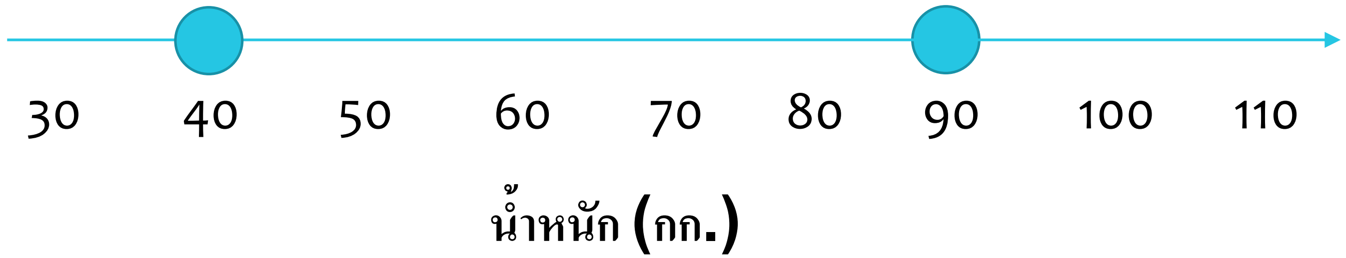
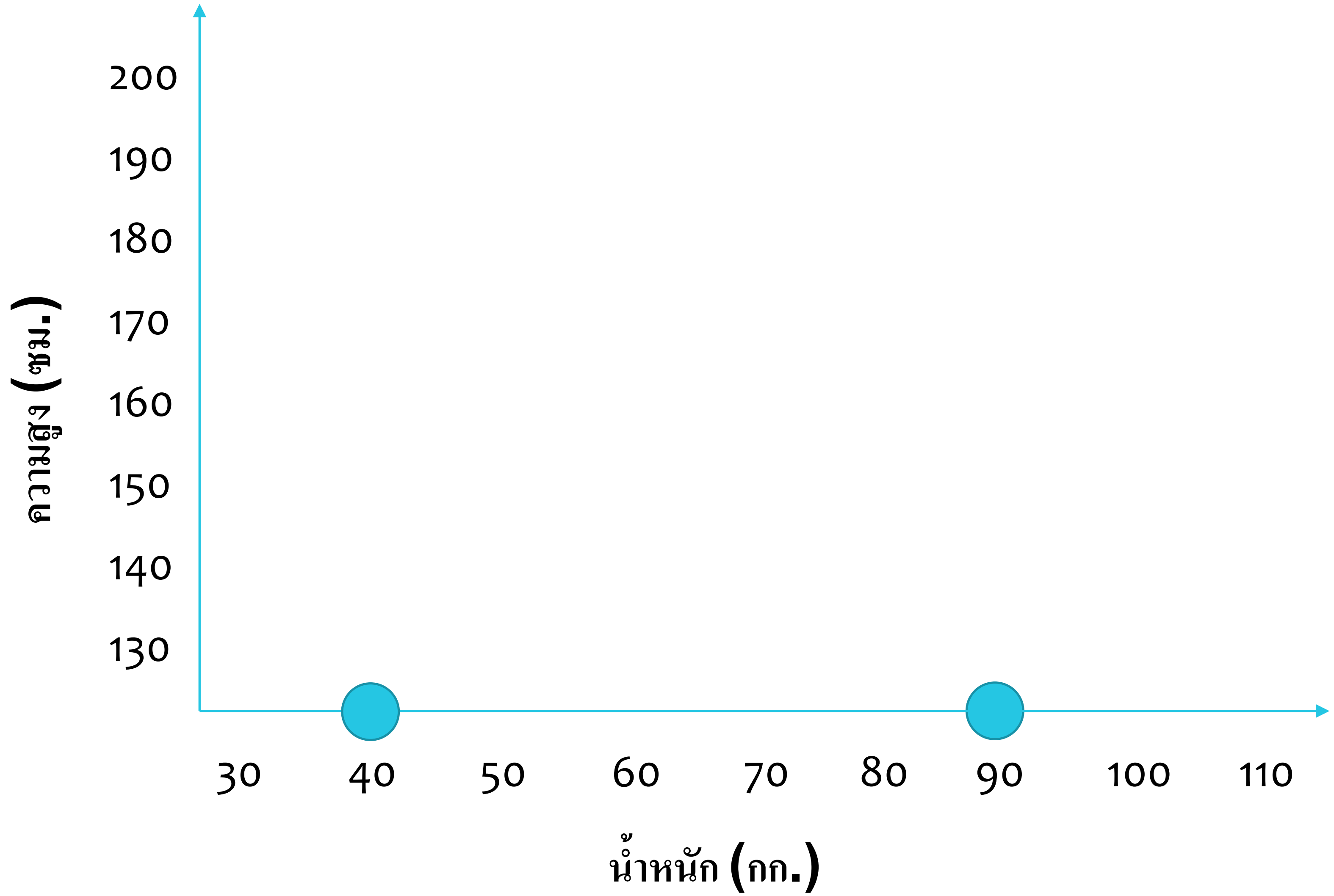
