

❑What Is a Recursive Neural Network?

❑Deep Learning is a subfield of machine learning and artificial intelligence (AI) that attempts to imitate how the human brain processes data and gains certain knowledge.

❑Neural Networks form the backbone of Deep Learning.

❑These are loosely modeled after the human brain and designed to accurately recognize underlying patterns in a data set.

❑If you want to predict the unpredictable, Deep Learning is the solution.

❑Due to their deep tree-like structure, Recursive Neural Networks can handle hierarchical data.

❑The tree structure means combining child nodes and producing parent nodes. Each child-parent bond has a weight matrix, and similar children have the same weights.

❑The number of children for every node in the tree is fixed to enable it to perform recursive operations and use the same weights.

❑RvNNs are used when there's a need to parse an entire sentence.

Recursive Neural Networks

❑Recurrent Neural Network Vs. Recursive Neural Networks

- Another well-known family of neural networks for processing sequential data is recurrent neural networks (RNNs). They are connected to the recursive neural network in a close way.
- Given that language-related data like sentences and paragraphs are sequential in nature, recurrent neural networks are useful for representing temporal sequences in natural language processing (NLP). Chain topologies are frequently used in recurrent networks. By distributing the weights along the entire chain length, the dimensionality is maintained.

Recursive Neural Networks

❑Recurrent Neural Network Vs. Recursive Neural Networks

- Recursive neural networks, on the other hand, work with hierarchical data models because of their tree structure. The tree may perform recursive operations and utilize the same weights at each step because each node has a set number of offspring. Parent representations are created by combining child representations.
- A feed-forward network is less efficient than a recursive network.
- Recursive networks are just a generalization of recurrent networks because recurrent networks are recurrent over time.

RvNNs for Natural Language Processing: Benefits

The structure and decrease in network depth of recursive neural networks are their two main advantages for natural language processing.

Recursive Neural Networks' tree structure, as previously mentioned, can manage hierarchical data, such as in parsing issues.

The ability for trees to have a logarithmic height is another advantage of RvNN. A recursive neural network could indeed represent a binary tree with a height of O(log n) when there are O(n) input words. The length between the first and the last input elements is shortened as a result. As a result, the long-term dependence becomes more manageable and shorter.

RvNNs for Natural Language Processing: Demerits

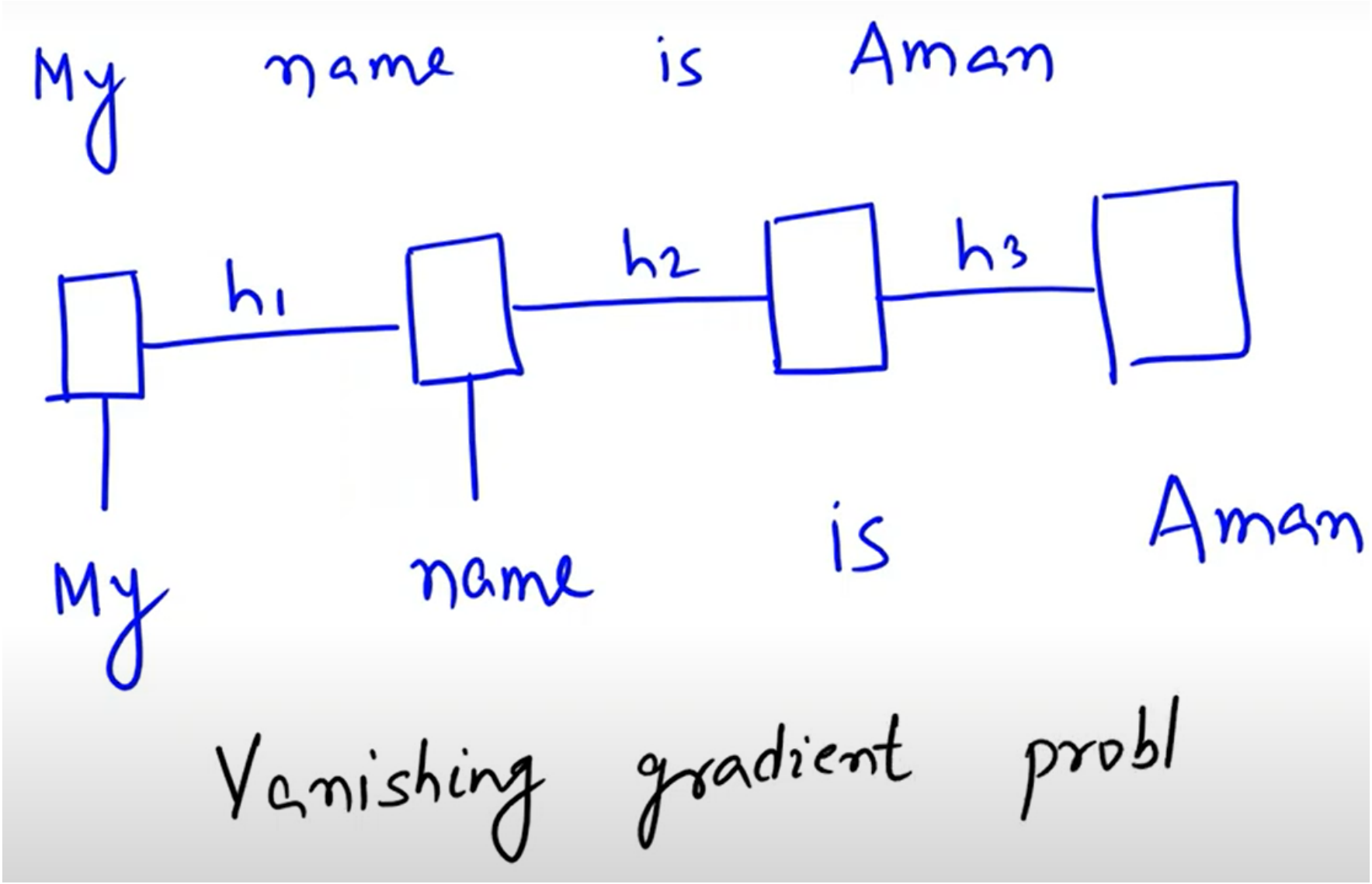
The tree structure of recursive neural networks may be their biggest drawback. Using the tree structure suggests giving our model a special inductive bias. The bias is consistent with the notion that the data are organized in a tree hierarchy. But the reality is different. As a result, the network might not be able to pick up on the current patterns.

The Recursive Neural Network also has a drawback in that sentence parsing can be cumbersome and slow. It's interesting that different parse trees can exist for the same text.

Additionally, labeling the training data for recursive neural networks takes more time and effort than building recurrent neural networks. It takes more time and effort to manually break down a sentence into smaller parts than it does to give it a label.

