Long Short-Term Memory

Joint Proseminar on Machine Learning and Image Processing

Fynn Jansen, Leon Benz

July 27, 2025

Chair of Computer Science 13 (Computer Vision)
RWTH Aachen

Contents

Introduction - The importance of Sequence Data

Real world data often best described by sequences:

Introduction - The importance of Sequence Data

Real world data often best described by sequences:

- Language

Introduction - The importance of Sequence Data

Real world data often best described by sequences:

- Language
- Music & Audio

Introduction - The importance of Sequence Data

Real world data often best described by sequences:

- Language
- Music & Audio
- (Stock) prices

Introduction - The importance of Sequence Data

Real world data often best described by sequences:

- Language
- Music & Audio
- (Stock) prices
- Weather

Introduction - The importance of Sequence Data

Real world data often best described by sequences:

- Language
- Music & Audio
- (Stock) prices
- Weather

Recurrent Neural Networks: Pick up on patterns in sequences

Introduction - The importance of Sequence Data

Real world data often best described by sequences:

- Language
- Music & Audio
- (Stock) prices
- Weather

Recurrent Neural Networks: Pick up on patterns in sequences

In practice: Difficult to train

Contents

Background – Neural Networks

- Artificial neurons: simplified models of biological neurons

Background – Neural Networks

- Artificial neurons: simplified models of biological neurons
- A **perceptron** computes a weighted sum of inputs:

$$y = f\left(\sum_{i=0}^{n} w_i x_i + b\right)$$

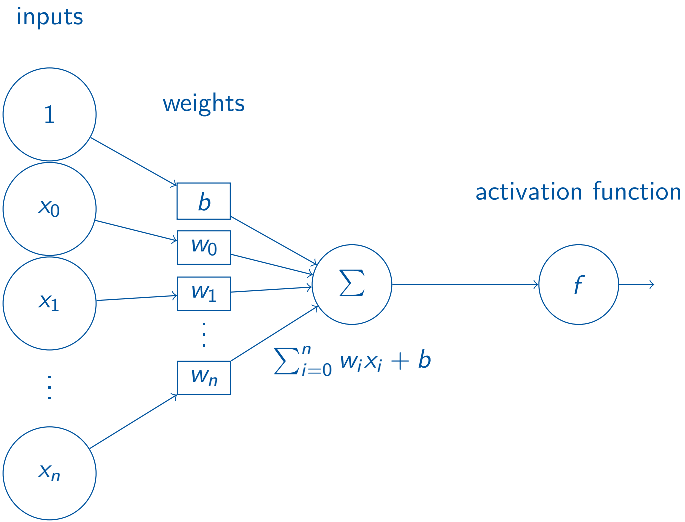
Background – Neural Networks

- Artificial neurons: simplified models of biological neurons
- A **perceptron** computes a weighted sum of inputs:

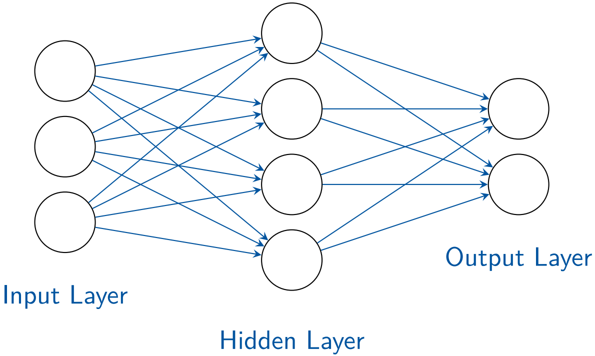
$$y = f\left(\sum_{i=0}^{n} w_i x_i + b\right)$$

- Non-linearity introduced via activation function f

Background - Neural Networks



Perceptrons are combined into multi-layer neural networks (MLPs)



- Neural networks learn by minimizing a **loss function**

Background – Training Neural Networks

- Neural networks learn by minimizing a **loss function**
- Use **stochastic gradient descent** (SGD):

$$\theta \leftarrow \theta - \eta \nabla_\theta L(\theta)$$

Background – Training Neural Networks

- Neural networks learn by minimizing a **loss function**
- Use **stochastic gradient descent** (SGD):

$$\theta \leftarrow \theta - \eta \nabla_\theta L(\theta)$$

- Gradients are computed using **backpropagation**

Background – Training Neural Networks

- Neural networks learn by minimizing a **loss function**
- Use **stochastic gradient descent** (SGD):

$$\theta \leftarrow \theta - \eta \nabla_\theta L(\theta)$$

- Gradients are computed using **backpropagation**
- Works well for fixed-size input/output — not ideal for sequences

Source: LeCun et al. 1998

Background – Recurrent Neural Networks

- Designed to process sequences of arbitrary length

Background – Recurrent Neural Networks

- Designed to process sequences of arbitrary length
- Maintain a **hidden state** h_t to capture temporal context

Background – Recurrent Neural Networks

- Designed to process sequences of arbitrary length
- Maintain a **hidden state** h_t to capture temporal context
- Reuse weights across time steps

