Introduction to Recurrent Neural Network

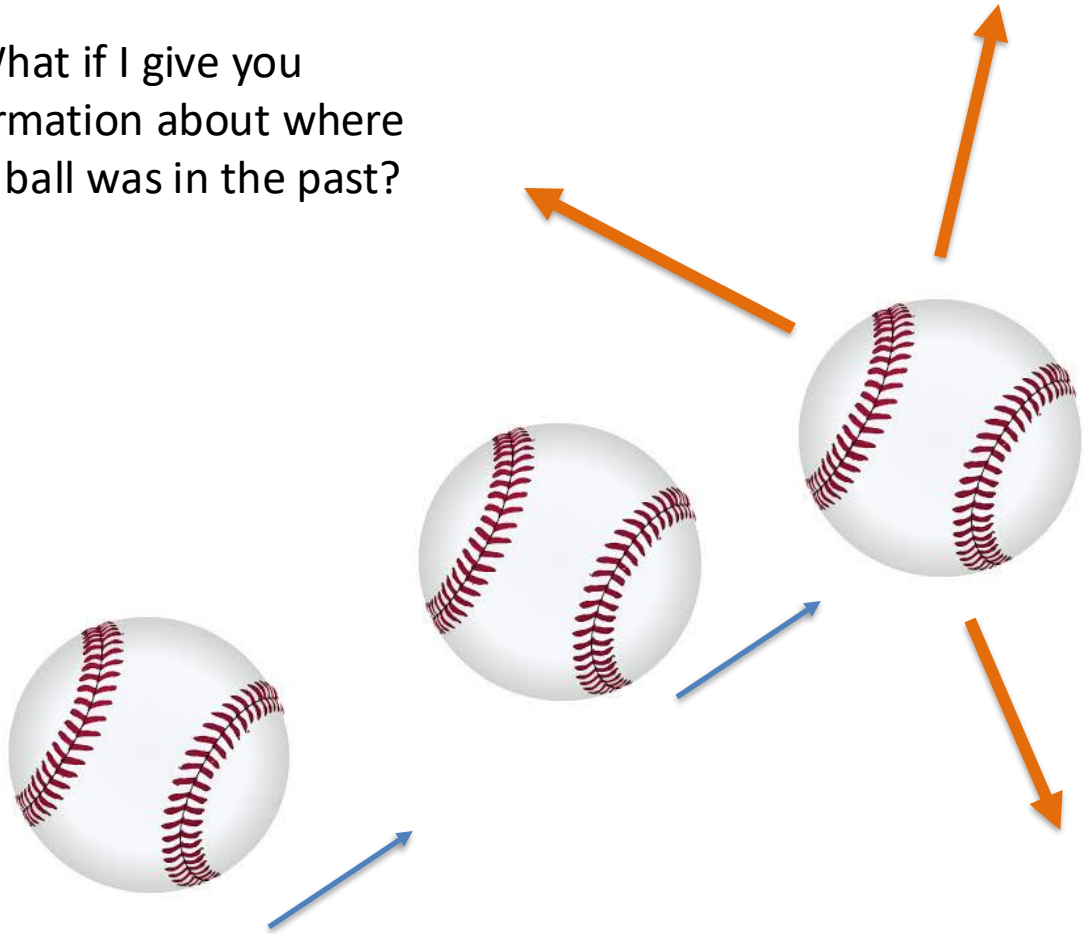
Sammie Omranian

June 2025

UWM | College of Engineering & Applied Science

What is Sequential Data?

3- What if I give you information about where that ball was in the past?



???

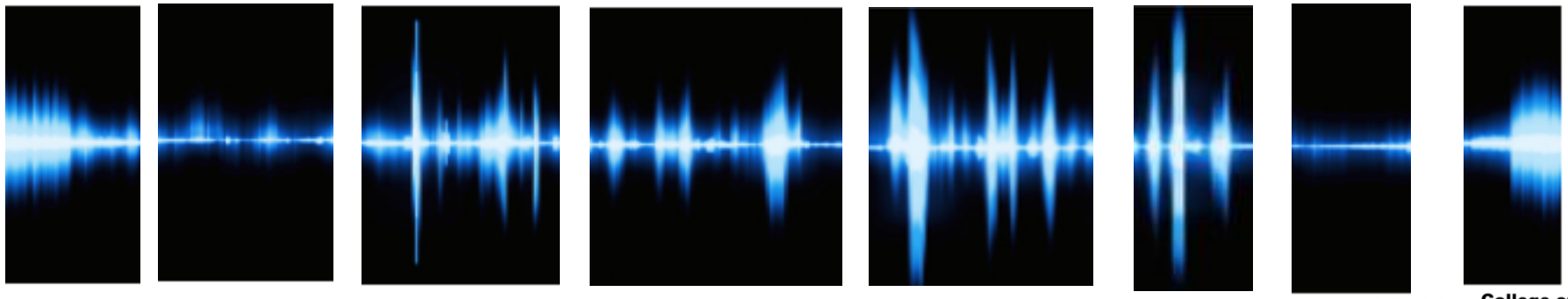
1- Your guess on its next position?

2- No prior information

Sequential Data

- **Sequential data is all around us!**
- **My voice!**

Audio wave form



Sequence of sound waves

Sequential Data

Language!

