Accelerator Architectures for Machine Learning (AAML)

Lecture 2: Basics of DNN Models

Tsung Tai Yeh
Department of Computer Science
National Yang-Ming Chiao Tung University

Acknowledgements and Disclaimer

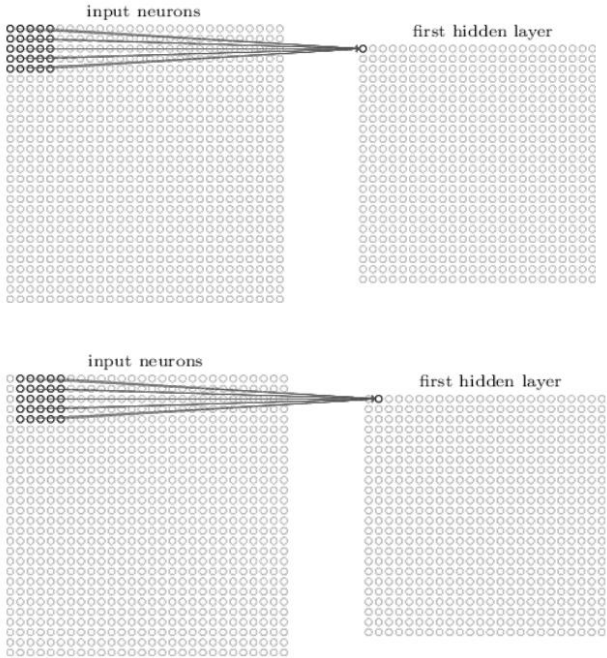
- Slides was developed in the reference with
  Joel Emer, Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, ISCA 2019 tutorial
  Efficient Processing of Deep Neural Network, Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, Morgan and Claypool Publisher, 2020
  Yakun Sophia Shao, EE290-2: Hardware for Machine Learning, UC Berkeley, 2020
  CS231n Convolutional Neural Networks for Visual Recognition, Stanford University, 2020

Outline

- Convolution Neural Network
  - Residual network
  - Depthwise convolution
- Transformer Model Architecture
  - Encoder-decoder based model
  - Large Language Models

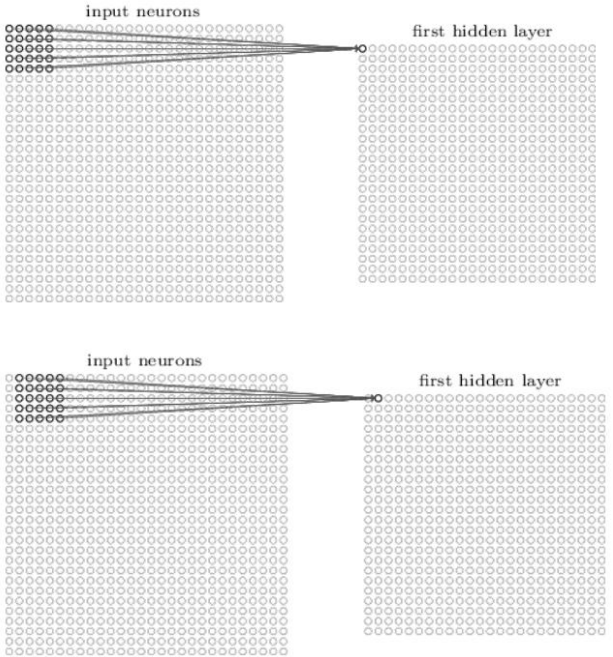
Deep convolutional neural networks

- Each neuron only sees a "**local receptive field**"
  - 5 x 5 grid of neurons in this example
  - The first neuron is looking for feature in the top-left 5x5 corner of the image
  - Combines the 25 inputs with 25 synaptic weights to decide its output
  - The set of 5x5 weights as a "**filter**"



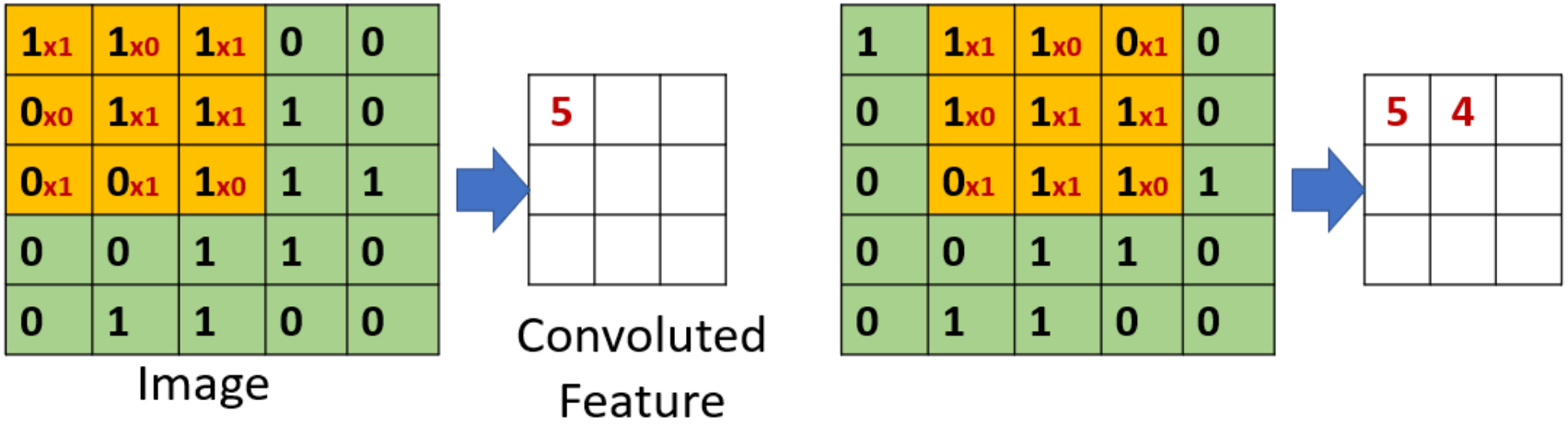
Deep convolutional neural networks

- **Convolution**
  - Applying a 5x5 filter (kernel) to each part of the image
  - All the neurons are sharing the same set of 25 weights (plus bias)
  - Why do we create small size filter ?
    - The small local receptive field and the use of shared weights can help for slow learning rate in early layers of the network



Convolutional Computation Details

- **Convolution**
  - Sliding dot product or cross-correlation
  - Convoluting a 5x5x1 image with a 3x3x1 filter kernel to get a 3x3x1 convoluted feature



CNN Dimension Parameters

- N – Number of **input fmaps**/**output fmaps** (**batch size**)
- C – Number of 2D **input fmaps**/**filters** (**channels**)
- H – Height of **input fmap** (**activations**)
- W – Width of **input fmap** (**activations**)
- R – Height of 2D **filter** (**weights**)
- S – Width of 2D **filter** (**weights**)
- M – Number of 2D **output fmaps** (**channels**)
- F – Width of **output fmap** (**activations**)
- E – Height of **output fmap** (**activations**)

CONV Layer Tensor Computation

Output fmaps (Y) **Bias (B)** **Input fmaps (X)** **Filter weights (W)**

$$Y[n][m][x][y] = Activation(B[m] + \sum_{i=0}^{R-1} \sum_{j=0}^{S-1} \sum_{k=0}^{C-1} X[n][k][Ux+i][Uy+j] \times W[m][k][i][j])$$

$$0 \le n \le N, 0 \le m \le M, 0 \le y \le E, 0 \le x \le F$$

$$E = (H - R + U)/U, F = (W - S + U)/U$$

Shape Parameter	Description
N	fmap batch size
M	# of filters or # of output fmap channels
C	# of input fmap or # of filter channels
U	Convolution stride

CONV Layer Implementation

```
for ( n = 0;   n < N;   n++) {
    for (m = 0;   m < M;   m++) {
        for   (x = 0;   x < F;   x++)      {
            for    ( y = 0;   y < E;   y++)    {
                        Y[n][m][x][y] = B[m];
                        for  ( i = 0;   i < R;  i++)    {
                            for  ( j = 0;  j < S;  j++)     {
                                    for  ( k = 0;  k < C;  k++) {
                                        Y[n][m][x][y] += X[n][k][Ux+i][Uy+j] x W[m][k][i][j]
                                    }
                            }
                        }
                        Y[n][m][x][y] = Activation(Y[n][m][x][y]);
                }
            }
    }
}
```

For each output fmap value

CONV & Activation

How to run CONV in parallel ?

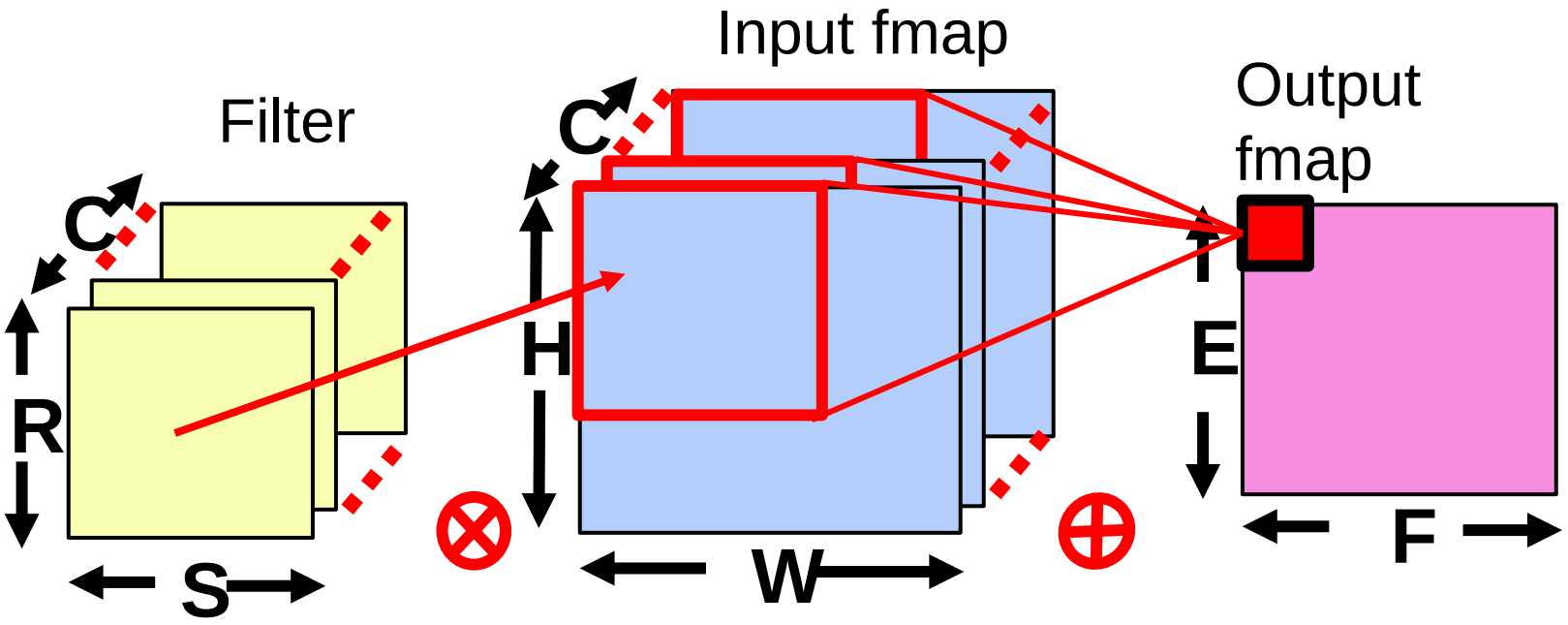
CONV Layer Parallel Implementation

```
Parallel_for ( n = 0;   n < N;   n++) {
    Parallel_for (m = 0;   m < M;   m++) {
        Parallel_for   (x = 0;   x < F;   x++)      {
            Parallel_for    ( y = 0;   y < E;   y++)    {
                        Y[n][m][x][y] = B[m];
                        for  ( i = 0;   i < R;  i++)    {
                            for  ( j = 0;  j < S;  j++)     {
                                    for  ( k = 0;  k < C;  k++) {
                                        Y[n][m][x][y] += X[n][k][Ux+i][Uy+j] x W[m][k][i][j]
                                    }
                            }
                        }
                        Y[n][m][x][y] = Activation(Y[n][m][x][y]);
                }
            }
        }
}
```