Background – Recurrent Neural Networks

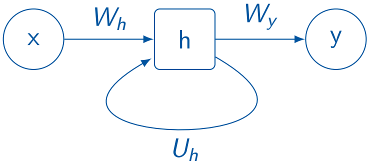
- Designed to process sequences of arbitrary length
- Maintain a **hidden state** h_t to capture temporal context
- Reuse weights across time steps
- Core equations:

$$h_t = \varphi(W_h x_t + U_h h_{t-1} + b_h) \qquad y_t = \psi(W_y h_t + b_y)$$

Background – Recurrent Neural Networks

- Designed to process sequences of arbitrary length
- Maintain a **hidden state** h_t to capture temporal context
- Reuse weights across time steps
- Core equations:

$$h_t = \varphi(W_h x_t + U_h h_{t-1} + b_h) \qquad y_t = \psi(W_y h_t + b_y)$$



Source: Elman 1990

Background – Unrolling an RNN

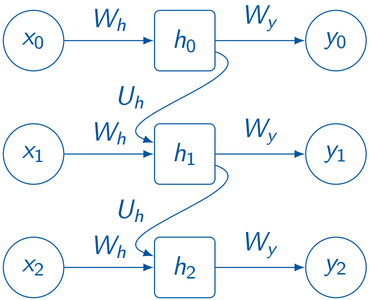
- RNNs can be visualized as a sequence of repeated cells

Background – Unrolling an RNN

- RNNs can be visualized as a sequence of repeated cells
- Each time step processes one input and updates the hidden state

Background – Unrolling an RNN

- RNNs can be visualized as a sequence of repeated cells
- Each time step processes one input and updates the hidden state
- Hidden state carries memory across time



Source: Elman 1990

Background – Backpropagation Through Time

- To train RNNs, we apply **Backpropagation Through Time (BPTT)**

Background – Backpropagation Through Time

- To train RNNs, we apply **Backpropagation Through Time (BPTT)**
- Compute gradients of loss at each time step t w.r.t. previous states

Background – Backpropagation Through Time

- To train RNNs, we apply **Backpropagation Through Time (BPTT)**
- Compute gradients of loss at each time step t w.r.t. previous states
- The gradient of the loss at time T w.r.t. a hidden state h_t:

$$\frac{\partial \mathcal{L}_T}{\partial h_t} = \sum_{k=t}^{T} \left(\frac{\partial \mathcal{L}_T}{\partial h_k} \cdot \prod_{j=t+1}^{k} \frac{\partial h_j}{\partial h_{j-1}} \right)$$

- To train RNNs, we apply **Backpropagation Through Time (BPTT)**
- Compute gradients of loss at each time step t w.r.t. previous states
- The gradient of the loss at time T w.r.t. a hidden state h_t:

$$\frac{\partial \mathcal{L}_T}{\partial h_t} = \sum_{k=t}^{T} \left(\frac{\partial \mathcal{L}_T}{\partial h_k} \cdot \prod_{j=t+1}^{k} \frac{\partial h_j}{\partial h_{j-1}} \right)$$

- Each $\frac{\partial h_j}{\partial h_{j-1}}$ is a Jacobian matrix

Source: Werbos 1990; Pascanu et al. 2013

Background – Vanishing and Exploding Gradients

- When multiplying many Jacobians, gradients can:
  - **Shrink rapidly** $\Rightarrow$ vanish
  - **Grow uncontrollably** $\Rightarrow$ explode

Background – Vanishing and Exploding Gradients

- When multiplying many Jacobians, gradients can:
  - **Shrink rapidly** $\Rightarrow$ vanish
  - **Grow uncontrollably** $\Rightarrow$ explode
- In practice: RNNs **forget early inputs** and may become unstable

Background – Vanishing and Exploding Gradients

- When multiplying many Jacobians, gradients can:
  - **Shrink rapidly** $\Rightarrow$ vanish
  - **Grow uncontrollably** $\Rightarrow$ explode
- In practice: RNNs **forget early inputs** and may become unstable
- **Gradient clipping** can limit explosion, but not vanishing

Background – Vanishing and Exploding Gradients

- When multiplying many Jacobians, gradients can:
  - **Shrink rapidly** $\Rightarrow$ vanish
  - **Grow uncontrollably** $\Rightarrow$ explode
- In practice: RNNs **forget early inputs** and may become unstable
- **Gradient clipping** can limit explosion, but not vanishing
- $\rightarrow$ We need an **architectural solution**

Source: Hochreiter 1991; Pascanu et al. 2013

Contents

Naive Solution - Constant Error Flow

Single Unit (j) Constant Error Carousel (CEC):

Source: Hochreiter et al. 1997

Naive Solution - Constant Error Flow

Single Unit (j) Constant Error Carousel (CEC):

- Linear activation function : $f(x) = x$

Source: Hochreiter et al. 1997

Naive Solution - Constant Error Flow

Single Unit (j) Constant Error Carousel (CEC):

