

CS 540 Introduction to Artificial Intelligence

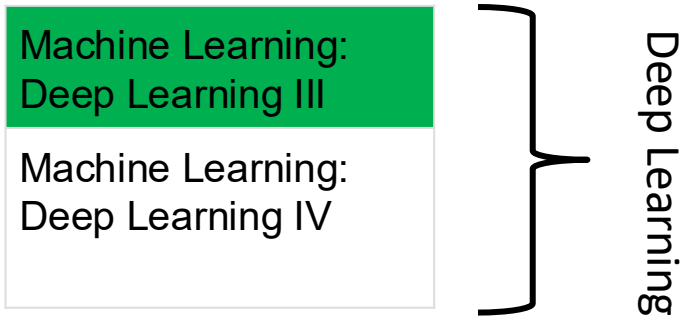
Deep Learning III

University of Wisconsin-Madison
Fall 2026 Sections 1 & 2

Announcements

- **Homework 7 released and due Friday Nov 13 at 11:59PM.**

- Class roadmap and schedule:



Outline for Today's Lecture

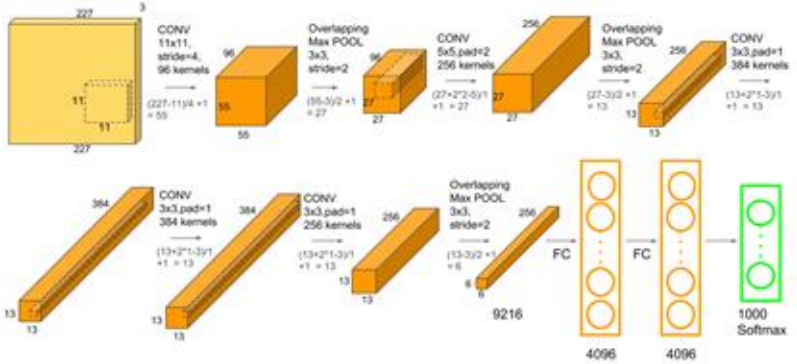
- Review ResNets
- Recurrent Neural Networks
- RNNs for Language Modeling
- LSTMs
- The Attention Mechanism
- Transformers

Last Time: CNNs

We talked about CNN components & architectures

- **Components**: convolutional layers, pooling layers (recall kernels, channels, strides, padding)
- **Architectures**: LeNet, AlexNet, VGG, ResNet

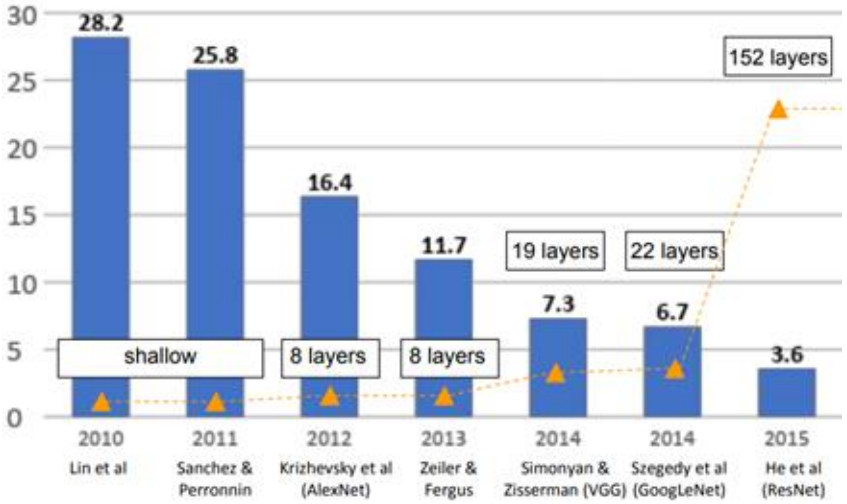
- Trend: bigger, deeper.



Credit: Mathworks

Evolution of CNNs

ImageNet competition (error rate)



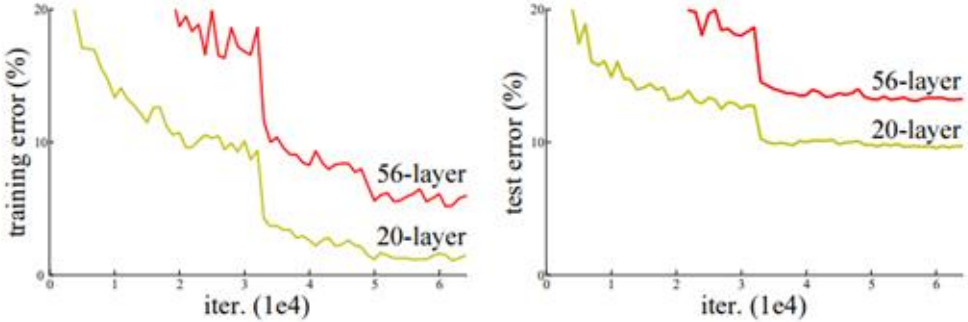
Credit: Stanford CS 231n

Can we just keep adding layers to get from VGG (19 layers) to ResNet (152 layers)?

No! Some problems:

- i) Vanishing gradients: more layers ➔ more likely
- ii) Instability: deeper models are harder to optimize

Reflected in training error:

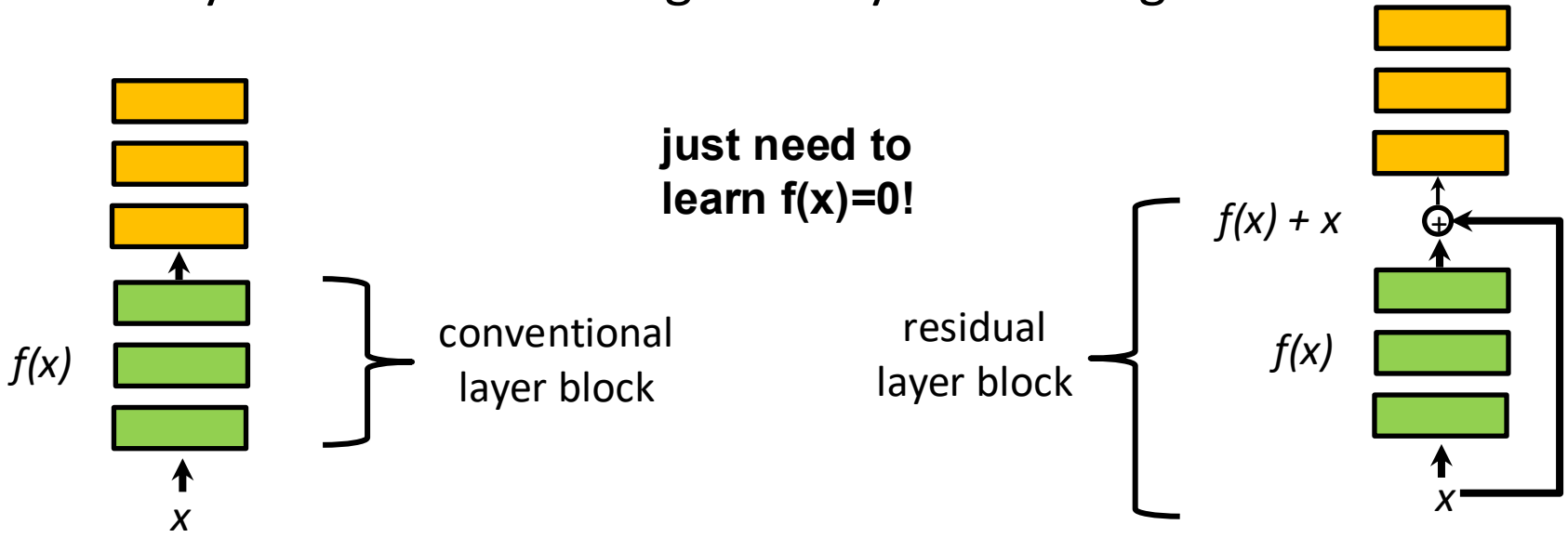


He et al: “Deep Residual Learning for Image Recognition”

Residual Connections

Idea: Identity might be hard to learn, but zero is easy!

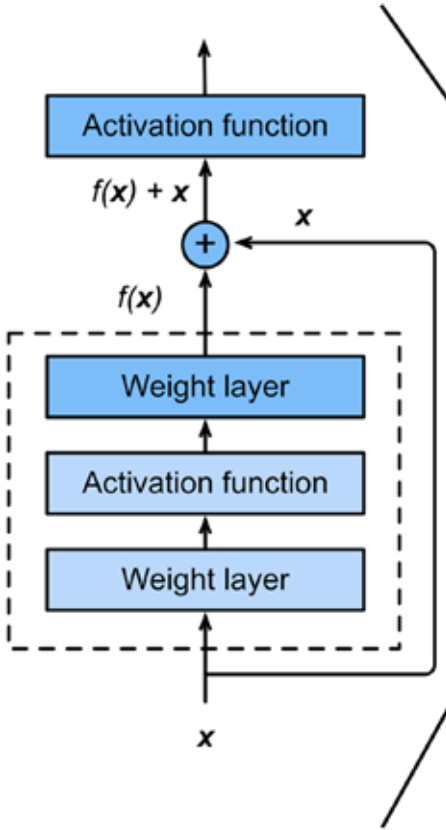
- Make all the weights tiny, produces zero for output
- Can easily transform learning identity to learning zero:

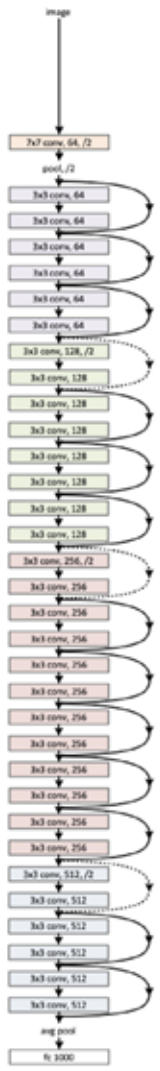


Full ResNet Architecture

[He et al. 2015]

- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2





(Figure from Stanford CS231n)

A Bit More on ResNets

Idea: Residual (skip) connections help make learning easier

- Note: Can also analyze from **backpropagation** p.o.v
  - Residual connections add paths to computation graph

- Also uses **batch normalization**

  Ioffe and Szegedy: *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*
  - Normalize the features at each layer to have same mean/variance
  - Common deep learning trick

- Highway networks: learn weights for residual connections



Recurrent Neural Networks (RNNs)

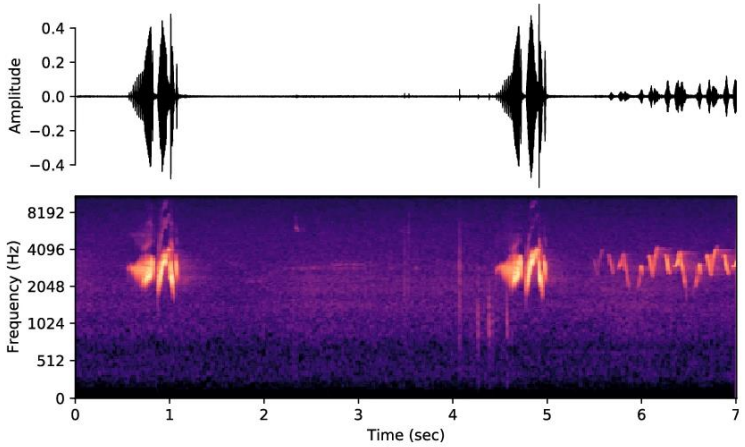
Recurrent Neural Networks (RNNs)

Why Recurrent Neural Networks?

- Handle sequential data (text, audio, speech, time series)
- Allow more general computations