Long-Term Dependencies

❑Why Long-Term Dependencies?



Long-Term Dependencies

❑**What are long-term dependencies?**

❑**Long-term dependencies** are the situations where the output of an RNN depends on the **input** that occurred many time steps ago.

❑For instance, consider the sentence "**The cat, which was very hungry, ate the mouse**".

❑To understand the meaning of this sentence, you need to remember that the **cat is the subject** of the **verb ate**, even though they are **separated by a long clause**.

❑This is a long-term dependency, and it can affect the performance of an RNN that tries to generate or **analyze such sentences.**

Long-Term Dependencies

❑**Why are long-term dependencies??**

❑Recurrent neural networks (RNNs) are powerful machine learning models that can process **sequential data**, such as **text**, **speech**, or **video**.

❑However, they often struggle to capture long-term dependencies, which are the relationships between **distant elements** in the **sequence.**

❑**Why are long-term dependencies hard to learn?**

❑The main reason why long-term dependencies are hard to learn is that **RNNs** suffer from the **vanishing** or **exploding gradient** problem.

❑This means that the gradient, which is the signal that tells the network how to update its **weights**, becomes either very small or very large as it propagates through the network.

❑When the gradient vanishes, the network cannot learn from the distant inputs, and when it explodes, the network becomes **unstable** and produces **erratic outputs**.

❑This problem is caused by the **repeated multiplication of the same matrix**, which represents the **connections between** the **hidden units, at each time step.**

Long-Term Dependencies

❑**How can you use gated units to handle long-term dependencies?**

❑Another way to handle **long-term dependencies** is to use **gated units**, which are special types of hidden units that can **control the flow of information in the network.**

❑The most popular gated units are the long short-term memory (**LSTM**) and the **gated** recurrent unit (**GRU**).

❑These units have **internal mechanisms** that allow them to **remember** or **forget** the **previous inputs and outputs**, depending on the **current input** and **output**.

❑This way, they can selectively access the **relevant information** from the and **ignore the irrelevant information such as distant past.**

Long-Term Dependencies

❑**How can you use attention mechanisms to handle long-term dependencies?**

❑Another way to handle **long-term dependencies** is to use attention mechanisms, which are modules that can learn to focus on the most important parts of the input or output sequence.

❑The most common attention mechanism is the self-attention, which computes the similarity between each element in the sequence and assigns a weight to each one.

❑Then, it uses these **weights** to **create a context vector**, which summarizes the information from the whole sequence.

❑This way, it can **capture the relationships between the distant elements** and **enhance the representation of the sequence.**

Long-Term Dependencies

❑Challenges

- Neural network optimization face a difficulty when computational graphs become deep, e.g.,
  - Feedforward networks with many layers
  - RNNs that repeatedly apply the same operation at each time step of a long temporal sequence
- Gradients propagated over many stages tend to either vanish (most of the time) or explode (damaging optimization)
- The difficulty with long-term dependencies arise from exponentially smaller weights given to long-term interactions (involving multiplication of many Jacobians)

Long-Term Dependencies

❑LSTM: The Solution To Long Term Dependencies

❑Sometimes we just need to look at recent information to perform the present task.

❑For example, consider a language model trying to predict the last word in " **the clouds are in the sky**".

❑Here it's easy to predict the next word as sky based on the previous words. But consider the sentence **I grew up in France I speak fluent French.**

❑" Here it is not easy to predict that the language is French directly.

❑It depends on previous input also.

❑In such sentences it's entirely possible for the gap between the relevant information and the point where it is needed to become very large.

❑In theory, RNN's are absolutely capable of handling such "long-term dependencies."

Long-Term Dependencies

❑LSTM: The Solution To Long Term Dependencies

- ❑A human could carefully pick parameters for them to solve toy problems of this form.
- ❑Sadly, in practice, recurrent neural network don’t seem to be able to learn them.
- ❑This problem is called Vanishing gradient problem.
- ❑The neural network updates the weight using the gradient descent algorithm.
- ❑The gradients grow smaller when the network progress down to lower layers.
- ❑The gradients will stay constant meaning there is no space for improvement.
- ❑The model learns from a change in the gradient.
- ❑This change affects the network’s output.
- ❑However, if the difference in the gradients is very small network will not learn anything and so no difference in the output.
- ❑Therefore, a network facing a vanishing gradient problem cannot converge towards a good solution.

Long-Term Dependencies

❑Long Short Term Memory networks

❑Long Short Term Memory networks (LSTMs) is a special kind of recurrent neural network capable of learning long-term dependencies.

❑They were introduced by Hochreiter & Schmidhuber in 1997.

❑Remembering information for longer periods of time is their default behavior.

❑The Long short-term memory (LSTM) is made up of a memory cell, an input gate, an output gate and a forget gate.

❑The memory cell is responsible for remembering the previous state while the gates are responsible for controlling the amount of memory to be exposed.

Long-Term Dependencies

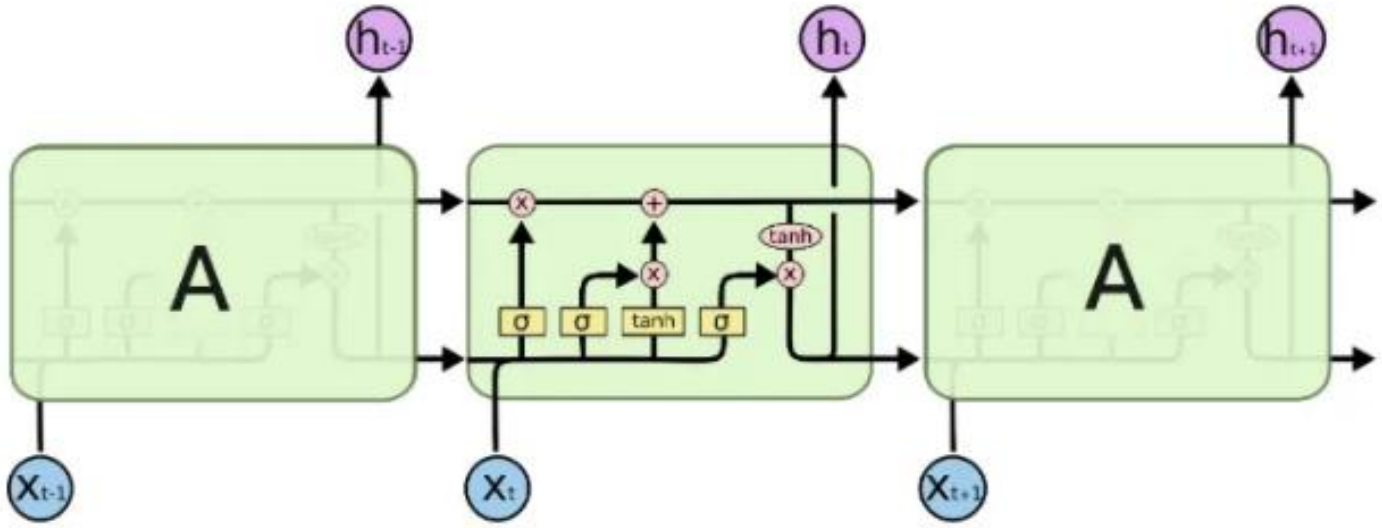
❑Long Short Term Memory networks

❑The memory cell is responsible for keeping track of the dependencies between the elements in the input sequence.