Written form

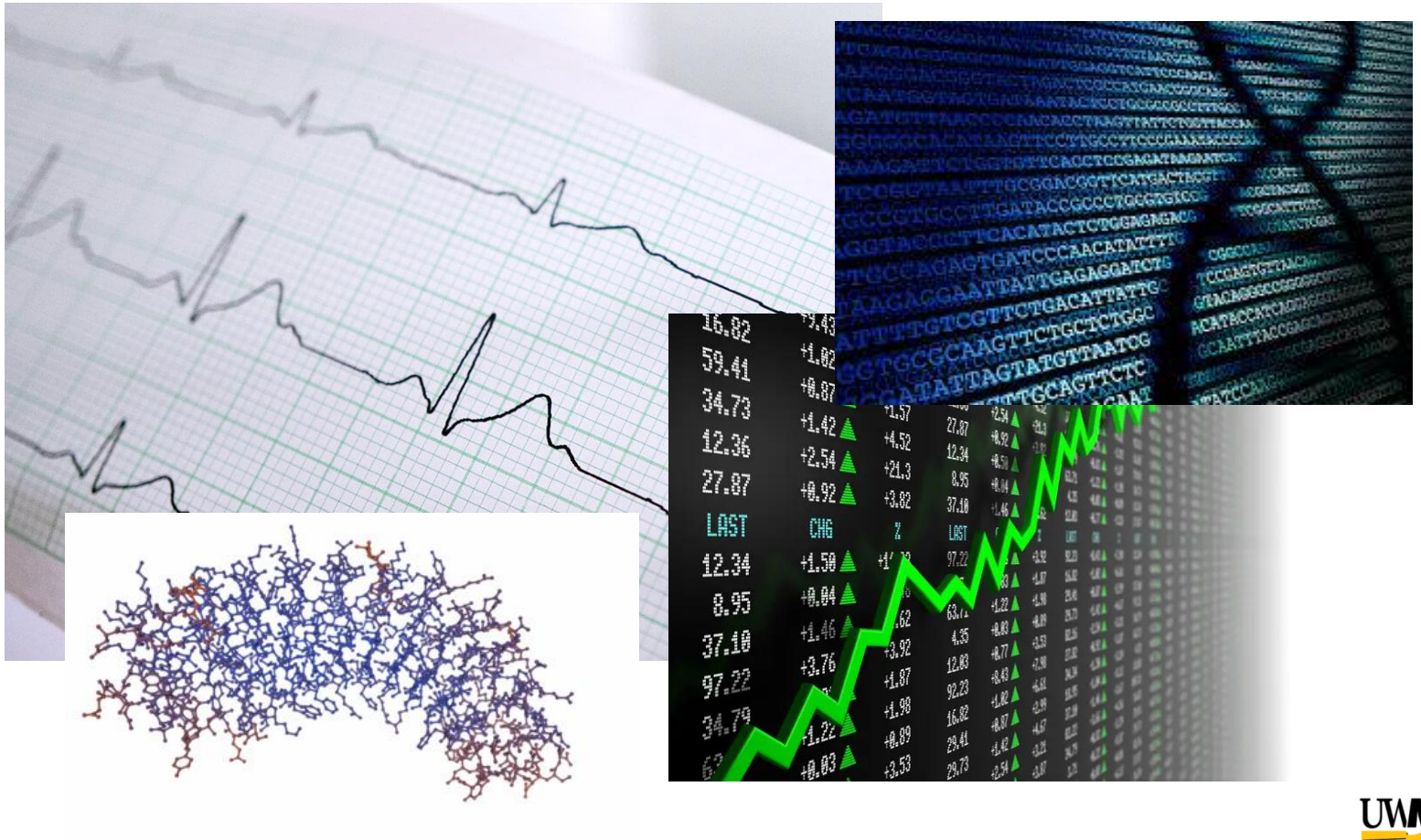
- **CIber CATSS 2025**

Characters: **2 0 2 5**

Words: **C I b e r C A T S S**

Sequence of words or letters

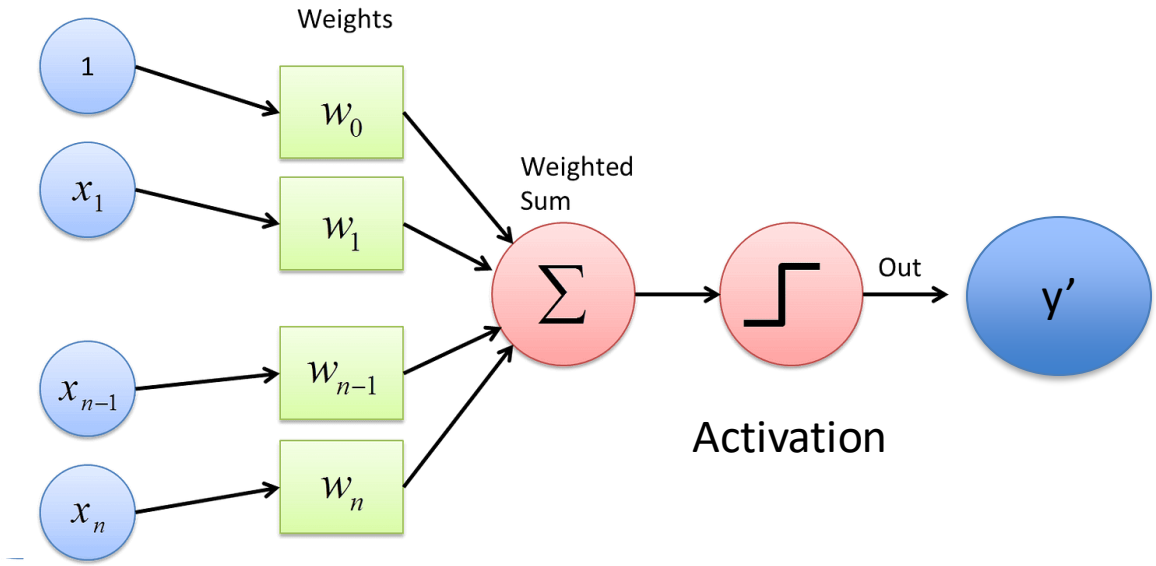
Sequential Data



This is a very rich and diverse type of data and class of problems we can work with

