Learning From Data
Lecture 12: Deep Learning Architectures for Computer Vision and Natural Language Processing

Yang Li yangli@sz.tsinghua.edu.cn

TBSI

December 13, 2024

Final Poster Session Information

When & Where

Time: December 27 9:00am-12:00pm

Location: TBA

What to include in the poster?

- **Abstract**: a short summary of your work (no more than 100 words)
- **Introduction/Motivation**: why is this problem important and what is your contribution?
- **Method**: the machine learning methodology used
- **Results**: the dataset you use and the experimental results (tables and figures)
- **Conclusion/Discussion**: conclude your technical/application contributions
- **Reference**: include 2-3 important references (You can use smaller fonts for this part.)

Each group needs to submit a poster in PDF format of A0-size to Web Learning before **December 25 12:00pm (noon)**.

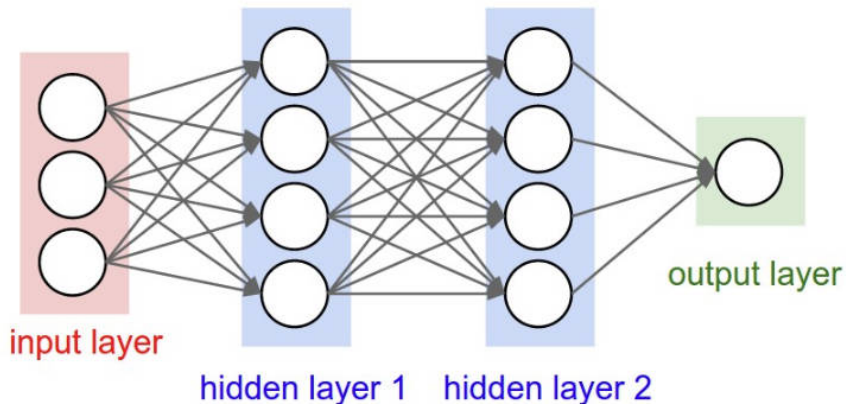
Today's Lecture

- Convolutional Neural Networks (CNN)
- Residual Neural Networks (ResNet)
- Recursive Neural Network (RNN)
- Attention & Transformer

Convoluntional Neural Networks

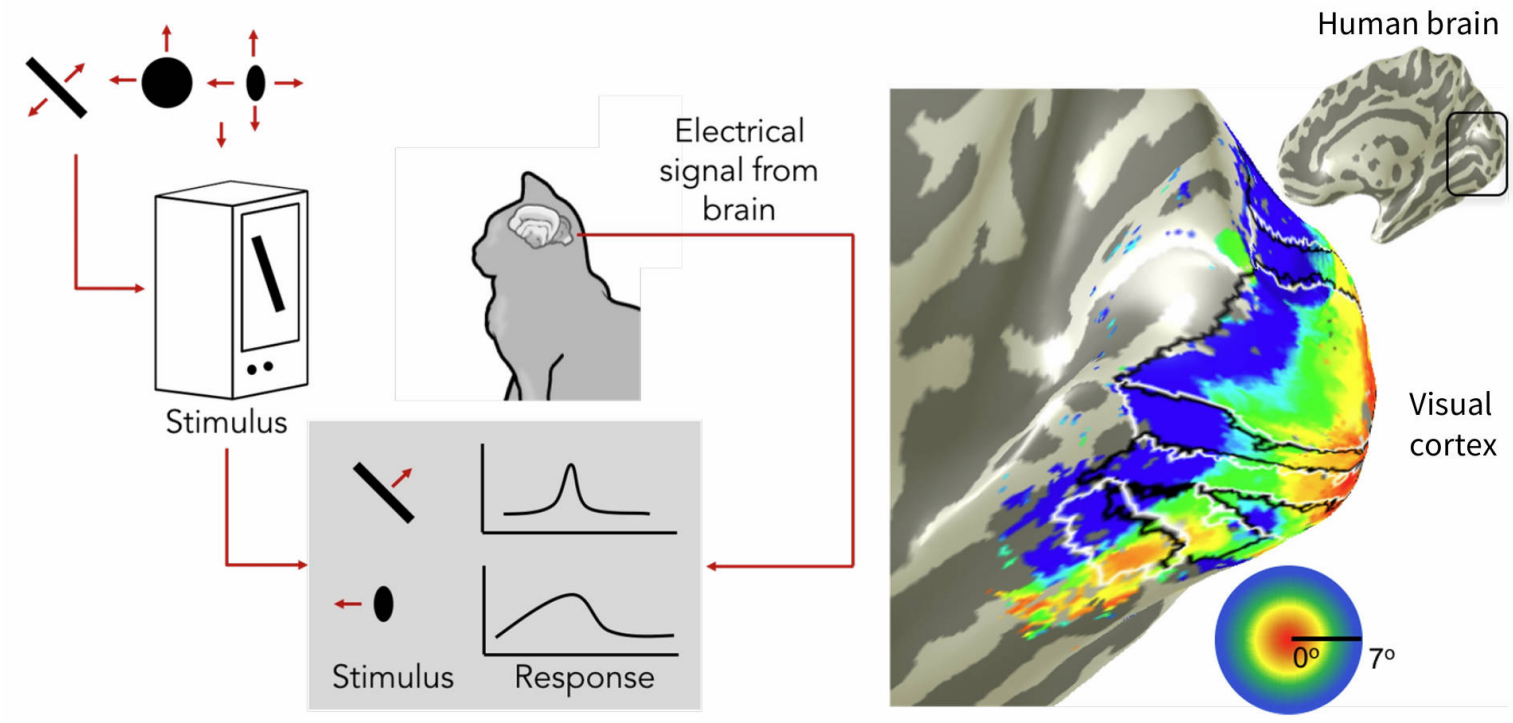
Limitations of MLPs

1. **Large Parameters**: MLPs have many parameters, increasing the risk of overfitting.
2. **Not translation invariant**: MLPs are sensitive to input shifts, which can affect their performance.
3. **Inefficient Computation**: MLPs are less efficient with large data, leading to longer training times.



A bit history of CNN

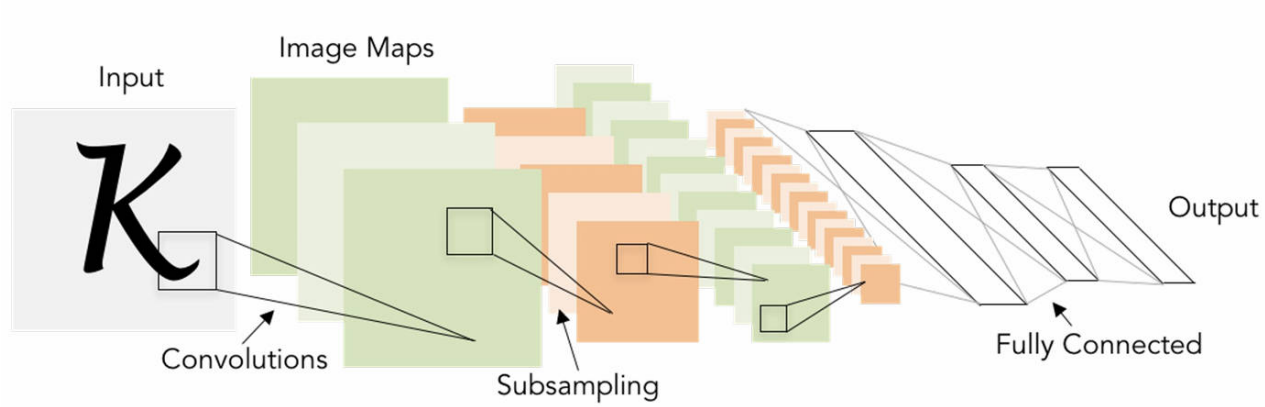
In the biological visual system, individual neurons are sensitive to visual information only in a specific region of the visual field, a.k.a. the receptive field. [1]



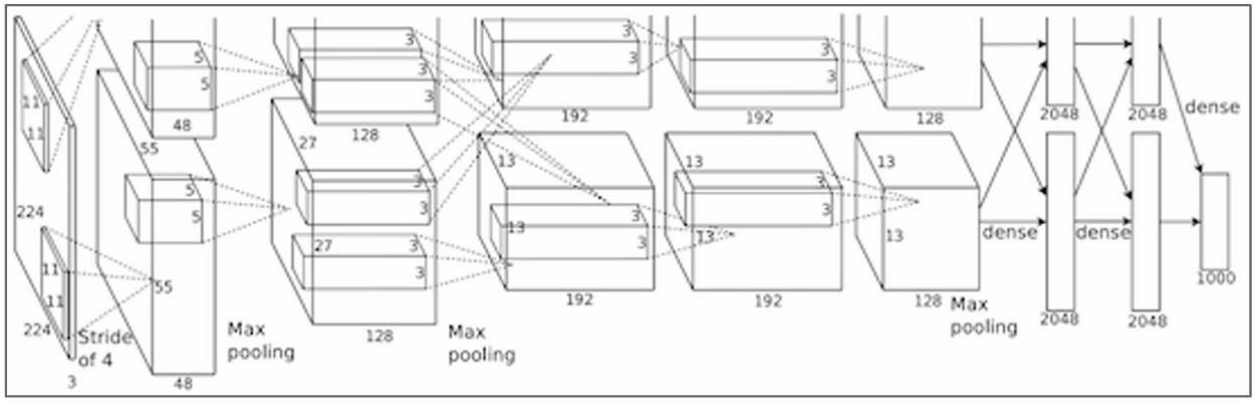
[1] Hubel, D. H., & Wiesel, T. N. (1962). Receptive Fields, Binocular Interaction and Functional Architecture in the Cat's Visual Cortex. Journal of Physiology, 160(1), 106-154.

A bit history of CNN

- LeNet-5: Early CNN architecture [2]



- AlexNet: Revolutionized computer vision by demonstrating the power of deep learning. [3]

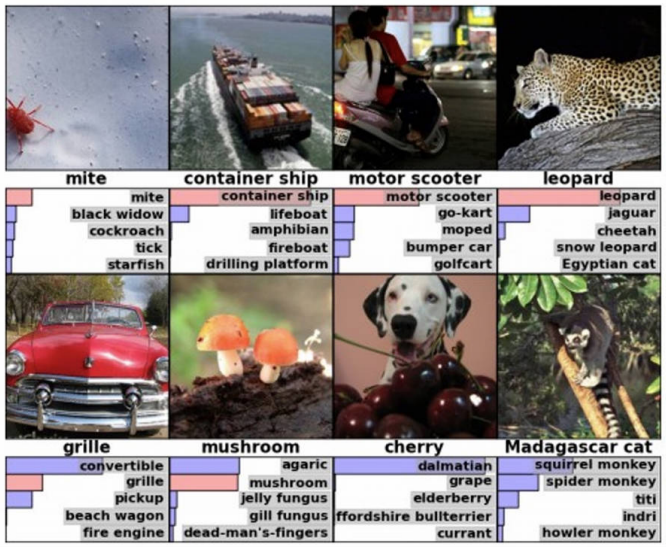


[2] Lecun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. (1998). "Gradient-based learning applied to document recognition" . Proceedings of the IEEE. 86 (11)

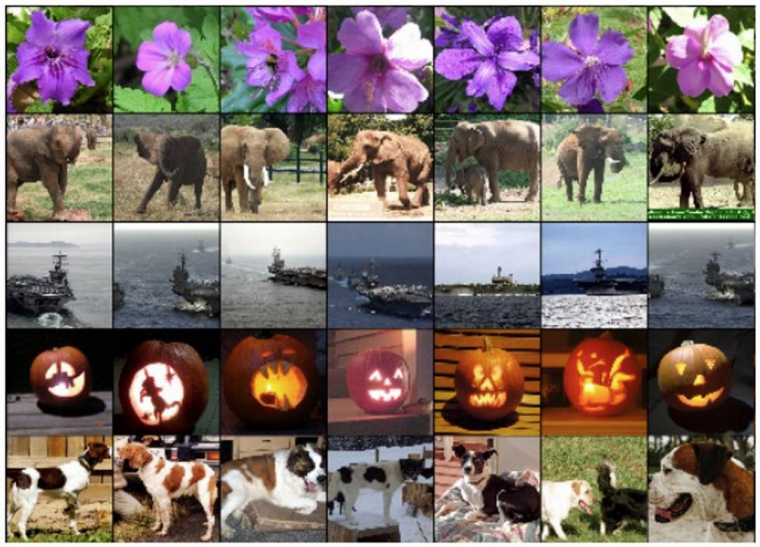
[3] Krizhevsky, Alex; Sutskever, Ilya; Hinton, Geoffrey E. (2017-05-24). "ImageNet classification with deep convolutional neural networks" . Communications of the ACM. 60 (6)