❑The present input and the previous is passed to forget gate and the output of this forget gate is fed to the previous cell state.

❑After that the output from the input gate is also fed to the previous cell state.

❑By using this the output gate operates and will generate the output.



Long-Term Dependencies

❑Long Short Term Memory networks

❑Forget Gate:

❑There are some information from the previous cell state that is not needed for the present unit in a LSTM.

❑A forget gate is responsible for removing this information from the cell state.

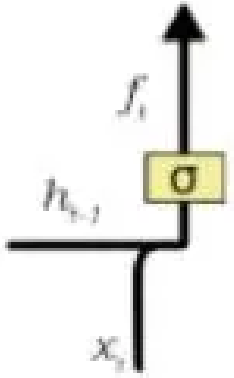
❑The information that is no longer required for the LSTM to understand or the information that is of less importance is removed via multiplication of a filter.

❑This is required for optimizing the performance of the LSTM network.

❑In other words we can say that it determines how much of previous state is to be passed to the next state.

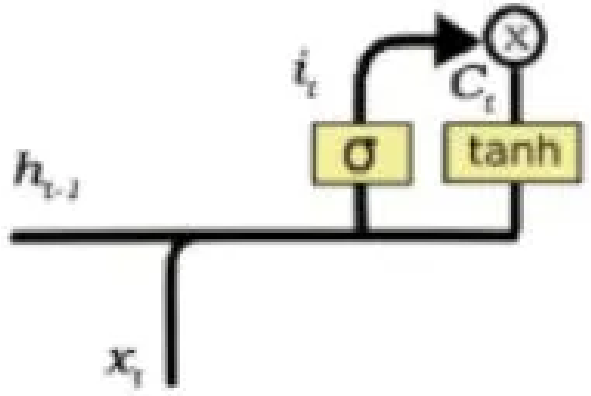
❑The gate has two inputs X t and h t-1. h t-1 is the output of the previous cell and x t is the input at that particular time step.

❑The given inputs are multiplied by the weight matrices and a bias is added. Following this, the sigmoid function(activation function) is applied to this value.



Long-Term Dependencies

- **Long Short Term Memory networks**
- **Input Gate:**
- The process of adding new information takes place in input gate.
- Here combination of x t and h t-1 is passed through sigmoid and tanh functions(activation functions) and added.
- Creating a vector containing all the possible values that can be added (as perceived from h t-1 and x t) to the cell state.
- This is done using the tanh function. By this step we ensure that only that information is added to the cell state that is important and is not redundant.



Long-Term Dependencies

❑Long Short Term Memory networks

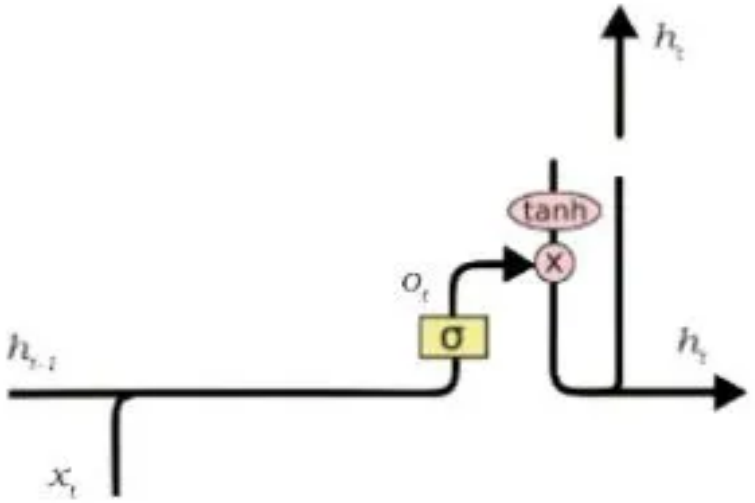
❑Output Gate:

❑A vector is created after applying tanh function to the cell state.

❑Then making a filter using the values of h t-1 and x t, such that it can regulate the values that need to be output from the vector created above.

❑This filter again employs a sigmoid function.

❑Then both of them are multiplied to form output of that cell state.



Long Short-Term Memory

- LSTM excels in sequence prediction tasks, capturing long-term dependencies.
- Ideal for time series, machine translation, and speech recognition due to order dependence.
- **What is LSTM?**
- Long Short-Term Memory is an improved version of recurrent neural network designed by Hochreiter & Schmidhuber.
- A traditional RNN has a single hidden state that is passed through time, which can make it difficult for the network to learn long-term dependencies.
- LSTMs model address this problem by introducing a memory cell, which is a container that can hold information for an extended period.
- LSTM architectures are capable of learning long-term dependencies in sequential data, which makes them well-suited for tasks such as language translation, speech recognition, and time series forecasting.

Long Short-Term Memory

❑LSTM Architecture:

❑The LSTM architectures involves the memory cell which is controlled by three gates: the input gate, the forget gate, and the output gate.

❑These gates decide what information to add to, remove from, and output from the memory cell.

1. The input gate controls what information is added to the memory cell.
2. The forget gate controls what information is removed from the memory cell.
3. The output gate controls what information is output from the memory cell.

❑This allows LSTM networks to selectively retain or discard information as it flows through the network, which allows them to learn long-term dependencies.

❑The LSTM maintains a hidden state, which acts as the short-term memory of the network.

❑The hidden state is updated based on the input, the previous hidden state, and the memory cell's current state.

Long Short-Term Memory

❑Bidirectional LSTM Model

❑Bidirectional LSTM (Bi LSTM/ BLSTM) is recurrent neural network (RNN) that is able to process sequential data in both forward and backward directions.

❑This allows Bi LSTM to learn longer-range dependencies in sequential data than traditional LSTMs, which can only process sequential data in one direction.