A quick review of NN

A Perceptron - revisit



A set of inputs

Weight matrix

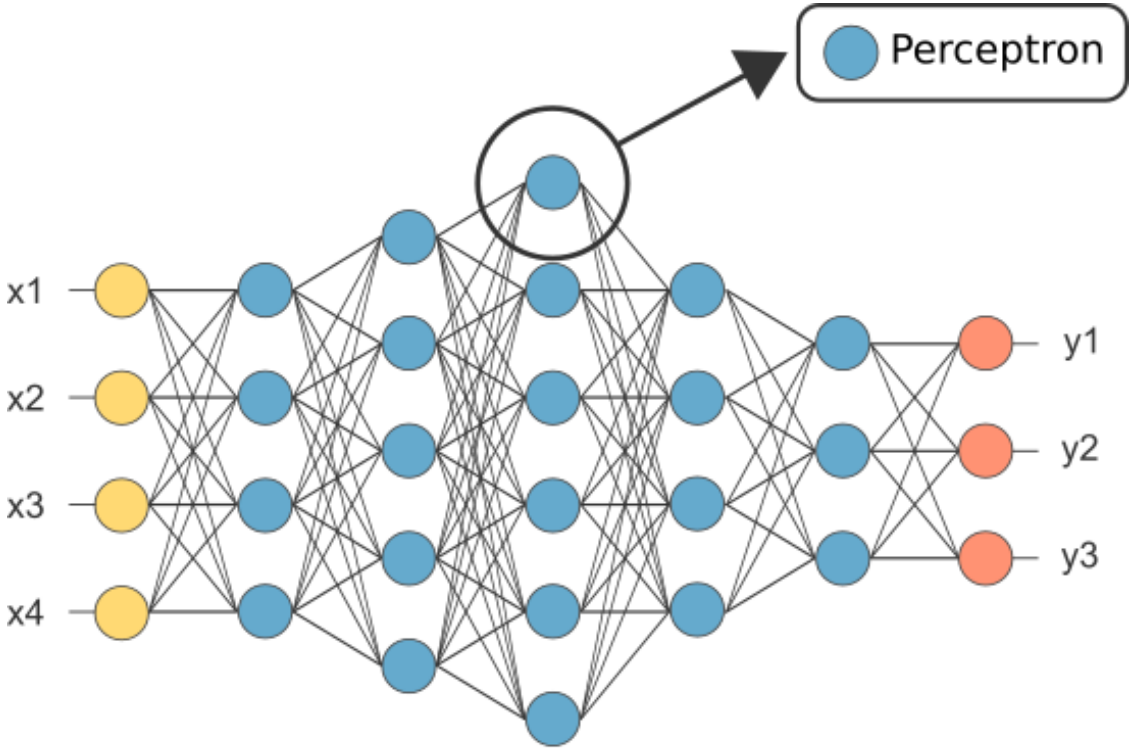
Linear combination

Applying non-linear activation function

Generating the output

A quick review of NN

Feed Forward Neural Networks



static single input
and
static single output

Image credit

A quick review of NN

Feed Forward Neural Networks

Feedforward Neural Networks process data in one direction, information flows straight from input to output without looking back or remembering anything.

This works well for tasks where each input is independent, like recognizing objects in pictures. But if the data has a sequence, like words in a sentence or time-series trends, feedforward networks can’t handle it well because they don’t have a memory to keep track of previous inputs.

Recurrent Neural Network

- A Recurrent Neural Network (**RNN**) is a type of neural network designed to work with sequential data by using memory of past inputs to inform current outputs.

- It's especially useful in applications like language modeling, time series prediction, and biological sequence analysis.

Recurrent Neural Network

- **Example**: Using past stock prices to predict tomorrow's

Recurrent Neural Network

Real-World Example (Biomedical)

In biomedical informatics, RNNs can be used for **predicting health events** from EHR time series:

x_t: patient vitals or lab results at time t,

y_t: probability of an adverse event (e.g., heart failure) occurring soon.

Recurrent Neural Network

Example in NLP:

Word “bank”

Consider these two sentences:

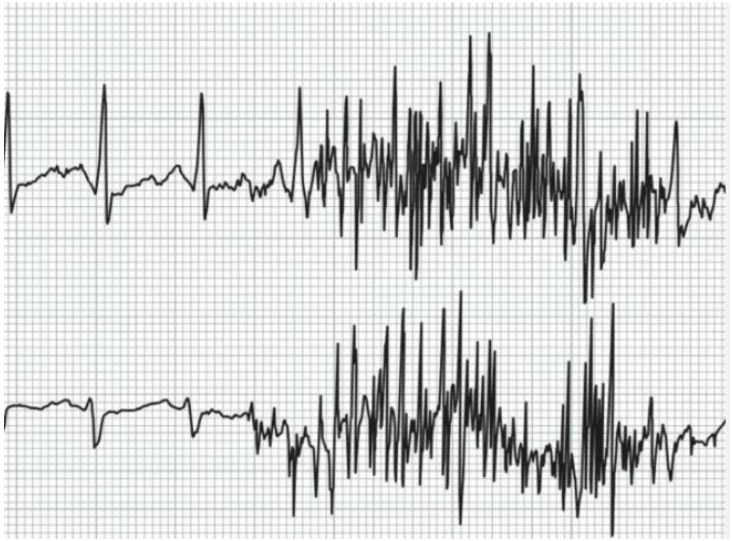
1."She went to the bank to deposit some money."
2."He sat by the bank and watched the river flow.”

In both cases, the word **"bank"** is the same **input**, but it means completely different things based on **context**.

Recurrent Neural Network

Example: Interpreting ECG Signals in Biomedical Engineering

Problem:
You're using an RNN to analyze **ECG (electrocardiogram) data**—a time series of electrical signals from the heart—to classify **arrhythmias** (irregular heartbeats).



Recurrent Neural Network

At a particular moment t, the signal pattern might look like a sudden spike or drop. But:

- If that spike is **preceded by a regular heartbeat pattern**, it might just be a **normal QRS complex**.
- If it's **preceded by irregular rhythm**, it might indicate a **ventricular fibrillation** (dangerous).

