Ahmed Ibrahim

ECE 9039/9309 MACHINE LEARNING

Last Lecture

- Assignment #2 Orientation
- Autoencoders
  - Autoencoder Architecture
  - The Latent Space
  - Common Types of Autoencoders
- Hyperparameter Optimization (HPO)
- Into. to Recurrent Neural Network

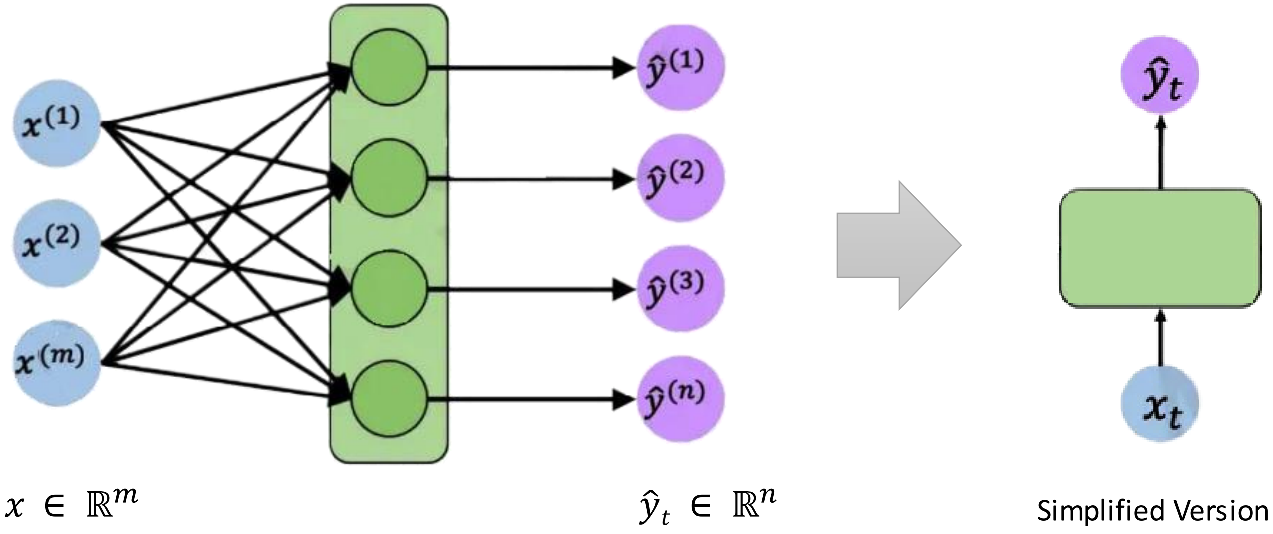
Outlines

- Recurrent Neural Network
- Long Short-Term Memory Networks
- Gated Recurrent Unit (GRU)
- Transformers

Recurrent Neural Networks (RNN)

RNN Review

- Let us consider the inputs (x_1,x_2,x_3) as (***location***, ***sq. ft***, and ***house type***) and outputs (y_1, y_2, y_3, y_4) as (***price***, ***safety status***, ***appraisal value***, ***# of people*** it can accommodate)
- i/p ex.: London, 1200, detached
- o/p ex.: 1000k, safe, 1200k, 5



Source: https://avs431.medium.com/explain-it-to-me-like-a-5-year-old-deep-sequence-modeling-introduction-to-recurrent-neural-beb2ee02bc6c

RNN Review

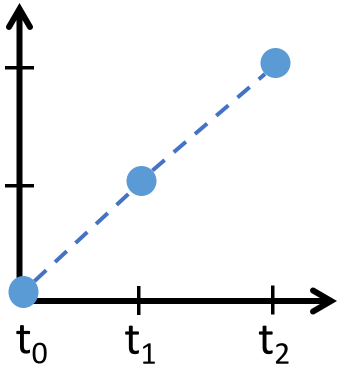
- Imagine placing several neural networks next to each other in a row, connected by a thin thread. This setup represents the continuous flow of information across a network of NNs. We refer to this thread as a "**hidden state**" (h).

$$h_2 = f(x_2, h_1)$$

Current state

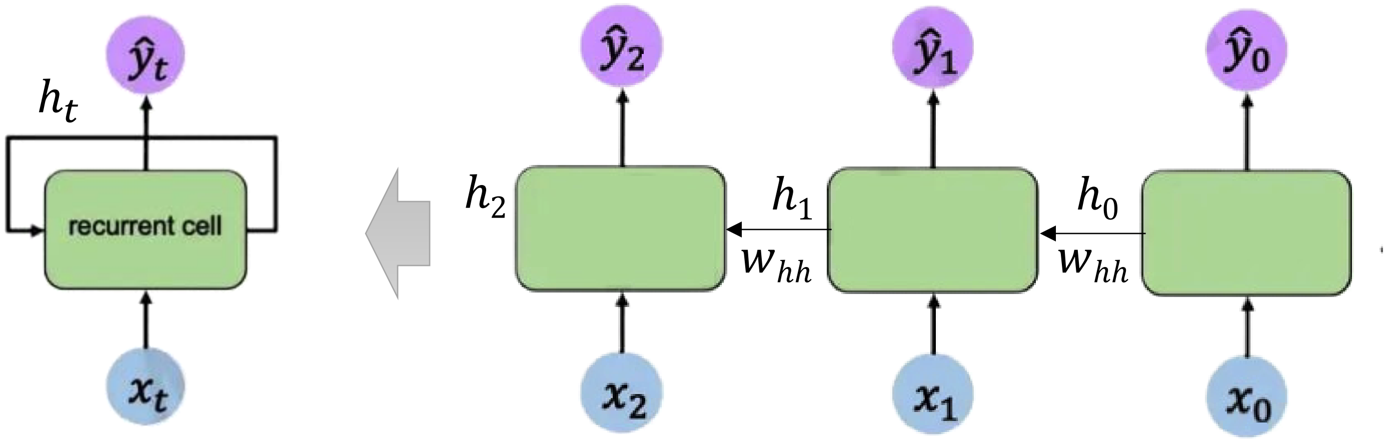
input

Past memory



$$\hat{y}_t = f(x_t, h_t)$$

Learning function

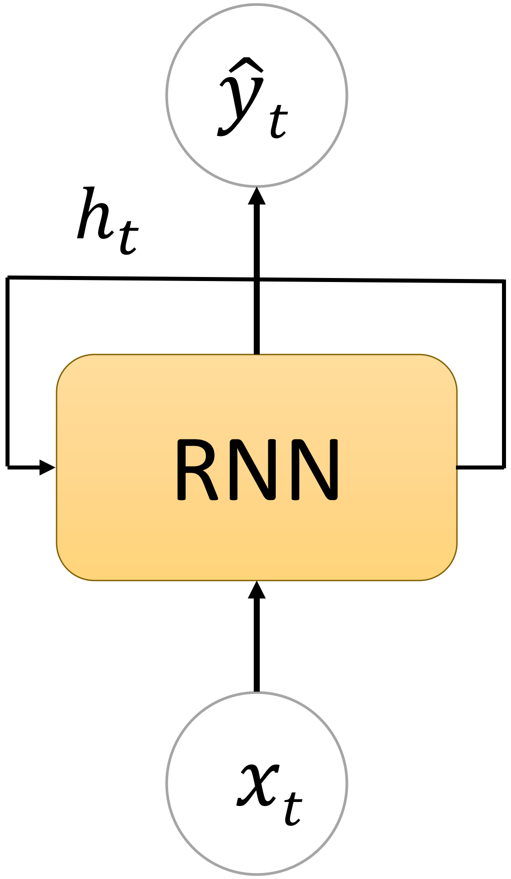


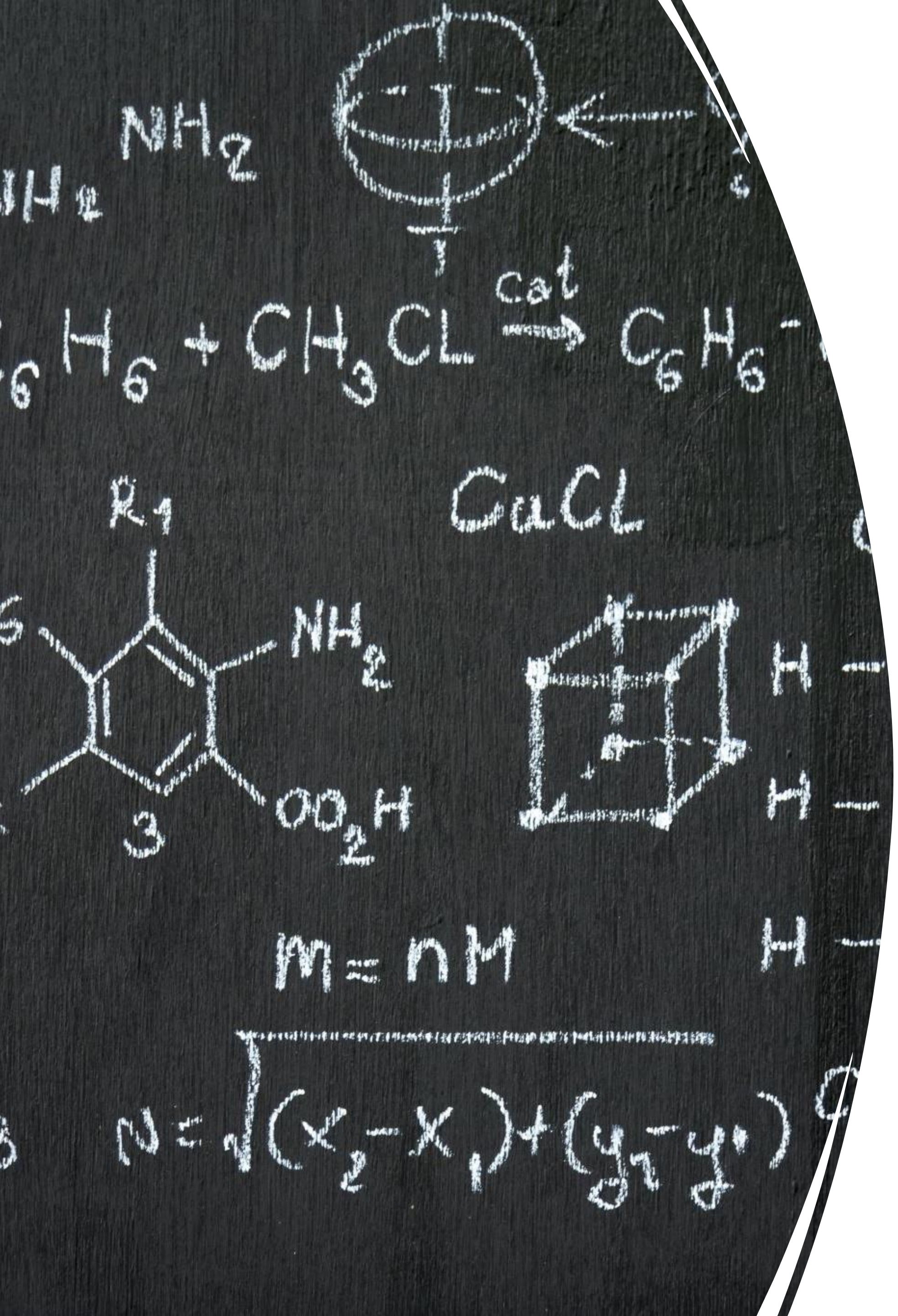
These models are copies from each other but with different inputs

Source: https://avs431.medium.com/explain-it-to-me-like-a-5-year-old-deep-sequence-modeling-introduction-to-recurrent-neural-beb2ee02bc6c

RNN Review

- **Recurrence** refers to the mechanism by which information loops back within the network. This loop enables the network to **retain the memory of previous inputs in its internal state, allowing it to process data sequences** (Sequential modeling).
- This characteristic is beneficial for tasks that require understanding temporal or sequential dependencies, such as **language modeling**, **time series prediction**, and **speech recognition**.
- Recurrence enables the network to make decisions based on current and past information.