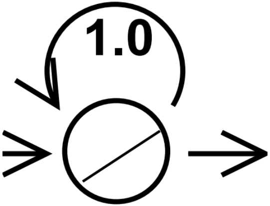
- Linear activation function : $f(x) = x$
- Feedback connection weight : $w_{jj} = 1$

Source: Hochreiter et al. 1997

Naive Solution - Constant Error Flow

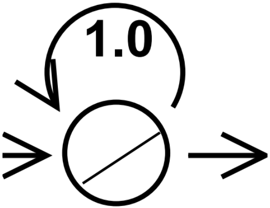
Single Unit (j) Constant Error Carousel (CEC):

- Linear activation function : $f(x) = x$
- Feedback connection weight : $w_{jj} = 1$



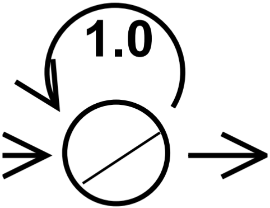
Source: Hochreiter et al. 1997

Naive Solution - Problems



Source: Hochreiter et al. 1997

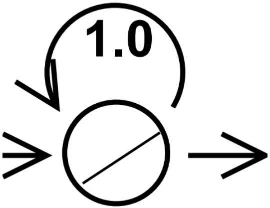
Naive Solution - Problems



Connected to other units:

Source: Hochreiter et al. 1997

Naive Solution - Problems

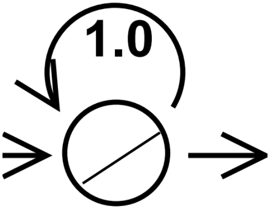


Connected to other units:

- Input from unit i

Source: Hochreiter et al. 1997

Naive Solution - Problems

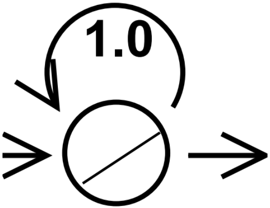


Connected to other units:

- Input from unit i $\implies$ Single weight w_{ji} for storing & ignoring inputs

Source: Hochreiter et al. 1997

Naive Solution - Problems

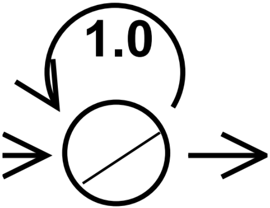


Connected to other units:

- Input from unit i $\Longrightarrow$ Single weight w_{ji} for storing & ignoring inputs
- Output to unit k

Source: Hochreiter et al. 1997

Naive Solution - Problems

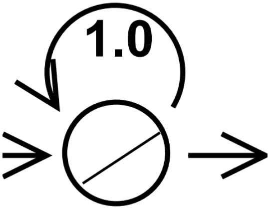


Connected to other units:

- Input from unit $i \implies$ Single weight w_{ji} for storing & ignoring inputs
- Output to unit $k \implies$ Single weight w_{kj} for controlling retrieval

Source: Hochreiter et al. 1997

Naive Solution - Problems



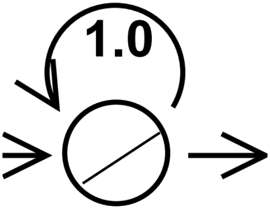
Connected to other units:

- Input from unit $i \implies$ Single weight w_{ji} for storing & ignoring inputs
- Output to unit $k \implies$ Single weight w_{kj} for controlling retrieval

Conflicting weight update signals for w_{ji} & w_{kj}

Source: Hochreiter et al. 1997

Naive Solution - Problems



Connected to other units:

- Input from unit $i \implies$ Single weight w_{ji} for storing & ignoring inputs
- Output to unit $k \implies$ Single weight w_{kj} for controlling retrieval

Conflicting weight update signals for w_{ji} & $w_{kj} \implies$ Mechanism for storage and retrieval