Fast-forward: ConvNets are everywhere

Classification

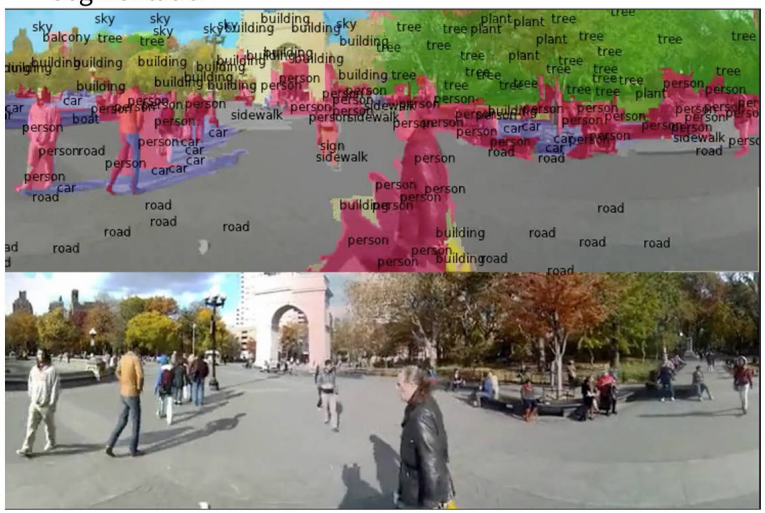


Retrieval



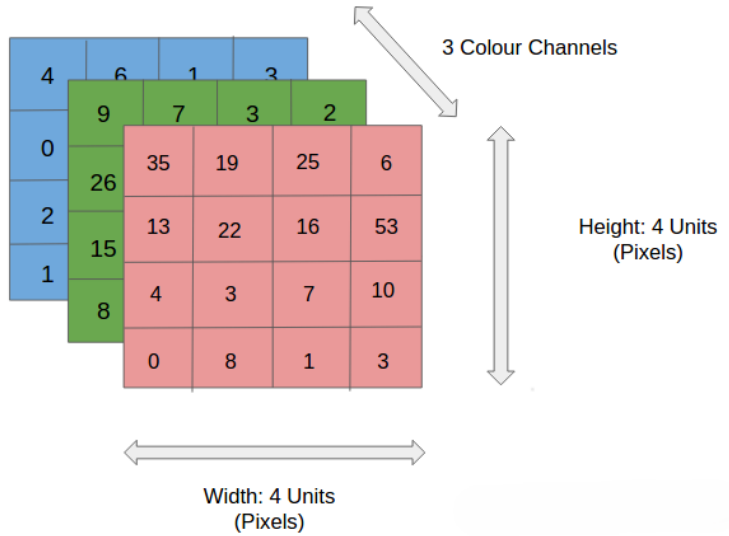
Detection

Segmentation



Convoluntional Layer

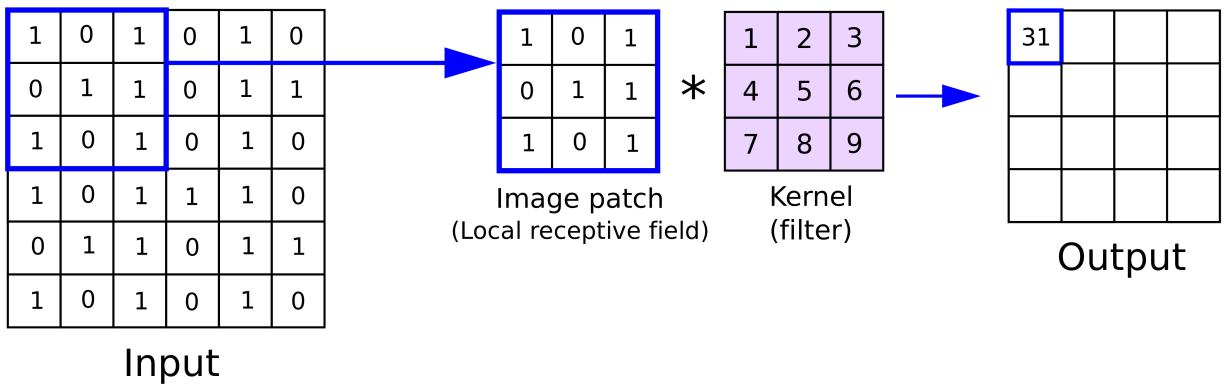
- Input: RGB image (3 channels)



- Filter/Kernel:
  Small learnable matrix W (e.g., 3×3, 5×5)
  Slides over input to detect features

Convoluntional Layer

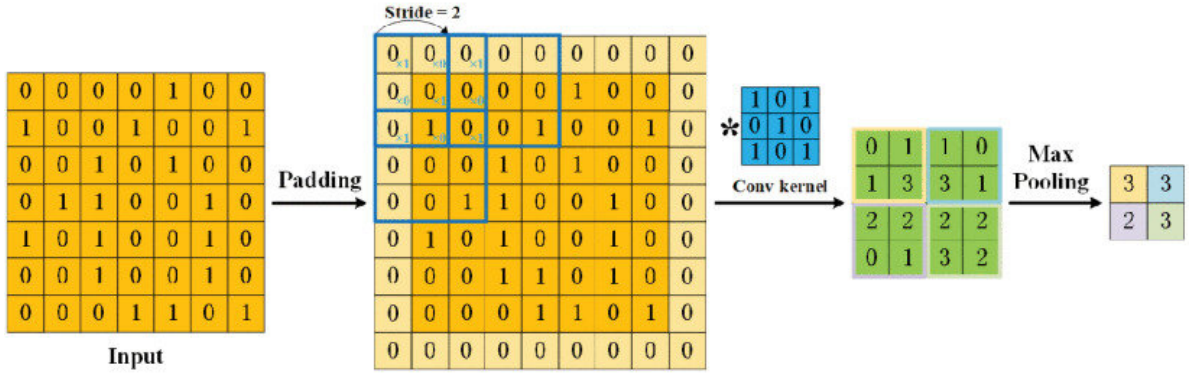
Convolve the filter with the image
slide over the image spatially, computing dot products



$$y_{i,j} = \sum_{m=0}^{k_h-1} \sum_{n=0}^{k_w-1} x_{i+m,j+n} \cdot w_{m,n} + b$$

Convoluntional Layer

- **Stride**: Number of pixels to shift the filter *Controls how filter moves across input Affects output spatial dimensions*

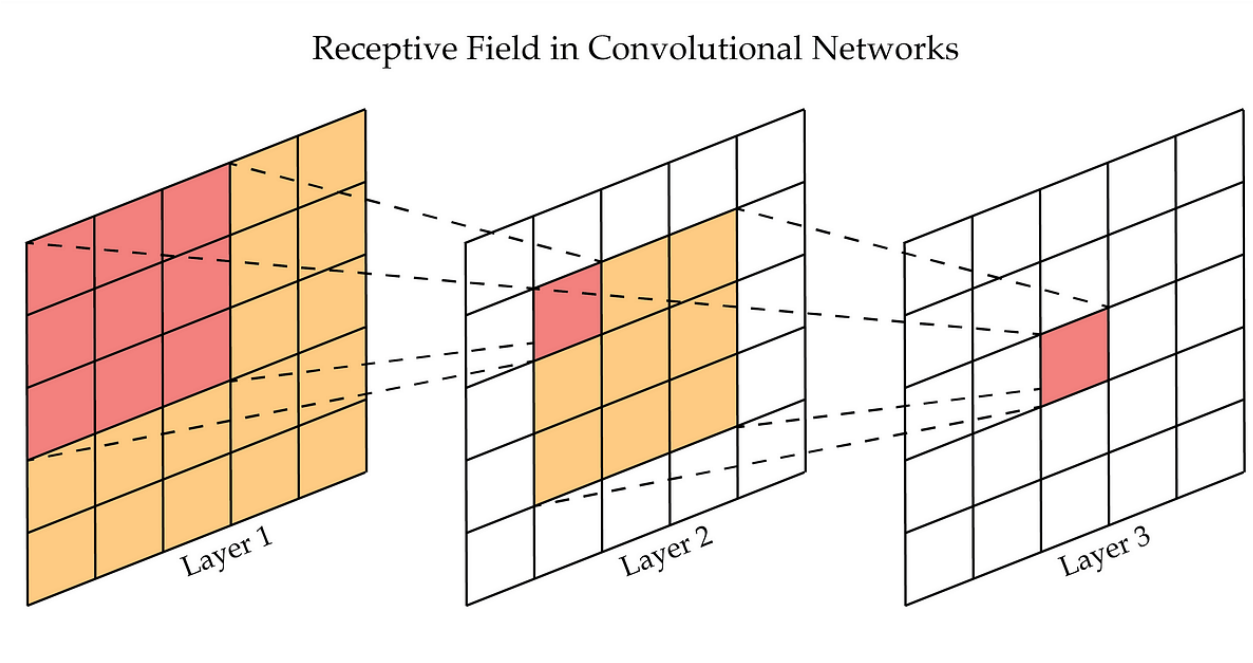


- **Padding**: Zero-padding around input borders *Preserves spatial dimensions*
- Output Feature Map Size:

$$H_{out} = \left\lfloor \frac{H_{in} - K + 2P}{S} \right\rfloor + 1$$

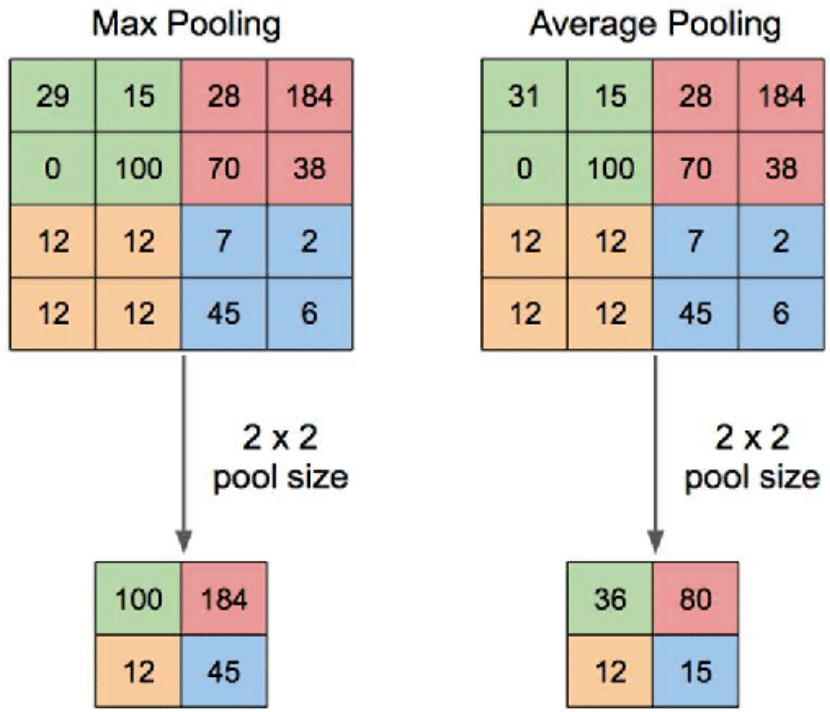
Convoluntional Layer

Receptive Field

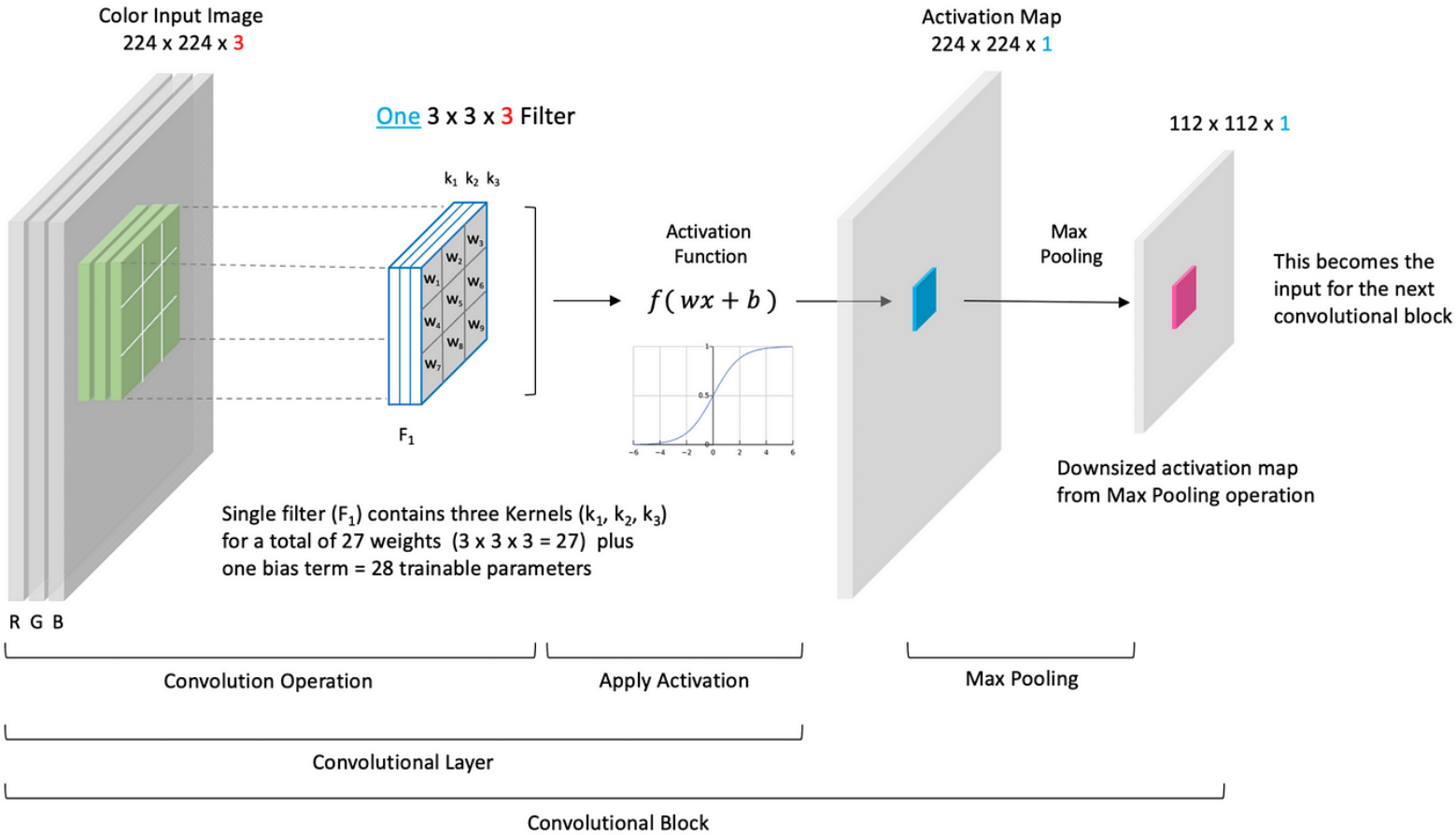


Pooling Layer

- Types of Pooling:
  Max Pooling: Takes maximum value in window
  Average Pooling: Takes average value in window
- Down-sampling Operation:
  Reduces spatial dimensions
  Maintains important features
  Makes features more robust to position changes

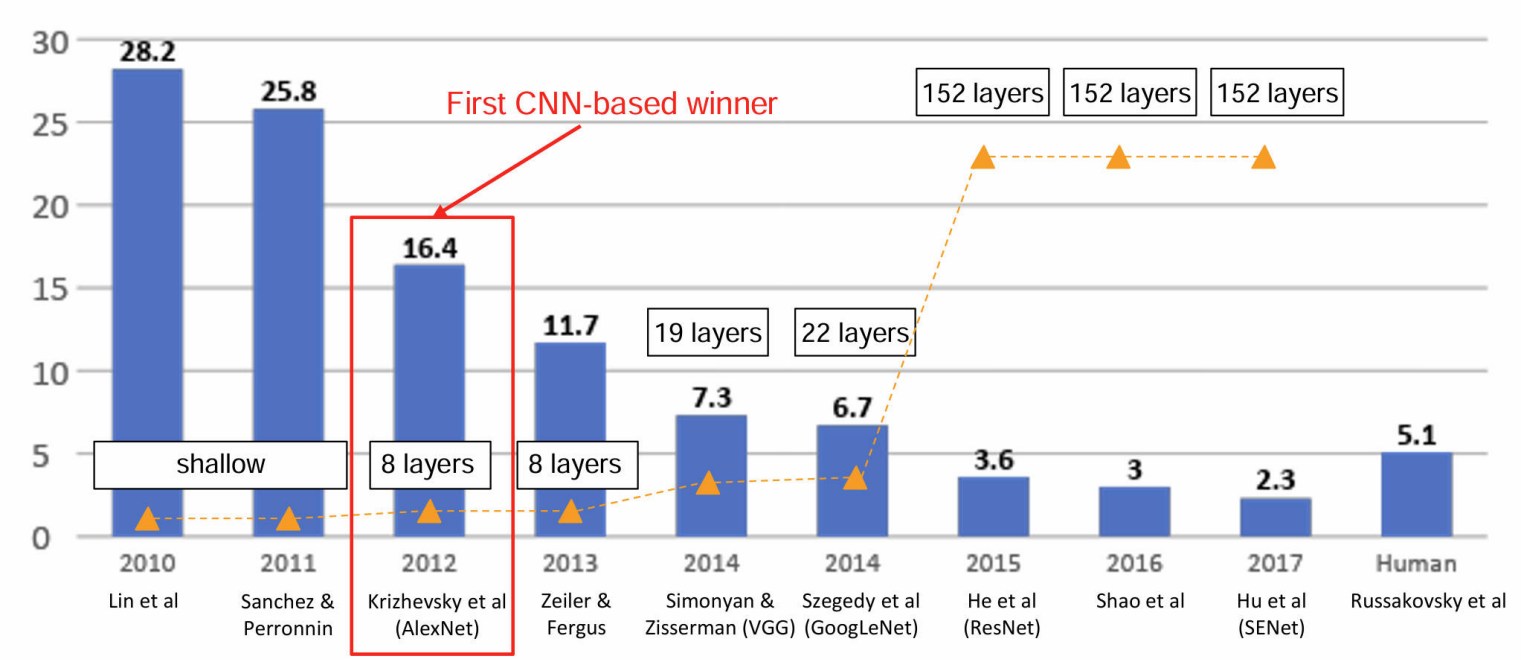


CNN Architectures



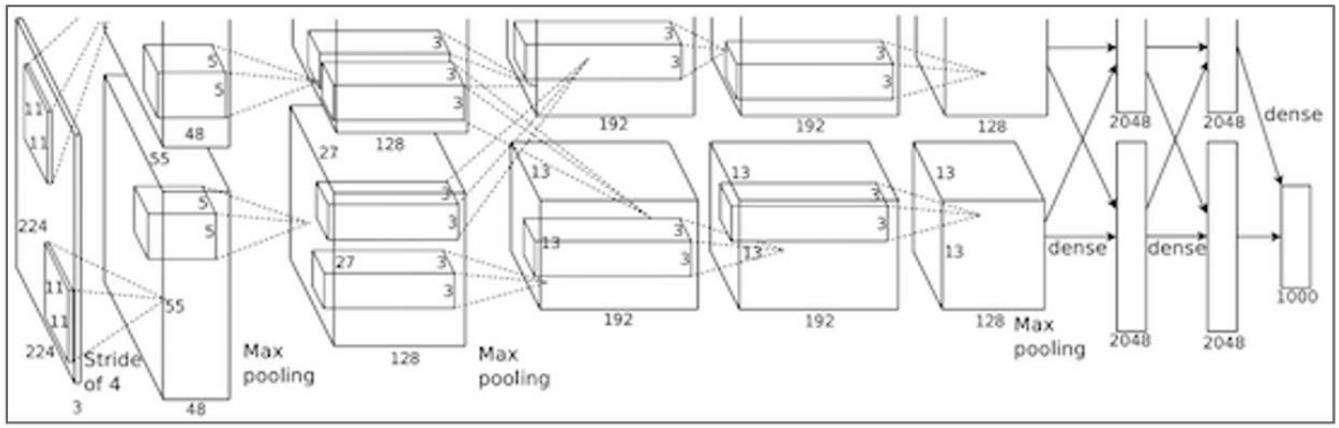
CNN Architectures

- AlexNet, VGGNet, ResNet...



ImageNet classification challenge winners from 2010 to 2017

Case Study: AlexNet



Input:
[227x227x3] INPUT
[55x55x96] **CONV1:** 96 11×11 filters at stride 4, pad 0
[27x27x96] **MAX POOL1:** 3×3 filters at stride 2
[27x27x96] **NORM1:** Normalization layer
[27x27x256] **CONV2:** 256 5×5 filters at stride 1, pad 2
[13x13x256] **MAX POOL2:** 3×3 filters at stride 2
[13x13x256] **NORM2:** Normalization layer
[13x13x384] **CONV3:** 384 3×3 filters at stride 1, pad 1
[13x13x384] **CONV4:** 384 3×3 filters at stride 1, pad 1
[13x13x256] **CONV5:** 256 3×3 filters at stride 1, pad 1
[6x6x256] **MAX POOL3:** 3×3 filters at stride 2
[4096] **FC6:** 4096 neurons
[4096] **FC7:** 4096 neurons
[1000] **FC8:** 1000 neurons (class scores)

Case Study: VGGNet

Small filters, Deeper networks

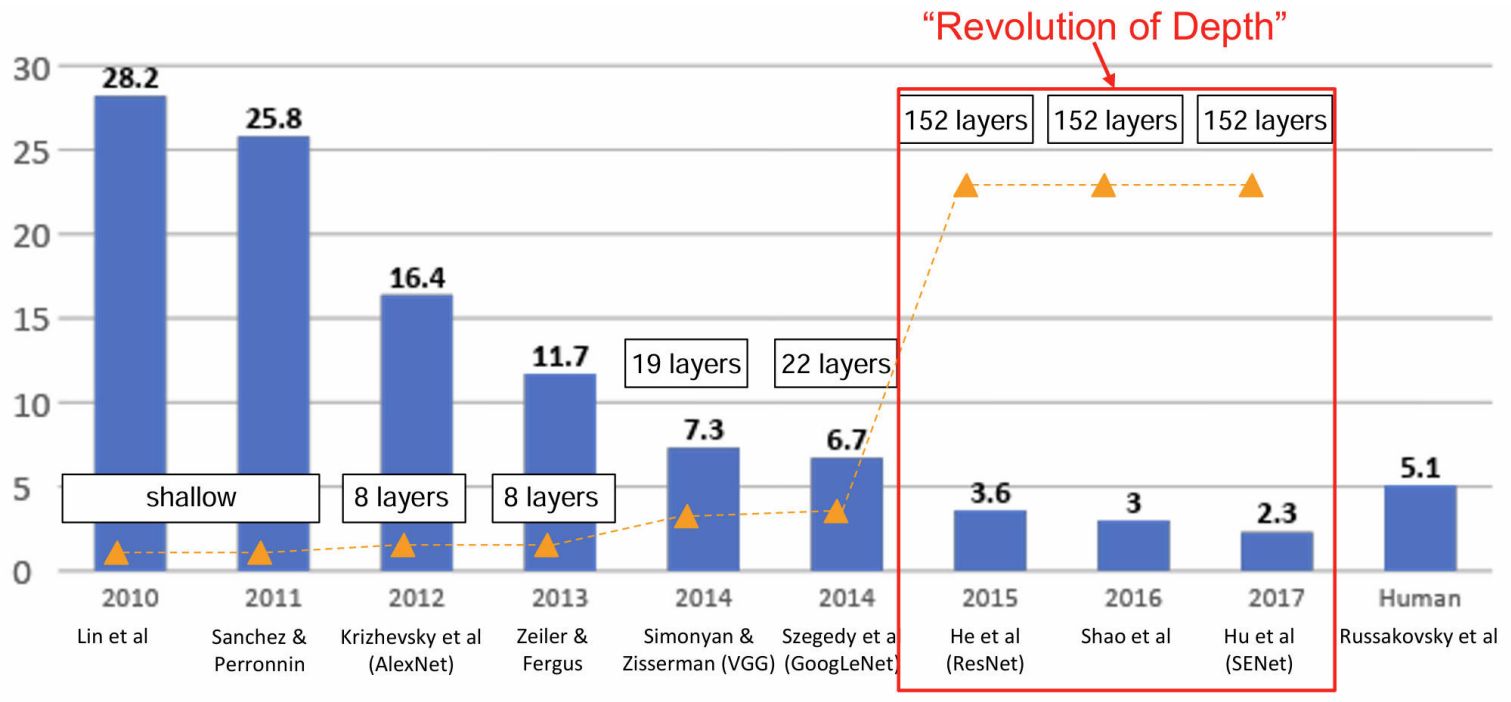
8 layers (AlexNet)
-> 16 - 19 layers (VGG16Net)