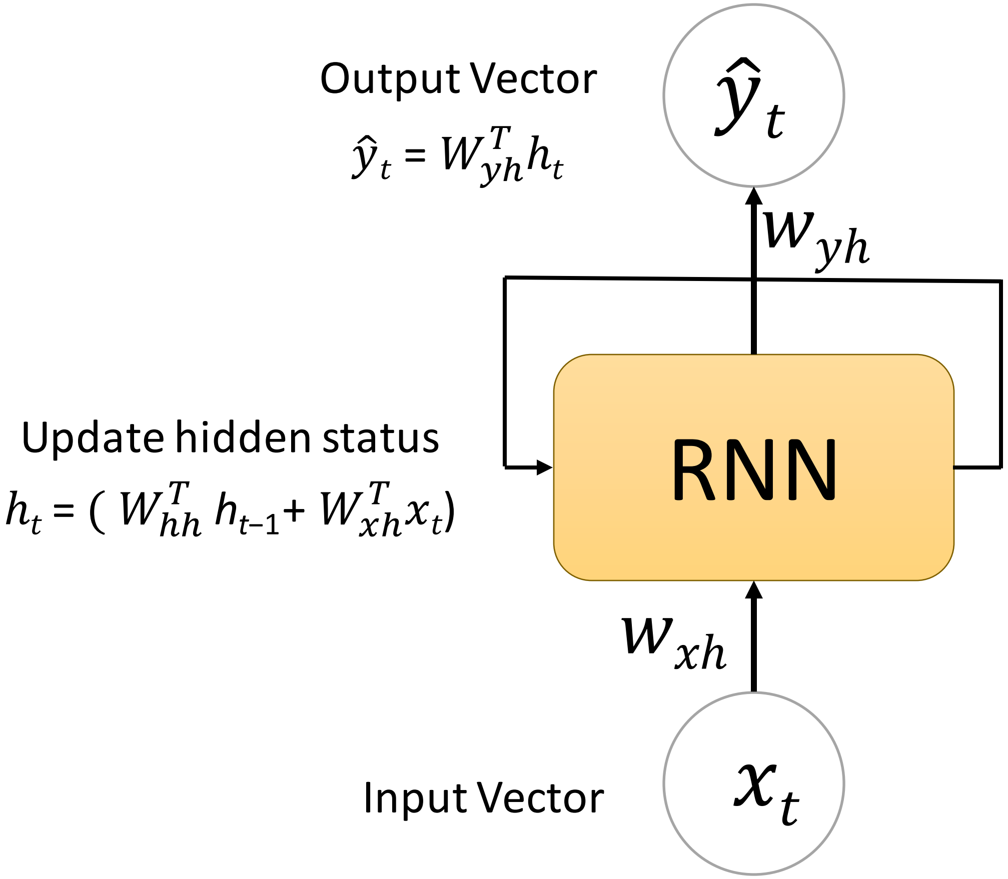


Word Prediction Using an RNN

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

sentence = ["Apple", "is", "my", "favorite",
"fruit"]

