AY 2024/2025 S1

EE6427 Video Signal Processing

Part 4 AI Models & Architectures

Dr Yap Kim Hui

Room: S2-B2b-53
Tel: 6790 4339
Email: ekhyap@ntu.edu.sg

References

- Stanford Lecture Notes, CS231n: Convolutional Neural Networks for Visual Recognition.
- Ian Goodfellow, Yoshua Bengio and Aaron Courville, Deep Learning, MIT Press, http://www.deeplearningbook.org
- Pytorch Tutorial. https://pytorch.org/tutorials/
- University of Wisconsin Madison Lecture Notes, CS638. University of Michigan, EECS 498-007 / 598-005: Deep Learning for Computer Vision
- Simplilearn, Recurrent Neural Network Tutorial
- Shusen Wang, Transformer Model, Youtube Online Videos
- Jay Alammar, The Illustrated Transformer.

Part 4 Outline

- AI Models & Architectures
  - Convolutional Neural Network (CNN)
  - Recurrent Neural Network (RNN)
  - Long Short-Term Memory (LSTM)
  - Transformer

Section I
Convolutional Neural Network (CNN)

Section I Overview

- The section covers the following topics:
  - Introduction
  - Linear Classifier
  - Convolutional Neural Networks (CNNs)
  - CNN Training & Optimization
  - Well-Known CNN Architectures
  - Applications

Different Deep Neural Network (DNN) Architectures

Common DNN Architectures

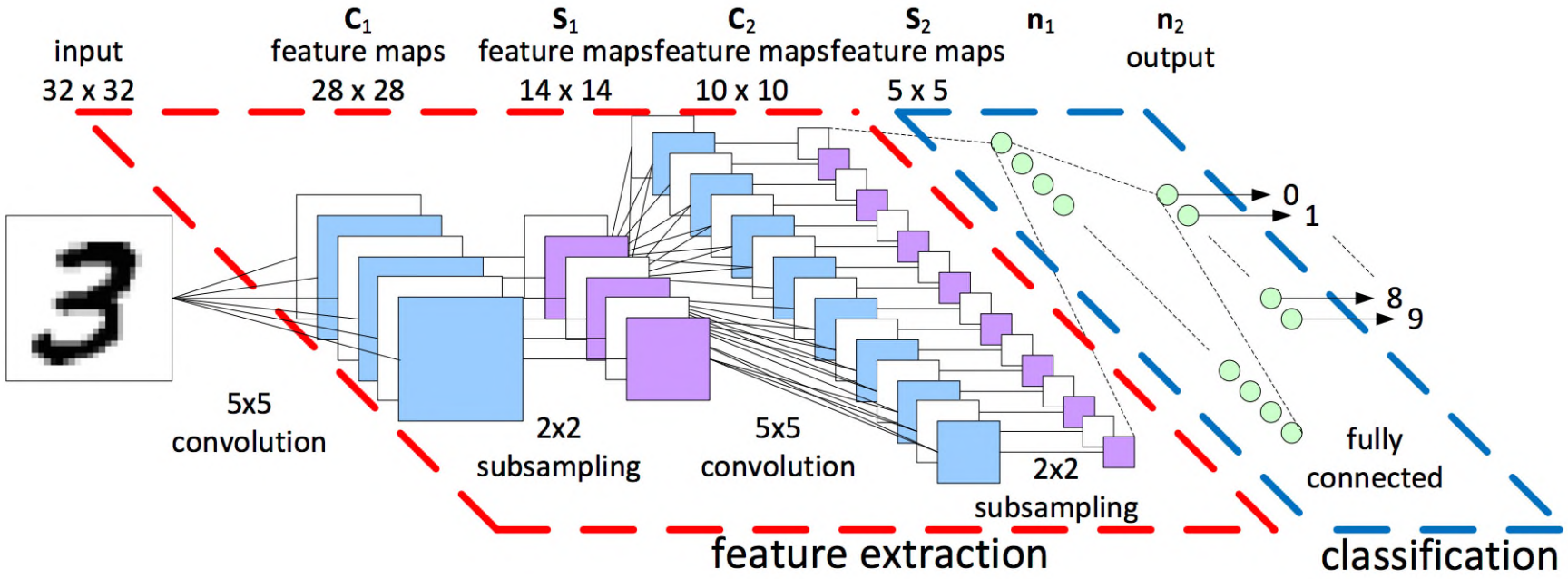
- Convolutional Neural Networks (CNNs)
- Recurrent Neural Networks (RNNs)
- Transformers
- Large Language Models (LLMs)
- Etc.

Why Different DNN Models?

- Different DNNs (tools) for different applications.
- Each DNN has its own unique features to address unique problem.

Convolutional Neural Network (CNN)

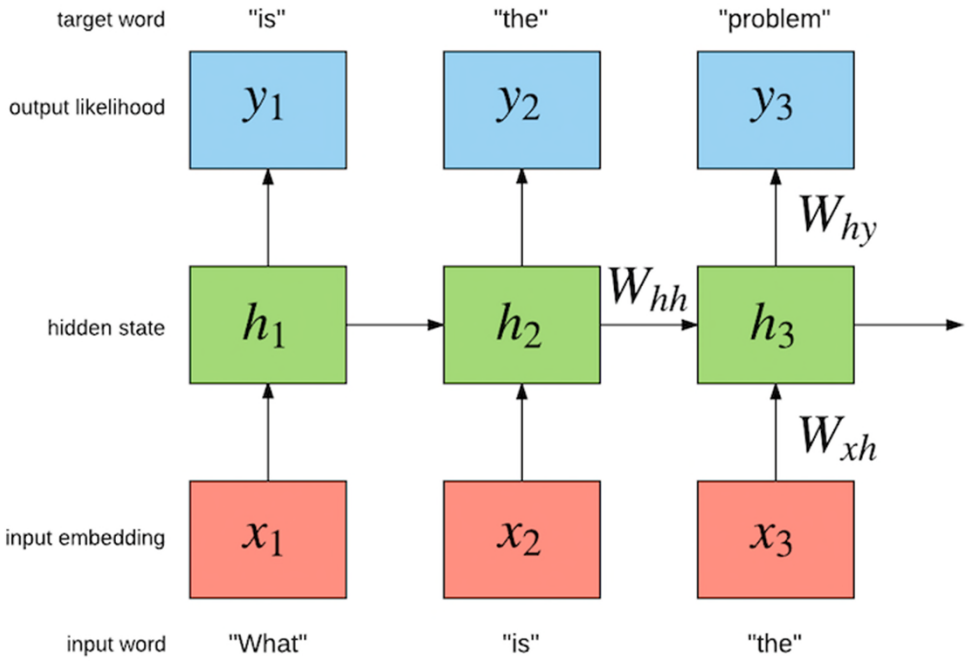
- Consist of deep layers to extract progressively higher-level abstraction features.
- Commonly used in classification and regression applications.



Source: “Efficiency Optimization of Trainable Feature Extractors for a Consumer Platform”, M.Peemen et al., 2011

Recurrent Neural Network (RNN)

- A type of neural network that specializes in processing sequences.
- Commonly used in applications involving time-series and state-series prediction and modelling.
- Example applications: stock price prediction, language translation.



Source: “Language modeling a billion words”, torch blog

Transformer

- A type of network that uses attention mechanism to process input sequence in parallel.
- Good at modelling long-range dependency.
- Achieve state-of-the-art performance in many vision and NLP applications.

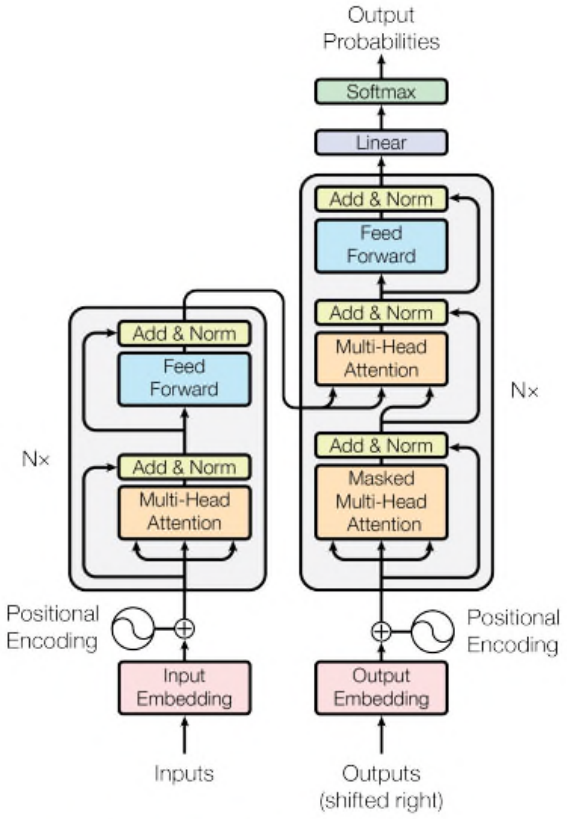
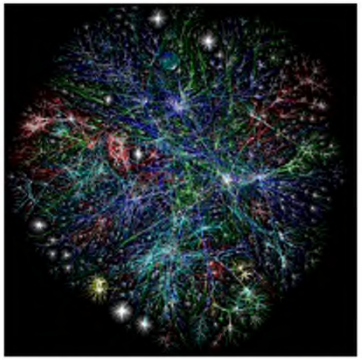


Figure 1: The Transformer - model architecture.

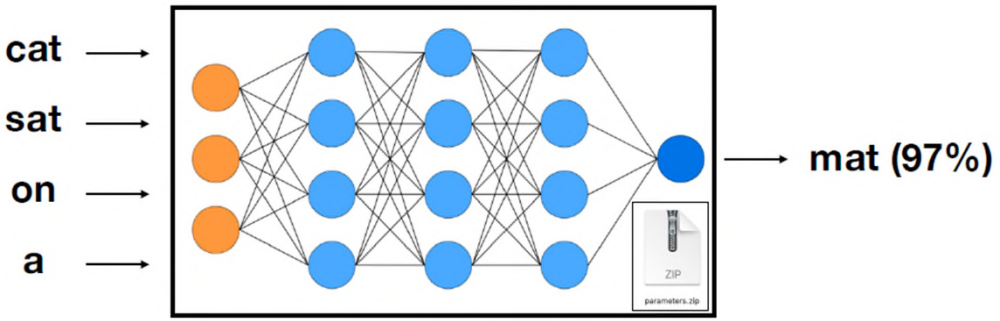
Large Language Models (LLMs)

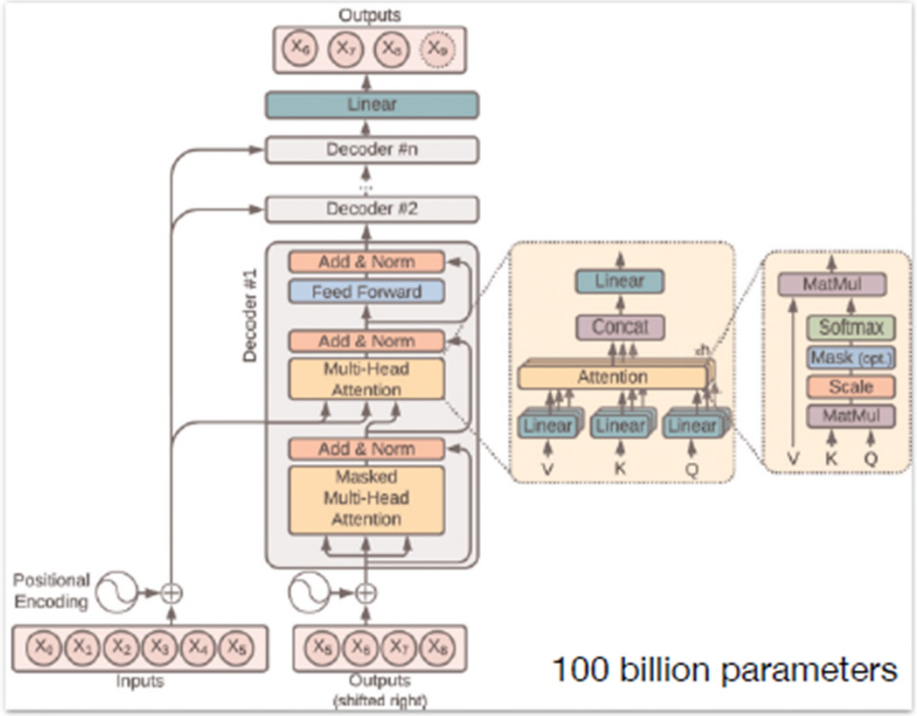


Chunk of the internet,
~10TB of text

6,000 GPUs for 12 days, ~$2M
~1e24 FLOPS

~140GB file





Source: Intro to Large Language Models, Andrej Karpathy, Nov 2023

Foundation Models (FMs)

- FMs are models that are trained on large scale broad data that can be adapted (finetuned) to a wide range of downstream tasks / applications.
- Examples: LLMs (e.g., GPT), Vision-Language Models (VLMs) (e.g., CLIP).

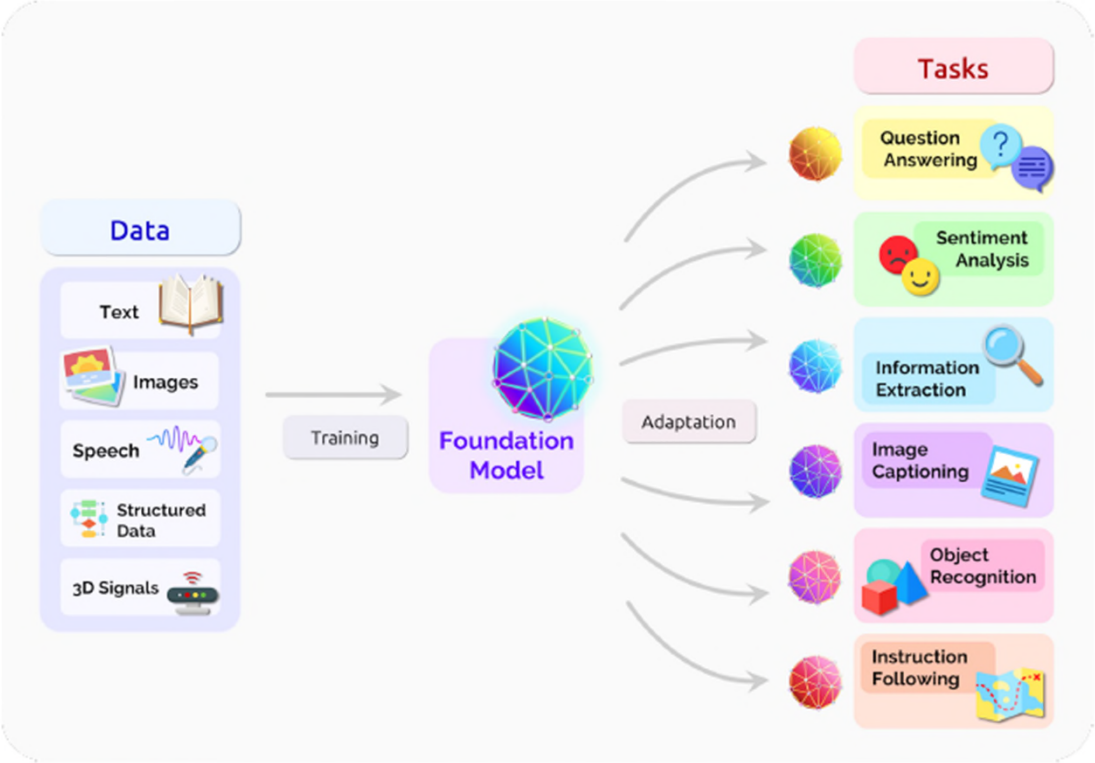


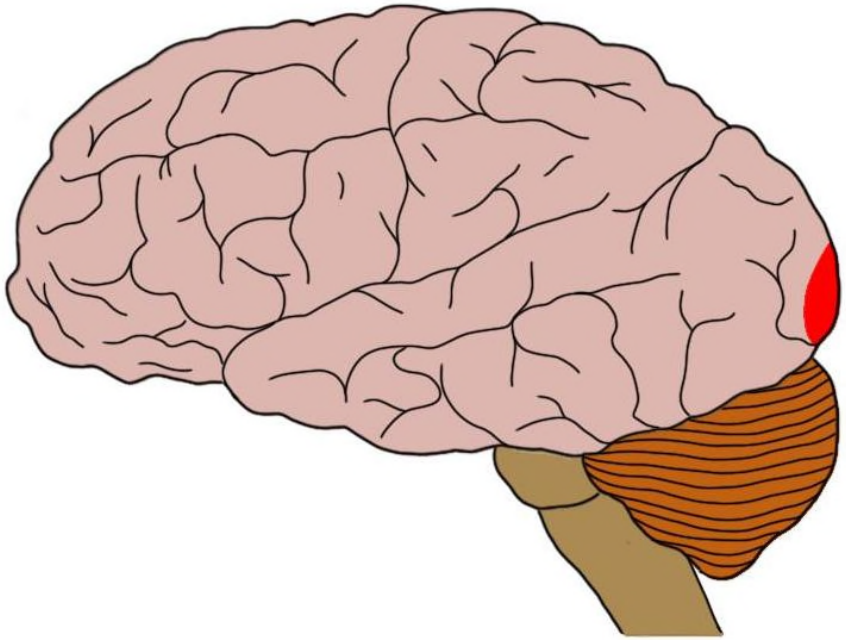
Fig. 2. A foundation model can centralize the information from all the data from various modalities. This one model can then be adapted to a wide range of downstream tasks.

Source: Rishi Bommasani, On the Opportunities and Risks of Foundation Models.

Linear Classifiers

Brain Visual Cortex

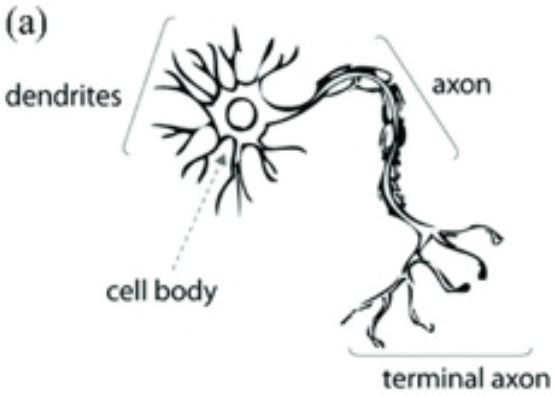
- Primary cortical region of brain that receives, integrates, and processes visual information relayed from the retinas.

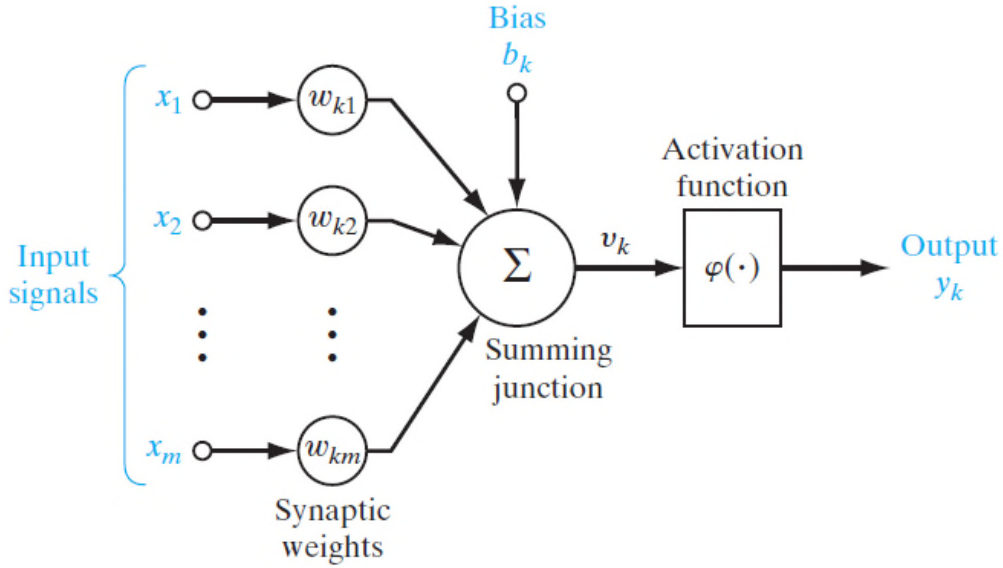


Source: ncbi.nlm.nih.gov

Biological Neural Network

- Multiple neurons interconnected together to form biological neural network.
- Perceptron model: artificial model of a biological neuron.

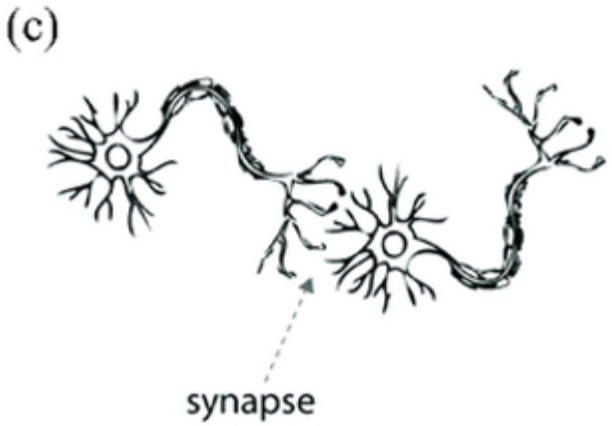


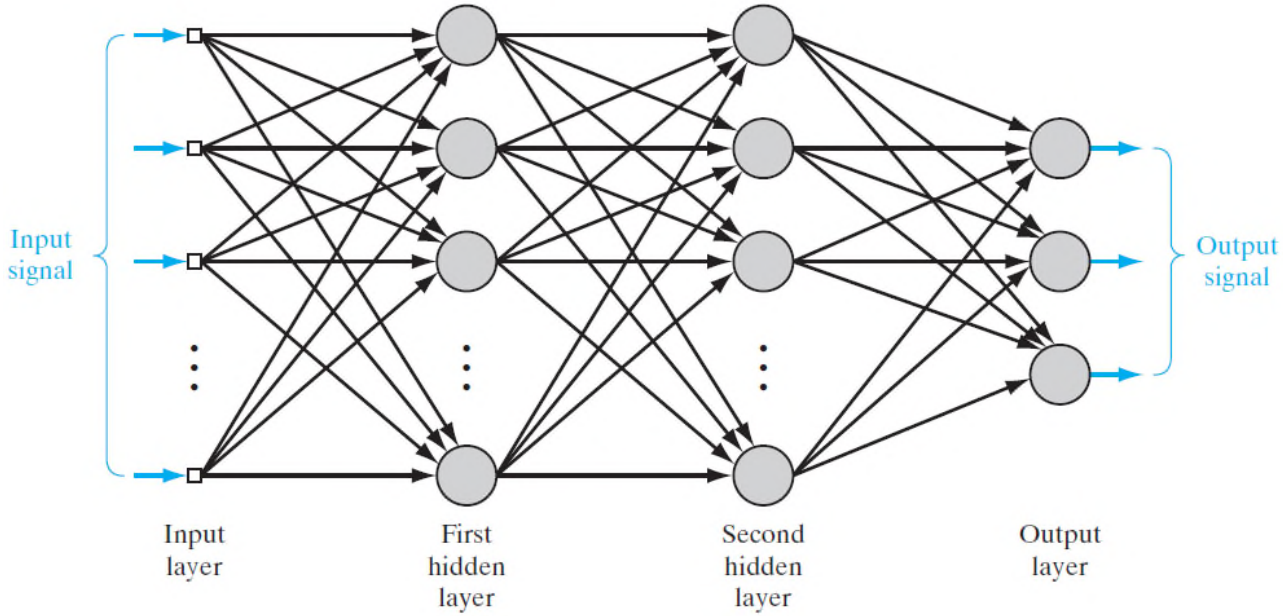


Source: “Neural Networks and Learning Machines”, Siamak Azodolmolky

Multilayer Perceptron (MLP)

- MLP is consisted of interconnection of perceptrons.
- Also known as Feed-Forward (FF) or dense network.





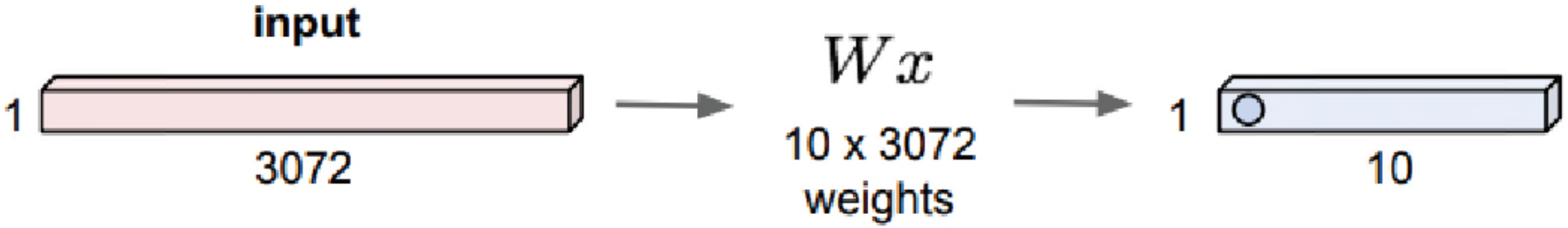
Source: Zhenzhu Meng, et al. Using a Data Driven Approach to Predict Waves Generated by Gravity Driven Mass Flows

Linear Classifier

- Linear classifier
  - Make a classification decision based on a linear combination of input data
- E.g., linear classifier for 10 class labels (e.g., animal classes)

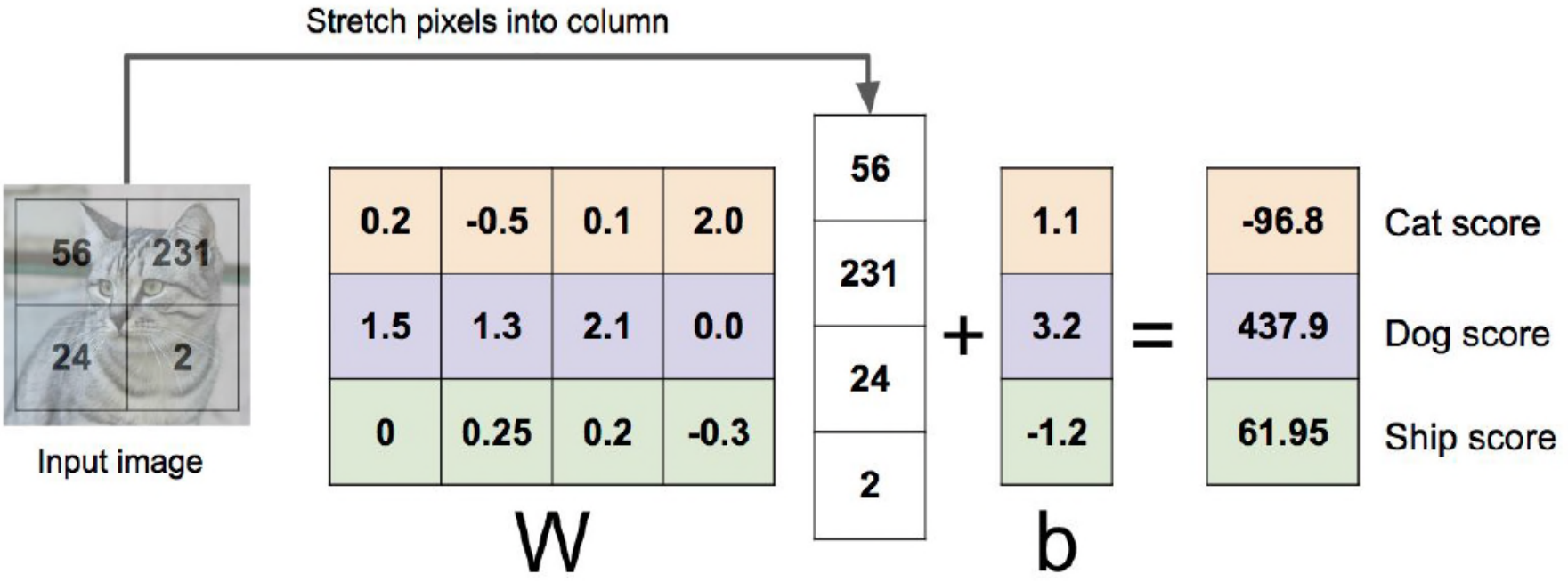
32*32*3 image -> stretch to 3072*1

$$f(x) = Wx + b$$



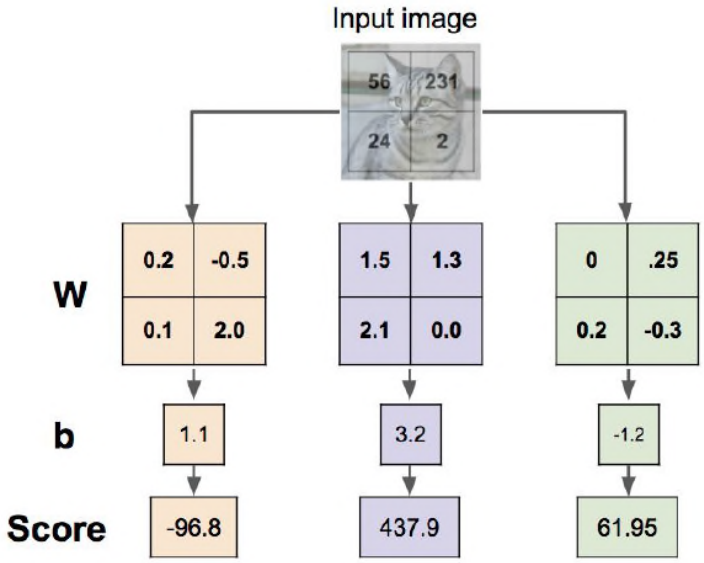
Source: Stanford Lecture Notes, CS231n

Algebraic Viewpoint



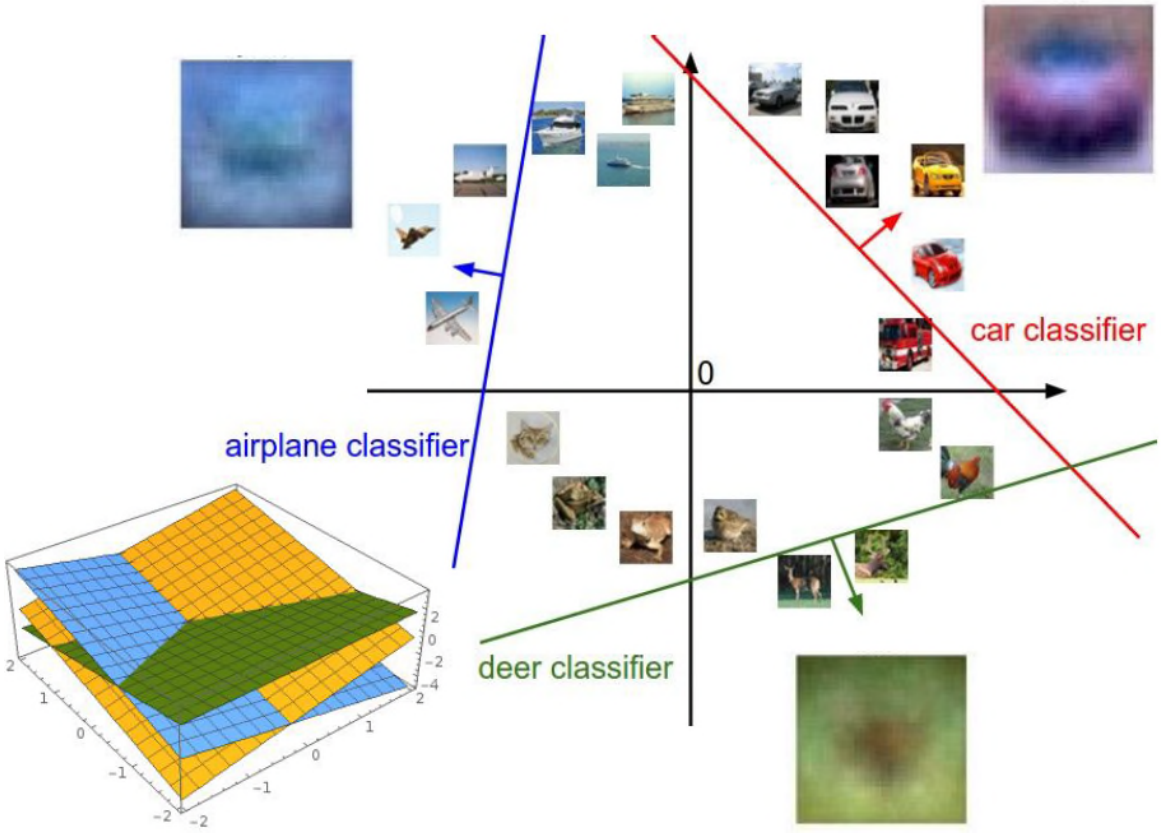
Source: Stanford Lecture Notes, CS231n

Visual Viewpoint



Source: Stanford Lecture Notes, CS231n

Geometric Viewpoint



$$f(x,W) = Wx + b$$

Array of **32x32x3** numbers
(3072 numbers total)

Source: Stanford Lecture Notes, CS231n

Loss Function / Error Function

- How do we determine **W** and **b**?
- We need a loss function (error measurement) which is a metric/distance between predicted values and output target values (teachers) during training.

Loss Function

- The loss function will depend on what type of problems we are addressing.
- Two common problems:
  - Regression
    - Target output is continuous value.
    - E.g., share price prediction, rain fall estimation, etc.
  - Classification
    - Predict the label/category of input.
    - Target output is discrete (from a set of possible labels).
    - E.g., cancer classification (binary), face recognition (multi-class).

Common Regression Loss Functions

- Square loss:

$$L(x, y) = \sum_i \left(y_i - f(x_i)\right)^2$$

- Mean Square Error (MSE):

$$MSE = \frac{1}{N} \sum_i \left(y_i - f(x_i)\right)^2$$

- Mean Absolute Error (MAE):

$$MAE = \frac{1}{N} \sum_i |y_i - f(x_i)|$$

x and y are the input data and target output value, $f(x)$ is the network predicted output.

Common Classification Loss Function

- Softmax loss:
  - Cross-entropy loss with softmax normalization.