Only 3x3 CONV stride 1, pad 1
and 2x2 MAX POOL stride 2

11.7% top 5 error in ILSVRC'13
(ZFNet)
-> 7.3% top 5 error in ILSVRC'14

CNN Architectures

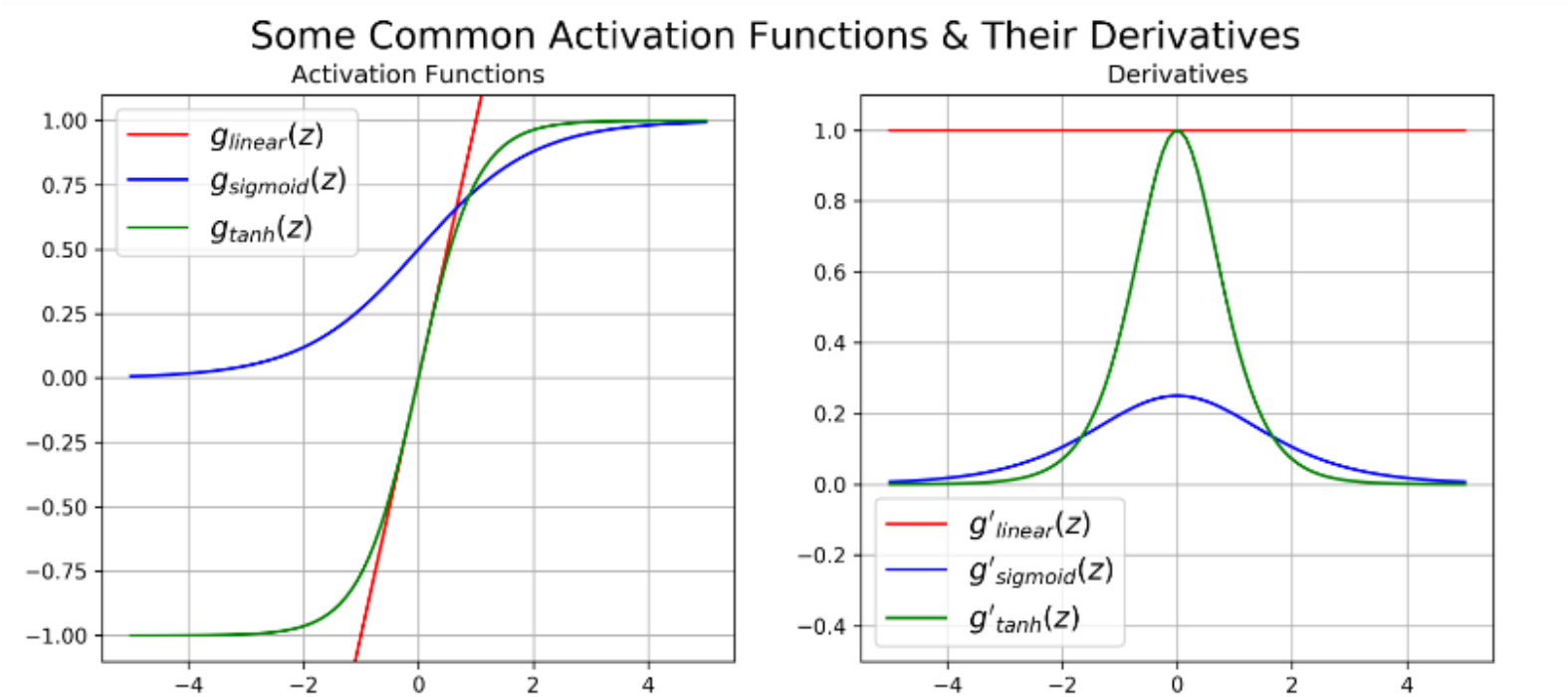
- Deeper network? - ResNet



Residual Neural Network

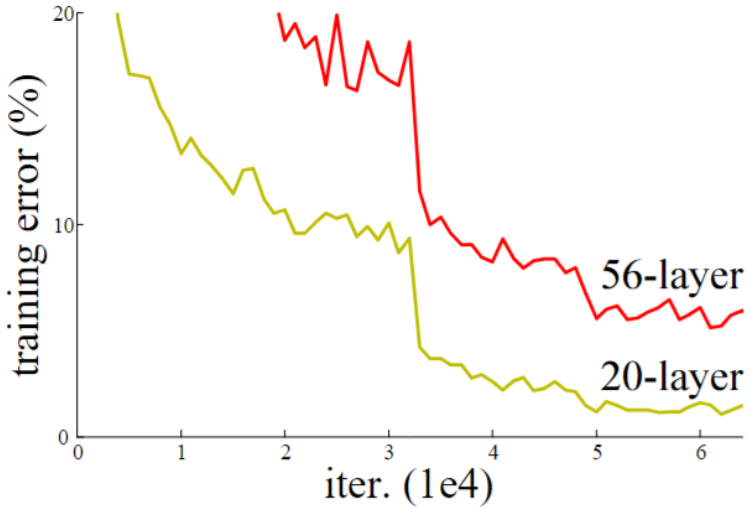
Motivation for Resnet

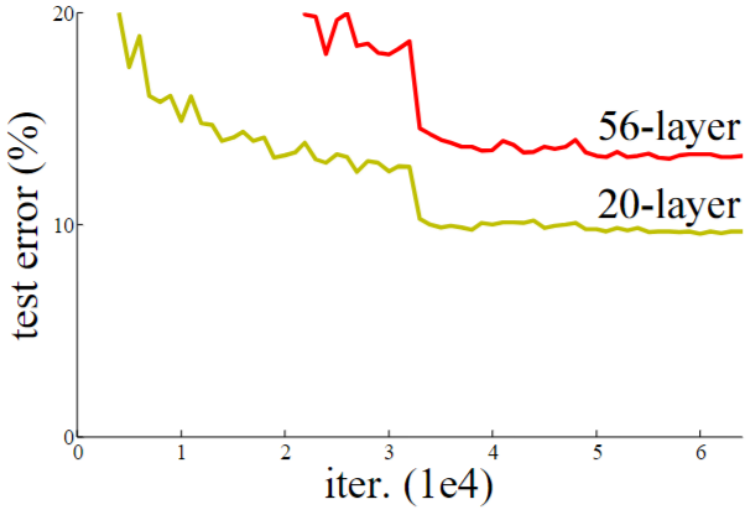
- **Vanishing Gradient Problem in Deep Networks.** Traditional deep neural networks struggle with the vanishing gradient problem as layers increase, leading to degraded performance during training.
- This can be addressed by:
  – Normalized Initialization
  – Batch Normalization
  – Appropriate activation function



Motivation for Resnet

- **Performance Saturation.**
  As network depth increases, the performance on both training and testing datasets tends to saturate or even degrade, highlighting the limitations in optimization in very deep networks.

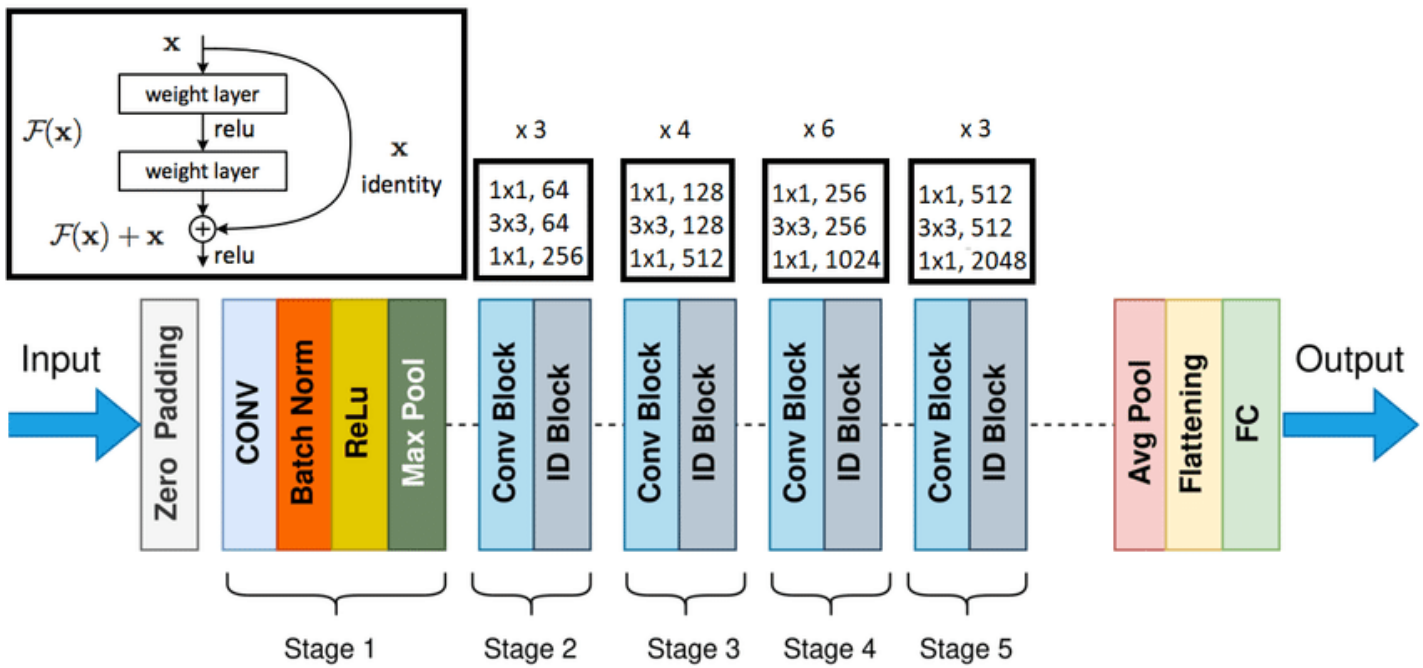




56-layer Cifar model has higher training error and test error than 20-layer model.

Network Design

- **Design Philosophy**
  Keep it simple, just deeper!
- **Residual Connections**
  Allow the network to "skip" certain layers, helping prevent the vanishing gradient problem, and enabling the training of much deeper networks.