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

next_word_prediction = prediction
```

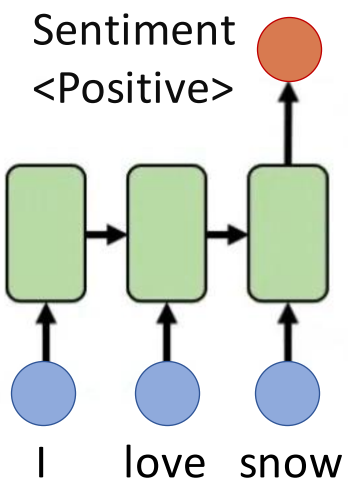
RNN Status Update

- Input Vector (x_t): This is the input to the RNN at a time step 't'.
- Weight Matrix (w_{xh}) weights are applied to the input vector.
- Hidden State (h_t): This is the memory component of the RNN, which gets updated at every time step.
- Output Vector ($\hat{y}_t$): This is the output of the RNN at time step 't'. It is calculated by applying another weight matrix (w_{yh}) to the hidden state (h_t).

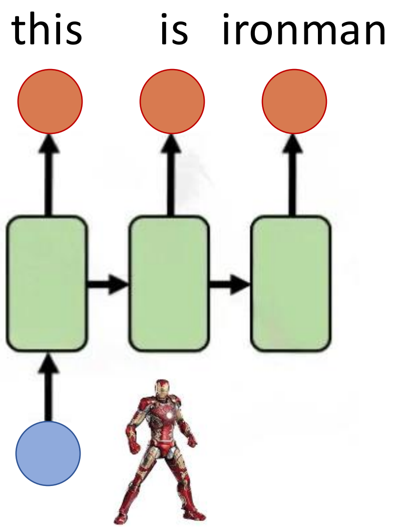


The diagram shows the update rule for the hidden state.

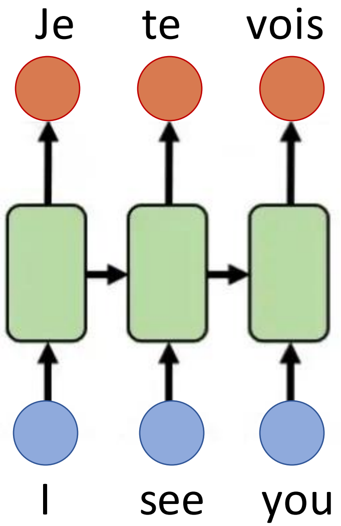
RNN for Sequence Modeling



- A model takes in numerous inputs and produces a single output value, such as **sentiment classification**, where the model processes a sequence of words and outputs a singular sentiment assessment.



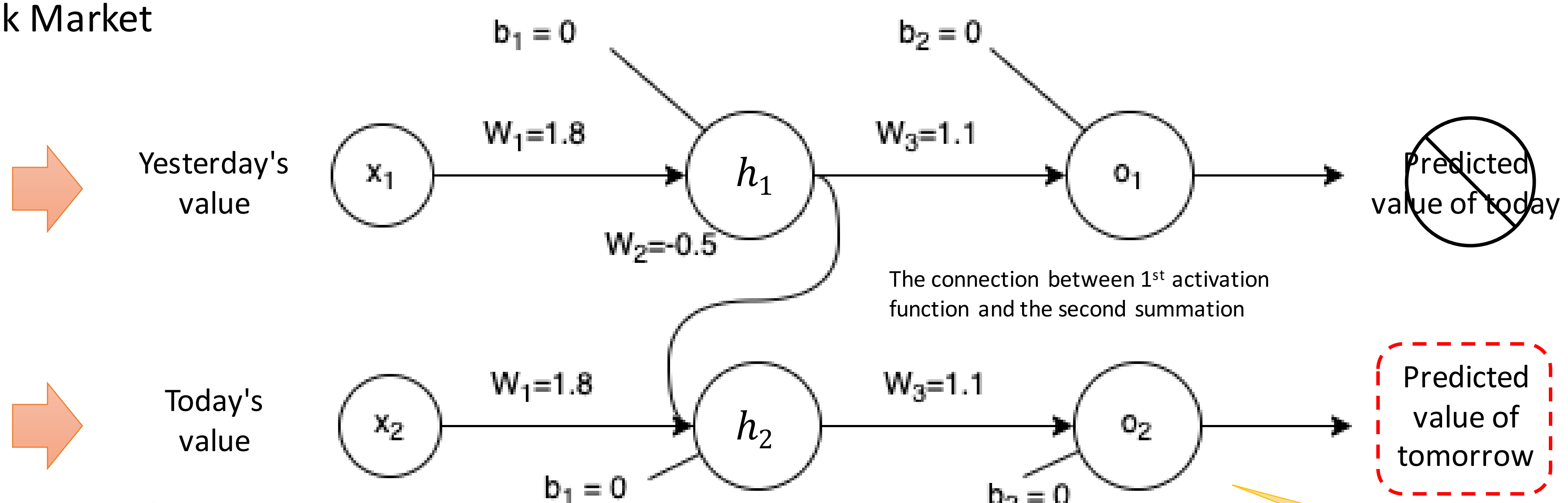
- A model accepts a single input and generates outputs of varying lengths, for instance, in **image captioning**, where the model is given <u>one image</u> and produces a descriptive caption detailing the contents of the image.

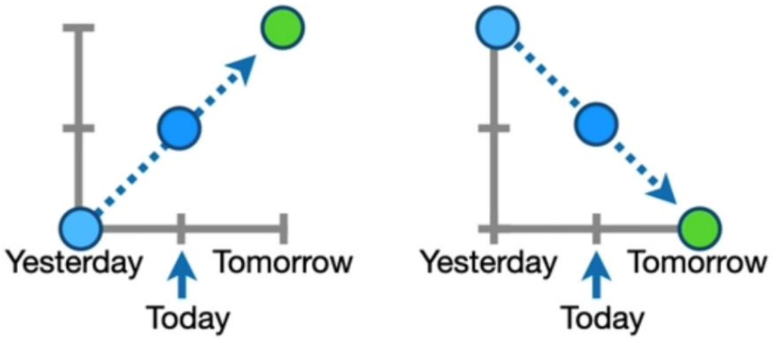


- A model handles multiple inputs and delivers outputs of flexible lengths, such as in **machine translation**, where the model receives a sentence in English and translates it into French, providing a variable-length output based on the input sentence.

Recall: The RNN Cell

- Example: Stock Market

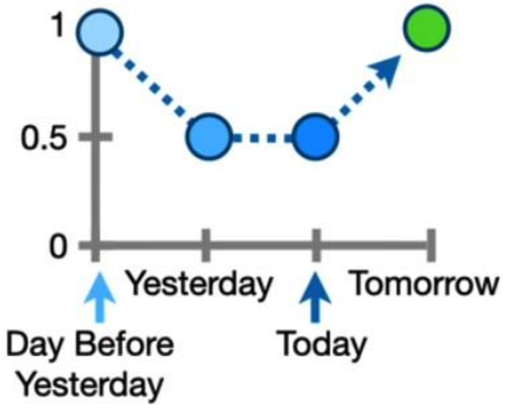


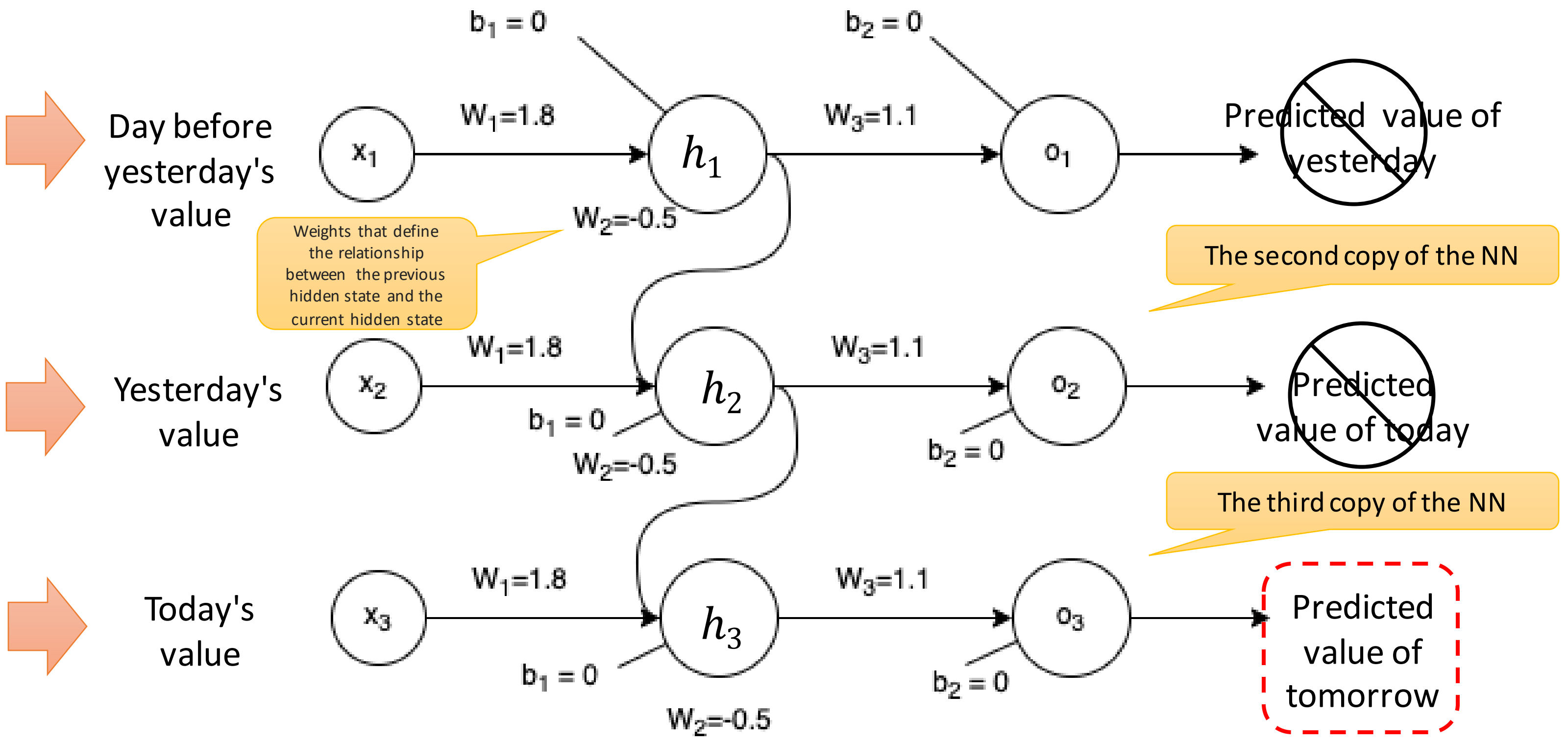


- This recurrent neural network performs great with 2 days' worth of data

Recall: The RNN Cell (cont.)

- This recurrent neural network performs great with 3 days' worth of data





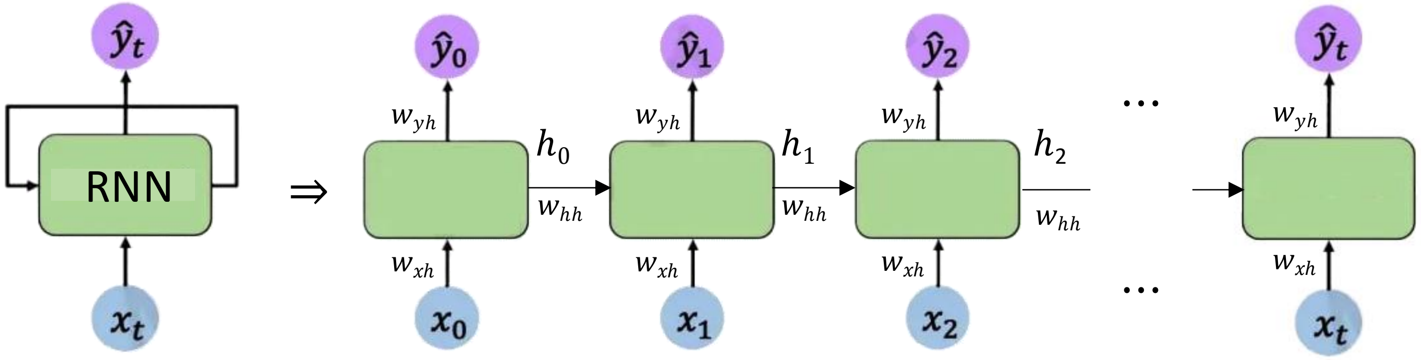
Weights that define the relationship between the previous hidden state and the current hidden state

The second copy of the NN

The third copy of the NN

Vanilla RNN Architecture

- Handling Variable-Length Sequences
- Memory Capability
- Parameter Sharing Across Time
- Applicability to a Wide Range of Tasks

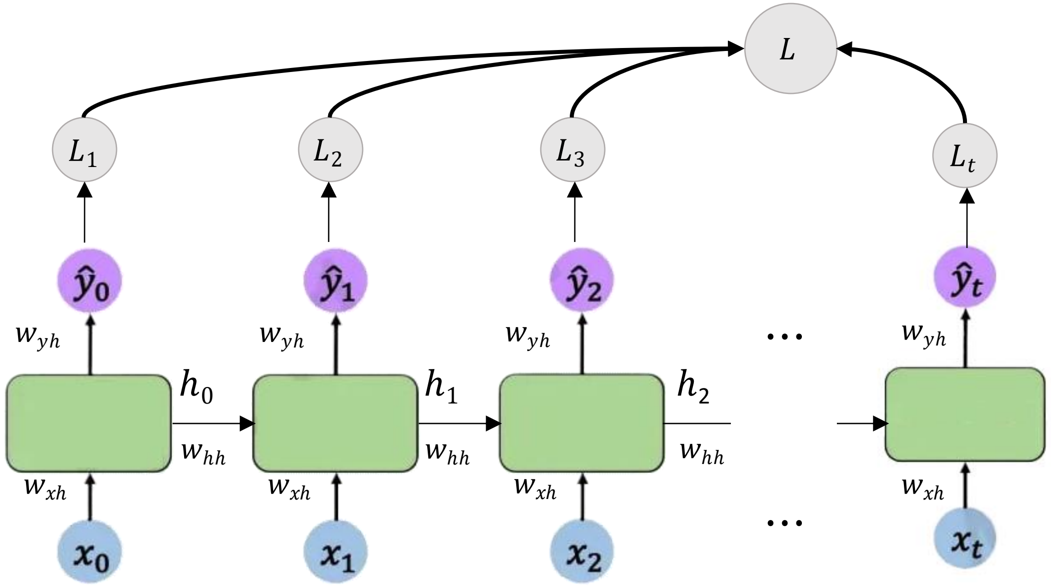


Unrolled

Re-use the same weight metrics at every time step

Source: https://avs431.medium.com/explain-it-to-me-like-a-5-year-old-deep-sequence-modeling-introduction-to-recurrent-neural-beb2ee02bc6c

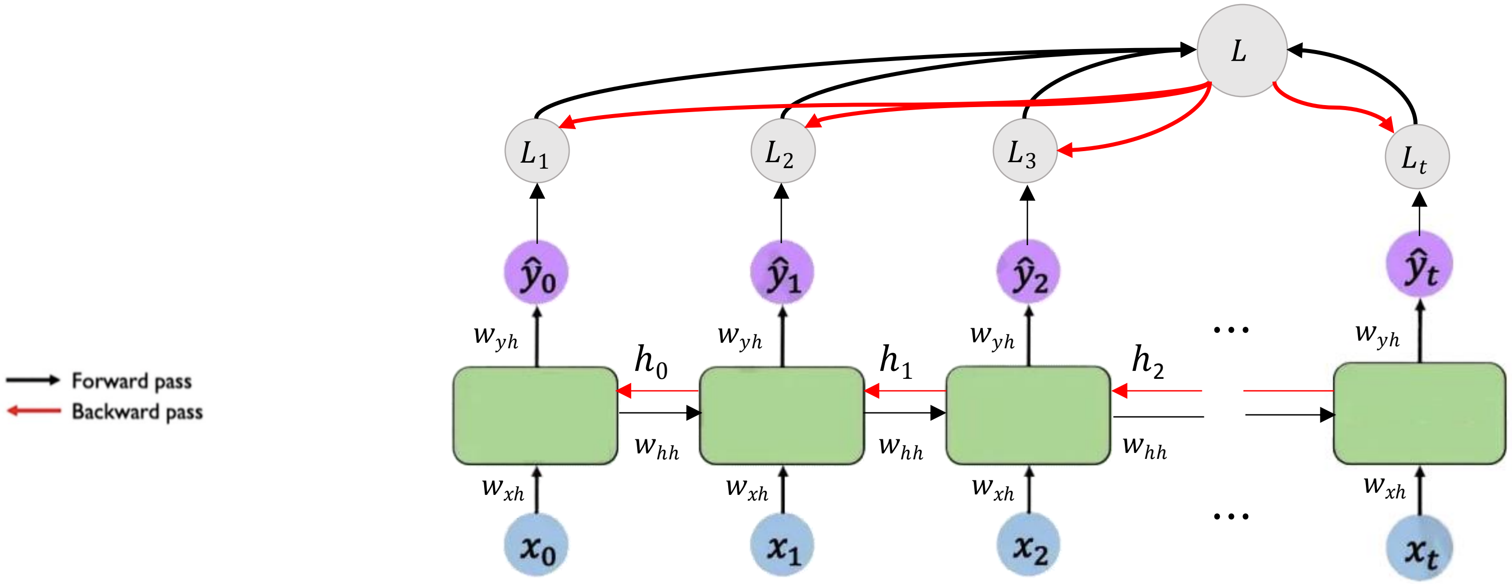
RNN Training: Forward Pass



- It is like a Multilayer Perceptron (MLP), with the distinction that the activations reaching the hidden layer originate from the current input, and the activation of the hidden layer reflects information from one timestep prior.

Source: https://avs431.medium.com/explain-it-to-me-like-a-5-year-old-deep-sequence-modeling-introduction-to-recurrent-neural-beb2ee02bc6c

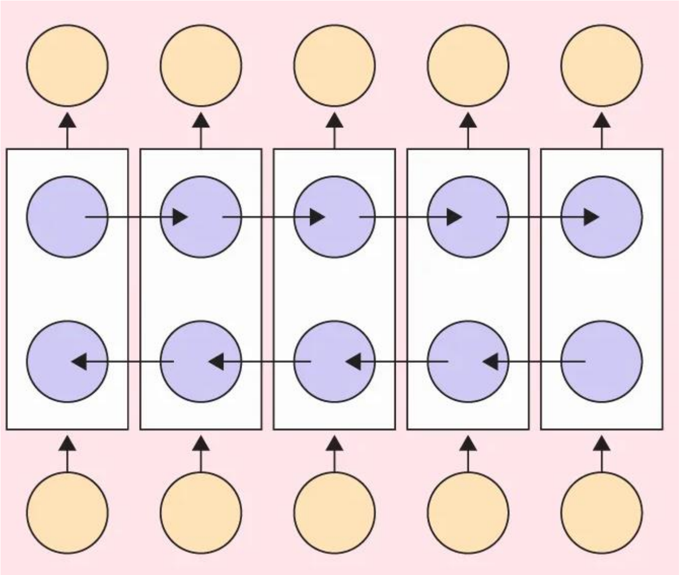
RNN Training: Backward Pass



- The backward pass indicates the flow of gradients during the training phase.
- Gradients of the loss are propagated back through the network to update the weights.
- The gradients are propagated back not only to the output weights w_{yh} but also back through the hidden states and their respective weights w_{hh} and w_{xh}, adjusting them to reduce the loss in future iterations.

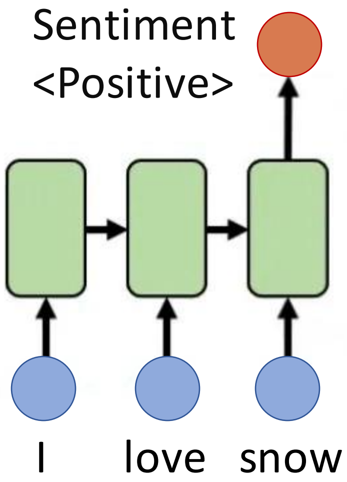
Bi-directional RNN

- Bidirectional Recurrent Neural Networks (Bi-RNNs) are a variation of the Vanilla RNNs designed to improve the model's ability to understand the context from the **past** and the **future** of a given point in the sequence.