For each output fmap value

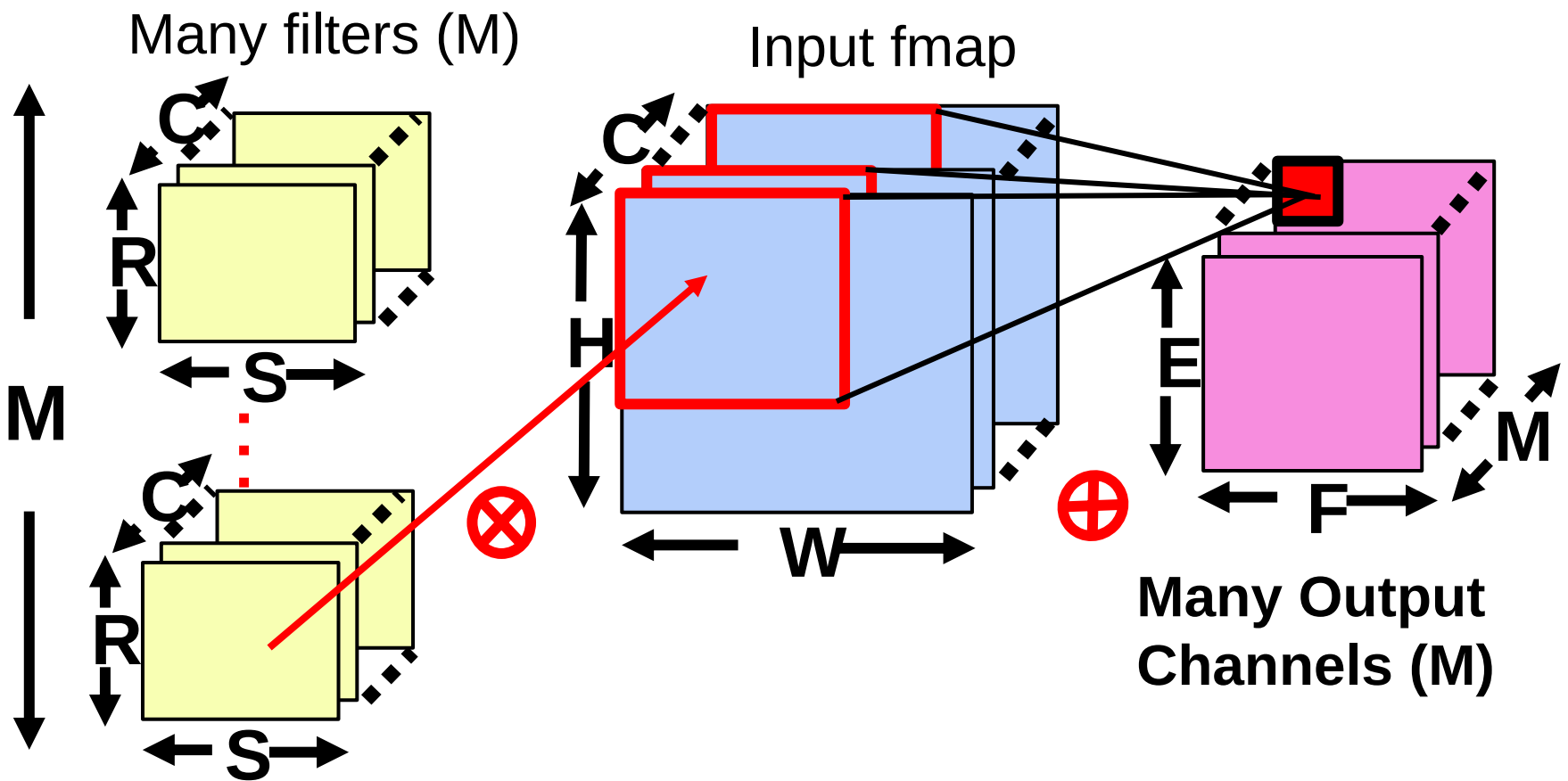
CONV & Activation

Convolution (CONV) Layer

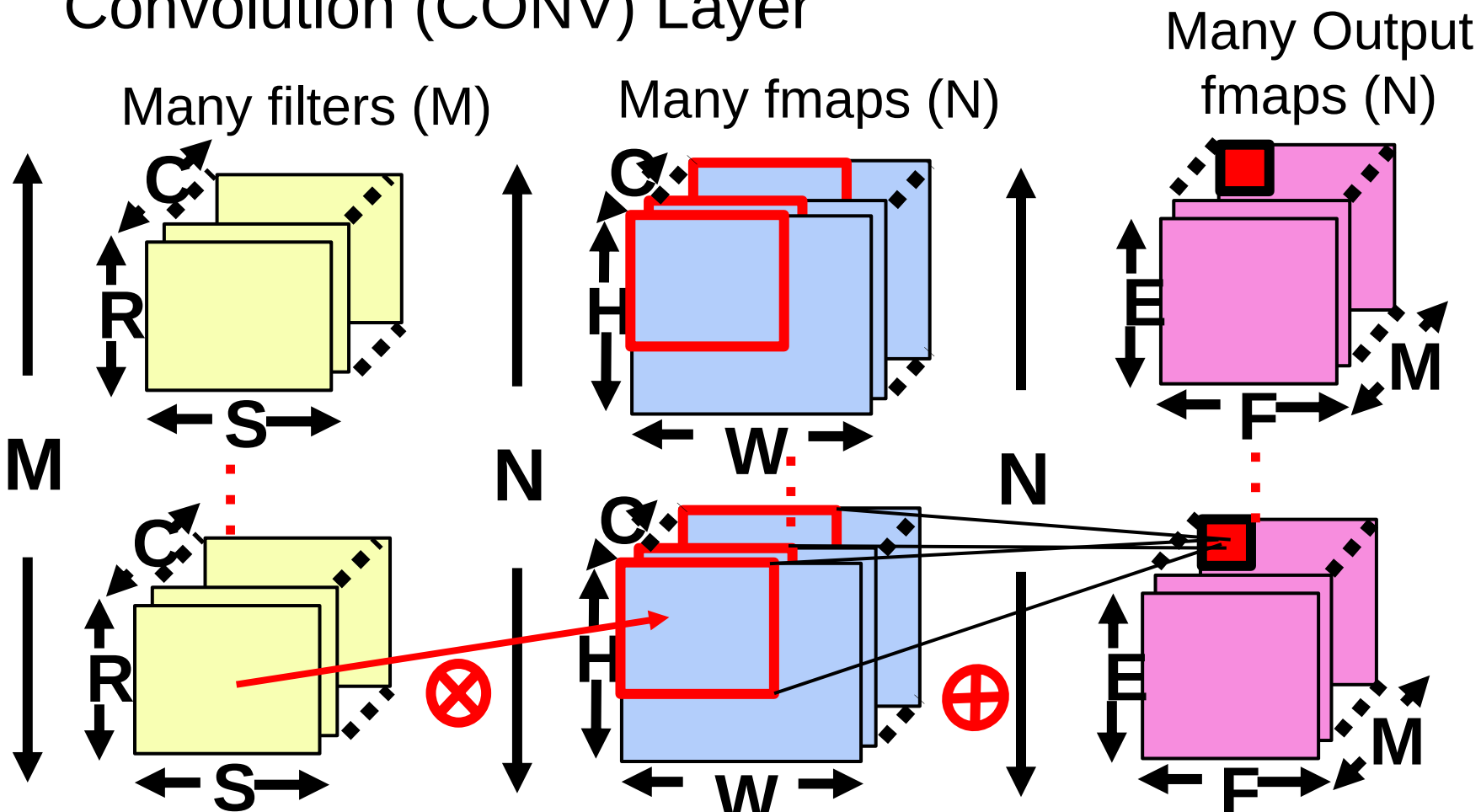


Many Input Channels (C), e.g. RGB in an image

Convolution (CONV) Layer



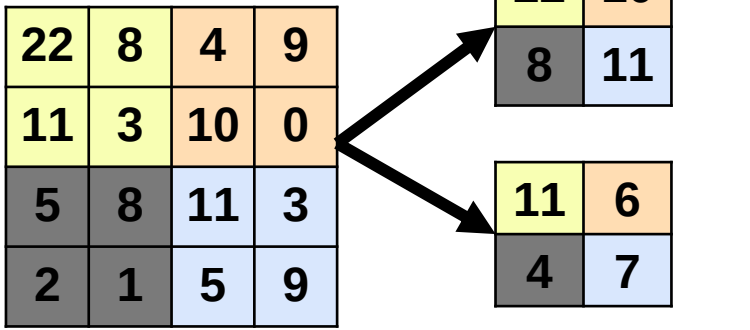
Convolution (CONV) Layer



Pooling

- **Pooling**
  - Once a feature has been found, its's exact location isn't as important as its relative location – help us reduce the parameters
  - Further reduce the network, say reduce 4 neurons into a single one

2x2 pooling, stride = 2



Input fmap size: $W_1 \times H_1 \times C_1$
Spatial extent: **F**
Stride **S**
Output fmap after pooling: $W_2 \times H_2 \times C_2$
$W_2 = (W_1 - F) / S + 1$
$H_2 = (H_1 - F) / S + 1$
$C_2 = C_1$

https://cs231n.github.io/convolutional-networks/

POOL Layer Implementation

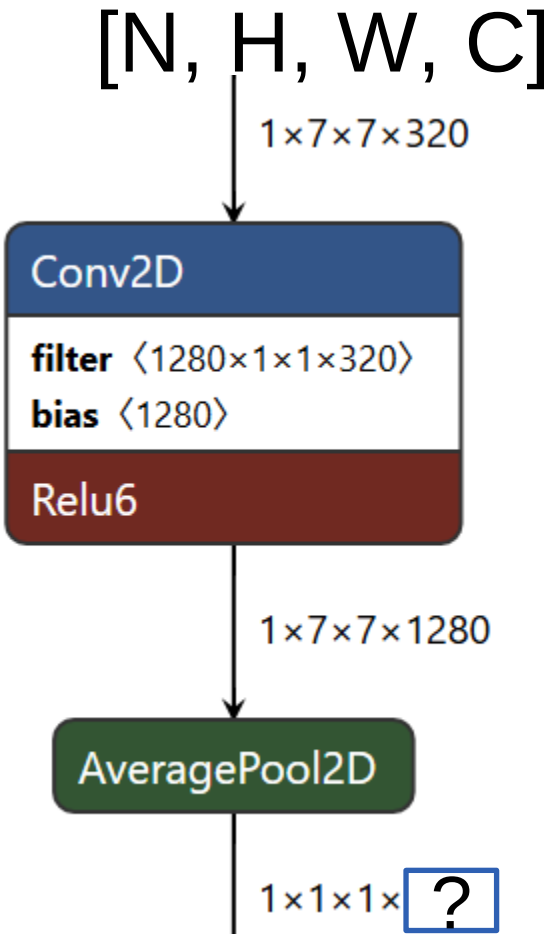
```
for ( n = 0;   n < N;   n++) {
    for (m = 0;   m < M;   m++) {
        for   (x = 0;   x < F;   x++)      {
            for    ( y = 0;   y < E;   y++)    {
                        max = -Inf
                        for  (  i = 0;   i < R;  i++)    {
                            for  (  j = 0;  j < S;  j++)     {
                                    if  (  X[n][m][Ux+i][Uy+j] > max) {
                                        max = X[n][m][Ux+i][Uy+j];
                                    }
                            }
                        }
                        Y[n][m][x][y] = max;
                }
            }
        }
    }
}
```

for each pooled value

Find the max in each window

Takeaway Questions

- What is the output channel of AveragePool2D?
  - (A) 1280
  - (B) 1280/7
  - (C) 640



Online Survey

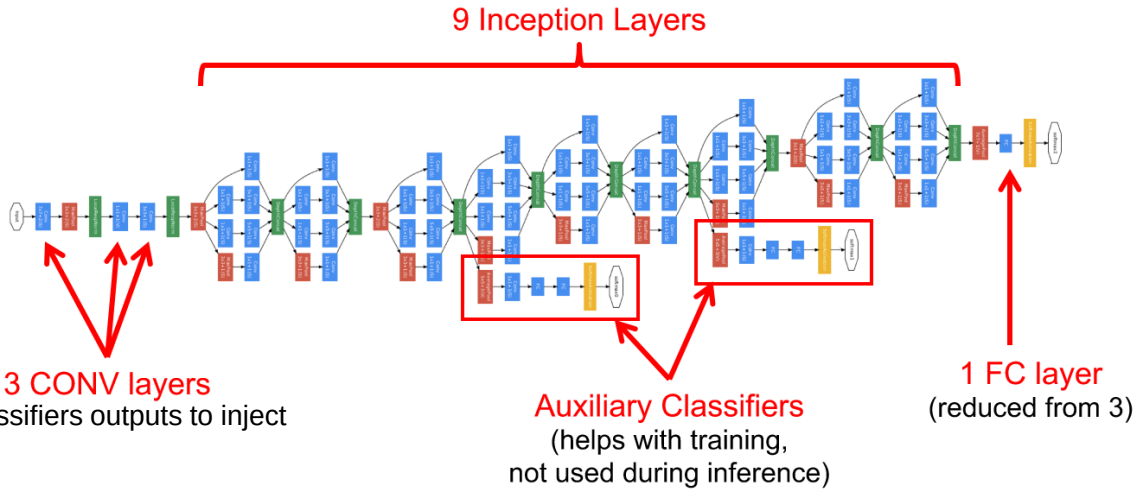
- How to reduce the MACs of the CNN model?

https://reurl.cc/ZNqMZQ

GoogleNet Inception Architecture

- 22 layers
- Fully-Connected Layers: 1
- **Weights: 7.0 M** (< VGG(19.7X) AlexNet(8.7X))
- MACs: **1.4G**

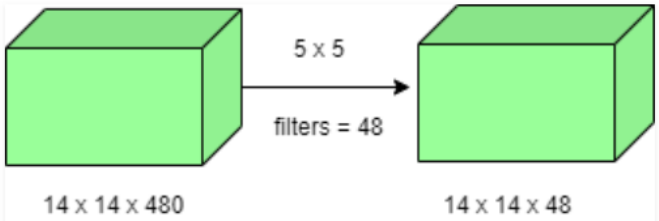
ILSCVR14 Winner
GoogleNet is used to classify images
GoogleNet top-5 error rate is 6.67%
over VGG 7.3%



http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture9.pdf

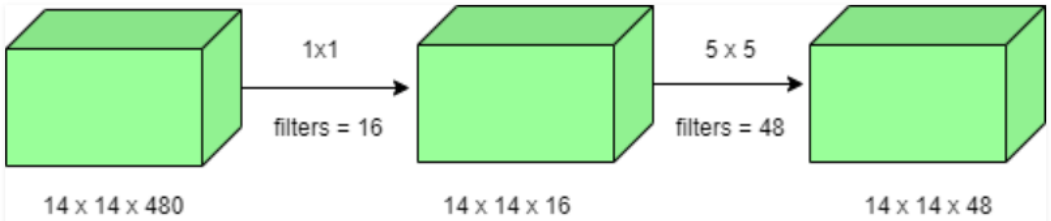
What's New in GoogleNet ?

- **1 x 1 CONV filter (why?)**
  - Decrease the number of parameters (weights and biases)
  - Increase the depth of the network



Case 1: 5x5 filter, # of filter = 48
Total MACs: (14x14x48)x(5x5x480) = **112.9M**

Case 2: 1x1 filter, # of filter = 16 as intermediate
Total MACs: (14x14x16)x(1x1x480) + (14x14x48)x(5x5x16)= **5.3M**

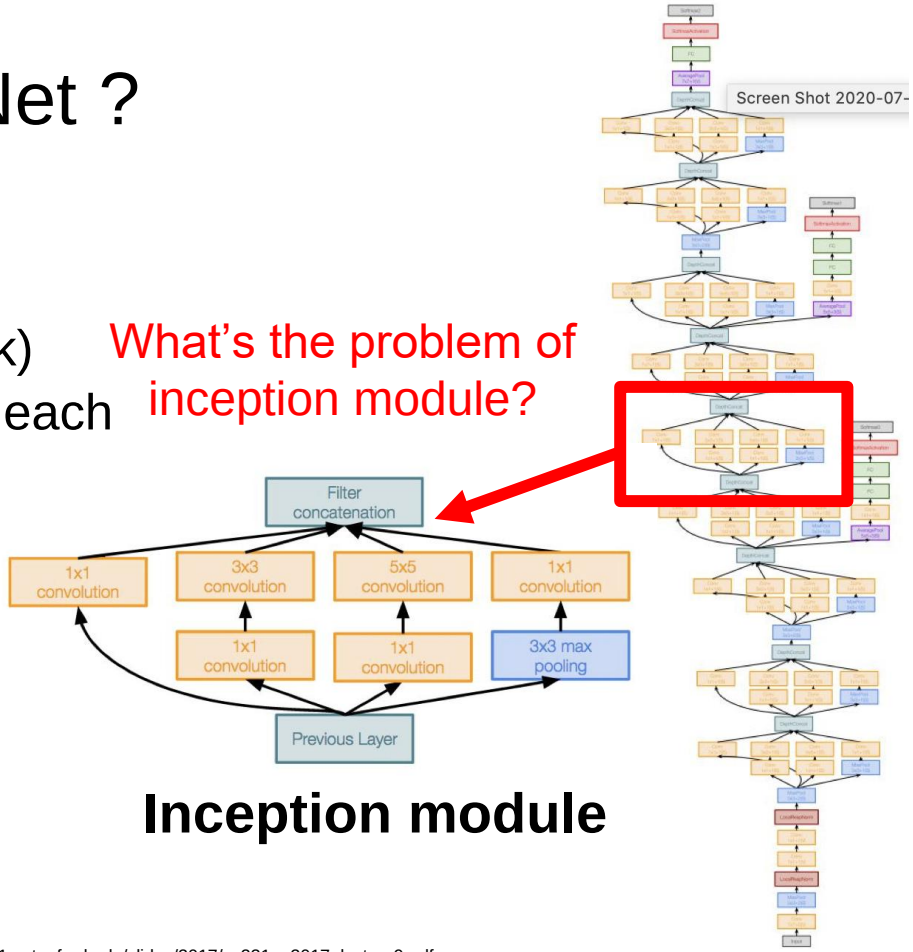


GoogleNet can be trained in a single machine such as (GPU with limit memory space) !!

https://www.geeksforgeeks.org/understanding-googlenet-model-cnn-architecture/

What's New in GoogleNet ?