Source: Hochreiter et al. 1997

Contents

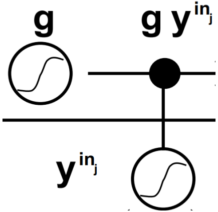
LSTM Architecture - Gate Unit

Idea: Restrict flow based on context

Source: Hochreiter et al. 1997

LSTM Architecture - Gate Unit

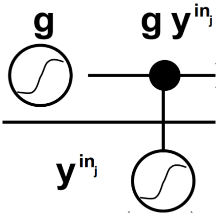
Idea: Restrict flow based on context



Source: Hochreiter et al. 1997

LSTM Architecture - Gate Unit

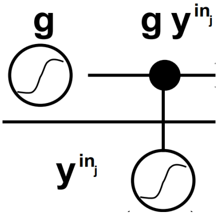
Idea: Restrict flow based on context



Control Input: y^{in_j}

Source: Hochreiter et al. 1997

Idea: Restrict flow based on context



Control Input: y^{in_j}

Input gets scaled by number computed from context

Source: Hochreiter et al. 1997

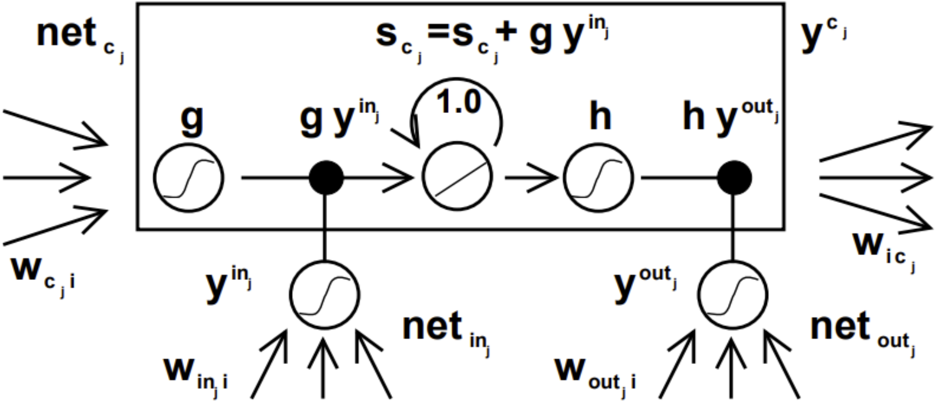
LSTM Architecture - Memory Cell

Memory Cell: CEC unit with input and output gate units

Source: Hochreiter et al. 1997

LSTM Architecture - Memory Cell

Memory Cell: CEC unit with input and output gate units



Source: Hochreiter et al. 1997

LSTM Architecture - Network Topology

Input for cell and gate units depend on specific implementation

LSTM Architecture - Network Topology

Input for cell and gate units depend on specific implementation

They can be:

LSTM Architecture - Network Topology

Input for cell and gate units depend on specific implementation

They can be:

- Input Units

LSTM Architecture - Network Topology

Input for cell and gate units depend on specific implementation

They can be:

- Input Units
- Gate Units

LSTM Architecture - Network Topology

Input for cell and gate units depend on specific implementation

They can be:

- Input Units
- Gate Units
- Memory Cells

LSTM Architecture - Network Topology

Input for cell and gate units depend on specific implementation

They can be:

- Input Units
- Gate Units
- Memory Cells
- Conventional Hidden Units

Input for cell and gate units depend on specific implementation

They can be:

- Input Units
- Gate Units
- Memory Cells
- Conventional Hidden Units

Commonly used topology: Inputs for Cell Gate are all input units memory cell outputs

Input for cell and gate units depend on specific implementation
They can be:

- Input Units
- Gate Units
- Memory Cells
- Conventional Hidden Units

Commonly used topology: Inputs for Cell Gate are all input units memory cell outputs
Input Gate: $i_t = \sigma_i \left(U_i h_{t-1} + W_i x_t + b_i \right)$

LSTM Architecture - Network Topology

Input for cell and gate units depend on specific implementation
They can be:

- Input Units
- Gate Units
- Memory Cells
- Conventional Hidden Units

Commonly used topology: Inputs for Cell Gate are all input units memory cell outputs
Input Gate: $i_t = \sigma_i \left(U_i h_{t-1} + W_i x_t + b_i \right)$
Output Gate: $o_t = \sigma_o \left(U_o h_{t-1} + W_o x_t + b_o \right)$

LSTM Architecture - Network Topology

Input for cell and gate units depend on specific implementation
They can be:

- Input Units
- Gate Units
- Memory Cells
- Conventional Hidden Units

Commonly used topology: Inputs for Cell Gate are all input units memory cell outputs
Input Gate: $i_t = \sigma_i \left(U_i h_{t-1} + W_i x_t + b_i \right)$
Output Gate: $o_t = \sigma_o \left(U_o h_{t-1} + W_o x_t + b_o \right)$
Internal State: $C_t = C_{t-1} + i_t \odot \sigma_c \left(U_c h_{t-1} + W_x x_t + b_c \right)$

LSTM Architecture - Network Topology

Input for cell and gate units depend on specific implementation
They can be:

- Input Units
- Gate Units
- Memory Cells
- Conventional Hidden Units

Commonly used topology: Inputs for Cell Gate are all input units memory cell outputs
Input Gate: $i_t = \sigma_i\left(U_i h_{t-1} + W_i x_t + b_i\right)$
Output Gate: $o_t = \sigma_o\left(U_o h_{t-1} + W_o x_t + b_o\right)$
Internal State: $C_t = C_{t-1} + i_t \odot \sigma_c\left(U_c h_{t-1} + W_x x_t + b_c\right)$
Output: $h_t = o_t \odot \sigma_h\left(C_t\right)$

LSTM Architecture - Results

LSTMs outperform classical RNNs in many ways:

LSTM Architecture - Results

LSTMs outperform classical RNNs in many ways:

- Less training examples needed

LSTM Architecture - Results

LSTMs outperform classical RNNs in many ways:

- Less training examples needed
- Higher success rate

LSTM Architecture - Results

LSTMs outperform classical RNNs in many ways:

- Less training examples needed
- Higher success rate
- Better performance on sequences that include random data

LSTM Architecture - Results

LSTMs outperform classical RNNs in many ways:

- Less training examples needed
- Higher success rate
- Better performance on sequences that include random data
- Better performance on noisy data