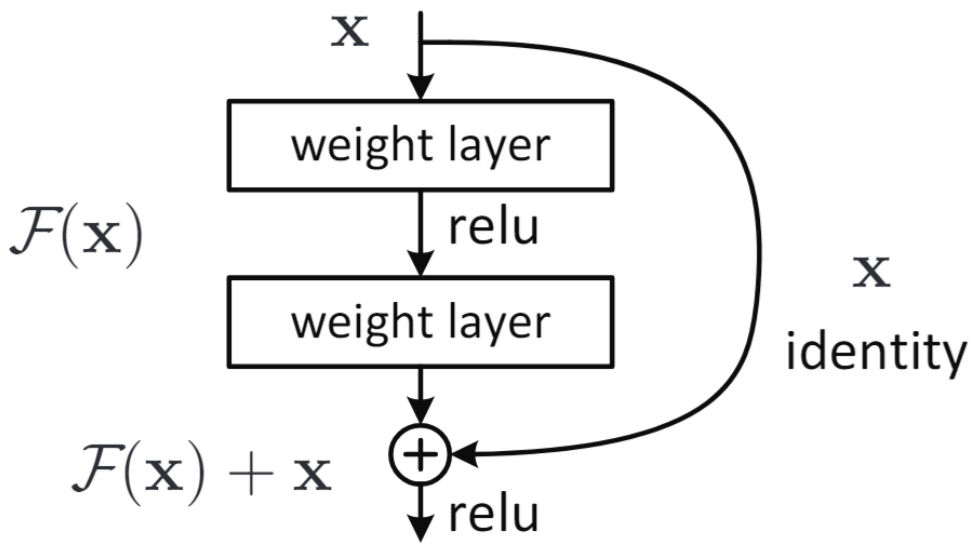
The architecture of resnet50 model.

Theorem (Residual Learning)

Given a deep neural network with input $\mathbf{x}$ *and a desired mapping* $H(\mathbf{x})$*, ResNet reformulates the mapping as:*

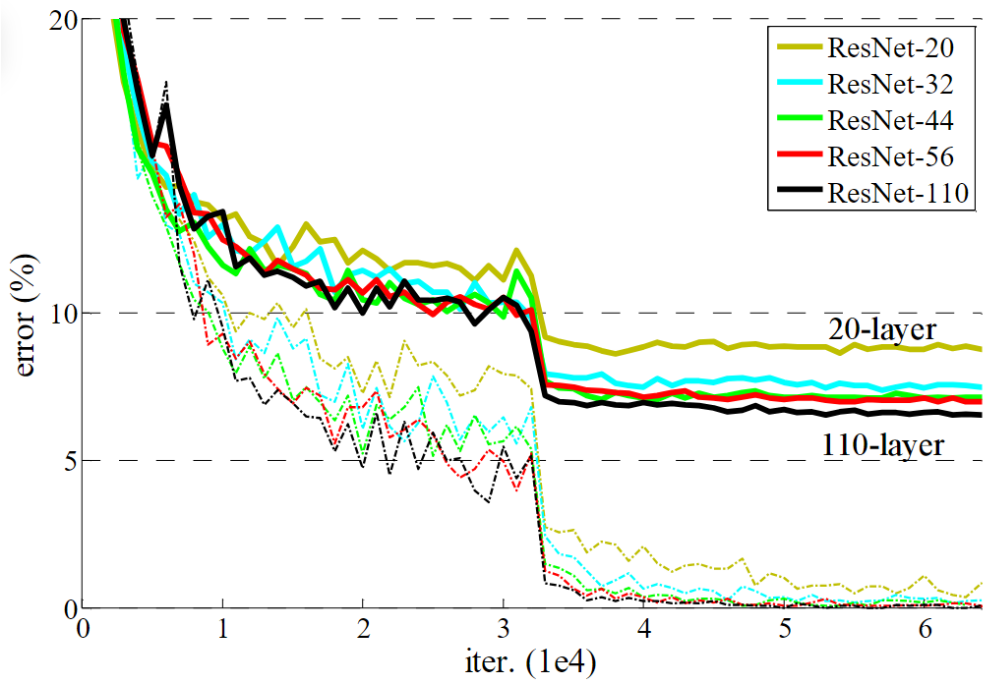
$$H(\mathbf{x}) = F(\mathbf{x}) + \mathbf{x}$$

where $F(\mathbf{x})$ *is the* **residual function** *to be learned. This structure facilitates gradient flow during backpropagation by allowing the network to learn residuals* $F(\mathbf{x})$ *instead of the full mapping* $H(\mathbf{x})$*.*



Depth and Optimization

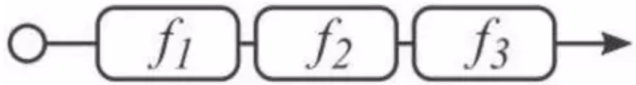
- Stacking more layers while maintaining residual connections ensures that additional layers optimize at least as well as a shallower counterpart.
- if $F(\mathbf{x}) = 0$ for additional layers, the residual structure guarantees that the output is unchanged, preventing performance degradation with increasing depth.



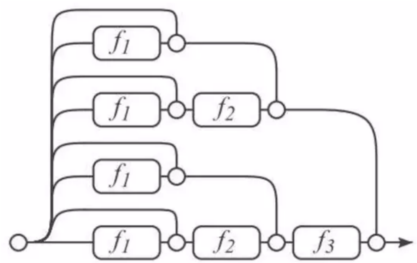
The 50-layer, 101-layer, and 152-layer ResNets are progressively better. The model enjoys significant accuracy gains from increased depth.

Compare with traditional Neural Network

- In traditional Netural Network, each layer only depends on the previous layer.



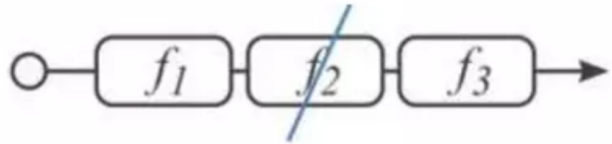
In ResNet, data flows along many paths from input to output. Each path is a unique configuration of which residual module to enter and which to skip.



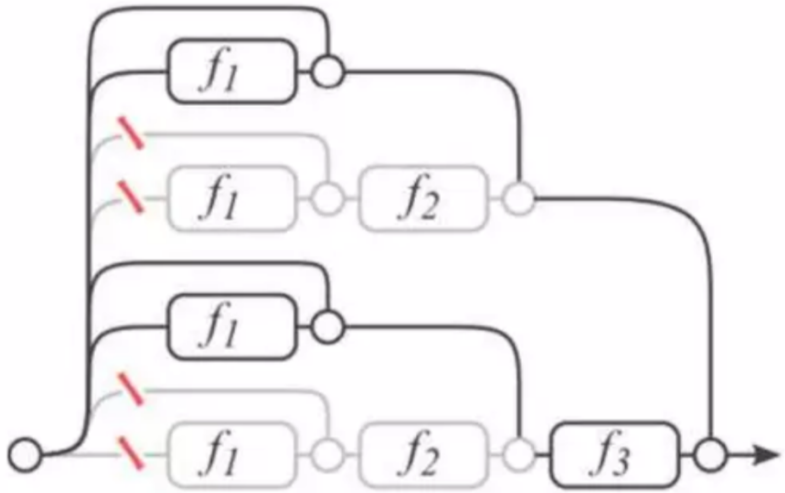
Feedforward in resnet.

Compare with traditional Neural Network

- In ordinary feed-forward networks such as VGG or AlexNet, deleting individual layers alters the only viable path from input to output.



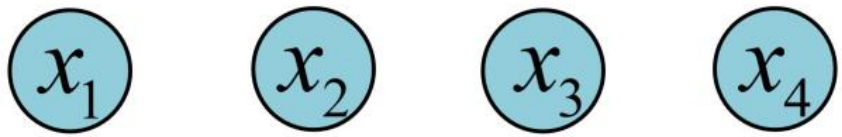
Deleting a layer in residual networks is equivalent to zeroing half of the paths



Recurrent Neural Network

Motivation

Using their internal state (memory) to process sequences of inputs.



- Natural Language Processing (NLP). x_1 can be regarded as the first word, x_2 can be regarded as the second word, and so on.
- Speech Processing. At this time, x_1, x_2, x_3, ... are the sound signals of each frame.
- Time Series Analysis. For example, daily stock prices, etc.

RNN Formulation[4]

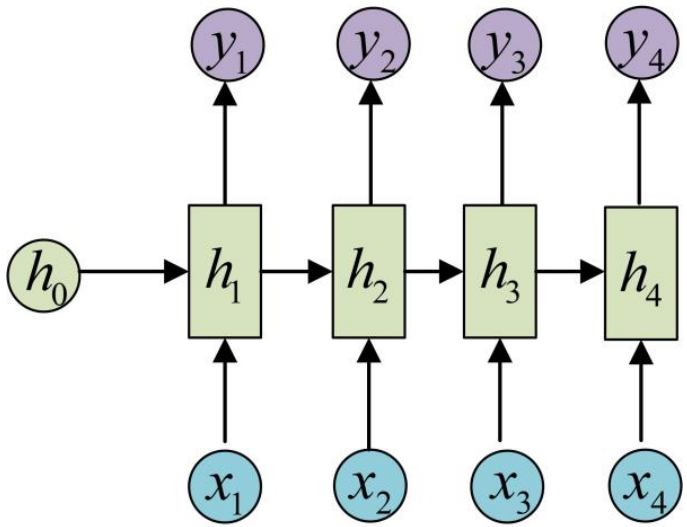
The formula for the current state:

$$h_t = f(h_{t-1}, x_t). \tag{1}$$

And for the output:

$$y_t = O(h_t), \tag{2}$$

where h_t denotes the hidden state (memory), x_t denotes the input and y_t denotes the output.



[4]Rumelhart, David E.; Hinton, Geoffrey E.; Williams, Ronald J. (October 1986). "Learning representations by back-propagating errors". Nature. 323 (6088): 533–536.

Backpropagation Through Time (BPTT)

Overview: Backpropagation Through Time is a method used to train Recurrent Neural Networks (RNNs) by unfolding the network across time steps.

Mathematics:

- The total loss across T time steps:

$$\mathcal{L} = \sum_{t=1}^{T} \mathcal{L}_t, \tag{3}$$

  where $\mathcal{L}_t$ is the loss at time step t.

- Gradients are computed using the chain rule:

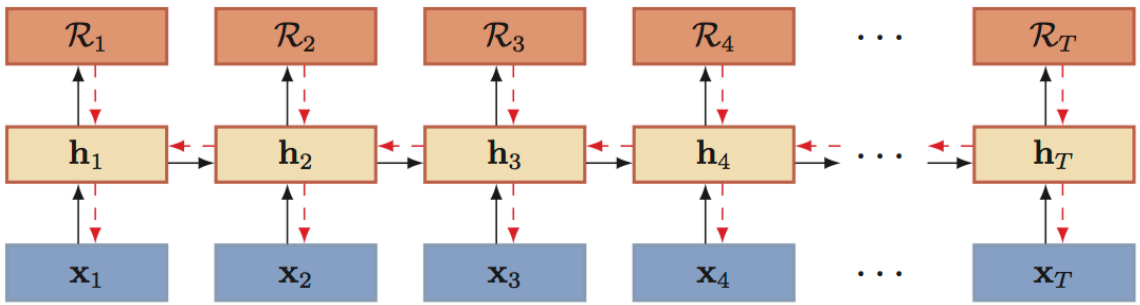
$$\frac{\partial \mathcal{L}}{\partial W} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}_t}{\partial h_t} \frac{\partial h_t}{\partial W}, \tag{4}$$

  where W represents the weights.

Backpropagation Through Time (BPTT)

Steps:

1. **Unfold the RNN:** Represent the RNN as a feedforward network across T time steps.
2. **Forward Pass:** Compute the hidden states h_t and outputs y_t for all time steps.
3. **Backward Pass:**
   - Compute the gradients of the loss function $\mathcal{L}$ with respect to outputs y_t.
   - Propagate the gradients back through time to update the weights.



Drawbacks of Traditional RNNs

Challenges:

- **Vanishing Gradients:** Gradients can shrink exponentially as they are propagated back through many time steps, making it difficult to learn long-term dependencies.
- **Exploding Gradients:** Conversely, gradients can grow exponentially, causing instability during training.
- **Short-Term Memory:** Traditional RNNs struggle to retain information over long sequences due to their design.
- **Difficulty Handling Long Sequences:** The computational cost and inability to capture long-term dependencies limit their effectiveness.

Solution: Long Short-Term Memory (LSTM)

- Designed to address these issues by introducing a memory cell structure.
- Enables learning of long-term dependencies effectively.