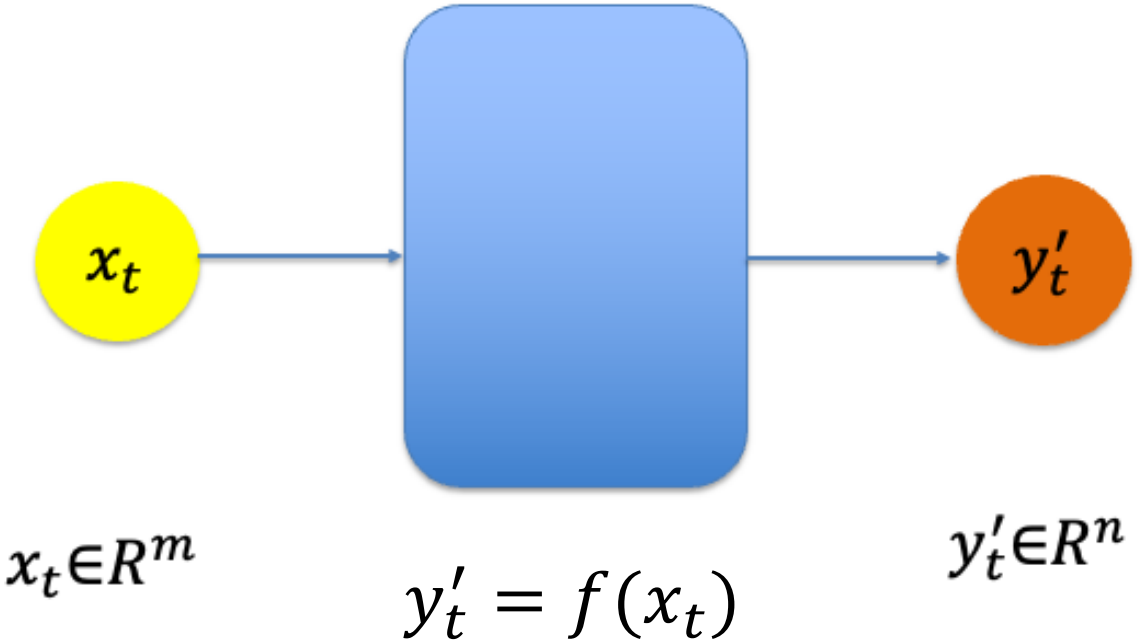
So, the **same electrical shape** in the ECG has **different meanings** depending on the **preceding context**.

Recurrent Neural Network

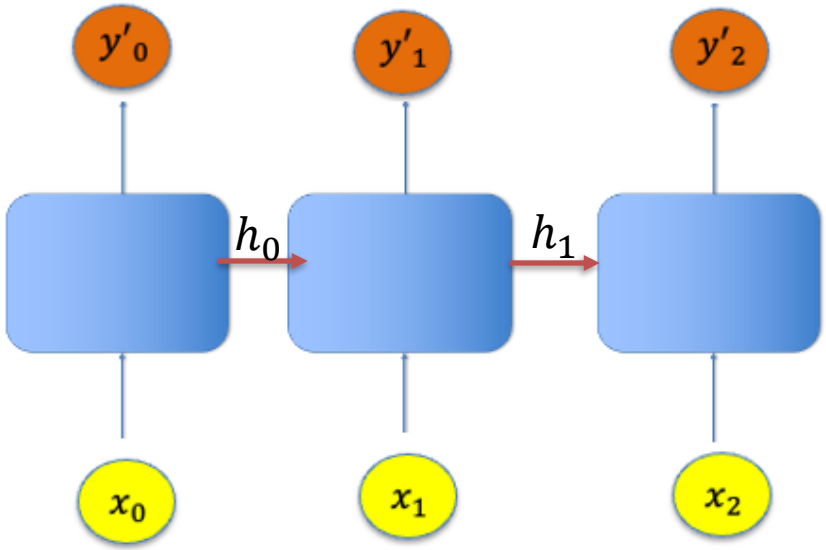
RNN's Role:

- RNN reads the signal **one timestep at a time**.
- The hidden state carries information about **what happened before** — like rhythm regularity, timing between beats, etc.
- When a new spike occurs, the model **uses context** to interpret whether it's normal or pathological.

Recurrent Neural Network



Recurrent Neural Network



$$y'_t = f(x_t, h_{t-1})$$

output input past memory

Hidden state is actually learned and computed by the neurons

Recurrent Neural Network

- Imagine you're reading a sentence **word by word**.
  As you read each word, your brain **remembers** what you've already seen, so you can understand the current word in context.

- In an RNN, this memory is stored in a **hidden state**, which is just a **vector of numbers** that keeps track of **important information from previous time steps**.

- This vector is updated at every step using a formula like:

$$h_t = \text{activation}(W \cdot x_t + U \cdot h_{t-1} + b)$$

where:

- x_t is the current input
- h_{t-1} is the memory from the previous step
- The weights W and U are learned during training

Recurrent Neural Network

Now that we've seen how the RNN updates its hidden state h_t at each time step, this slide shows how we generate the **final output y_t** from it.

The formula is:

$$y_t = g(W_y h_t + b_y)$$

Here's what each part means:

- h_t: This is the **hidden state**, which summarizes the memory up to time `t`.
- W_y: A **weight matrix** that learns how to map that memory into an output (e.g., a class label or prediction).
- b_y: A **bias term** added to adjust the output.
- g: An **activation function** like:
  - **Softmax** if you're doing classification (e.g., next word prediction),
  - or **Sigmoid** for binary classification.

Recurrent Neural Network

Key Observations:

- **Weight Sharing**:
  - W_h, W_x, W_y, b_h, and b_y remain the same for all t.
  - This ensures that the model applies the same transformation at each time step, capturing temporal dependencies consistently.
- **Temporal Dependencies**:
  - The model learns patterns like "what comes after this input" or "what the sequence should look like over time" by adjusting the shared weights during training.

Recurrent Neural Network

Advantages of Shared Weights

1. **Generalization Across Time Steps**:
    - The same weights ensure that the model treats each time step equivalently, which is essential for identifying patterns across the sequence.
2. **Reduced Complexity**:
    - The total number of parameters remains manageable, even for long sequences.
3. **Easier Training**:
    - Weight sharing reduces the risk of overfitting and simplifies backpropagation through time (BPTT), the algorithm used to train RNNs.

