CSE 435: Data Mining

RNNs, Sequence-to-Sequence Learning, Attention Mechanism, Transformers and Applications

Md Atikuzzaman
Lecturer
Department of Computer Science & Engineering
atik@cse.green.edu.bd

GREEN UNIVERSITY OF BANGLADESH

Outline

RNNs

- Sequence data
- Recurrent state
- Unrolling
- Long-term dependence
- BPTT
- LSTM / GRU

Seq2Seq

- Encoder–decoder
- Context vector
- Variable-length output
- Teacher forcing
- Decoding

Attention

- Motivation
- Alignment
- Context vectors
- Q, K, V
- Scaled attention

Transformers

- Self-attention
- Multi-head attention
- Position encoding
- Encoder / decoder
- Applications

Why Do We Need Sequence Models?

Sequence data contain **order** and **dependencies** between observations.

Examples

- Sentences and documents
- Speech and audio
- Sensor measurements
- Financial time series
- Weather observations
- User click sequences

Important Questions

- What happened previously?
- Does order matter?
- Which past event affects now?
- How long does dependency last?
- Is prediction causal?

Key Idea

Randomly shuffling sequence elements may destroy the temporal or linguistic information that must be learned.

RNNs Introduce Recurrence

A feed-forward network processes each input independently.
An **RNN** introduces a hidden state that carries information from earlier time steps.

Feed-Forward Network

$$\mathbf{x}_t \rightarrow \mathbf{h}_t \rightarrow \hat{\mathbf{y}}_t$$

No explicit connection exists to the previous state.

Recurrent Neural Network

$$(\mathbf{x}_t, \mathbf{h}_{t-1}) \rightarrow \mathbf{h}_t \rightarrow \hat{\mathbf{y}}_t$$

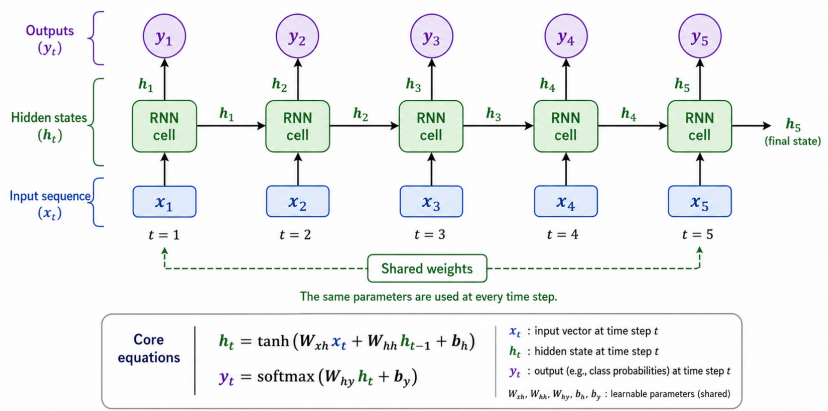
The previous hidden state influences the current state.

$$\boxed{\mathbf{h}_t = f(\mathbf{W}_{xh}\mathbf{x}_t + \mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{b}_h)}$$

Important

The same recurrent parameters are reused at every time step.

RNN Architecture: Unrolled Through Time



Core Idea

- One recurrent cell is reused at every time step.

Sequence Flow

- Inputs $x_1, x_2, \ldots, x_T$ enter sequentially.
- Outputs $\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_T$ may be produced

Different Input–Output Structures of RNNs

Structure	Example	Purpose
One-to-One	Image classification	Single input to single output
One-to-Many	Image captioning	Single input generates sequence
Many-to-One	Sentiment analysis	Sequence mapped to one label
Many-to-Many Aligned	POS tagging	One output for each input
Many-to-Many Unaligned	Machine translation	Input and output lengths differ

Important

Sequence-to-sequence learning is particularly useful for **many-to-many unaligned** problems.

Long-Term Dependencies and BPTT

RNNs are trained using **Backpropagation Through Time (BPTT)**.

$$\frac{\partial \mathcal{L}}{\partial \mathbf{h}_k} = \frac{\partial \mathcal{L}}{\partial \mathbf{h}_t} \prod_{j=k+1}^{t} \frac{\partial \mathbf{h}_j}{\partial \mathbf{h}_{j-1}}$$

Repeated multiplication of derivatives creates two major problems.