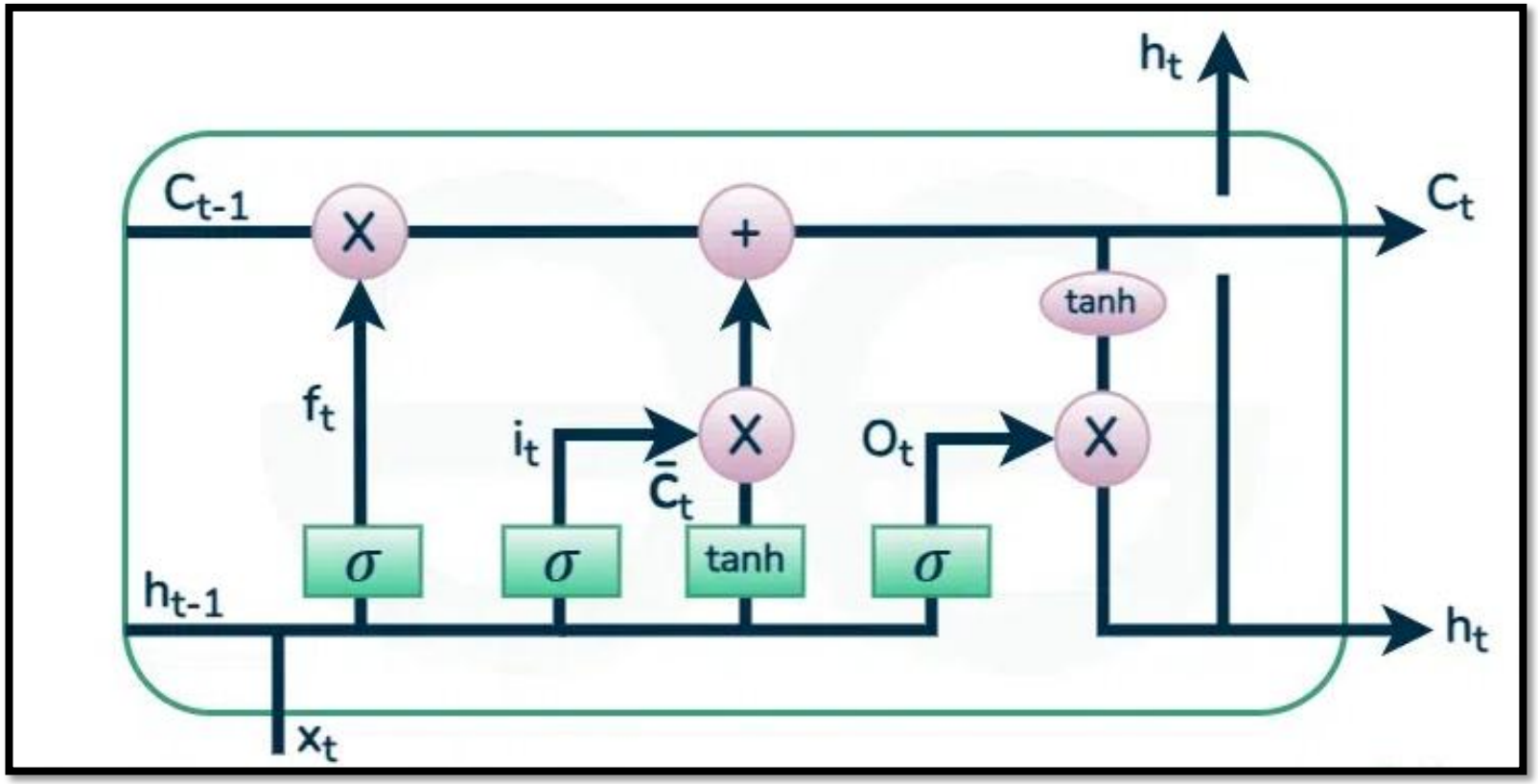
❑Bi LSTMs are made up of two LSTM networks, one that processes the input sequence in the forward direction and one that processes the input sequence in the backward direction.

1. The outputs of the two LSTM networks are then combined to produce the final output.
2. Networks in LSTM architectures can be stacked to create deep architectures, enabling the learning of even more complex patterns and hierarchies in sequential data.

❑Each LSTM layer in a stacked configuration captures different levels of abstraction and temporal dependencies within the input data.

Long Short-Term Memory

❑**LSTM Working:** LSTM architecture has a chain structure that contains four neural networks and different memory blocks called **cells**.



❑Information is retained by the cells and the memory manipulations are done by the **three gates:**

Forget Gate, Input gate, Output gate

Long Short-Term Memory

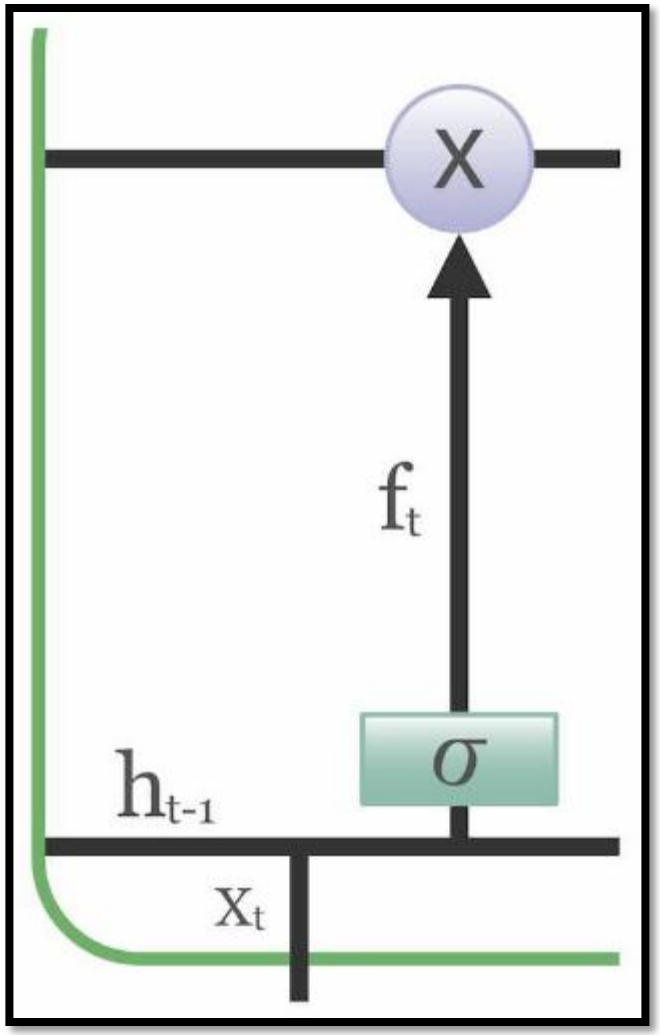
❑LSTM Working: Forget Gate

❑The information that is no longer useful in the cell state is removed with the forget gate.

❑Two inputs x_t (input at the particular time) and h_{t-1} (previous cell output) are fed to the gate and multiplied with weight matrices followed by the addition of bias.