LSTM Architecture - Results

LSTMs outperform classical RNNs in many ways:

- Less training examples needed
- Higher success rate
- Better performance on sequences that include random data
- Better performance on noisy data
- Can solve tasks RNNs previously could not (sequences with no local regularities)

LSTM Architecture - Limitations

Drawbacks:

LSTM Architecture - Limitations

Drawbacks:

- Exploding Gradients

LSTM Architecture - Limitations

Drawbacks:

- Exploding Gradients
- Computational Intensity

LSTM Architecture - Limitations

Drawbacks:

- Exploding Gradients
- Computational Intensity
- Limited GPU Utilization

LSTM Architecture - Limitations

Drawbacks:

- Exploding Gradients
- Computational Intensity
- Limited GPU Utilization
- Memory Bottlenecks

LSTM Architecture - Limitations

Drawbacks:

- Exploding Gradients
- Computational Intensity
- Limited GPU Utilization
- Memory Bottlenecks
- Worse performance on short sequences

LSTM Architecture - Limitations

Drawbacks:

- Exploding Gradients
- Computational Intensity
- Limited GPU Utilization
- Memory Bottlenecks
- Worse performance on short sequences
- State can not be reset

Contents

Variants - LSTM with forget gate

Forget Gate:

Source: Gers et al. 1999

Variants - LSTM with forget gate

Forget Gate:

- "Forgetting" by resetting state

Source: Gers et al. 1999

Variants - LSTM with forget gate

Forget Gate:

- "Forgetting" by resetting state
- Achieved by gating the previous state

Source: Gers et al. 1999

Variants - LSTM with forget gate

Forget Gate:

- "Forgetting" by resetting state
- Achieved by gating the previous state
- $f_t = \sigma_i \left(U_f h_{t-1} + W_f x_t + b_f \right)$

Source: Gers et al. 1999

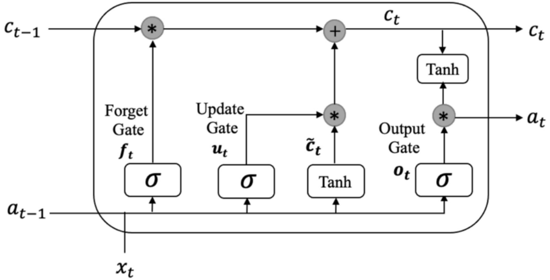
Variants - LSTM with forget gate

Forget Gate:

- "Forgetting" by resetting state
- Achieved by gating the previous state
- $f_t = \sigma_i\left(U_f h_{t-1} + W_f x_t + b_f\right)$
- $C_t = f_t \odot C_{t-1} + i_t \odot \sigma_c\left(U_c h_{t-1} + W_x x_t + b_c\right)$

Source: Gers et al. 1999

Variants - LSTM with forget gate



Source: Nguyen et al. 2022

Variants - Bidirectional LSTM

- Standard LSTM processes sequence **left to right**

Variants - Bidirectional LSTM

- Standard LSTM processes sequence **left to right**
- Bidirectional LSTM (BiLSTM) runs **two LSTMs**:
  - One forward ($\rightarrow$)
  - One backward ($\leftarrow$)

Variants - Bidirectional LSTM

- Standard LSTM processes sequence **left to right**
- Bidirectional LSTM (BiLSTM) runs **two LSTMs**:
  - One forward ($\rightarrow$)
  - One backward ($\leftarrow$)
- Output combines both:

$$h_t = [\overrightarrow{h}_t; \overleftarrow{h}_t]$$

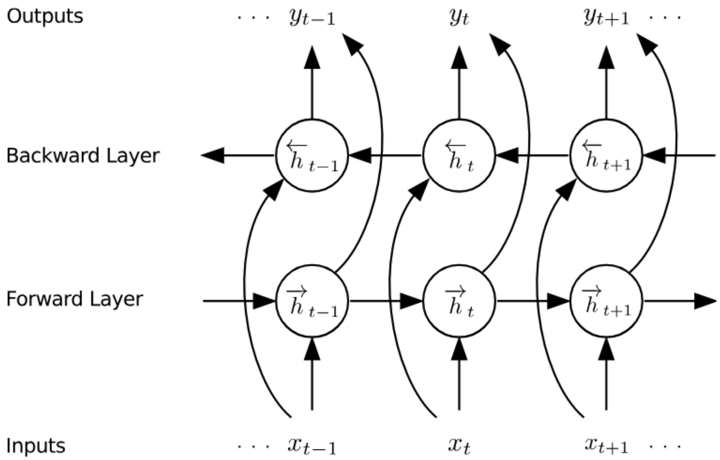
Variants - Bidirectional LSTM

- Standard LSTM processes sequence **left to right**
- Bidirectional LSTM (BiLSTM) runs **two LSTMs**:
  - One forward ($\rightarrow$)
  - One backward ($\leftarrow$)
- Output combines both:

$$h_t = [\overrightarrow{h}_t ; \overleftarrow{h}_t]$$

- Can use **past and future** context $\rightarrow$ better performance in many NLP tasks