Vanishing Gradient

$$\|\text{gradient}\| \to 0$$

- Early inputs receive little learning signal
- Long-term dependencies are difficult

Exploding Gradient

$$\|\text{gradient}\| \to \infty$$

- Very large parameter updates
- Unstable training

LSTM: Long Short-Term Memory

The PPTX introduces LSTM as a solution for preserving information over longer sequences.

Simple RNN

- One hidden state
- Repeated nonlinear transformation
- Long-term influence can decay rapidly

LSTM

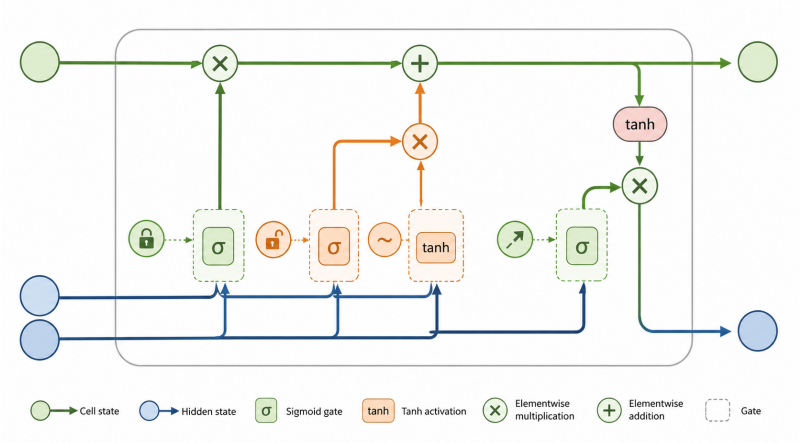
- Adds a persistent cell state $\mathbf{c}_t$
- Uses gates to control information flow
- Better suited to long dependencies

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t$$

Three major gates:

Forget Gate Input Gate Output Gate

LSTM Cell Architecture



- The cell-state path preserves long-term information.
- Forget, input, and output gates regulate memory flow.

LSTM Gates and Hidden-State Update

Forget Gate

$$\mathbf{f}_t = \sigma(W_f[\mathbf{h}_{t-1}, \mathbf{x}_t] + b_f)$$

Controls how much old memory remains.

Input Gate

$$\mathbf{i}_t = \sigma(W_i[\mathbf{h}_{t-1}, \mathbf{x}_t] + b_i)$$

Controls how much new information is stored.
Cell and hidden states:

Candidate Memory

$$\tilde{\mathbf{c}}_t = \tanh(W_c[\mathbf{h}_{t-1}, \mathbf{x}_t] + b_c)$$

Output Gate

$$\mathbf{o}_t = \sigma(W_o[\mathbf{h}_{t-1}, \mathbf{x}_t] + b_o)$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t$$

$$\boxed{\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)}$$

GRU: A Simpler Gated RNN

The GRU is a commonly used LSTM variant with fewer gates.

Update Gate

$$\mathbf{z}_t = \sigma(W_z[\mathbf{h}_{t-1}, \mathbf{x}_t])$$

Controls the balance between old and new information.

Reset Gate

$$\mathbf{r}_t = \sigma(W_r[\mathbf{h}_{t-1}, \mathbf{x}_t])$$

Controls how much past information affects the candidate state.

Candidate

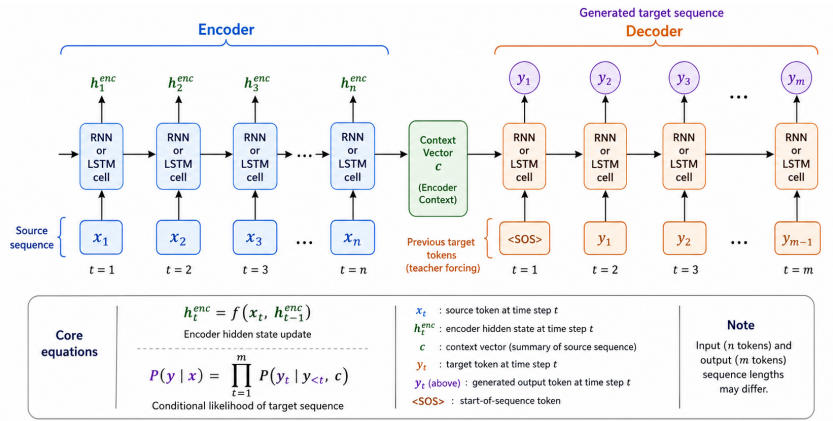
$$\tilde{\mathbf{h}}_t = \tanh(W_h[\mathbf{r}_t \odot \mathbf{h}_{t-1}, \mathbf{x}_t])$$