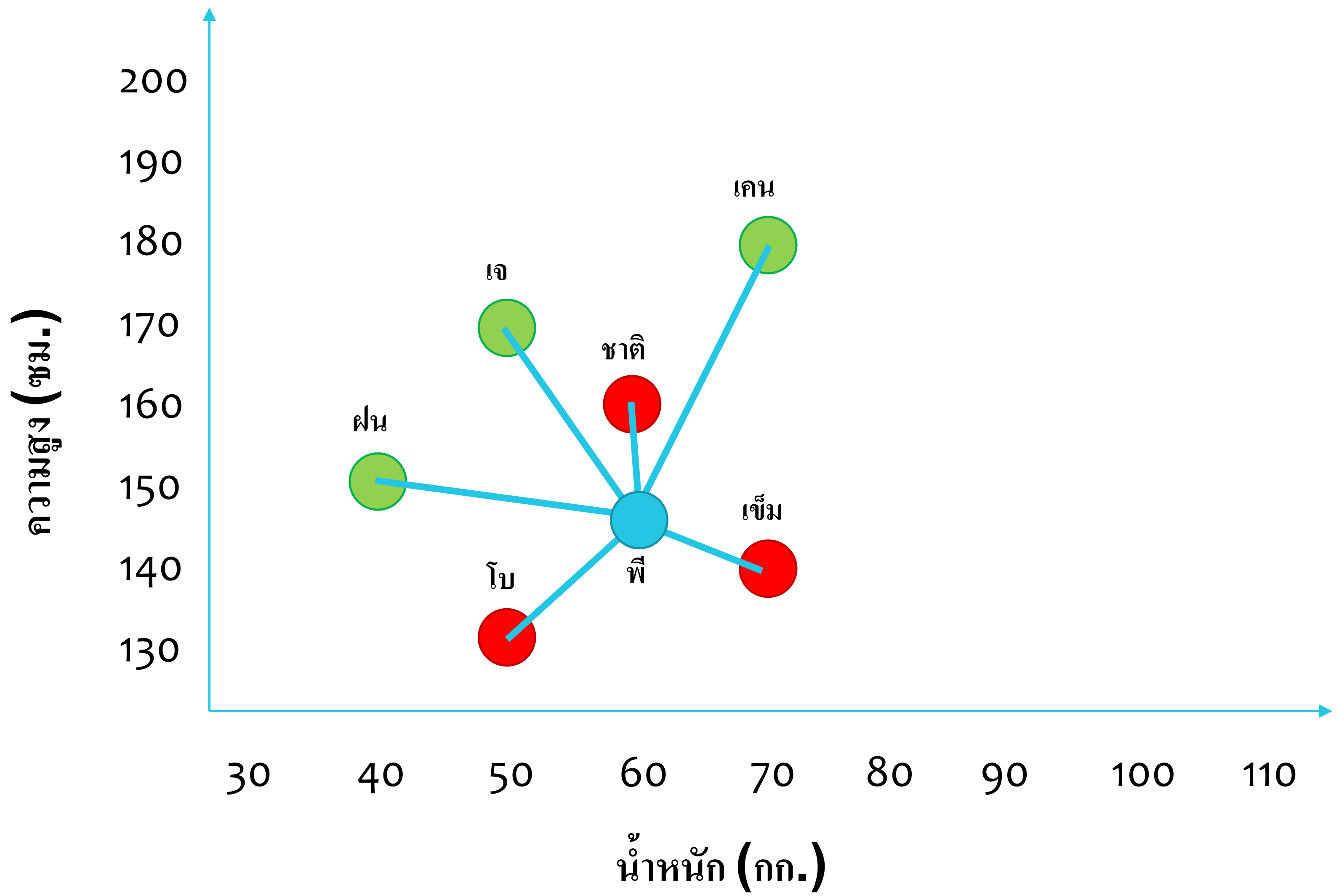
Deep Learning