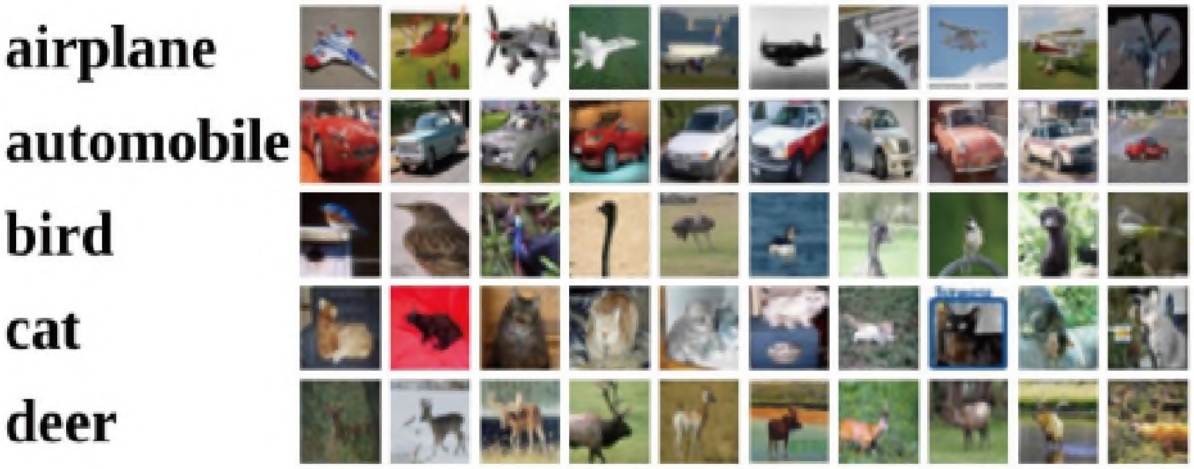
$$p_j = \frac{e^{z_j}}{\sum_k e^{z_k}}, \text{ where } z_j = f(x_j)$$

$$L = -\sum_j y_j \log_e p_j$$

Softmax Loss Example

- The output scores of a linear classifier for an input cat image are given as follows. What is the softmax loss for this training data?

$$p_j = \frac{e^{z_j}}{\sum_k e^{z_k}}$$



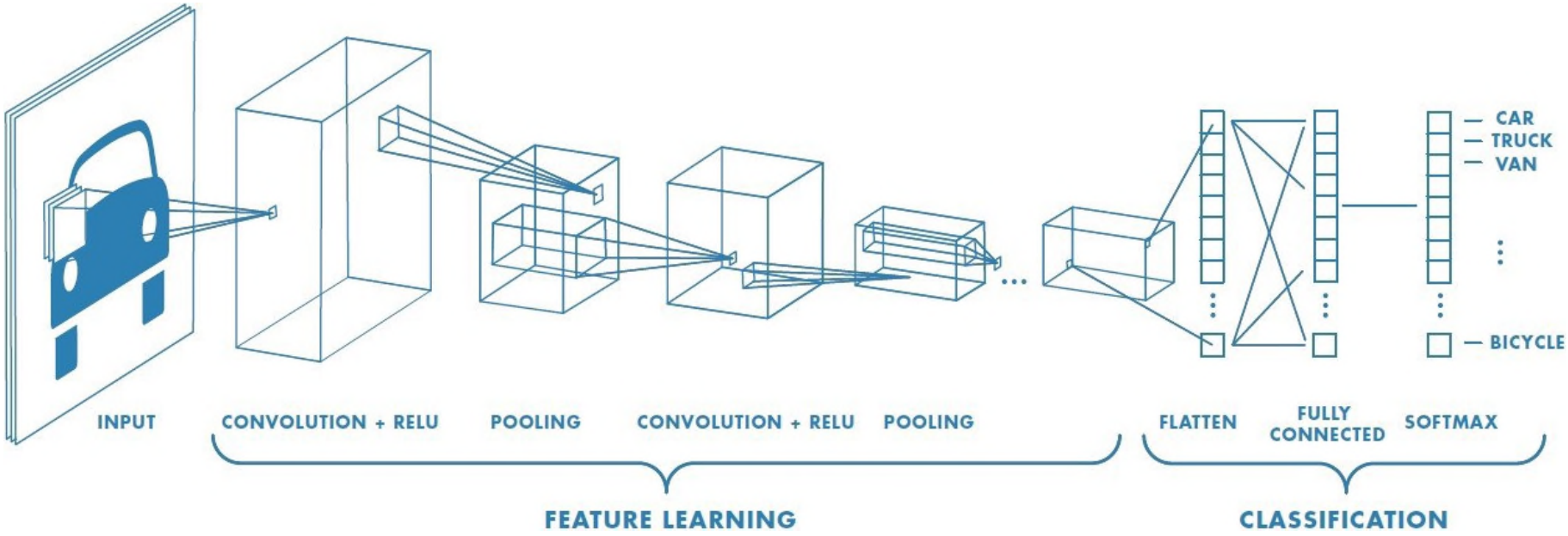
	Predicted output Scores (z)	Ground truth Output Labels (y)	Normalized probabilities (p)
airplane	0.7	0	0.109
automobile	0.1	0	0.060
bird	1.6	0	0.269
cat	2.2	1	0.489
deer	0.3	0	0.073

$$L = -\sum_j y_j \log p_j = 0.715$$

(Note: The log is natural log namely ln)

CNN Architecture

CNN Architecture



Source: “A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way”, towardsdatascience.com

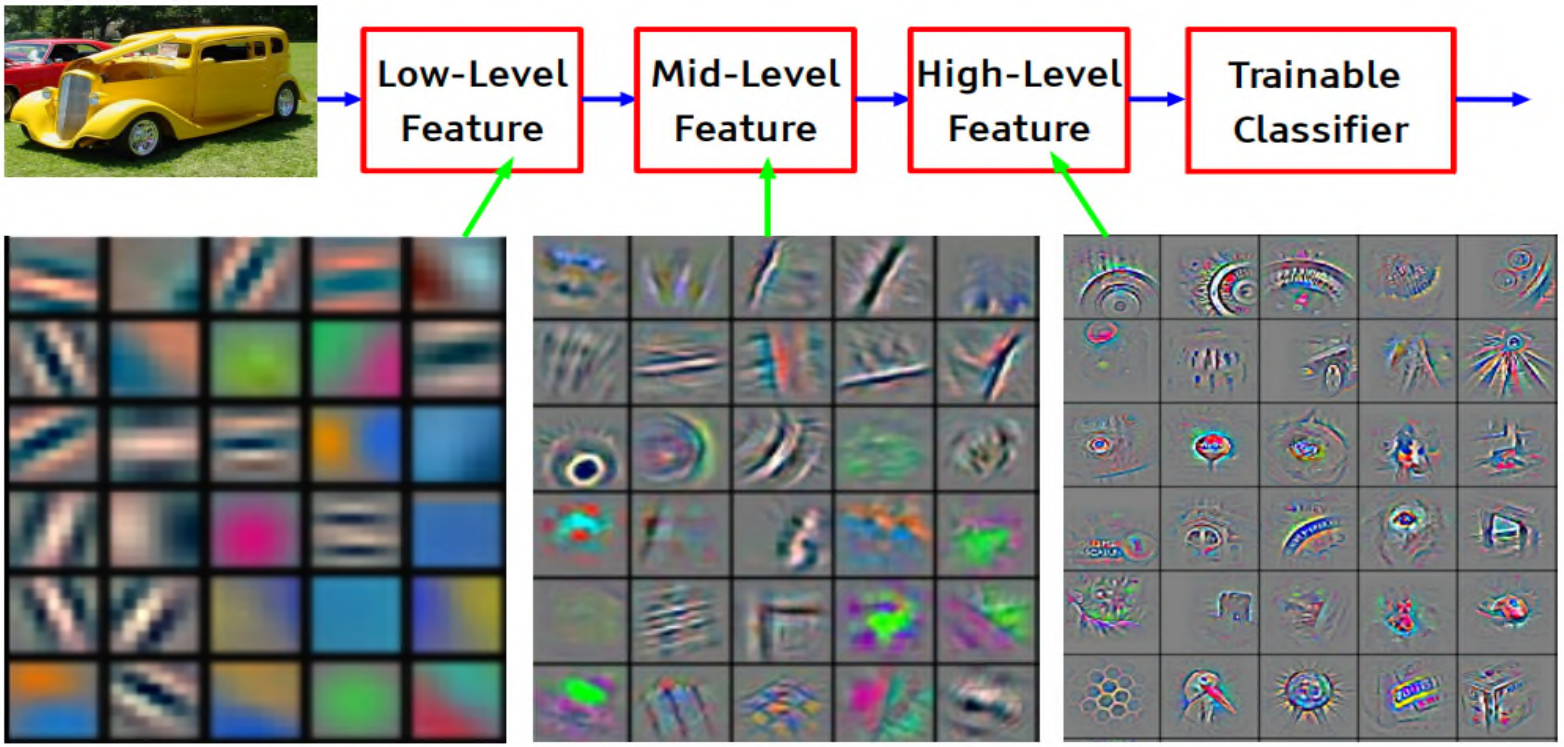
CNN Architecture

- Convolutional Layer
- Activation Function Layer
- Pooling Layer
- Fully-Connected (FC) Layer / Linear Layer
- Softmax Layer

Convolutional Layer

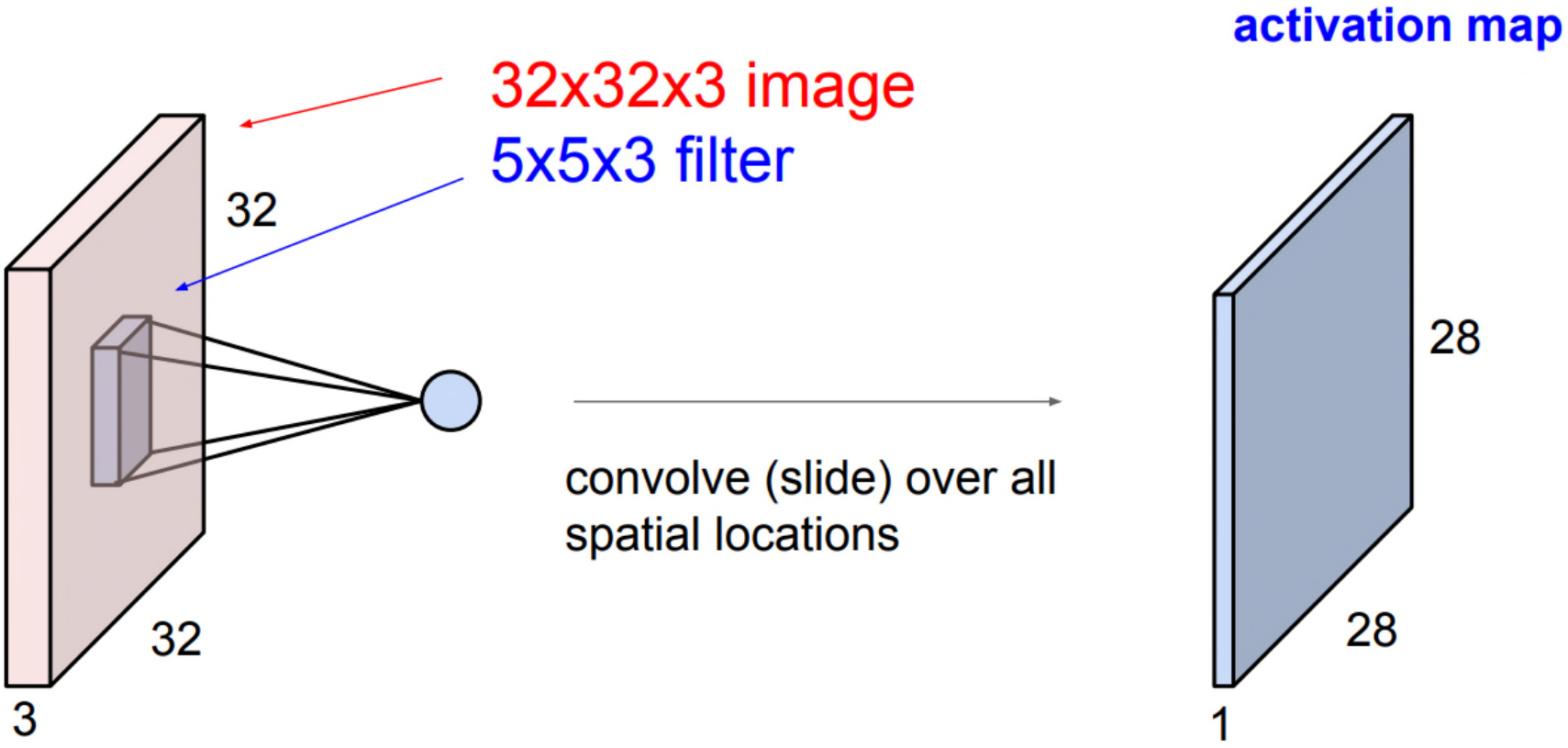
- Extract features in a hierarchical manner.
- Early layers extract low-level features whereas later layers extract high-level features.
- Reduce parameters to be trained by sharing weights through convolution kernel.

Convolutional Layer Feature Visualization



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

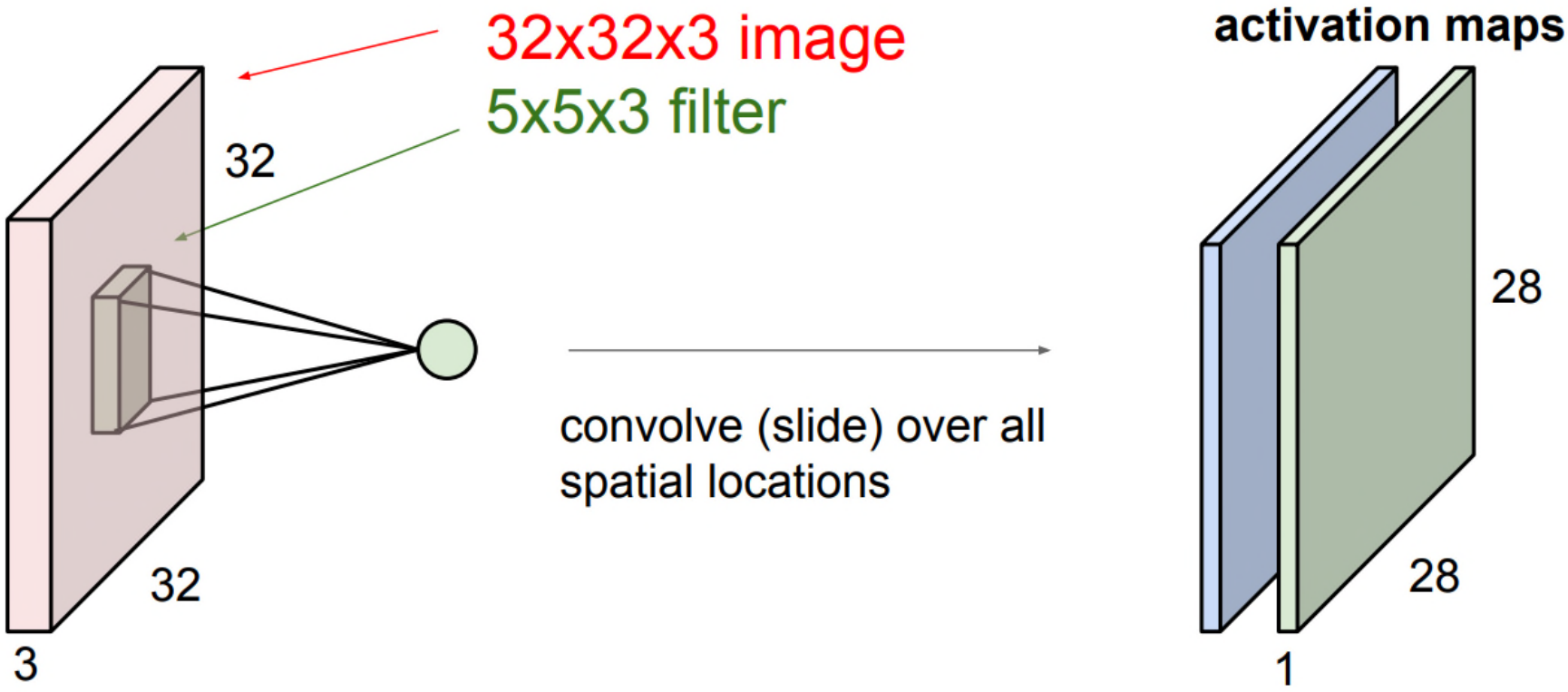
Convolutional Layer



Source: Stanford Lecture Notes, CS231n

Convolutional Layer

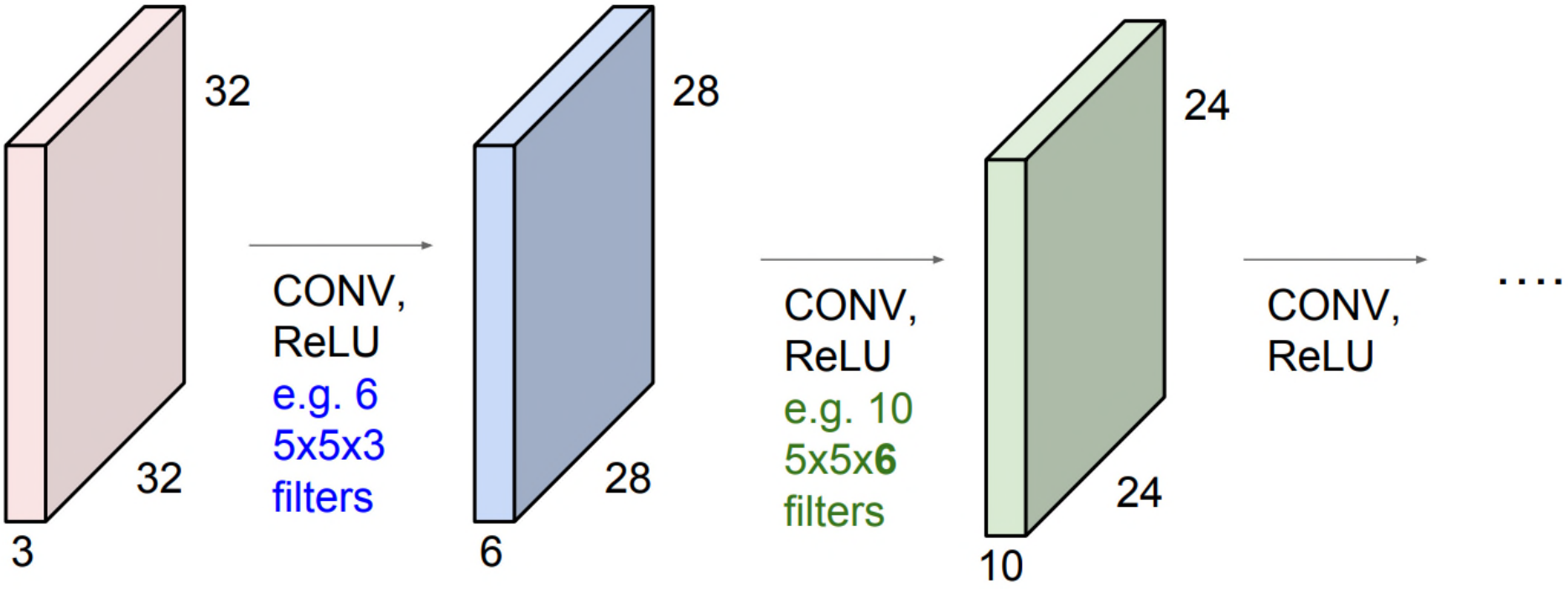
consider a second, green filter



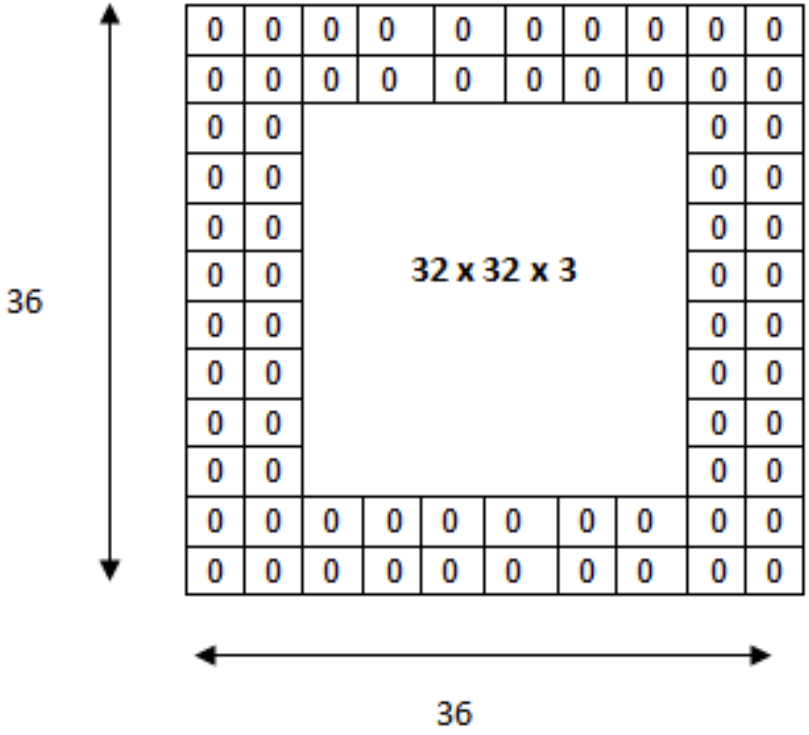
Source: Stanford Lecture Notes, CS231n

Convolutional Layers

Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



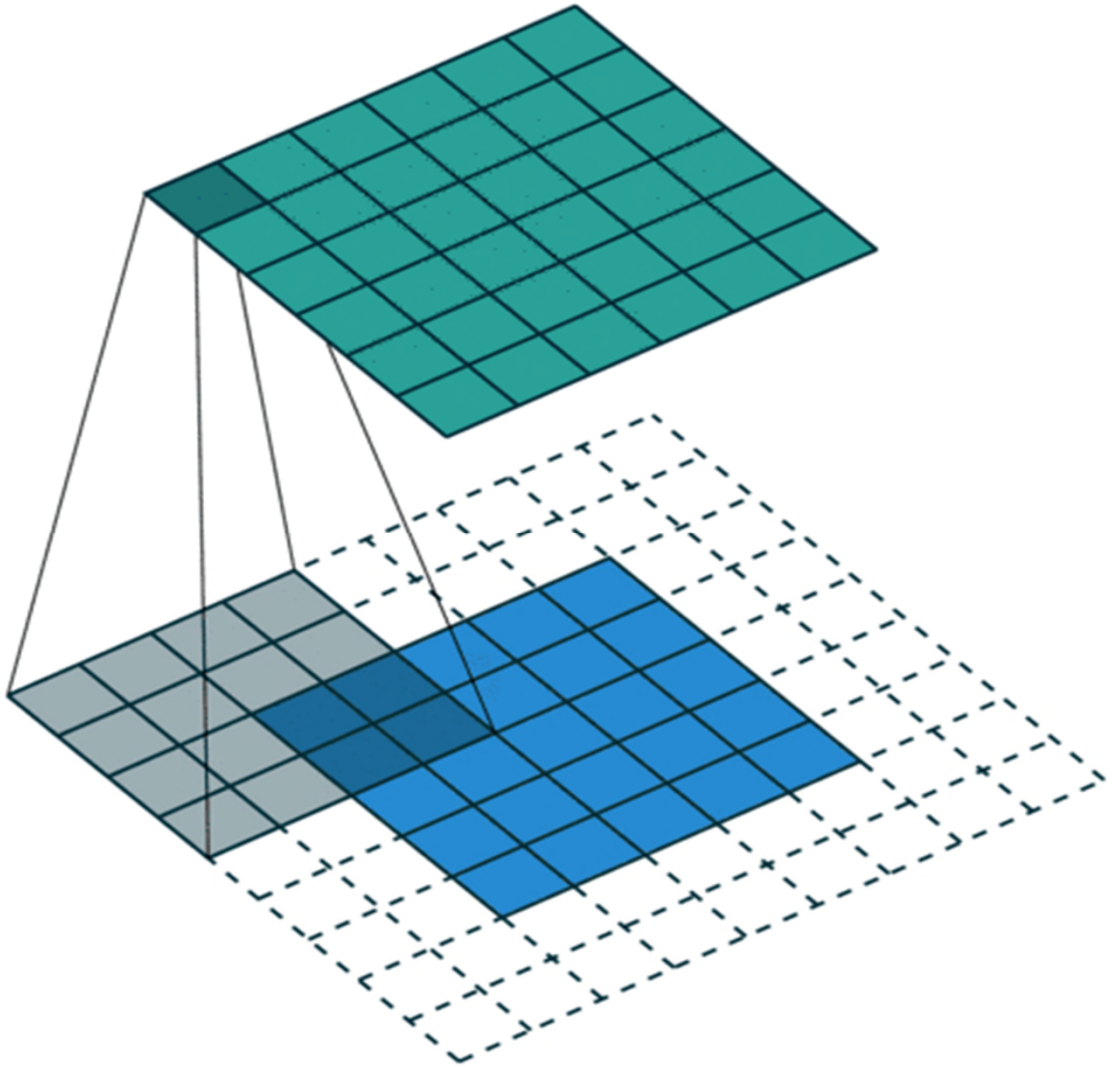
Padding



The input volume is $32 \times 32 \times 3$. If we pad two borders of zeros around the volume, this gives us a $36 \times 36 \times 3$ volume. Then, when we apply our conv layer with three $5 \times 5 \times 3$ filters and a stride of 1, we will get a $32 \times 32 \times 3$ output volume.

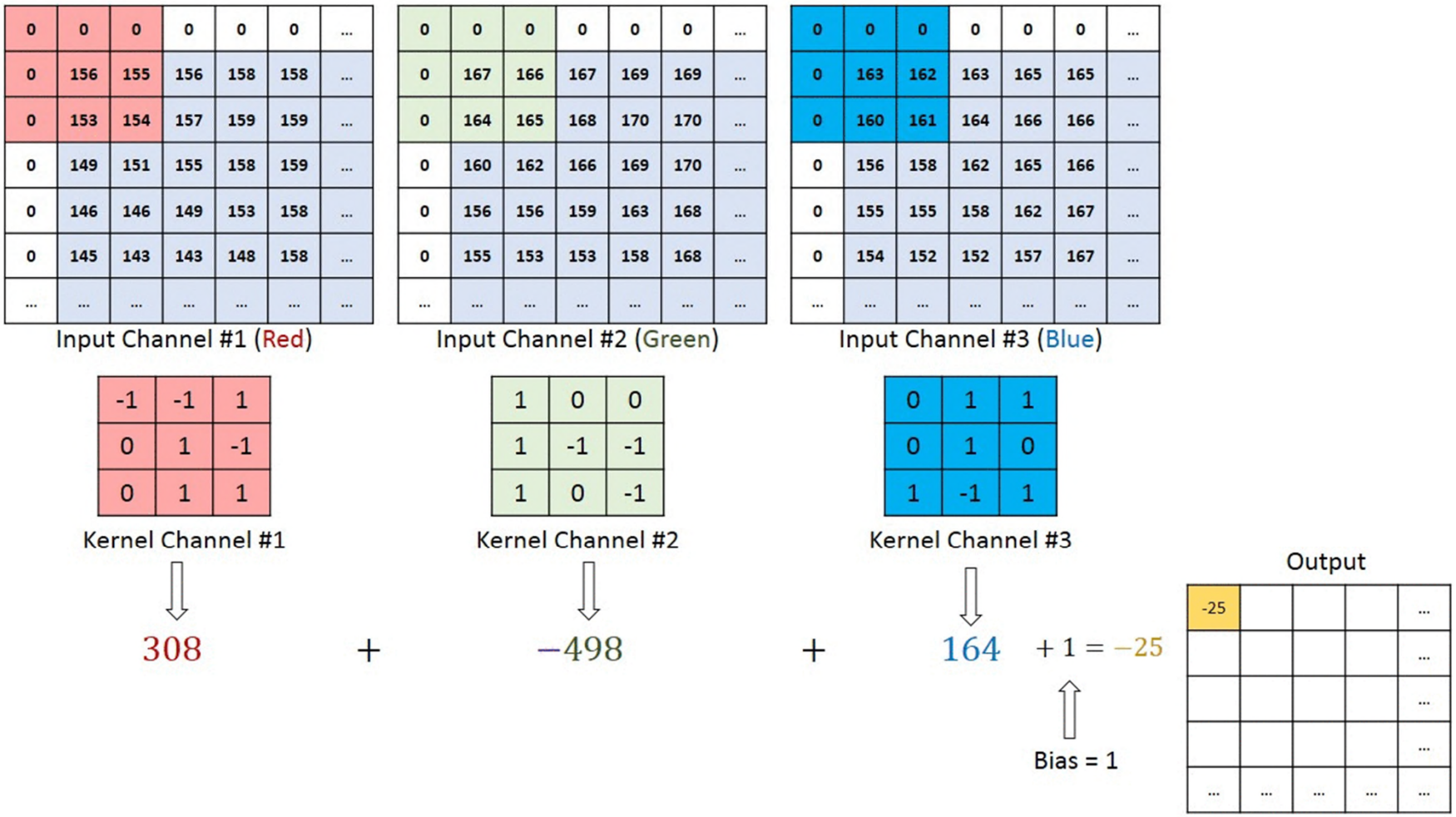
Source: “A Beginner's Guide To Understanding Convolutional Neural Networks Part 2”, Adit Deshpande, github.io

Padding



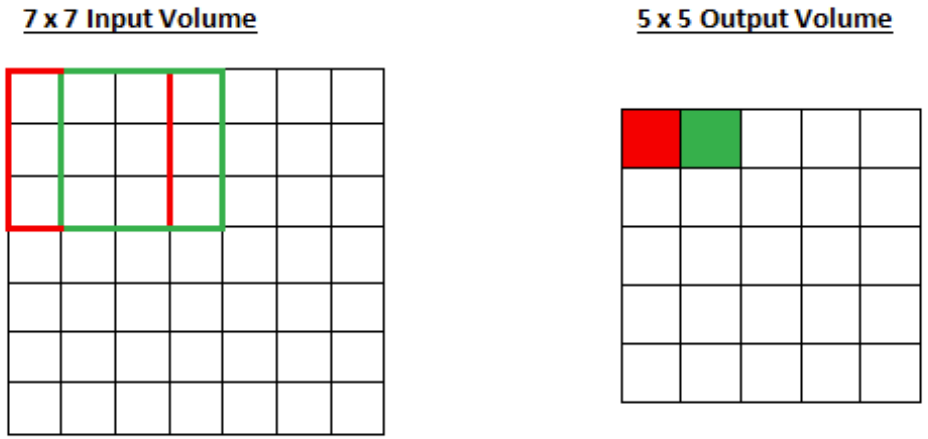
Source: cloud.tencent.com/developer/article/1031131

Padding + Convolution

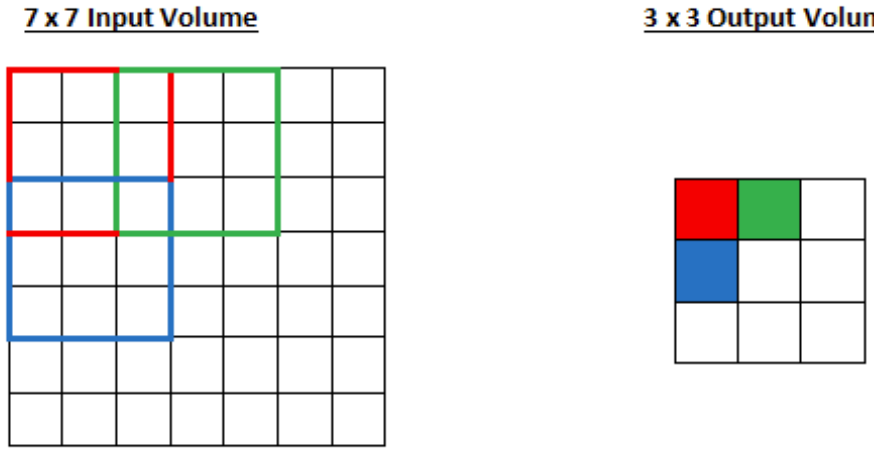


Source: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks

Stride



3 × 3 filter with stride 1

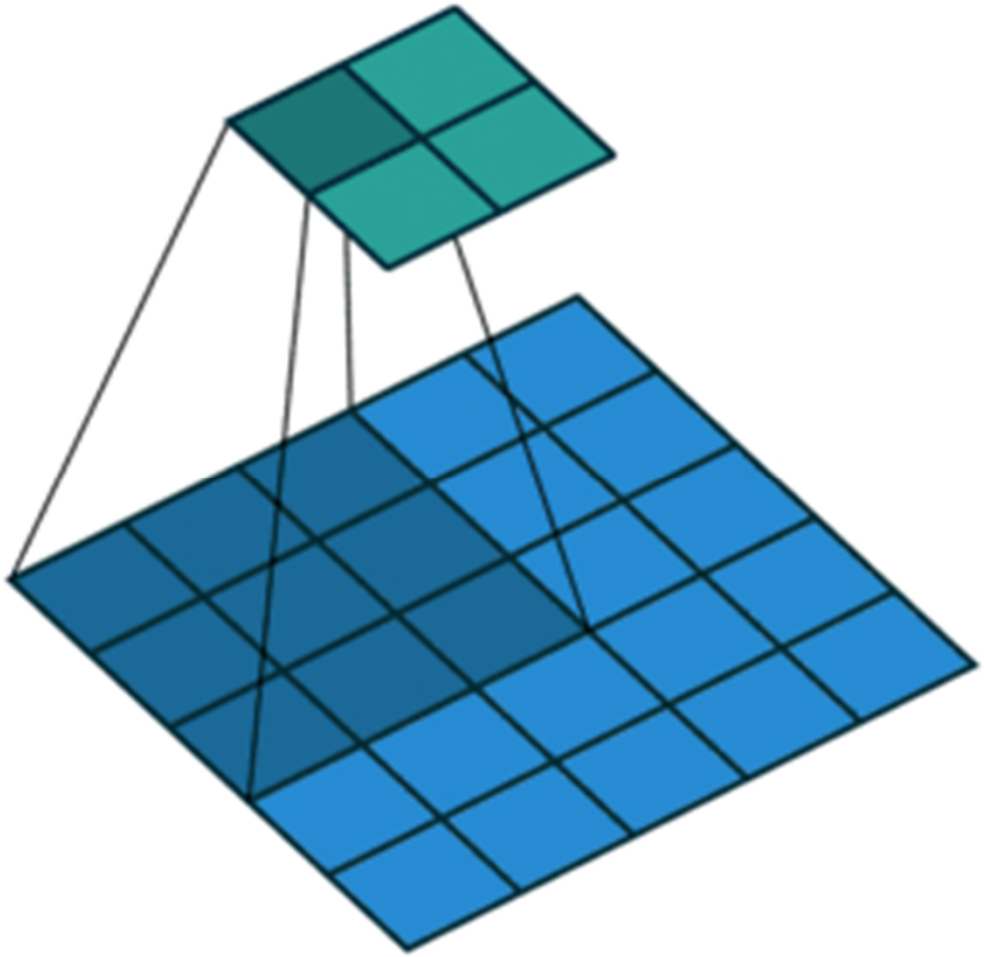


3 × 3 filter with stride 2

Source: “A Beginner's Guide To Understanding Convolutional Neural Networks Part 2”, Adit Deshpande, github.io

Stride

- Stride = 2:



Source: cloud.tencent.com/developer/article/1031131

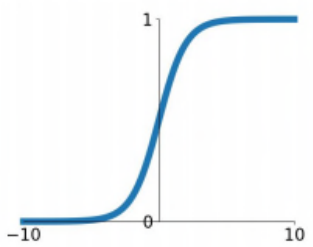
Activation Layer

- Perform element-by-element nonlinear activation function mapping.
- RELU is one of most popular activation functions.
- Often combined with convolution layer.