State Update

$$\boxed{\mathbf{h}_t = (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \tilde{\mathbf{h}}_t}$$

GRUs usually contain fewer parameters than LSTMs, but the better choice is task-dependent.

Seq2Seq Architecture: Encoder–Decoder



Encoder

- Reads source tokens $x_1, \ldots, x_n$.
- Produces a context representation.

Decoder

- Generates target tokens $y_1, \ldots, y_m$.
- Uses previous target tokens during

Basic Encoder–Decoder Model

Encoder

$$\mathbf{h}_t^{enc} = f_{enc}(\mathbf{x}_t, \mathbf{h}_{t-1}^{enc})$$

After reading the final source token:

$$\boxed{\mathbf{c} = \mathbf{h}_T^{enc}}$$

The entire source sequence is compressed into one vector.

Decoder

$$\mathbf{h}_t^{dec} = f_{dec}(y_{t-1}, \mathbf{h}_{t-1}^{dec}, \mathbf{c})$$

$$P(\mathbf{y}|\mathbf{x}) = \prod_{t=1}^{T_y} P(y_t|y_{<t}, \mathbf{c})$$

Fixed-Vector Bottleneck

One context vector must preserve all source information. Performance therefore tends to degrade for long or complex sequences.

Training and Decoding Seq2Seq Models

Teacher Forcing
During training, use the correct previous target:

$$y_{t-1}^{true} \rightarrow Decoder \rightarrow \hat{y}_t$$

- Stabilizes training
- Speeds convergence
- Creates exposure bias

Inference
The true previous token is unavailable.

$$\hat{y}_{t-1} \rightarrow Decoder \rightarrow \hat{y}_t$$

Common strategies:

- Greedy decoding
- Beam search

Generation usually begins with `<SOS>` and stops at `<EOS>`.

Why Do We Need Attention?

Traditional Seq2Seq:

$$\text{Source Sequence} \rightarrow \boxed{\text{One Summary Vector}} \rightarrow \text{Target Sequence}$$

This fixed representation creates an information bottleneck.

Attention instead allows the decoder to dynamically access **all encoder hidden states**.

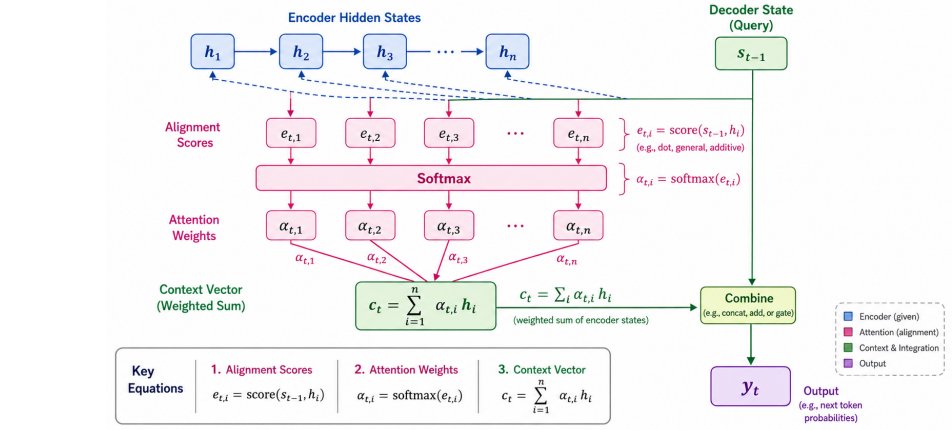
$$\boxed{\mathbf{c}_t = \sum_{i=1}^{T_x} \alpha_{t,i} \mathbf{h}_i^{enc}}$$

- $\alpha_{t,i}$ measures relevance of source position i.
- A different context vector is created for each output step.
- Important source positions receive larger weights.