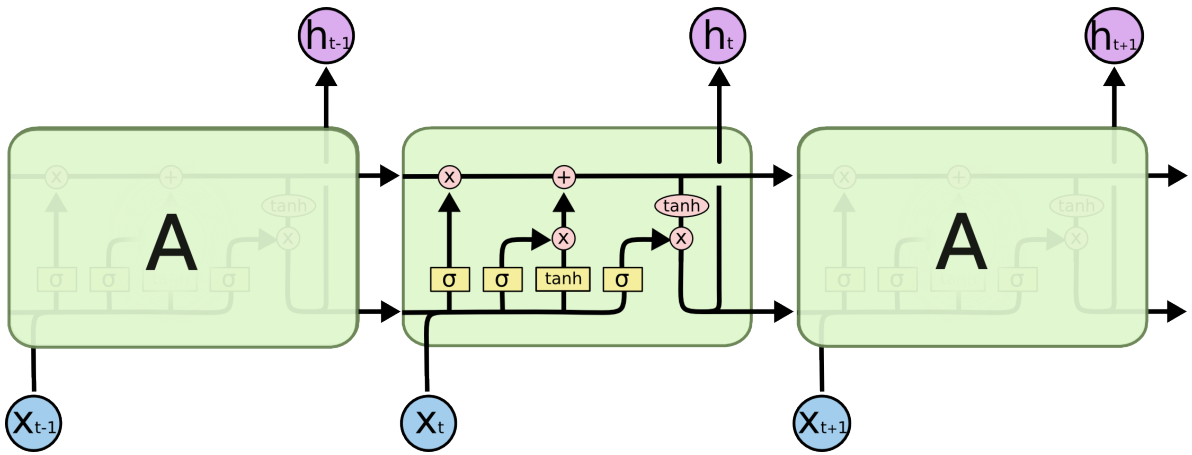
Long Short-Term Memory (LSTM)[5]

Motivation:

- Designed to overcome the limitations of traditional RNNs.
- Introduces mechanisms to maintain long-term dependencies effectively.

Key Idea:

- Use a **cell state** and **gates** to control the flow of information.
- Allows the network to decide what to keep, update, and output.



[5] Hochreiter, Sepp; Schmidhuber, Jürgen (1997-11-01). "Long Short-Term Memory". Neural Computation. 9 (8): 1735–1780

LSTM Architecture: Cell State and Gates

Components of LSTM:

- **Cell State (C_t):** The memory of the network that carries information across time steps.
- **Forget Gate:**

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{5}$$

  Determines what information to discard from the cell state.

- **Input Gate:**

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{6}$$

  Decides what new information to add to the cell state.

LSTM Architecture: Updates and Outputs

Further Components:

- **Candidate Values:**

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{7}$$

New candidate values to update the cell state.

- **Cell State Update:**

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \tag{8}$$

Combines forget gate, input gate, and candidate values.

- **Output Gate:**

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{9}$$

Decides the output based on the cell state.

- **Hidden State (h_t):**

$$h_t = o_t \odot \tanh(C_t) \tag{10}$$

Represents the output of the LSTM at time t.

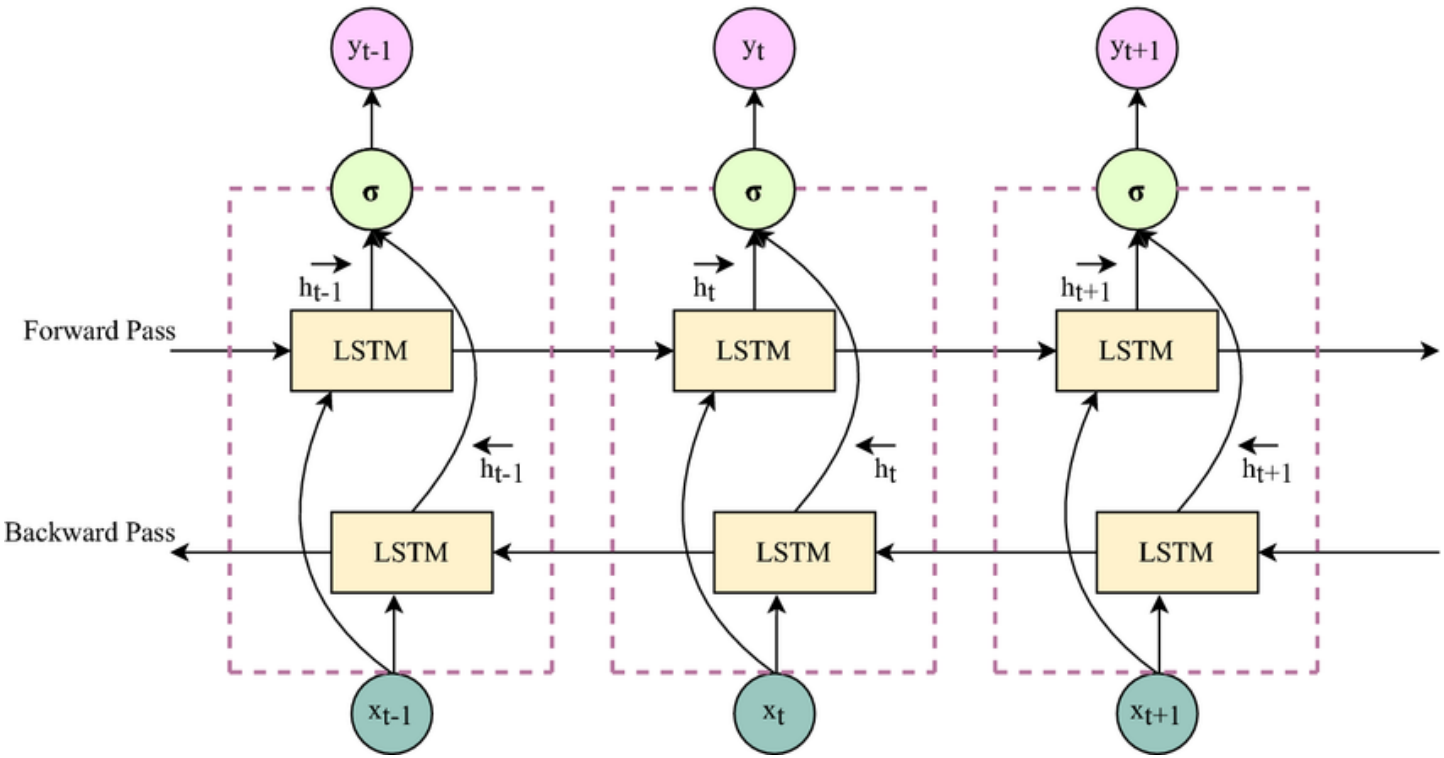
Advantages of LSTM

Why LSTM is Powerful:

- Effectively handles vanishing and exploding gradients.
- Retains information over long sequences.
- Adaptable for various tasks such as:
  - Sequence prediction (e.g., text generation).
  - Language modeling (e.g., machine translation).
  - Time-series analysis (e.g., stock price prediction).
- Flexible architecture allows stacking for more complexity.

Bidirectional RNN (BRNN)

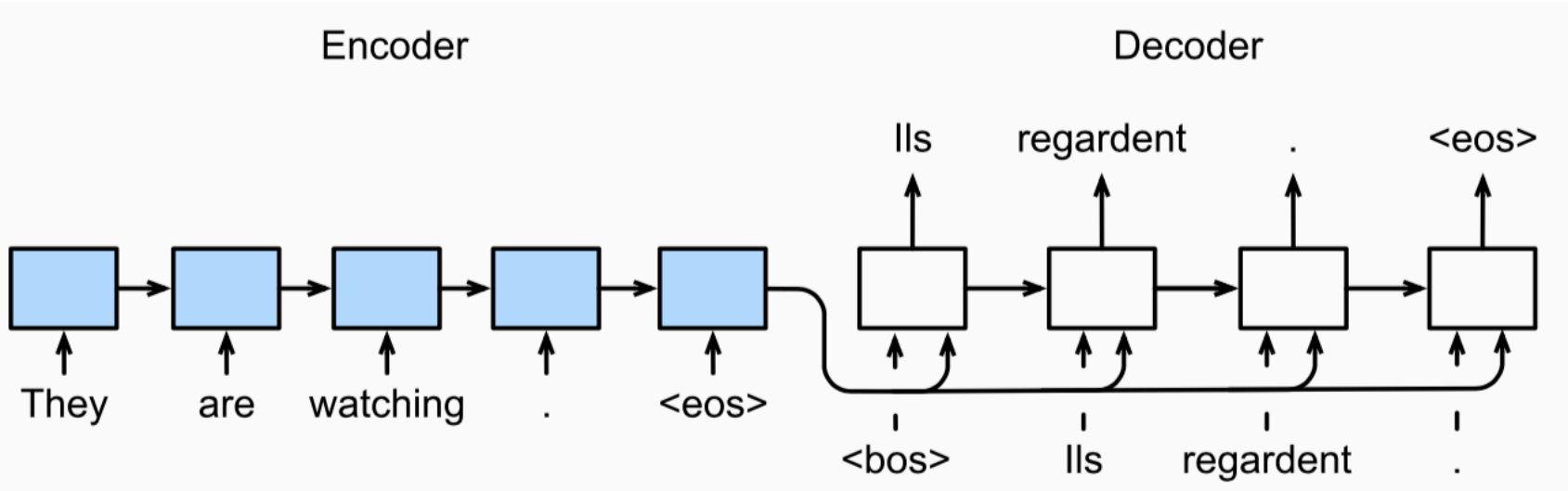
- Uses two RNNs that processe the same input in opposite directions
- BiLSTM Revolutionize speech recognition, machine translation, language modeling at around 2006 [6]



[6] A. Graves and J. Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. Neural Networks, August 2005

Sequence-to-Sequence (Seq2Seq) Models

- Employs two RNNs (e.g. LSTM), an "encoder" and a "decoder", for sequence-to-sequence prediction
- Current state-of-the-art model for NLP, e.g. machine translation



Sequence to sequence model for machine translation. <BOS> and <EOS> tokens mark the beginning and end of a sentence.