The quick brown fox

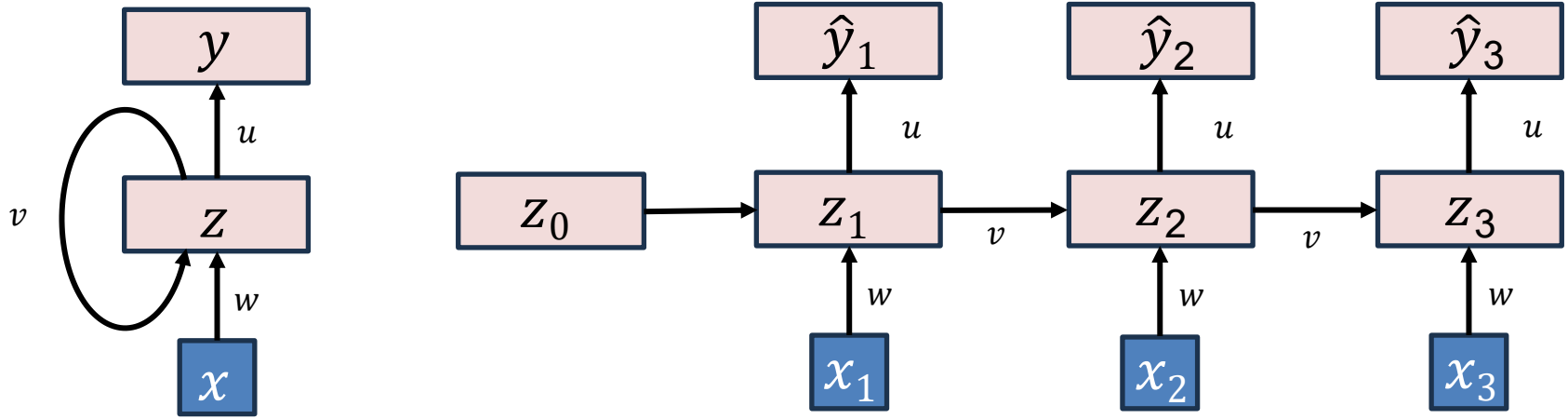
The → quick → brown → fox



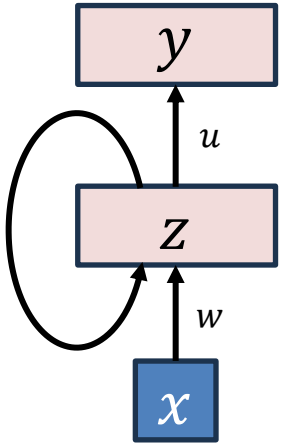
Gupta, G., Kshirsagar, M., Zhong, M. et al. Comparing recurrent convolutional neural networks for large scale bird species classification.

Recurrent Neural Networks (RNNs)

- RNNs introduce **cycles** in the computational graph
- Allowing information to persist; **memory**

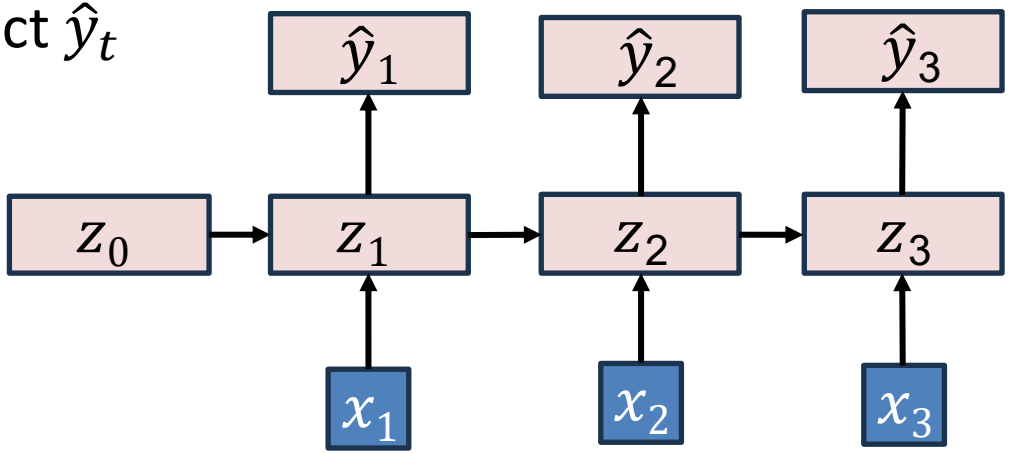


RNNs: High Level



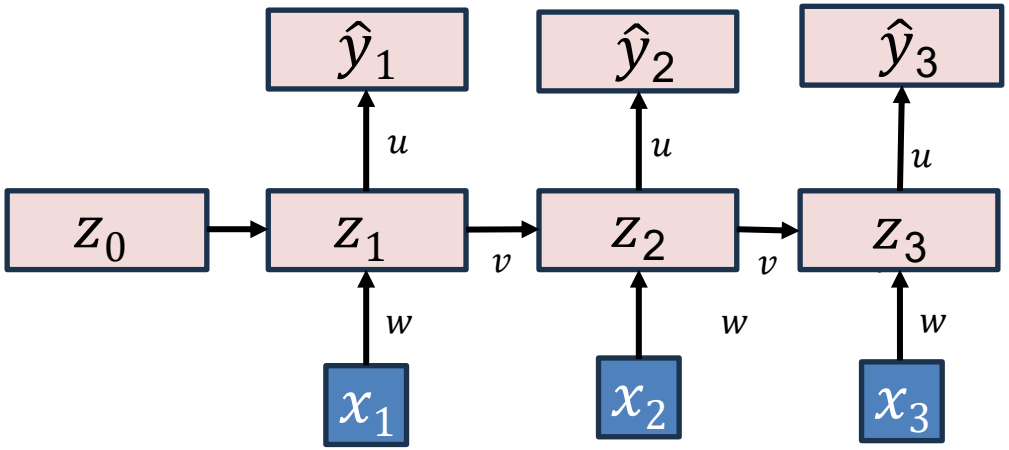
At each time step t:

- Receive input token x_t
- Receive old hidden state z_{t-1}
- Use x_t and z_{t-1} to compute new hidden state z_t
- Use z_t to (possibly) predict $\hat{y}_t$



Recurrent Neural Networks (RNNs)

- In each time step, the **input value** and the **output of previous hidden state** are used in the computation
- Internal state, **memory** – inputs received at earlier time steps affect the RNN's response to the current input.



Activation Function

$$\hat{y}_t = g_y(Uz_t)$$

$$z_t = g_z(Vz_{t-1} + Wx + b)$$

Activation Function



RNNs for Language Modeling

Key Application: Language Modeling

- Basic idea: assign probability to text

$$P(w_1, w_2, \ldots, w_n)$$

$$P(w_{\text{next}} \mid w_1, \ldots, w_{n-1})$$

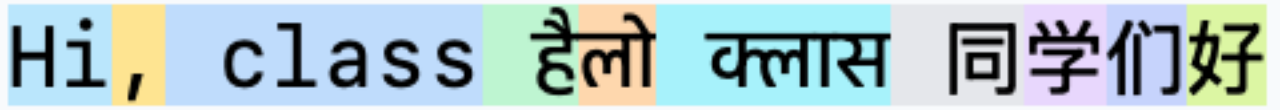
- Underlies ChatGPT, Claude, etc.