- **Inception module**
  - A local network topology (network within a network)
  - Stack modules on top of each other
  - Multiple receptive field sizes for CONV (1x1, 3x3, 5x5)
  - Pooling operation (3x3)
  - Depth-wise filter concatenation

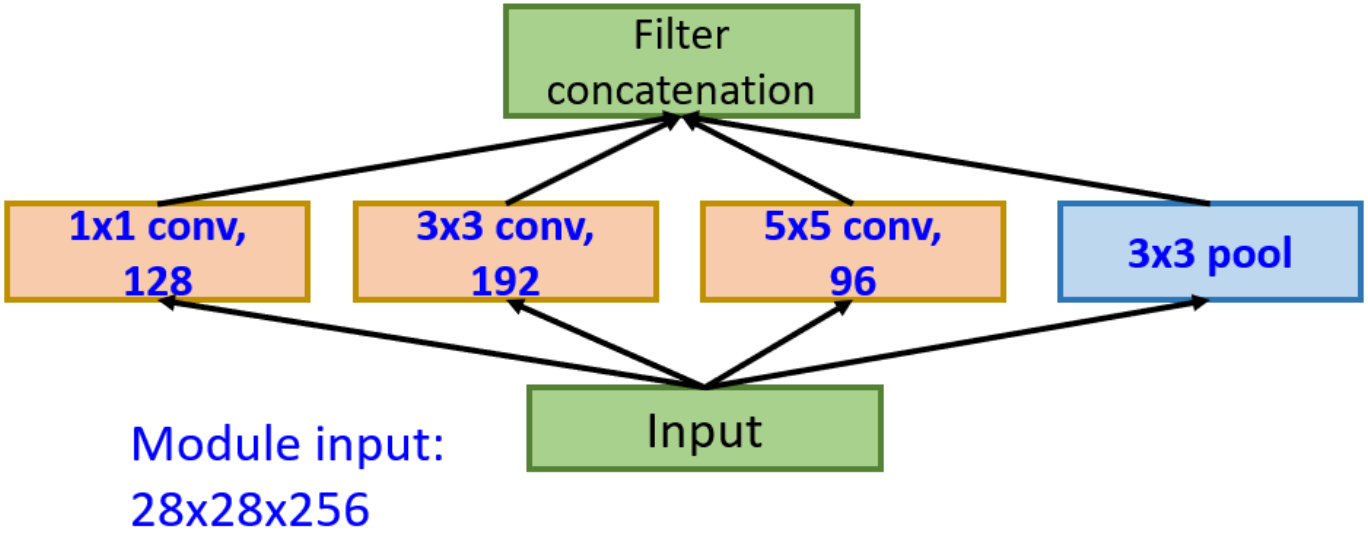


Inception module

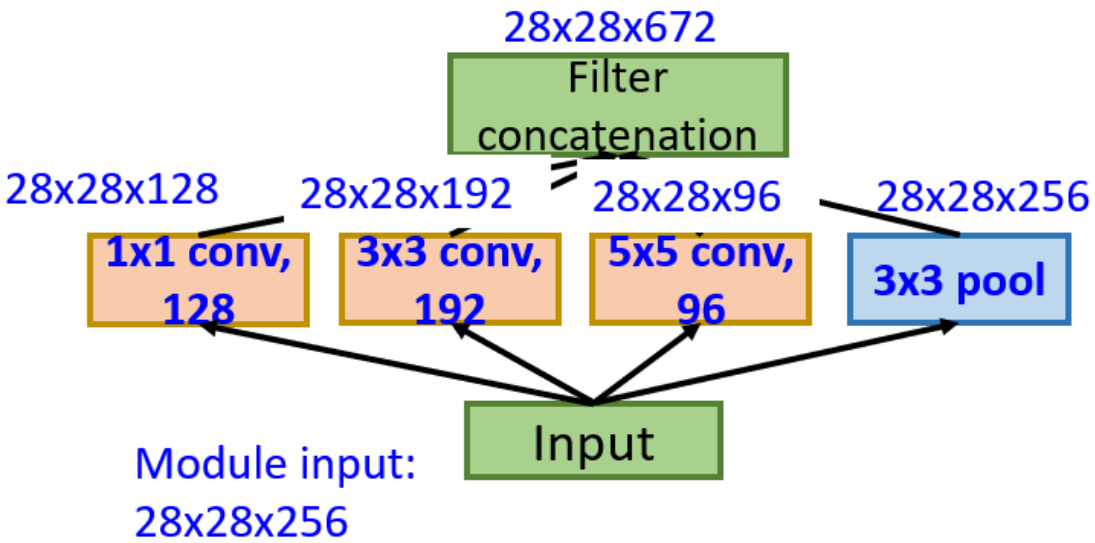
http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture9.pdf

GoogleNet Inception Module Problems?

- What is the output size of 1x1 conv, with 128 filters ?



GoogleNet Inception Module Problems?



CONV Ops:

[1x1 conv, 128] 28x28x128x1x1x256

[3x3 conv, 192] 28x28x192x3x3x256

[5x5 conv, 96] 28x28x96x5x5x256

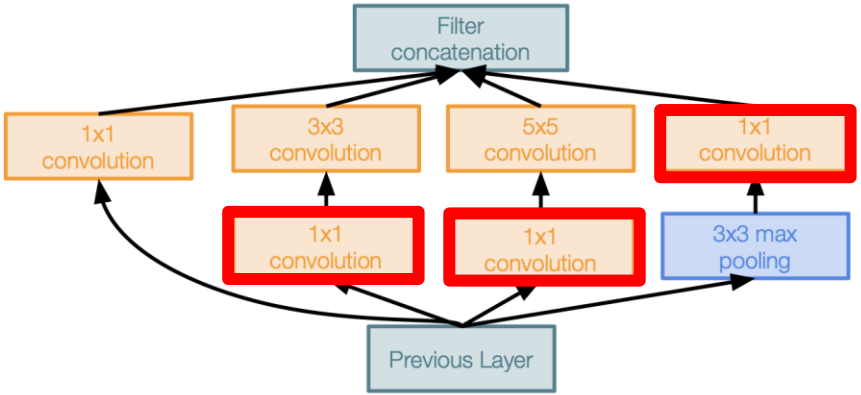
Total: 854 M ops

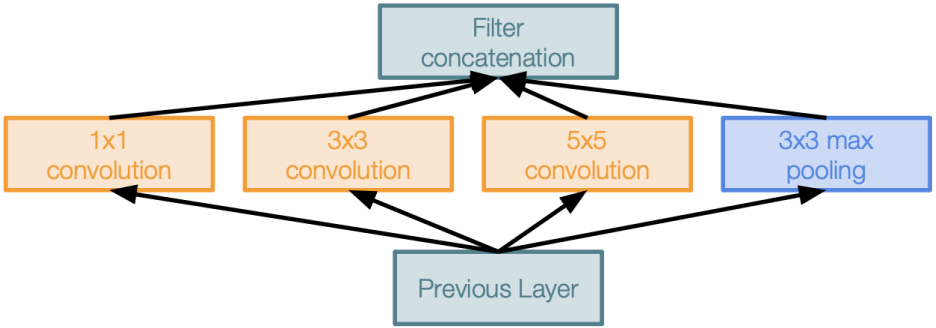
Very expensive compute

Solution: "bottleneck" layers that use 1x1 convolutions to reduce feature depth

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture9.pdf

Dimension Reduction on GoogleNet



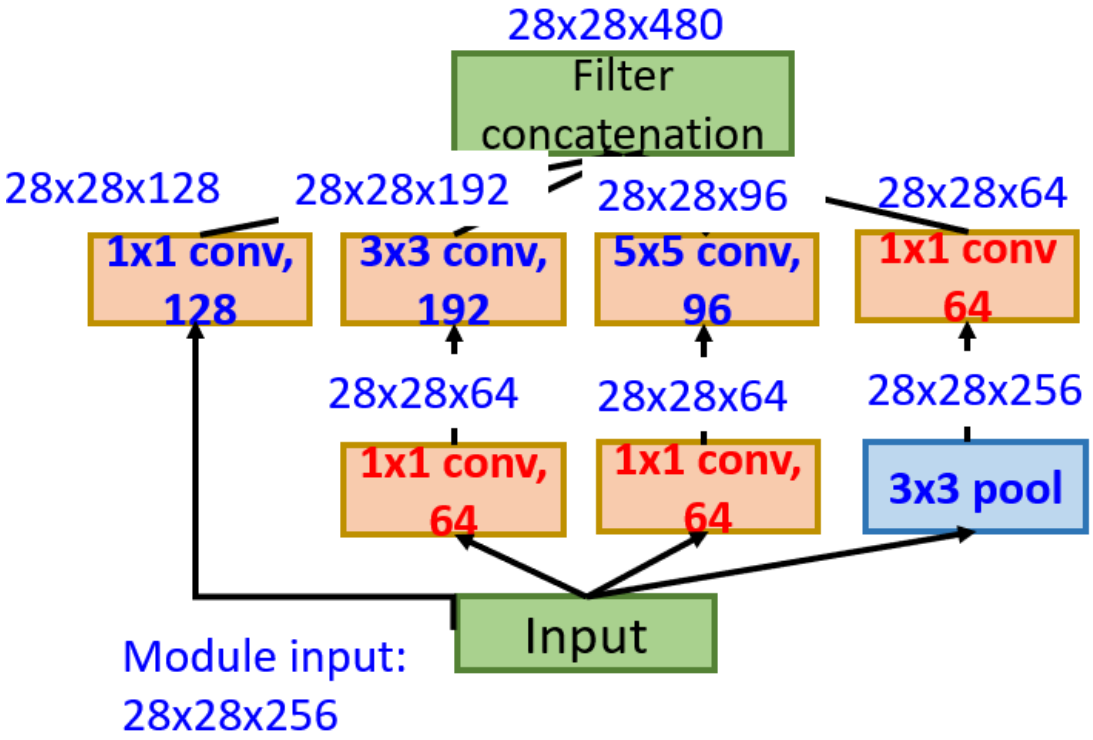


Inception module with **dimension reduction** using **1x1 conv "bottleneck" layers**

Naïve inception module

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture9.pdf

GoogleNet 1x1 Bottleneck Layer Ops



Conv Ops:

[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 128] 28x28x128x1x1x256
[3x3 conv, 192] 28x28x192x3x3x64
[5x5 conv, 96] 28x28x96x5x5x64
[1x1 conv, 64] 28x28x64x1x1x256

Total: 358 M ops

Naïve version has **854M ops**
Bottleneck layer can reduce ops using dimension reduction

Online Survey

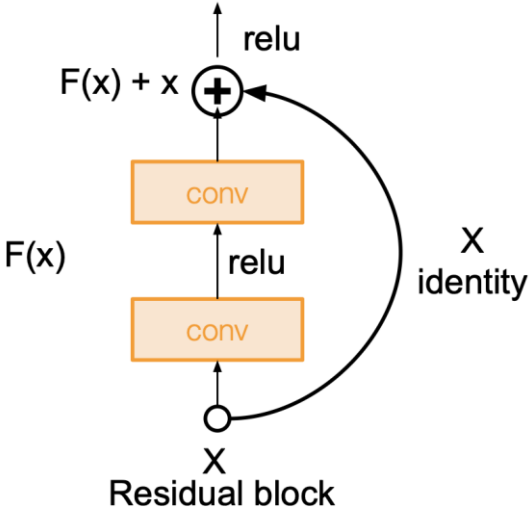
- How to train a deep neural network?

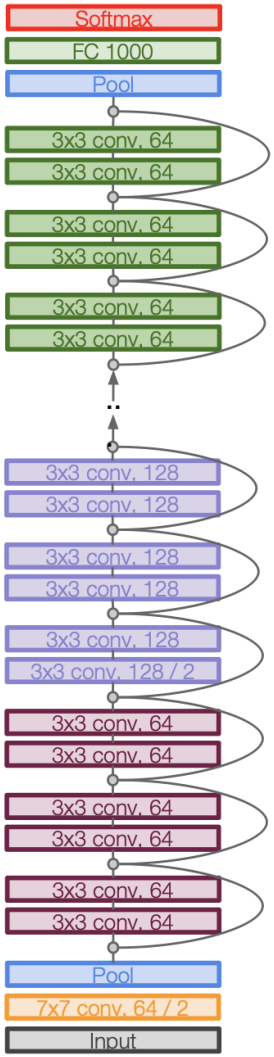
https://reurl.cc/ZNqMZQ

ResNet Model Overview

- 152-layer model for ImageNet Classification
- ILSVRC'15 winner (3.57% top-5 error)
- Using residual blocks and connections

How to train the data in ULTRA-DEEP network (over 1000 layers)?

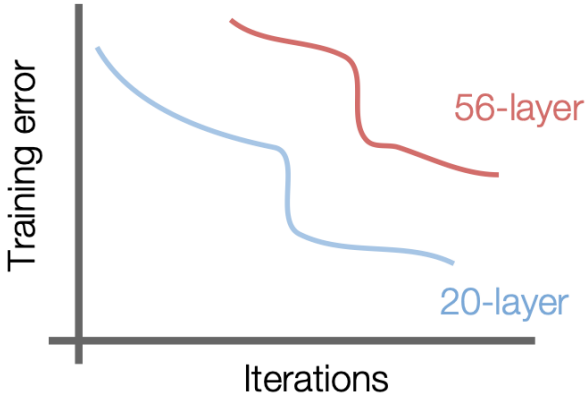


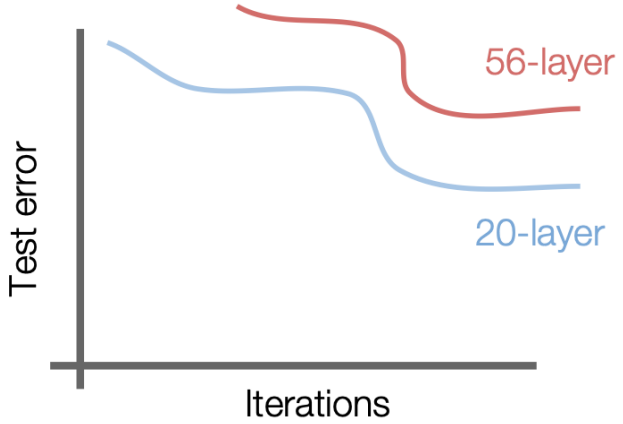


http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture9.pdf

Deep Network Training Problems on ResNet

- Training and test error are increased with the length of networks



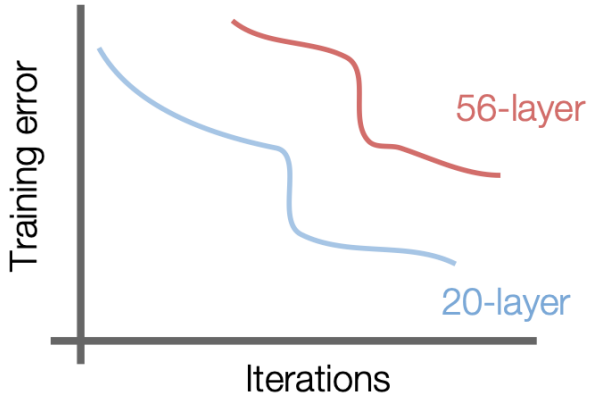


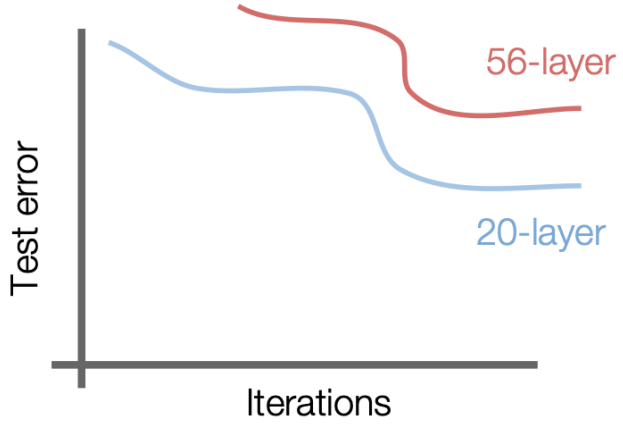
What is wrong when increasing the length of networks?
- Deeper model performs worse on both training and test error (overfitting?)

[He et al., 2015]

Deep Network Training Problems on ResNet

- Why overfitting isn't the main reason to increase error rate of 56-layer?
- Hypothesis: vanishing gradient raises error rate of ultra-deep networks?
  - Solution: Add layers to fit a residual mapping instead of fitting a desired underlying mapping directly (skipping connection)(What?)