Recurrent Neural Network

RNN Intuition

```
my_rnn = RNN()
hidden_state = [0, 0, 0, 0]

sentence = ["I", "love", "recurrent", "neural"]

for word in sentence:
    prediction, hidden_state = my_rnn(word, hidden_state)

next_word_prediction = prediction
# >>> "networks!"
```

Introducing LSTMs: Tackling the Challenges of RNNs

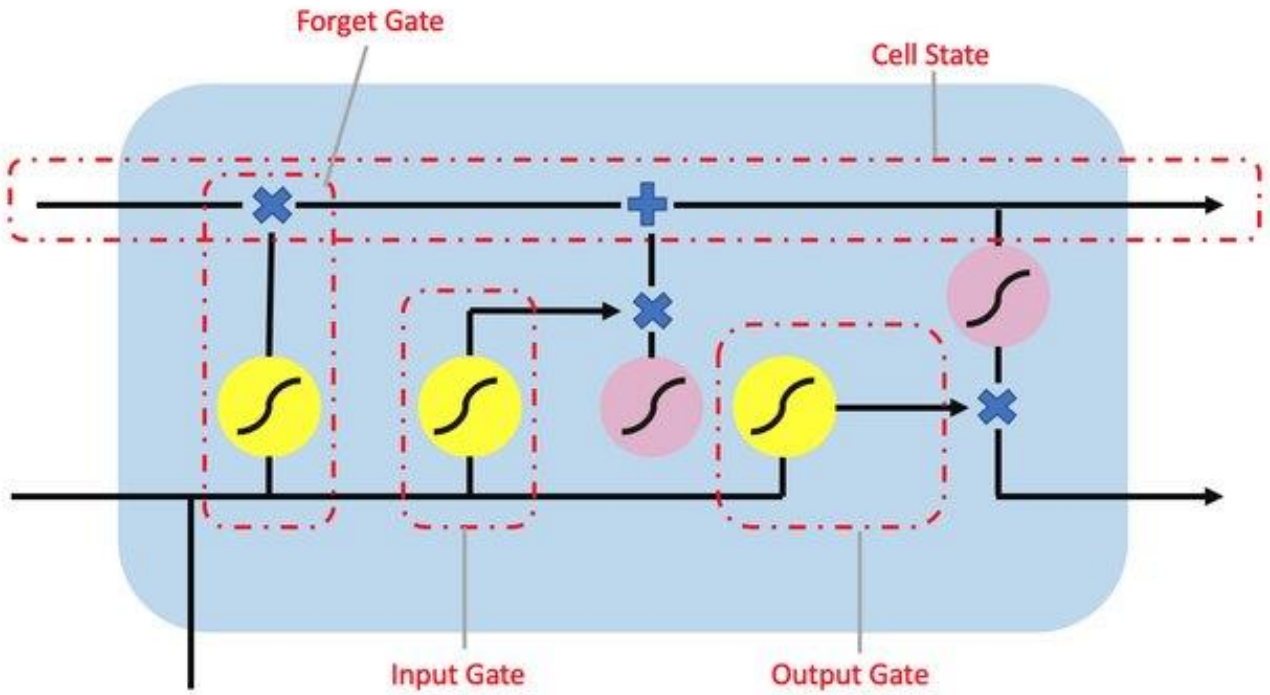
Long Short-Term Memory Networks

We need a more sophisticated mechanism to decide what information to keep, what to forget, and how to effectively store long-term context. This is where LSTMs come in.

LSTMs introduce special components (called **gated cells**) that allow the model to:

1. Decide what information to **remember**.
2. Decide what information to **forget** (irrelevant details).
3. Avoid the issues of vanishing and exploding gradients by carefully regulating memory flow.

LSTMs



Forget Gate (Left in the Diagram)

Decide which information to discard from the cell state.

Formula:

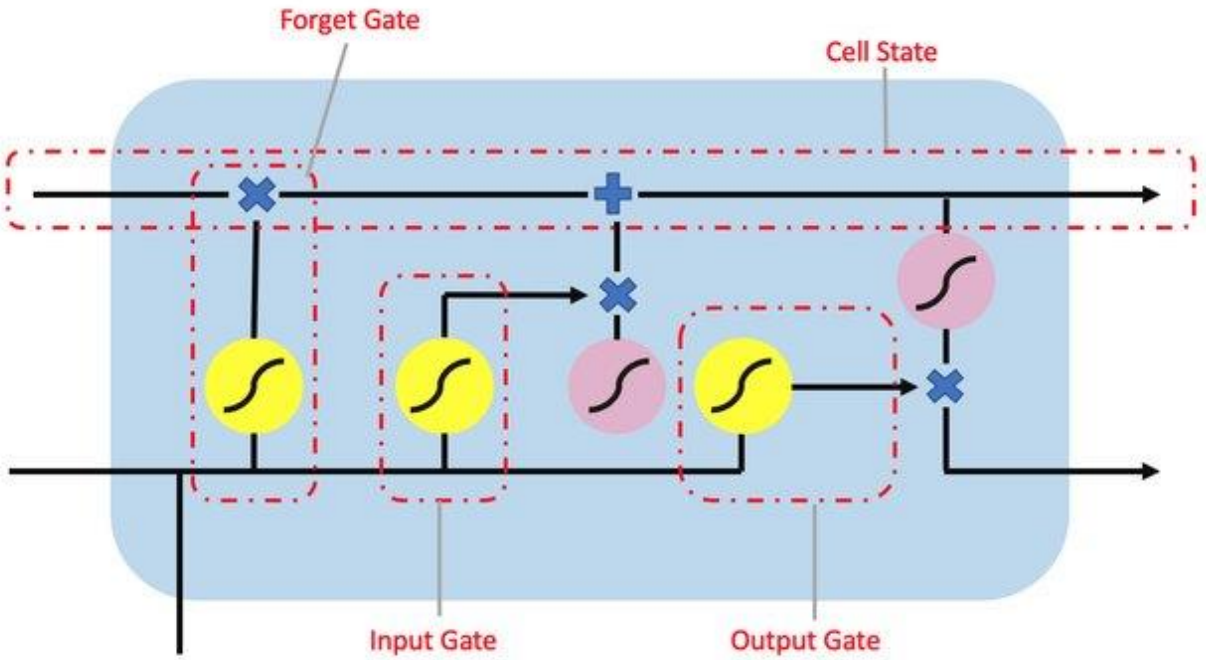
$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

- x_t: Current input.
- h_{t-1}: Previous hidden state.
- f_t: Forget gate output (values between 0 and 1).