Next-Token Prediction

- Treat text as a sequence of **tokens.**
- Task: predict the next token
- Simplest: tokens = characters

“Hi, class” ⇒ 

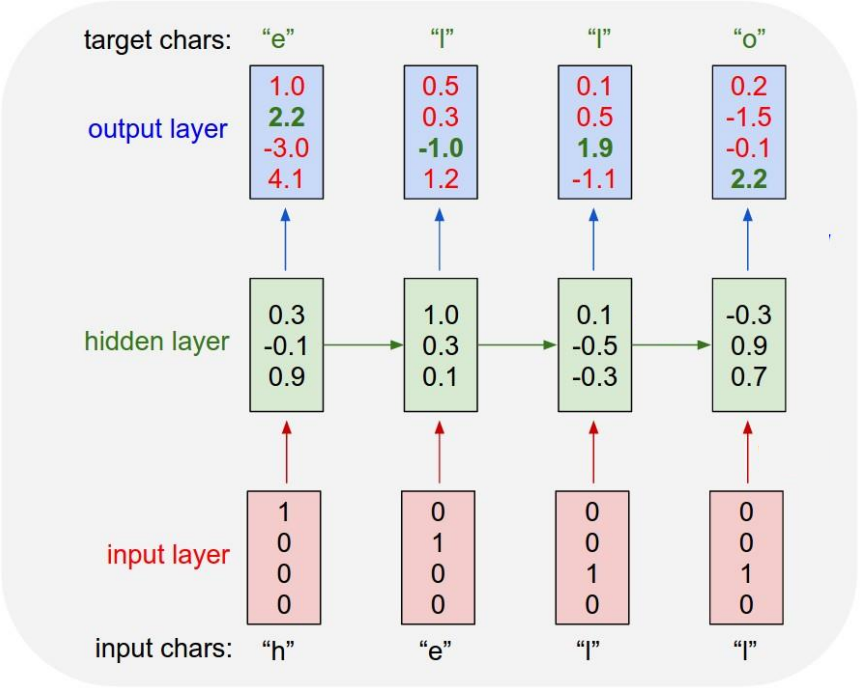
- OpenAI’s GPT4o uses over 200,000 tokens



https://tiktokenizer.vercel.app

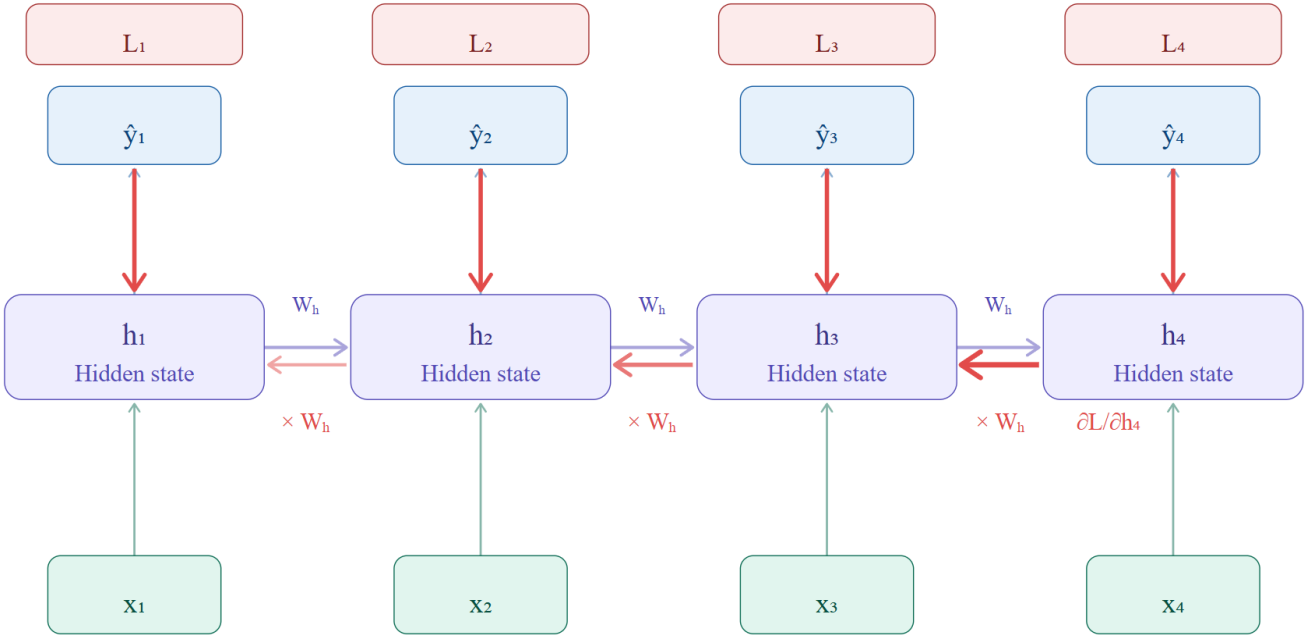
Example: RNNs on Text

- Simple example
  - 4 tokens: “h”, “e”, “l”, “o”
  - Hidden state has 3 dimensions
- Training: try to make output match targets
- Training: Back-propagation Through Time (BPTT)
- Generation: sample!
  - (Same as with n-gram)



https://karpathy.github.io/2015/05/21/rnn-effectiveness/

Back-propagation Through Time (BPTT)



Vanishing \ Exploding Gradients issues for long sequences



Long Short-term Memory (LSTM)

Why LSTMs?

- **RNNs** handle sequences but struggle with long-term dependencies (**short memory**)
- **Vanishing \ Exploding Gradients** due to backprop through time (BPTT)



- What if we could **remember important information** and **forget unimportant** information over long sequences?