Parinya Sanguansat

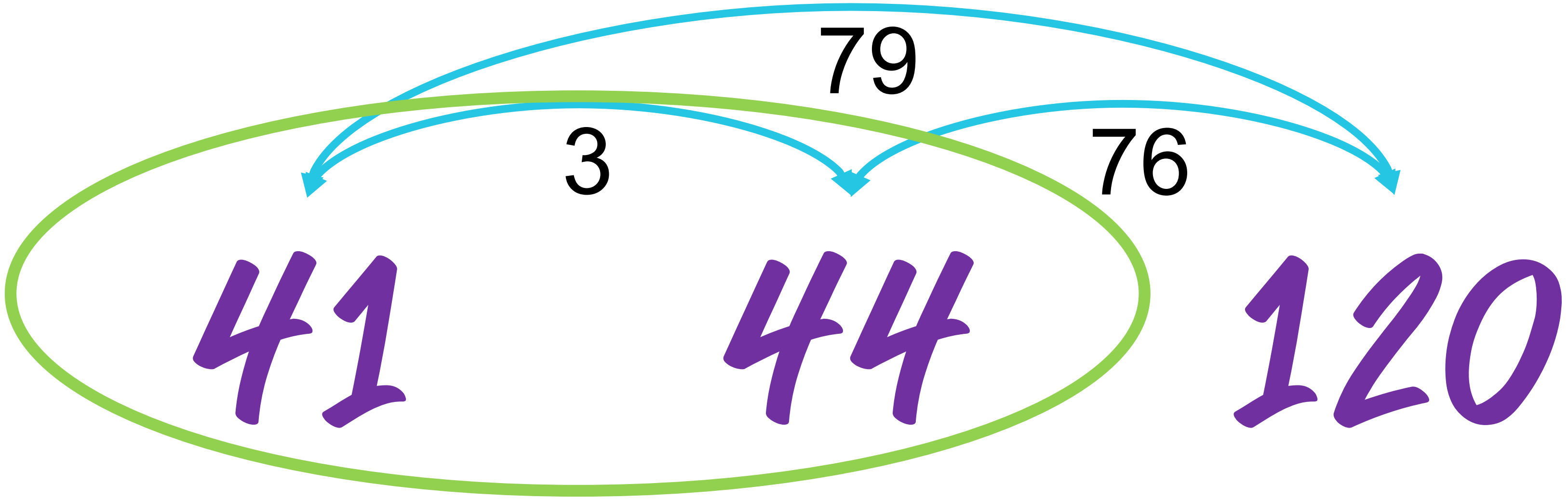
Who is normal?


30
40
50
60
70
80
90
100
110
น้ำหนัก (กก.)


200
190
180
170
160
150
140
130
ความสูง (ซม.)
30
40
50
60
70
80
90
100
110
น้ำหนัก (กก.)


ความสูง (ซม.)
200
190
180
170
160
150
140
130
เคน
เจ
ชาติ
ฝน
เข็ม
โบ
พี
30
40
50
60
70
80
90
100
110
น้ำหนัก (กก.)