❑The resultant is passed through an activation function which gives a binary output.

❑If for a particular cell state the output is 0, the piece of information is forgotten and for output 1, the information is retained for future use.



❑LSTM Working: Forget Gate

❑The equation for the forget gate is:

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

❑where:

❑W_f represents the weight matrix associated with the forget gate.

❑[h_t-1, x_t] denotes the concatenation of the current input and the previous hidden state.

❑b_f is the bias with the forget gate.

❑σ is the sigmoid activation function.

Long Short-Term Memory

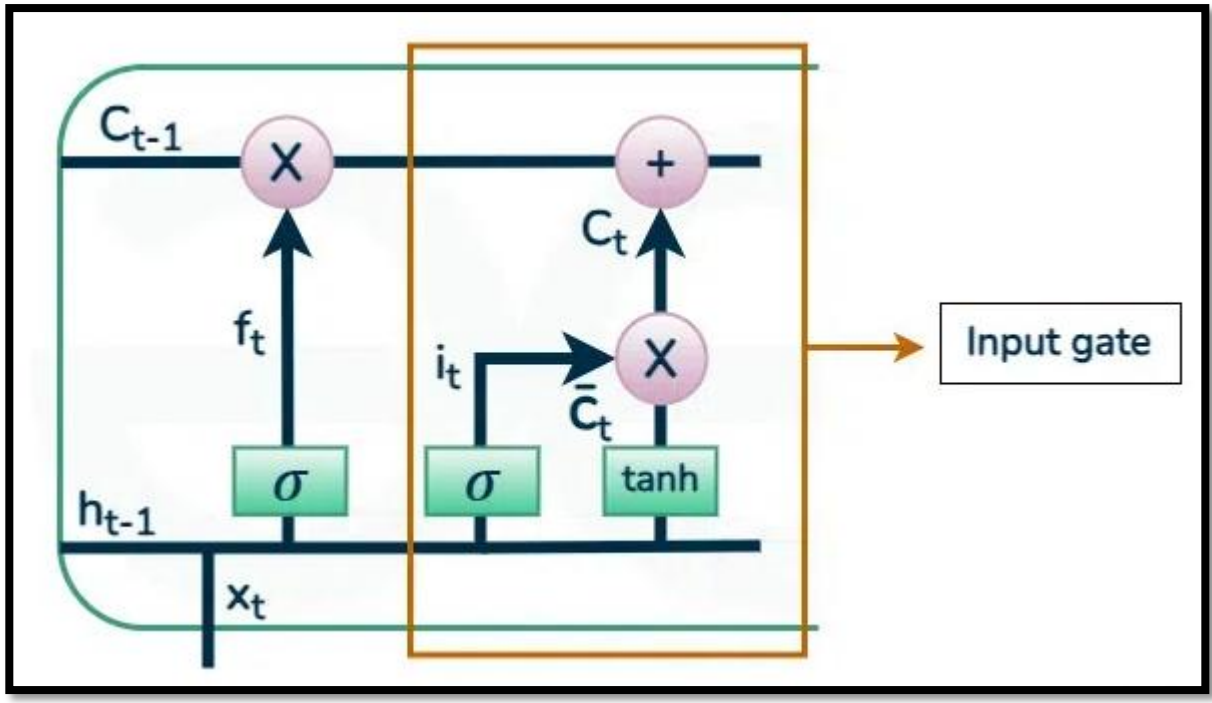
❑LSTM Working: Input gate

❑The addition of useful information to the cell state is done by the input gate.

❑First, the information is regulated using the sigmoid function and filter the values to be remembered similar to the forget gate using inputs h_{t-1} and $x_{t.}$.

❑Then, a vector is created using *tanh* function that gives an output from -1 to +1, which contains all the possible values from h_{t-1} and x_t.

❑At last, the values of the vector and the regulated values are multiplied to obtain the useful information.



❑LSTM Working: Input gate

❑The equation for the input gate is:

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

$$\hat{C}_t = tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$$

We multiply the previous state by f_t, disregarding the information we had previously chosen to ignore. Next, we include $i_t * C_t$. This represents the updated candidate values, adjusted for the amount that we chose to update each state value.

$$C_t = f_t \odot C_{t-1} + i_t \odot \hat{C}_t$$

where

- $\odot$ denotes element-wise multiplication
- tanh is tanh activation function

Long Short-Term Memory

❑LSTM Working: Output gate

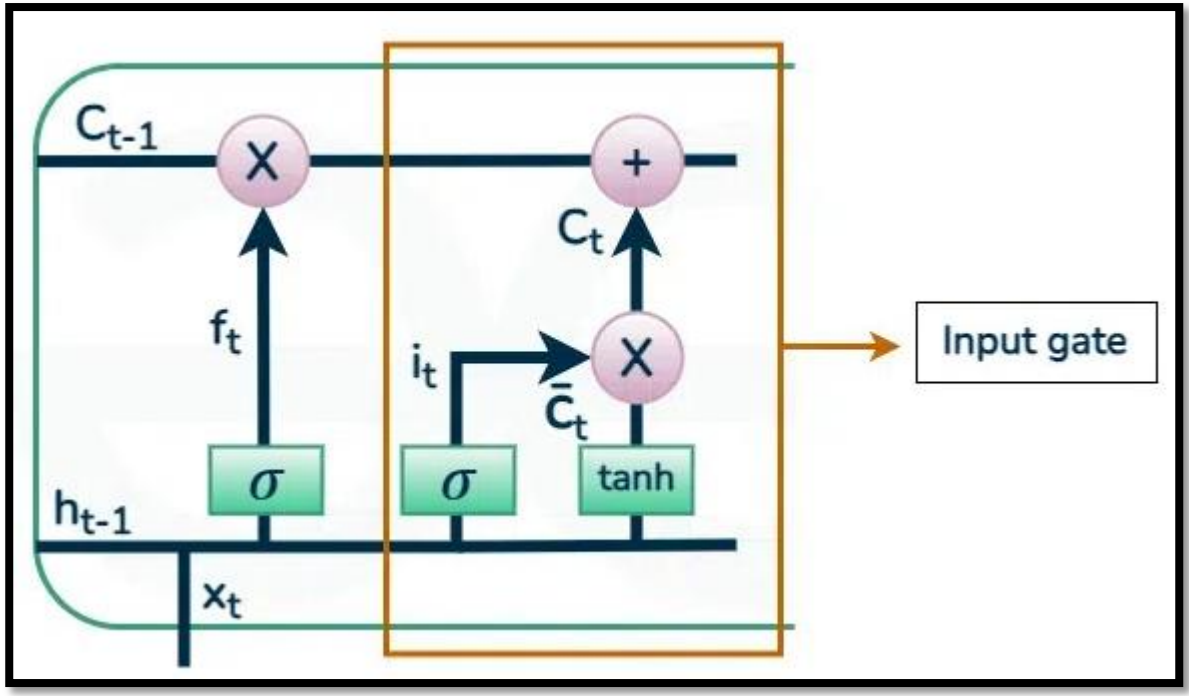
❑The task of extracting useful information from the current cell state to be presented as output is done by the output gate.