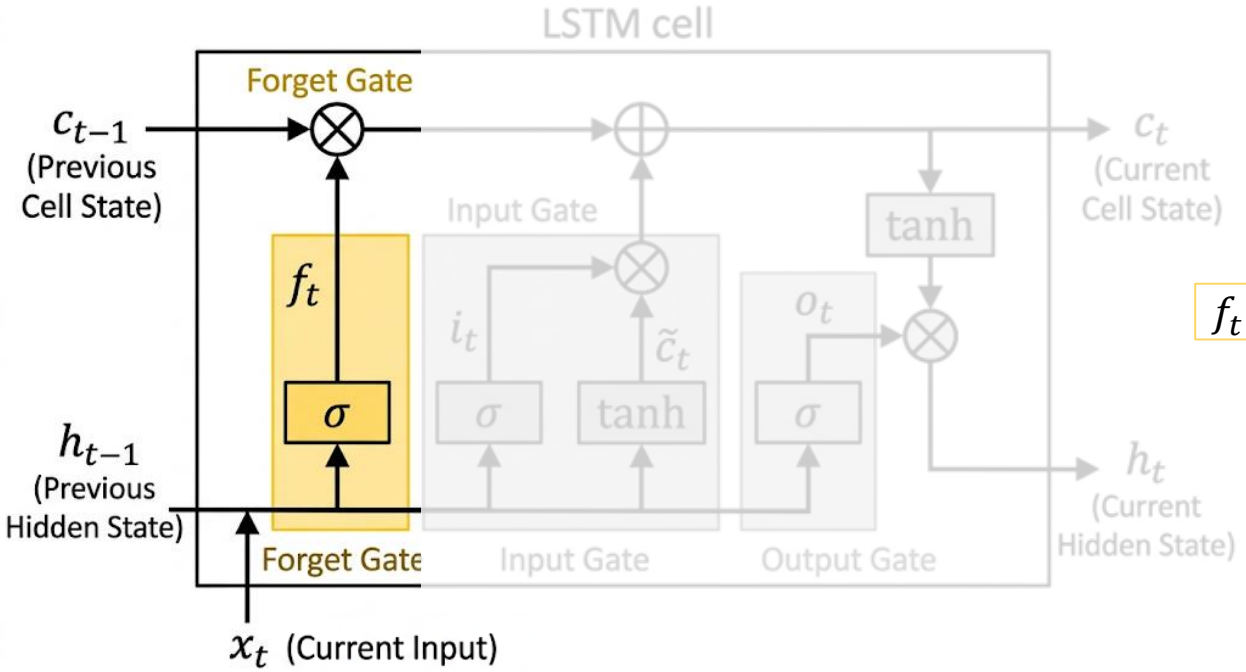
Long Short-term Memory (LSTM)

An RNN designed to processes data one step at a time while maintaining a memory of what came before.

- Cell state (c): Long-term memory component of LSTM, **copied** from time-step to time-step. (In contrast, the basic RNN multiplies its memory by a weight matrix)
- Hidden State (h): The short-term memory/ working output

Long Short-term Memory (LSTM)

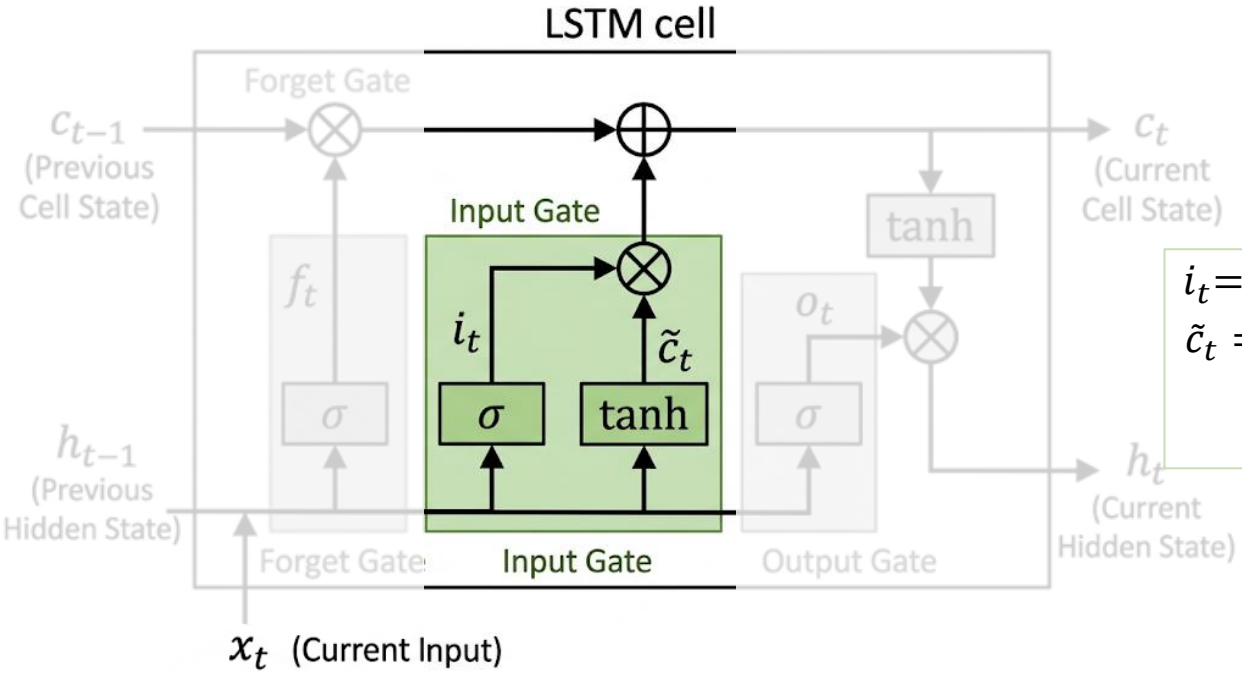
The **forget gate** (f) determines if each element of the memory cell is remembered (copied to the next time step) or forgotten (reset to zero).



$$f_t = \sigma(W_{x,f} \cdot x_t + W_{h,f} \cdot h_{t-1} + b)$$

Long Short-term Memory (LSTM)

The **input gate** ($\boldsymbol{i}$) determines if each element of the memory cell is updated additively by new information from the input vector at the current timestep.