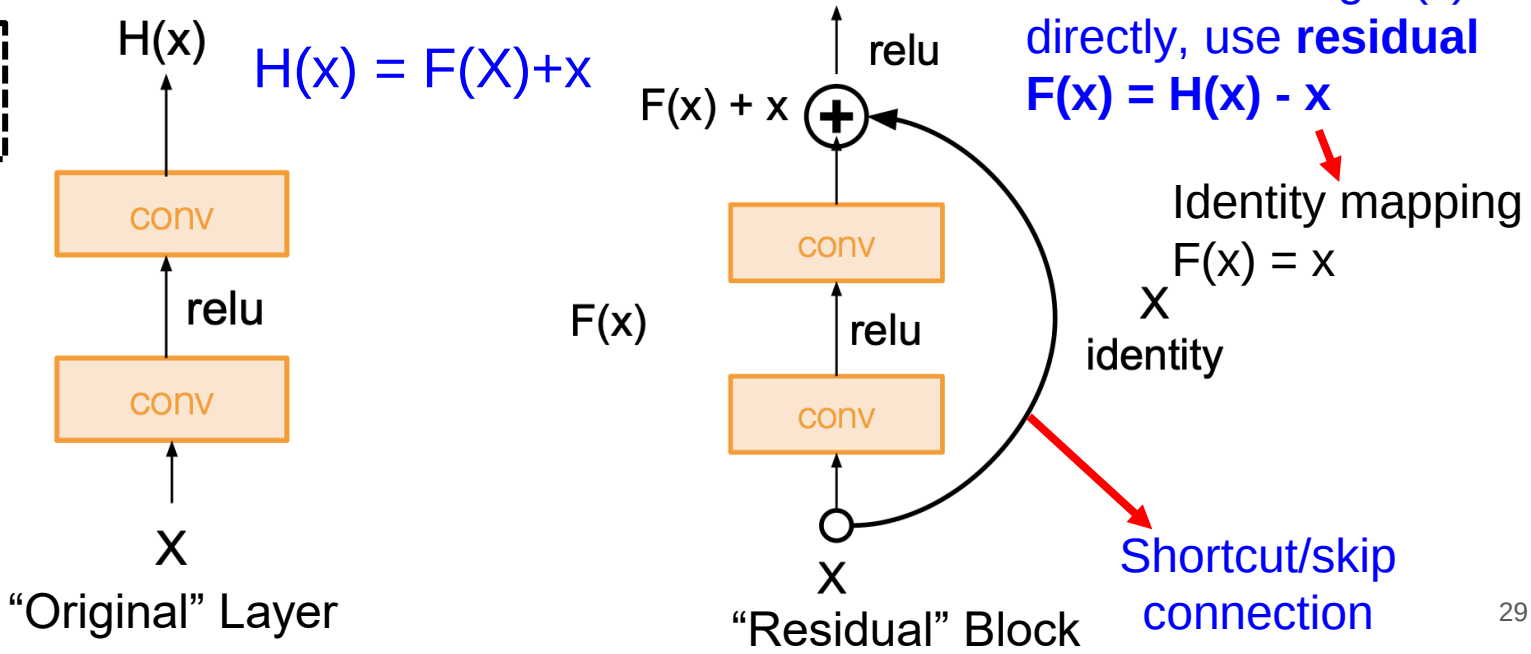




Deep Network Training Problems on ResNet

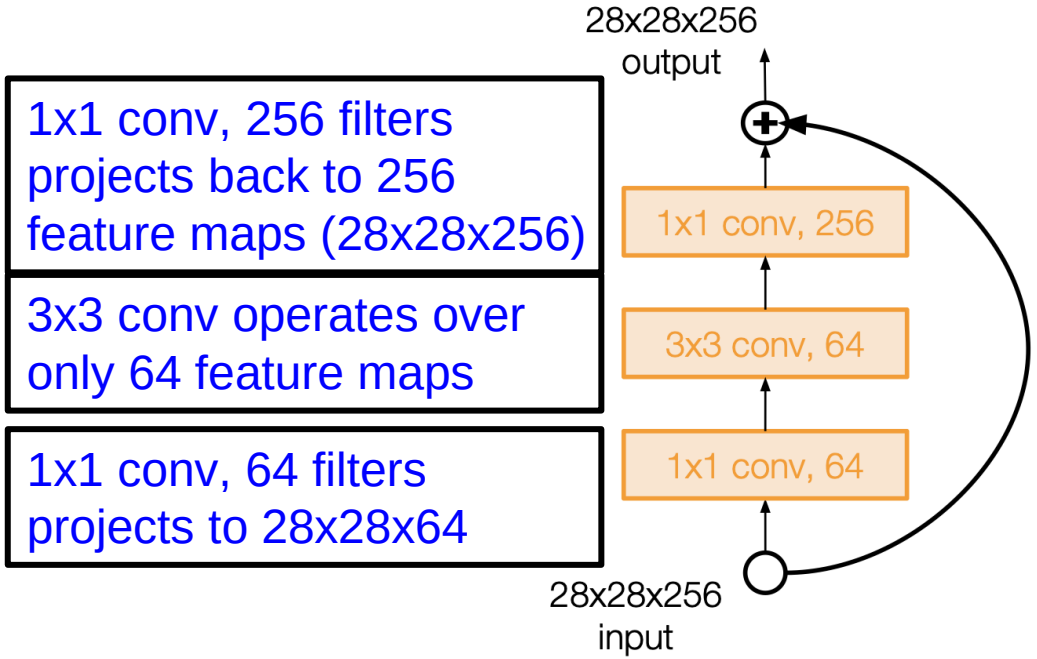
- Solution: Add layers to fit a residual mapping instead of fitting a desired underlying mapping directly

Why not use H(x) directly?



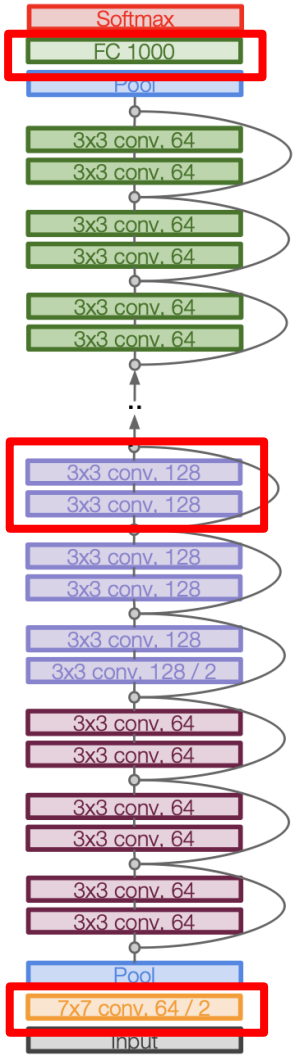
Bottleneck Layer on ResNet

- ResNet50+ also uses “bottleneck” layer to improve efficiency for deep networks (similar to GoogleNet)



ResNet Model Details

- Full ResNet architecture
  - Stack residual blocks
  - Every residual block has two 3x3 conv layers
  - Periodically, double the number of filters and down-sample using stride 2 (/2 in each dimension)
  - Additional conv layer at the beginning
  - Only FC 1000 to output class
  - Total depths of 34, 50, 101 or 152 layers for ImageNet



http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture9.pdf

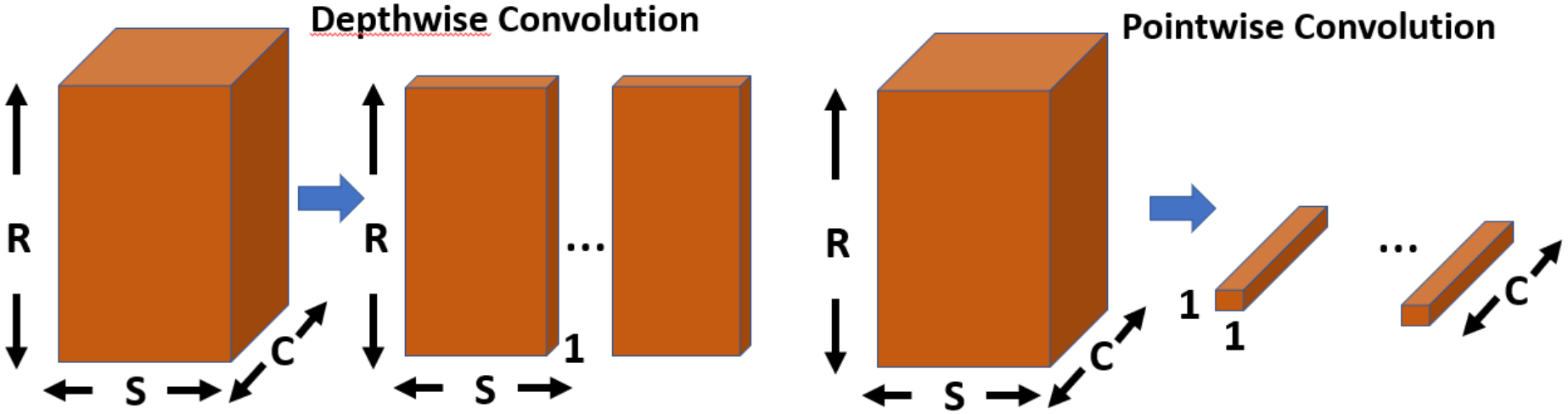
Online Survey

- How to reduce model parameters?

https://reurl.cc/ZNqMZQ

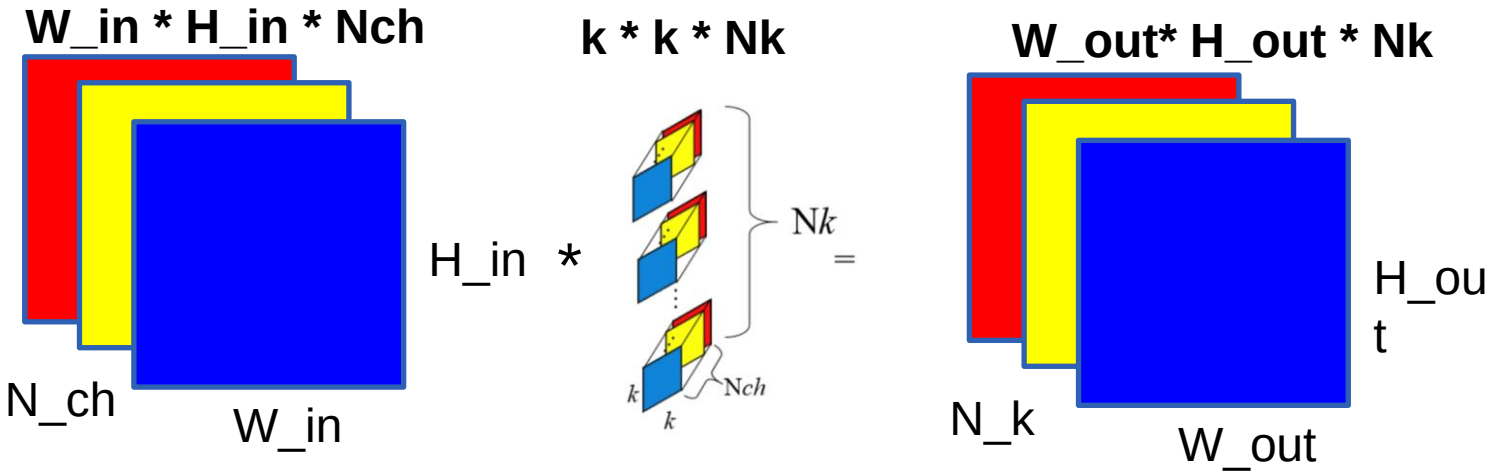
MobileNet v1: Depthwise Separable CONV

- Decouple **cross-channel correlations** and **spatial correlations** in the feature maps
- How to reduce # of parameters? [Andrew et. al. arxiv, 2017]



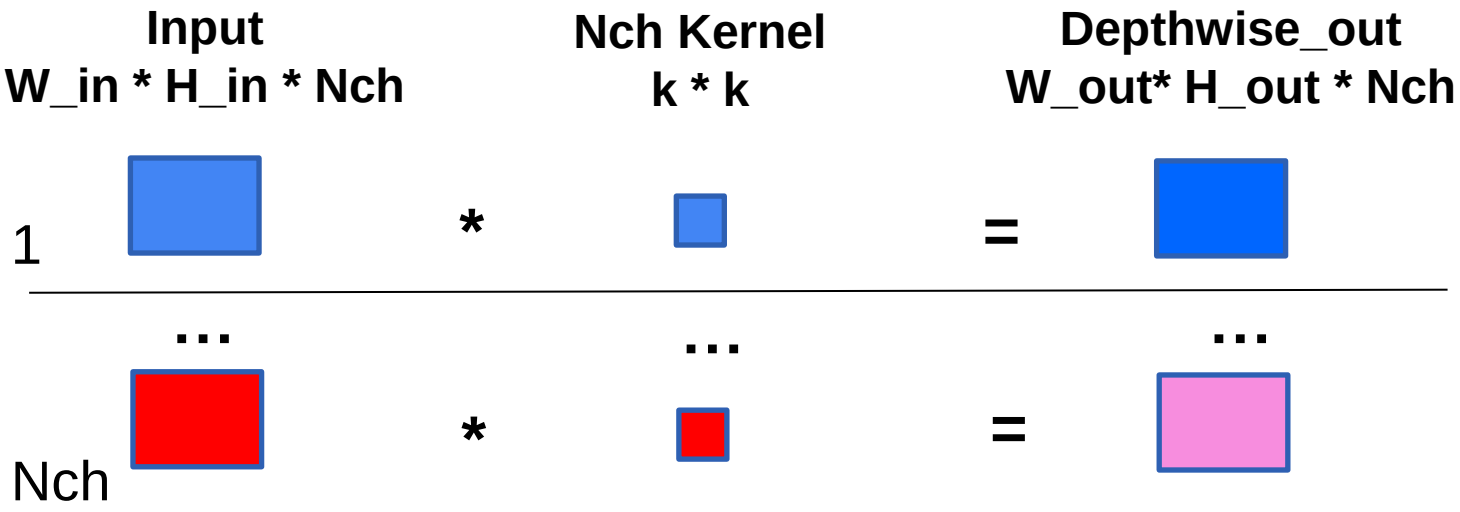
What is Depthwise Separable Convolution ?

- **Purpose**: Reduce the amount of CONV computation
- **Input:** W_in * H_in * Nch (# of channels)
- **Kernel:** k * k * Nk (# of kernels)
- **Output:** W_out * H_out * Nk (# of kernels)



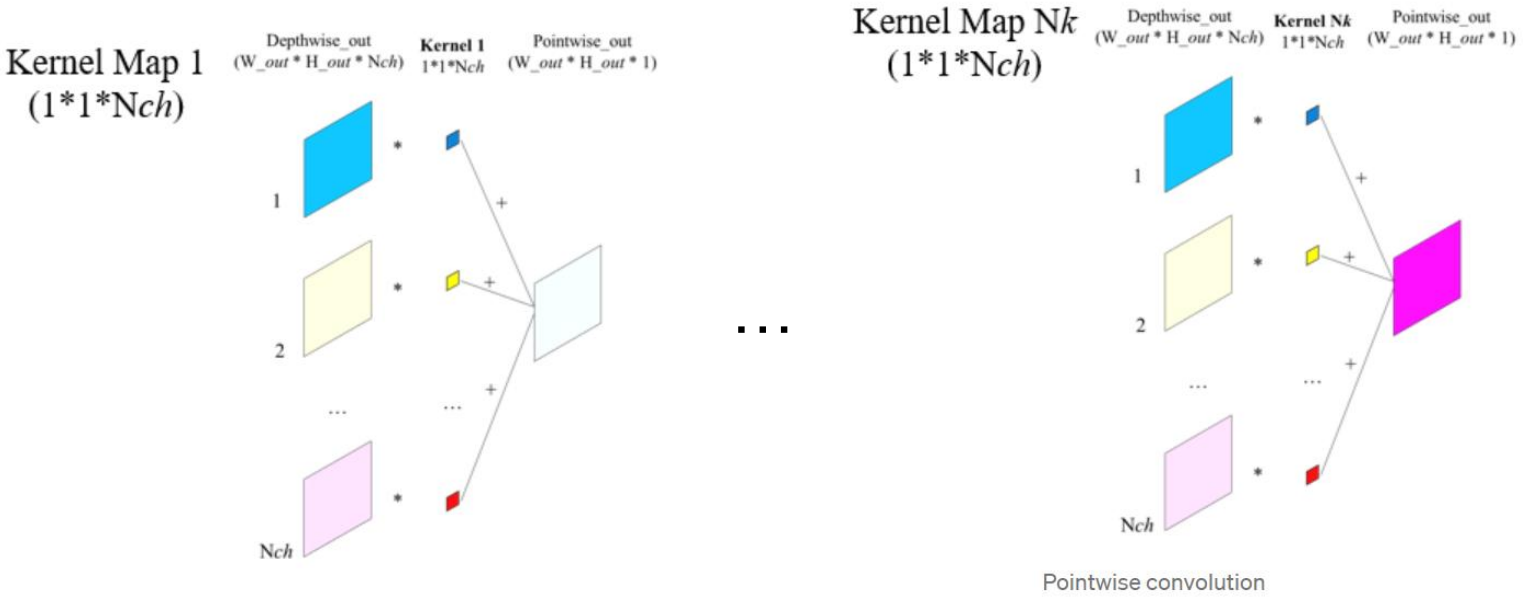
Depthwise Convolution

- Each channel of inputs has a **k * k** kernel
- Separate the convolution of each channel
- **Difference:** Every kernel convolves with all channels in standard CONV



Pointwise Convolution

- The number of kernel: Nk with (1 * 1 * Nch) size
- Do CONV on the outputs of depthwise convolution