Activation Layer

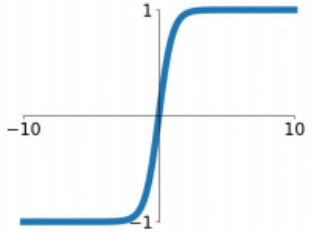
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



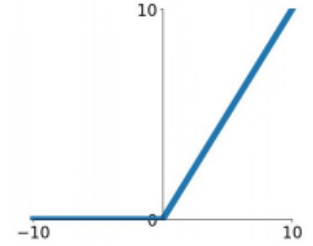
tanh

$$\tanh(x)$$



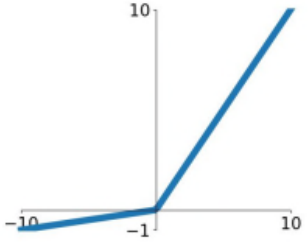
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

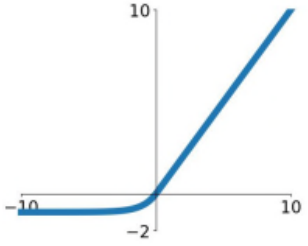


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

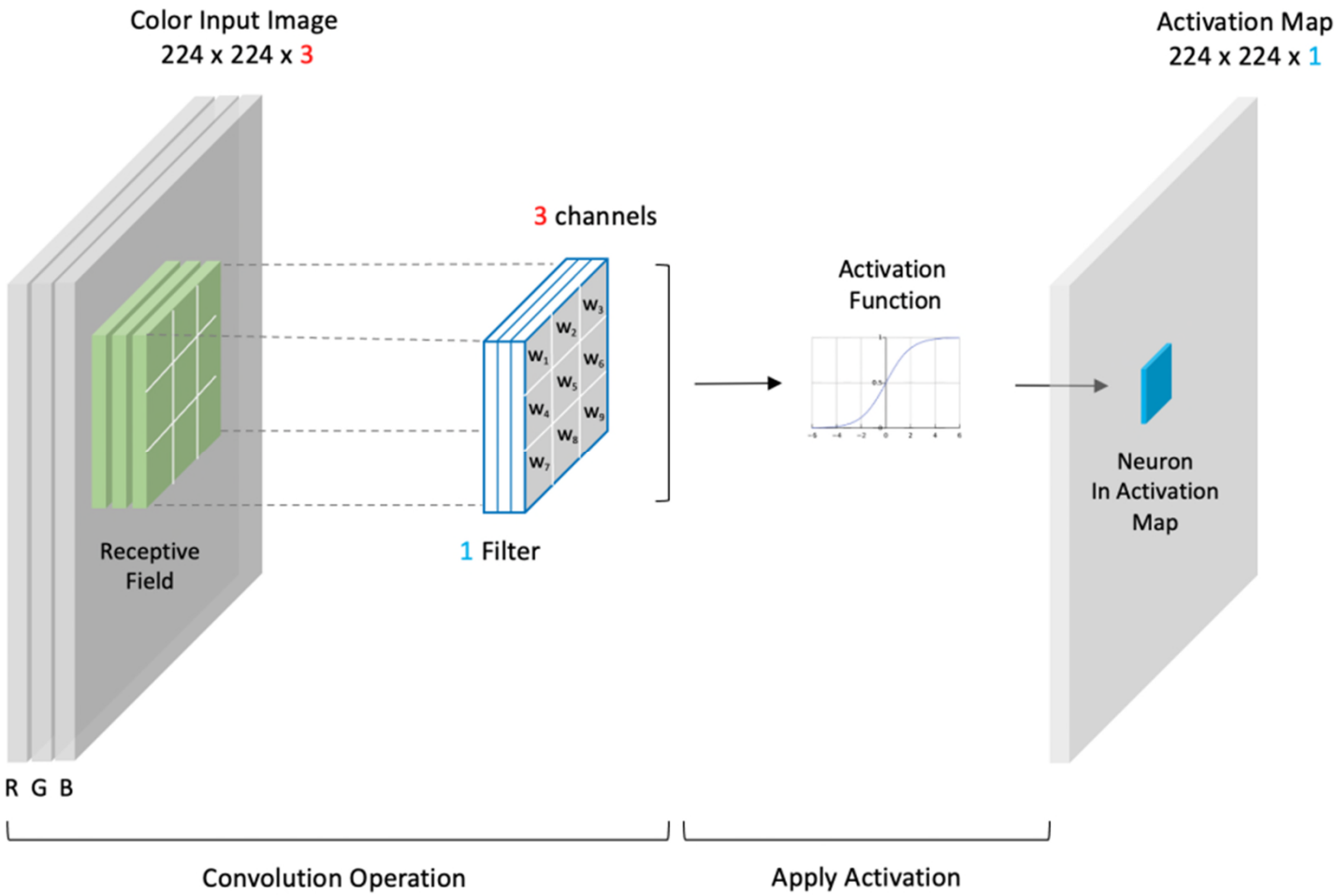
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Source: Stanford Lecture Notes, CS231n

Conv + RELU

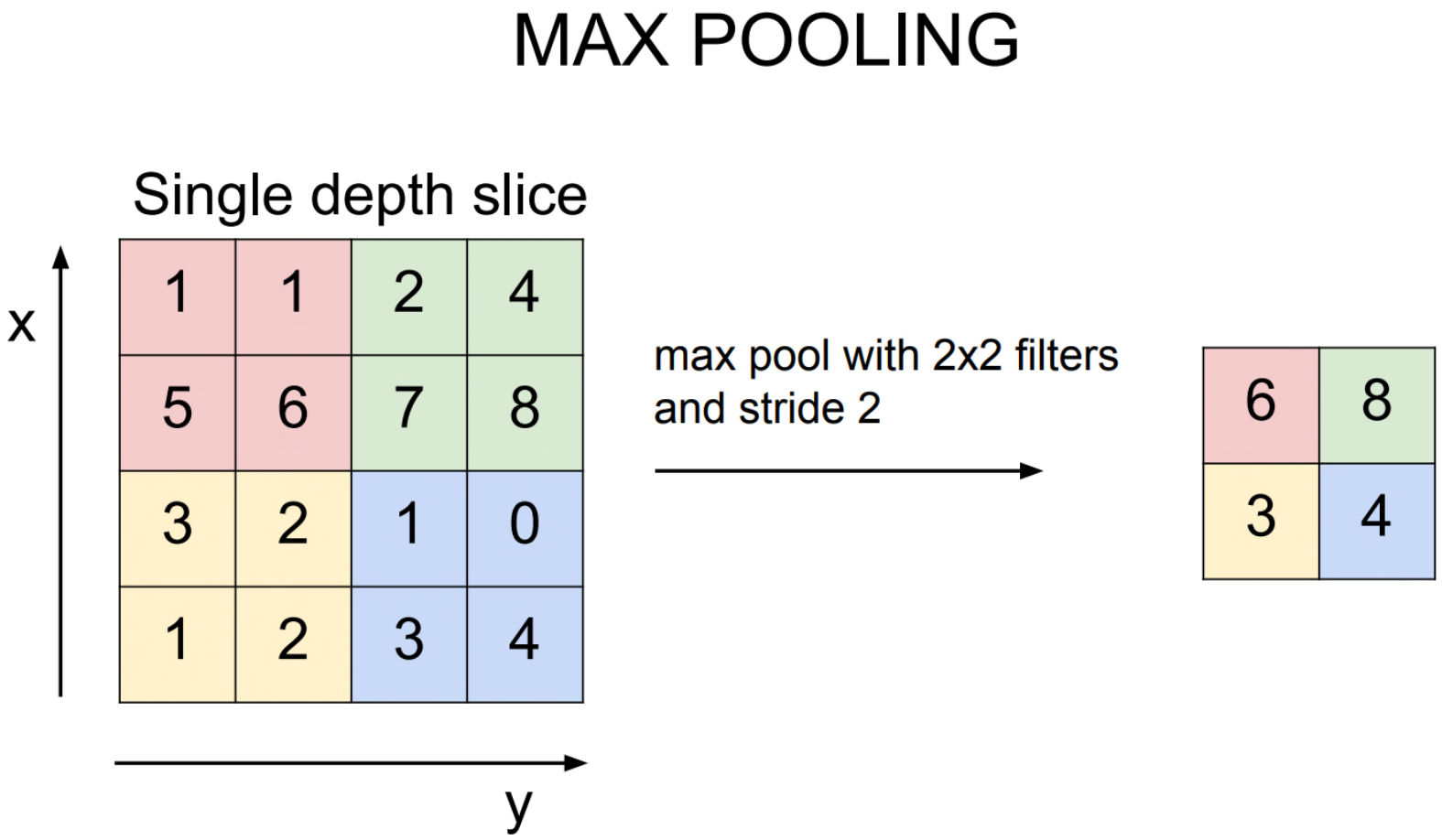


Source: https://learnopencv.com/understanding-convolutional-neural-networks-cnn/

Pooling Layer

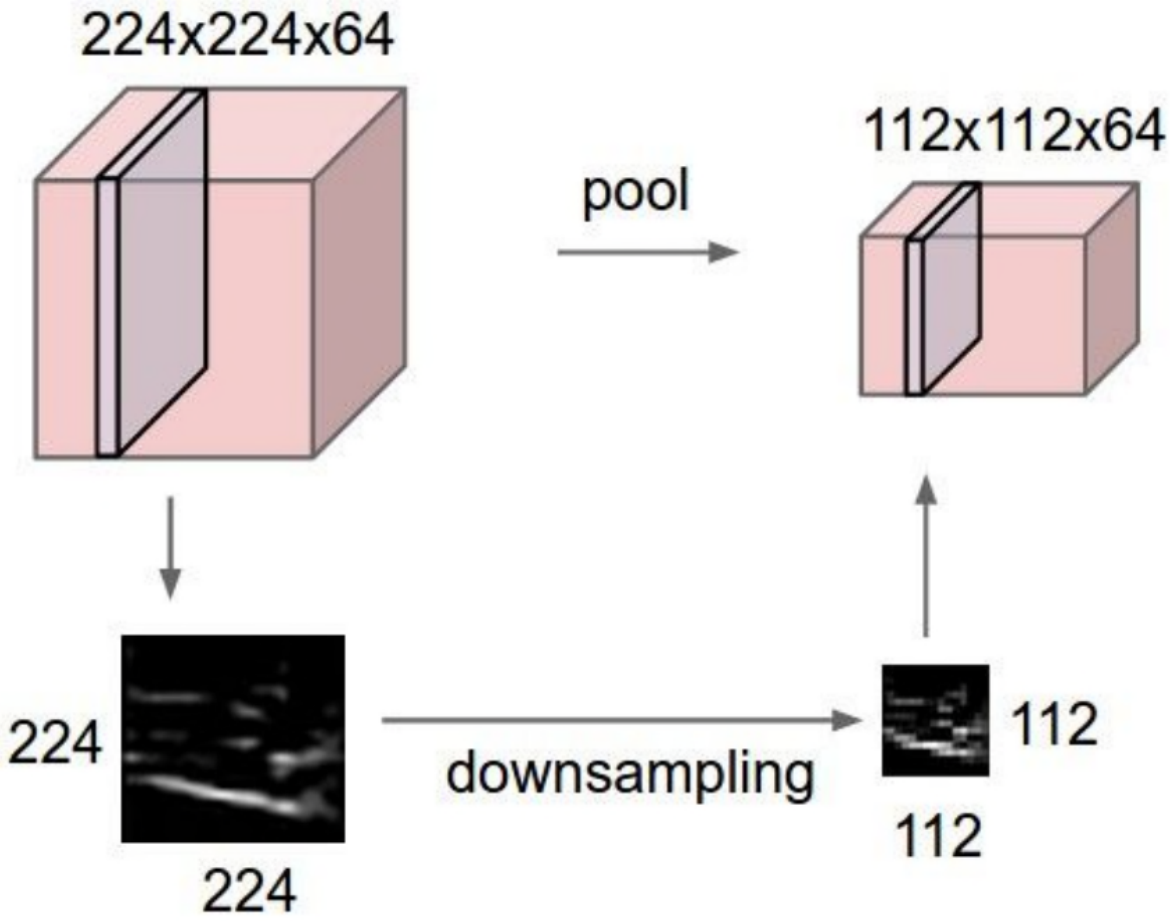
- Reduce activation map dimension, hence reducing computation and storage requirement.
- Common pooling operation: max pooling, average pooling.
- Each channel of activation/feature maps operate independently.

Pooling Layer



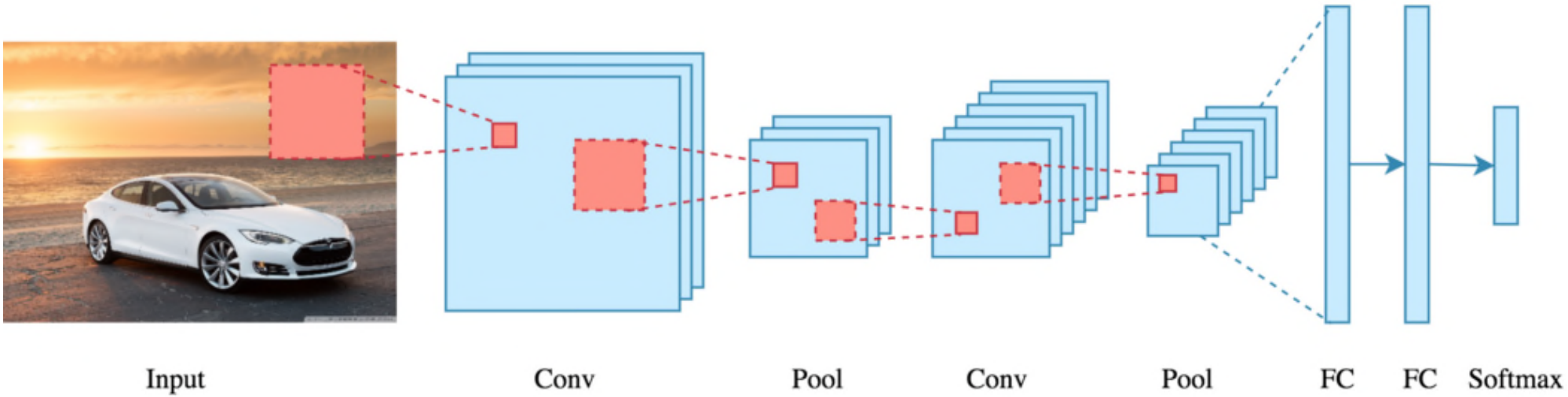
Source: Stanford Lecture Notes, CS231n

Pooling Layer



Source: Stanford Lecture Notes, CS231n

FC Layer



Source: https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

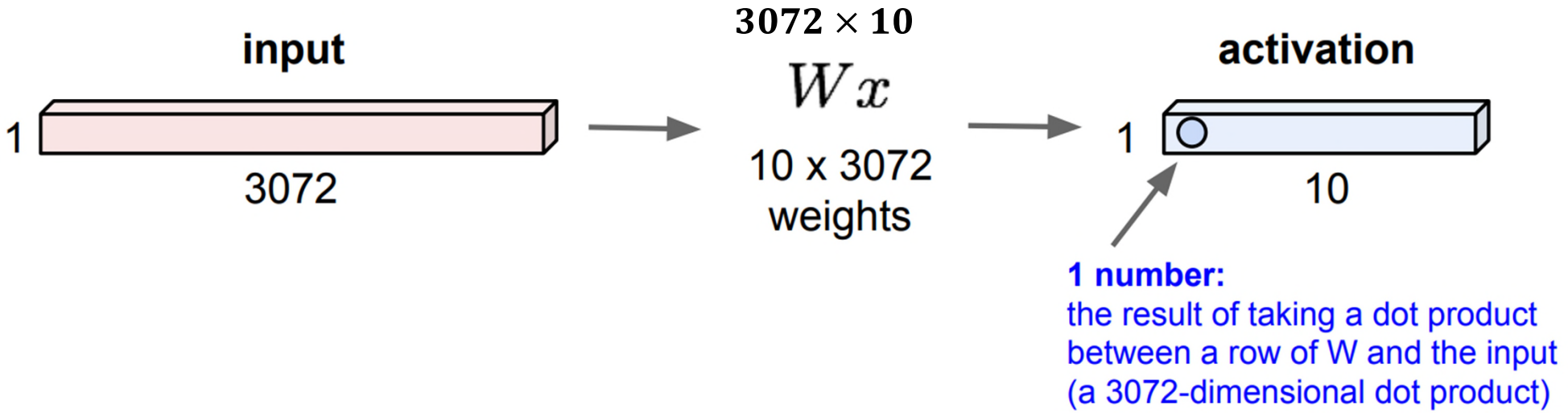
Fully Connected (FC) Layer

- All nodes in one layer are connected to all nodes in the next layer.
- FC layer feature vector can be used as feature (also known as embedding) to represent input image.
- FC layer behaves like a linear layer for classification.

FC Layer

32x32x3 image -> stretch to 3072 x 1

Each neuron looks at the full input volume

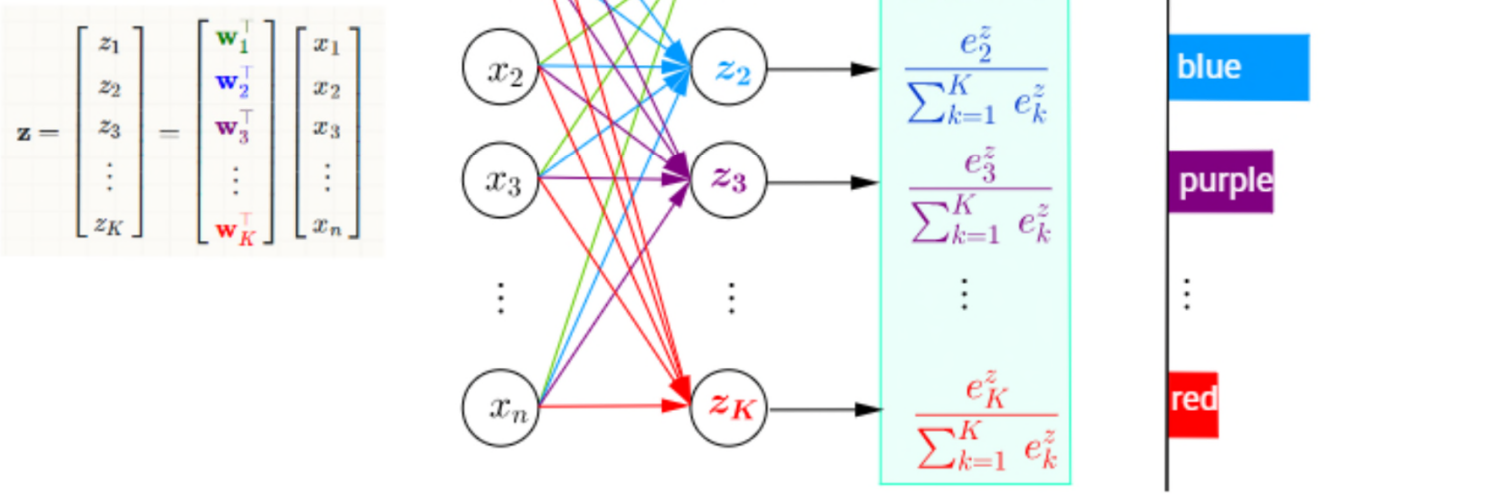


Source: Stanford Lecture Notes, CS231n

Softmax Layer

- Map the output scores (logits) from last FC layer into probabilities for classification problem.
- Softmax loss is computed based on the probabilities.

$$p_j = \frac{e^{z_j}}{\sum_k e^{z_k}}$$



Source: https://stats.stackexchange.com/questions/265905/derivative-of-softmax-with-respect-to-weights