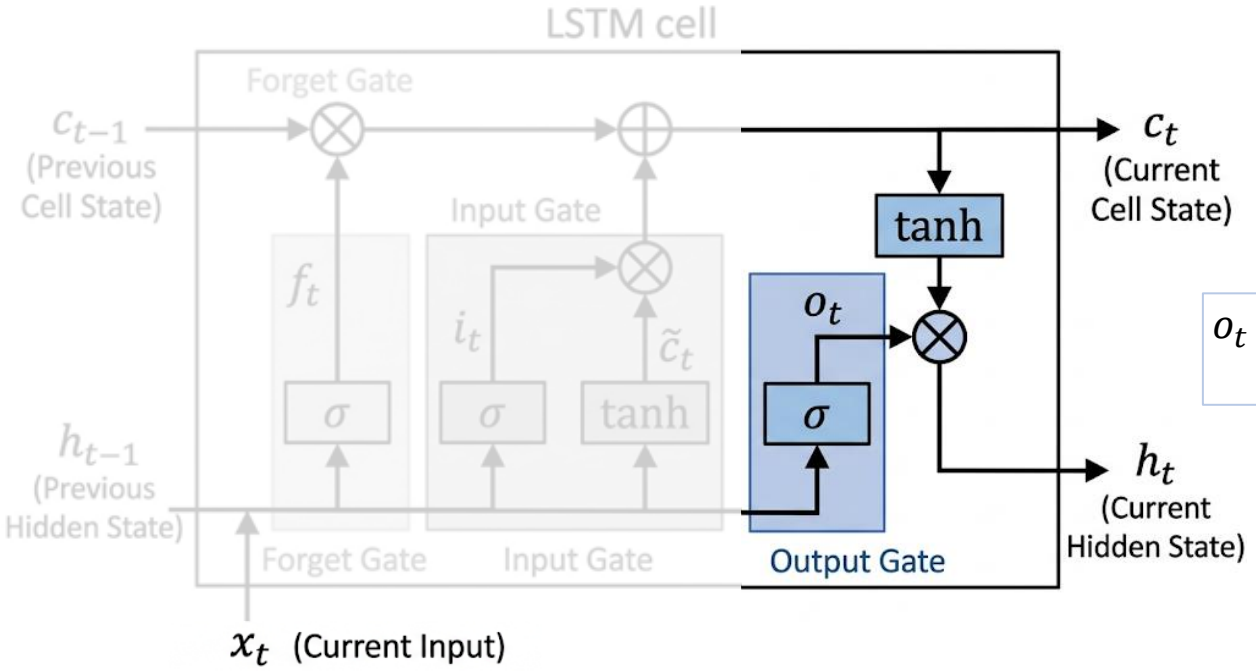


$$i_t = \sigma\left(W_{x,i} \cdot x_t + W_{h,i} \cdot h_{t-1} + b\right)$$
$$\tilde{c}_t = tanh\left(W_{x,\tilde{c}} \cdot x_t + W_{h,\tilde{c}} \cdot h_{t-1} + b\right)$$

$$c_t = f_t * c_{t-1} + i_t * \tilde{c}_t$$

Long Short-term Memory (LSTM)

The **Output Gate** ($\boldsymbol{o}$) filters the cell state to produce the current output.



$$o_t = \sigma\left(W_{x,o} \cdot x_t + W_{h,o} \cdot h_{t-1} + b\right)$$
$$h_t = tanh(c_t) * o_t$$

LSTM Key Steps

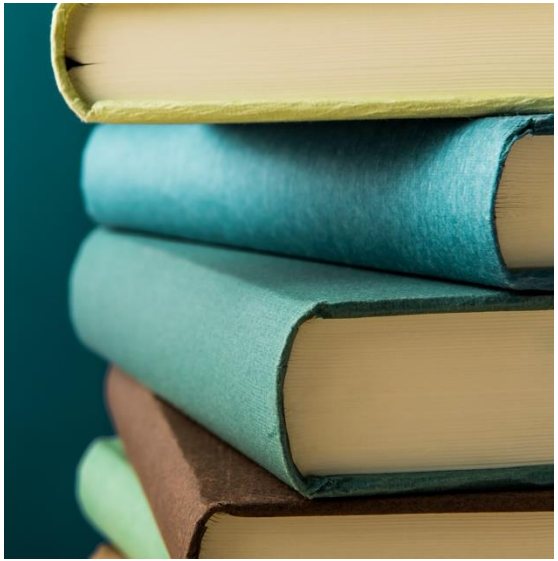
- **Forget gate** decides what to erase from memory
- **Input gate** decides what new information to write
- **Cell state** updates: old × forget + new × input
- **Output gate** decides what to reveal
- Produce **hidden state** (the current output)



The Attention Mechanism

From RNNS to Transformers

- RNNs process one word at a time, which is hard to parallelize, making them slow to train on GPUs
- Even LSTMs struggle with very long sequences
- All context gets squeezed into one hidden state vector



Word Representations & Context

- We use vectors to represent words (“embeddings”)
- Recall:



“dog”

  - One-hot representation $[0\ \ 1\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0]$
  - Dense embedding $[0.13 \quad 0.87 \quad -0.23 \quad 0.46\ ]$
    - Vector captures **meaning**

Problem: the meaning of a word depends on the words around it

Word Representations & Context

"The monkey ate the banana. It was ____, wasn't it?"

Does "it" mean **monkey** or **banana**?

Could be:

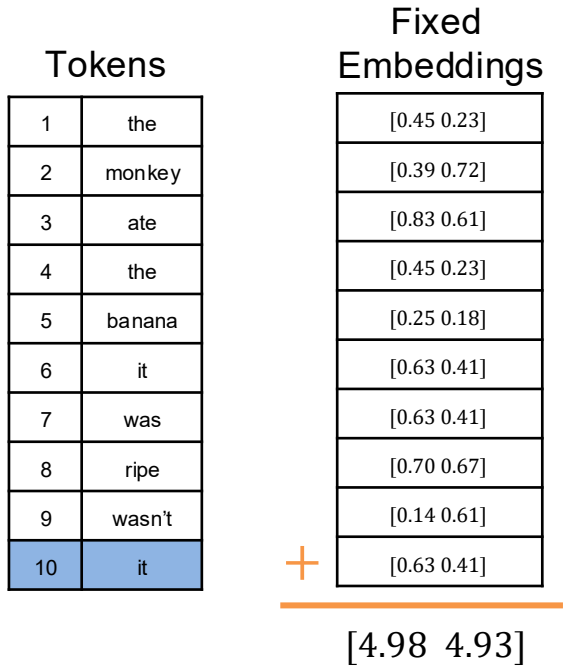
- The monkey ate the banana. It was ripe, wasn't it?
- The monkey ate the banana. It was hungry, wasn't it?

Meaning depends on past words (**context**)

The attention mechanism produces **contextual embeddings.**

Attempt 1: Naïve Contextual Embedding

- Each token has a fixed embedding vector x_i
- A crude attempt at contextual embedding: average over context
- Problem: Equal "attention" to every previous token