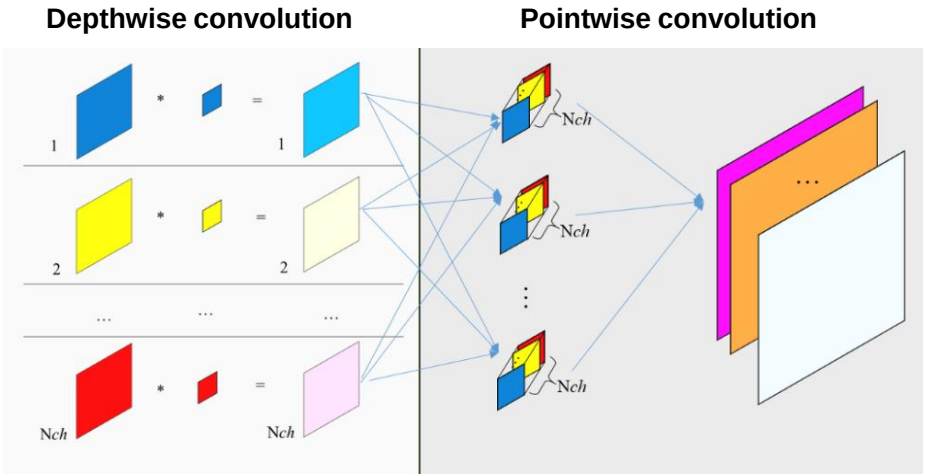


深度學習-MobileNet (Depthwise separable convolution) | by Tommy Huang | Medium

Depthwise + Pointwise Convolution

Depthwise convolution
Input: W_in * H_in * Nch
Nch Kernel (k * k)
Output: W_out * H_out * Nch

Pointwise convolution
Input: W_out * H_out * Nch
Nk kernel = (1 * 1 * Nk)
Output = W_out * H_out * Nk



深度學習-MobileNet (Depthwise separable convolution) | by Tommy Huang | Medium

Depthwise Separable Convolution

- **Standard CONV**
  - Input: W_in * H_in * Nch
  - Kernel: k * k * Nk
  - Output: W_out * H_out * Nk
  - Computation: W_in * H_in * Nch * k * k * Nk
- **Depthwise separable convolution**
  - Depthwise CONV computation: W_in * H_in * Nch * k * k
  - Pointwise CONV computation: Nch * Nk * W_in * H_in

$$\frac{\text{Depthwise separable CONV}}{\text{Standard CONV}} = \frac{W\_in * H\_in * Nch * k * k + Nch * Nk * W\_in * H\_in}{W\_in * H\_in * Nch * k * k * Nk}$$

$$= \frac{1}{Nk} + \frac{1}{K*k}$$

Depthwise Separable Convolution

- Depthwise separable convolution can save more computation when
  - kernel size is large
  - The number of kernel is increased
- Suppose input is 416 * 416 * 50, # of filter is 10, its size is 3 * 3.
- How much computation can be saved by depthwise separable convolution ?
  - 1/10 + 1/9 = 0.22

Takeaway Questions

- What are problems in ultra-deep neural networks ?
  - (A) Over-fitting
  - (B) Gradient vanishing
  - (C) Low training accuracy
- Given a CNN model below, how many channels are in the second layer ?
  - (A) 4
  - (B) 8
  - (C) 16

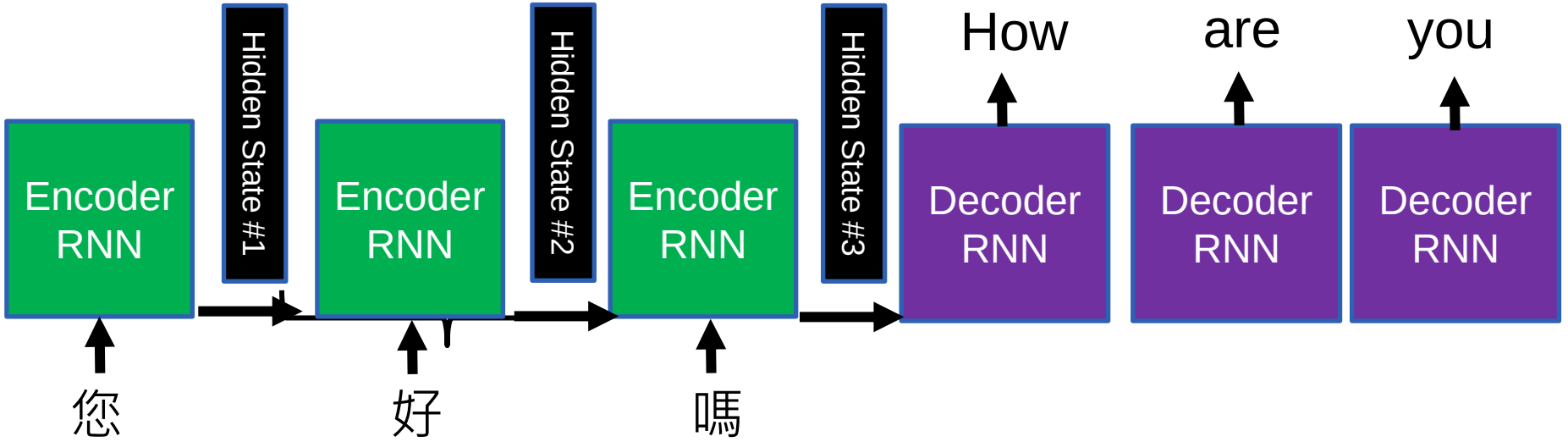
	Input size	# of filter	Filter size	# of channel
Layer1	12 x 12	4	3x3	64
Layer2	12 x 12	16	3x3	

Takeaway Questions

- A standard CONV layer
  - Input: W_in * H_in * Nch = (32 * 32 * 16)
  - Kernel: k * k * Nk = (3 * 3 * 8)
  - Computation: W_in * H_in * Nch * k * k * Nk = (32 * 32 * 16 * 3 * 3 * 8)
- What is the amount of computation that is carried out by **depthwise separable convolution ?**
  - (A) (32 * 32 * 16 * 3 * 3) + (3 * 8 * 8)
  - (B) (32 * 32 * 3 * 3 * 8) + (16 * 3 * 3)
  - (C) (32 * 32 * 16 * 3 * 3) + (16 * 8 * 32 * 32)

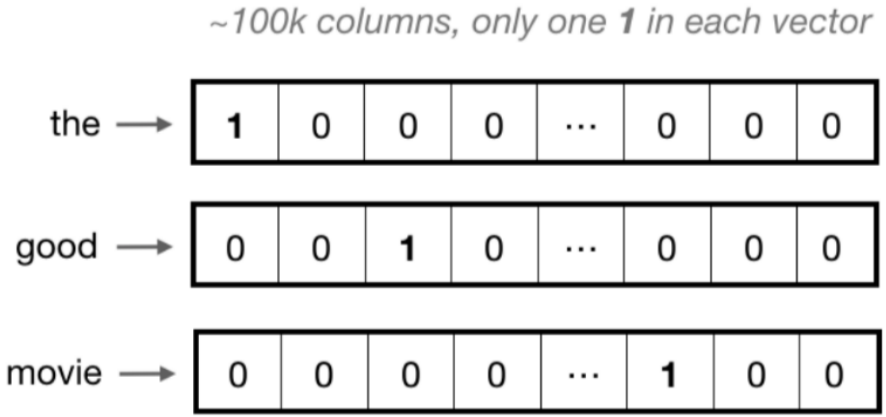
Classical Sequence-to-Sequence Model

- Pass the last hidden state of the encoding stage
- Decoder uses this last hidden state to do the prediction



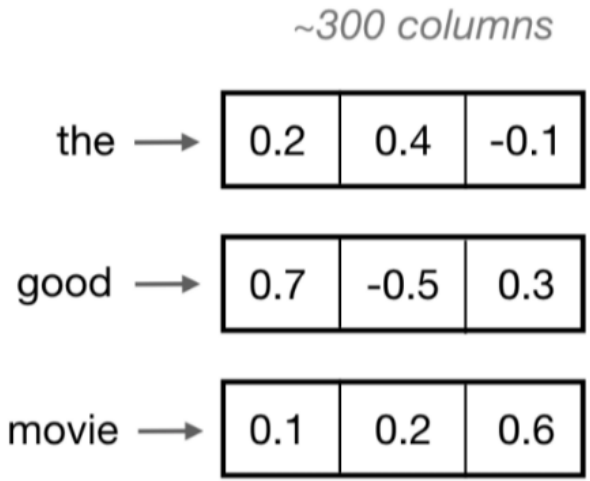
Word Representation

- **One-Hot Encoding**
  - Representing each word as a vector that has as many values in it
  - Each column in a vector is one possible word in a vocabulary
  - Problem
    - In large vocabularies, these vectors can get very long
    - Contain all 0's except for one value
    - Sparse representation



Word Representation

- **Word Embedding**
  - Map the word index to a continuous word embedding through a look-up table
  - Popular pre-trained word embeddings
    - Word2Vec, GloVe



Positional Encoding (PE)

- **Positional encoding (PE)**
  - Information to each word about its position in the sentence
  - **Unique** encoding for each word's position in a sentence
  - Distance between any two positions is **consistent** across sentences with different lengths
  - Encode words by using **sin(), cos()** with different frequencies
  - **Deterministic** and **generalize** to longer sentences

$$\overrightarrow{p_t}^{(i)} = f(t)^{(i)} := \begin{cases} \sin(\omega_k . t), & \text{if } i = 2k \\ \cos(\omega_k . t), & \text{if } i = 2k+1 \end{cases} \qquad \omega_k = \frac{1}{10000^{2k/d}}$$

Positional Encoding (PE)

- Arguments
  - L: maximum # of possible positions
  - d_{model}: dimension of the embeddings
  - n: can be set to any value
  - k: position
  - i: dimension

$$\text{pos(k)} = \begin{bmatrix} \sin \omega_1 \cdot \text{k} \\ \cos \omega_1 \cdot \text{k} \\ \sin \omega_2 \cdot \text{k} \\ \cos \omega_2 \cdot \text{k} \\ \vdots \\ \sin \omega_{\text{d}_{\text{model}}/2} \cdot \text{k} \\ \cos \omega_{\text{d}_{\text{model}}/2} \cdot \text{k} \end{bmatrix}_{\text{d}_{\text{model}}}, \text{where } \omega_{\text{i}} = 10000^{\frac{2\text{i}}{\text{d}_{\text{model}}}}$$

- For each $\text{k} = 0$ to $L - 1$
  - For each $i = 0$ to $\frac{d_{model}}{2}$
    - $PE_{(k,2i)} = \sin(\frac{k}{n^{\frac{2i}{d_{model}}}})$
    - $PE_{(k,2i+1)} = \cos(\frac{k}{n^{\frac{2i}{d_{model}}}})$

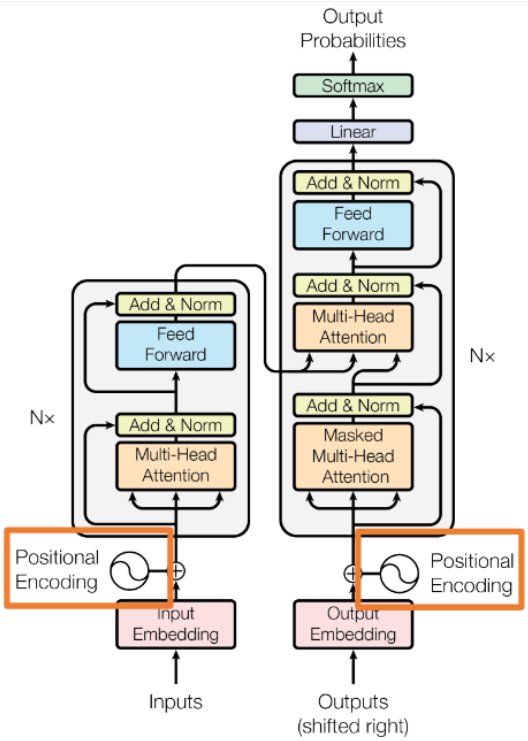
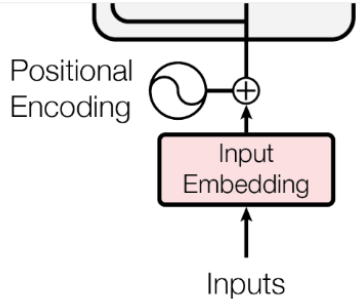
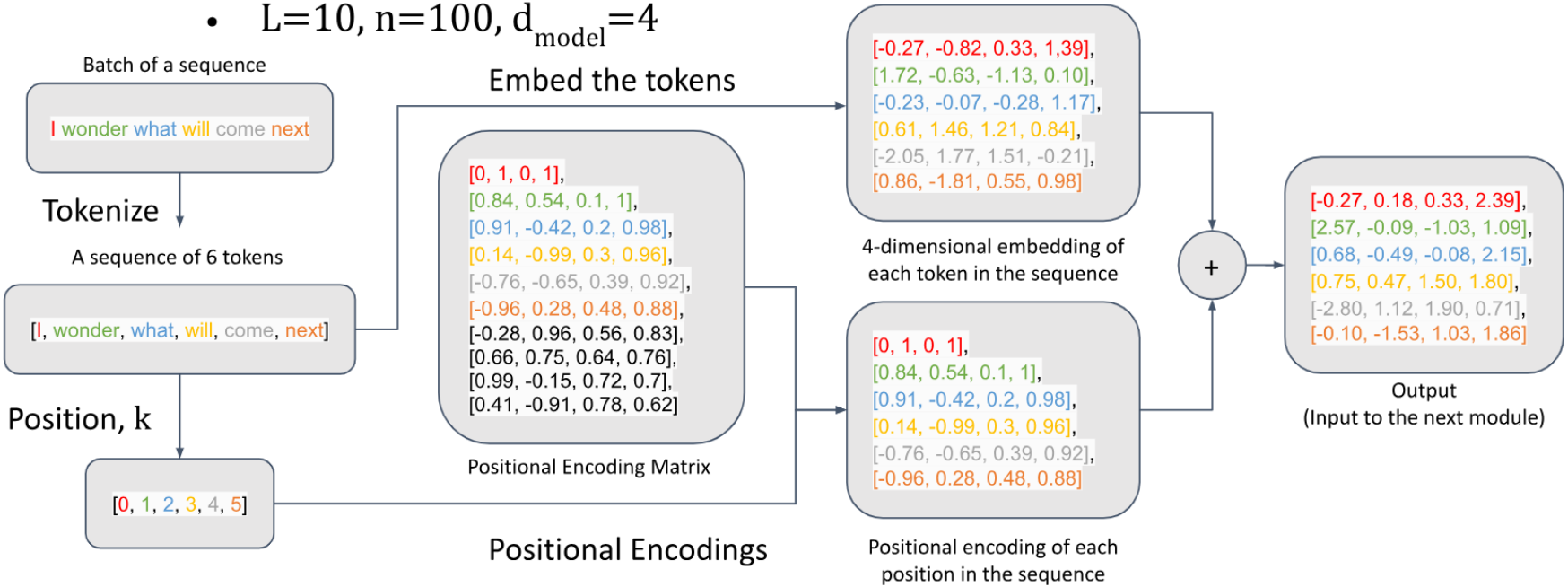


Figure 1: The Transformer - model architecture.