Sample Pytorch CNN Implementation

```
class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        # 1 input image channel, 6 output channels, 3x3 square convolution
        # kernel
        self.conv1 = nn.Conv2d(1, 6, 3)
        self.conv2 = nn.Conv2d(6, 16, 3)
        # an affine operation: y = Wx + b
        self.fc1 = nn.Linear(16 * 6 * 6, 120)  # 6*6 from image dimension
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        # Max pooling over a (2, 2) window
        x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
        # If the size is a square you can only specify a single number
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = x.view(-1, self.num_flat_features(x))
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

Conv Layer

Fully-connected Layer

Pooling Layer

Activation Layer

Layer Definition

Feed Forward Logic

Exercise: CNN

(a) In a simple Convolutional Neural Network (CNN), an input image **A** passes through a convolutional layer, followed by an activation layer and a max pooling layer. The output from the max pooling layer is then used for further processing. The grayscale image **A** is given by:

$$\mathbf{A} = \begin{bmatrix} 4 & 0 & 1 \\ 4 & 0 & 2 \\ 0 & 2 & 2 \end{bmatrix}.$$

The convolutional layer has the following settings: the current filter is given by **F** below, the amount of zero padding at each side of the image is 1, and the stride both horizontally and vertically is 2.

$$\mathbf{F} = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

The activation function used in the activation layer is given by a sigmoid function:

$$\sigma(x) = \frac{1}{1+e^{-x}} \quad .$$

The max pooling layer uses 2×2 max pooling with a stride of 2.

Exercise: CNN

(i) Find the output after the convolution layer.

(ii) Briefly discuss the effect of filter **F** when applied to the input image.

(iii) Find the output after the activation layer.

(iv) Find the output after the max pooling layer.

(v) A student would like to make the following changes to the input image and the convolutional layer:

- Change the grayscale image **A** to an RGB image **B** with a spatial dimension of 100×100.
- Change the channel number of the output feature maps to 6 for the new convolutional layer. Assume the spatial dimension of the filter remains the same.

Find the number of trainable parameters of the new convolutional layer after the changes. Assume no bias is used in the calculation.

(18 Marks)

Solution

Exercise: CNN FC and Softmax Layers

An input feature vector $\mathbf{x}$ passes through a Fully Connected (FC) layer of a CNN with the following settings:

Input: $\mathbf{x} = \begin{bmatrix} 0.3 \\ 0 \\ 0.2 \\ 0.1 \end{bmatrix}$, Weights: $\mathbf{W} = \begin{bmatrix} 0 & 3 & 7 & 8 \\ 1 & 8 & 0 & 0 \\ 0 & 8 & 1 & 0 \end{bmatrix}$, Bias: $\mathbf{b} = \begin{bmatrix} -1 \\ 1 \\ 2 \end{bmatrix}$

Softmax function is then applied to the output of this FC layer.

(a) Find the output after the FC layer.

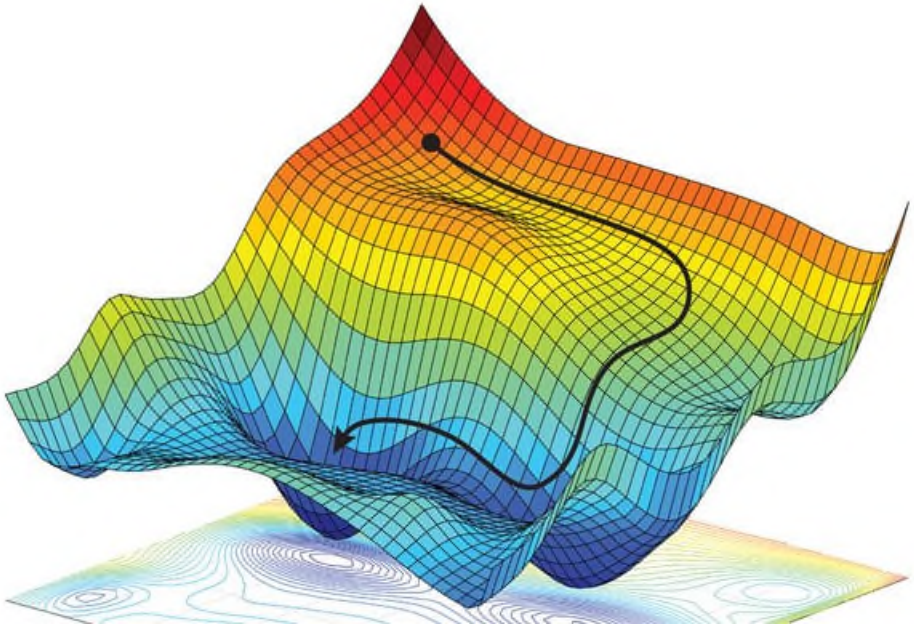
(b) Find the output after the Softmax layer.

Solution

CNN Training & Optimization

CNN Training

- The objective is to minimize/optimize the loss function.
- Common strategy is centered to stochastic gradient descent (SGD) and its variants.
- Use computational graph to compute the gradient descent and parameter update.



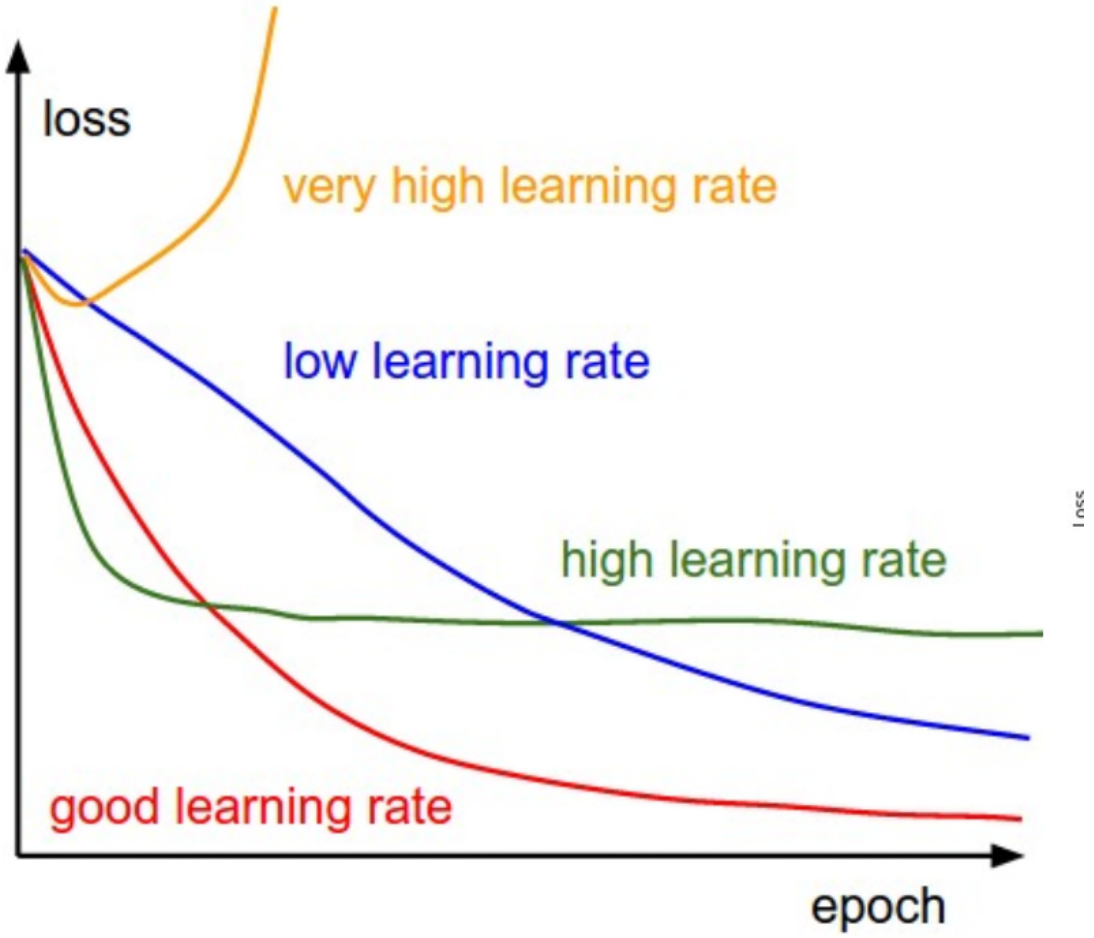
Source: “Coding Deep Learning for Beginners — Linear Regression Part 3”, Kamil Krzyk, towardsdatascience.com

CNN Training

$$L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(x_i, y_i, W)$$

$$\nabla_W L(W) = \frac{1}{N}\sum_{i=1}^{N} \nabla_W L_i(x_i, y_i, W)$$

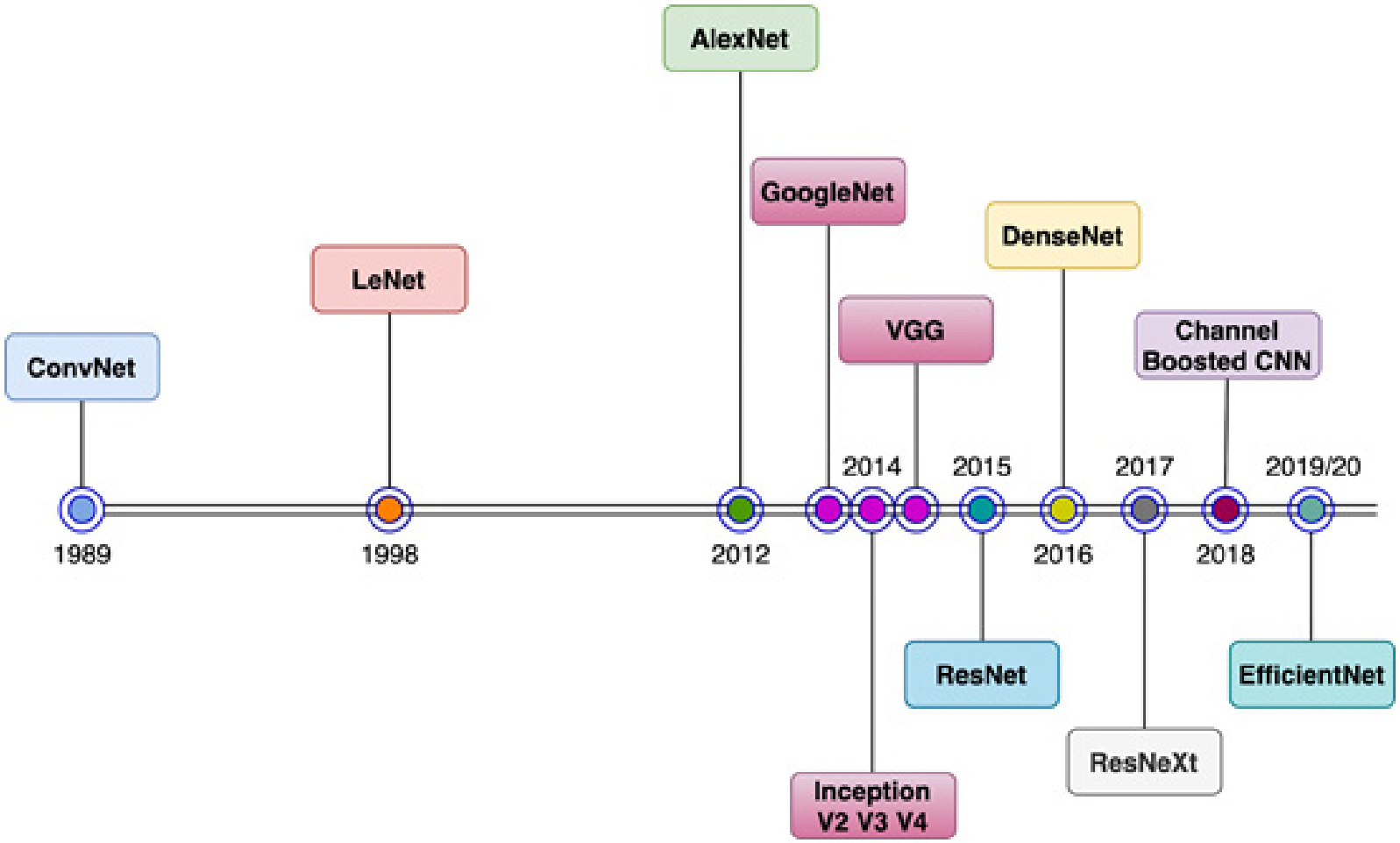
$$w_{t+1} = w_t - \alpha \nabla L(w_t)$$



Source: Stanford Lecture Notes, CS231n

Well-Known CNN Architectures

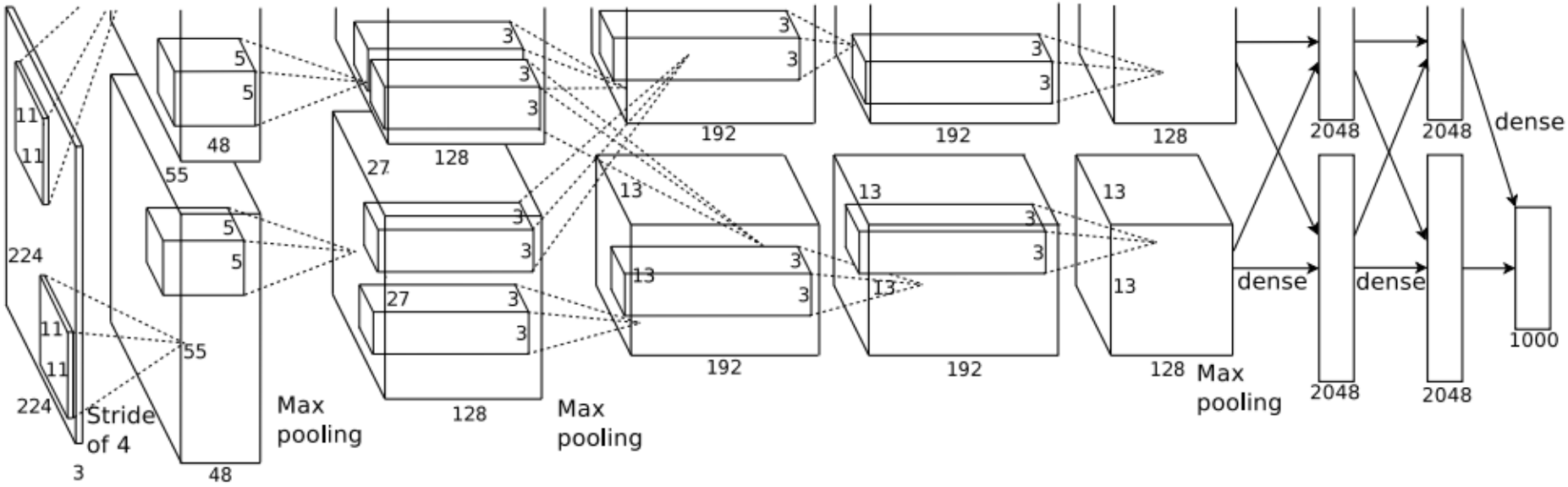
CNN Timeline / Milestone



Source: packtpub.com/../evolution-of-cnn-architectures

AlexNet

- First neural network applied on large scale image data.
- Classical structure for image classification consisting of convolution layers and fully connected layers.
- ILSVRC champion in 2012.

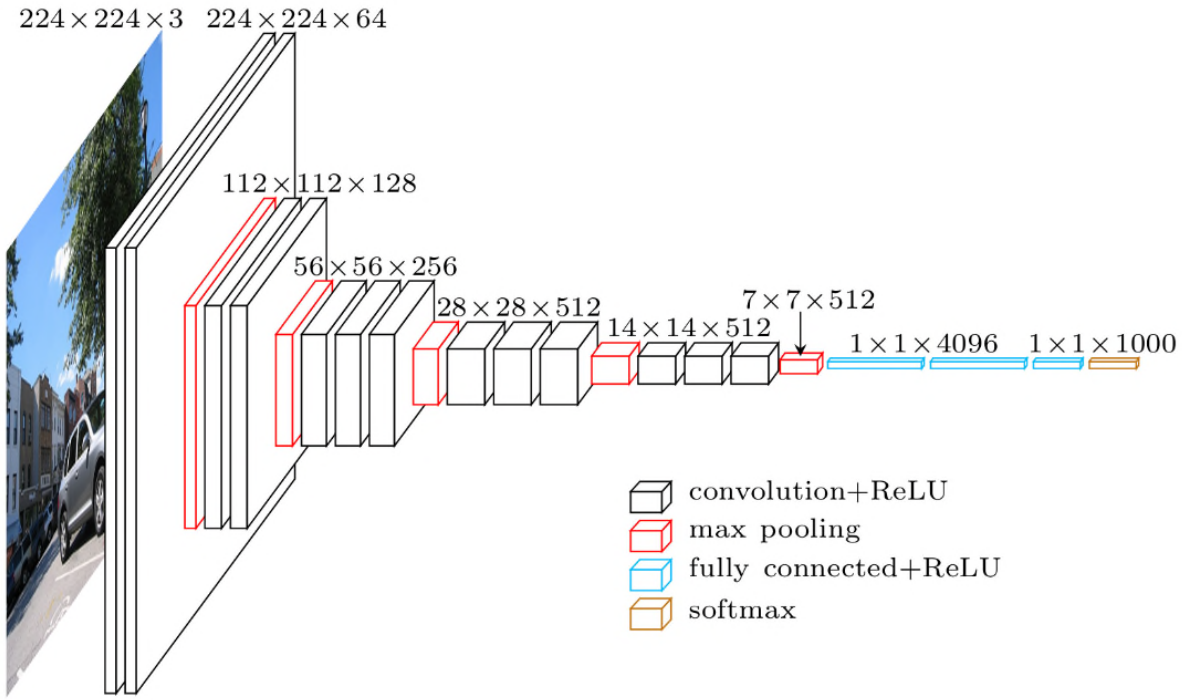


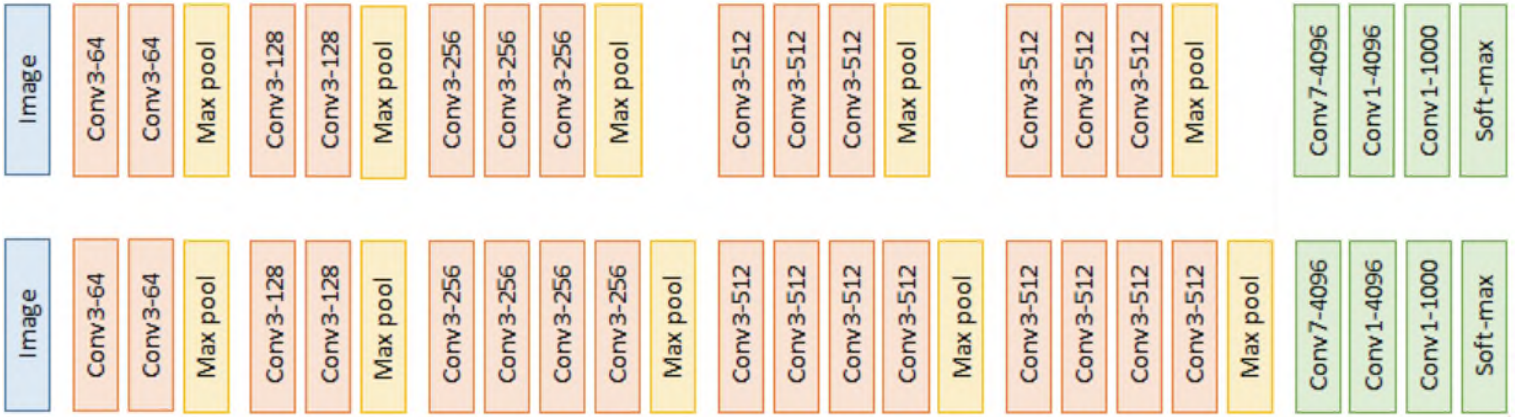
Source: “Imagenet classification with deep convolutional neural networks”, Krizhevsky Alex et al., 2012

VGG Network

- Runner up of ILSVRC 2014.
- Elegant network architecture.
- Use deep network with small 3×3 kernels.
- Require large number of parameters.
- Two common variants: VGG-16 and VGG-19

VGG Network



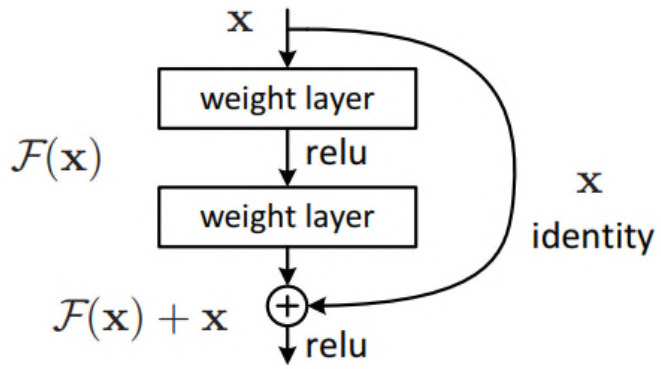


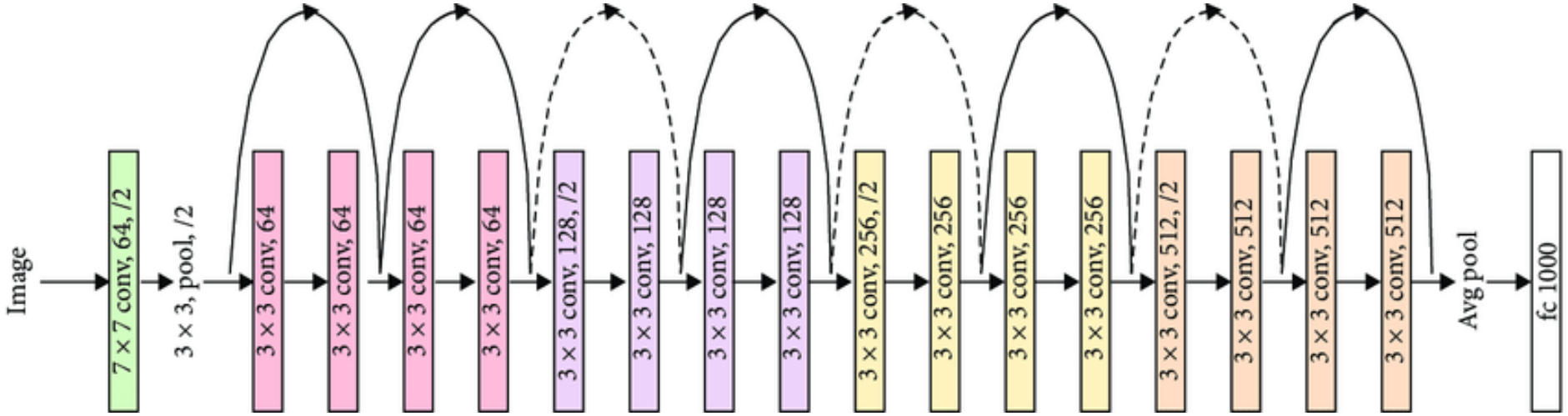
Source: “Very deep convolutional networks for large-scale image recognition”, K. Simonyan et al., 2014

ResNet

- Winner of ILSVRC 2015
- Very deep structure
- Use residual block and highway/skip connection for better gradient backpropagation.

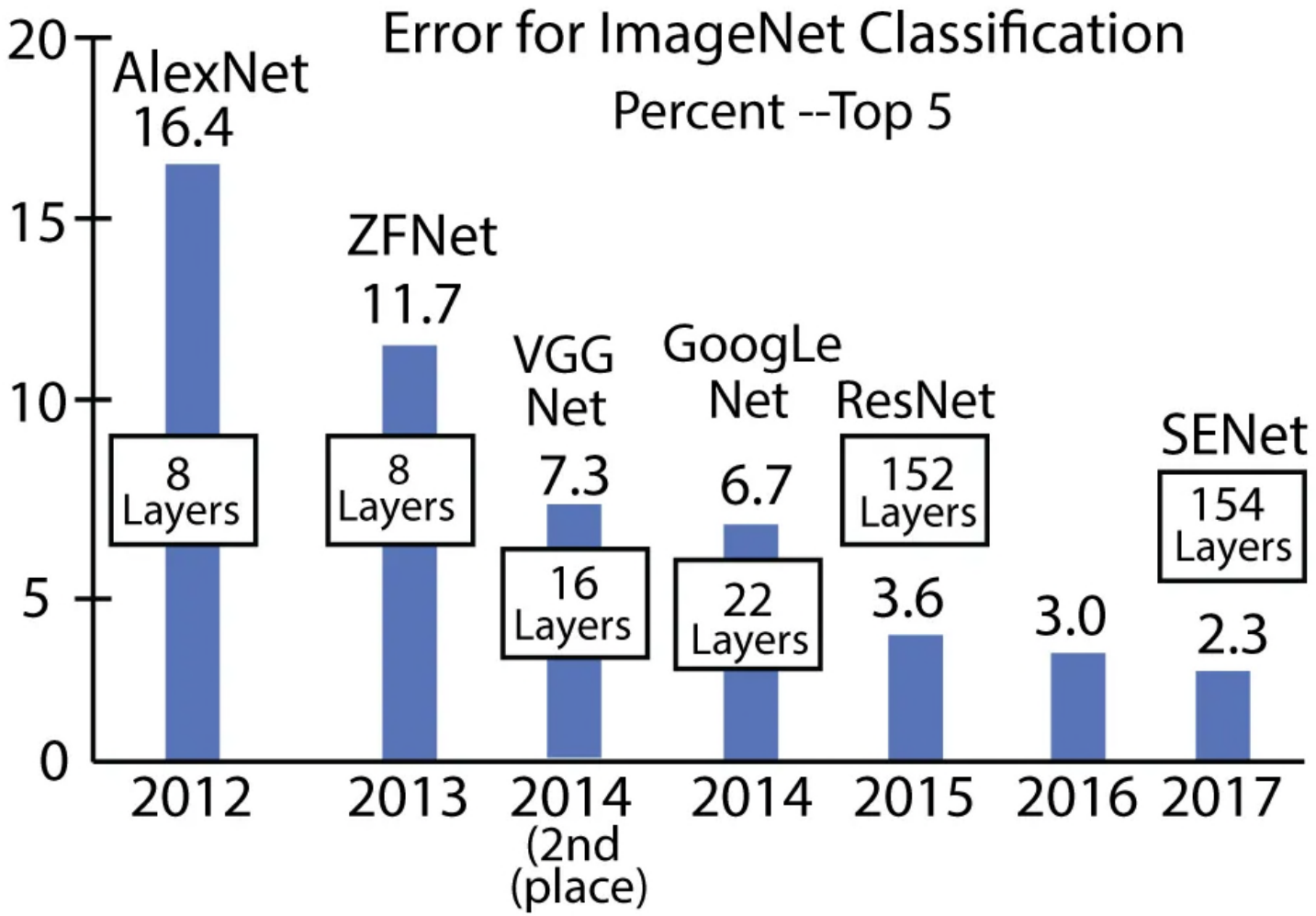
ResNet





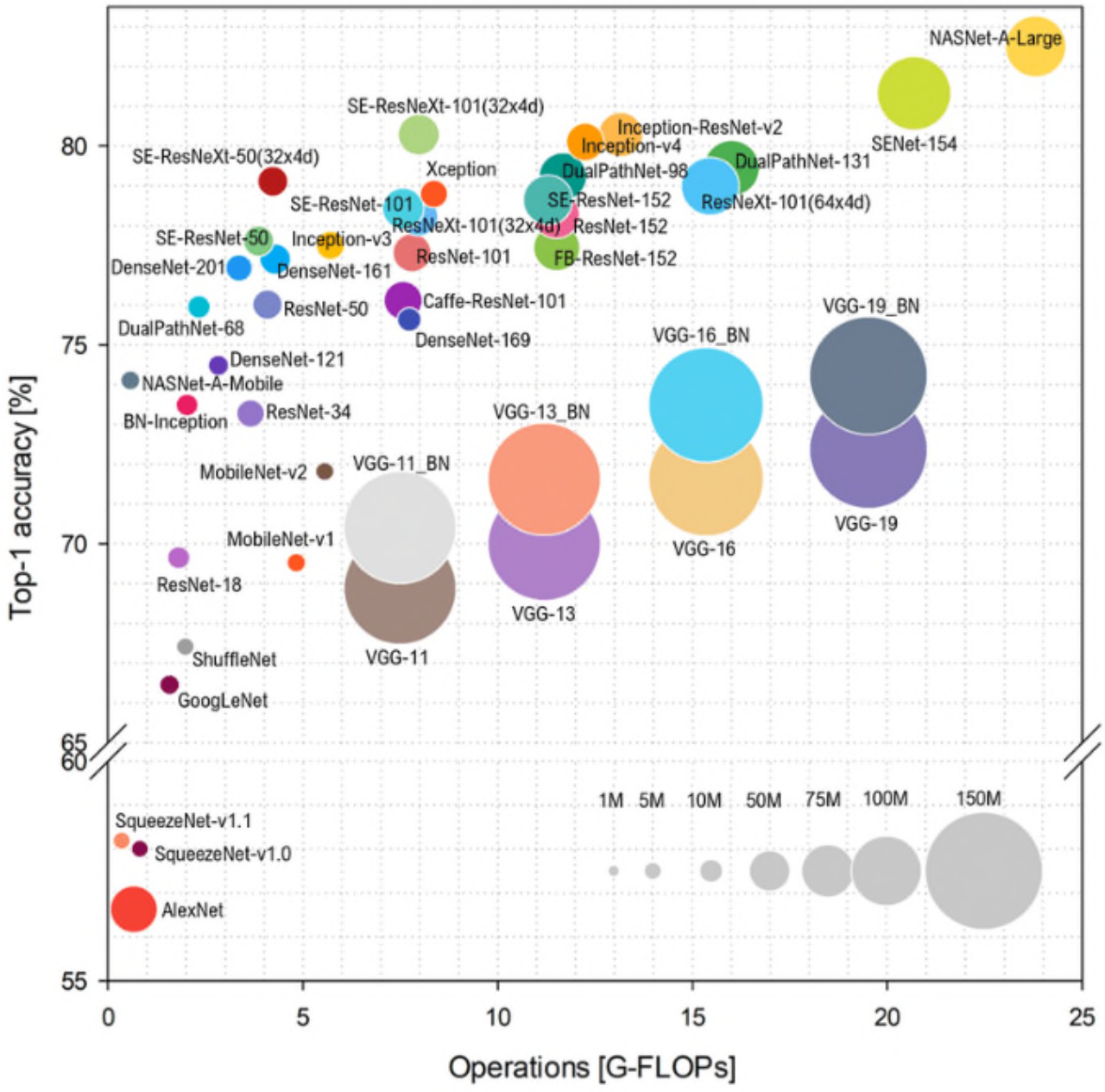
Source: “Deep residual learning for image recognition”, Kaiming He et al., 2016

Performance Comparison



Source: “Are Deep Learning Networks Getting Too Deep?”, principlesofdeeplearning.com

Performance Comparison on ImageNet



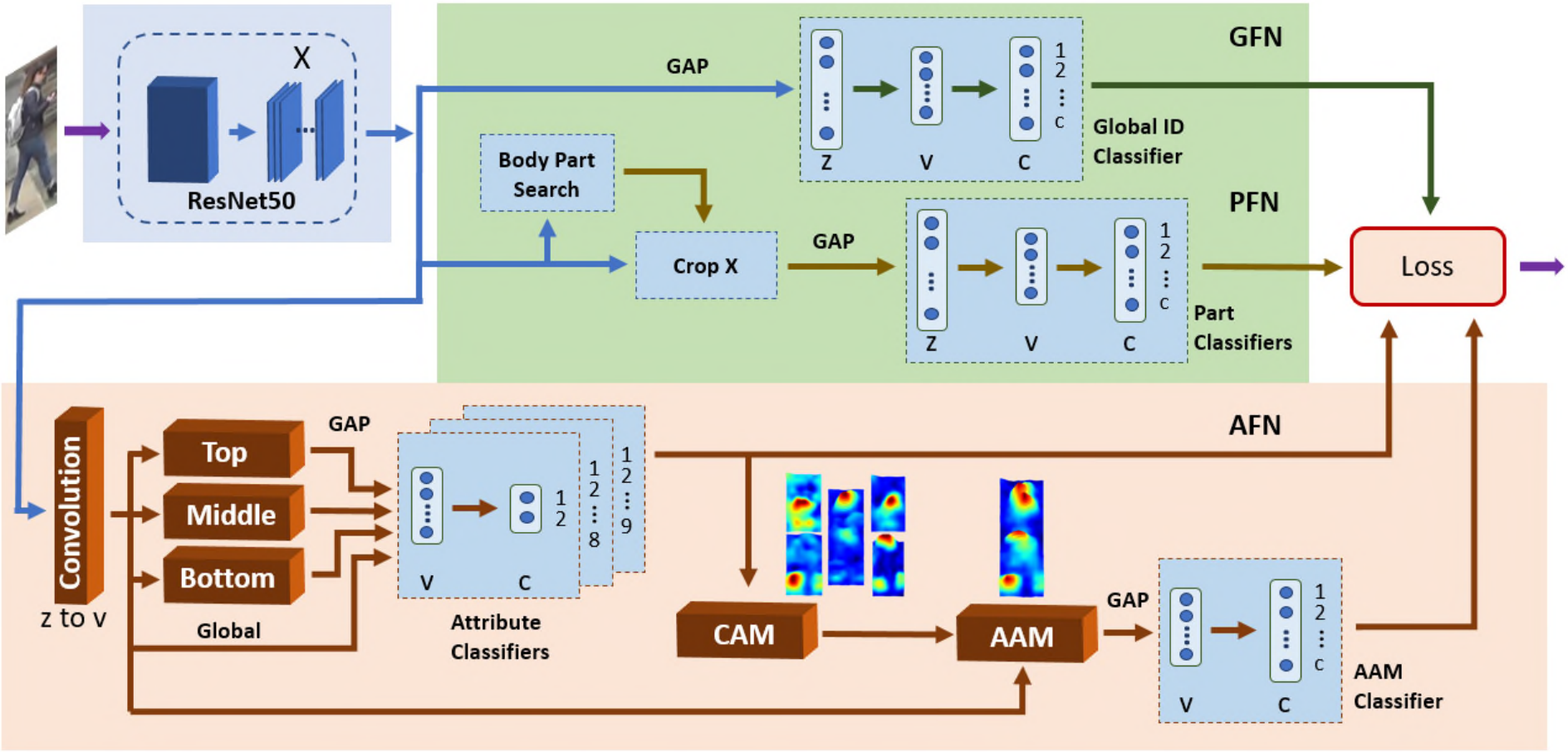
Key Performance Metrics

- Accuracy
- Memory footprint (parameters/weights + activation maps)
- Speed/computational complexity (FLOPS)