Intuition: If $f_t \approx 1$, the information is retained; if $f_t \approx 0$, the information is discarded.

source

LSTMs



input Gate (middle in the diagram)

Decide what new information to add to the cell state.

Formula:

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

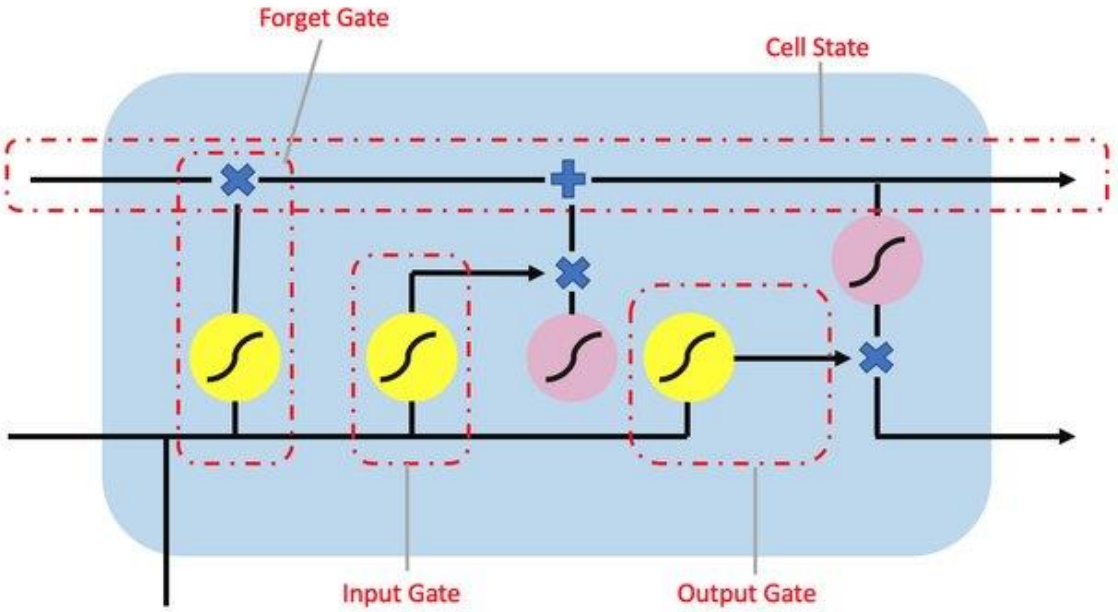
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

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

source

- i_t: Input gate output (values between 0 and 1).
- $\tilde{C}_t$: Candidate values to add to the cell state (scaled by i_t).

LSTMs



Output Gate (right in the diagram)

Decide what information to send to the hidden state and output.

Formula:

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

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

- o_t: Output gate output (values between 0 and 1).
- h_t: New hidden state, which is passed to the next timestep.

source

In the context of LSTMs, ff is an **activation function**, typically the **sigmoid function** (σ).

Practical Examples of LSTMs

1. Natural Language Processing (NLP)

a. Text Generation

- LSTMs are used to generate coherent text, word by word, or character by character.
- Example:
  - **Chatbots** that generate human-like responses.
  - **Story writing** and **poetry generation**.

b. Machine Translation

- LSTMs power models that translate text from one language to another.
- Example:
  - Translating "How are you?" (English) into "¿Cómo estás?" (Spanish).
  - Google Translate has historically used LSTM-based models.

Practical Examples of LSTMs

c. Sentiment Analysis

- LSTMs can analyze text data (e.g., customer reviews, social media posts) to predict sentiment (positive, negative, neutral).
- Example:
  - Classifying tweets as **happy** or **angry**.

d. Speech Recognition

- LSTMs process audio features to recognize and transcribe spoken words.
- Example:
  - **Voice assistants** like Siri and Alexa.

Practical Examples of LSTMs

2. Time-Series Analysis

a. Stock Price Prediction

- LSTMs predict future stock prices by analyzing historical data.
- Example:
  - Forecasting tomorrow's stock price based on the past year's prices.

b. Weather Forecasting

- LSTMs analyze past weather patterns (temperature, humidity) to predict future conditions.
- Example:
  - Predicting the next week’s temperature or rainfall.

c. Anomaly Detection

- LSTMs detect unusual patterns in sequential data.
- Example:
  - **Fraud detection** in financial transactions.
  - **Network traffic anomalies** in cybersecurity.

Practical Examples of LSTMs

3. Healthcare and Medicine

a. Patient Monitoring

- LSTMs analyze patient data like heart rate, blood pressure, or glucose levels over time to predict health issues.
- Example:
    - Detecting abnormal heart rhythms or seizures.

b. Disease Prediction

- Analyzing medical histories to predict diseases.
- Example:
    - Predicting the likelihood of diabetes based on time-series data of blood sugar levels.

Practical Examples of LSTMs

4. Audio and Video Processing

a. Music Generation

- LSTMs generate new music sequences by learning patterns in existing compositions.
- Example:
  - AI-generated piano or orchestral pieces.

b. Video Analysis

- LSTMs analyze video frames to recognize activities or events over time.
- Example:
  - **Surveillance systems** for detecting suspicious activities.

Practical Examples of LSTMs

5. Recommendation Systems

- LSTMs power personalized recommendations by analyzing a user's sequential behavior (e.g., clicks, views).
- Example:
  - Netflix recommending the next movie based on your viewing history.

6. Robotics and Control Systems

- LSTMs predict sequences of actions for robots to perform tasks.
- Example:
  - A robot learning to navigate through a warehouse based on sensor data over time.

Practical Examples of LSTMs

7. Finance

a. Risk Assessment

- Predicting risks in loan defaults or market crashes using historical data.
- Example:
    - Banking systems assessing credit risk based on payment history.

b. Portfolio Optimization

- Analyzing stock performance trends to allocate investments effectively.