- In a Bi-RNN, the input data is passed through two **separate RNNs**: one processes the data in the forward direction, while the other processes it in the reverse direction.
- The outputs of these two RNNs are then combined in some way to produce the final output.
- Consider an example where we could use BiRNN:
  - Apple is my favorite _____.
  - Apple is my favorite _____, and I work there.
  - Apple is my favorite _____, and I am going to buy one.
- The answer could be **fruit**, **company**, or **phone** in the first sentence. But it can not be a fruit in the second and third sentences.

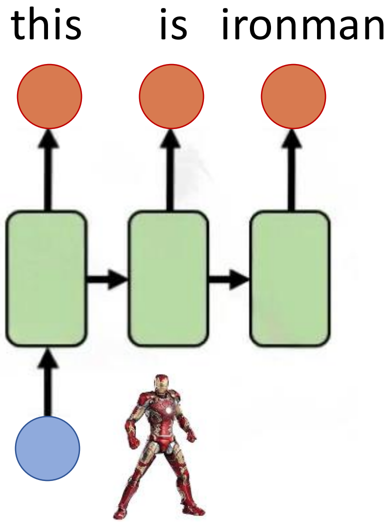


Sequence Modeling

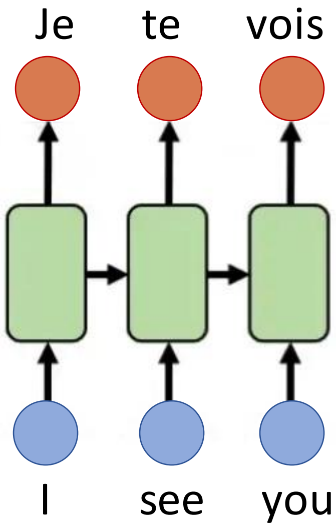
Recall: RNN for Sequence Modeling



- A model takes in numerous inputs and produces a single output value, such as **sentiment classification**, where the model processes a sequence of words and outputs a singular sentiment assessment.



- A model accepts a single input and generates outputs of varying lengths, for instance, in **image captioning**, where the model is given one image and produces a descriptive caption detailing the contents of the image.



- A model handles multiple inputs and delivers outputs of flexible lengths, such as in **machine translation**, where the model receives a sentence in English and translates it into French, providing a variable-length output based on the input sentence.

Sequence Modeling: Design Criteria

- **To model a sequence, we need to handle:**

✓ Handle Variable Sequence Lengths

- "The food was great"
- "We visited a restaurant for lunch"
- "We were hungry but cleaned the house before eating"

✓ Model Long-Term dependencies

"I grew up in the Middle East but now live in London. I speak fluent _????_."

- We need information from the distant past to accurately predict the correct word.

✓ Capture differences in sequence order

"The food was **good**, not **bad** at all."

"The food was **bad**, not **good** at all."

Recurrent Neural Networks (RNNs) meet these sequence modeling design criteria

Source: MIT 6.5191

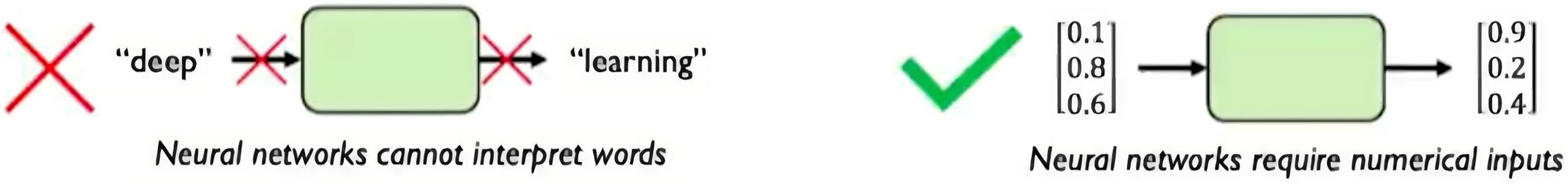
Representing Language to NN

- RNNs can't directly understand raw data like words or sounds. **Encoding** is the step that changes this raw data into a clear format that RNNs can work with.

"Today I took my kids to school."

given these words

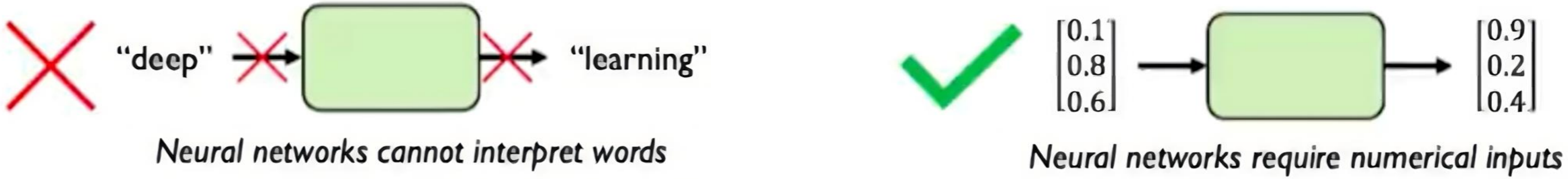
Predict the next word



"deep"
"learning"
Neural networks cannot interpret words
$\begin{bmatrix}0.1\\0.8\\0.6\end{bmatrix}$
$\begin{bmatrix}0.9\\0.2\\0.4\end{bmatrix}$
Neural networks require numerical inputs

Source: MIT 6.5191

Encoding Language for a NN

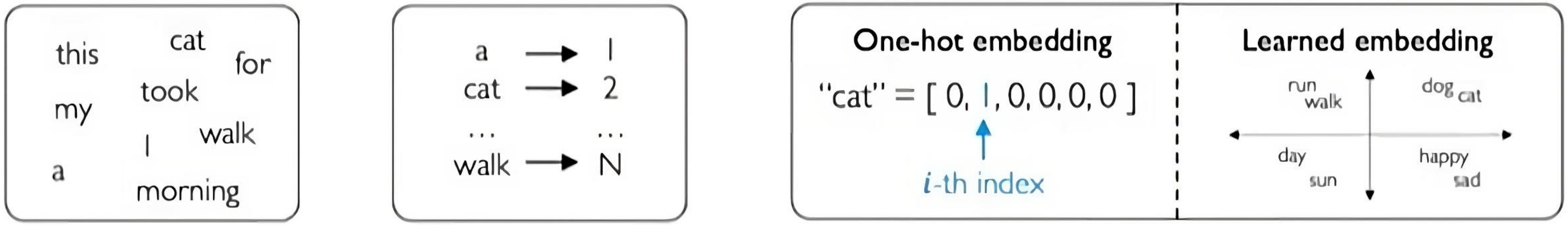


Neural networks cannot interpret words

Neural networks require numerical inputs

- Encoding words (**embeddings**) involves representing them as vectors in a high-dimensional space, where each dimension captures some aspect of the word's meaning.

Embedding: transform indexes into a vector of fixed size.



1. Vocabulary:
Corpus of words

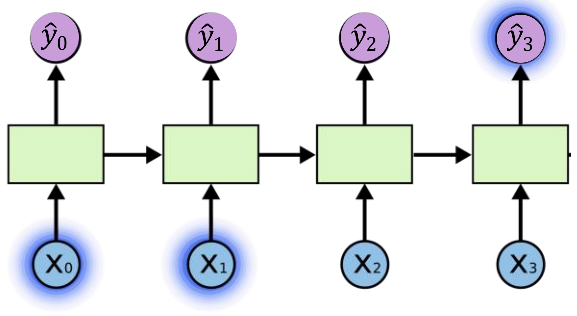
2. Indexing:
Word to index

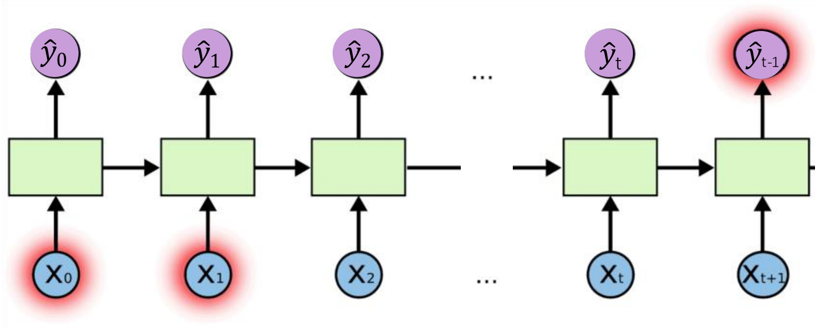
3. Embedding:
Index to fixed-sized vector

Source: MIT 6.5191

RNN Limitations

- **Encoding bottleneck** - RNNs compress all the necessary information of the input sequence into its hidden states, which can lead to a loss of information, particularly with longer sequences.
- **Not long memory** - Traditional RNNs struggle with remembering information from early in the input sequence, which is necessary for making predictions later in the sequence (this is known as the **vanishing gradient problem**).
- **Slow, no parallelization** - RNNs process data sequentially, which means each step depends on the computations of the previous step.