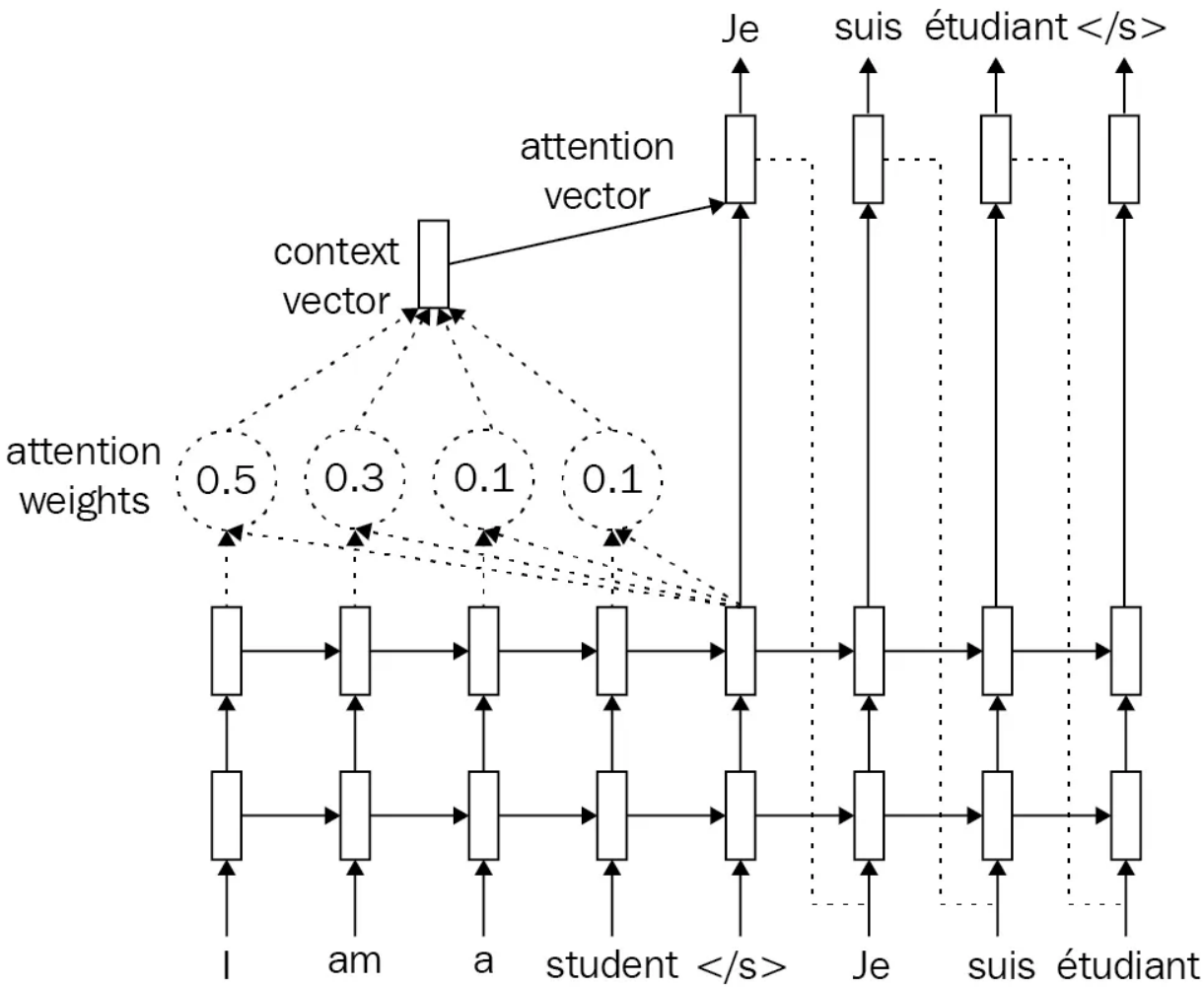
Attention Mechanism & Transformers

Motivation: Limitations of Classic Encoder-Decoder Models

- **Long-Range Dependencies**
  - RNNs struggle with long-range dependencies due to vanishing or exploding gradient problems, especially when capturing relationships between distant tokens.
  - Example: In document summarization, an RNN might lose context over long passages.
- **Parallelization**
  - RNNs process input sequences step-by-step (sequentially), making it difficult to parallelize computations, leading to slower training times, especially for long sequences.

Motivation: Encoder-Decoder Model with Attention

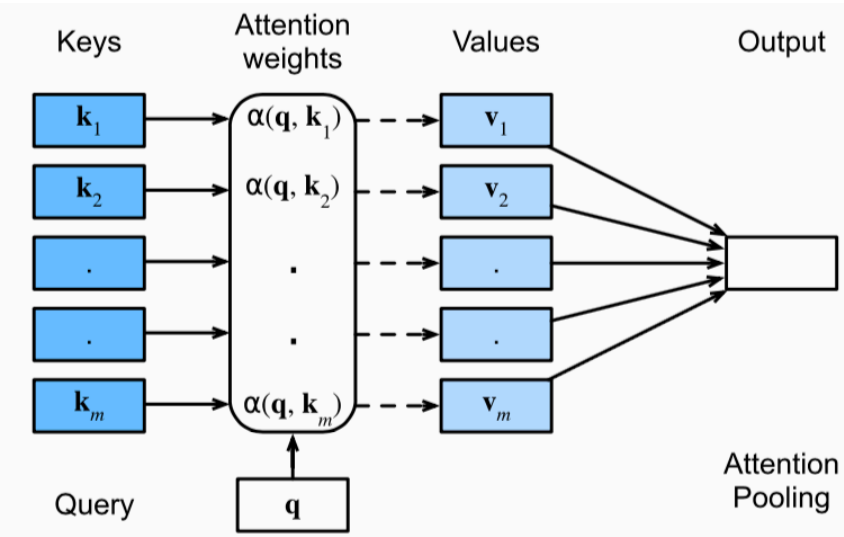
- Pay attention to the most relevant source tokens for each step of prediction (e.g. translation)
- **Attention vector**: weighted sum of encoder states with attention weights (distribution over source tokens)



Attention Mechanism: Queries, Keys, and Values

The attention mechanism of a query $\mathbf{q}$ and dataset D of key-value pairs [7],

$$\text{Attention}(\mathbf{q}, D) = \sum_{i=1}^{m} \alpha(\mathbf{q}, \mathbf{k}_i)\mathbf{v}_i,$$



- **Query** q: from which the attention is looking (e.g. target token state)
- **Key** $k_1, \ldots, k_m$: vector at which the query looks to compute weights (e.g. encoded input token state)
- **Value** $v_1, \ldots, v_m$: information to be weighted (e.g. encoded input token state)
- $\alpha(\mathbf{q}, \mathbf{k}_i)$: attention weight assigned to each $\mathbf{v}_i$.

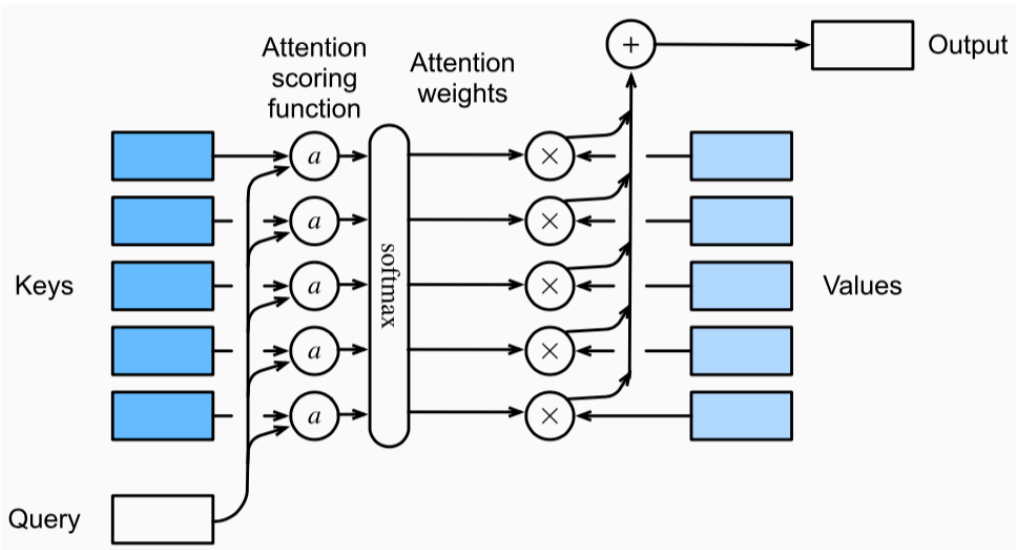
[7] Bahdanau, D., Cho, K., Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate.

Attention Score Function: Dot Product Attention

- The relevance score for Dot Product Attention:

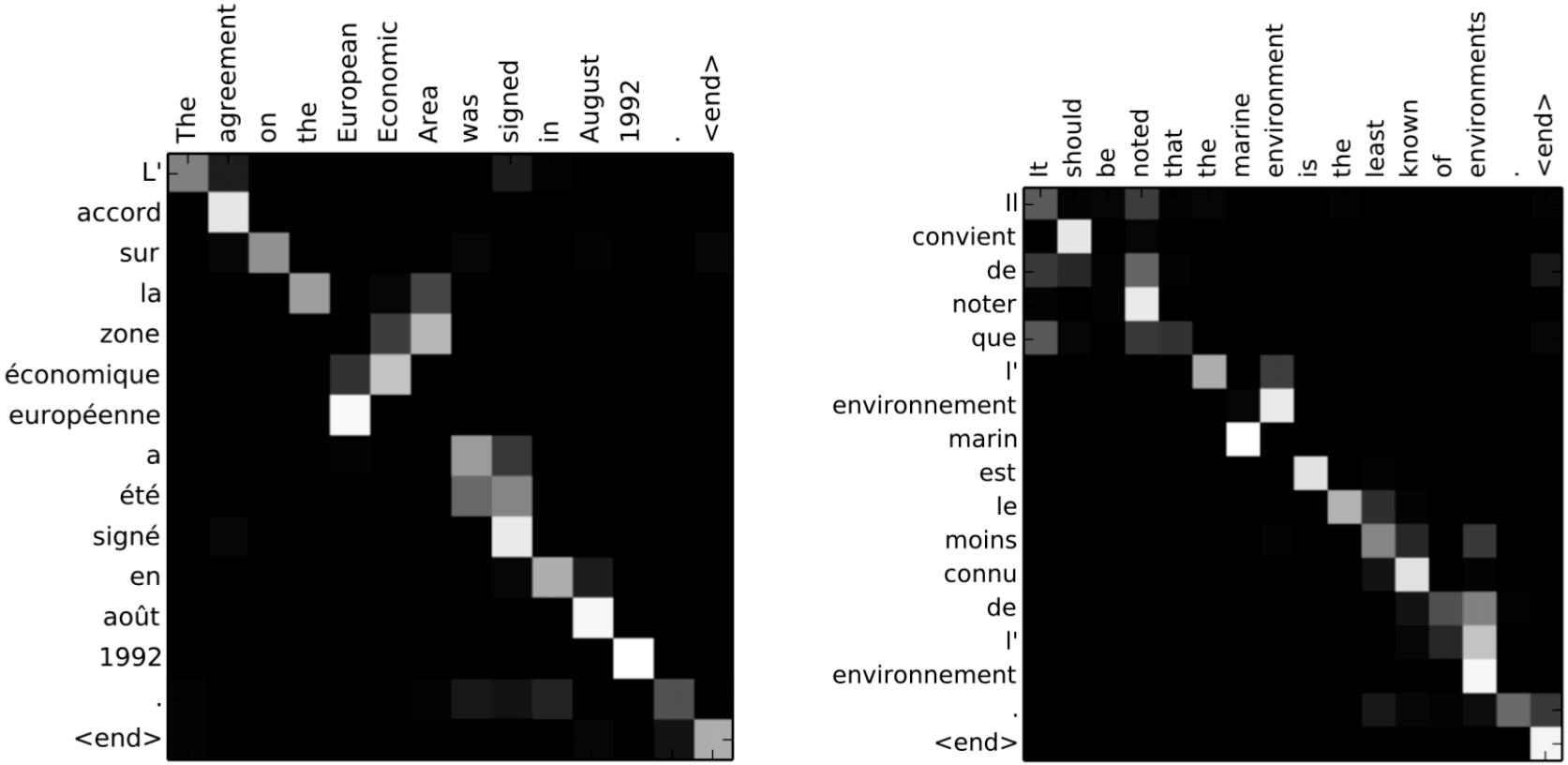
$$\alpha(\mathbf{q}, \mathbf{k}_i) = \frac{\mathbf{q}^\top \mathbf{k}_i}{\sqrt{d}}. \tag{11}$$

- The scaling factor $\sqrt{d}$ improving numerical stability during the softmax computation.



Attention Score Example

Visualize the attention weights between words in the source sentence (English) and the generated translation (French):



Each pixel shows the weight α_{ij} of the annotation of the j-th source word for the i-th target word (Bahdanau, et.al 2014).