Variants - Bidirectional LSTM



Source: Graves & Schmidhuber 2005

Gated Recurrent Unit

- GRU is a **simplified version of LSTM** with fewer gates

Gated Recurrent Unit

- GRU is a **simplified version of LSTM** with fewer gates
- Uses:
  - an **update gate** $\rightarrow$ controls how much past info to keep
  - a **reset gate** $\rightarrow$ controls how much past info to forget

Gated Recurrent Unit

- GRU is a **simplified version of LSTM** with fewer gates
- Uses:
  - an **update gate** $\rightarrow$ controls how much past info to keep
  - a **reset gate** $\rightarrow$ controls how much past info to forget
- Learns to balance:
  - remembering important past features
  - reacting to new input

Gated Recurrent Unit

- GRU is a **simplified version of LSTM** with fewer gates
- Uses:
  - an **update gate** $\rightarrow$ controls how much past info to keep
  - a **reset gate** $\rightarrow$ controls how much past info to forget
- Learns to balance:
  - remembering important past features
  - reacting to new input
- Often performs well with:
  - less data
  - shorter sequences
  - faster training needs

Source: Cho et al. 2014

GRU vs. LSTM

- **LSTM**
  - Uses a separate memory cell c_t
  - Has input and output gates (forget gate was added later)
  - Two internal states: h_t and c_t

GRU vs. LSTM

- **LSTM**
  - Uses a separate memory cell c_t
  - Has input and output gates (forget gate was added later)
  - Two internal states: h_t and c_t
- **GRU**
  - Merges cell and hidden state into a single h_t
  - Uses update and reset gates
  - Simpler architecture, faster training

GRU simplifies LSTM by merging its memory and control into fewer components.

Contents

LSTM vs. Transformer – Motivation

- LSTMs have been widely used for sequence modeling since the 1990s

LSTM vs. Transformer – Motivation

- LSTMs have been widely used for sequence modeling since the 1990s
- In recent years, **Transformers** have become the dominant model in NLP and beyond

LSTM vs. Transformer – Motivation

- LSTMs have been widely used for sequence modeling since the 1990s
- In recent years, **Transformers** have become the dominant model in NLP and beyond
- Transformers offer a different approach: no recurrence, full attention

Source: Vaswani et al. 2017

Transformers – Self-Attention Mechanism

- Core idea: Each word **attends to all others** in the input

Transformers – Self-Attention Mechanism

- Core idea: Each word **attends to all others** in the input
- Use **self-attention** to compute weighted combinations of all inputs

Transformers – Self-Attention Mechanism

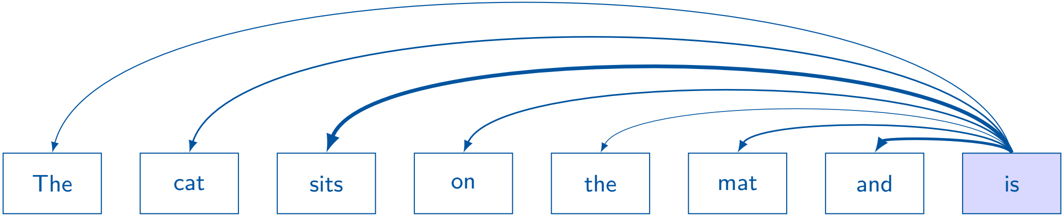
- Core idea: Each word **attends to all others** in the input
- Use **self-attention** to compute weighted combinations of all inputs
- Attention weights learned dynamically — no recurrence

Transformers – Self-Attention Mechanism

- Core idea: Each word **attends to all others** in the input
- Use **self-attention** to compute weighted combinations of all inputs
- Attention weights learned dynamically — no recurrence
- Enables **parallelization** and efficient learning of long-range dependencies

Transformers – Self-Attention Mechanism

- Core idea: Each word **attends to all others** in the input
- Use **self-attention** to compute weighted combinations of all inputs
- Attention weights learned dynamically — no recurrence
- Enables **parallelization** and efficient learning of long-range dependencies



Source: Vaswani et al. 2017

LSTM vs. Transformer – Comparison

LSTM

- Recurrence-based
- Hidden state carries memory
- Sequential processing
- Better on smaller data
- Fewer resources needed

Transformer

- Attention-based
- Global context at each step
- Fully parallelizable
- Excels with large datasets
- High compute demand

Contents

Application – Speech Recognition

- Spoken language is a sequence of sounds that unfolds over time

Application – Speech Recognition

- Spoken language is a sequence of sounds that unfolds over time
- LSTMs can learn how current sounds depend on earlier ones:
  - e.g. understanding that "s" at the end of "cats" depends on earlier phonemes

Application – Speech Recognition

- Spoken language is a sequence of sounds that unfolds over time
- LSTMs can learn how current sounds depend on earlier ones:
  - e.g. understanding that ”s” at the end of “cats” depends on earlier phonemes
- Bidirectional LSTMs are especially useful:
  - They consider both past and future context in a sentence
  - This helps distinguish similar sounds in different contexts