- Often, what is the important metric will depend on the applications/problems.

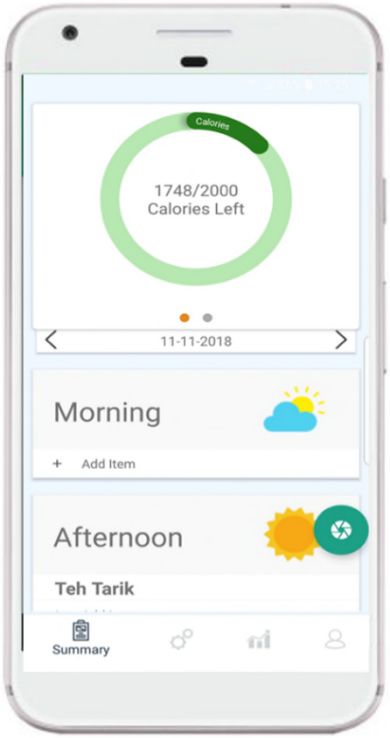
Applications

Person Re-Identification

Person Re-Identification

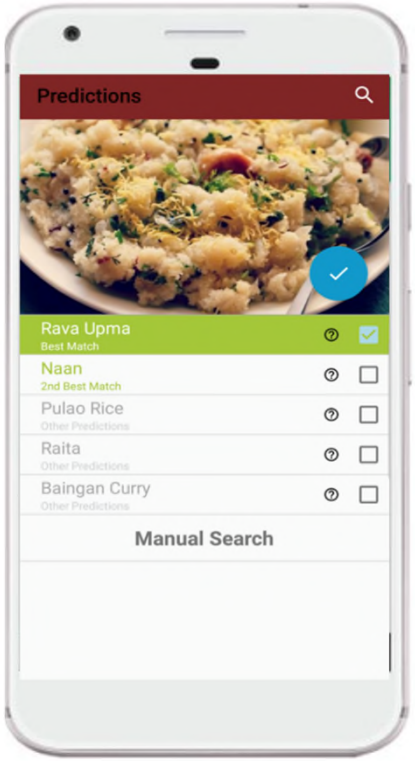


Visual Food Recognition



Home page

Taking picture



Food recognition

Visual Food Recognition

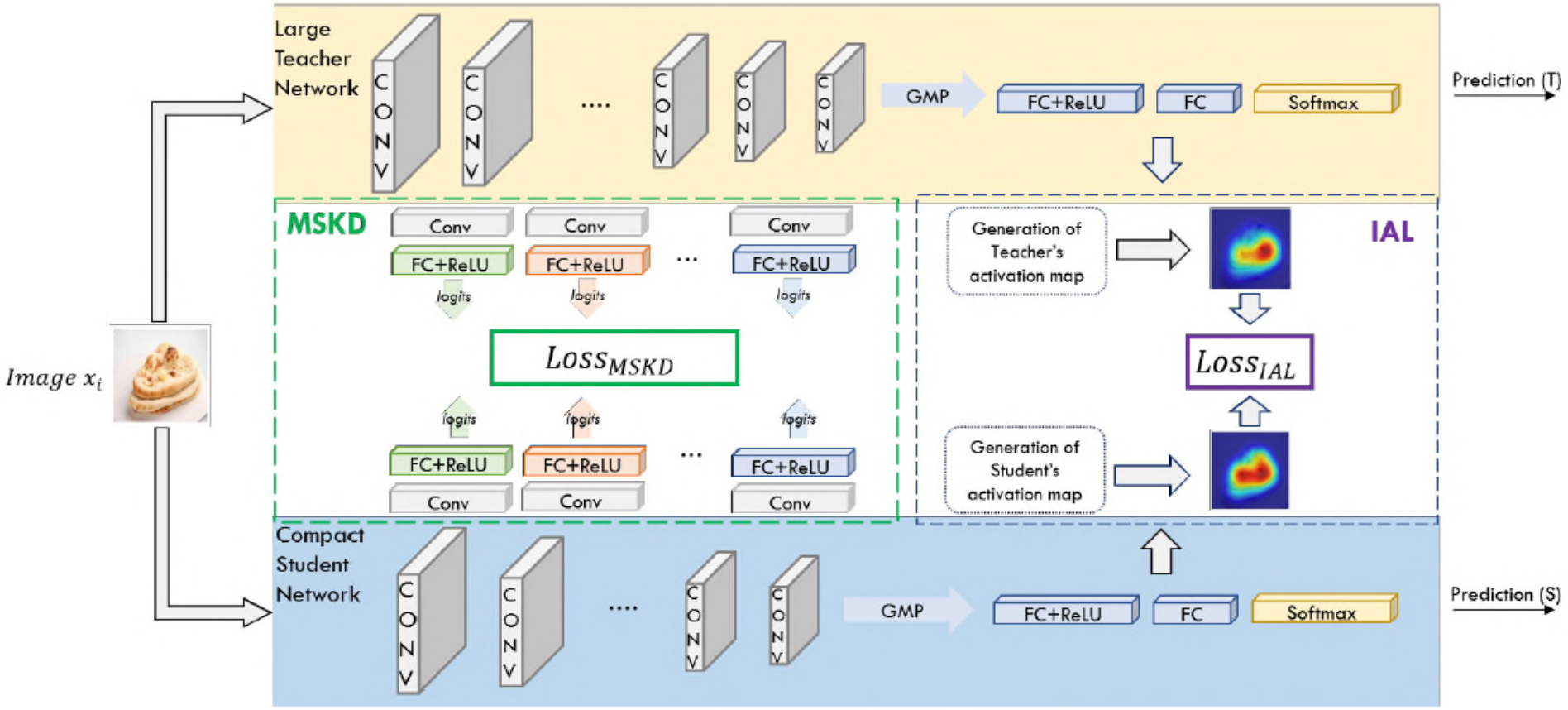


Figure 2: Overview of the proposed JLDNet. A full teacher network and a compact student network are jointly trained using the proposed Multi-Stage Knowledge Distillation (MSKD) and Instance Activation Learning (IAL) methods to achieve high recognition accuracy at low memory footprint.

Section I Summary

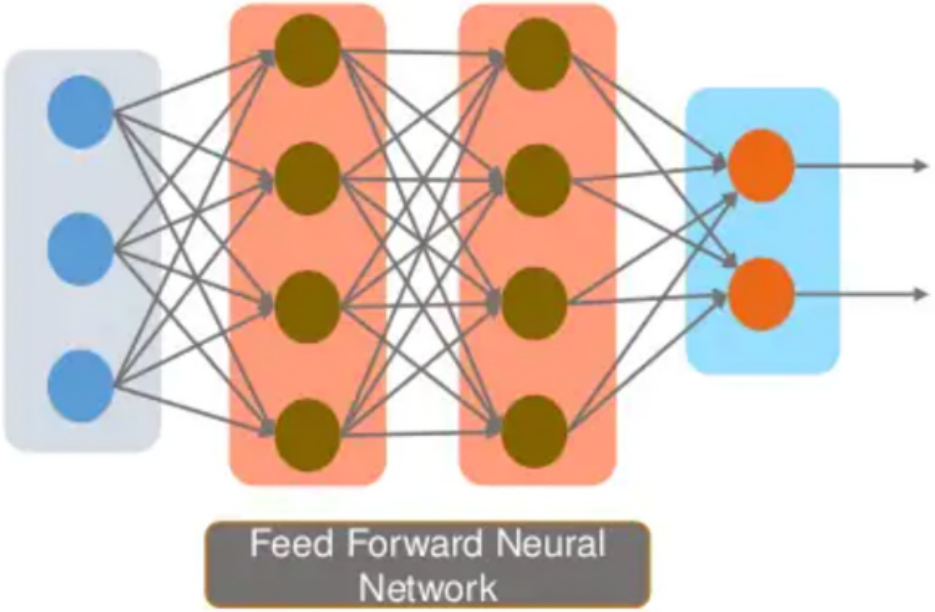
- The section covers the following topics:
  - Introduction
  - Linear Classifier
  - Convolutional Neural Networks (CNNs)
  - CNN Training & Optimization
  - Well-Known CNN Architectures
  - Applications

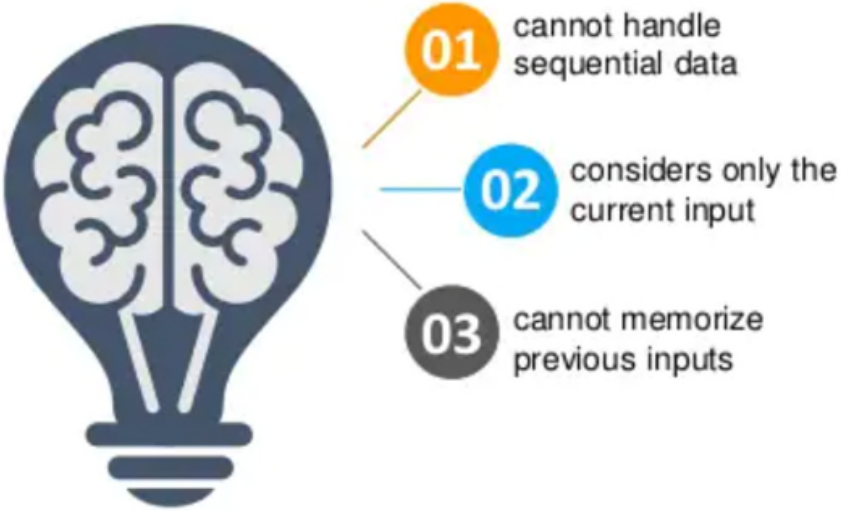
Section II
Recurrent Neural Network (RNN) & Long Short-Term Memory (LSTM)

Section II Overview

- The section covers the following topics:
  - Introduction
  - Recurrent Neural Network (RNN)
  - RNN Training & Optimization
  - Long Short-Term Memory (LSTM)
  - Applications

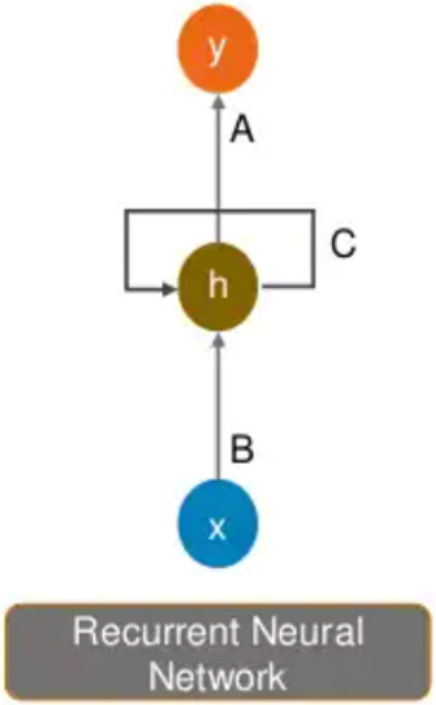
Limitations of Feedforward Network

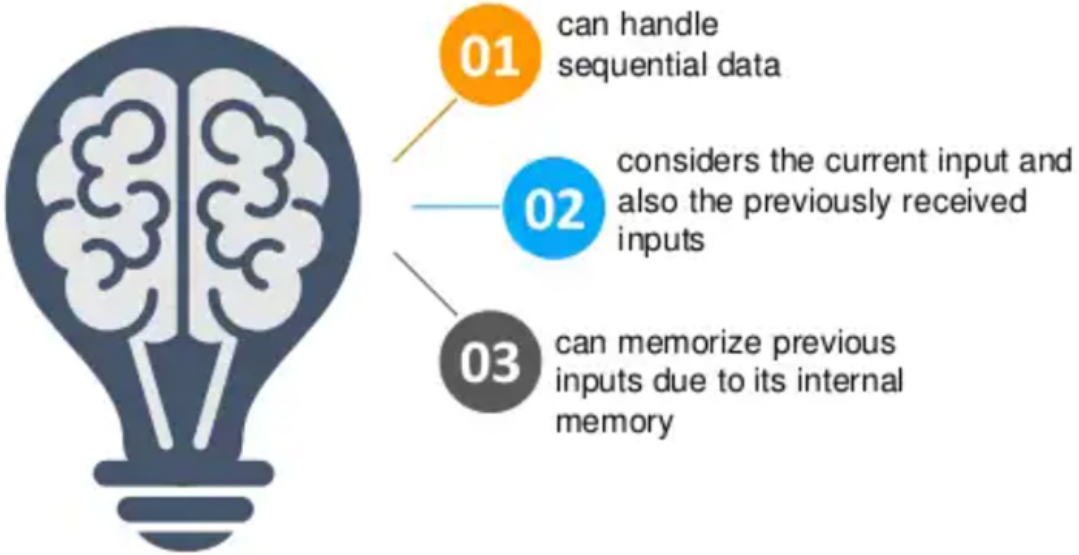




Source: Recurrent Neural Network (RNN) | RNN LSTM Tutorial | Deep Learning Course | Simplilearn

Why Recurrent Neural Network?





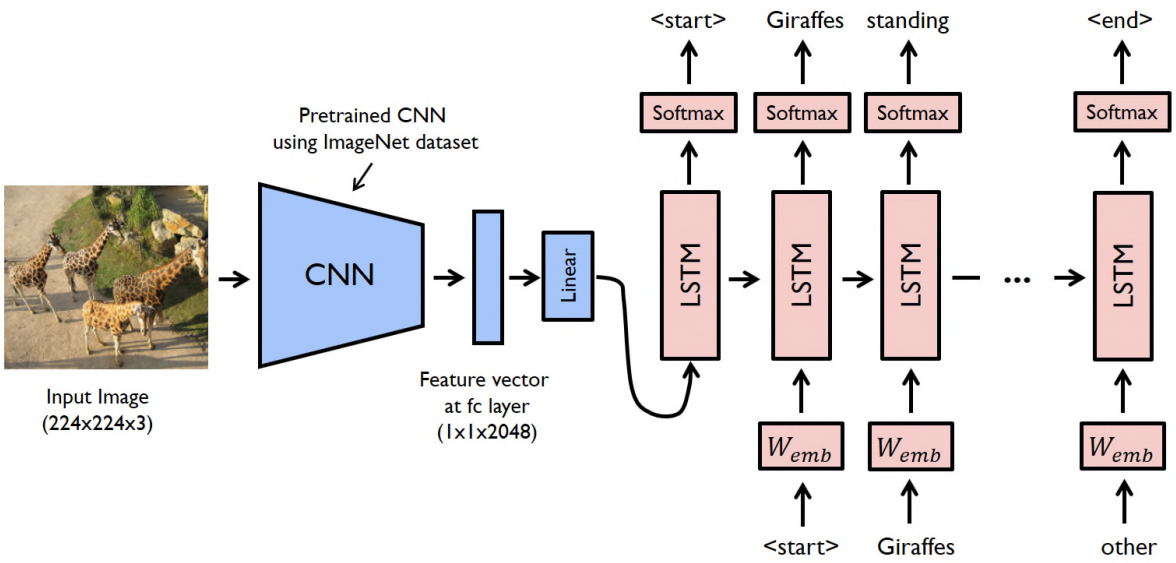
Sample Applications

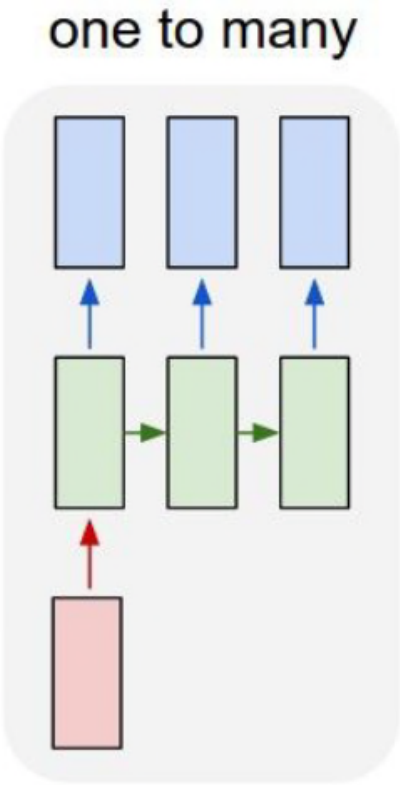
- Time series prediction / forecasting
  - Prediction of stock prices, product sales, etc.
- Speech recognition
- Natural Language Processing (NLP)
  - Language translation
  - Text sentiment classification
- Image captioning

Sequence Modelling

Sequence Modelling (One-to-Many Mapping)

- One input, many outputs (sequence of outputs)
- E.g., image captioning
  - Image → sequence of words



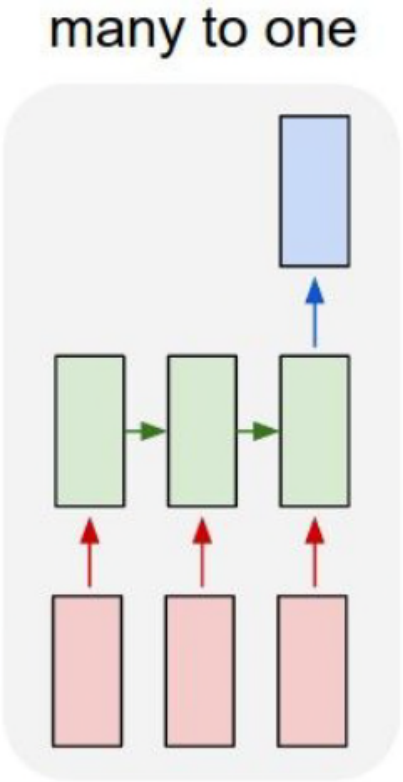


Source: UMich EECS 498-007 / 598-005: Deep Learning for Computer Vision

Source: https://www.analyticsvidhya.com/blog/2018/04/solving-an-image-captioning-task-using-deep-learning/

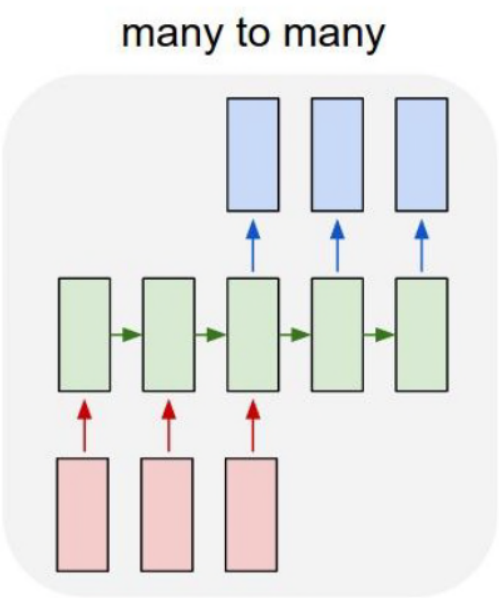
Sequence Modelling (Many-to-One Mapping)

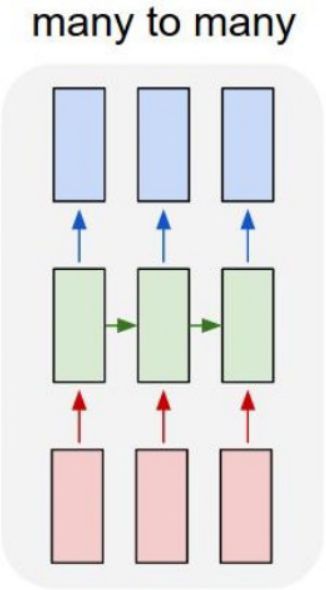
- Many inputs (sequence of inputs), one output
- E.g., Video classification
  - Sequence of images → Label
- Sentiment classification
  - Sequence of words → Label



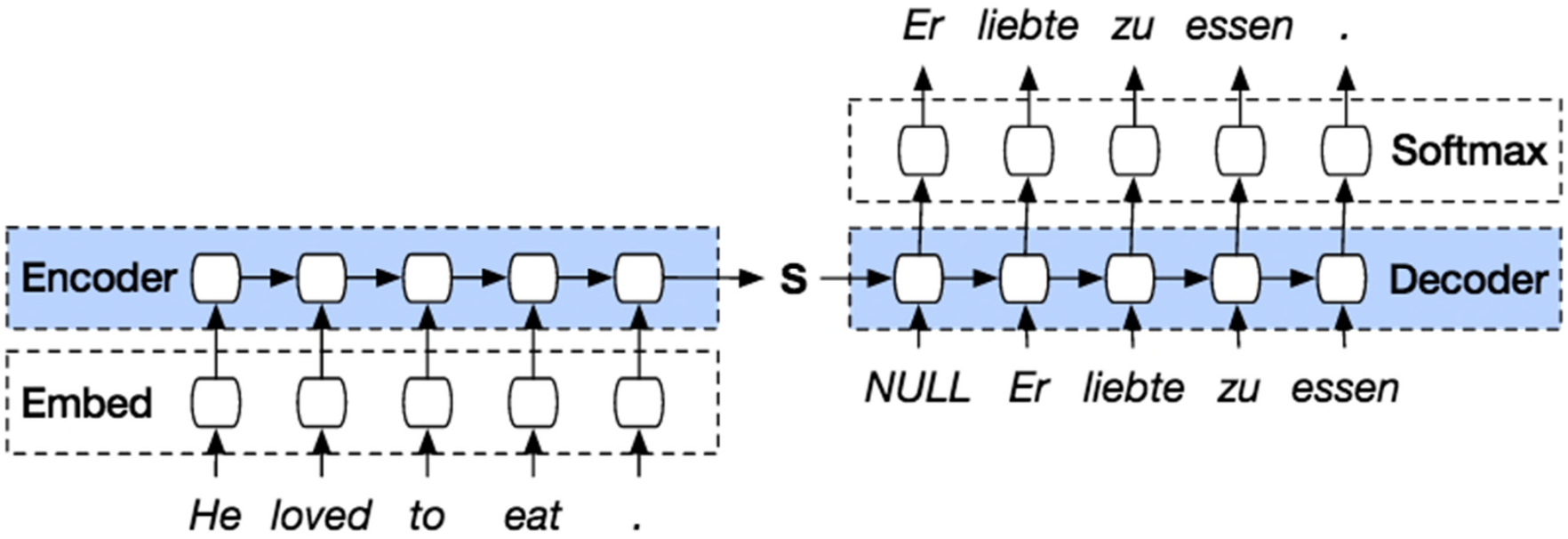
Sequence Modelling (Many-to-Many Mapping)

- Many inputs (sequence of inputs), many outputs (sequence of outputs)
- E.g., machine translation
  - Sequence of words → sequence of words
- Per-frame video classification
  - Sequence of images → sequence of labels





Sequence to Sequence: Encoder + Decoder Modelling



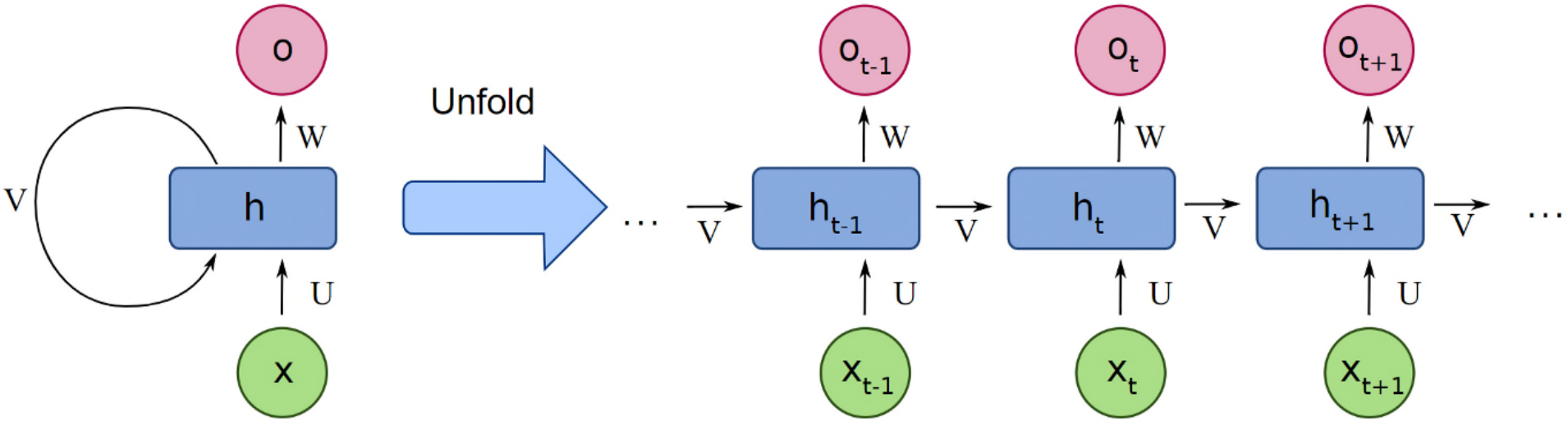
Source: https://medium.com/analytics-vidhya/seq2seq-model-and-the-exposure-bias-problem-962bb5607097

Recurrent Neural Networks (RNNs)

What are RNNs?

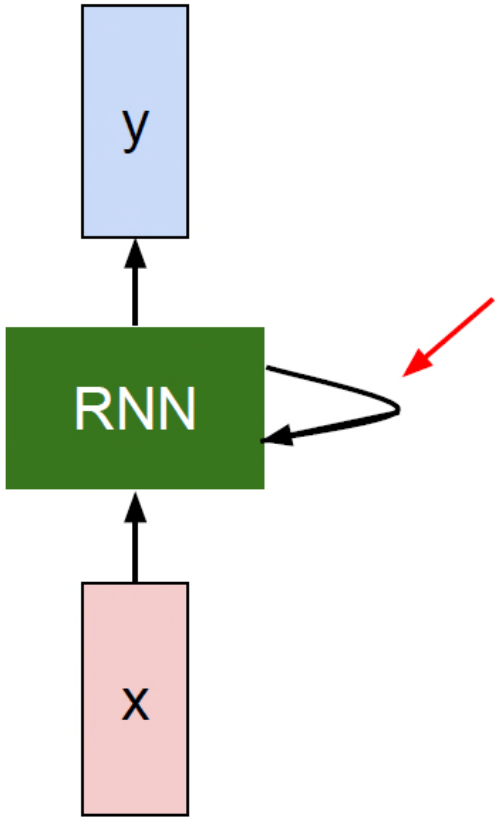
- RNNs are a class of neural networks which have inputs in the forms of sequential data, e.g., time series data.
- Commonly used in analysis of temporal / sequential data.
- Widely deployed in applications including Natural Language Processing (NLP), machine translation, image captioning, etc.

RNN Architecture



https://medium.com/deeplearningbrasilia/deep-learning-recurrent-neural-networks-f9482a24d010

RNN Model



Key idea: RNNs have an "internal state" that is updated as a sequence is processed

Source: Stanford Lecture Notes, CS231n

Hidden State Update

We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

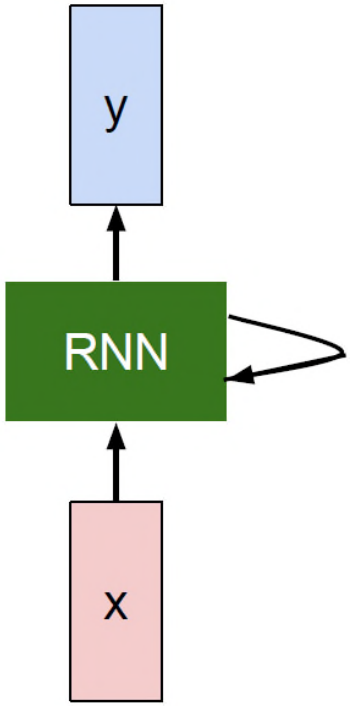
$$h_t = f_W(h_{t-1}, x_t)$$

new state

some function with parameters W

old state

input vector at some time step



Source: Stanford Lecture Notes, CS231n

Output Generation

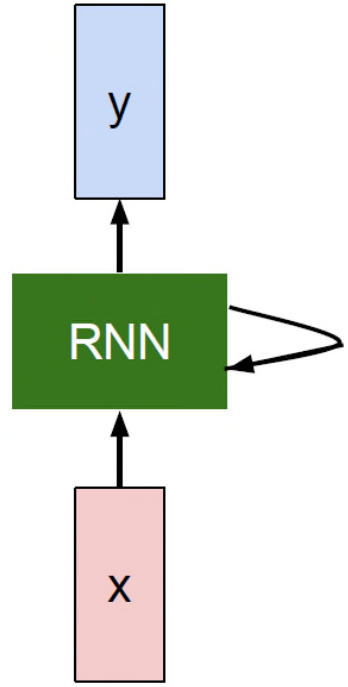
We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

$$y_t = f_{W_{hy}}(h_t)$$

output

new state

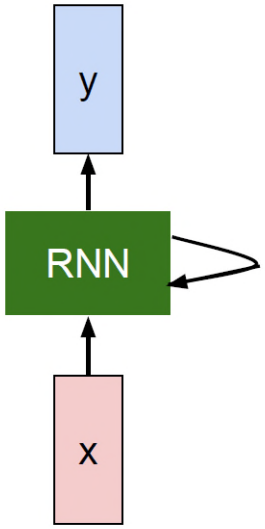
another function with parameters W_o



Source: Stanford Lecture Notes, CS231n

Vanilla RNN (Elman RNN)

The state consists of a single *"hidden"* vector **h**:

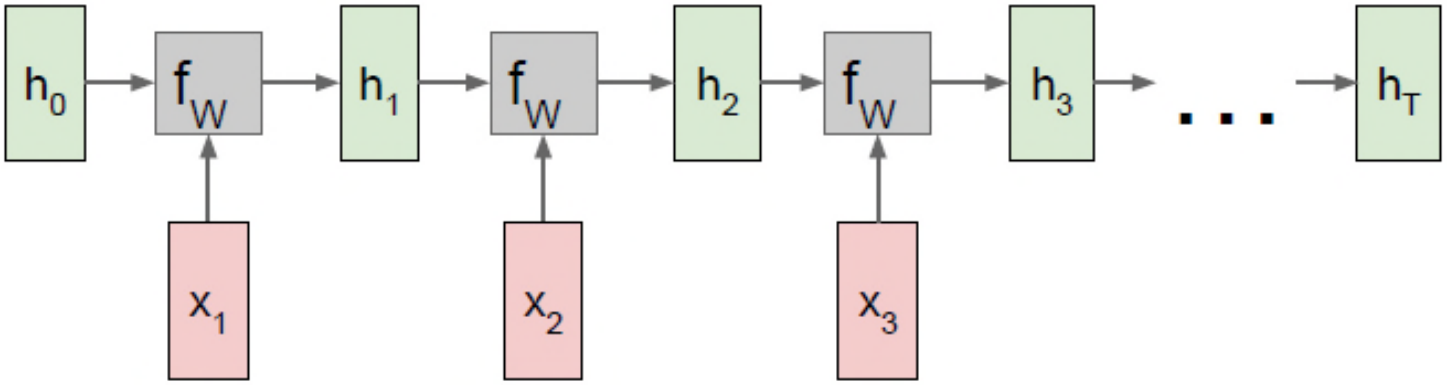


$$h_t = f_W(h_{t-1}, x_t)$$

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

Sometimes called a "Vanilla RNN" or an "Elman RNN" after Prof. Jeffrey Elman



Source: Stanford Lecture Notes, CS231n

Exercise: RNN

(a) A Vanilla Recurrent Neural Network (RNN) has the following settings.

Initial hidden state, $\mathbf{h}_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$,

Hidden state weight matrix, $\mathbf{W}_{hh} = \begin{bmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \end{bmatrix}$,

Input weight matrix, $\mathbf{W}_{xh} = \begin{bmatrix} 0.5 & 0.2 \\ 0.2 & 0.1 \end{bmatrix}$,

Output weight matrix, $\mathbf{W}_{hy} = \begin{bmatrix} 0.1 & 0.4 \end{bmatrix}$.

Assume no bias is used in the computation of the RNN.

A 2-timestep input is given by $\mathbf{x} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 \end{bmatrix}$ where $\mathbf{x}_1 = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$ and $\mathbf{x}_2 = \begin{bmatrix} 1 \\ 6 \end{bmatrix}$.

(i) Find the hidden state $\mathbf{h}_1$ at timestep $t = 1$.

(ii) Find the output y_1 at timestep $t = 1$.

