# RNN

$h_t = f(W_x x_t + W_h h_{t-1} + b)$

- sequence modeling architecture for sequential data
  - explicitly models temporal dependencies

- processes inputs step by step in time order
    - hidden state carries information across time steps
    - same weights shared across time
- order-sensitive & variable-length input handling
    - hidden state acts as a compressed summary of the past

## Backpropagation Through Time (BPTT)

- unfold the RNN over time → apply backpropagation across time steps
- prone to vanishing or exploding gradients
    - gradients flow backward through many steps

## Limitations

- vanishing gradients
    - long-range dependencies are hard to learn
- sequential computation
    - no parallelism across time steps
- long-context modeling is inefficient

## LSTM / GRU

- introduce gating mechanisms →mitigate vanishing gradient problem
- still sequential → scalability issues remain

## RNN vs Transformer

- RNN
    - strong inductive bias for sequence order
    - sequential, memory-based

- add an identity (skip) path to the main transformation
- effects
  - optimization becomes easier
    - skip path provides a low-resistance gradient route
  - improves conditioning
    - reduces sensitivity to depth in practice
  - allows deeper models without degradation
- where used
  - ResNet blocks
  - transformers (residual around attention and MLP)
  - diffusion U-Nets (skip connections across scales)

---

# Transformer

- sequence-to-sequence architecture based on attention
  - parallel computation over sequence positions

## Encoder

- input sequence → encoding (mapping) → contextualized representations (context-aware embeddings)
- **Self-attention**
  - each token attends to all other tokens in the input sequence
    - from the same sequence
  - capture global context within representations

## Decoder

- generate the output sequence autoregressively
- **Masked self-attention**
  - prevent attending to future tokens
  - previously generated tokens from the decoder
  - causal mask → preventing future information leakage
- **Cross-attention**
  - give attention over encoder outputs
    - allowing the decoder to condition on the input
  - query: decoder
  - key, value: encoder

## Scaled Dot-Product Attention

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

- Why divide by $\sqrt{d_k}$?
  - dot products grow with dimension → divergence
  - stabilizing gradients & preventing softmax saturation

## Multi-Head Attention

- each head attends to different subspaces → expressiveness ↑
  - captures diverse relationships & different representation subspaces simultaneously

## Attention Complexity

- computation: $O(n^2 d)$
  - for calculating $QK^T$
- memory: $O(n^2)$

    - building $n \times n$ attention weights (matrix)
- how to raovid long-context approaches
    - linear attention
    - sparse attention (local + global)
    - sliding window
    - flash attention

## Linear Attention

$\text{softmax}(QK^T) \approx \phi(Q)\phi(K)^T$

- rewriting attention using a kernel feature map
    - avoid forming $n \times n$ matrix ($QK^T$) → complexity ↓ (quadratic → linear)
    - at the cost of approximating softmax attention
- asymptotic complexity: $O(nd^2) \approx O(n)$ ($n >> d$)
    - first computing $\phi(K)^T V \rightarrow O(nd^2)$ ($d \times d$ matrix)
    - then calculating $\phi(Q) \cdot (\cdot) \rightarrow O(nd^2)$
- drawbacks: approximated softmax attention result

## Sliding Window Attention

- each token attends only to a local window of neighboring tokens
    - limiting receptive field → reducing qudratic attention cost
- pros: preserves local context well
- cons
    - cannot directly capture long-range dependencies
    - global information may be lost

## Positional Encoding

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

- transformer: permutation-invariant (no notion of order) → inject positional information
  - added to token embeddings as an additive bias
- sinusoidal positional encoding
  - deterministic & parameter-free
  - encode absolute position only
    - relative distance → not explicitly modeled in attention scores
  - low-dimension pair → low frequency

## Rotary Positional Encoding (RoPE)

$$\begin{pmatrix} x'_{2i} \\ x'_{2i+1} \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x_{2i} \\ x_{2i+1} \end{pmatrix}$$

$$z_i(p) = z_i \cdot e^{i\theta_i p}$$

- directly and naturally encode relative positional information
  - rotating each pair of dimensions in query and key vectors
    - represent pairs of embedding dimensions as complex numbers
  - position-dependent rotations using Euler's formula
    - dot products between rotated queries and keys
    - dependent on relative position (not absolute position)
  - not into input token embedding
- better extrapolation to long sequences
  - widely used in modern LLMs

---

# Vision Transformer

1) patchify 2) transformer encoder-only 3) [CLS] token

- applies the Transformer encoder to images
  - image → sequence of patch tokens
- replaces convolution with self-attention
  - global context modeling from the first layer
- **Key trade-off**
  - global modeling & scalability
  - weaker inductive bias → large data required
    - **Inductive bias**: what a model assumes is likely to be true about the world
      - assumptions a model makes about the structure of the data before seeing any data

## Patch embedding

- split image into fixed-size patches
- flatten + linear projection → tokens

## Positional encoding

- inject spatial information into patch embeddings
- required due to permutation-invariant attention

## `[CLS]` token

- prepend a learnable classification token to the patch sequence
- participates in self-attention with all patches
  - aggregates global image information across layers
- final classification head applied to the `[CLS]` representation

---