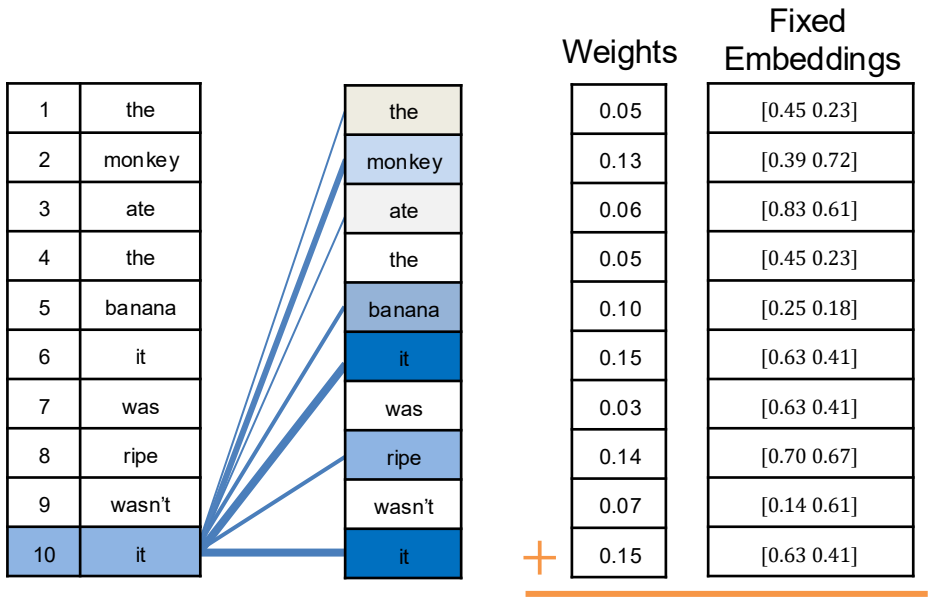
In math:
for the i-th token

$$c_i = \frac{1}{i}\sum_{j=1}^{i} x_j$$

Contextual embedding for "it": $[0.498 \;\; 0.493]$

Attempt 2: Assigning Weights

- Humans focus selectively — machines can too
- We can assign weights based on relevance
  - Idea: weight similar words higher
  - If $\langle x_i, x_j \rangle$ large, assign large weight
- Then take weighted sum



Contextual embedding for "it": $[0.37 \;\; 0.42]$

In math: for i-th token

$$r_{ij} = \frac{1}{\sqrt{d}}\langle x_i, x_j \rangle$$

$$p_{i,:} = \mathrm{softmax}(r_{i,:})$$

$$c_i = \sum_{j=1}^{i} p_{ij} \cdot x_j$$

Final Attempt: The Attention Mechanism

Previous attempt:

- Fixed embeddings in three locations.

$$r_{ij} = \frac{1}{\sqrt{d}} \cdot \langle x_i, x_j \rangle$$

$$p_{i,:} = \mathrm{softmax}(r_{i,:})$$

$$c_i = \sum_{j=1}^{i} p_{ij} \cdot x_j$$

In the **attention mechanism**:

- Each token is associated with **three** vectors
- Query: q_i, the one attended from
- Key: k_j, the one attended to
- Value: v_j, the context being generated

Query Key

$$r_{ij} = \frac{1}{\sqrt{d}} \cdot \langle q_i, k_j \rangle$$

$$p_{i,:} = \mathrm{softmax}(r_{i,:})$$

Value

$$c_i = \sum_{j=1}^{i} p_{ij} \cdot v_j$$

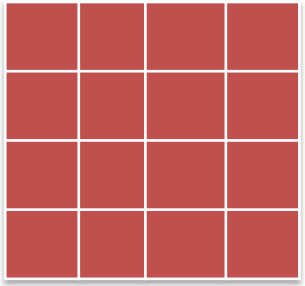
Query, Key, and Value Matrices

Input vector

x



W_q

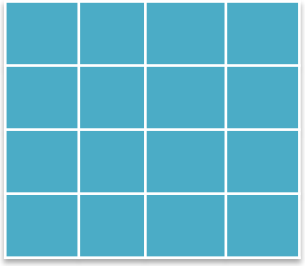


$q = W_q x$



Query

W_k

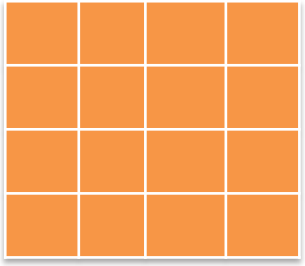


$k = W_k x$



Key

W_v



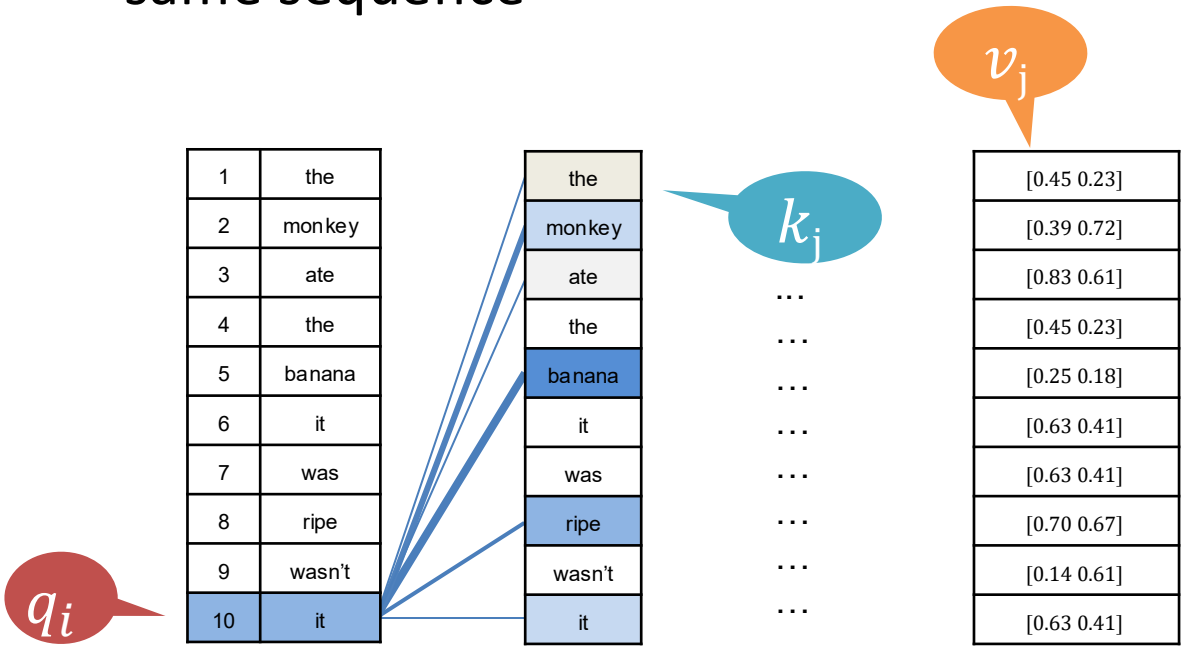
$v = W_v\ x$



Value

The Attention Mechanism

Each token attends to all tokens in the same sequence



$$r_{ij} = \frac{\langle q_i, k_j \rangle}{\sqrt{d}}$$

$$\mathrm{p_{i,:}} = \mathrm{softmax}(r_{i,:})$$

$$c_i = \sum_j p_{ij} \cdot \mathrm{v_j}$$

Notation for Attention

Queries, keys and values are written as matrices $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}$

$$\text{Attention}\,(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)\text{V}$$

$$Q, K, V \in \mathbb{R}^{L \times d}$$

$L =$ sequence length

$d =$ embedding dimension

Thus attention transforms one length-L sequence of d-dimensional embeddings into another.