Attention Scoring Functions

Definition (Scaled Dot Product Attention)

Given a set of queries $\mathbf{Q} \in \mathbb{R}^{n_q \times d}$, keys $\mathbf{K} \in \mathbb{R}^{n_k \times d}$, and values $\mathbf{V} \in \mathbb{R}^{n_k \times d_v}$, the scaled dot product attention is computed as:

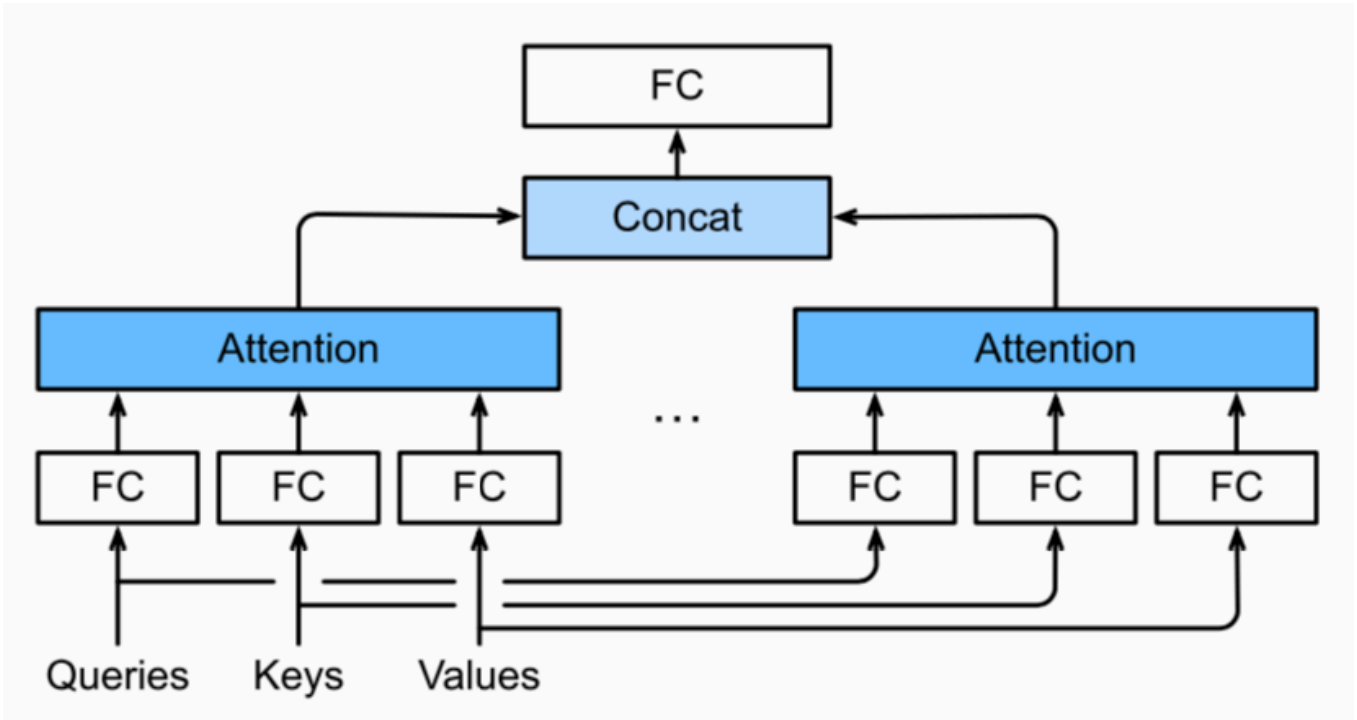
$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d}}\right)\mathbf{V}. \tag{12}$$

The output is a matrix of aggregated weighted values with size $\mathbb{R}^{n_q \times d_v}$, where each query attends to all keys.

Multi-Head Attention Mechanism

Definition (Multi-Head Attention)

Multi-head attention focuses on different positions within the input sequence by computing attention multiple times with separate sets of learned projections, known as "heads." Each head captures distinct aspects of the input data.



Multi-Head Attention Mechanism

Definition (Multi-Head Attention)

Given a query $\mathbf{q} \in \mathbb{R}^{d_q}$, a key $\mathbf{k} \in \mathbb{R}^{d_k}$, and a value $\mathbf{v} \in \mathbb{R}^{d_v}$, each attention head $\mathbf{h}_i$ $(i = 1, \ldots, h)$ is computed as:

$$\mathbf{h}_i = f\left(\mathbf{W}_i^{(q)}\mathbf{q}, \mathbf{W}_i^{(k)}\mathbf{k}, \mathbf{W}_i^{(v)}\mathbf{v}\right) \in \mathbb{R}^{p_v}, \tag{13}$$

where:

- $\mathbf{W}_i^{(q)} \in \mathbb{R}^{p_q \times d_q}$: Query projection matrix for the i-th head.
- $\mathbf{W}_i^{(k)} \in \mathbb{R}^{p_k \times d_k}$: Key projection matrix for the i-th head.
- $\mathbf{W}_i^{(v)} \in \mathbb{R}^{p_v \times d_v}$: Value projection matrix for the i-th head.
- f: Attention pooling function, such as scaled dot product attention.

Multi-Head Attention Mechanism

The output of multi-head attention is obtained by concatenating the outputs of all h heads and applying a linear transformation with learnable parameters $\mathbf{W}_o \in \mathbb{R}^{p_o \times hp_v}$:

$$\text{Multi-Head Attention} = \mathbf{W}_o \begin{bmatrix} \mathbf{h}_1 \\ \vdots \\ \mathbf{h}_h \end{bmatrix} \in \mathbb{R}^{p_o}. \tag{14}$$

Self-Attention: Mathematical Definition

Definition (Self-Attention)

Given a sequence of input tokens $\mathbf{x}_1, \ldots, \mathbf{x}_n$, where any $\mathbf{x}_i \in \mathbb{R}^d$ $(1 \leq i \leq n)$, the self-attention mechanism outputs a sequence of the same length $\mathbf{y}_1, \ldots, \mathbf{y}_n$, where:

$$\mathbf{y}_i = f\big(\mathbf{x}_i, (\mathbf{x}_1, \mathbf{x}_1), \ldots, (\mathbf{x}_n, \mathbf{x}_n)\big) \in \mathbb{R}^d. \tag{15}$$

Here:

- f: Attention pooling function as defined in earlier sections.
- Each $\mathbf{y}_i$ is computed as a weighted combination of the entire input sequence $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$.

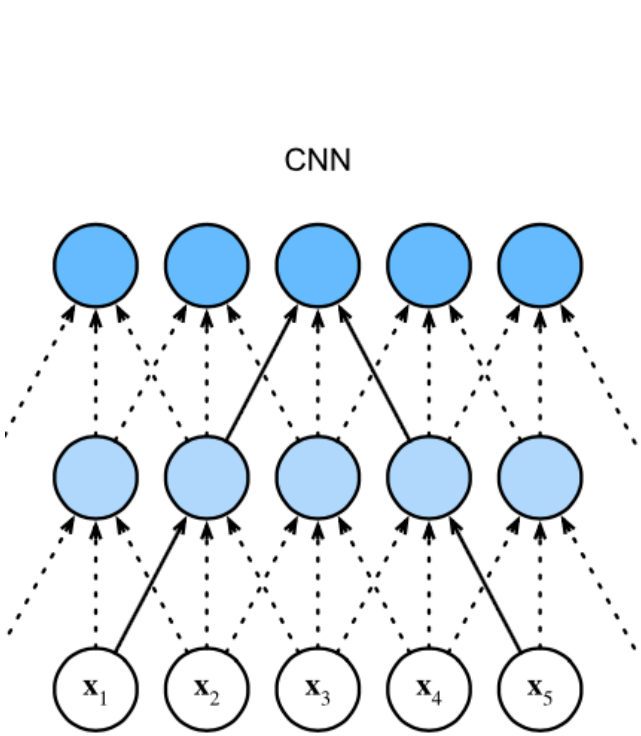
Using multi-head attention, this mechanism computes the self-attention of a tensor with the shape:

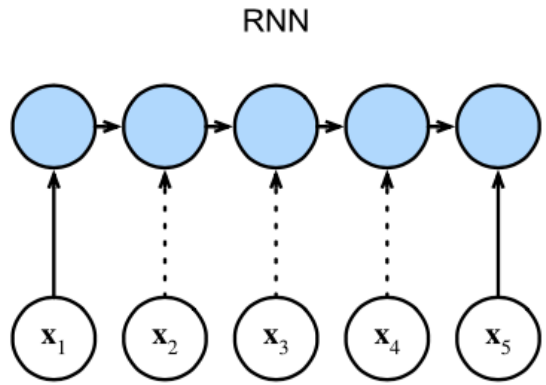
$$\text{(batch size, sequence length, token dimension } d\text{)}.$$

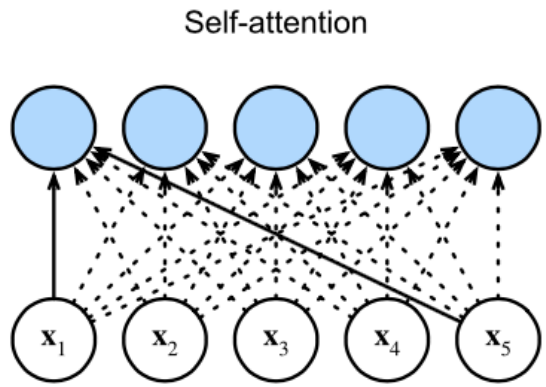
The output tensor retains the same shape as the input.

Comparing CNNs, RNNs, and Self-Attention

The illustration of the three methods by neurons and connections.







Comparing CNNs, RNNs, and Self-Attention

Aspect	CNNs	RNNs	Self-Attention
Complexity	$O(k \cdot n \cdot d^2)$	$O(n \cdot d^2)$	$O(n^2 \cdot d)$
Seq. Operations	$O(1)$	$O(n)$	$O(1)$
Path Length	$O\left(\frac{n}{k}\right)$	$O(n)$	$O(1)$
Focus	Local	Sequential	Global

Summary:

- CNNs and Self-Attention support parallelism; CNNs focus on local interactions.
- RNNs handle sequential data but are less efficient.
- Self-Attention excels at long-range dependencies but has high complexity for long sequences.

Positional Encoding

Motivation: Self-attention mechanisms process tokens in parallel and lack inherent sequence order information. To address this, **positional encoding** is added to input embeddings, providing positional context.

Fixed Positional Encoding:

- Uses sine and cosine functions to encode positions.
- For position i and embedding dimension j:

$$p_{i,2j} = \sin\left(\frac{i}{10000^{2j/d}}\right), \tag{16}$$

$$p_{i,2j+1} = \cos\left(\frac{i}{10000^{2j/d}}\right). \tag{17}$$

Components of the Transformer Architecture

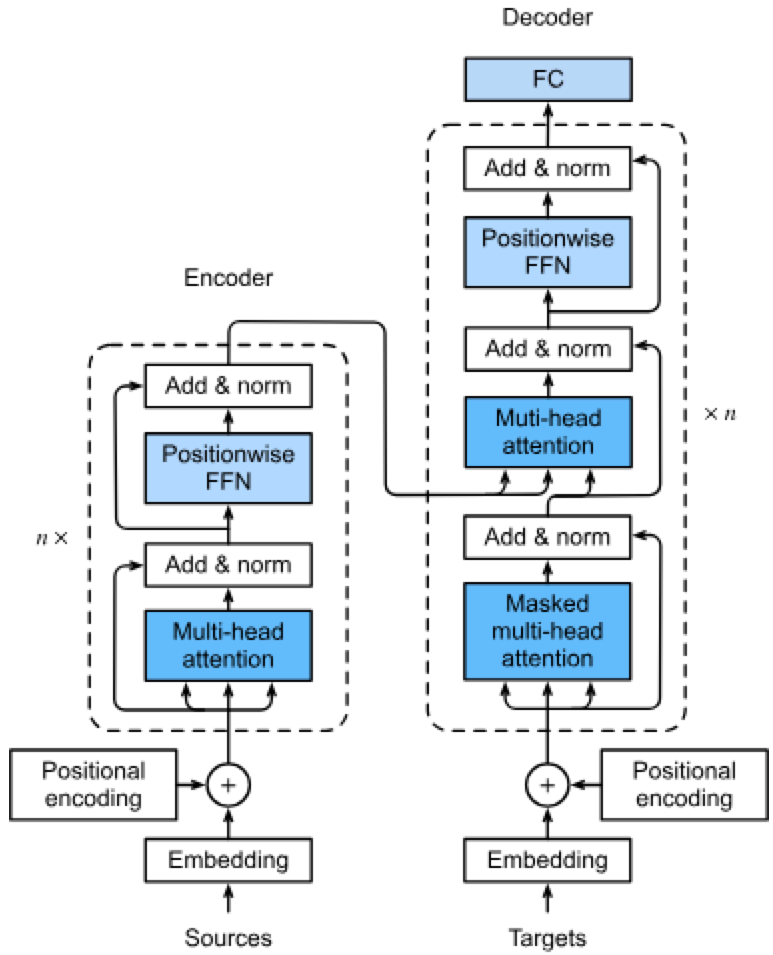
Encoder Components:

- Multi-Head Self-Attention
- Positionwise Feed-Forward Networks
- Residual and Layer Norm

Decoder Components:

- Masked Multi-Head Self-Attention
- Encoder-Decoder Attention
- Positionwise Feed-Forward Networks
- Residual and Layer Norm

Positional Encoding



Transformer Architecture

Backbone Models Based on Transformers

- BERT and GPT Families.
- Vision Transformer (ViT).
- Extensions to Multimodal Tasks. Such as CLIP, BLIP, and Llava...