Practical Examples of LSTMs

8. Other Notable Applications

a. Handwriting Recognition

- LSTMs recognize handwritten sequences, such as entire sentences.
- Example:
    - Digitizing scanned documents or signatures.

b. DNA Sequence Analysis

- LSTMs analyze genetic data to predict mutations or identify patterns.
- Example:
    - Understanding DNA sequences to detect diseases.

Transition from LSTMs to **Transformers**

LSTM Limitations

1. **Sequential Processing:**
    - LSTMs process data step-by-step (one timestep at a time), which limits parallelization during training.
    - This makes them slower for long sequences.
2. **Difficulty Capturing Long Dependencies:**
    - While LSTMs mitigate the vanishing gradient problem, they still struggle with very long-term dependencies because of the way information flows sequentially.
3. **Fixed Memory Size:**
    - LSTMs have a fixed memory size (hidden state), which limits their ability to store information about very long sequences.

Transition from LSTMs to **Transformers**

Why Transformers?

- **Transformers solve these issues by eliminating sequential processing.**
- They rely entirely on **self-attention** mechanisms, which process all timesteps simultaneously, enabling:
  - **Parallelization** for faster training.
  - **Better long-range dependencies.**

Transformers

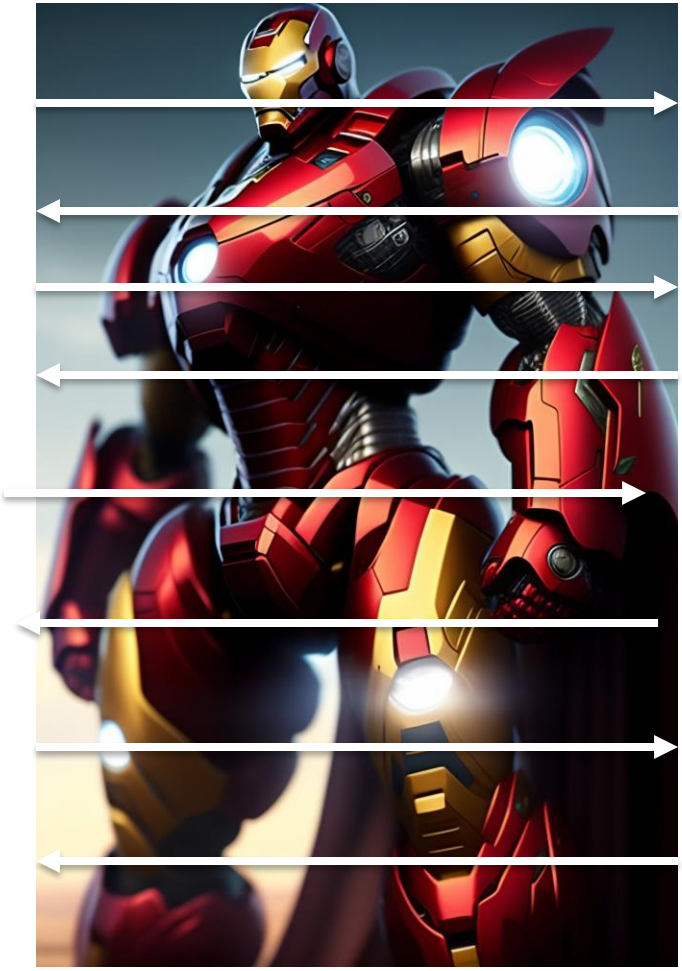
Attention Is All You Need

written by researchers at Google Research

This paper introduced the **Transformer architecture**, which has become the foundation for many modern AI models, including BERT, GPT, and ChatGPT. It was published in 2017.

The transformer is a type of neural network architecture that is built on Attention as its foundation mechanism.

Transformers

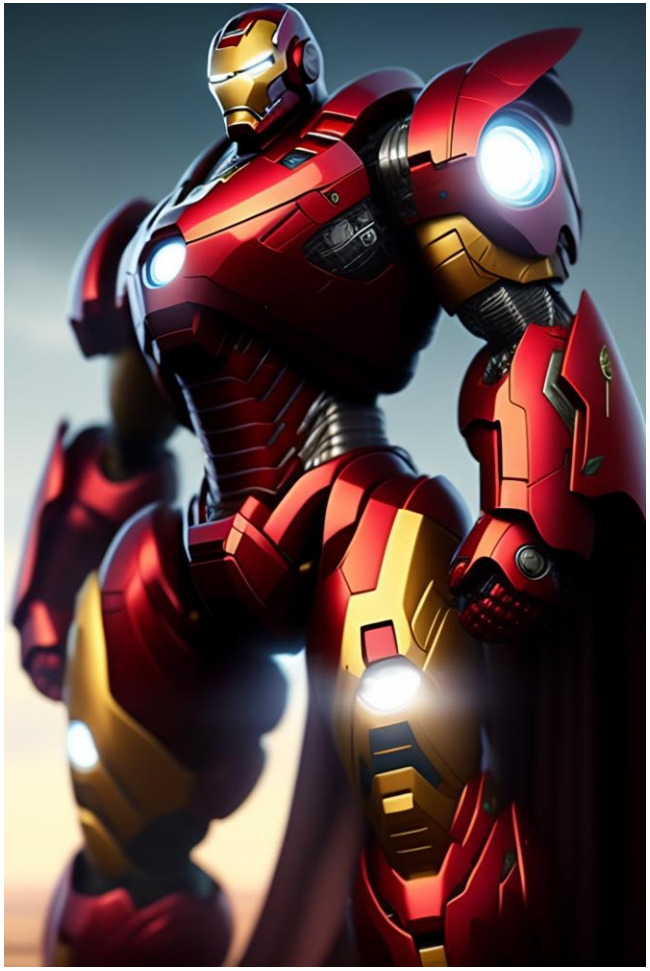


Question: How can we figure out what's important in this image?

The intuition is this idea of let's attend to and extract the most
Important part of the input!

We will ne focusing on self-attention (attending to the parts of the input itself)

Transformers



Similar to searching!

Our brains are immediately able to look at this and pick out, YES! Iron Man is important.