Application – Speech Recognition

- Spoken language is a sequence of sounds that unfolds over time
- LSTMs can learn how current sounds depend on earlier ones:
  - e.g. understanding that "s" at the end of "cats" depends on earlier phonemes
- Bidirectional LSTMs are especially useful:
  - They consider both past and future context in a sentence
  - This helps distinguish similar sounds in different contexts
- LSTMs reduce the need for manual feature extraction

Applications - Time Series Forecasting

LSTMs perform better on:

Applications - Time Series Forecasting

LSTMs perform better on:

- Long sequences

Applications - Time Series Forecasting

LSTMs perform better on:

- Long sequences
- Complex sequences

Applications - Time Series Forecasting

LSTMs perform better on:

- Long sequences
- Complex sequences

Examples:

Applications - Time Series Forecasting

LSTMs perform better on:

- Long sequences
- Complex sequences

Examples:

- 2022: Deep LSTM Network forecasts electricity demand (1.5% accuracy)

Applications - Time Series Forecasting

LSTMs perform better on:

- Long sequences
- Complex sequences

Examples:

- 2022: Deep LSTM Network forecasts electricity demand (1.5% accuracy)
- 2024: LSTM Architecture forcasts electricity prices

Applications - Music Generation

Music has temporal structure:

Applications - Music Generation

Music has temporal structure:

- Rhythms

Music has temporal structure:

- Rhythms
- Chord progressions

Applications - Music Generation

Music has temporal structure:

- Rhythms
- Chord progressions
- Melodic motives

Applications - Music Generation

Music has temporal structure:

- Rhythms
- Chord progressions
- Melodic motives

Easily representable as numbers

Applications - Music Generation

Music has temporal structure:

- Rhythms
- Chord progressions
- Melodic motives

Easily representable as numbers

2002: LSTM Architecture used to compose music

Contents

Discussion - the Future of LSTMs

Future of the LSTM architecture:

Source: Zhao et al. 2025

Discussion - the Future of LSTMs

Future of the LSTM architecture:

- Largely replaced by transformers in NLP and vision, but:

Source: Zhao et al. 2025

Discussion - the Future of LSTMs

Future of the LSTM architecture:

- Largely replaced by transformers in NLP and vision, but:
  - LSTMs still useful for small datasets (better performance)

Source: Zhao et al. 2025

Discussion - the Future of LSTMs

Future of the LSTM architecture:

- Largely replaced by transformers in NLP and vision, but:
  - LSTMs still useful for small datasets (better performance)
  - LSTMs require fewer Resources

Source: Zhao et al. 2025

Discussion - the Future of LSTMs

Future of the LSTM architecture:

- Largely replaced by transformers in NLP and vision, but:
  - LSTMs still useful for small datasets (better performance)
  - LSTMs require fewer Resources
  - LSTM - Transformer hybrid architectures

Source: Zhao et al. 2025

Discussion - the Future of LSTMs

Future of the LSTM architecture:

- Largely replaced by transformers in NLP and vision, but:
  - LSTMs still useful for small datasets (better performance)
  - LSTMs require fewer Resources
  - LSTM - Transformer hybrid architectures
- Inspired many Variants and Architecture

Source: Zhao et al. 2025

Contents

Conclusion

LSTM Architecture:

Conclusion

LSTM Architecture:

- Internal CEC unit $\Longrightarrow$ Internal State (Memory)

Conclusion

LSTM Architecture:

- Internal CEC unit $\Longrightarrow$ Internal State (Memory)
- Input Gate $\Longrightarrow$ Protects memory from unwanted perturbation

Conclusion

LSTM Architecture:

- Internal CEC unit $\Longrightarrow$ Internal State (Memory)
- Input Gate $\Longrightarrow$ Protects memory from unwanted perturbation
- Output Gate $\Longrightarrow$ Controls output of the memory to other units

Conclusion

LSTM Architecture:

- Internal CEC unit $\implies$ Internal State (Memory)
- Input Gate $\implies$ Protects memory from unwanted perturbation
- Output Gate $\implies$ Controls output of the memory to other units
- Eliminates vanishing Gradient Problem

Conclusion

LSTM Architecture:

- Internal CEC unit $\Longrightarrow$ Internal State (Memory)
- Input Gate $\Longrightarrow$ Protects memory from unwanted perturbation
- Output Gate $\Longrightarrow$ Controls output of the memory to other units
- Eliminates vanishing Gradient Problem
- Inspired many Variants and RNN Architectures

Conclusion

LSTM Architecture:

- Internal CEC unit $\implies$ Internal State (Memory)
- Input Gate $\implies$ Protects memory from unwanted perturbation
- Output Gate $\implies$ Controls output of the memory to other units
- Eliminates vanishing Gradient Problem
- Inspired many Variants and RNN Architectures

Currently:

Conclusion

LSTM Architecture:

- Internal CEC unit $\Longrightarrow$ Internal State (Memory)
- Input Gate $\Longrightarrow$ Protects memory from unwanted perturbation
- Output Gate $\Longrightarrow$ Controls output of the memory to other units
- Eliminates vanishing Gradient Problem
- Inspired many Variants and RNN Architectures