Main Improvement

The decoder no longer depends on only one fixed source summary.

Attention Architecture: Dynamic Context Vector



- The decoder attends to all encoder hidden states instead of one fixed summary.
- Larger attention weights contribute more to the context vector.

Attention Alignment and Context Vector

For decoder position t, compute relevance scores:

$$e_{t,i} = Score(\mathbf{h}_{t-1}^{dec}, \mathbf{h}_i^{enc})$$

Normalize using softmax:

$$\alpha_{t,i} = \frac{\exp(e_{t,i})}{\sum_j \exp(e_{t,j})}$$

Then construct the context vector:

$$\boxed{\mathbf{c}_t = \sum_i \alpha_{t,i} \mathbf{h}_i^{enc}}$$

Large $\alpha_{t,i}$

- Source position is highly relevant
- Contributes strongly to context

Small $\alpha_{t,i}$

- Source position is less relevant
- Has little influence at that output step

Worked Attention Example

Suppose attention scores for three source positions are:

$$(1,\ 2,\ 0)$$

Softmax converts them into attention weights:

$$\alpha_i = \frac{e^{e_i}}{e^1 + e^2 + e^0}$$

$$(\alpha_1, \alpha_2, \alpha_3) = (0.245,\ 0.665,\ 0.090)$$

Suppose the corresponding scalar values are:

$$v_1 = 2, \qquad v_2 = 4, \qquad v_3 = 1$$

The context becomes:

$$c = (0.245)(2) + (0.665)(4) + (0.090)(1)$$

Query, Key, and Value

Modern attention separates three roles.

Query q
What information is the current position looking for?

Key k
What information does each candidate position represent?

Value v
What information is returned if that position is selected?

$$\mathit{Score}(\mathbf{q}, \mathbf{k}) = \mathbf{q}^\top \mathbf{k}$$

$$\text{Attention weights} = \mathit{Softmax}(\mathit{Score})$$

$$\boxed{\mathit{Output} = \sum_i \alpha_i \mathbf{v}_i}$$

This query–key–value formulation becomes the foundation of Transformer attention.

From RNN Attention to Transformers

The uploaded PPTX connects attention with **Transformer** and **BERT**-style architectures.

RNN-Based Sequence Model

$$h_1 \rightarrow h_2 \rightarrow h_3 \rightarrow \cdots$$

- Sequential computation
- Hidden-state recurrence
- Limited training parallelism

Transformer

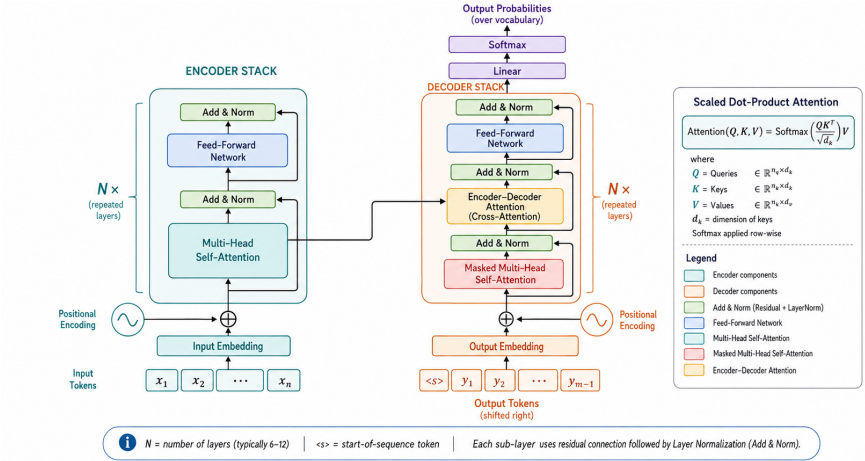
$$x_1 \leftrightarrow x_2 \leftrightarrow x_3 \leftrightarrow \cdots$$

- No recurrent hidden state required
- Direct token-to-token interaction
- Highly parallel training

Core Idea

Transformers replace recurrence with attention-based interactions between sequence positions.

Transformer Architecture: Encoder–Decoder Stack



- Encoder blocks use self-attention and feed-forward layers.
- Decoder blocks use masked self-attention and cross-attention to encoder outputs.