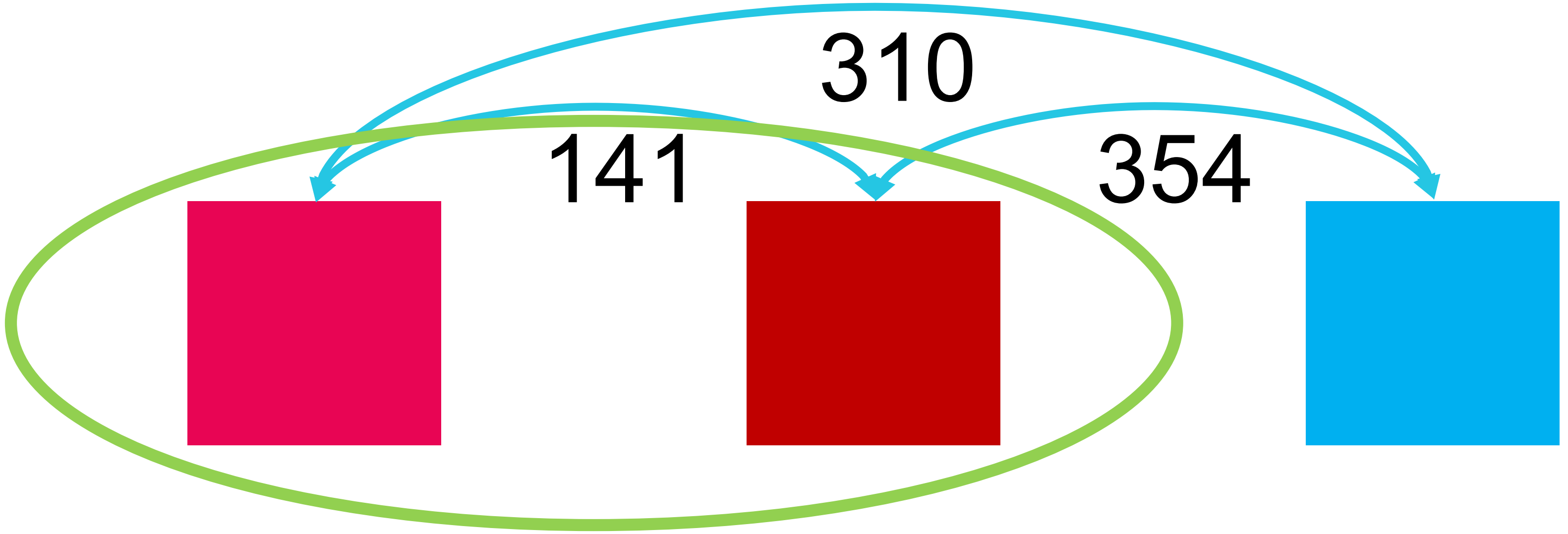
Similarity



Similarity

$$\begin{bmatrix} 237 \\ 125 \\ 46 \end{bmatrix} \qquad \begin{bmatrix} 192 \\ 0 \\ 0 \end{bmatrix} \qquad \begin{bmatrix} 0 \\ 176 \\ 240 \end{bmatrix}$$

Similarity



$$\begin{bmatrix}237\\125\\46\end{bmatrix} \qquad \begin{bmatrix}192\\0\\0\end{bmatrix} \qquad \begin{bmatrix}0\\176\\240\end{bmatrix}$$