(iii) Find the output y_2 at timestep $t = 2$.

Solution

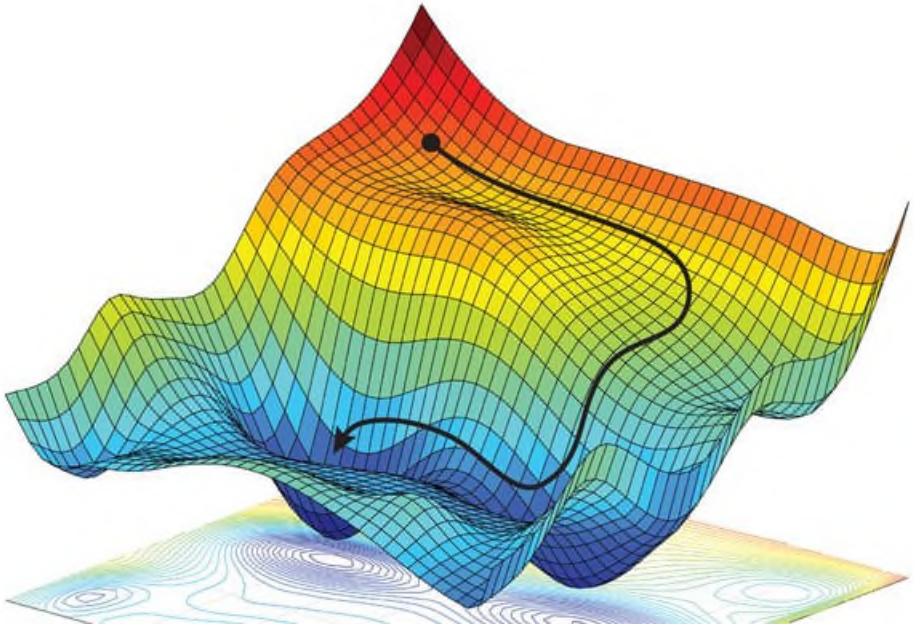
RNN Pros and Cons

- Pros:
  - Can process any length of input
  - Computation for step *t* can in theory use information from many steps back
  - Model size does not increase for longer input
  - Same weights applied for every timestep, so there is consistency in how inputs are processed.
- Cons:
  - Recurrent computation is slow
  - In practice, difficult to leverage information from many steps back

RNN Training & Optimization

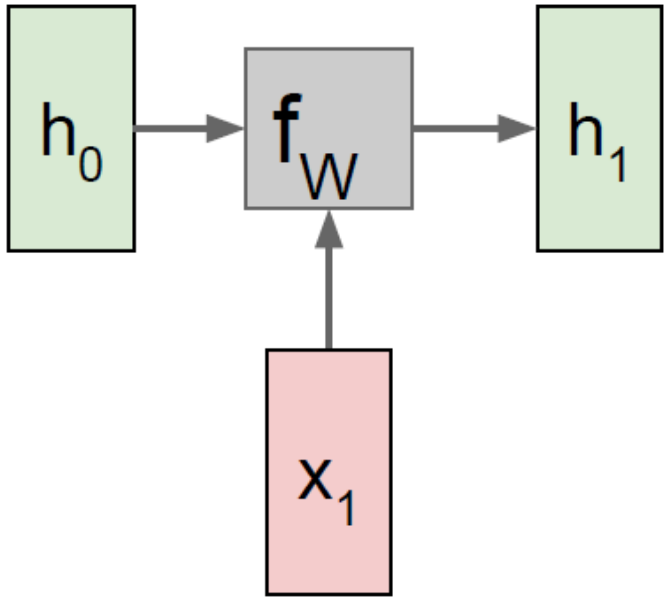
Recall: CNN Training

- The objective is to minimize/optimize the loss function.
- Common strategy is centered on stochastic gradient descent (SGD) and its extension.
- Use computational graph to compute the gradient descent and parameter update.



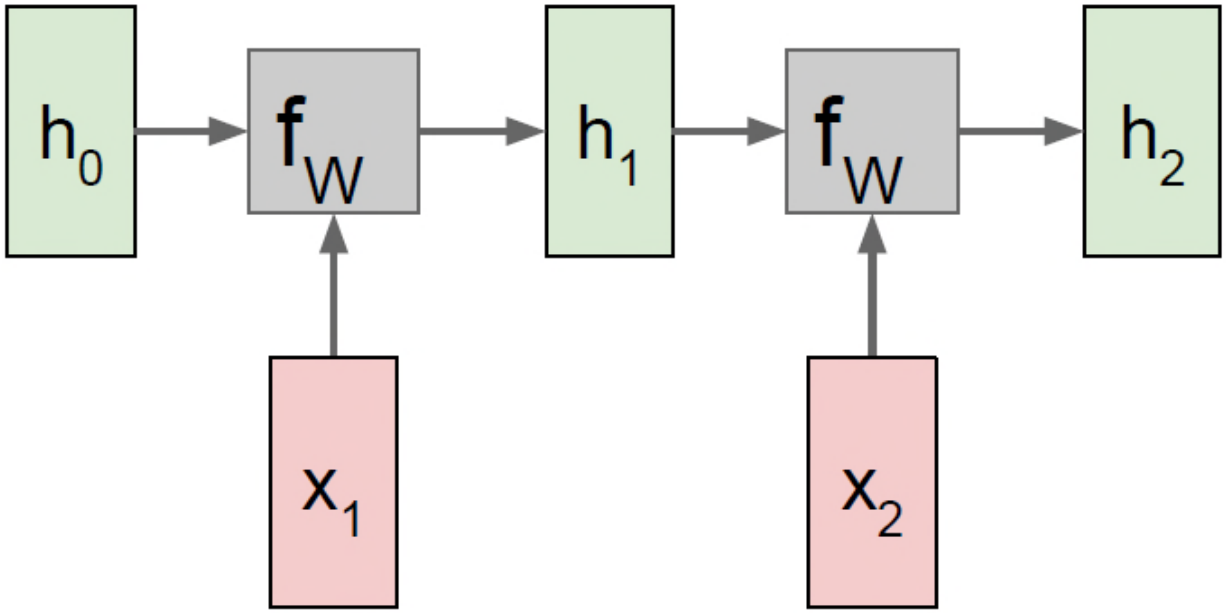
Source: “Coding Deep Learning for Beginners — Linear Regression Part 3”, Kamil Krzyk, towardsdatascience.com

Computational Graph (1)



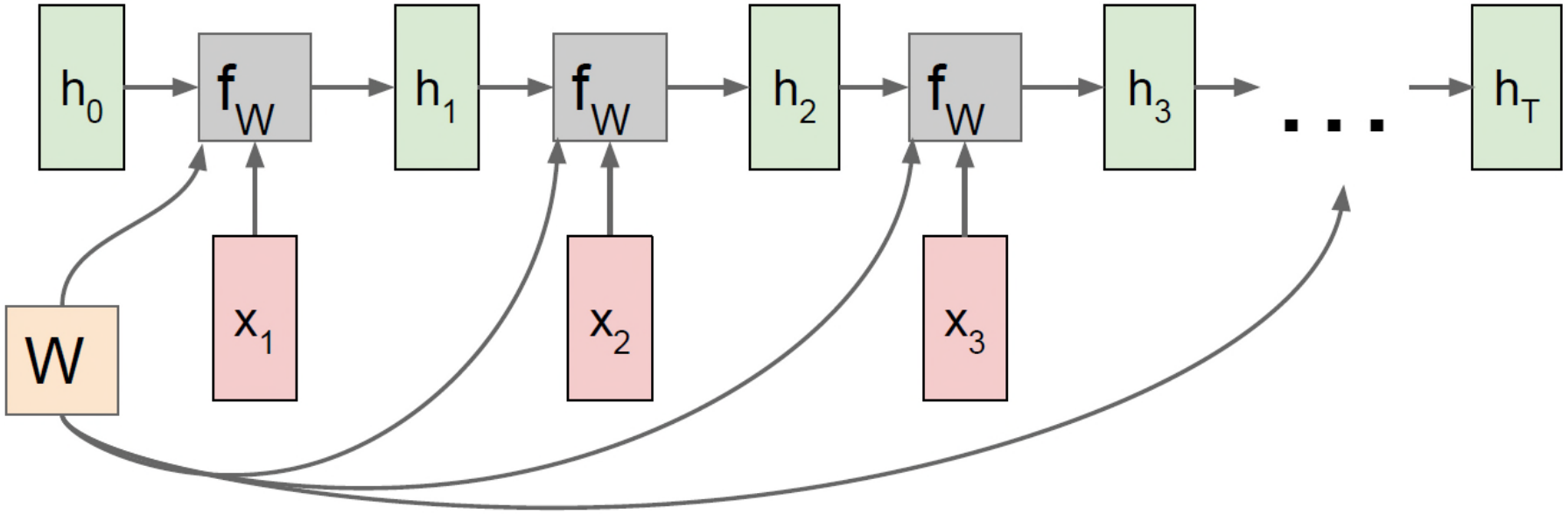
Source: Stanford Lecture Notes, CS231n

Computational Graph (2)

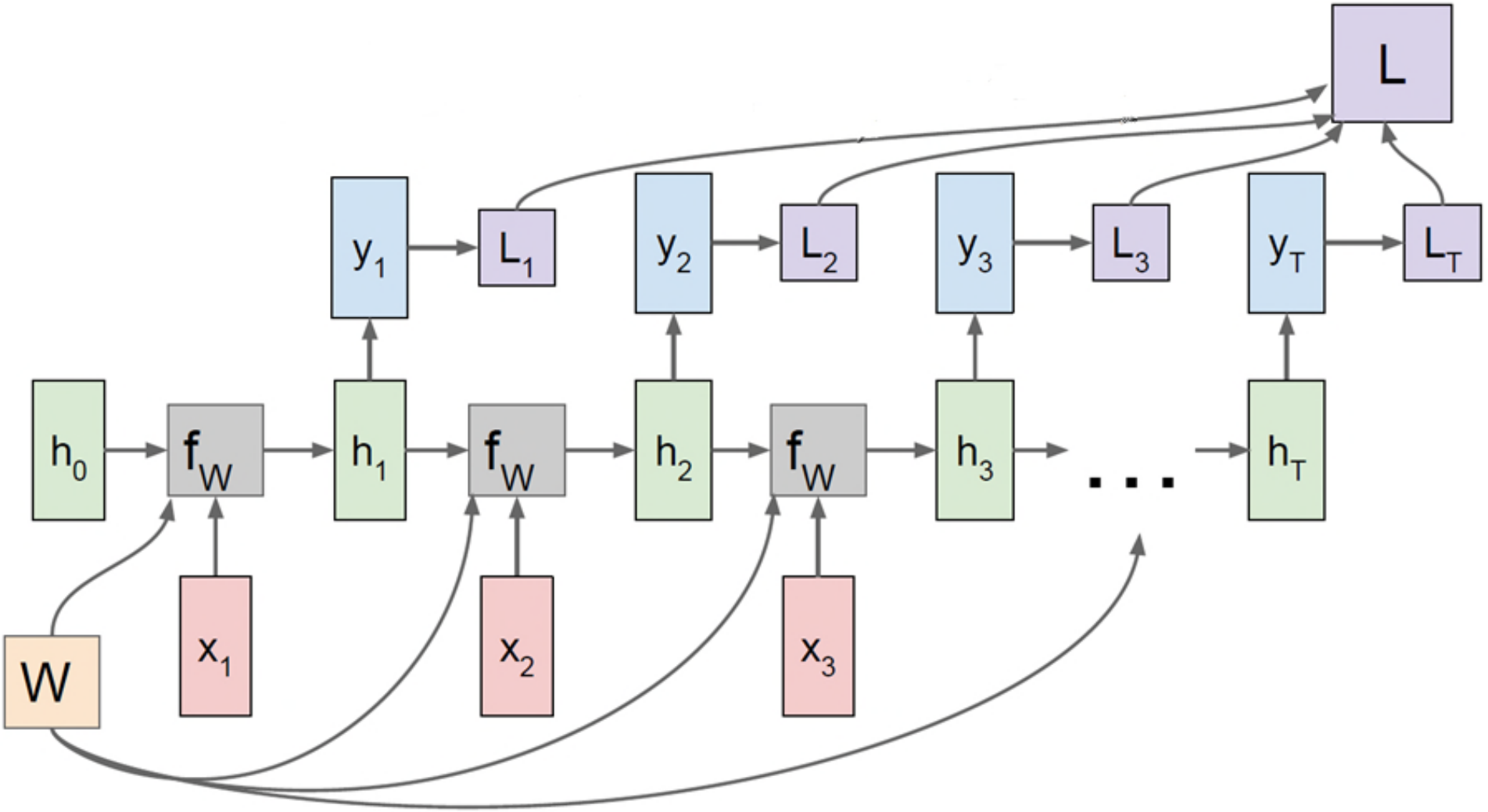


Computational Graph (3)

Re-use the same weight matrix at every time-step

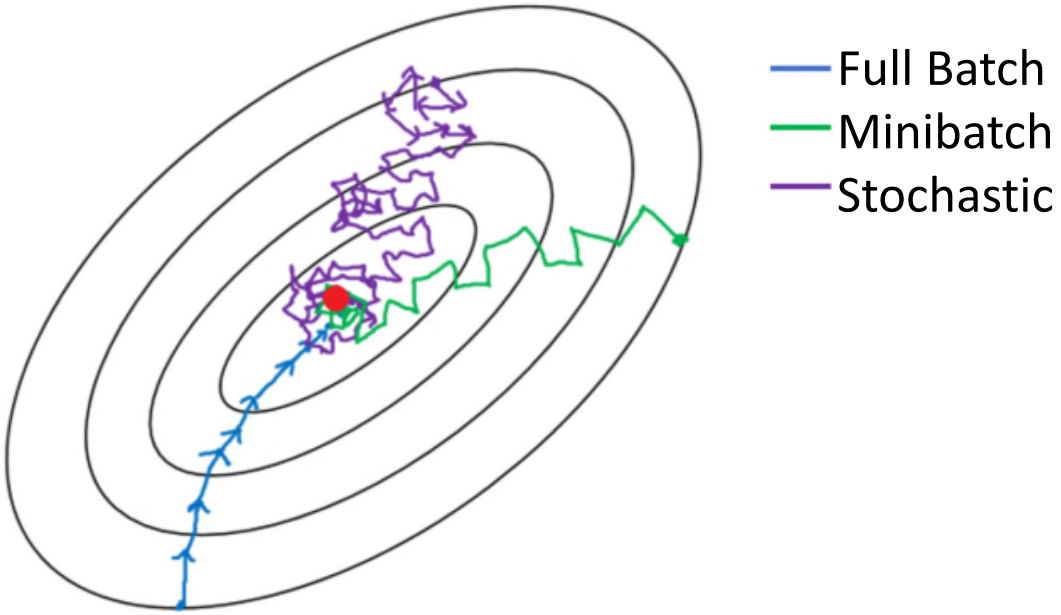


Computational Graph (4)



Batch Training

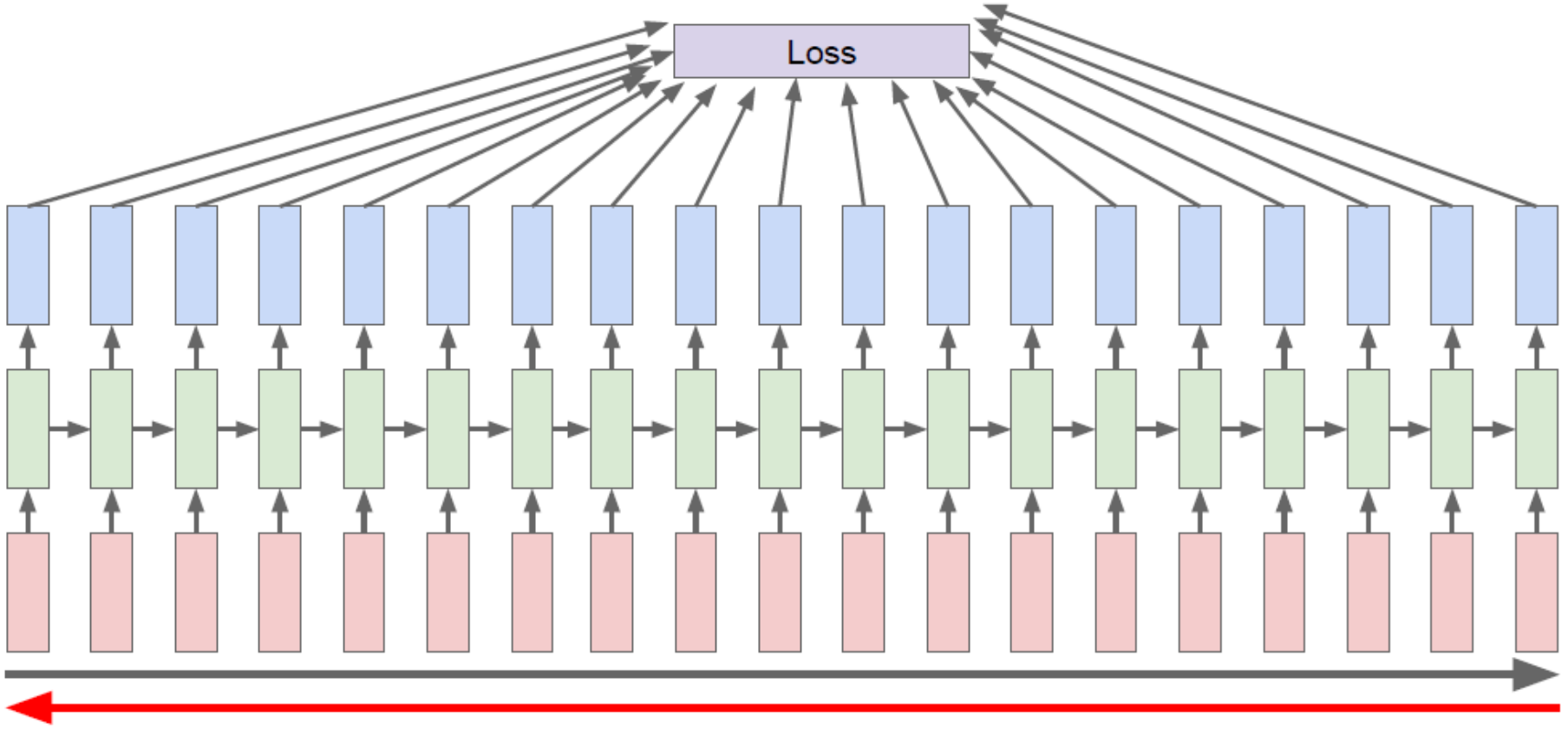
- Full Batch: use entire set of training sequences at each iteration
- Stochastic: use a single sequence at each iteration
- Minibatch (Stochastic): use a few training sequences at each iteration



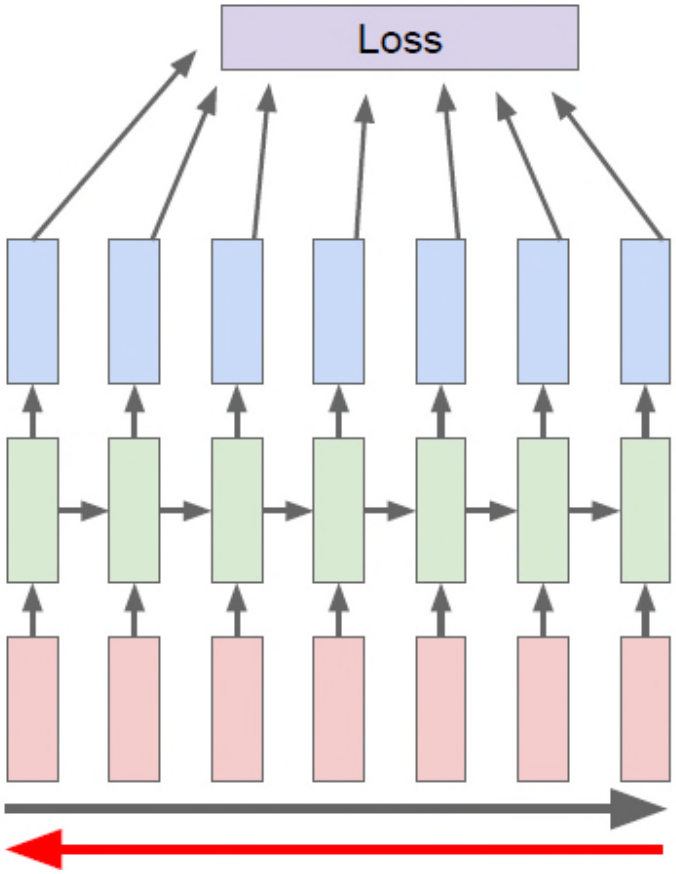
Source: https://medium.com/analytics-vidhya/gradient-descent-vs-stochastic-gd-vs-mini-batch-sgd-fbd3a2cb4ba4

Backpropagation Through Time (BPTT)

Forward through entire sequence to compute loss, then backward through entire sequence to compute gradient

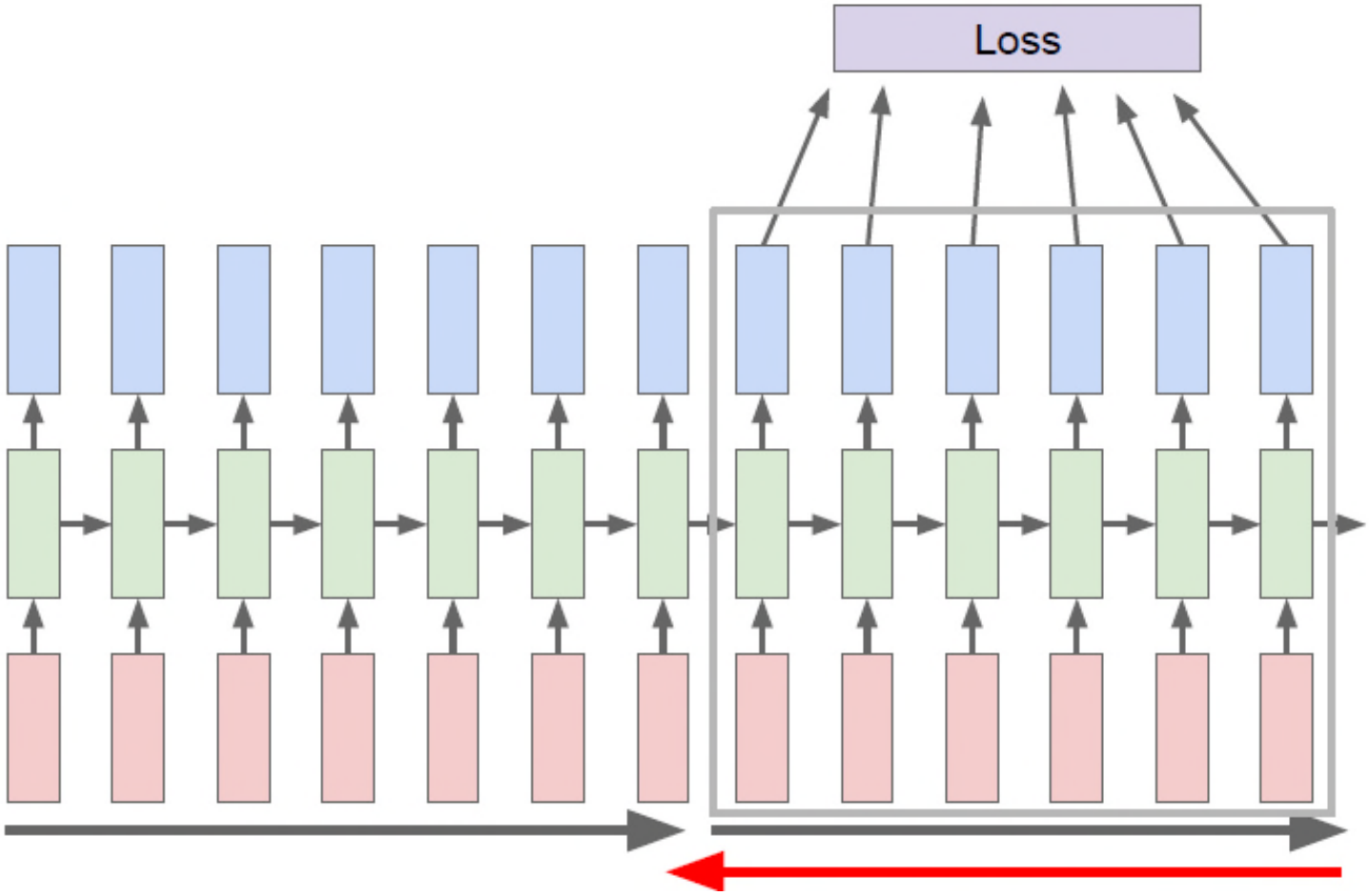


Truncated Backpropagation Through Time (1)



Run forward and backward through chunks of the sequence instead of whole sequence

Truncated Backpropagation Through Time (2)

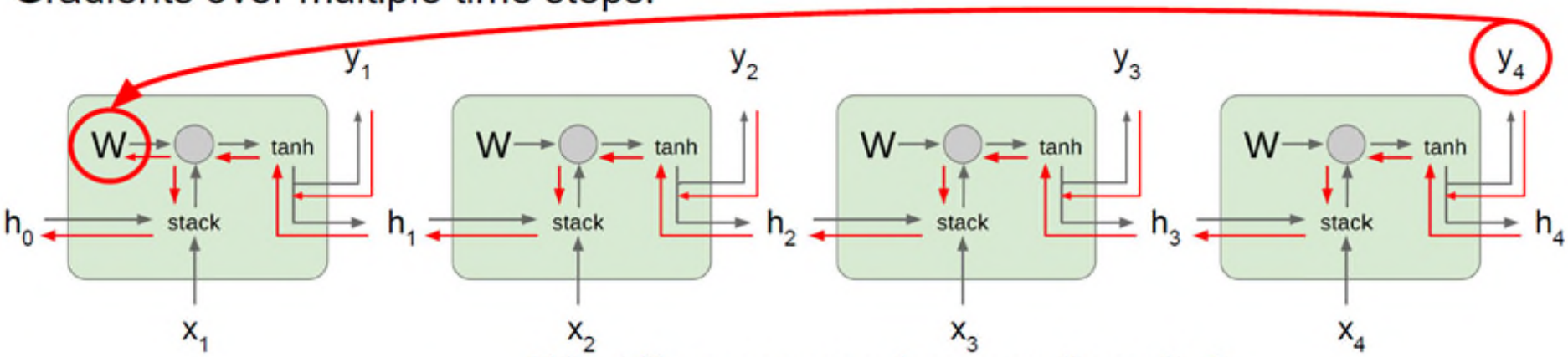


Carry hidden states forward in time forever, but only backpropagate for some smaller number of steps

Exploding / Vanishing Gradients (1)

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013

Gradients over multiple time steps:



What if we assumed no non-linearity?

$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} W_{hh}^{T-1} \frac{\partial h_1}{\partial W}$$

Largest singular value > 1:
Exploding gradients

→ **Gradient clipping**: Scale gradient if its norm is too big

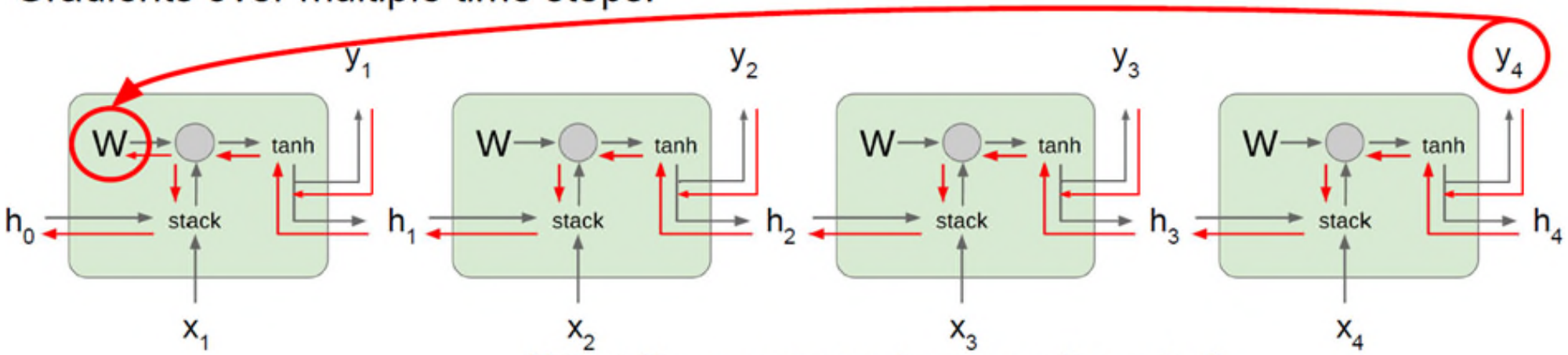
```
grad_norm = np.sum(grad * grad)
if grad_norm > threshold:
  grad *= (threshold / grad_norm)
```

Largest singular value < 1:
Vanishing gradients

Exploding / Vanishing Gradients (2)

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013

Gradients over multiple time steps:



What if we assumed no non-linearity?

$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

Largest singular value > 1:
Exploding gradients

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} W_{hh}^{T-1} \frac{\partial h_1}{\partial W}$$

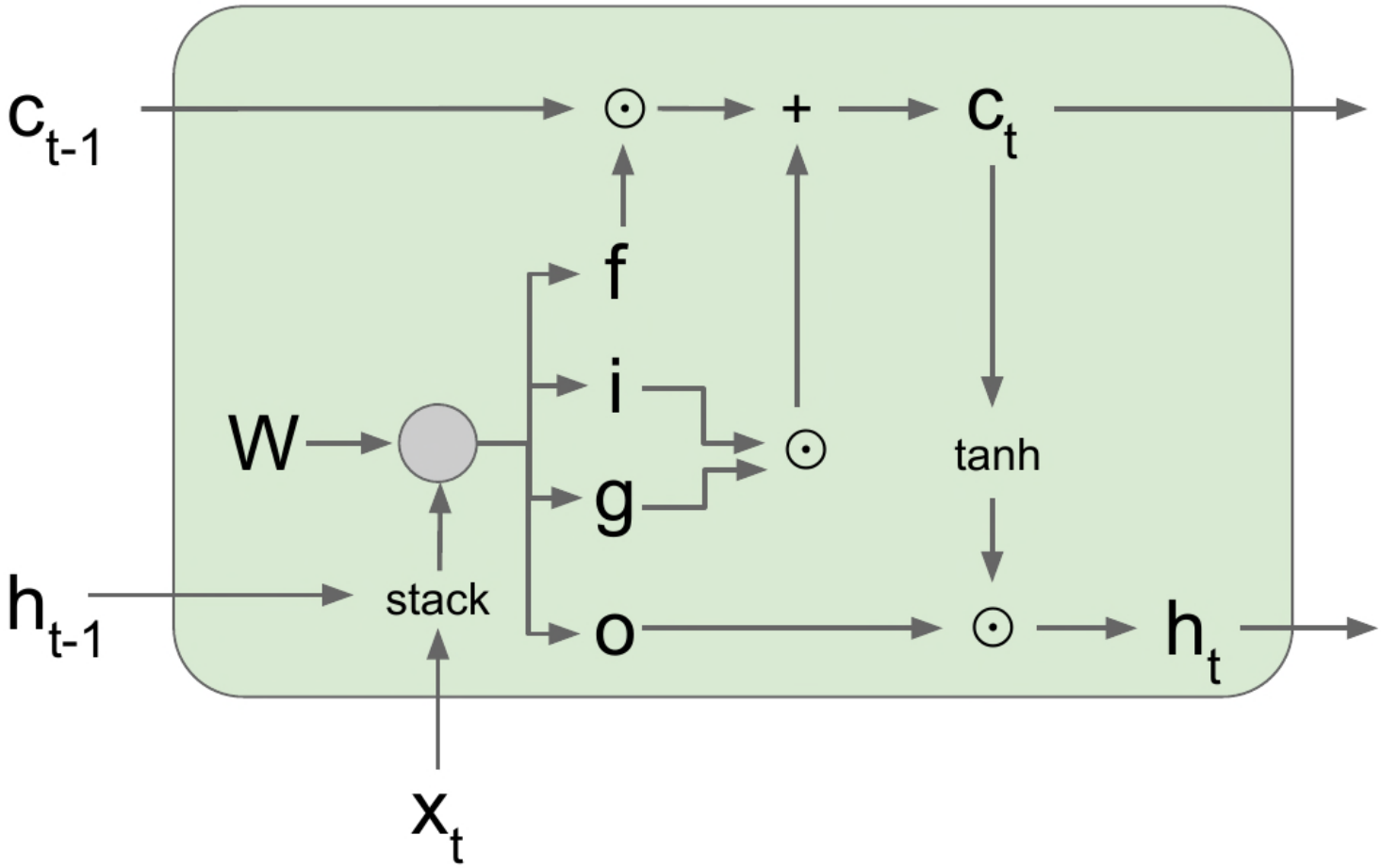
Largest singular value < 1:
Vanishing gradients

→ Change RNN architecture

Source: Stanford Lecture Notes, CS231n

Long Short-Term Memory (LSTM)

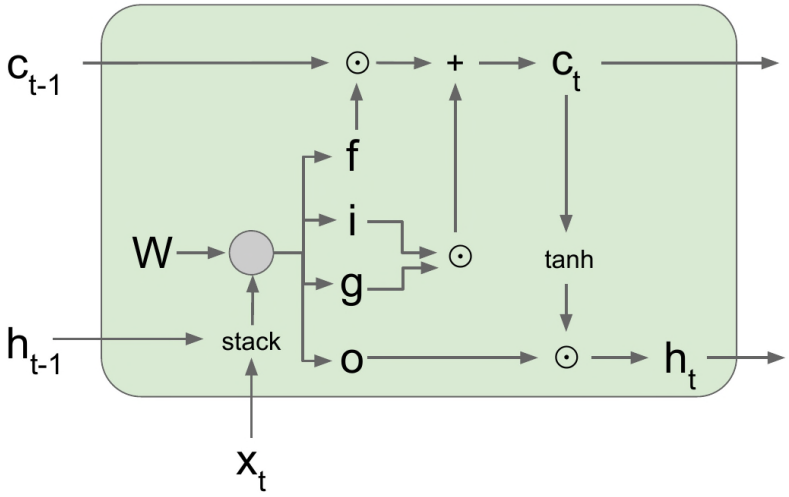
LSTM Architecture



Source: Stanford Lecture Notes, CS231n

Key Ideas

- A memory cell which can maintain its state over time.
- Consist of cell state (c_t), hidden state (h_t) and 4 gates (i, f, o, g).
- c_t : long-term memory, h_t : short-term memory.
- Cell state (c_t) undergoes changes via forgetting old memory (through forget (f) gate) and adding new memory (through input (i) gate & gate (g) gate).
- Hidden state (h_t) is updated by passing cell state (c_t) through output gate.
- Gates control the flow of information to the memory.
- Gate are obtained through a sigmoid/tanh layer, and they update c_t and h_t cell states using pointwise multiplication operator.



$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

RNNs vs LSTMs

Vanilla RNN

$$h_t = \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

LSTM

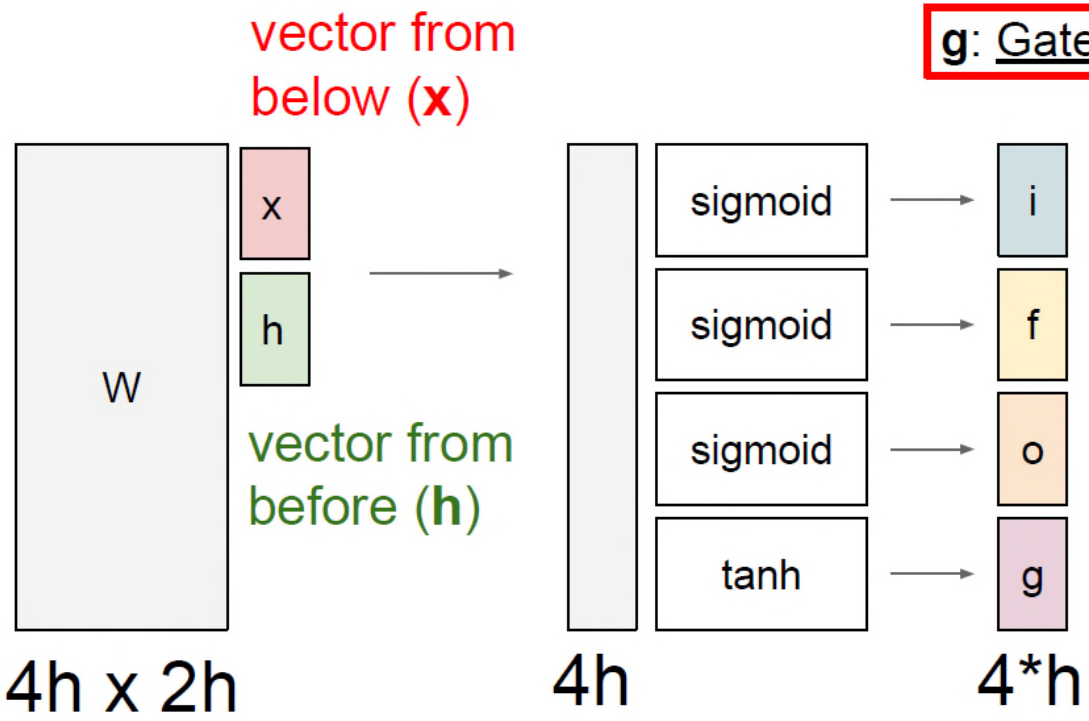
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Source: Stanford Lecture Notes, CS231n

LSTM Computation (1)



g: Gate gate (?), How much to write to cell

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

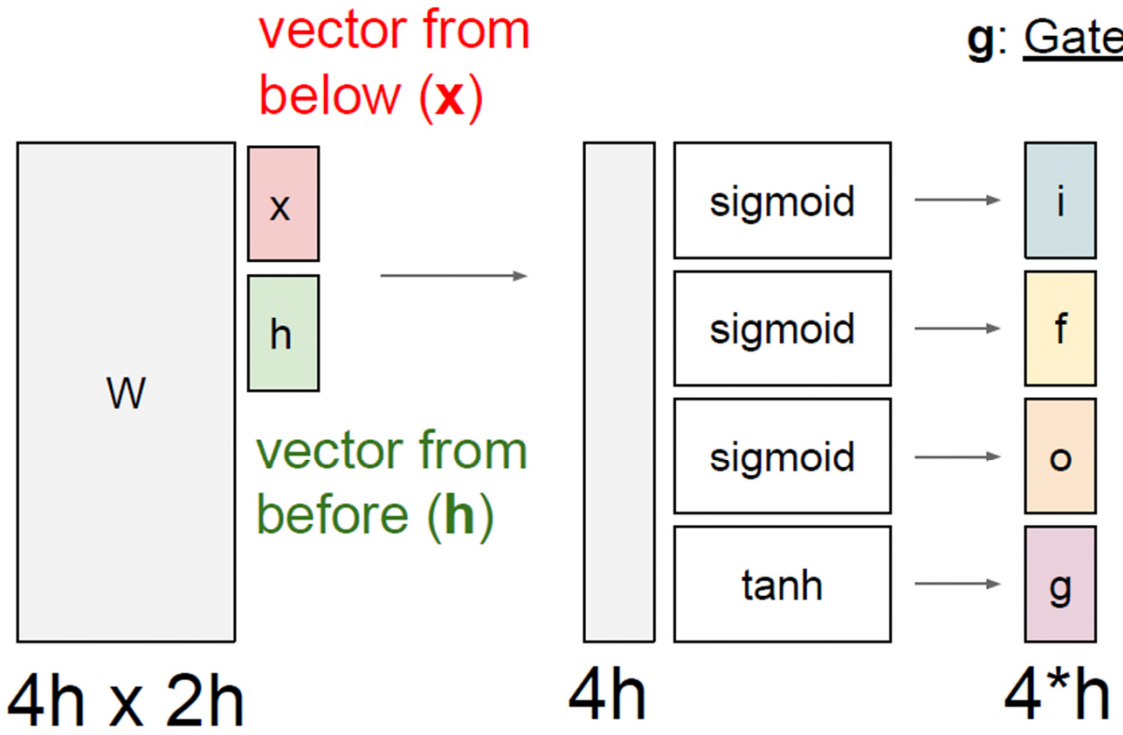
$$h_t = o \odot \tanh(c_t)$$

Source: Stanford Lecture Notes, CS231n

LSTM Computation (2)

i: Input gate, whether to write to cell

g: Gate gate (?), How much to write to cell



$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

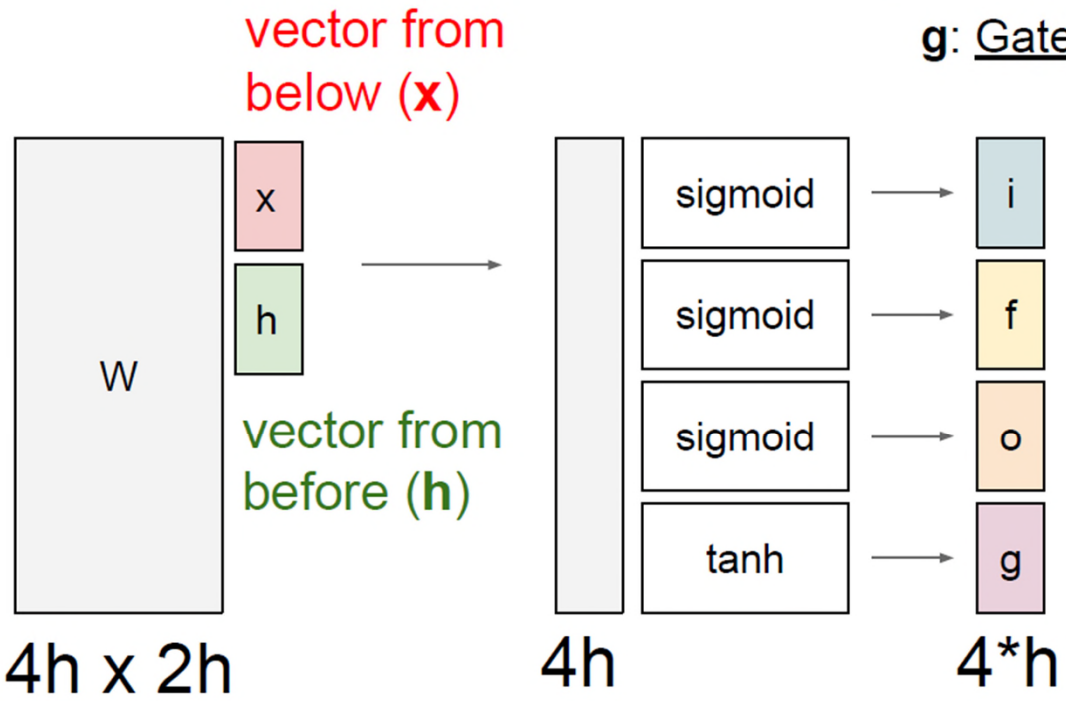
Source: Stanford Lecture Notes, CS231n

LSTM Computation (3)

i: Input gate, whether to write to cell

f: Forget gate, Whether to erase cell

g: Gate gate (?), How much to write to cell



$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

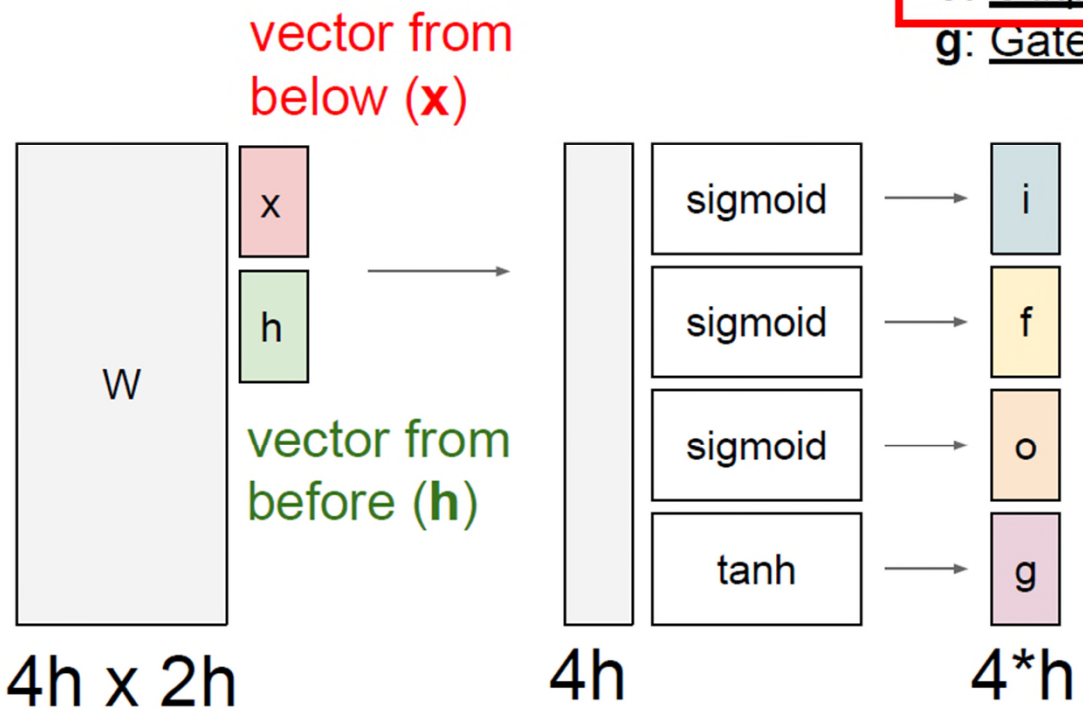
$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Source: Stanford Lecture Notes, CS231n

LSTM Computation (4)

i: Input gate, whether to write to cell
f: Forget gate, Whether to erase cell
o: Output gate, How much to reveal cell
g: Gate gate (?), How much to write to cell



$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Source: Stanford Lecture Notes, CS231n