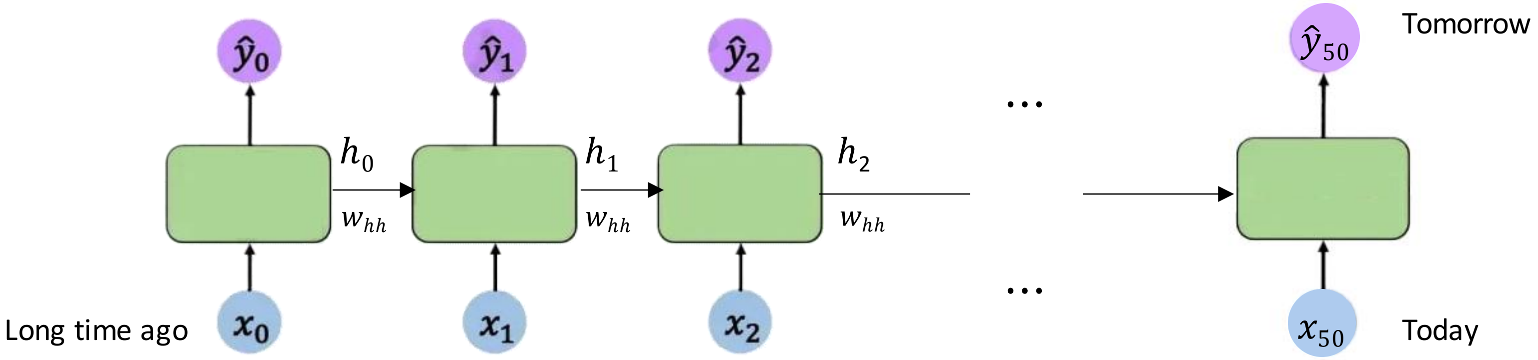




Source: https://cv-tricks.com/rnn/getting-started-recurrent-neural-networks/

Issues with the Vanilla RNNs: Gradient **Vanishing**

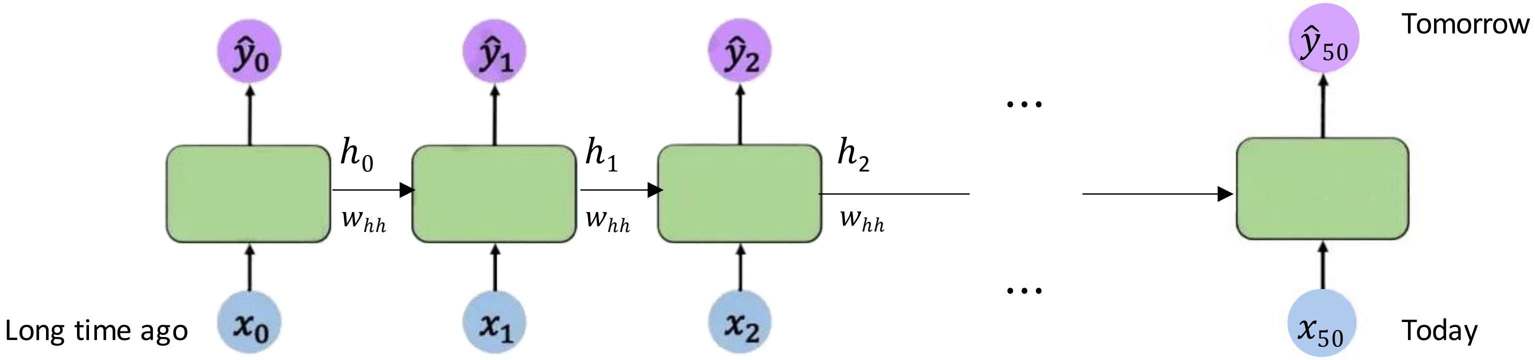
- Imagine setting the weight W_{hh} to 0.5 and having a sequence of 50 data points. If we unroll the network for each point, each input gets multiplied repeatedly by 0.5 across the 50-time steps, equivalent to raising 0.5 to the power of 50. This results in an extremely small number of **8.88×10^{-16}**.
- An extremely small gradient can contribute to the vanishing gradient problem, where the gradient becomes so small that it provides almost no learning signal to layers in the network.



- Solution: introducing **gated cells** to selectively add/remove information within each recurrent unit (LSTM/GRU, etc.)

Issues with the Vanilla RNNs: **Gradient Exploding**

- Imagine the weight W_{hh}= 2
- Let’s say we have 50 sequential data points. Then, we would unroll the network 50 times (as illustrated). We’ll end up multiplying the input by a huge number => 2^{50} is equal to is equal to 1,125,899,906,842,624.
- This huge number causes the gradient in the gradient descent calculation to explode.
- exploding gradients → learning diverges

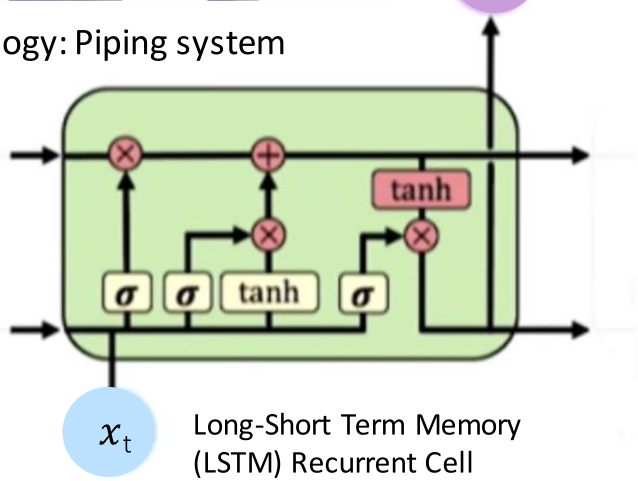


- Solution: clip the gradients to a certain max value or limit weights to less than 1.

Gated Cells

- Gates are mechanisms within each recurrent unit that **regulate** the flow of information. They decide what should be **retained** or **discarded** as the network processes each piece of information.
- RNNs often struggle with long-term dependencies in which information's influence weakens as it passes through each step.
- **Gated RNNs** such as Long Short-Term Memory (LSTM) networks and Gated Recurrent Units (GRUs) were developed to overcome these issues. These advanced RNNs use gates to selectively remember and forget information.

Analogy: Piping system



Long-Short Term Memory (LSTM) Recurrent Cell

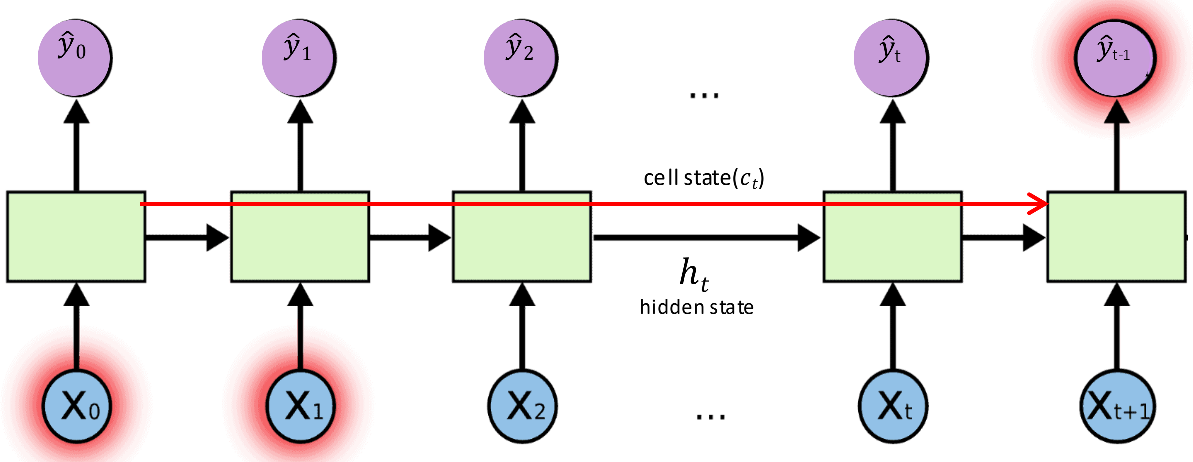
Attendance

You can use the provided link if you don't have a cell phone or if your phone lacks a QR-Code reader.

Long Short Term Memory (LSTM)

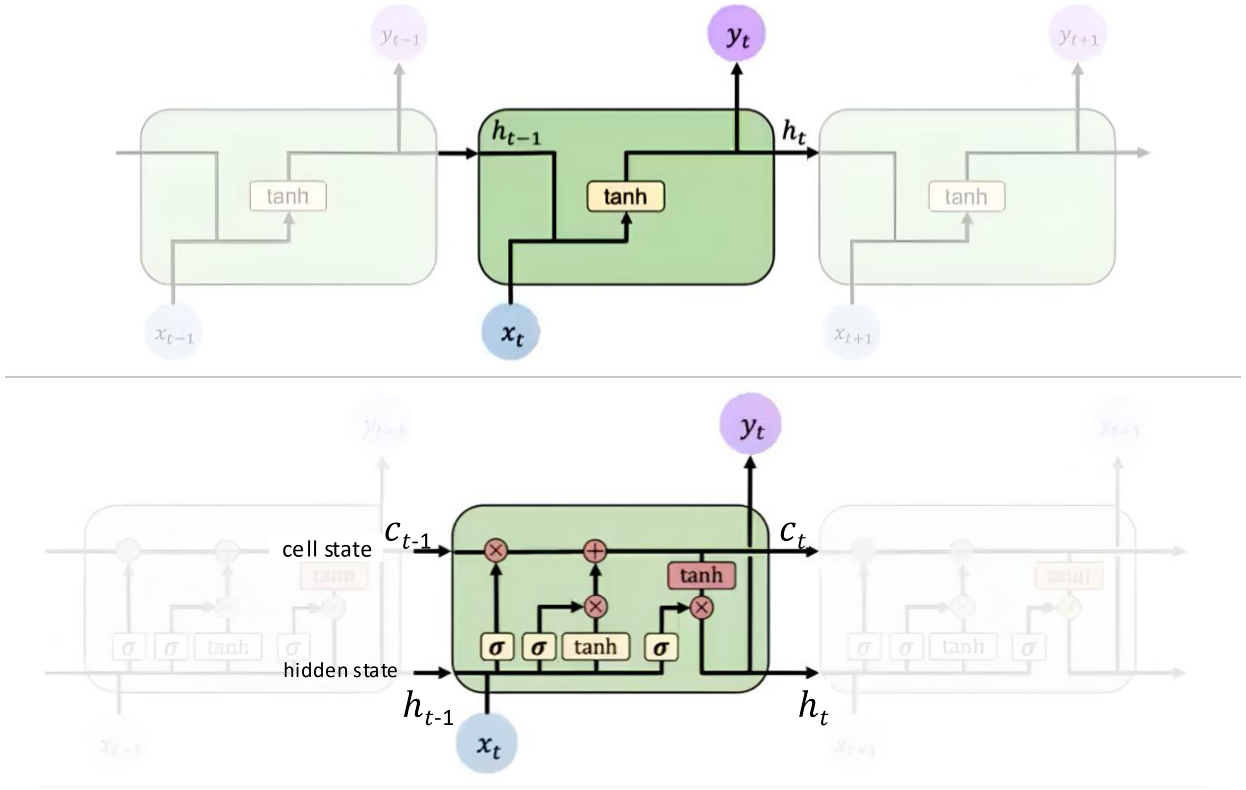
Long Short Term Memory (LSTM)

- LSTM, or Long-Short-Term Memory, is a variant of RNN (Recurrent Neural Networks) that has been groundbreaking in various computer science domains due to its ability to avoid the challenges of "**long-term dependencies**."
- LSTMs distinguish themselves through a unique architecture in which each unit (or module) incorporates several layers and gates. These components are essential in managing the module's state.



Conventional RNN vs. LSTM

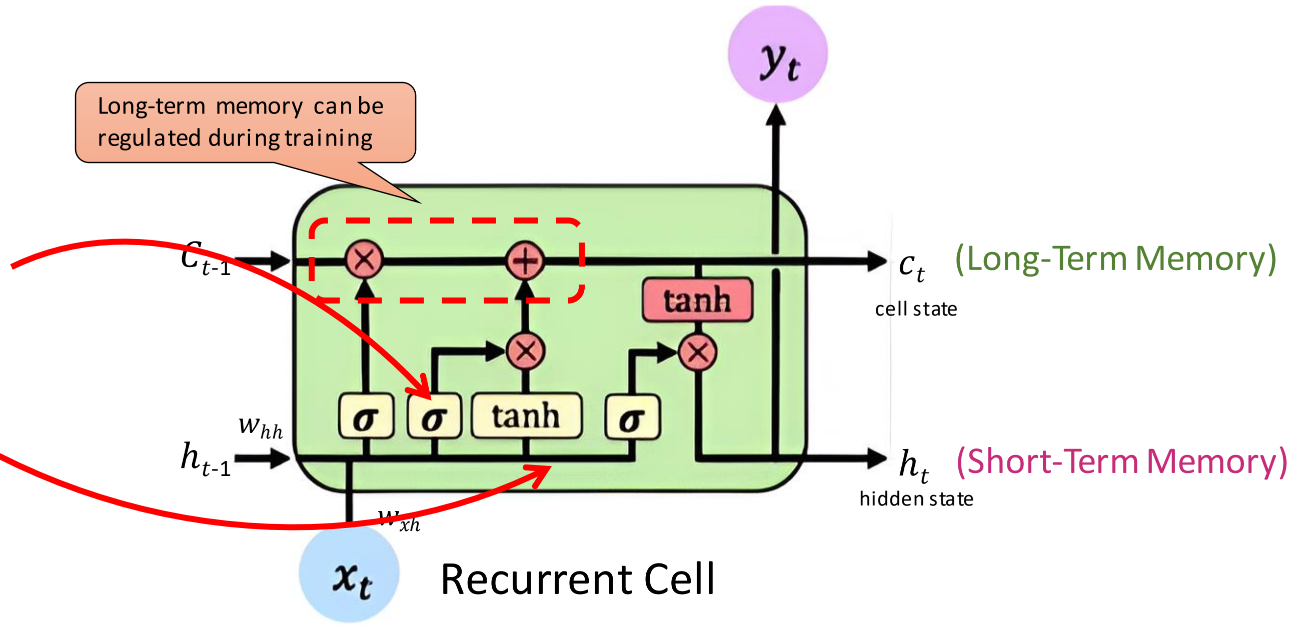
- In a conventional RNN, each recurrent module consists of a basic computational unit.
- LSTM cells contain computational blocks that control the information flow. Such cells can maintain information throughout many timesteps.
- The operation of an LSTM cell is comprised of various gates: 1) **Forget** Gate 2) **Input** Gate 3) **Cell State Update** 4) **Output** Gate



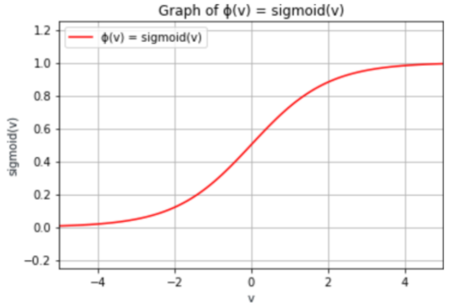
Source: MIT 6.5191

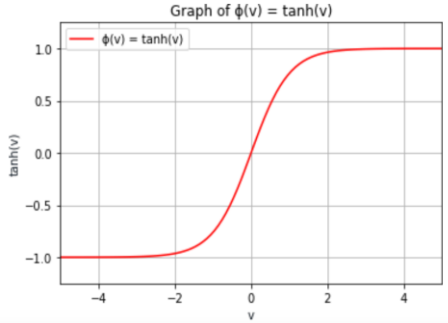
Long Short Term Memory (LSTM) Cell

LSTM uses *sigmoid* and *tanh* activation functions


Long-term memory can be regulated during training

Recurrent Cell

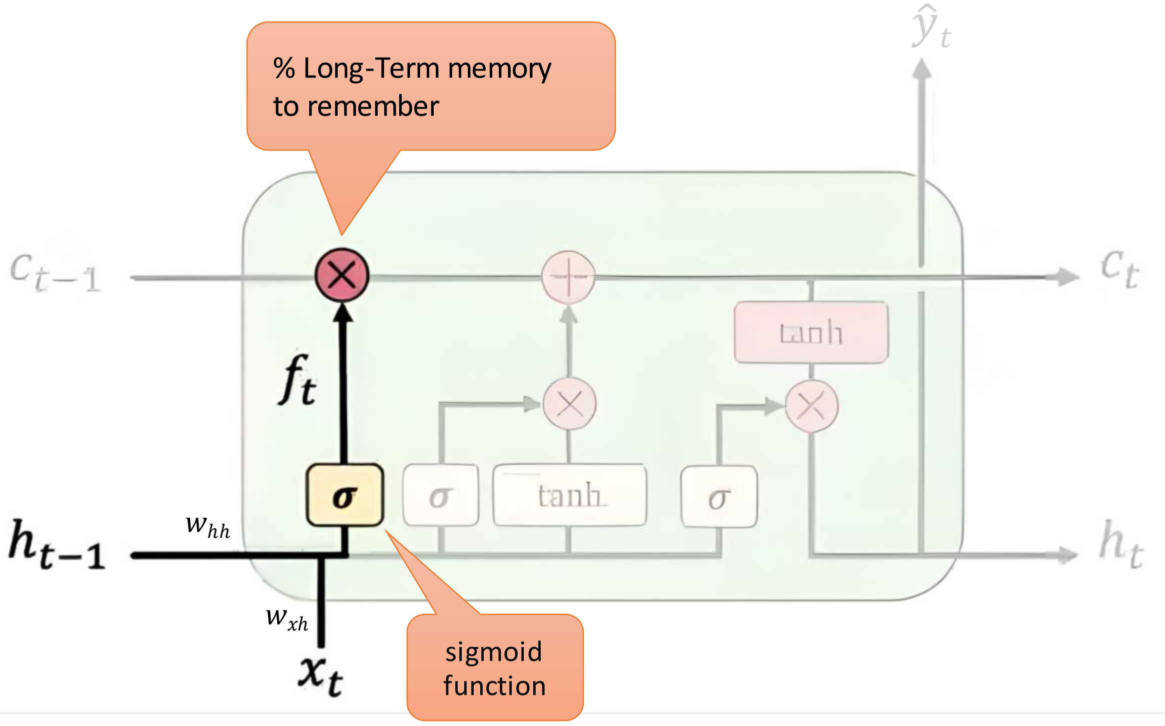




- The gated mechanism carefully regulates the **cell state** updates to prevent the gradients from **exploding** or **vanishing** as the information flows through a sequence of time steps.