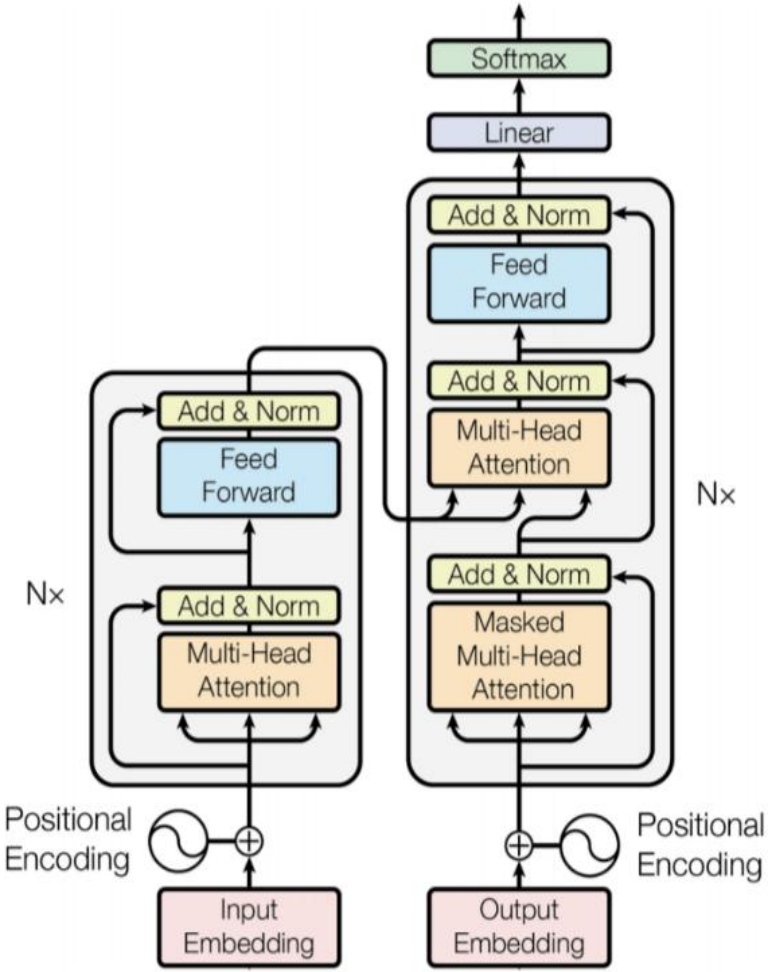
Positional Encoding (PE)

- Arguments
  - $L=10, n=100, d_{model}=4$



Transformer Model Architecture

- Each **encoder** block has two sub-layers
  - Multi-head self-attention
  - A position-wise fully connected feed-forward
- Each **decoder** block has an additional third sub-layer
  - The third is a masked multi-head attention over the output of the encoder stack
- A **residual connection** is added around each of the two sub-layers
- The decoder yields the output sequence of symbols one element at a time

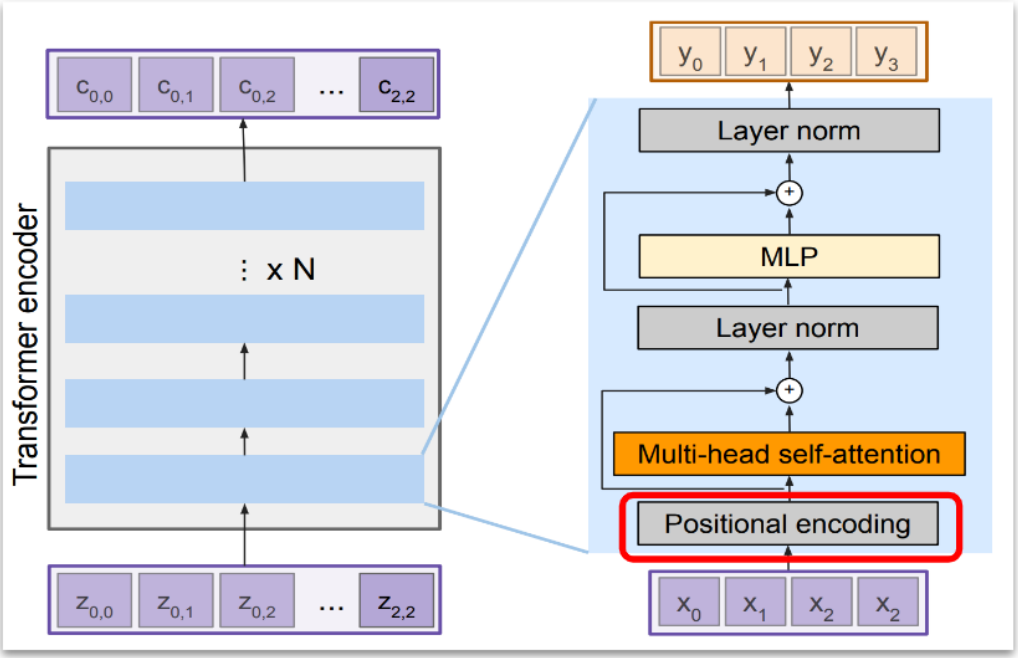


https://web.stanford.edu/class/cs224n/slides/cs224n-2020-lecture10-QA.pdf

Vaswani et al. 2017 https://arxiv.org/pdf/1706.03762.pdf

Transformer Model Architecture

- Transformer encoder



img src: Stanford CS 231n

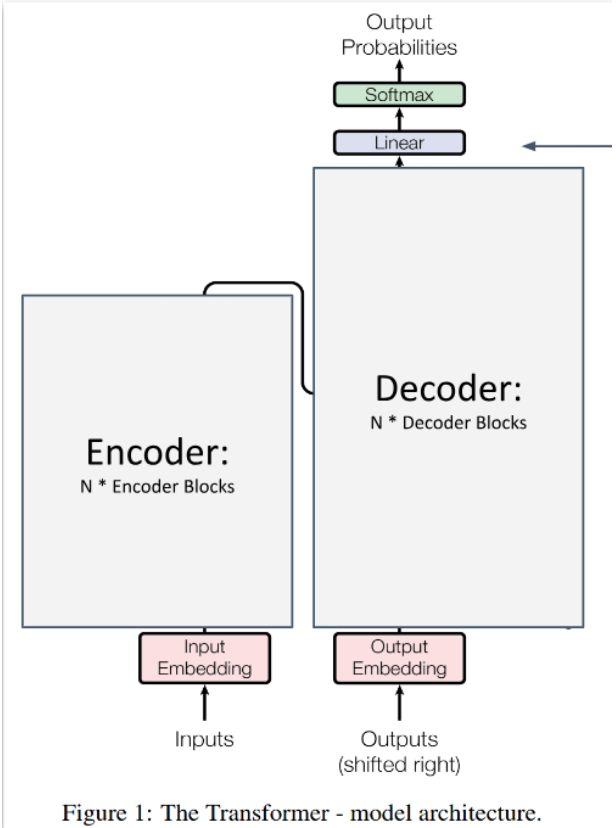
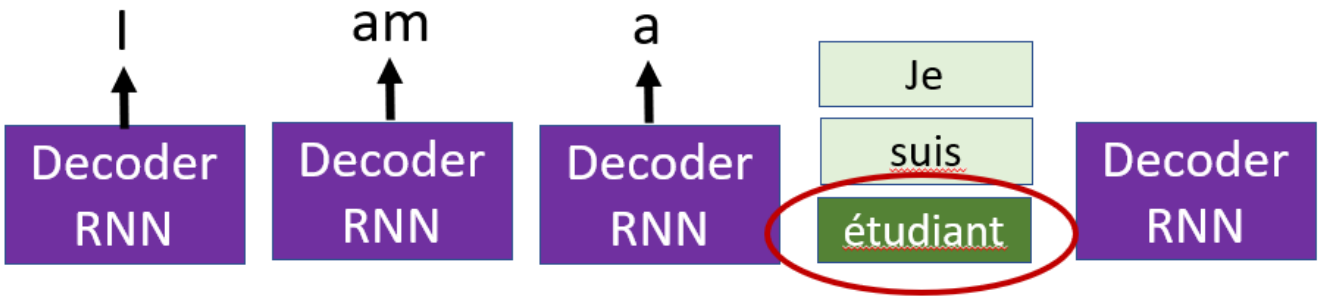


Figure 1: The Transformer - model architecture.

Bottleneck of Sequence-to-Sequence Model

- It is challenging for the model to deal with long sentences
- **Attention**
  - The encoder passes all the hidden states to the decoder
  - The attention enables the decoder to focus on the word before it generates the English translation
  - This ability amplifies the signal from the relevant part of the input sentence



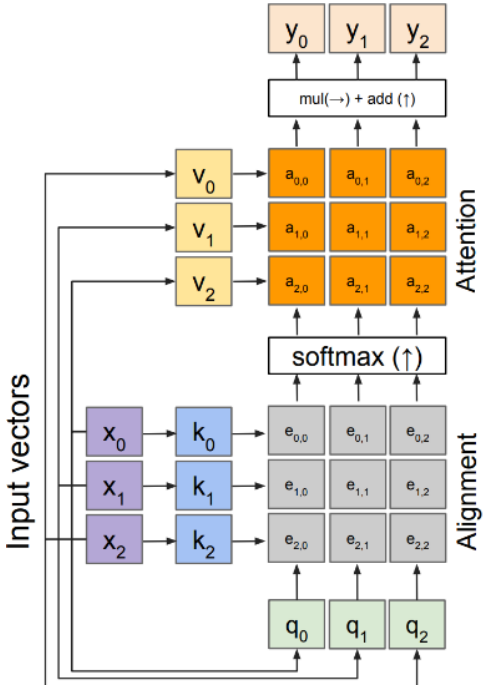
Transformer Model Architecture

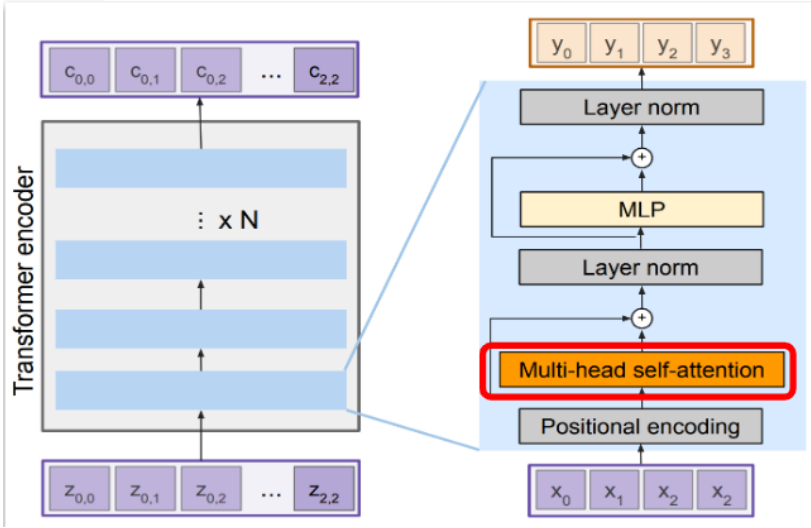
- $Q = X \cdot W^Q$
- $K = X \cdot W^K$
- $V = X \cdot W^V$

Q: query
K: key
V: value

- Encoder – Multi-head self-attention

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

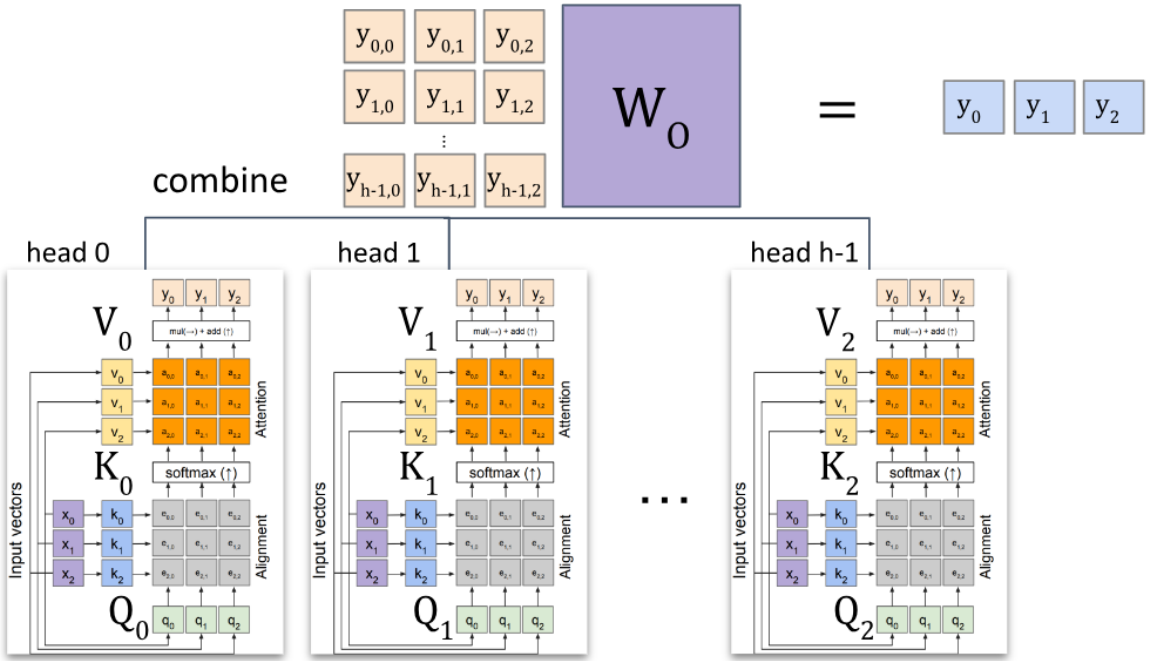




Transformer Model Architecture

- $Q_i = \mathbf{X} \cdot W^Q_i$
- $K_i = \mathbf{X} \cdot W^K_i$
- $V_i = \mathbf{X} \cdot W^V_i$

- Encoder – Multi-head self-attention

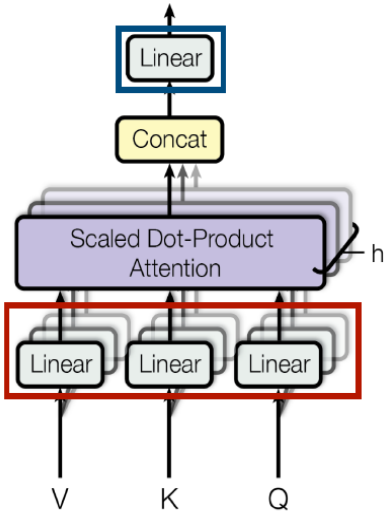


Multi-head Self-attention

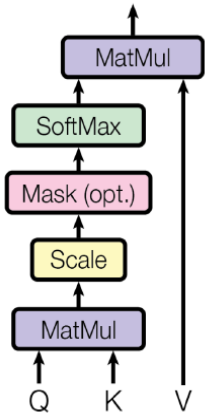
Multi-Head Self-Attention (MHSA)

- **Project** Q, K, and V with h **different** learned linear projections
- Perform the **scaled dot-product attention** function on each of Q, K, V **in parallel**
- **Concatenate** the output values
- **Project** the output values again, resulting in the final values

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$



Scaled Dot-Product Attention

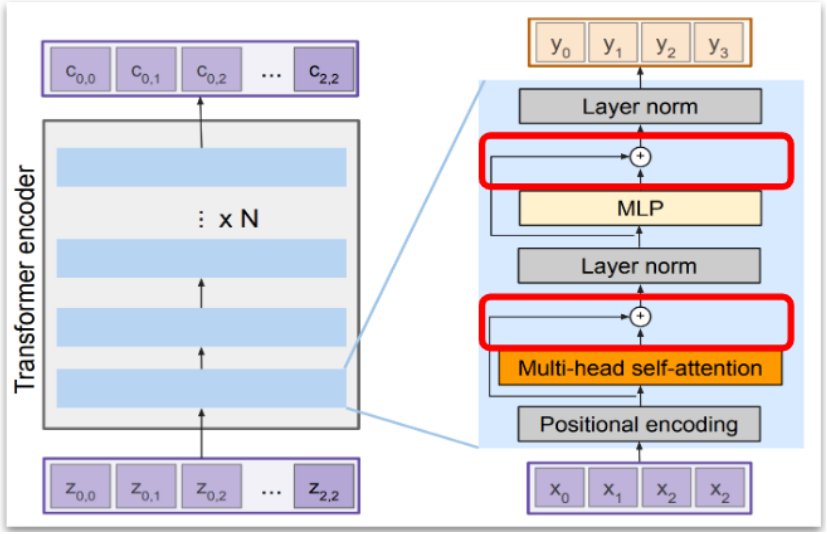


$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Transformer Model Architecture

- Encoder – Residual connection
  - Address exploding/vanishing gradients
  - Equation
    - F(x): output of previous layer
    - x: input of previous layer

$$y = x + F(x)$$



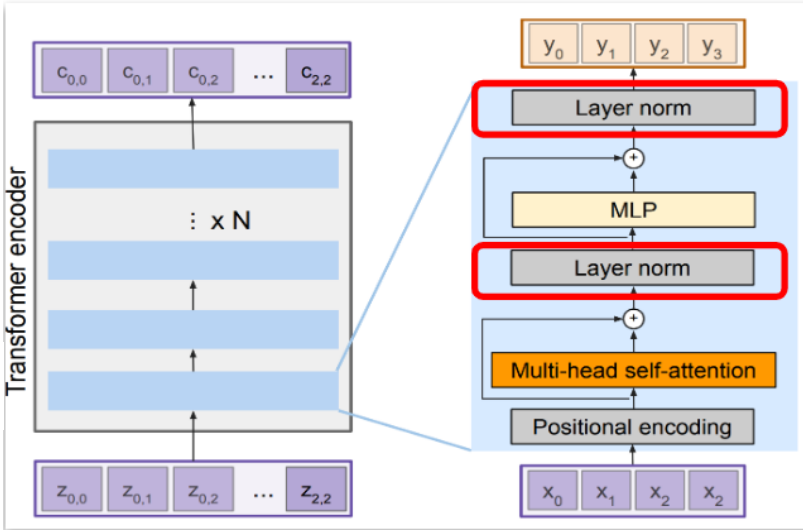
Transformer Model Architecture

- Encoder – Layer normalization
  - For each input vector x = (x1 ..Xm)
    - Calculate Mean, variance
    - Normalize the vector x
      - Small ε -> avoid dividing 0
    - Scale and shift