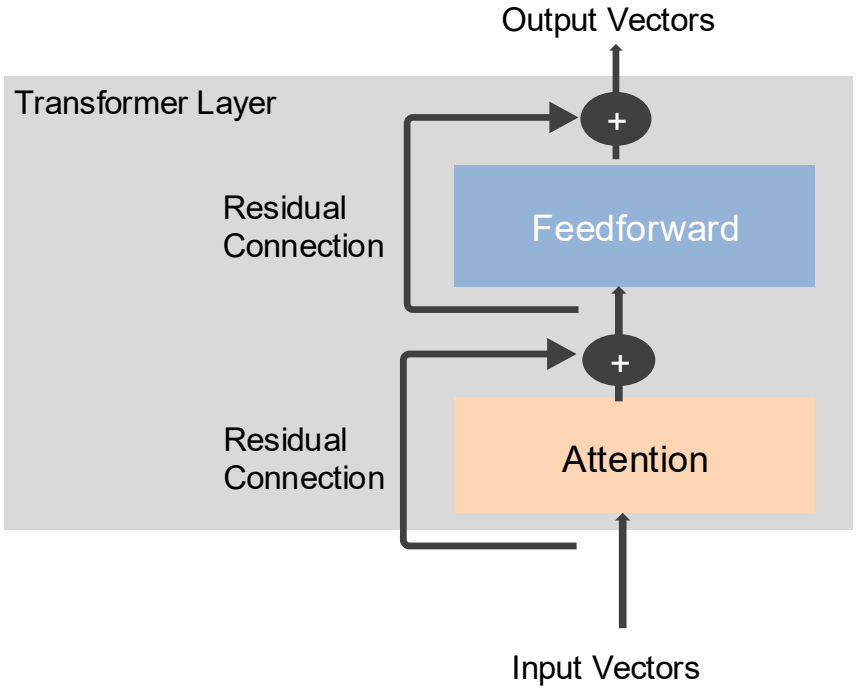


The Transformer Architecture

From Attention to Transformer

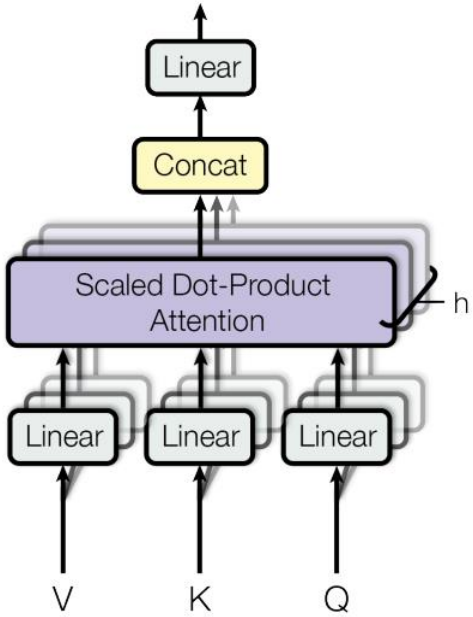
A single layer transformer consists of:

- Attention Mechanism
- Feed-Forward Network
- Residual Connections



Multi-Head Attention

- Outputs combined for richer representations
- Multiple heads learn different relationships (syntax, meaning, position)



Vaswani, A., et al. (2017). Attention Is All You Need

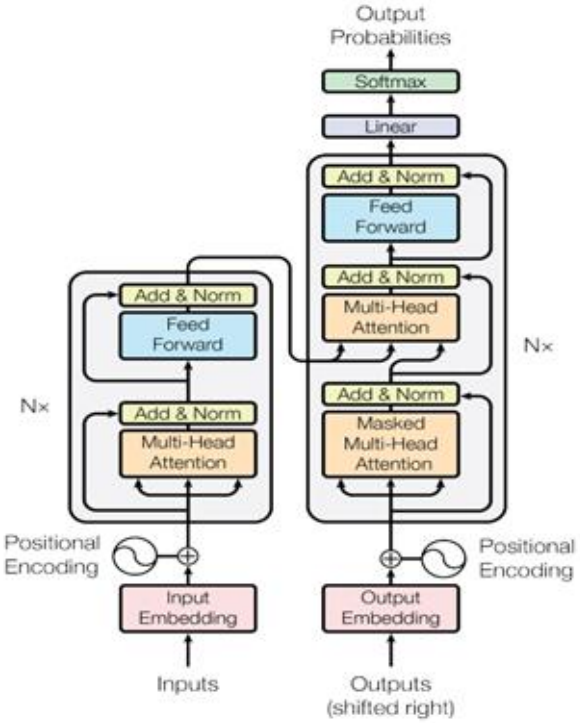
Positional Encoding

- Transformers have no recurrence — order must be added explicitly; otherwise, “dog bit man” and “man bit dog” are processed the same way.

- **Positional Encoding**: Information about the relative or absolute position of the tokens in the sequence (we will skip how to do this in class)

Transformer Architecture

- Encoder–Decoder structure
- **Encoder**: maps an input sequence to a sequence of continuous representations z.
  - Useful for classification
- **Decoder**: Given z, the decoder generates an output sequence of symbols one element at a time.
  - Useful for generation



Vaswani, A., et al. (2017). Attention Is All You Need

Applications

- Language Models: GPT, BERT, T5
- Vision: ViT (Vision Transformer)
- Multimodal: CLIP, DALL·E, GPT-4v
- Scientific: AlphaFold, time-series modeling, robotics, physics

Further Reading/Viewing

- Jurafsky & Martin, Chapter 8
  - https://web.stanford.edu/~jurafsky/slp3/ed3book_aug25.pdf
- Russell & Norvig, Chapter 21.6 and 24

- Andrej Karpathy tutorial
  - https://karpathy.ai/zero-to-hero.html
- 3Blue1Brown:
  - https://www.youtube.com/watch?v=eMlx5fFNoYc
- The Illustrated Transformer
  - https://jalammar.github.io/illustrated-transformer/