❑First, a vector is generated by applying tanh function on the cell.

❑Then, the information is regulated using the sigmoid function and filter by the values to be remembered using inputs h_{t-1} and x_t .

❑At last, the values of the vector and the regulated values are multiplied to be sent as an output and input to the next cell.

❑The equation for the output gate is:



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

Long Short-Term Memory

❑LSTM Applications

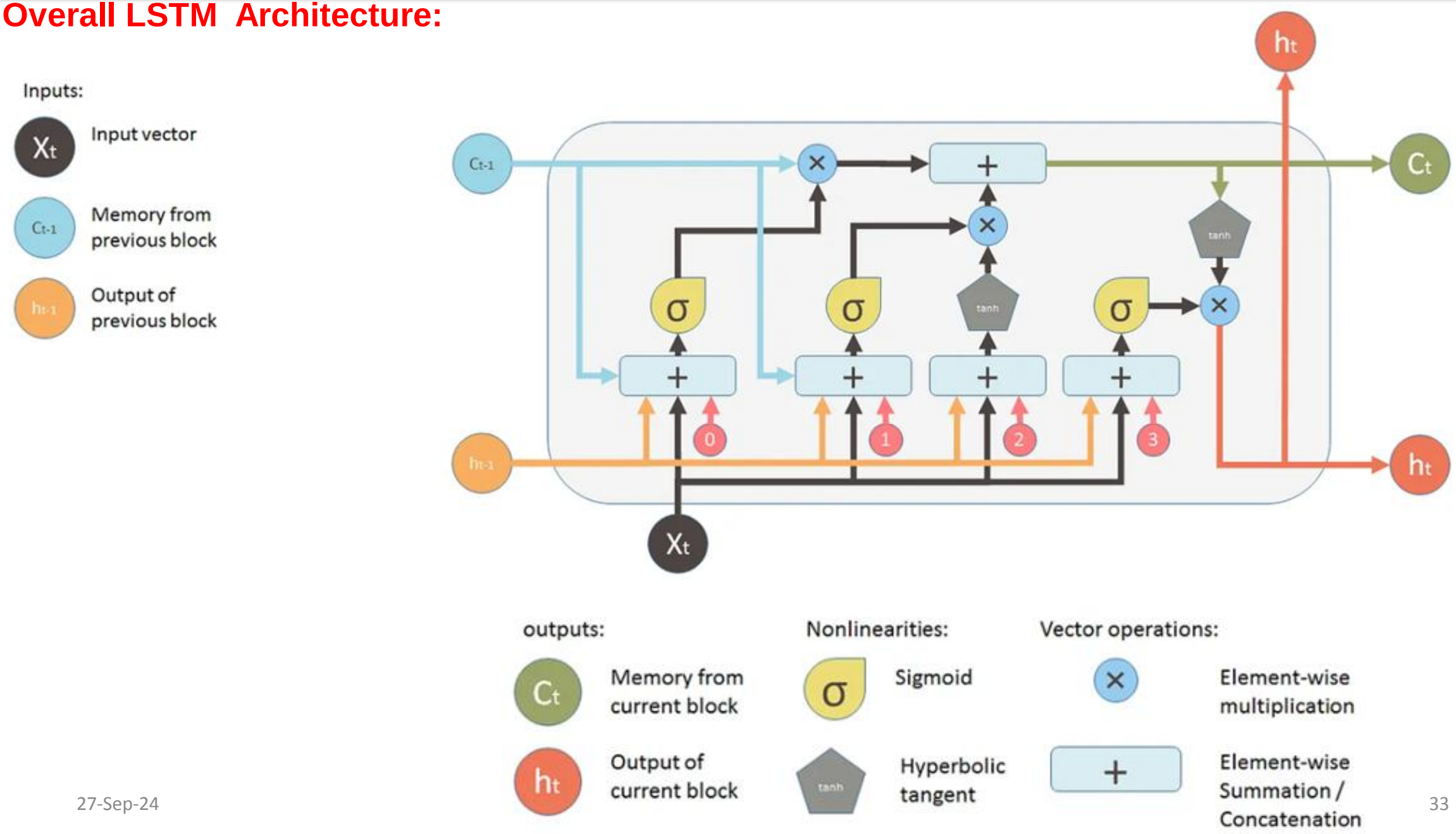
1. **Language Modeling:** LSTMs have been used for natural language processing tasks such as language modeling, machine translation, and text summarization. They can be trained to generate coherent and grammatically correct sentences by learning the dependencies between words in a sentence.
2. **Speech 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.
3. **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.

❑LSTM Applications

4. **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.
5. **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.
6. **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.

Long Short-Term Memory

❑ Overall LSTM Architecture:



Other Gated RNNs Gated Recurrent Unit

❑What is Gated Recurrent Unit(GRU)?

❑GRU stands for Gated Recurrent Unit, which is a type of recurrent neural network (RNN) architecture that is similar to LSTM (Long Short-Term Memory).

❑Like LSTM, GRU is designed to model sequential data by allowing information to be selectively remembered or forgotten over time.

❑However, GRU has a simpler architecture than LSTM, with fewer parameters, which can make it easier to train and more computationally efficient.

❑The main difference between GRU and LSTM is the way they handle the memory cell state.

❑In LSTM, the memory cell state is maintained separately from the hidden state and is updated using three gates: the input gate, output gate, and forget gate.

❑In GRU, the memory cell state is replaced with a “candidate activation vector,” which is updated using two gates: the reset gate and update gate.

Other Gated RNNs Gated Recurrent Unit

❑What is Gated Recurrent Unit(GRU)?

❑The reset gate determines how much of the previous hidden state to forget, while the update gate determines how much of the candidate activation vector to incorporate into the new hidden state.

❑Overall, GRU is a popular alternative to LSTM for modeling sequential data, especially in cases where computational resources are limited or where a simpler architecture is desired.

Other Gated RNNs Gated Recurrent Unit

❑How GRU Works?

❑Like other recurrent neural network architectures, GRU processes sequential data one element at a time, updating its hidden state based on the current input and the previous hidden state.

❑At each time step, the GRU computes a “candidate activation vector” that combines information from the input and the previous hidden state.

❑This candidate vector is then used to update the hidden state for the next time step.

❑The candidate activation vector is computed using two gates: the reset gate and the update gate.