γ, β are learnable parameters

$$\mu = \frac{\sum_i x_i}{m} \qquad \hat{x}_i = \frac{(x_i - \mu)}{\sqrt{\sigma^2 + \varepsilon}}$$

$$\sigma^2 = \frac{\sum_i (x_i - \mu)^2}{m} \qquad y_i = \gamma \hat{x}_i + \beta$$



Transformer Model Architecture

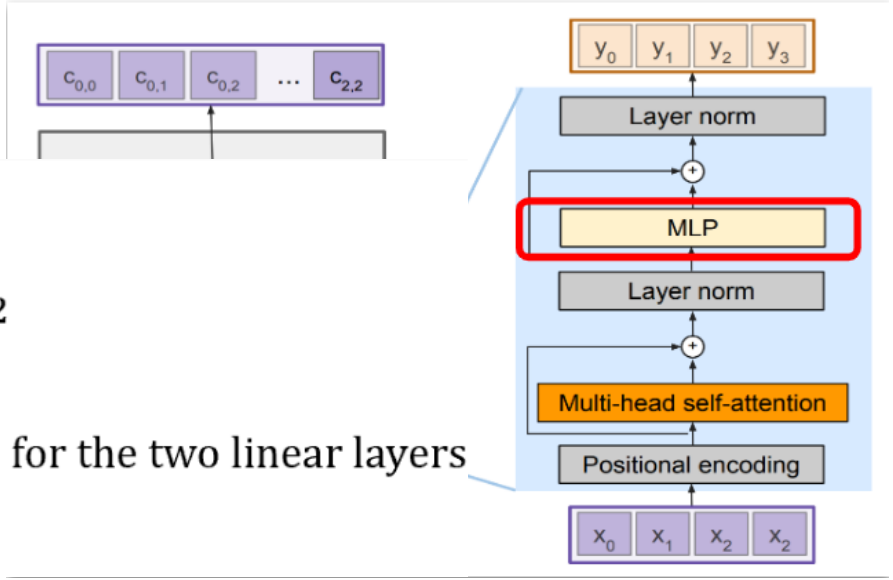
- Encoder – Feed Forward Network (FFN)
  - Contains 2 linear transformations and 1 ReLU

$$FFN(x) = ReLU(W_1 x + b_1)W_2 + b_2$$
$$= \max(0, W_1 x + b_1)W_2 + b_2$$

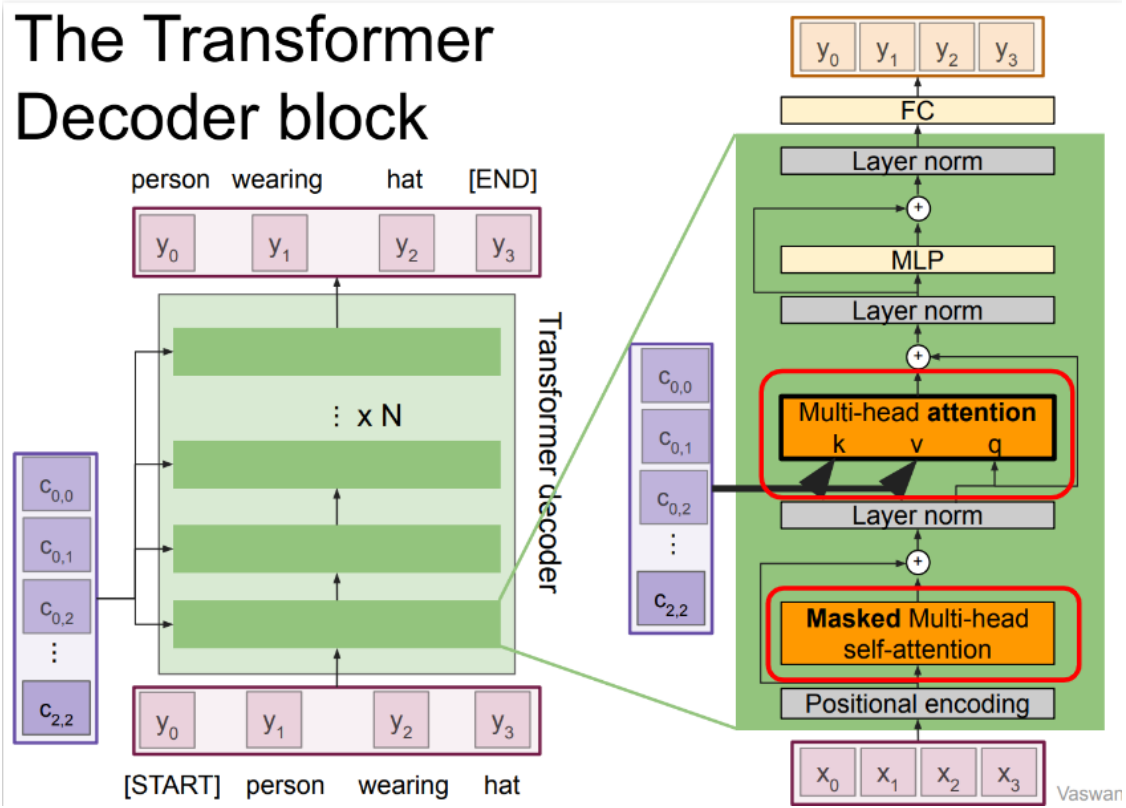
x: output of the previous layer

$W_{1(d_{model} \times d_{ff})}, W_{2(d_{ff} \times d_{model})}$: weight matrices for the two linear layers

b_1, b_2: biases for the two linear layers

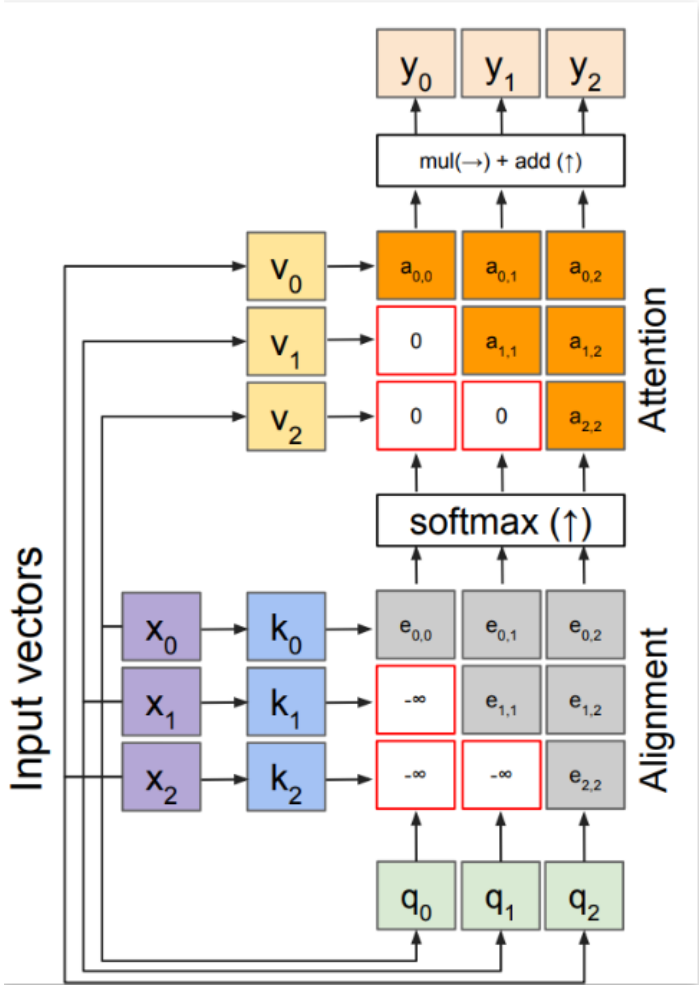


Transformer Model Architecture



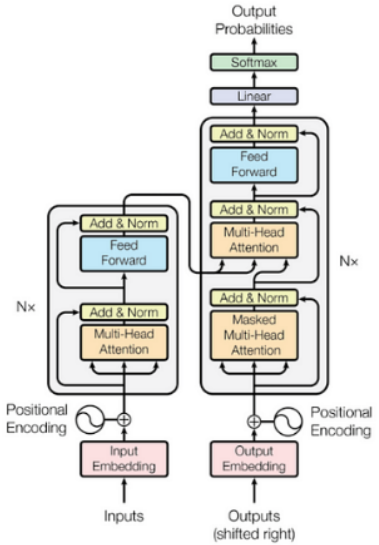
Transformer Model Architecture

- Decoder – Mask self-attention
  - The output of a certain position can only depend on the words on the previous positions
  - Set alignment scores of successive position to negative infinity

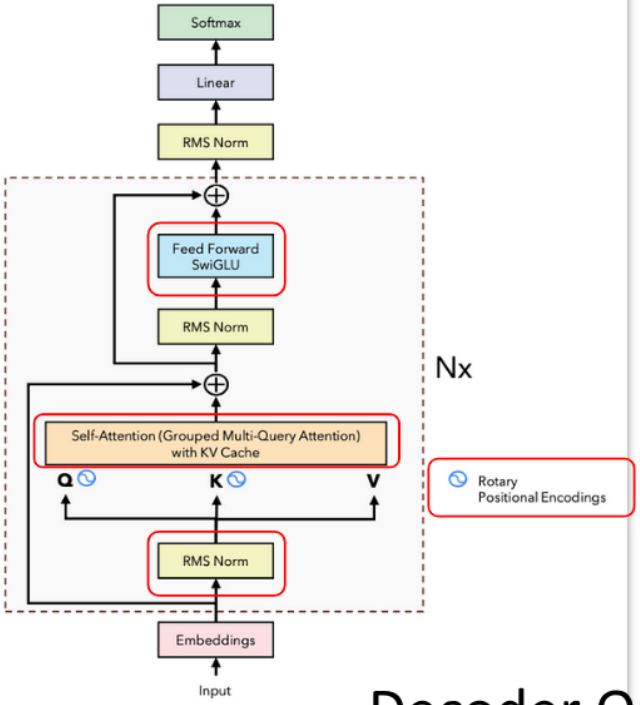


LLaMA Model Architecture

Transformer vs LLaMA



Transformer
("Attention is all you need")



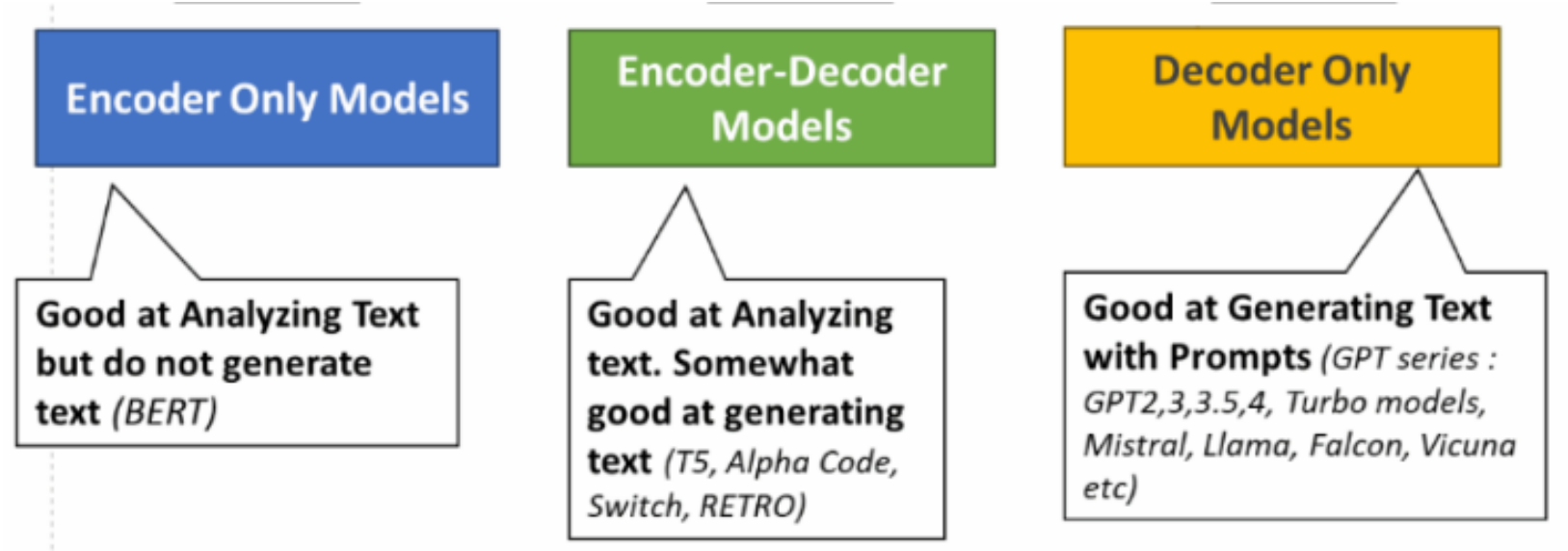
LLaMA

Decoder Only!

img src: medium @ccibeekeoc42

LLaMA Model Architecture

- Why LLaMA is decoder-only model?



https://www.53ai.com/news/qianyanjishu/1539.html

LLaMA Model Architecture

- Difference between Transformer and LLaMA
  - Decoder-only model
  - Pre-Norm (root mean square (RMS) norm)
  - Rotary positional embedding (RoPE)
  - KV cache
  - Grouped multi query attention
  - SwiGLU activation function rather than ReLU in FFN

LLaMA Model Architecture

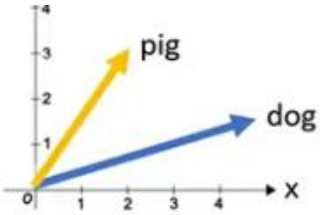
- Rotary positional embedding (RoPE)
  - Combine absolute and relative encoding
    - Absolute positional embedding
      - Assigns a unique vector to each position and doesn't scale well to capture relative position
    - Relative embeddings
      - Focuses on the distance between tokens
      - Enhance the model's understanding of token relationship
  - Rotational mechanism
    - Each position in the sequence is represented by a rotation in the embedding space

LLaMA Model Architecture

- Rotary positional embedding (RoPE)
  - RoPE applies a rotation to the word vector
  - The equation incorporates a rotation matrix that rotates a vector by an angle of Mθ, where M is the absolute position in the sentence.
  - This rotation is applied to the query and key vectors in the self-attention

$$f_{\{q,k\}}(\boldsymbol{x}_m, m) = \begin{pmatrix} \cos m\theta & -\sin m\theta \\ \sin m\theta & \cos m\theta \end{pmatrix} \begin{pmatrix} W^{(11)}_{\{q,k\}} & W^{(12)}_{\{q,k\}} \\ W^{(21)}_{\{q,k\}} & W^{(22)}_{\{q,k\}} \end{pmatrix} \begin{pmatrix} x^{(1)}_m \\ x^{(2)}_m \end{pmatrix}$$

The pig chased the dog



LLaMA Model Architecture

- Root Mean Square (RMS) normalization
  - Calculate the RMS of the input vectors rather than the mean and variance
  - Efficient normalization
    - No subtracting mean before squaring
    - No shifting in implementation

$$\mathrm{RMS}(x) = \sqrt{\frac{1}{m}\sum_i x_i^2}$$

$$\hat{x}_i = \frac{x_i}{\mathrm{RMS}(x)+\varepsilon}$$

$$y_i = \gamma \hat{x}_i$$

$$\mu = \frac{\sum_i x_i}{m}$$

$$\sigma^2 = \frac{\sum_i (x_i-\mu)^2}{m}$$

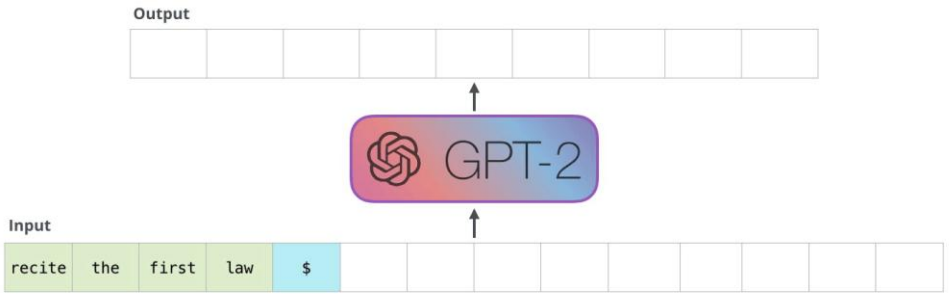
$$\hat{x}_i = \frac{(x_i-\mu)}{\sqrt{\sigma^2+\varepsilon}}$$