Operations of an LSTM Cell: Forget Gate

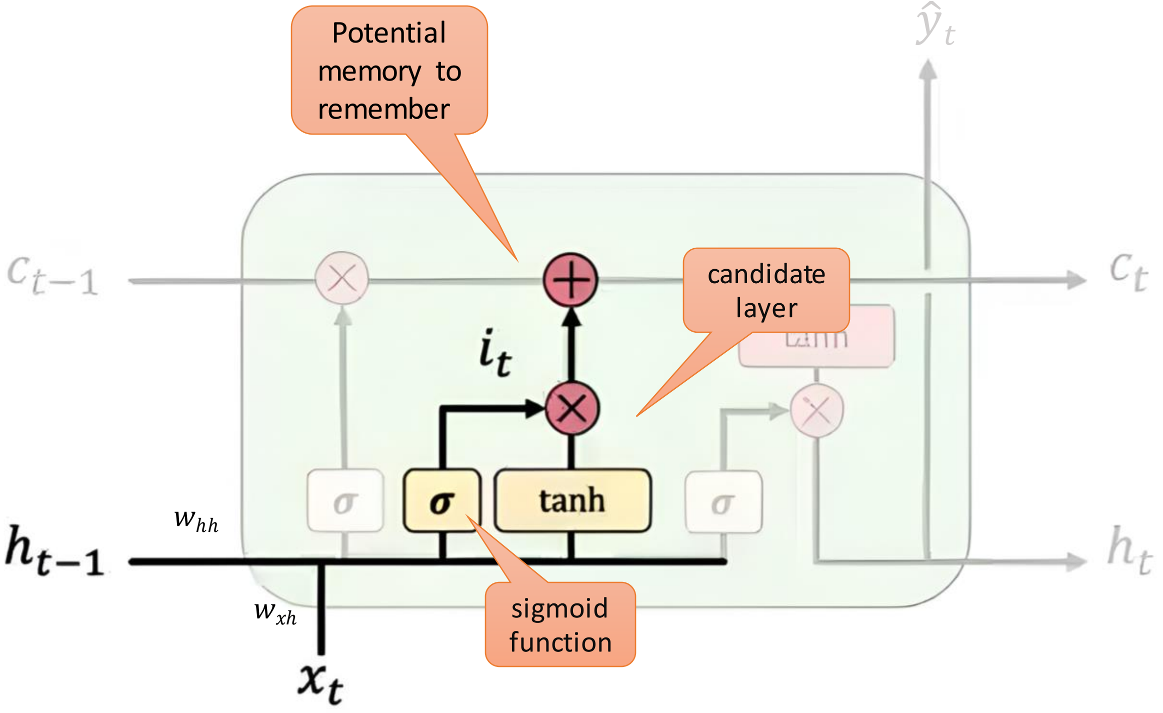
- The **Forget** Gate determines the percentage of information that should be discarded from the cell state with each time step in a sequence.
- This is done by using a **sigmoid** function. A value close to 0 means "forget this completely," while a value close to 1 means "retain this completely."



Source: MIT 6.5191

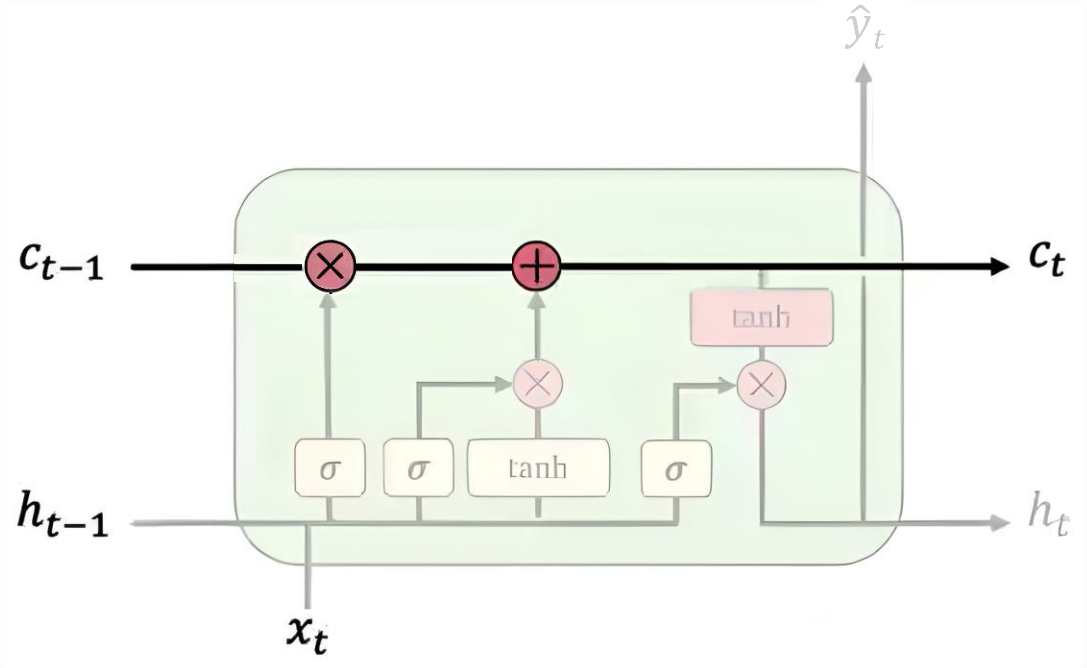
Operations of an LSTM Cell: Input Gate

- **Input Gate ($\boldsymbol{i_t}$)** – This gate assesses each incoming data and decides how much it should be incorporated into the cell state.
- **Candidate Layer (CL) –** CL Generates possible new values for the cell state. These values, shaped by the '*tanh*' function to regulate their scale, are proposed updates that could refresh the cell's memory with new, relevant information.



Source: MIT 6.5191

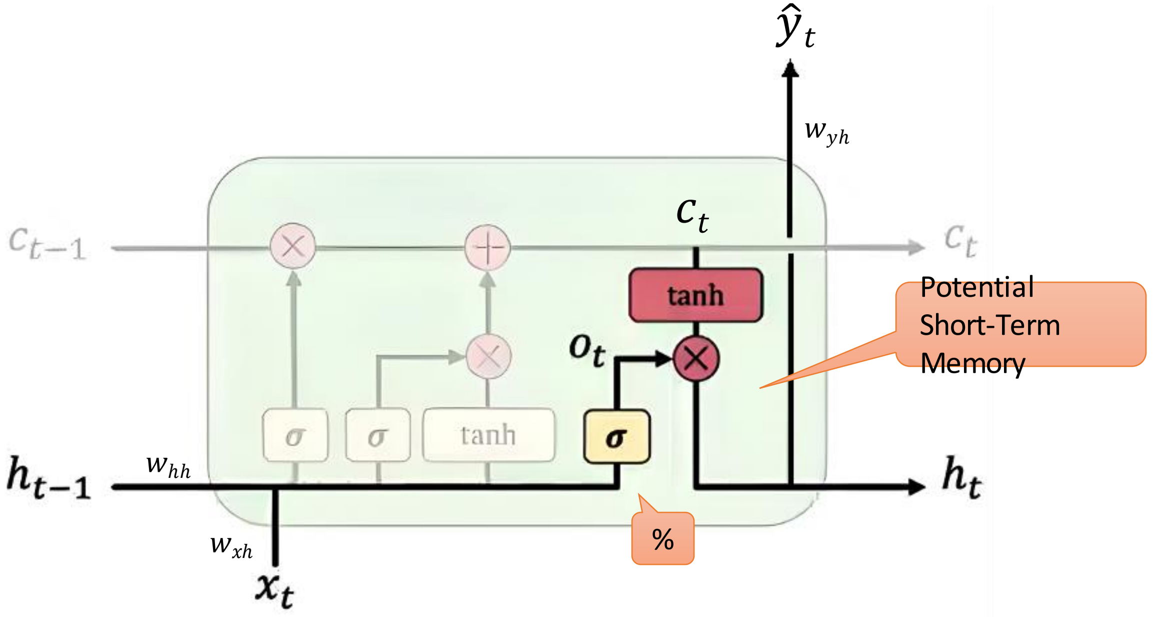
Operations of an LSTM Cell: Update Gate



- **Cell State Update** – This gate combines the % of the Long-Term memory to remember through the forget gate and input gate results with '+' (addition) operation, indicating that the new candidate values are combined with the existing cell state.

Source: MIT 6.5191

Operations of an LSTM Cell: Output Gate



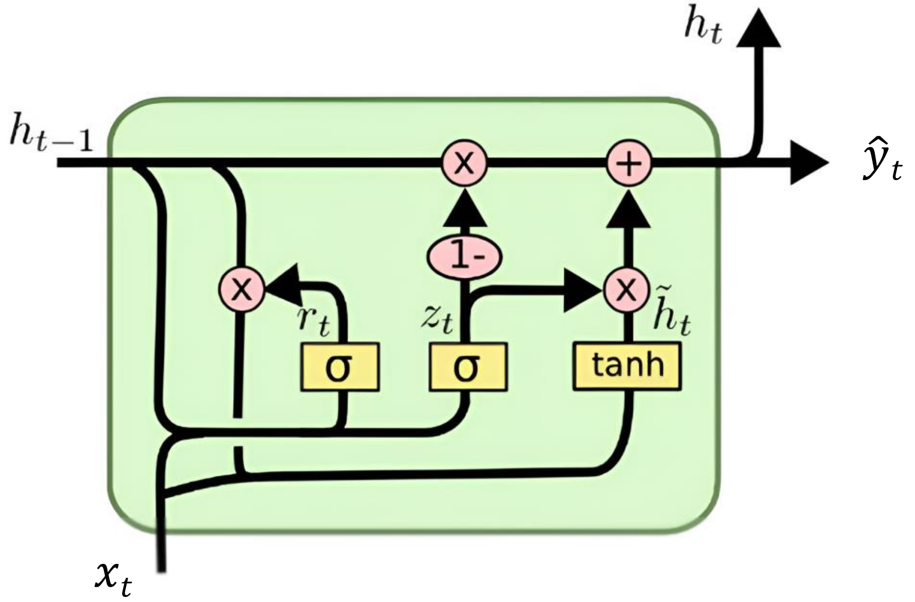
Potential Short-Term Memory

- Based on the updated cell state (c_t), and the output gate (o_t), we will be able to decide what the next hidden state (h_t) should be. This hidden state is used for predictions at the current time step ($\hat{y}_t$) and will be passed on to the next time step.
- **Output Gate ($\boldsymbol{o_t}$)** – Output gate (usually represented by another '*sigma*' function) – Determines what part of the cell state should be output as the hidden state.

Source: MIT 6.5191

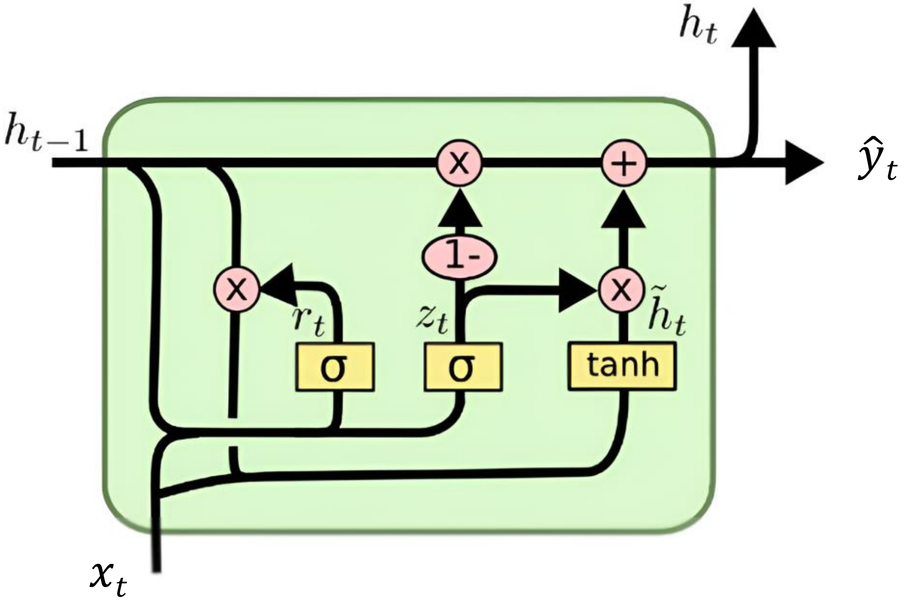
Gated Recurrent Unit (GRU)

- Another type of RNN network called a *Gated Recurrent Unit* (GRU)
- In a GRU, two gates regulate the flow of information:
  - **Update Gate** (z_t): This gate decides how much of the past information (previous hidden state ($h_{t\text{-}1}$)) needs to be passed along to the future.
  - A value close to 1 in the **update gate** indicates that most of the past information will not be carried forward, whereas a value close to 0 will allow more past information to be used.



Gated Recurrent Unit (GRU)

- **Reset Gate** (r_t): This gate determines how much of the past information will be used to compute the current content state, often denoted by $\widetilde{h}_t$, which is the **candidate activation**.
  - If the reset gate is close to 0, it effectively makes the unit forget the previously computed state.
- The candidate activation $\widetilde{h}_t$ is calculated using the current input x_t , and the past hidden state $h_{t\text{-}1}$, adjusted by the **reset gate**. It represents the new information to be added to the memory.
- The hidden state (h_t) for the current time step is computed as a combination of the previous hidden state and the candidate activation.



Limitations of RNNs

- **Information Loss**: Encoding at each step can lead to the loss of earlier information.
- **Speed**: RNNs process data sequentially, which cannot be parallelized, resulting in slower performance.
- **Memory Constraints**: Traditional RNNs struggle to maintain information over long sequences. While LSTMs & GRUs aim to address this, they still face information loss and speed challenges.

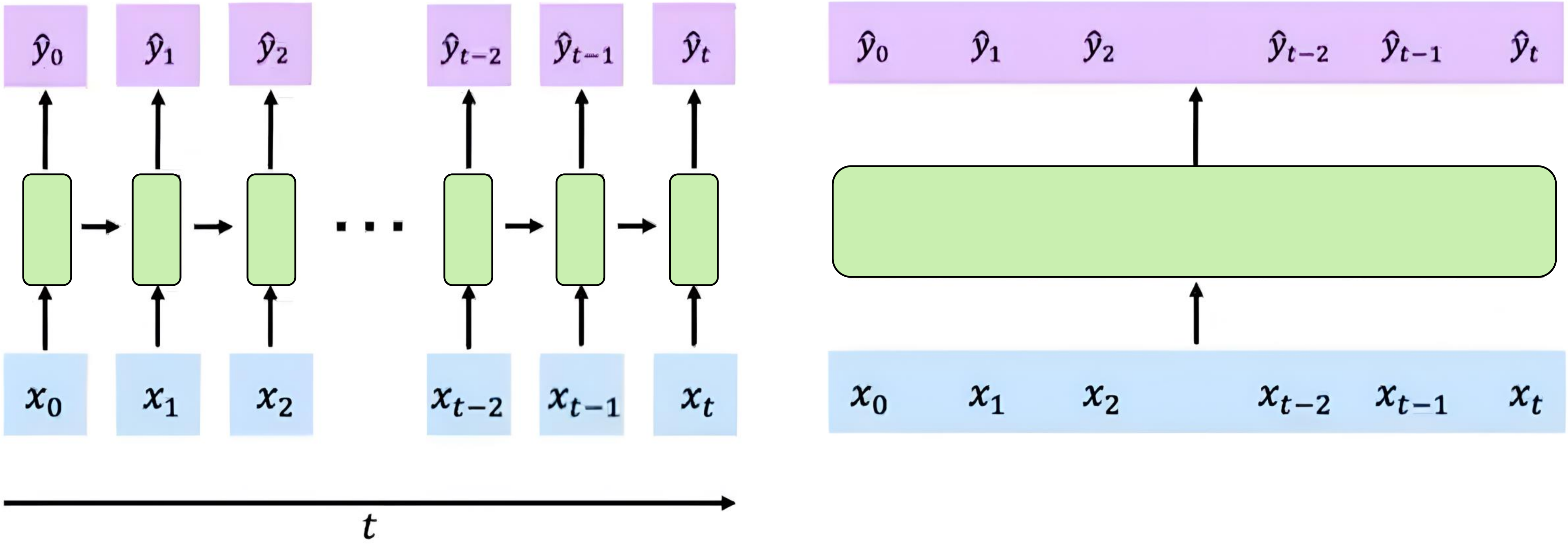
Potential Alternative Approach

- **Idea**: What if, instead of handling data sequentially, we input all our data into a standard feedforward neural network at once?