Exercise: LSTM

(a) A Long Short-Term Memory (LSTM) network has the following settings.

Initial hidden state, $\mathbf{h}_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$, Initial cell state, $\mathbf{c}_0 = \begin{bmatrix} 0.1 \\ 0.2 \end{bmatrix}$,

Forget gate weight matrix, $\mathbf{W}_f = \begin{bmatrix} \mathbf{W}_{hf} & \mathbf{W}_{xf} \end{bmatrix} = \begin{bmatrix} 0.1 & 0.2 & 0.5 & 0.6 \\ 0.3 & 0.4 & 0.7 & 0.8 \end{bmatrix}$,

Input gate at timestep $t = 1$, $\mathbf{i}_1 = \begin{bmatrix} 0.3 \\ 0.4 \end{bmatrix}$,

Gate gate at timestep $t = 1$, $\mathbf{g}_1 = \begin{bmatrix} 0.5 \\ 0.6 \end{bmatrix}$,

Output gate at timestep $t = 1$, $\mathbf{o}_1 = \begin{bmatrix} 0.4 \\ 0.6 \end{bmatrix}$,

Input at timestep $t = 1$, $\mathbf{x}_1 = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$.

Assume no bias is used in the computation of the LSTM. The sigmoid and tanh functions are given as follows.

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

(i) Find the forget gate $\mathbf{f}_1$ at timestep $t = 1$. Comment on your obtained result.

(ii) Find the cell state $\mathbf{c}_1$ at timestep $t = 1$.

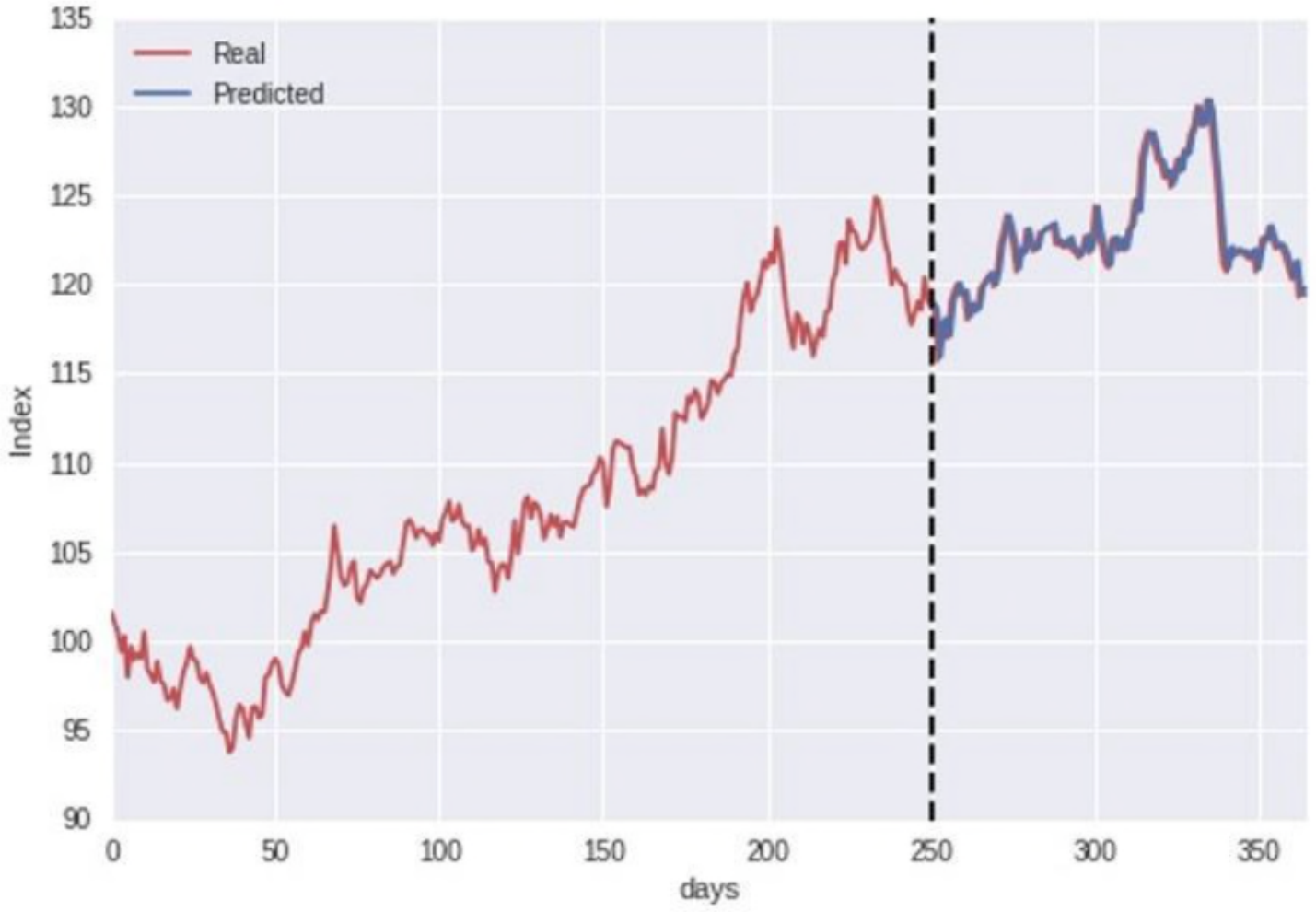
(iii) Find the hidden state $\mathbf{h}_1$ at timestep $t = 1$.

(13 Marks)

Solution

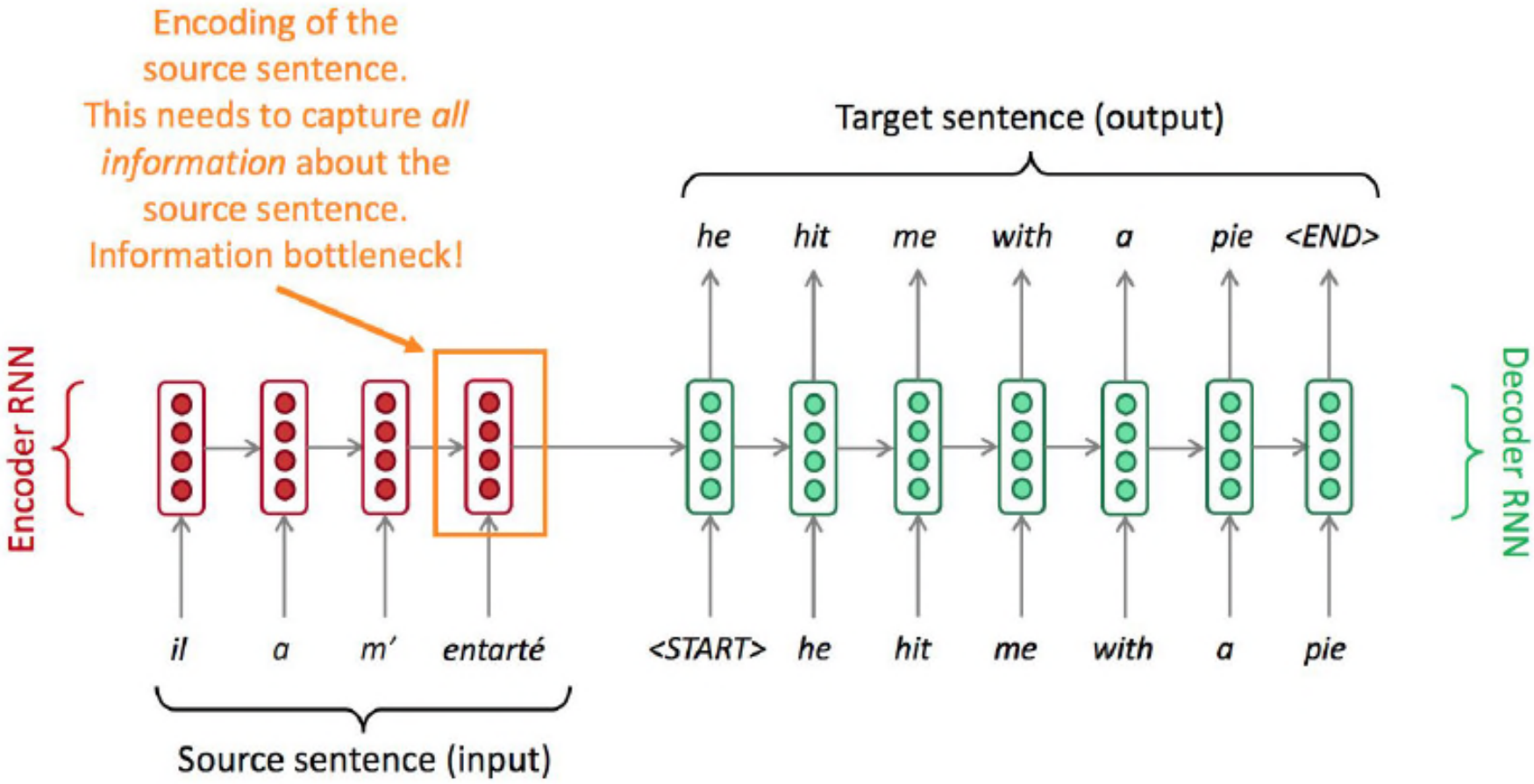
Applications

Time Series Prediction



Source: https://www.kdnuggets.com/2019/05/machine-learning-time-series-forecasting.html

Machine Translation



Source: Stanford Lecture Notes, CS224n

Section II Summary

- The section covers the following topics:
  - Introduction
  - Recurrent Neural Network (RNN)
  - RNN Training & Optimization
  - Long Short-Term Memory (LSTM)
  - Applications

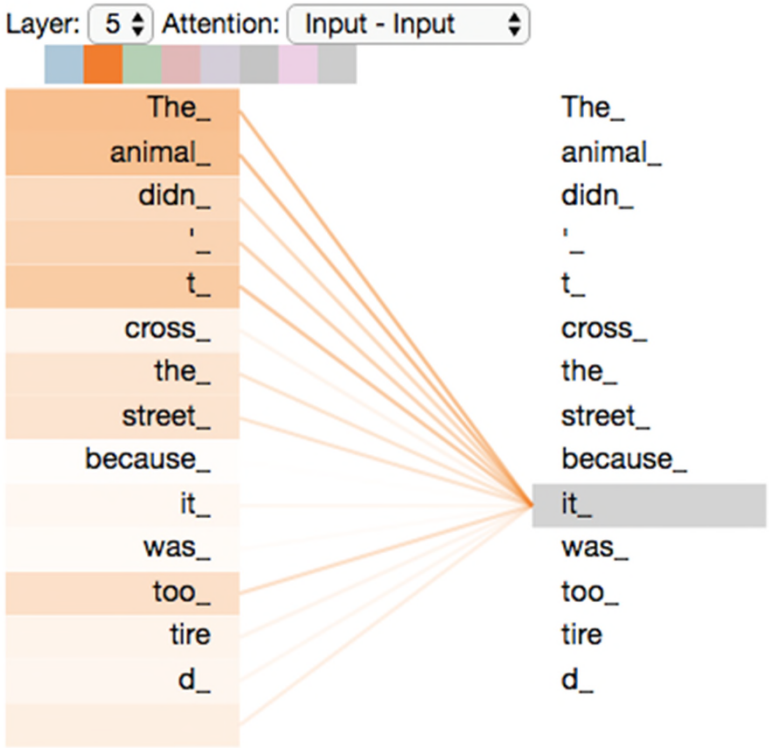
Section III
Transformer

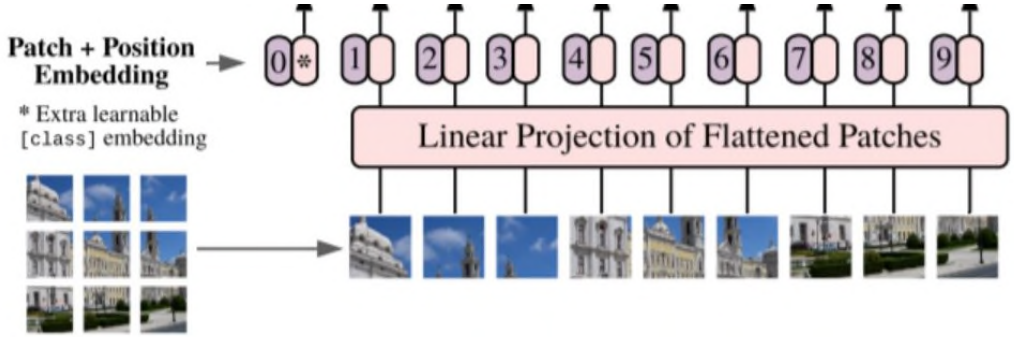
Section III Overview

- The section covers the following topics:
  - Attention Concept
  - Transformer Architecture
  - Vision Transformer

What is Attention?

- Attention is used to determine which input tokens (e.g., words in NLP, image patches in CV) are relevant to the current input / token.
- Attention is computed through correlation (dot product) between 2 vectors.
- Correlation -> similarity / relatedness / importance -> attention





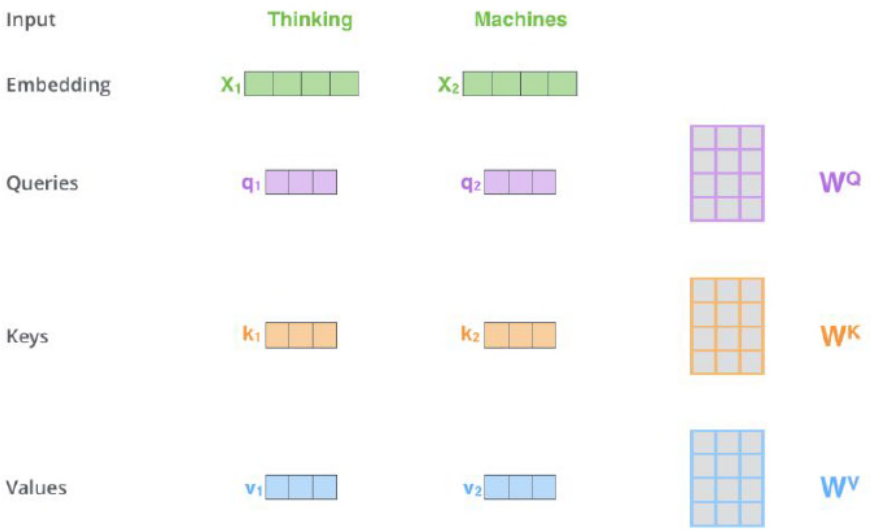
Source: https://jalammar.github.io/illustrated-transformer/

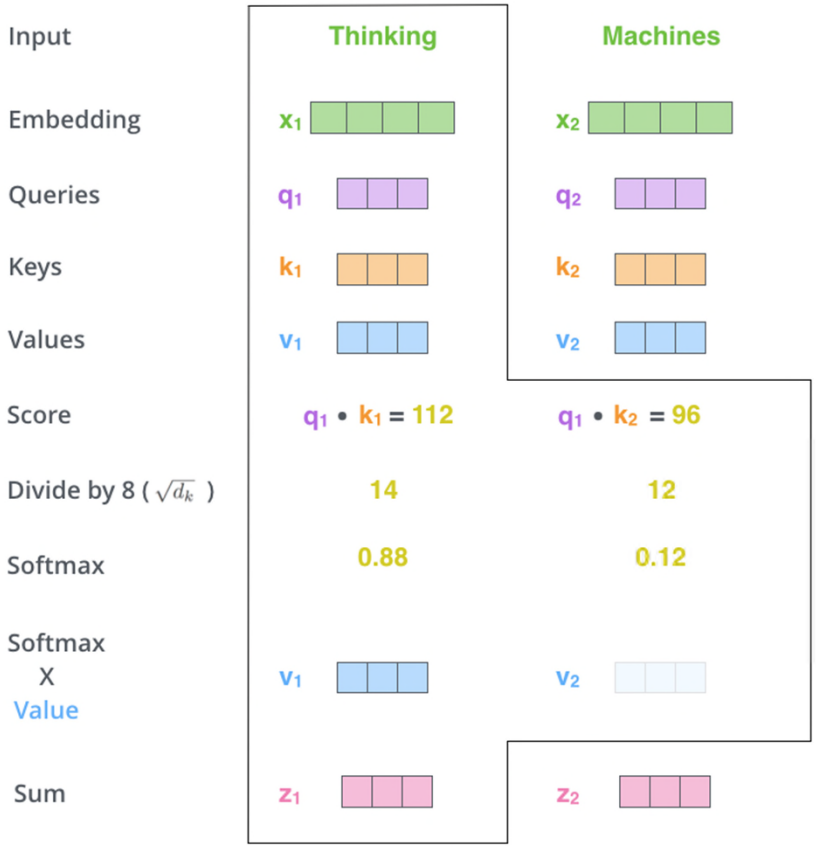
An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale (2020), Alexey Dosovitskiy et. al.

How is Attention Computed?

- Each input token generates 3 vectors: query (q), key (k) and value (v). These vectors provide more flexible representation through linear mapping (W_Q, W_K, W_v) to learn the underlying relationship / attention between input tokens.
- Attention is computed using:
  - Step 1: compute the correlation (dot product) between the query (q) and key (k) vectors.
  - Step 2: correlation values from Step 1 are scaled and normalized using Softmax function.
  - Step 3: multiplied output from Step 2 by corresponding value (v) vectors and sum them up.

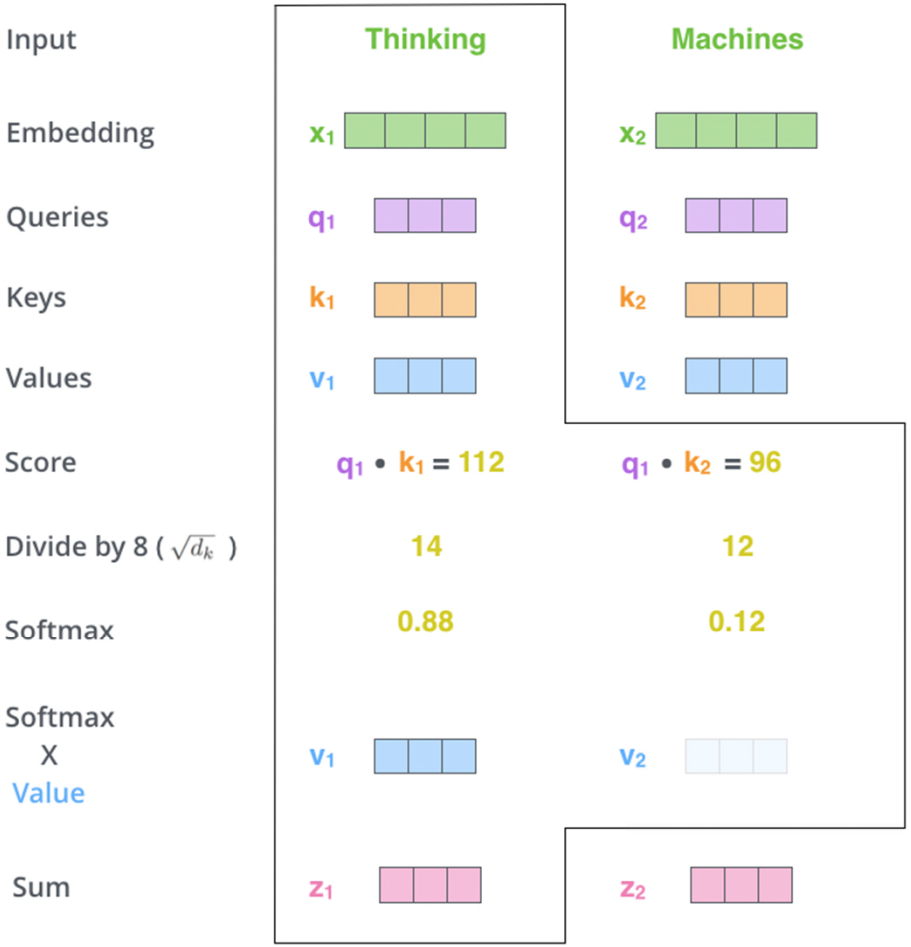
$$A(Q,K,V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$



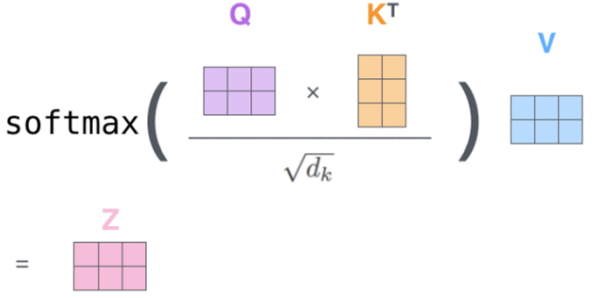


Scaled Dot-Product Attention

- Step 1: compute the correlation (dot product) between the query (q) and key (k) vectors.
- Step 2: correlation values from Step 1 are scaled and normalized using Softmax function.
- Step 3: multiplied output from Step 2 by corresponding value (v) vectors and sum them up.



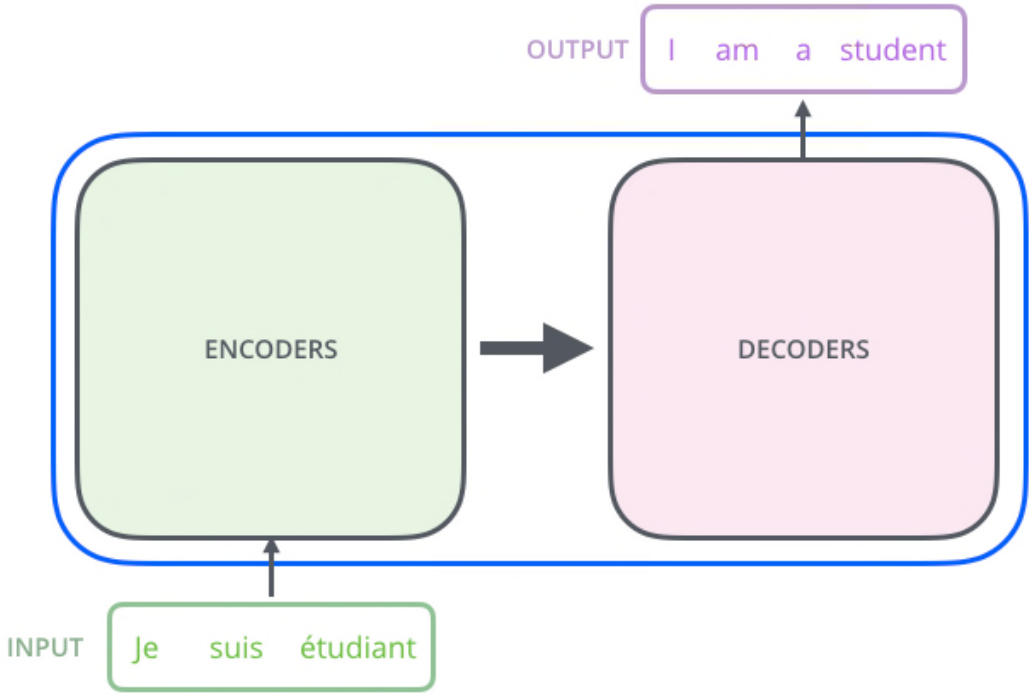
$$A(Q,K,V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$



What is a Transformer?

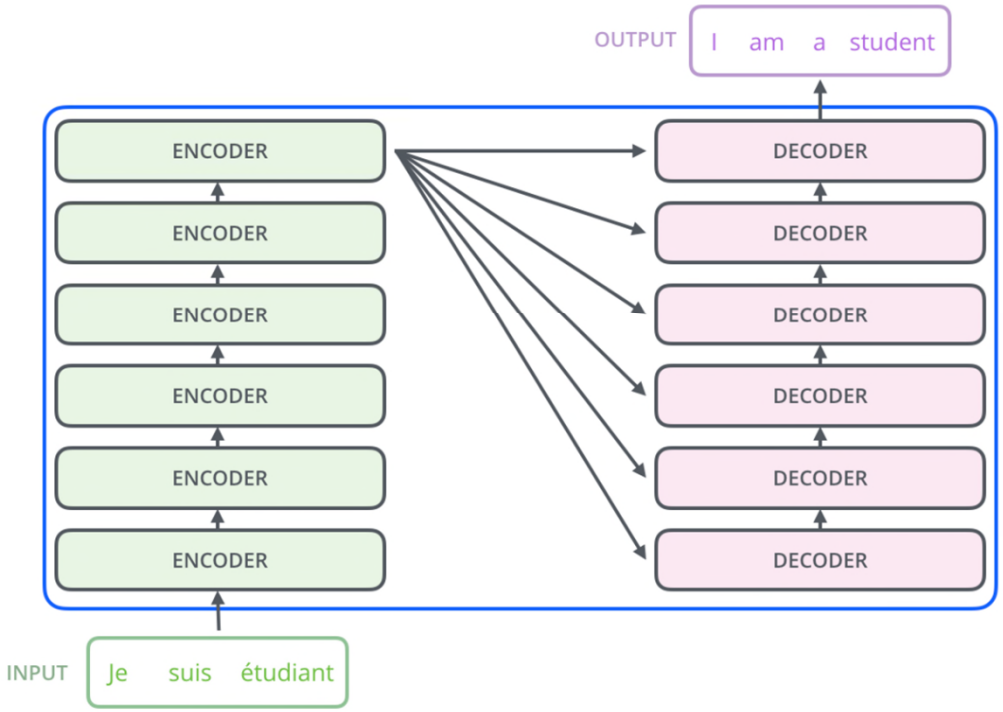
- Transformer uses attention mechanism to process input sequence in parallel.
- Use attention mechanism and dense/feedforward/MLP layers.
- Highly parallelizable.
- Can offer global attention.
- Good at modelling long-range dependency.
- Achieve state-of-the-art performance in many vision and NLP applications.
- Lead to other SOTA methods such as BERT: Pre-training of Deep Bidirectional Transformers for Language.

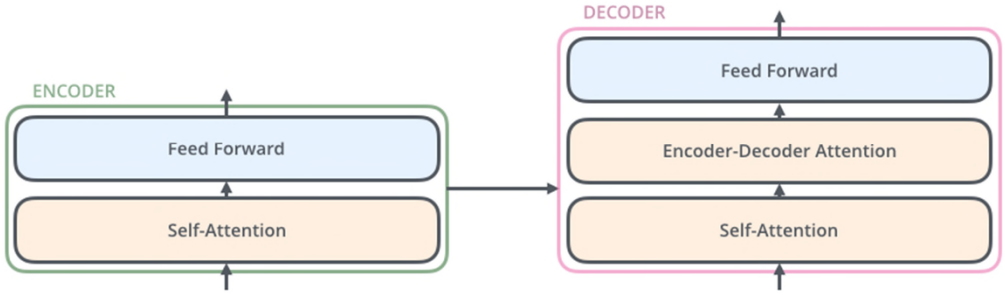
Transformer in Machine Translation (1)

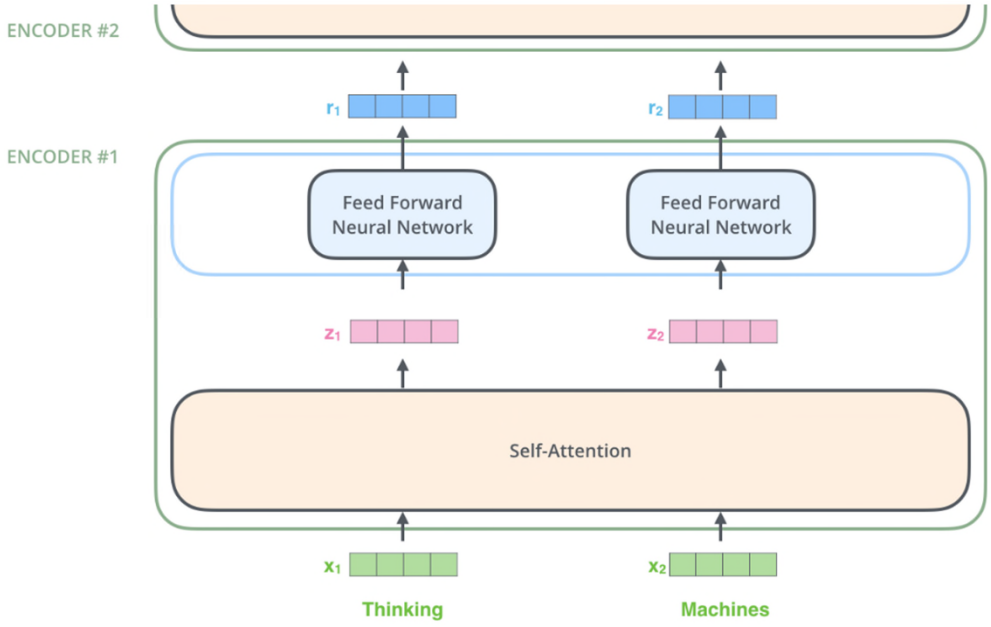


Source: https://jalammar.github.io/illustrated-transformer/

Transformer in Machine Translation (2)







Source: https://jalammar.github.io/illustrated-transformer/

Attention is All You Need

- Initially designed for neural machine translation.
- Later extended to visual tasks such as recognition, detection, etc, with great success.
- Leverage on attention mechanism to analyze the importance of a token (word/image patch) with respect to other tokens/image patches.
- Consist of transformer encoder and transformer decoder.

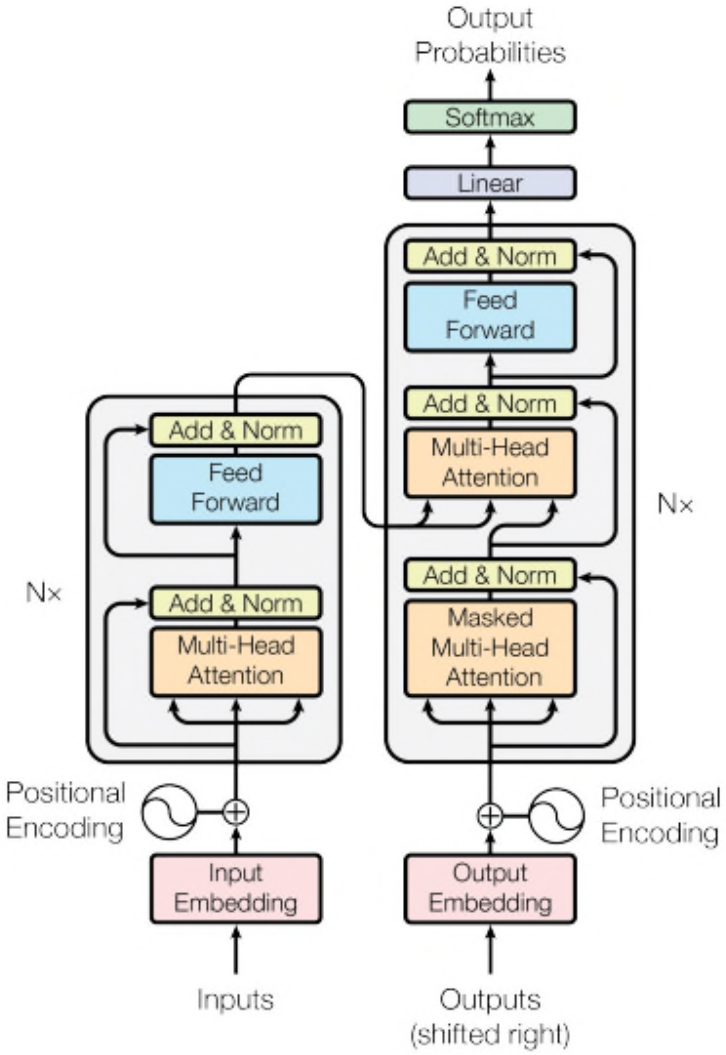


Figure 1: The Transformer - model architecture.

Attention Is All You Need (2017) https://arxiv.org/abs/1706.03762
Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin

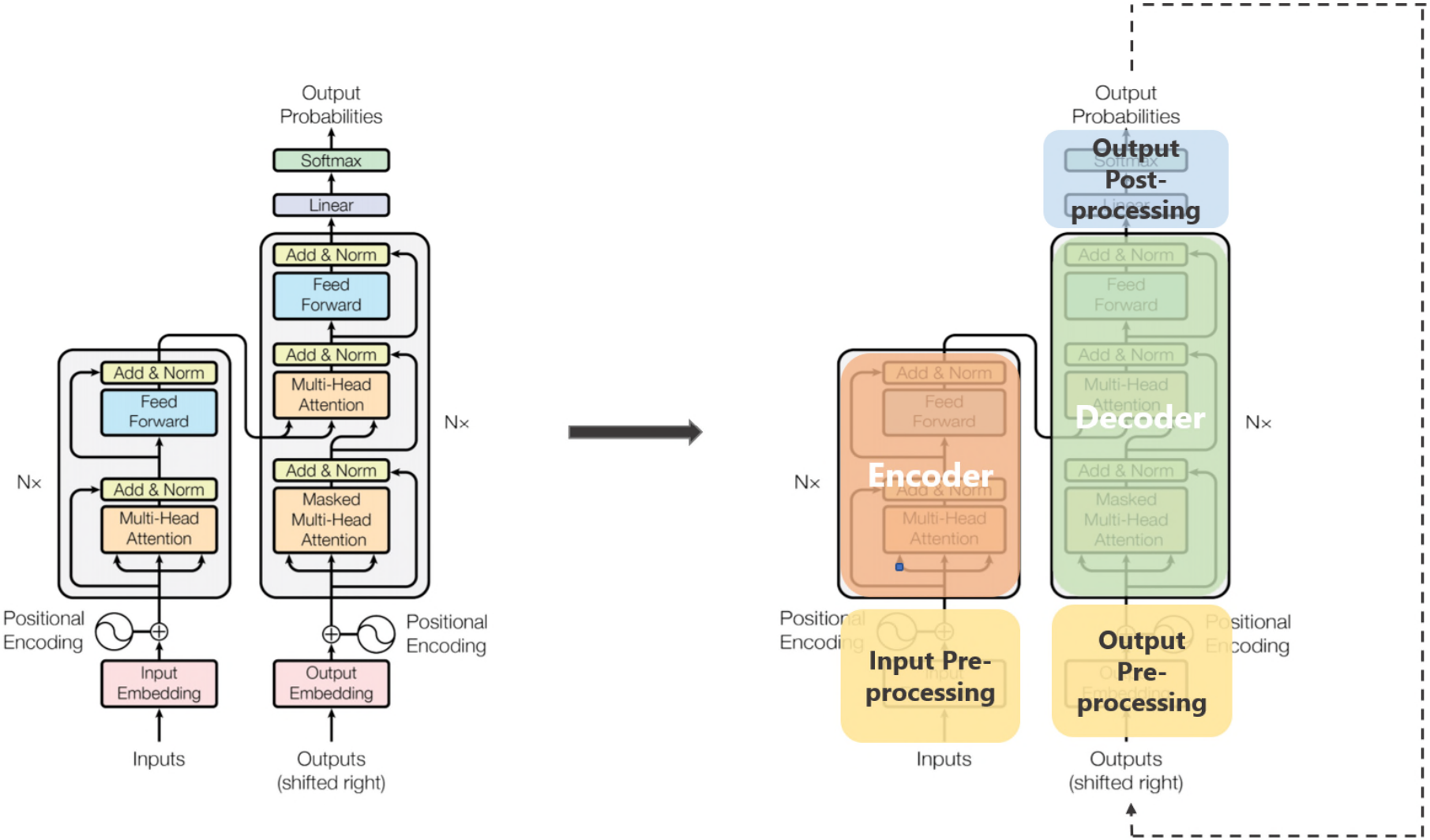
Transformer Encoder

- Input pre-processing:
  - Map input words / tokens (e.g., French words) into text embedding / vectors.
  - Add position encoding info of input words.
- Encoder:
  - Map input vectors from pre-processing into context vectors using attention mechanism.
  - Context vectors pass through feedforward layer to generate encoder outputs.