❑The reset gate determines how much of the previous hidden state to forget, while the update gate determines how much of the candidate activation vector to incorporate into the new hidden state.

Other Gated RNNs Gated Recurrent Unit

❑How GRU Works?

❑Here's the math behind the GRU architecture:

1. The reset gate r and update gate z are computed using the current input x and the previous hidden state h_t-1

r_t = sigmoid(W_r * [h_t-1, x_t])

z_t = sigmoid(W_z * [h_t-1, x_t])

where W_r and W_z are weight matrices that are learned during training.

2. The candidate activation vector h_t~ is computed using the current input x and a modified version of the previous hidden state that is "reset" by the reset gate:

h_t~ = tanh(W_h * [r_t * h_t-1, x_t])

where W_h is another weight matrix.

Other Gated RNNs Gated Recurrent Unit

❑How GRU Works?

❑The new hidden state h_t is computed by combining the candidate activation vector with the previous hidden state, weighted by the update gate:

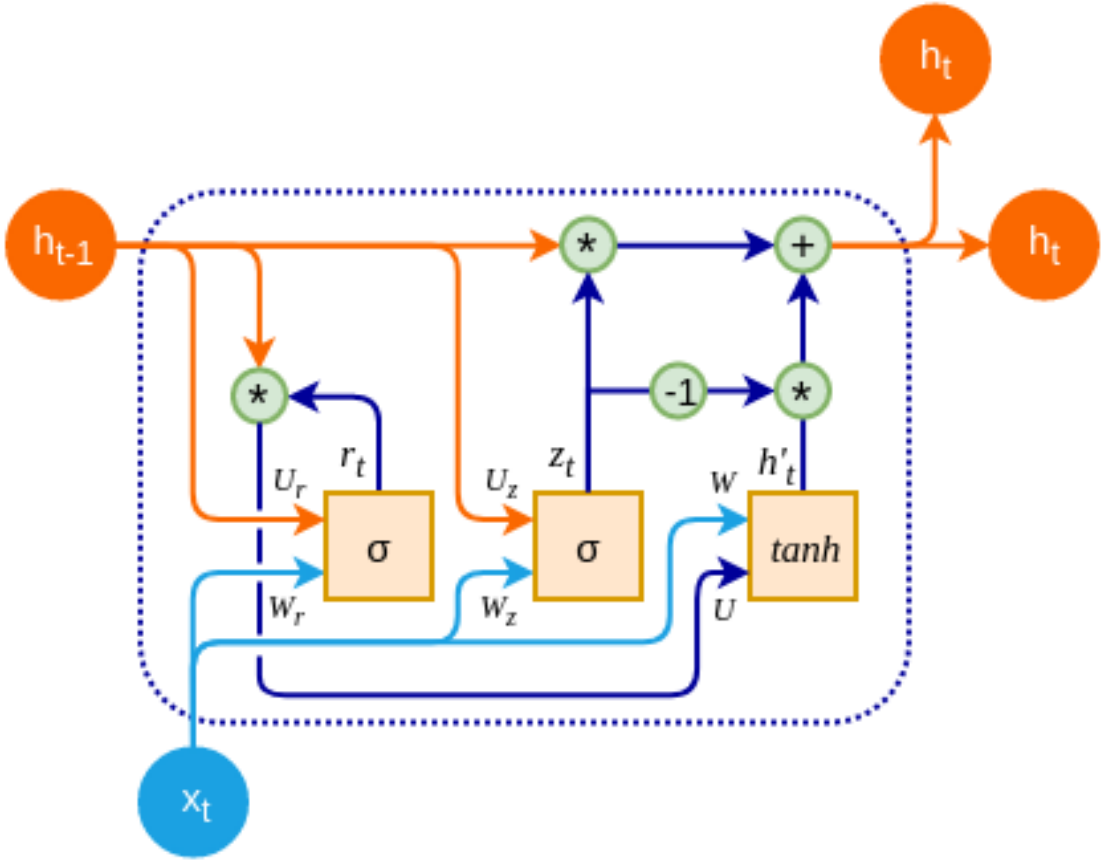
$$h_t = (1 - z_t) * h_{t-1} + z_t * h_t\sim$$

❑Overall, the reset gate determines how much of the previous hidden state to remember or forget, while the update gate determines how much of the candidate activation vector to incorporate into the new hidden state.

❑The result is a compact architecture that is able to selectively update its hidden state based on the input and previous hidden state, without the need for a separate memory cell state like in LSTM.

Other Gated RNNs Gated Recurrent Unit

❑GRU Architecture



Other Gated RNNs Gated Recurrent Unit

❑GRU Architecture

1. **Input layer:** The input layer takes in sequential data, such as a sequence of words or a time series of values, and feeds it into the GRU.
2. **Hidden layer:** The hidden layer is where the recurrent computation occurs. At each time step, the hidden state is updated based on the current input and the previous hidden state. The hidden state is a vector of numbers that represents the network's "memory" of the previous inputs.
3. **Reset gate:** The reset gate determines how much of the previous hidden state to forget. It takes as input the previous hidden state and the current input, and produces a vector of numbers between 0 and 1 that controls the degree to which the previous hidden state is "reset" at the current time step.

Other Gated RNNs Gated Recurrent Unit

❑GRU Architecture

4. **Update gate:** The update gate determines how much of the candidate activation vector to incorporate into the new hidden state. It takes as input the previous hidden state and the current input, and produces a vector of numbers between 0 and 1 that controls the degree to which the candidate activation vector is incorporated into the new hidden state.
5. **Candidate activation vector:** The candidate activation vector is a modified version of the previous hidden state that is “reset” by the reset gate and combined with the current input. It is computed using a tanh activation function that squashes its output between -1 and
6. **Output layer:** The output layer takes the final hidden state as input and produces the network’s output. This could be a single number, a sequence of numbers, or a probability distribution over classes, depending on the task at hand.

Optimization for Long-Term Dependencies

❑Optimizing for long-term dependencies in models, especially in the context of sequence data (like time series or natural language), can be challenging.

❑Here are some strategies that are commonly used to enhance the ability of models to capture these dependencies:

1.Recurrent Neural Networks (RNNs)

▪**LSTM and GRU**: Use Long Short-Term Memory (LSTM) networks or Gated Recurrent Units (GRUs), which are designed to remember information for long periods and mitigate issues like vanishing gradients.

2. Attention Mechanisms

▪**Self-Attention**: Incorporate self-attention mechanisms, as seen in Transformer models. These allow the model to weigh the importance of different parts of the input sequence, effectively capturing long-range dependencies.

▪**Multi-Head Attention**: Using multiple attention heads can help the model focus on different parts of the sequence simultaneously.