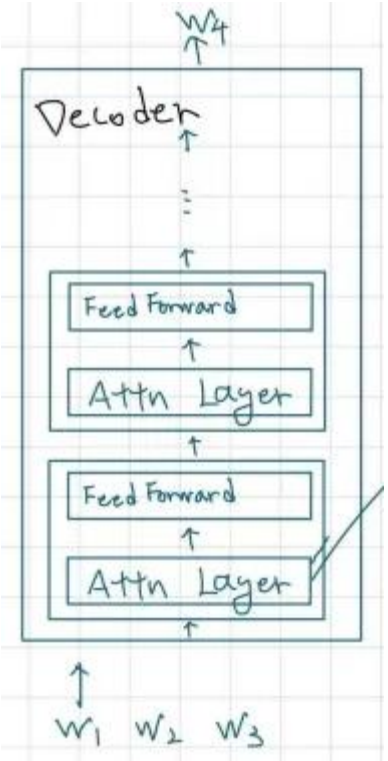
$$y_i = \gamma \hat{x}_i + \beta$$

Layer norm

LLaMA Model Architecture

- LLM decoder
  - The decoder works in an auto-regressive fashion
    - Given an input, the model predict the next token
    - Taking the combined input in the next step

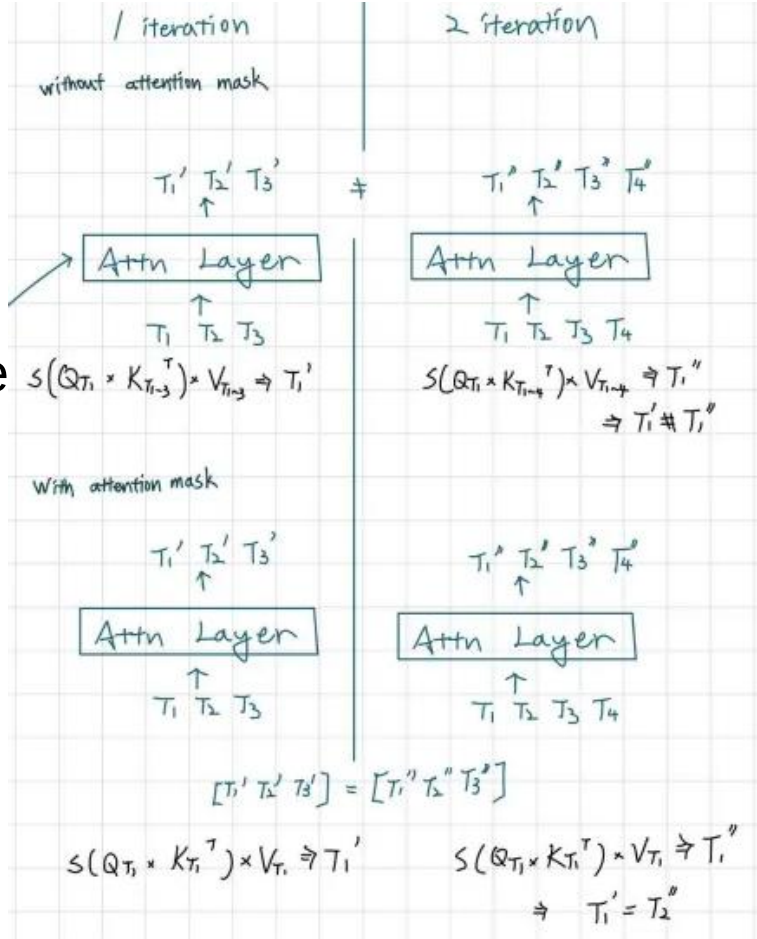




https://medium.com/@joaolages/kv-caching-explained-276520203249

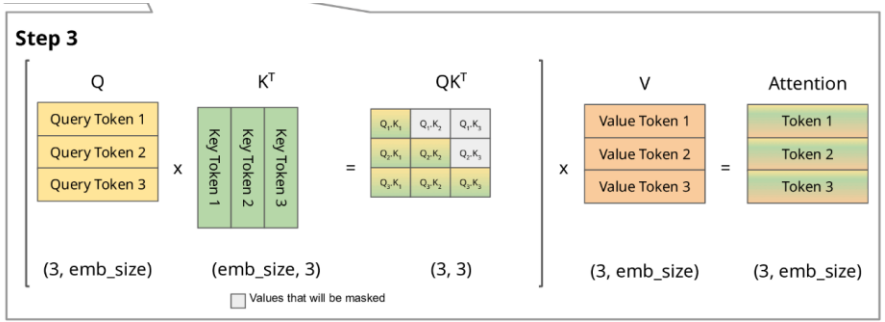
LLaMA Model Architecture

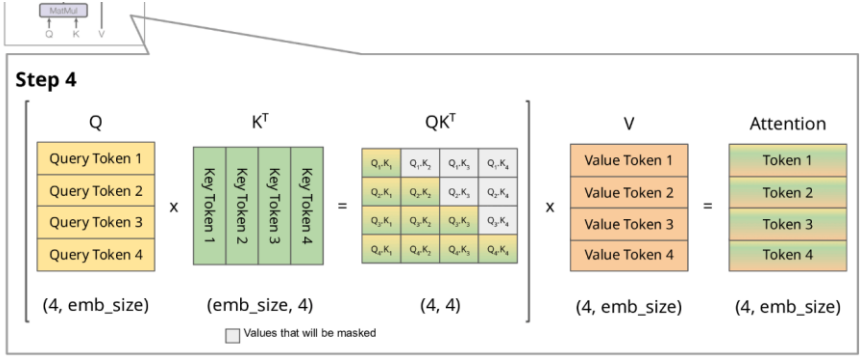
- Sequence mask (in decoder)
  - The decoder cannot see the message in the coming future
  - Use the mask to enable the decoder to only rely on the previous outputs to do the inference → training the decoder



LLaMA Model Architecture

- The scaled dot-product attention
  - The attention of a token only depends on its preceding tokens
  - At each generation step we are recalculating the same previous token attention, when we actually just want to calculate the attention for the new token

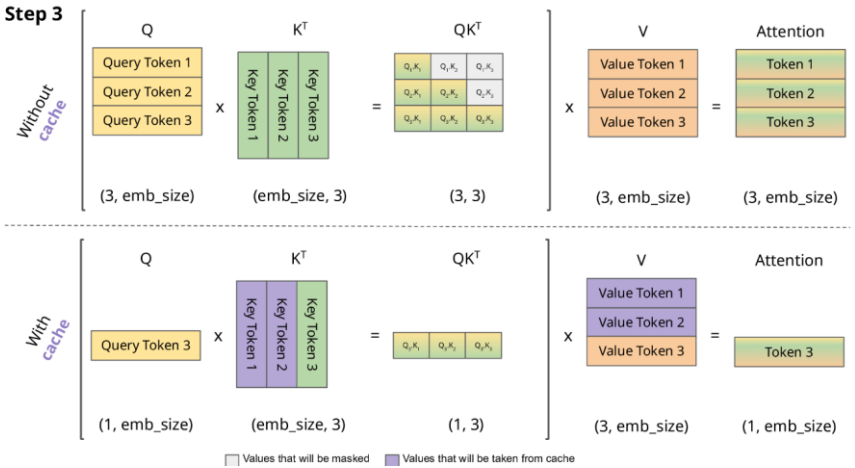


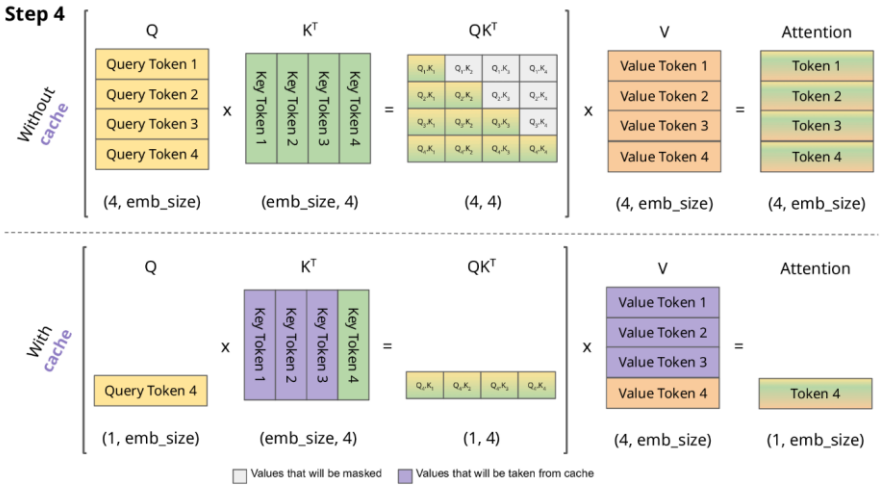


https://medium.com/@joaolages/kv-caching-explained-276520203249

LLaMA Model Architecture

- Key-Value (KV) cache
  - By caching the previous Keys and Values, we can focus on only calculating the attention for the new token.

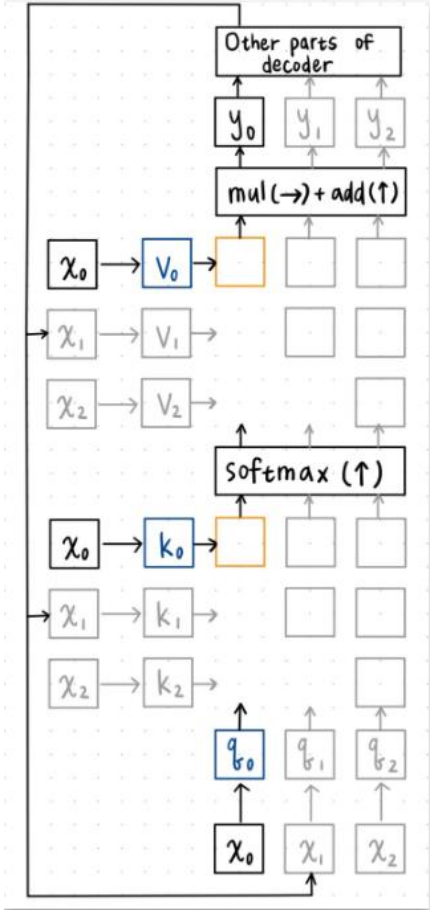




https://medium.com/@joaolages/kv-caching-explained-276520203249

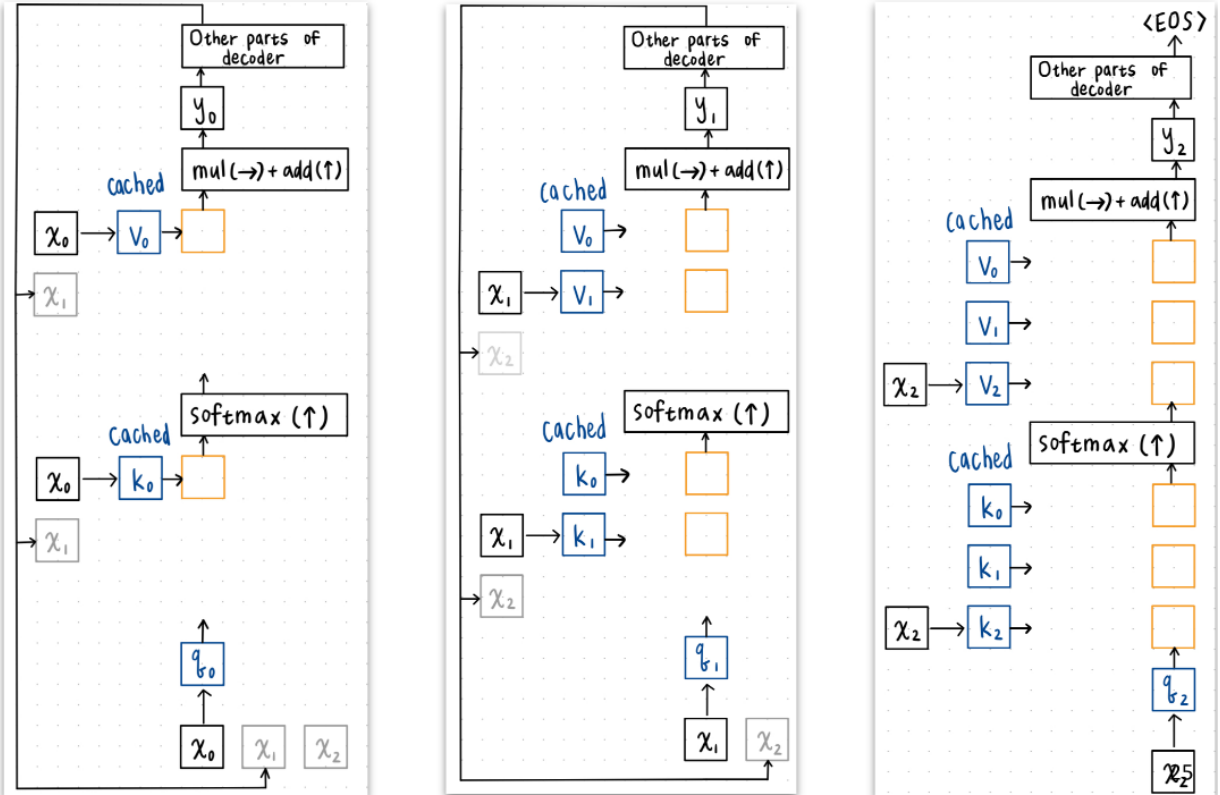
LLaMA Model Architecture

- Key-Value (KV) cache
  - LLM models can generate only one token at a time
  - Each new prediction is dependent on the previous context
    - To predict token number 1000 in the generation, you need information from the previous 999 tokens
  - Optimize the sequential generation process by storing previous calculations to reuse in subsequent tokens, so they don't need to be computed again.



LLaMA Model Architecture

- KV cache



LLaMA Model Architecture

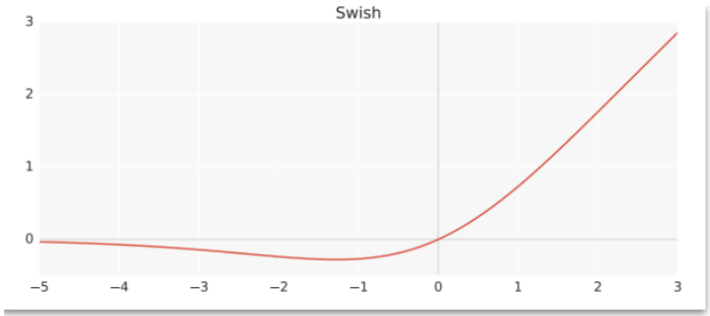
- Key-Value (KV) cache
  - The matrices obtained with KV caching are way smaller, which leads to faster matrix multiplications
  - The downside of the KV cache is
    - When the length of sequences is becoming long
    - Needs the large memory to cache the Key and Value states

LLaMA Model Architecture

- SwiGLU (Swish and Gated Linear Unit)
  - LLU such as PALM and LLAMA use SwiGLU in FFN rather than the usual ReLU -> SwiGLU tackles minus value better than ReLU

$$\mathrm{FFN}_{SwiGLU}(x, W, V, W_2) = (Swish_1(xW) \otimes xV)W_2,$$

$$\text{where } Swish_\beta(x) = x \cdot sigmoid(\beta x) = \frac{x}{1+e^{-\beta x}}$$



img src: medium@neuralnets

Takeaway Questions

- What's problem the "Attention" aiming to solve?
  - (A) Gradient vanishing
  - (B) Message passing in the long sequence of data
  - (C) Over-fitting
- What are benefits of the "Transformer" ?
  - (A) Large hidden layer
  - (B) The amount of computation is small
  - (C) More data parallelism

Takeaway Questions

- How does the “self-attention” help the encoder?
  - (A) Looking at other works in the input sentence
  - (B) Memorizing the more messages within a network
  - (C) Focus on a specific word