Currently:

- Replaced by transformers in many Applications

Conclusion

LSTM Architecture:

- Internal CEC unit $\implies$ Internal State (Memory)
- Input Gate $\implies$ Protects memory from unwanted perturbation
- Output Gate $\implies$ Controls output of the memory to other units
- Eliminates vanishing Gradient Problem
- Inspired many Variants and RNN Architectures

Currently:

- Replaced by transformers in many Applications
- Development of new LSTM based architectures

Conclusion

LSTM Architecture:

- Internal CEC unit $\Longrightarrow$ Internal State (Memory)
- Input Gate $\Longrightarrow$ Protects memory from unwanted perturbation
- Output Gate $\Longrightarrow$ Controls output of the memory to other units
- Eliminates vanishing Gradient Problem
- Inspired many Variants and RNN Architectures

Currently:

- Replaced by transformers in many Applications
- Development of new LSTM based architectures
- Remain useful in some cases

References i

Safwan Mahmood Al-Selwi, Mohd Fadzil Hassan, Said Jadid Abdulkadir, Amgad Muneer, Ebrahim Hamid Sumiea, Alawi Alqushaibi, and Mohammed Gamal Ragab.
Rnn-lstm: From applications to modeling techniques and beyond—systematic review.
Journal of King Saud University - Computer and Information Sciences, 36(5):102068, 2024.

Roland Bolboaca and Haller Piroska.
Performance analysis of long short-term memory predictive neural networks on time series data.
Mathematics, 11:1432, 03 2023.

Jeffrey L. Elman.
Finding structure in time.
Cognitive Science, 14(2):179–211, 1990.

D. Eck and J. Schmidhuber.
Finding temporal structure in music: blues improvisation with lstm recurrent networks.
In *Proceedings of the 12th IEEE Workshop on Neural Networks for Signal Processing*, pages 747–756, 2002.

References ii

Felix A. Gers, Douglas Eck, and Jürgen Schmidhuber.
Applying lstm to time series predictable through time-window approaches.
In Roberto Tagliaferri and Maria Marinaro, editors, *Neural Nets WIRN Vietri-01*, pages 193–200, London, 2002. Springer London.

Tayfun Gokmen, Malte J. Rasch, and Wilfried Haensch.
Training lstm networks with resistive cross-point devices.
Frontiers in Neuroscience, Volume 12 - 2018, 2018.

F.A. Gers, J. Schmidhuber, and F. Cummins.
Learning to forget: continual prediction with lstm.
In *1999 Ninth International Conference on Artificial Neural Networks ICANN 99. (Conf. Publ. No. 470)*, volume 2, pages 850–855 vol.2, 1999.

Zellig S. Harris.
Distributional structure.
WORD, 10(2-3):146–162, 1954.

J. Hochreiter.
Untersuchungen zu dynamischen neuronalen netzen.
https://people.idsia.ch/ juergen/SeppHochreiter1991ThesisAdvisorSchmidhuber.pdf, 1991.
Accessed: June 2025.

Sepp Hochreiter and Jürgen Schmidhuber.
Long short-term memory.
Neural computation, 9(8):1735–1780, 1997.

Deniz Kenan Kılıç, Peter Nielsen, and Amila Thibbotuwawa.
Intraday electricity price forecasting via lstm and trading strategy for the power market: A case study of the west denmark dk1 grid region.
Energies, 17(12), 2024.

Tomáš Mikolov.
STATISTICAL LANGUAGE MODELS BASED ON NEURAL NETWORKS.
Ph.d. thesis, Brno University of Technology, Faculty of Information Technology, 2012.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio.
On the difficulty of training recurrent neural networks.
In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1310–1318, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

J.F. Torres, F. Martínez-Álvárez, and A. Troncoso.
"a deep lstm network for the spanish electricity consumption forecasting".
"Neural Computing and Applications ", 34:10533–10545, 2022.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin.
Attention is all you need.
In *NIPS*, 2017.

P.J. Werbos.
Backpropagation through time: what it does and how to do it.
Proceedings of the IEEE, 78(10):1550–1560, 1990.

Hongyu Zhu, Mohamed Akrout, Bojian Zheng, Andrew Pelegris, Anand Jayarajan, Amar Phanishayee, Bianca Schroeder, and Gennady Pekhimenko.
Benchmarking and analyzing deep neural network training.
In *2018 IEEE International Symposium on Workload Characterization (IISWC)*, pages 88–100, 2018.

Yali Zhao, Yingying Guo, and Xuecheng Wang.
Hybrid lstm–transformer architecture with multi-scale feature fusion for high-accuracy gold futures price forecasting.
Mathematics, 13(10), 2025.

Bojian Zheng, Nandita Vijaykumar, and Gennady Pekhimenko.
Echo: Compiler-based GPU Memory Footprint Reduction for LSTM RNN Training .
In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 1089–1102, Los Alamitos, CA, USA, June 2020. IEEE Computer Society.