- **Challenges with Feedforward Networks:**
  - **Variable-Length Sequences**: These networks are not designed to handle sequences of varying lengths, such as sentences.
  - **Temporal Order**: When data is presented simultaneously, the sequential order of data, essential for context and meaning, is lost.

- **What could be a more efficient solution that retains the benefits of RNNs without limitations?**

From RNNs to Transformers


$\hat{y}_0$ $\hat{y}_1$ $\hat{y}_2$ $\hat{y}_{t-2}$ $\hat{y}_{t-1}$ $\hat{y}_t$
x_0 x_1 x_2 x_{t-2} x_{t-1} x_t
t
$\hat{y}_0$ $\hat{y}_1$ $\hat{y}_2$ $\hat{y}_{t-2}$ $\hat{y}_{t-1}$ $\hat{y}_t$
x_0 x_1 x_2 x_{t-2} x_{t-1} x_t

Transformers

Attention is all you need!

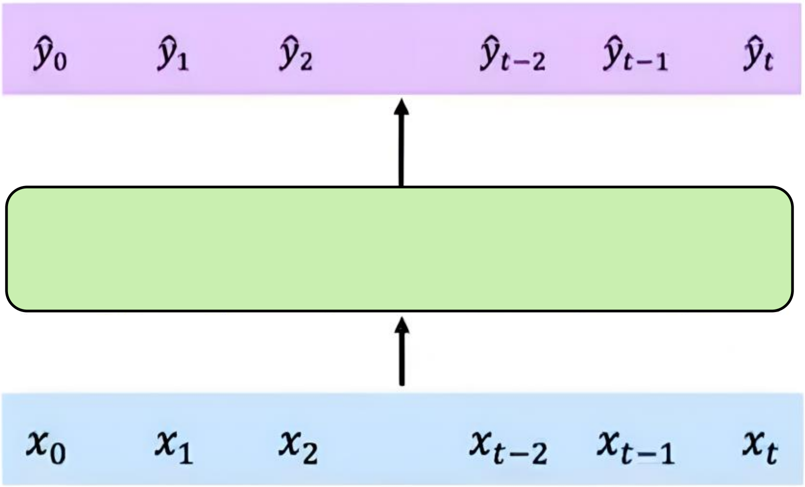
- This idea was introduced in NLP for the first time in machine translation
- Attending to the most important parts of input (just like human perception)
- Identify which parts to attend
- Extract features from those parts of the input

Intuition Behind Self-Attention

- Focusing on the most critical elements of an input:
  - Find which element(s) to prioritize.
  - Capture the characteristics with significant focus.

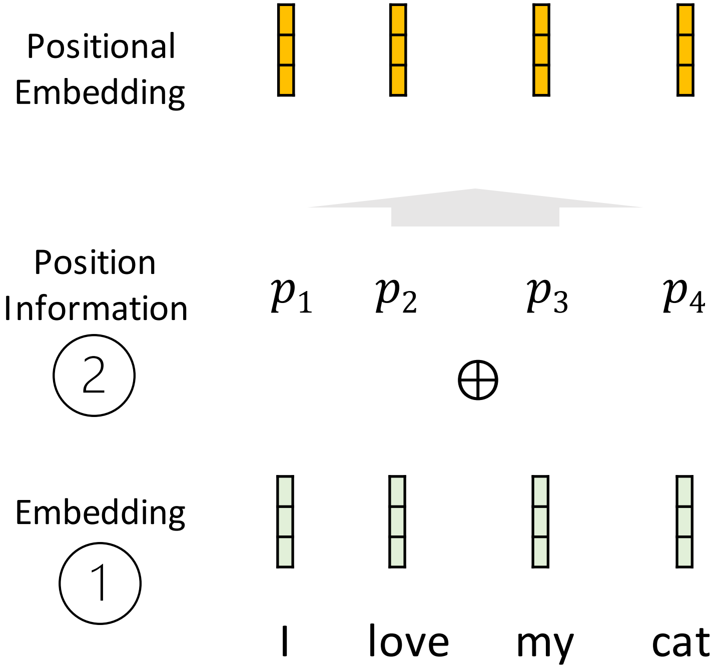
This is like a search problem!

Encode Positional Information

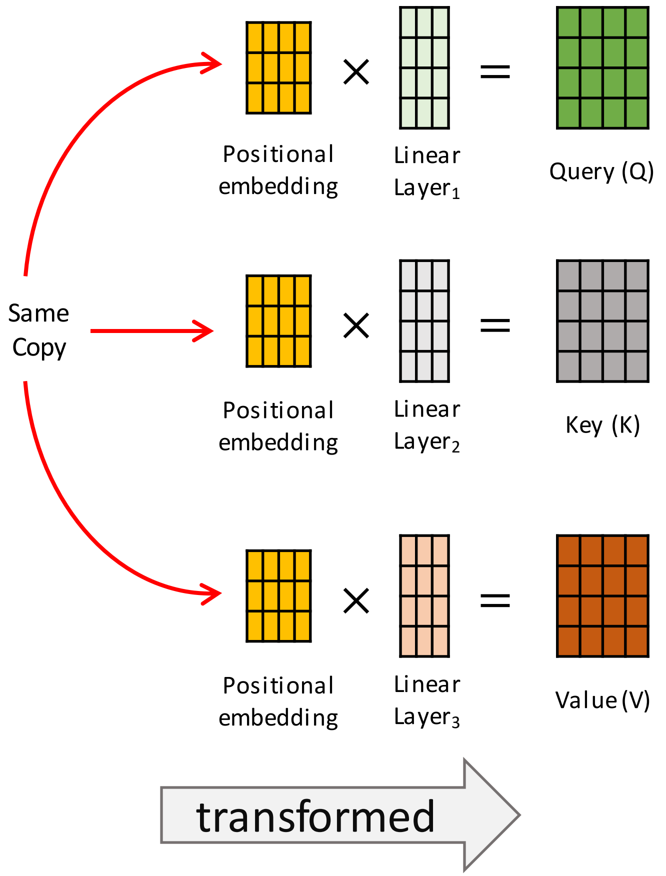


Data is fed to all at once

Thus, we need to encode position information to preserve the order.

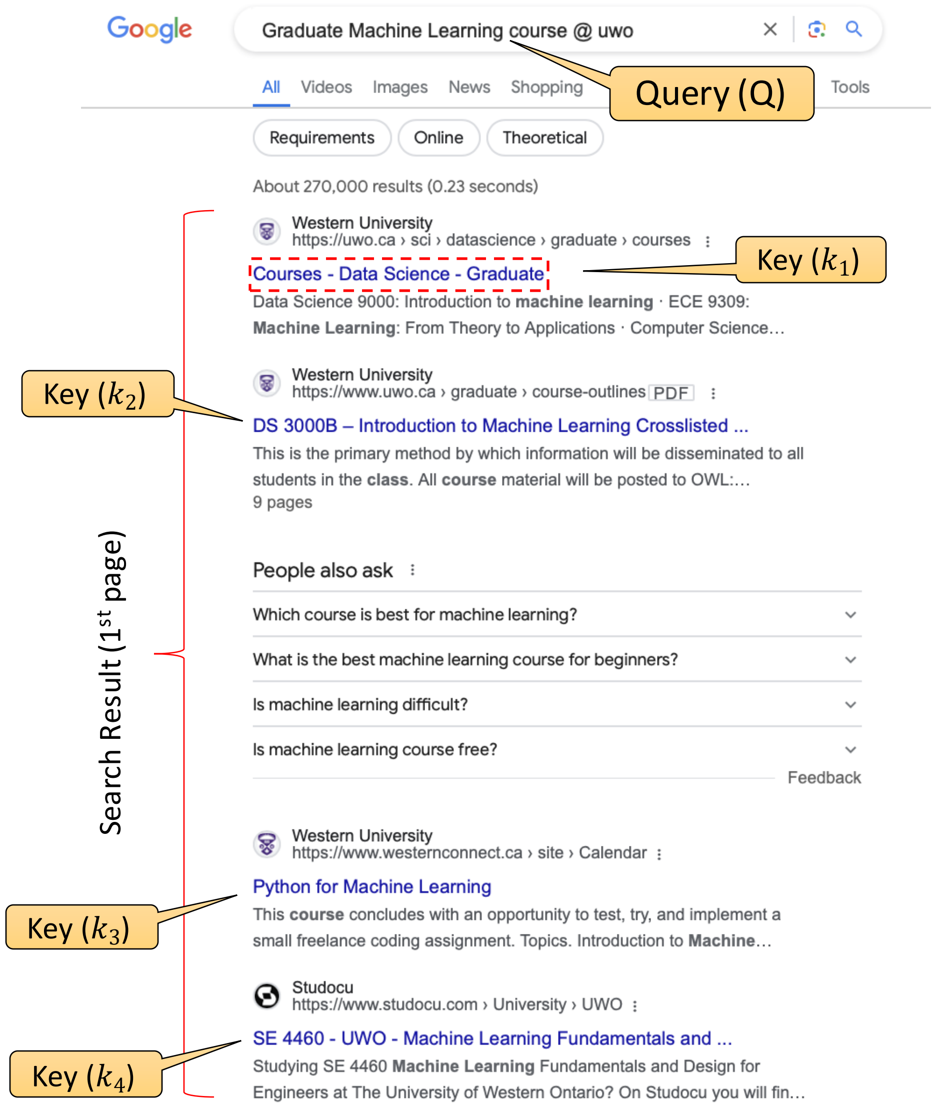


Extract Query, Key, and Value for Search



- Input elements are transformed into a **query** vector, a **key** vector, and a **value** vector.
- The **query** represents the current word(s) we want to generate attention for.
- The **keys** are compared with the **query** to compute <u>attention weights</u>, and **values** are the <u>actual representations</u> of the words.
- A linear layer is often called a dense or fully connected layer. In this layer, every input node is linked to every output node through a series of weights, facilitating a **linear transformation** of the input data.
- These vectors allow for the parallel processing of the self-attention mechanism across all positions in the sequence.

Analogy



Google

Graduate Machine Learning course @ uwo

All Videos Images News Shopping Tools

Requirements Online Theoretical

About 270,000 results (0.23 seconds)

Western University
https://uwo.ca › sci › datascience › graduate › courses

Courses - Data Science - Graduate

Data Science 9000: Introduction to **machine learning** · ECE 9309: **Machine Learning**: From Theory to Applications · Computer Science...

Western University
https://www.uwo.ca › graduate › course-outlines PDF

DS 3000B – Introduction to Machine Learning Crosslisted ...

This is the primary method by which information will be disseminated to all students in the **class**. All **course** material will be posted to OWL:...
9 pages

People also ask

Which course is best for machine learning?

What is the best machine learning course for beginners?

Is machine learning difficult?

Is machine learning course free?

Feedback

Western University
https://www.westernconnect.ca › site › Calendar

Python for Machine Learning

This **course** concludes with an opportunity to test, try, and implement a small freelance coding assignment. Topics. Introduction to **Machine**...

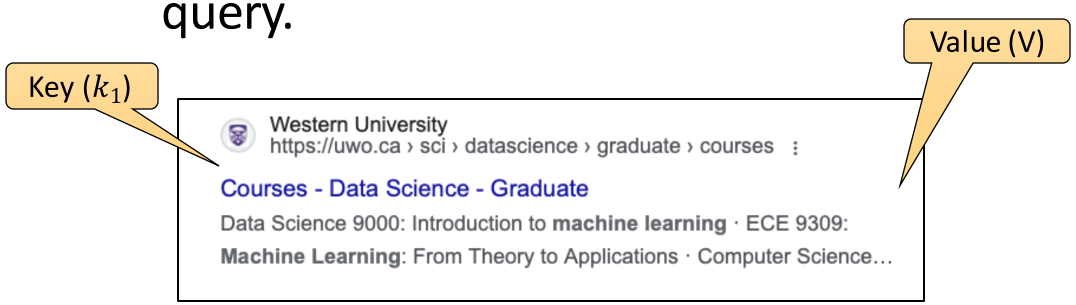
Studocu
https://www.studocu.com › University › UWO

SE 4460 - UWO - Machine Learning Fundamentals and ...

Studying SE 4460 **Machine Learning** Fundamentals and Design for Engineers at The University of Western Ontario? On Studocu you will fin...

How similar is the key to the query?