  - Encoder outputs have better representation than input vectors as they leverage the context information of other input tokens due to attention mechanism.

Transformer Decoder

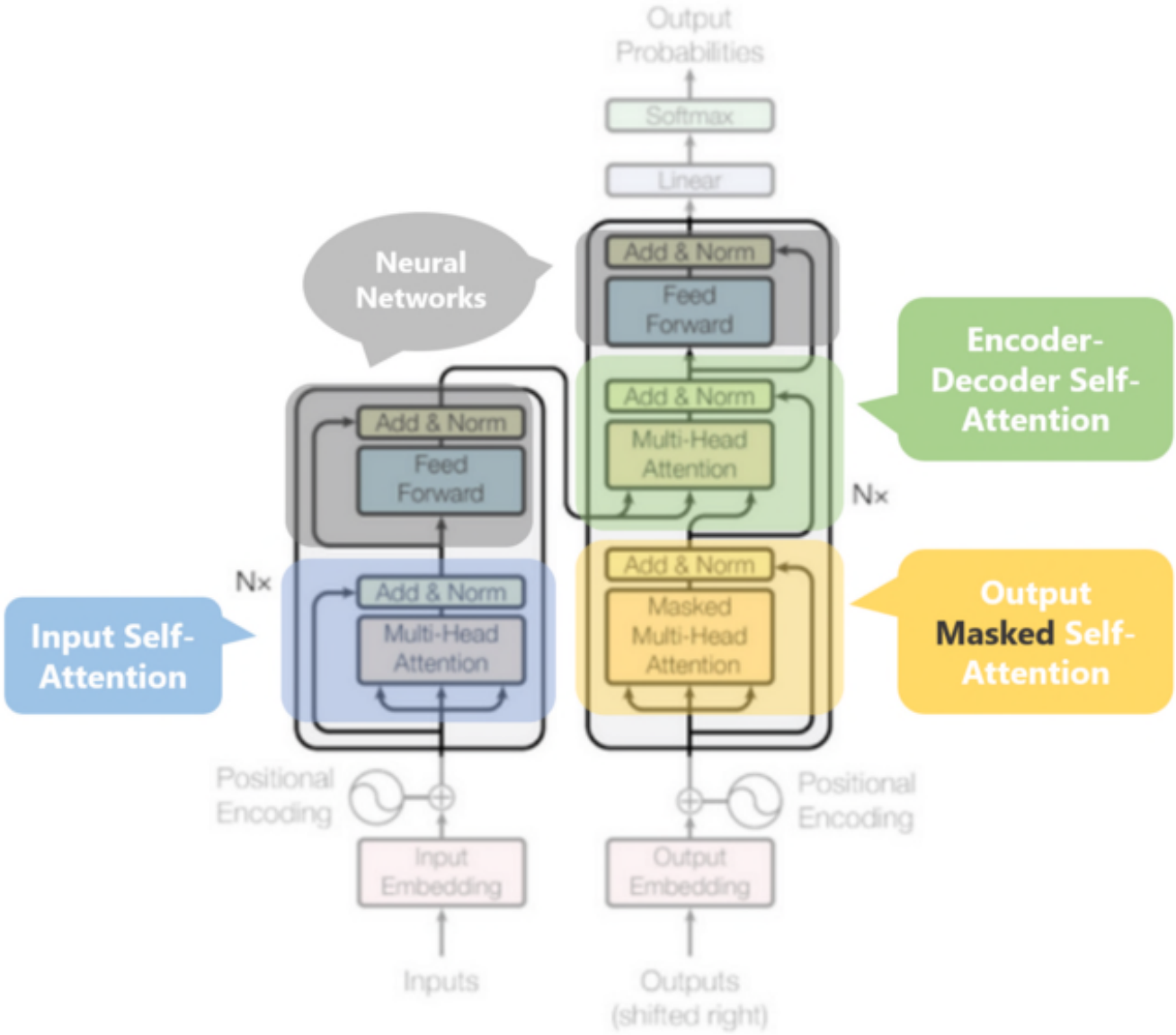
- Output pre-processing:
  - Map output words / tokens (e.g. English words) into text embedding.
  - Add position encoding info of output words.
- Decoder:
  - Output masked self-attention: map output vectors into context vectors of output words (e.g., English words). Masking is used to hide unseen words during training.
  - Encoder decoder self-attention (or cross-attention): perform cross-attention between context vectors from self-attention stage (e.g., English words) and encoder outputs (e.g., French words).
  - Resulting vectors pass through feedforward layer to generate decoder outputs.
  - Decoder outputs leverage (1) self-attention of English word and (2) cross attention French-English words to obtain better representation.
- Output post-processing:
  - Map the decoder outputs into probability using Softmax function to generate the next output word (i.e., English word).

Transformer Architecture (1)



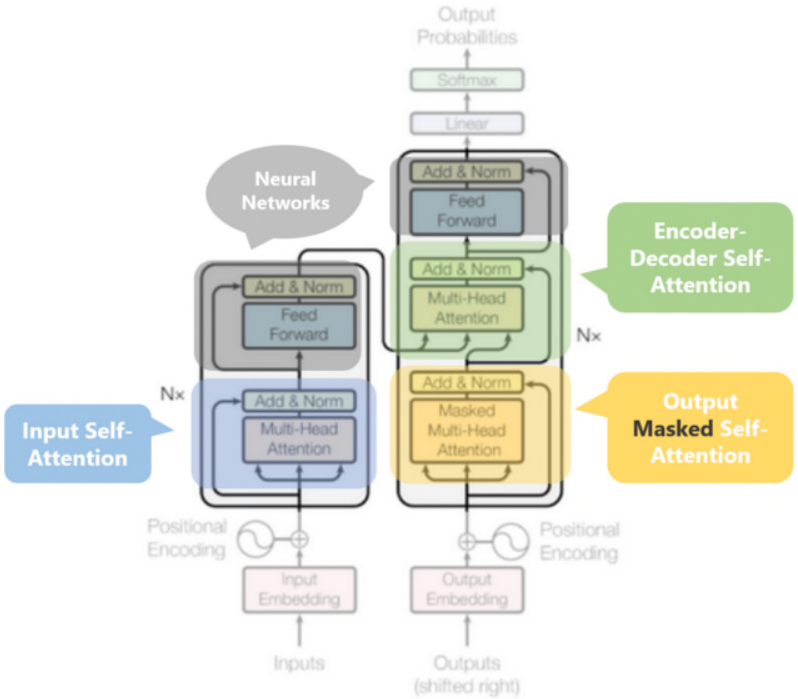
Source: Ria Kulshrestha, Transformers. https://towardsdatascience.com/transformers-89034557de14

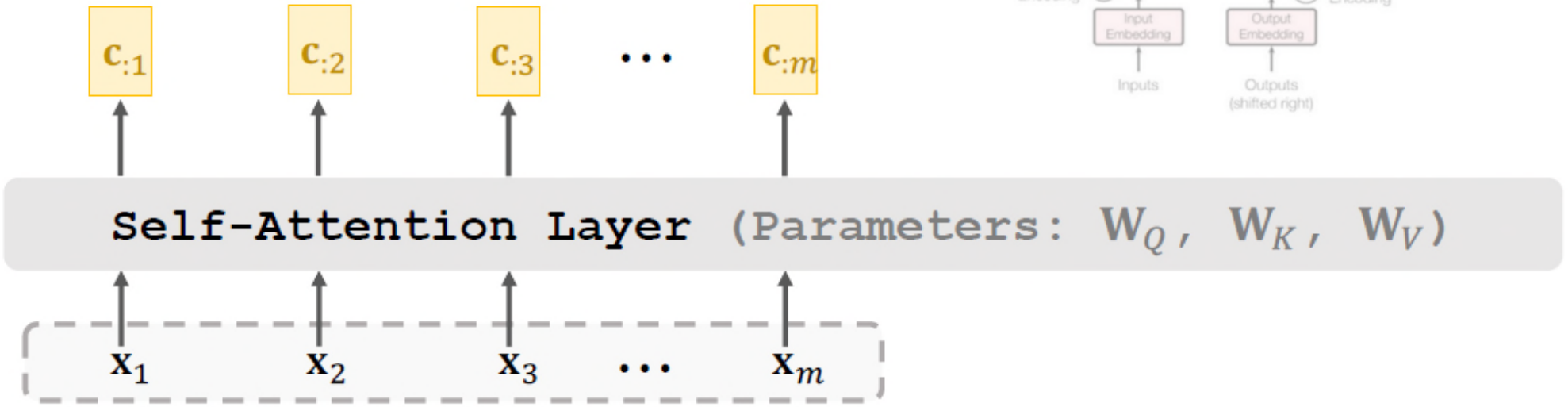
Transformer Architecture (2)



Self-Attention Layer Overview

- Self-attention layer: $\mathbf{C} = \mathrm{Attn}(\mathbf{X}, \mathbf{X})$.
  - Inputs: $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_m]$.
  - Parameters: $\mathbf{W}_Q$, $\mathbf{W}_K$, $\mathbf{W}_V$.



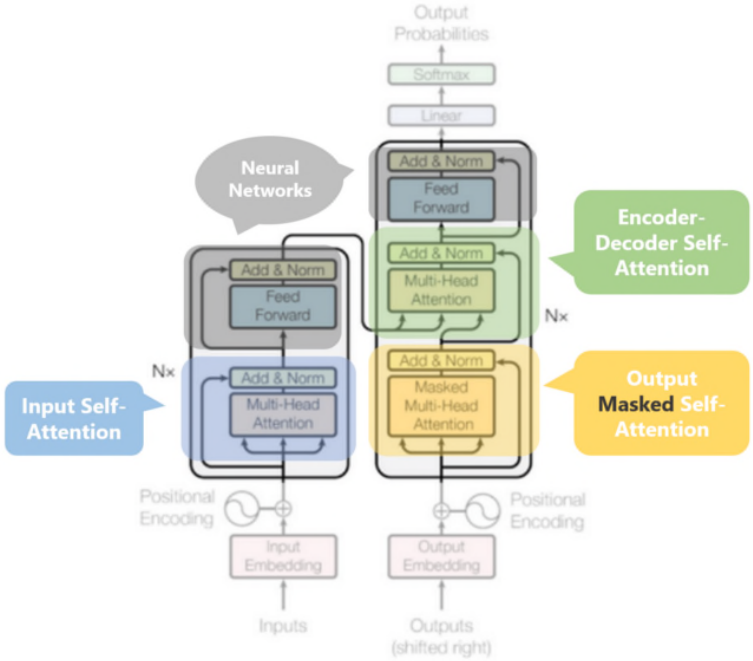


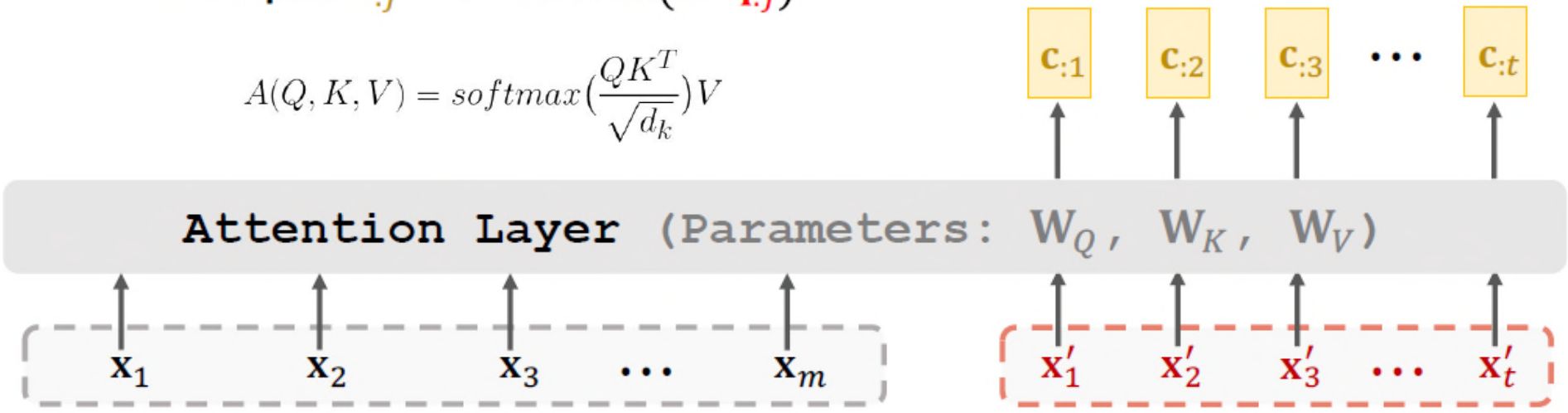
Source: Shusen Wang, Transformer Model

Cross-Attention / Encoder-Decoder Self-Attention

- Attention layer: $\mathbf{C} = \mathrm{Attn}(\mathbf{X}, \mathbf{X}')$.
  - **Query:** $\mathbf{q}_{:j} = \mathbf{W}_Q \mathbf{x}'_j$,
  - **Key:** $\mathbf{k}_{:i} = \mathbf{W}_K \mathbf{x}_i$,
  - **Value:** $\mathbf{v}_{:i} = \mathbf{W}_V \mathbf{x}_i$.
  - **Output:** $\mathbf{c}_{:j} = \mathbf{V} \cdot \mathrm{Softmax}(\mathbf{K}^T \mathbf{q}_{:j})$.

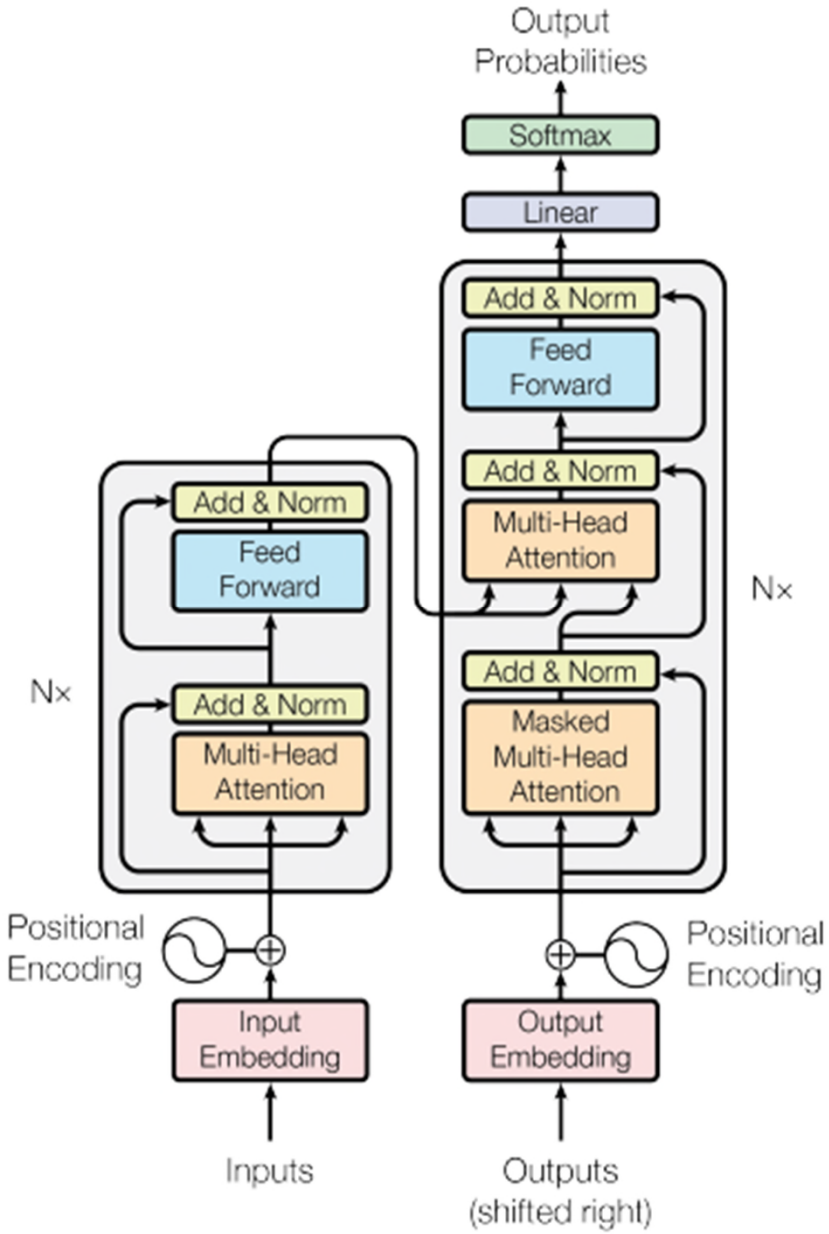
$$A(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$



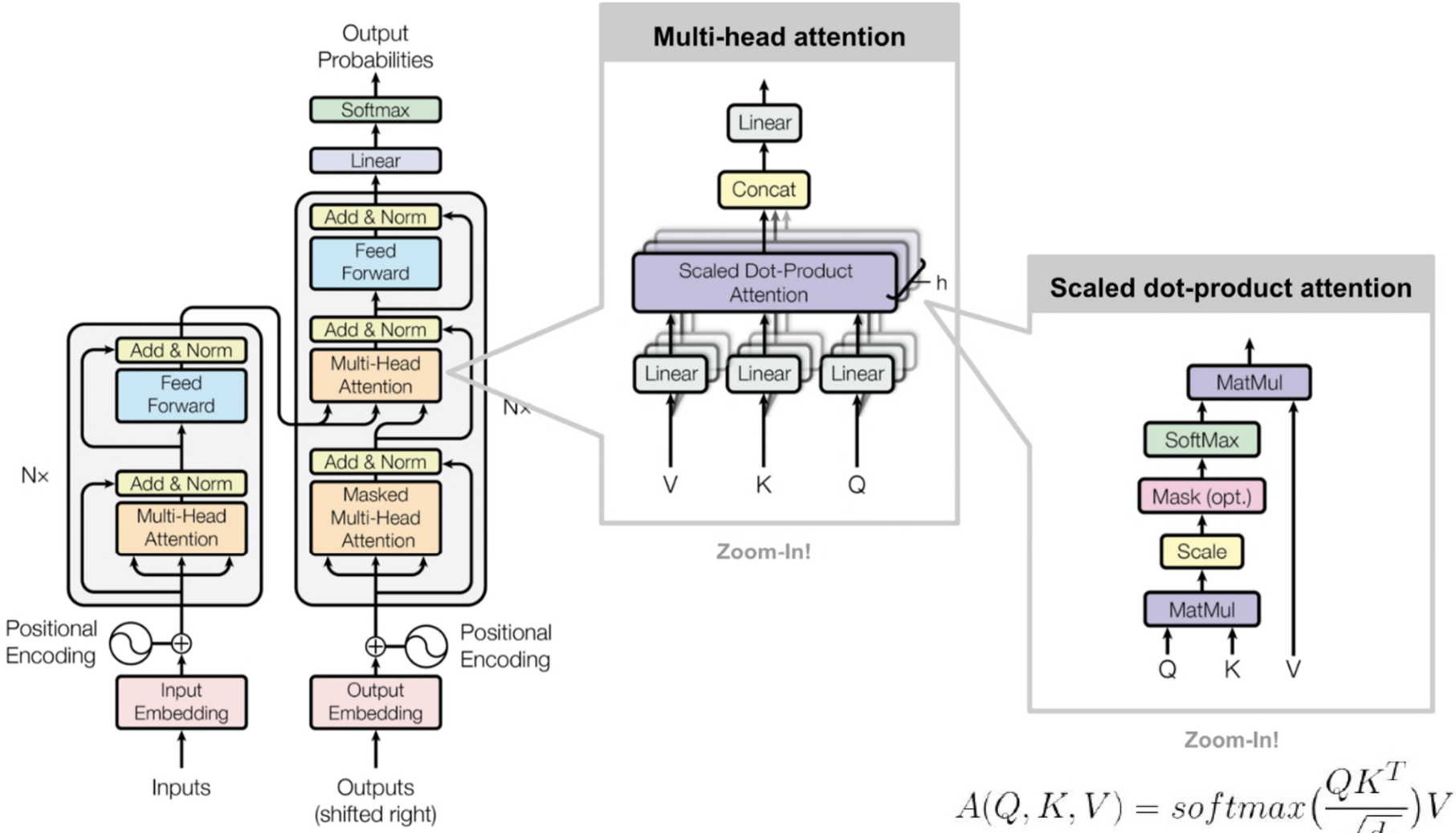


More Details

- Multi-Head Attention
- Feed Forward Network
- Positional Encoding
- Residual Connection
- Layer Normalization (Norm)
- Input/Output Embedding
- Masked Attention



Multi-Head Attention

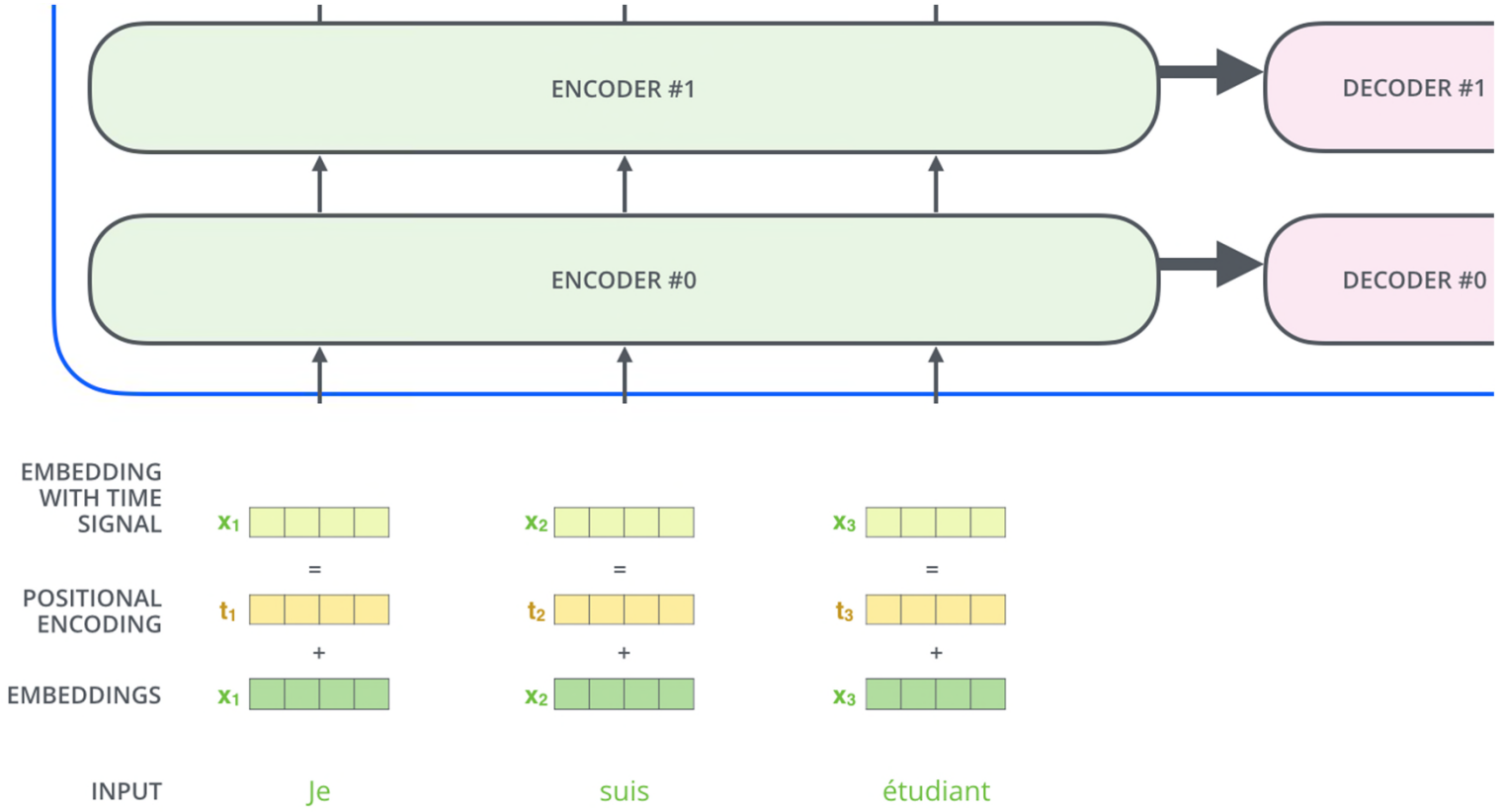


$$A(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

Source: https://deepfrench.gitlab.io/deep-learning-project/

Position Encoding

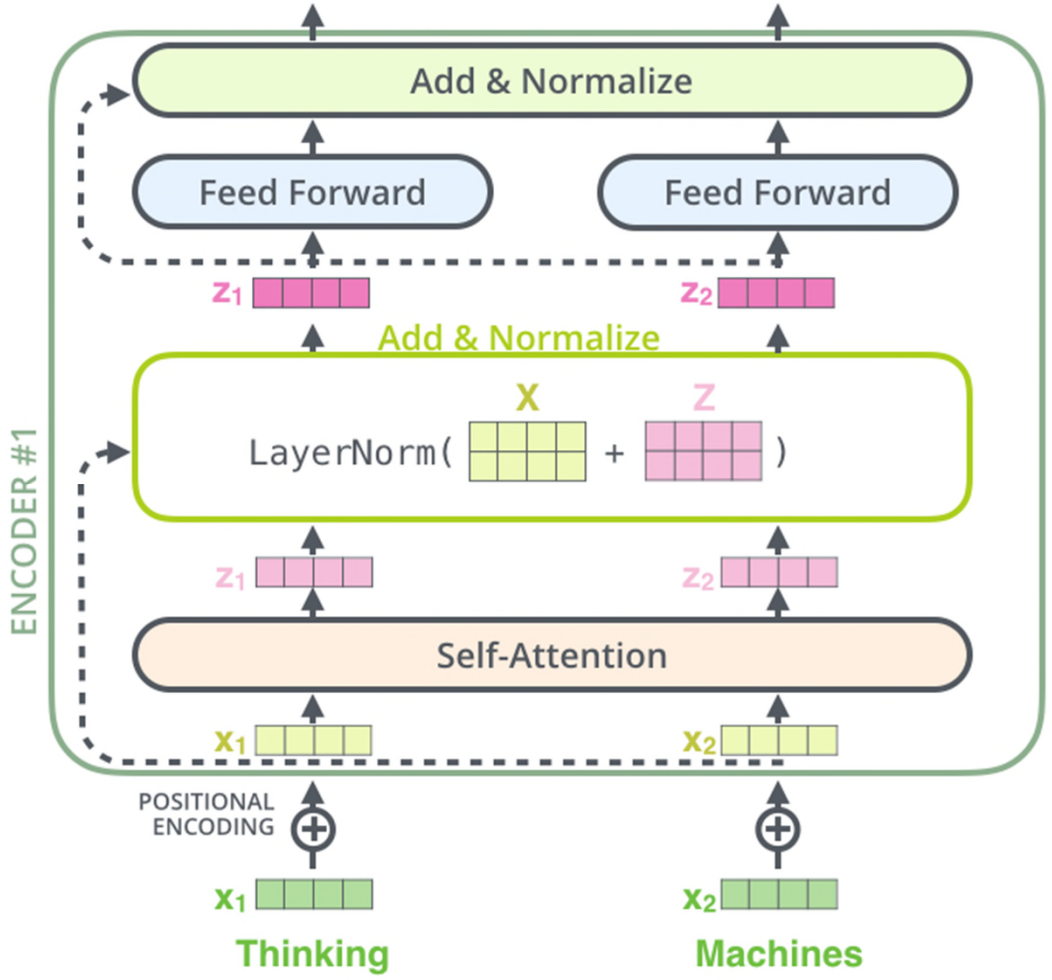
- Represent the position information of individual input tokens.
- Position encoding using sin / cos functions are often used.



Source: https://jalammar.github.io/illustrated-transformer/

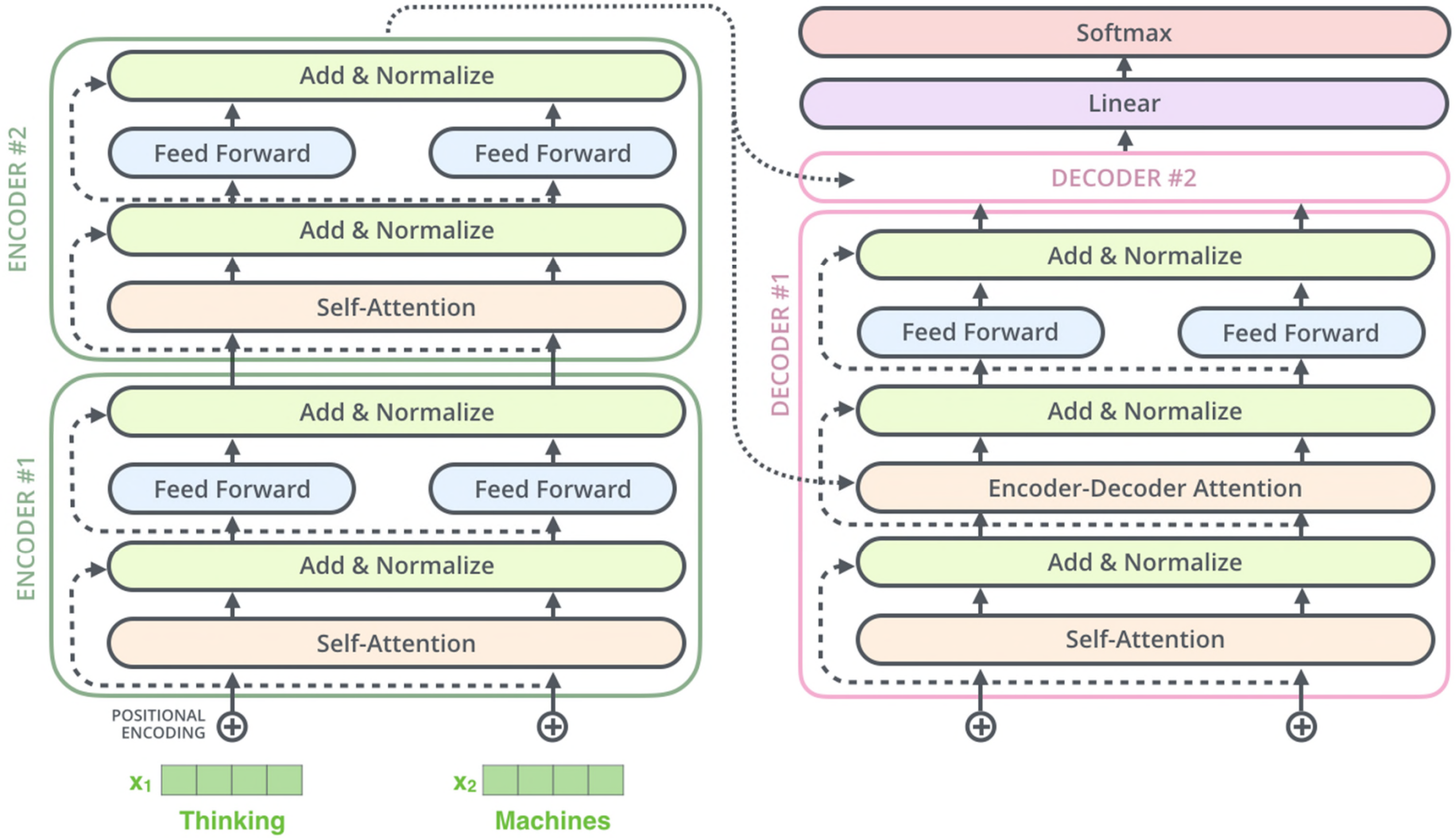
Residual Connection & Layer Norm

- Residual connection leverages the idea of residual learning in Resnet.
- Layer Norm is used to perform normalization.



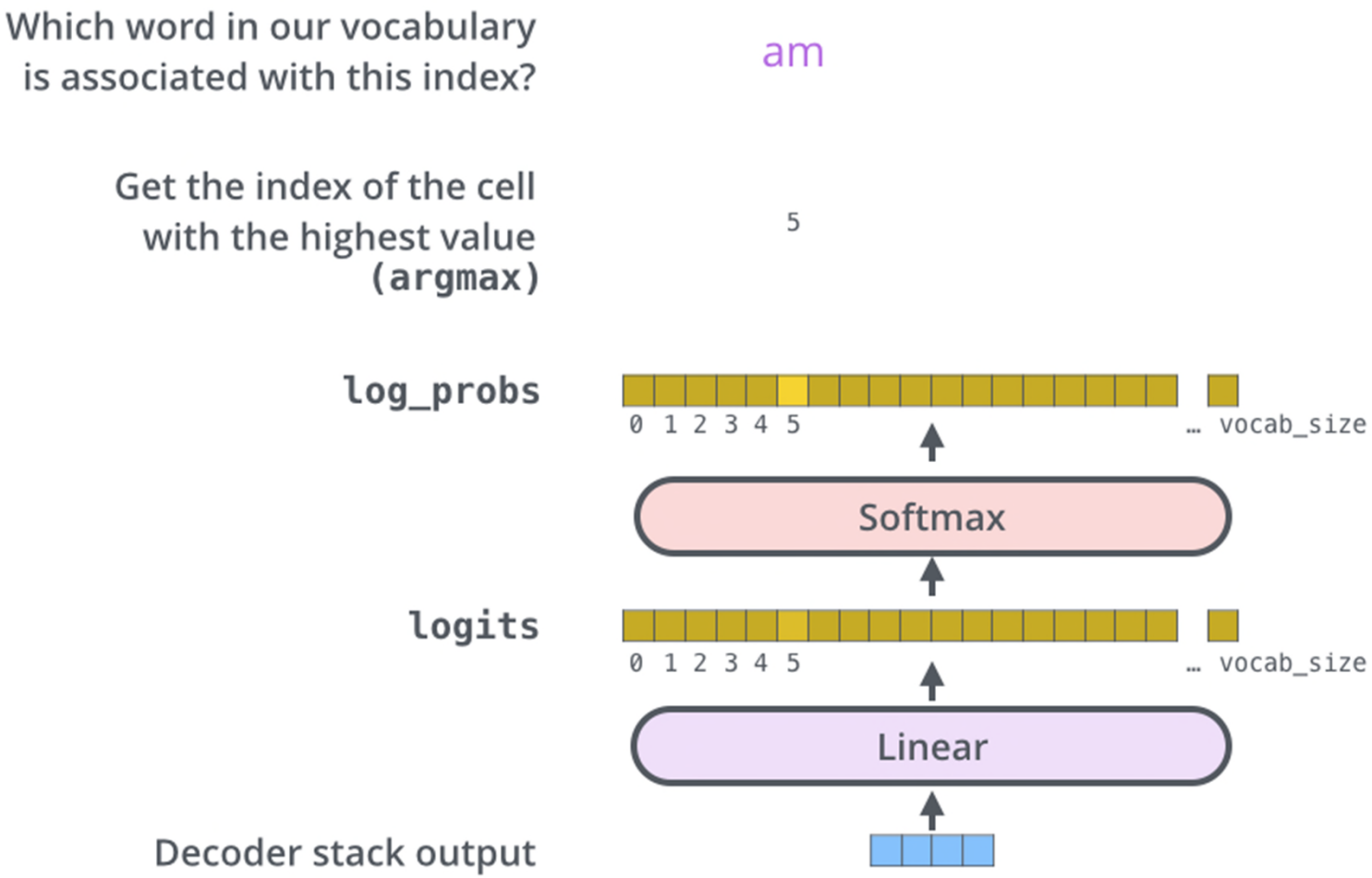
Source: https://jalammar.github.io/illustrated-transformer/

Transformer Encoder + Decoder Architecture



Source: https://jalammar.github.io/illustrated-transformer/

Final Linear and Softmax Layer



Source: https://jalammar.github.io/illustrated-transformer/

LSTM vs Transformer

	LSTM	Transformer
Pros	• Work reasonably well for long sequences	• Excellent at long sequences as attention looks at all inputs. • Parallel computation.
Cons	• Sequential computation	• Require large memory and training data

Vision Transformer (1)

- A SOTA image classification model based on attention.

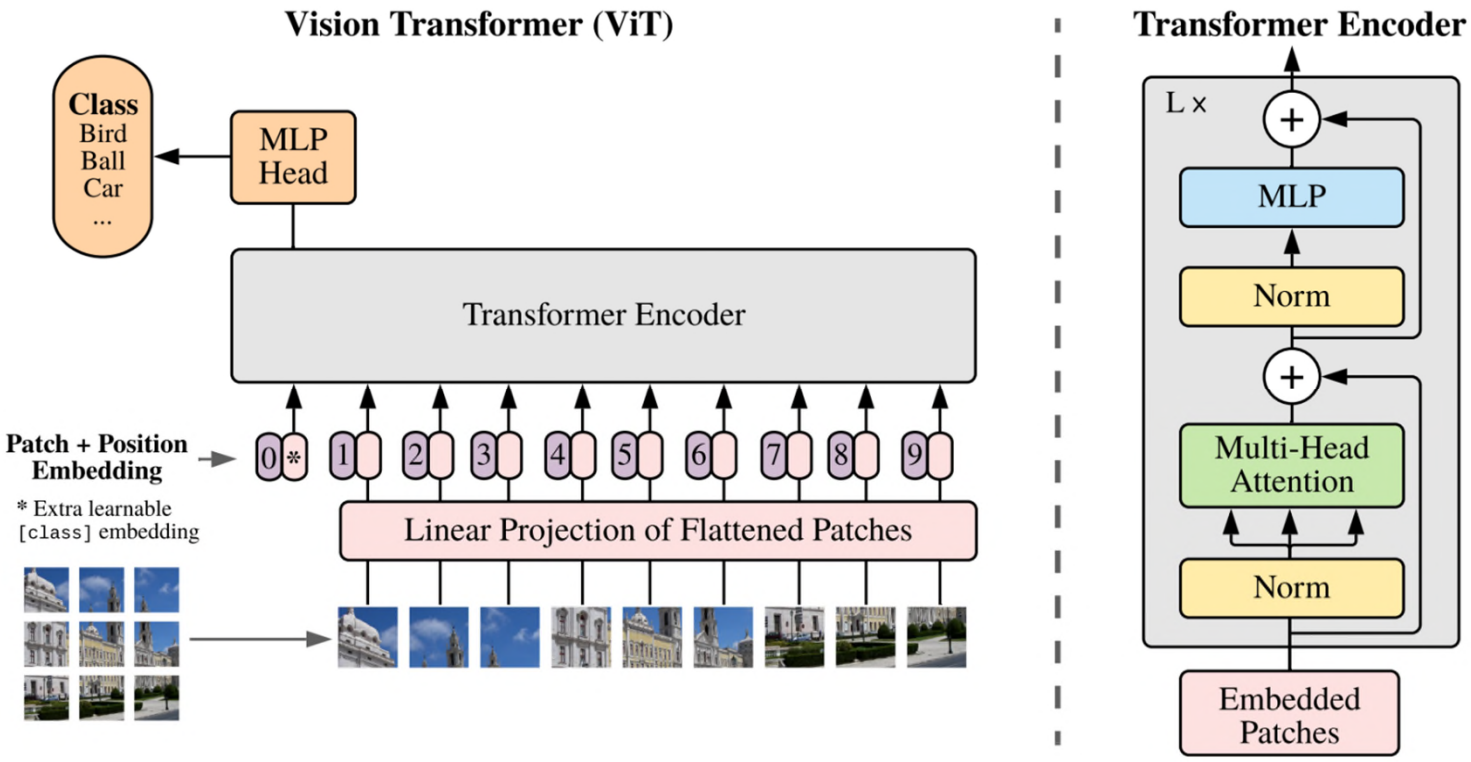


Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable "classification token" to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).

An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale (2020)
Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, Neil Houlsby

Vision Transformer (2)

- Key steps of Vision Transformer (ViT):
  - Partition an image into patches / tokens.
  - Flatten the patches / tokens using lexicographical ordering.
  - Generate linear embeddings from the flattened patches / tokens.
  - Introduce an extra learnable class embedding.
  - Add positional embeddings / encoding.
  - Pass the tokens to a transformer encoder.
  - Encoder output of the extra learnable token will pass through a MLP head network for classification.
  - Training will involve using a pretrain model, and finetune it using the target dataset for image classification.

Exercise: Model Comparison

(b) A user would like to develop an image classification application using a model that can achieve good accuracy and uses attention mechanism when performing classification. He is considering the following 3 candidate models: (i) VGG, (ii) Vision Transformer (ViT), and (iii) Long Short-Term Memory (LSTM). State which model is most likely going to meet the user's need and briefly justify your answer.

(5 Marks)

Solution

Section III Summary

- The section covers the following topics:
  - Attention Concept
  - Transformer Architecture
  - Vision Transformer

Part 4 Summary

- This part covers the following:
  - Convolutional Neural Network (CNN)
  - Recurrent Neural Network (RNN)
  - Long Short-Term Memory (LSTM)
  - Transformer