We can focus in and attend to that!

That's the intuition! Identifying which parts of an input to attend to and pulling out the associated features that has this HIGH attention score.

Transformers

Transformers Process All Timesteps at Once

Unlike LSTMs (or RNNs), which process sequences **one timestep at a time** (sequentially), Transformers use a mechanism called **self-attention** to look at the **entire sequence simultaneously**. This means:

- The Transformer can compare every word (or token) in the sequence with every other word, regardless of how far apart they are.
- It computes relationships between words (or tokens) across the whole sequence, ensuring that it captures **long-term dependencies** effectively.

Transformers

How Does It "Keep the Relevant Information"?

Transformers achieve this through **self-attention**.

1. Self-Attention Mechanism

- At each timestep, the Transformer calculates how much **attention** each word in the sequence should pay to every other word.
- Example:
  In the sentence **"The cat sat on the mat because it was warm,"**
  - The Transformer uses self-attention to determine that **"it"** refers to **"the mat"** by focusing more on **"mat"** than on irrelevant words like **"cat"** or **"sat."**

Transformers

2. How Self-Attention Works

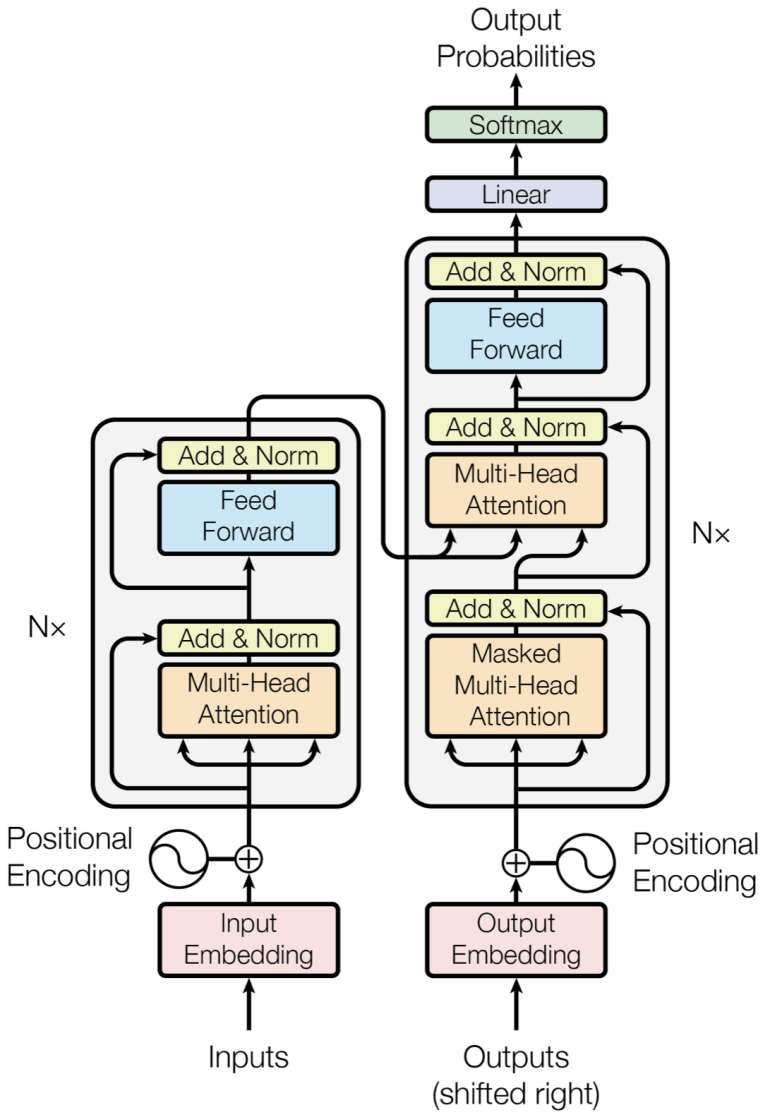
- Every word (or token) generates three vectors:
  - **Query (Q):** What am I looking for?
  - **Key (K):** What information do I have?
  - **Value (V):** What is the actual content I'm passing along?
- For each word, the model computes a **weighted sum of all the values** based on how similar the query and key vectors are.
  - Words with higher relevance to the query (determined by a score) are given higher weights.
  - This ensures that the model **focuses on relevant information** for each word.

Transformers

3. Capturing Long-Term Dependencies

- Because every word can attend to **all other words** in the sequence, Transformers can capture relationships between distant words.
- Example:
  In **"The cat ran because the dog barked,"** the word **"because"** connects **"ran"** (cat's action) to **"barked"** (dog's action), even though they are far apart in the sequence.

Transformers



Transformers

1. Encoder: Understanding the Input

- The **encoder** processes the **input** (e.g., a sentence) and converts it into a **meaningful representation** that captures relationships between words.
- Example:
  Input: "The cat sat on the mat."
  - The encoder figures out that "cat" is related to "sat" and "mat" but not to "the" directly.

Transformers

2. Decoder: Generating the Output

- The **decoder** takes the **encoder's output** (the understanding of the input) and generates the **output sequence** step by step.
- Example:
  Output: "El gato se sentó en la alfombra" (Spanish translation of "The cat sat on the mat").
  - The decoder aligns the encoded input ("cat") with the corresponding word in the output ("gato").

Transformers

What About Sequence Order?

Transformers don't process sequences step-by-step, so they don't inherently understand the **order** of the words. To solve this, they use **positional encodings**:

- Positional encodings add information about the position of each word in the sequence (e.g., first, second, third).
- This ensures the Transformer knows where each word is in the sequence.

Transformers

Why Is This Approach Powerful?

1. **Parallelization:**
    - Transformers process all timesteps at once, making them much faster to train than LSTMs.
2. **Focus on Relevance:**
    - Self-attention allows the model to ignore irrelevant information and focus only on important relationships in the sequence.
3. **Handles Long Sequences Better:**
    - Because self-attention looks at the entire sequence, Transformers capture long-term dependencies better than LSTMs, which struggle with very long sequences.