Step 1: Generate an attention score/weight by computing the similarity of each key to the desired query.



Western University
https://uwo.ca › sci › datascience › graduate › courses

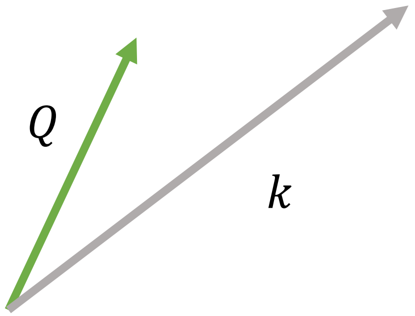
Courses - Data Science - Graduate

Data Science 9000: Introduction to **machine learning** · ECE 9309: **Machine Learning**: From Theory to Applications · Computer Science...

Step 2: Extract values based on attention (return the values with the highest attention)

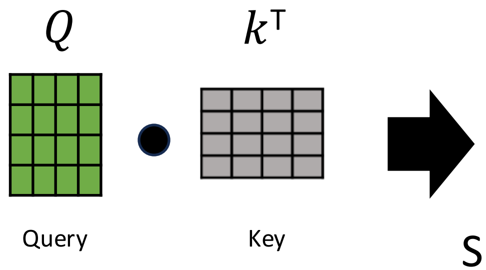
Compute Attention Weights

- **Attention Weights** : Compute the pairwise similarity between the query and each key



How do we compute the similarity

between two feature vectors?



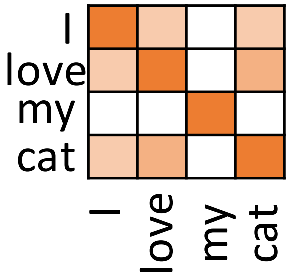
$$\frac{Q \cdot k^{\top}}{\sqrt{d_k}}$$

Similarity metric

- The dot product is scaled down by the **square root of the dimension** of the key vectors.
- This scaling prevents having extremely small gradients during backpropagation, which can happen when the dot product values are very large.

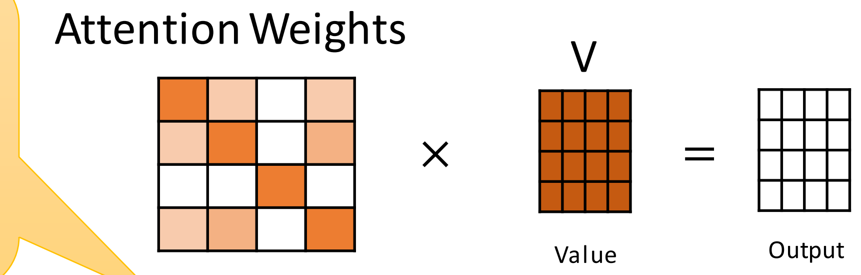
$$\text{Attention Weights} = \ Softmax(\frac{Q \cdot k^{\top}}{\sqrt{d_k}})$$

- The softmax operation ensures that the weights for each row sum to 1

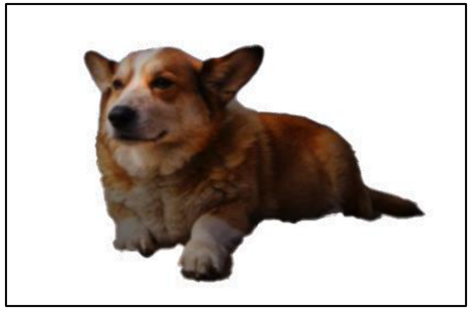


Self-Attending

Calculate the attention weights, representing the similarity between different positions in the sequence. This step is essential for identifying which parts of the sequence are most relevant to each other.



$$Softmax(\frac{Q\cdot k^{T}}{\sqrt{d_k}})\cdot V = A(Q,K,V) = c^{(t)}$$

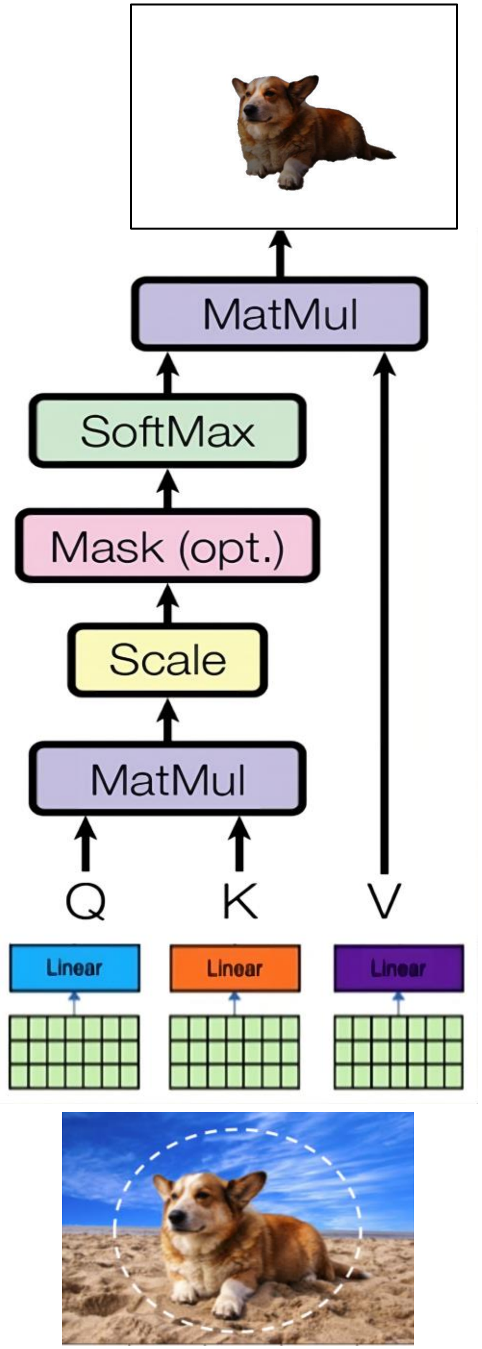


- The result of the weighted sum is the new set of vectors $c^{(t)}$, often called **context vectors**.
- Each context vector combines value vectors, weighted by their relevance to the corresponding query vector.

- Self-Attending: Extract features with high attention

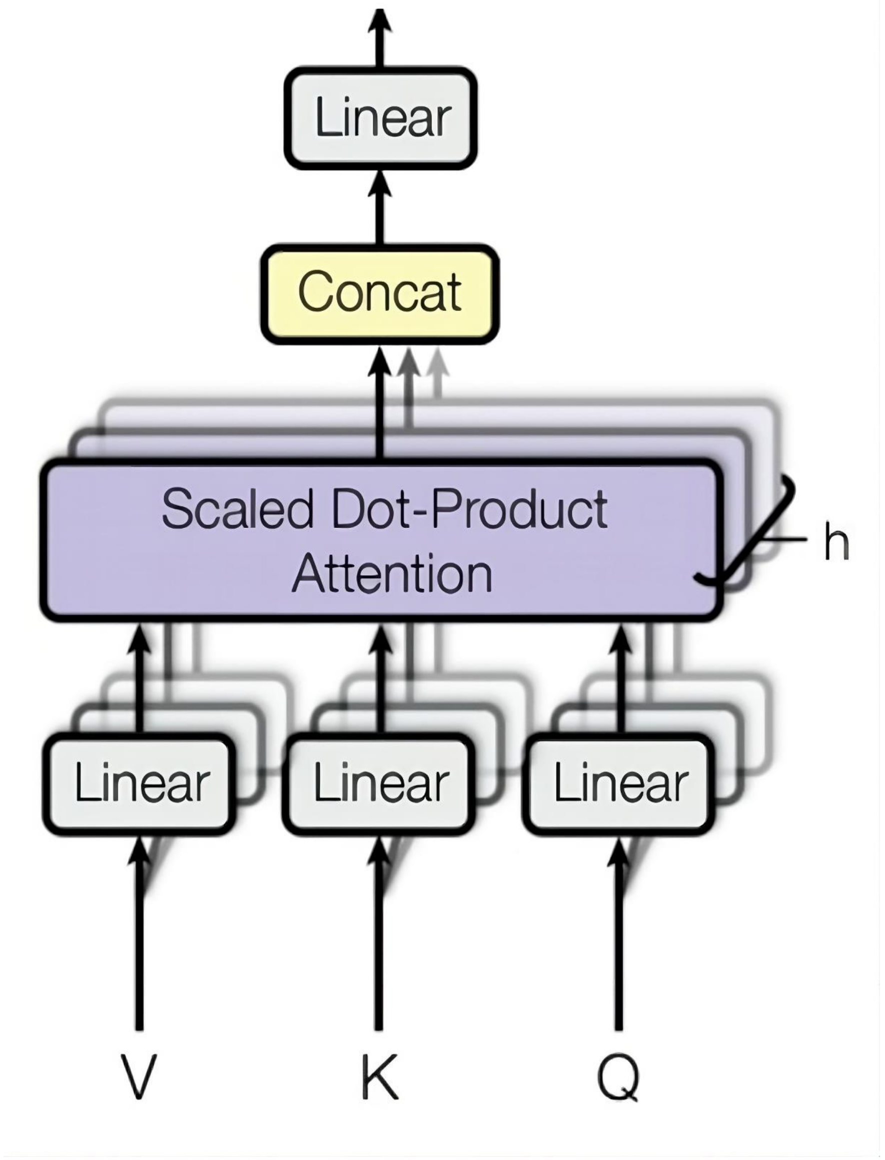
Attention Head

- An attention head is a component in models like the Transformer.
- Each attention head can be considered an **independent feature extractor** focusing on a different input data part or aspect.
- By performing the self-attention calculation, the attention head determines the relevance or importance of different parts of the input relative to each other.



Multiple Attention Heads

- Different Attention Heads (Scaled Dot-Product Attention) may focus on different types of relationships within the data.
- For example, in the context of language, one head might pay more attention to syntactic structures, another to semantic relationships, and another to co-reference resolution.
- The outputs of all attention heads are typically combined, either through **concatenation** or some other method, to form a single output that integrates the diverse perspectives provided by each head.



Applications: Generative Pre-trained Transformer (GPTs)

- A pre-trained Transformer is a type of transformer that has been **initially trained** on a large dataset to learn a wide range of patterns, structures, and nuances in the data.

- The pre-training phase in GPTs involves teaching the model to perform language-related tasks without human-labeled guidance, such as predicting the next word in a sentence.

References

- MIT Introduction to Deep Learning, “Deep Sequence Modeling”.
- StatQuest with Josh Starmer, “Recurrent Neural Networks (RNNs), Clearly Explained!!!”
- cs231 Li et al., “ Recurrent Neural Networks”, 2022.
- N. Zolaktaf, “Recurrent Neural Networks”, 2016.
- C. C. Aggarwal, Neural Networks and Deep Learning, Chapter 7.1–7.2, IBM T J Watson Research Center Yorktown Heights, NY

Next

- Federated Learning

Thank
you