Optimization for Long-Term Dependencies

3. Positional Encoding

▪In Transformer models, use positional encoding to maintain information about the position of elements in the sequence, which helps in understanding the order and context over long ranges.

4. Hierarchical Models

▪Implement hierarchical structures that process data at different levels of granularity. For example, you can first capture local dependencies and then model global dependencies.

5. Dilated Convolutions

▪Use dilated convolutions in convolutional neural networks (CNNs) to capture wider contexts without increasing the number of parameters significantly.

6. Memory-Augmented Networks

▪Consider models that utilize external memory (like Neural Turing Machines or Differentiable Neural Computers) to store and retrieve information over long periods.

Optimization for Long-Term Dependencies

7. Regularization Techniques

Employ regularization methods to prevent overfitting, which can make the model less sensitive to long-term dependencies.

8. Feature Engineering

Design features that explicitly capture long-term trends, such as moving averages or seasonal indicators in time series data.

9. Data Augmentation

Use techniques that artificially expand the training dataset to expose the model to more varied long-term dependencies.

10. Gradient Clipping

Implement gradient clipping to manage exploding gradients, allowing the model to learn from longer sequences without losing important information.

Optimization for Long-Term Dependencies

11. Batch Normalization and Layer Normalization

Normalize activations to stabilize learning and allow the model to better capture dependencies.

12. Training Techniques

Use techniques like curriculum learning, where the model is first trained on simpler sequences before gradually increasing complexity.

13. Advanced Architectures

Explore architectures like Transformers with memory networks or other novel designs that inherently address long-range dependencies.

Echo State Networks

- Echo State Networks (ESNs) are a specific kind of recurrent neural network (RNN) designed to efficiently handle sequential data.
- In Python, ESNs are created as a reservoir computing framework, which includes a fixed, randomly initialized recurrent layer known as the "reservoir."
- The key feature of ESNs is their ability to make the most of the echo-like dynamics of the reservoir, allowing them to effectively capture and replicate temporal patterns in sequential input.
- The way ESNs work is by linearly combining the input and reservoir states to generate the network's output.
- The reservoir weights remain fixed, this unique approach makes ESNs particularly useful in tasks where capturing temporal dependencies is critical, such as time-series prediction and signal processing.
- Implementing ESNs in Python, researchers and practitioners often turn to libraries like PyTorch or specialized reservoir computing frameworks.

Echo State Networks

❑WHAT IS ECHO STATE NETWORKS?

❑Echo State Networks (ESNs) in Python are a fascinating type of recurrent neural network (RNN) tailored for handling sequential data.

❑Imagine it as a three-part orchestra: there's the input layer, a reservoir filled with randomly initialized interconnected neurons, and the output layer.

❑The magic lies in the reservoir, where the weights are like a musical improvisation – fixed and randomly assigned.

❑This creates an "echo" effect, capturing the dynamics of the input signal.

❑During training, we tweak only the output layer, guiding it to map the reservoir's states to the desired output.

❑An Echo State Network (ESN) in Python is like a smart system that can predict what comes next in a sequence of data.

❑Imagine you have a list of numbers or values, like the temperature each day.

❑An ESN can learn from this data and then try to guess the temperature for the next day.

Echo State Networks

❑WHAT IS ECHO STATE NETWORKS?

1. **Reservoir:** It has a special part called a “reservoir,” which is like a pool of interconnected neurons. These neurons work together to remember patterns in the data. It’s like having a memory of what happened before.
2. **Training:** We show the ESN some of our data and let it learn. It doesn’t learn everything but gets the hang of the patterns. It’s like giving a few examples to a friend so they can understand how to predict.

Echo State Networks

❑WHAT IS ECHO STATE NETWORKS?

3. **Predicting:** Now, when we give the ESN a new piece of data (like the past temperatures), it uses what it learned to make a guess about what comes next.
4. **Output:** The ESN gives us its prediction, and we can compare it to the real answer. If it's good, great! If not, we might need to tweak things a bit.

Echo State Networks

❑Applications of Echo-State Networks

1. **Time Series Prediction:** ESNs excel at predicting future values in a time series. They are particularly effective when dealing with sequences of data where there are complex patterns and dependencies.
2. **Efficient Training:** ESNs have a unique training approach. While the reservoir is randomly generated and fixed, only the output weights are trained. This makes training computationally efficient compared to traditional recurrent neural networks (RNNs) and allows ESNs to be trained on smaller datasets.

Echo State Networks

❑Applications of Echo-State Networks

3. **Nonlinear Mapping**: The reservoir in an ESN introduces nonlinearity to the model. This helps capture and model complex relationships in the data that linear models might struggle with.
4. **Robustness to Noise:** ESNs are known for being robust to noise in the input data. The reservoir's dynamic nature allows it to filter out irrelevant information and focus on the essential patterns.

Echo State Networks

❑Applications of Echo-State Networks

5. **Universal Approximator:** Theoretically, ESNs are capable of approximating any dynamical system. This means they can be applied to a wide range of tasks, from simple pattern recognition to more complex tasks like chaotic time series prediction.
6. **Ease of Implementation:** Implementing ESNs is often simpler compared to training traditional RNNs. The fixed random reservoir and the straightforward training of output weights make ESNs more accessible for practical use.

Echo State Networks

❑Concepts of Echo-State Networks

- **Reservoir Computing:** Reservoir Computing is a framework that includes Echo State Networks. In ESNs, the reservoir is a dynamic memory of randomly initialized recurrent neurons that captures temporal patterns in sequential data.
- **Reservoir:** The reservoir is a fixed and randomly initialized collection of recurrent neurons in an ESN. It serves as a dynamic memory capturing temporal dependencies in the input data.

Echo State Networks

❑Concepts of Echo-State Networks

- **Echo State Property:** The Echo State Property is a key characteristic of the reservoir, indicating that its dynamics amplify and retain information from the input over time. This property ensures effective learning and memory of temporal patterns.
- **Input Layer:** The input layer of an ESN receives sequential input data. Each input corresponds to a time step in the sequence.

Echo State Networks

❑Concepts of Echo-State Networks

- **Input Layer:** The input layer of an ESN receives sequential input data. Each input corresponds to a time step in the sequence.
- **Output Layer:** The output layer produces the final result based on the combination of the reservoir states. It is typically a linear combination of the reservoir states, and only the output layer is trained.

Echo State Networks

❑Concepts of Echo-State Networks

- **Training:** During training, the ESN learns to map the reservoir states to the desired output. The training focuses on adjusting the weights in the output layer while keeping the reservoir weights fixed.
- **Fixed Weights:** The weights in the reservoir are randomly initialized and remain fixed during training. This simplifies the training process and enhances the network's ability to capture temporal patterns.

Note for Students

- This power point presentation is for lecture, therefore it is suggested that also utilize the text books and lecture notes.