Scaled Dot-Product Attention

The fundamental Transformer operation is:

$$\textit{Attention}(Q, K, V) = \textit{Softmax}\left(\frac{QK^\top}{\sqrt{d_k}} + M\right) V$$

Components

- Q: query matrix
- K: key matrix
- V: value matrix
- d_k: key dimension
- M: optional mask

Operations

1. Compute $QK^\top$
2. Scale by $\sqrt{d_k}$
3. Apply mask if required
4. Apply softmax
5. Combine values

Scaling prevents large dot products from making softmax excessively sharp.

Self-Attention

For sequence representation:

$$X \in \mathbb{R}^{T \times d_{model}}$$

create:

$$Q = XW^Q, \qquad K = XW^K, \qquad V = XW^V$$

Then:

$$\boxed{Z = Softmax\left(\frac{QK^\top}{\sqrt{d_k}}\right)V}$$

- Each position can attend to other relevant positions.
- Dependencies do not need to pass through many recurrent steps.
- Encoder self-attention can access the full input sequence.
- Decoder self-attention masks future positions.

Multi-Head Attention and Positional Information

Multi-Head Attention

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

$$MHA(Q, K, V) = Concat(head_1, \ldots, head_H)W^O$$

Different heads can learn different relationships.

Positional Information

Self-attention alone does not encode sequence order.

One classical solution is:

$$PE(pos, 2i) = \sin\left(\frac{pos}{10000^{2i/d}}\right)$$

$$PE(pos, 2i+1) = \cos\left(\frac{pos}{10000^{2i/d}}\right)$$

Transformer Encoder and Decoder

Encoder Layer

1. Multi-head self-attention
2. Residual connection
3. Layer normalization
4. Feed-forward network
5. Residual + normalization

Encoder attention normally sees the complete source sequence.

Decoder Layer

1. Masked self-attention
2. Encoder–decoder attention
3. Feed-forward network
4. Residual connections
5. Layer normalization

Decoder masking prevents access to future target tokens.

$$P(\mathbf{y}|\mathbf{x}) = \prod_t P(y_t|y_{<t}, \mathbf{x})$$

RNNs vs. Transformers

Property	RNN / LSTM / GRU	Transformer
Sequence processing	Sequential	Highly parallel during training
Memory mechanism	Hidden / cell state	Attention-based representations
Long dependencies	Can be difficult	Direct attention path
Streaming	Naturally suitable	Requires causal/chunked design
Long-sequence cost	Sequential computation	Standard attention is quadratic
Small datasets	Often effective	Frequently benefits from pretraining
Typical strengths	Compact temporal modelling	Large-scale representation and generation

Model Selection

The most powerful architecture is not always the most appropriate. Choose according to data, sequence length, latency, causality, and hardware.

Applications of RNNs, Attention and Transformers

Domain	Applications	Suitable Models
Natural Language	Sentiment, translation, summarization, QA	BiRNN, LSTM, Transformer
Speech	Recognition, synthesis, translation	LSTM, GRU, Transformer
Time Series	Forecasting, anomaly detection, monitoring	RNN, LSTM, Transformer
Recommendation	Next-item prediction, click sequences	GRU, sequential Transformer
Computer Vision	Image classification, detection	Vision Transformer
Healthcare	Patient sequences, event prediction	LSTM, Transformer
Multimodal AI	Text–image, audio–text modelling	Transformer-based architectures

References


S. Hochreiter and J. Schmidhuber,
“Long Short-Term Memory,”
Neural Computation, 1997.


M. Schuster and K. K. Paliwal,
“Bidirectional Recurrent Neural Networks,”
IEEE Transactions on Signal Processing, 1997.


I. Sutskever, O. Vinyals, and Q. V. Le,
“Sequence to Sequence Learning with Neural Networks,”
NeurIPS, 2014.


D. Bahdanau, K. Cho, and Y. Bengio,
“Neural Machine Translation by Jointly Learning to Align and Translate,”
ICLR, 2015.


A. Vaswani et al.,
“Attention Is All You Need,”
NeurIPS, 2017.


A. Zhang et al.,
Dive into Deep Learning,
Cambridge University Press, 2023.