Iris dataset

Virginica

Versicolor

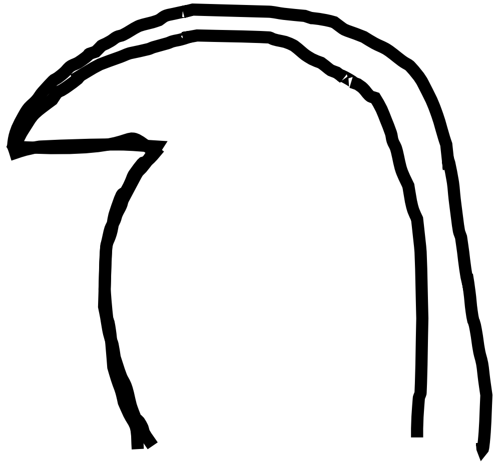
Setosa

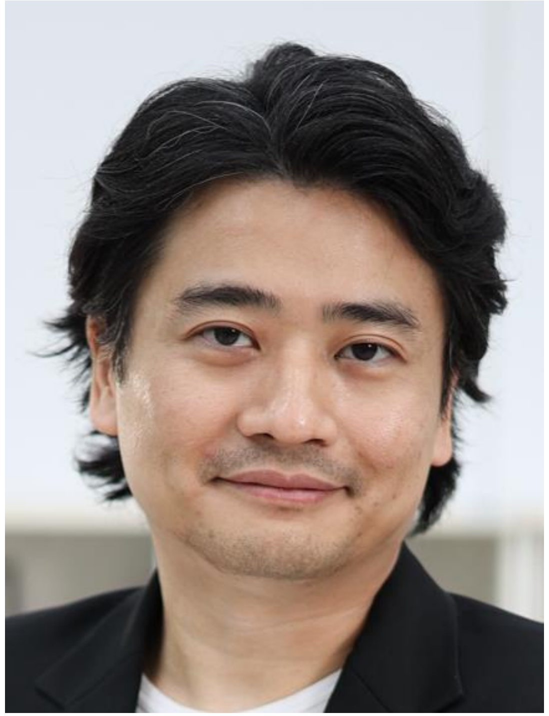
Observation

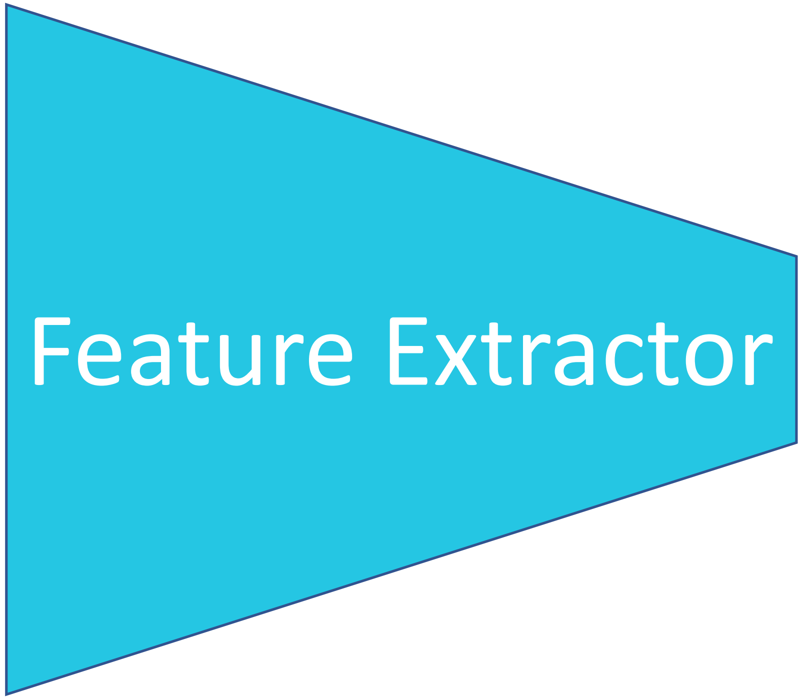
- Extract information

Distance of image?

Distance of image?

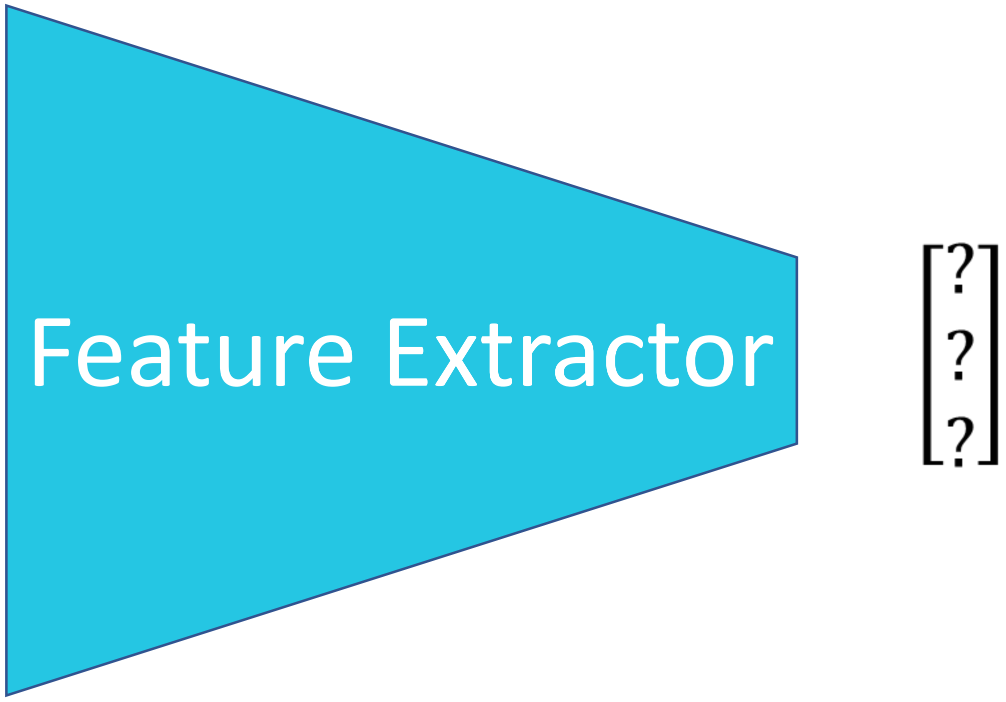




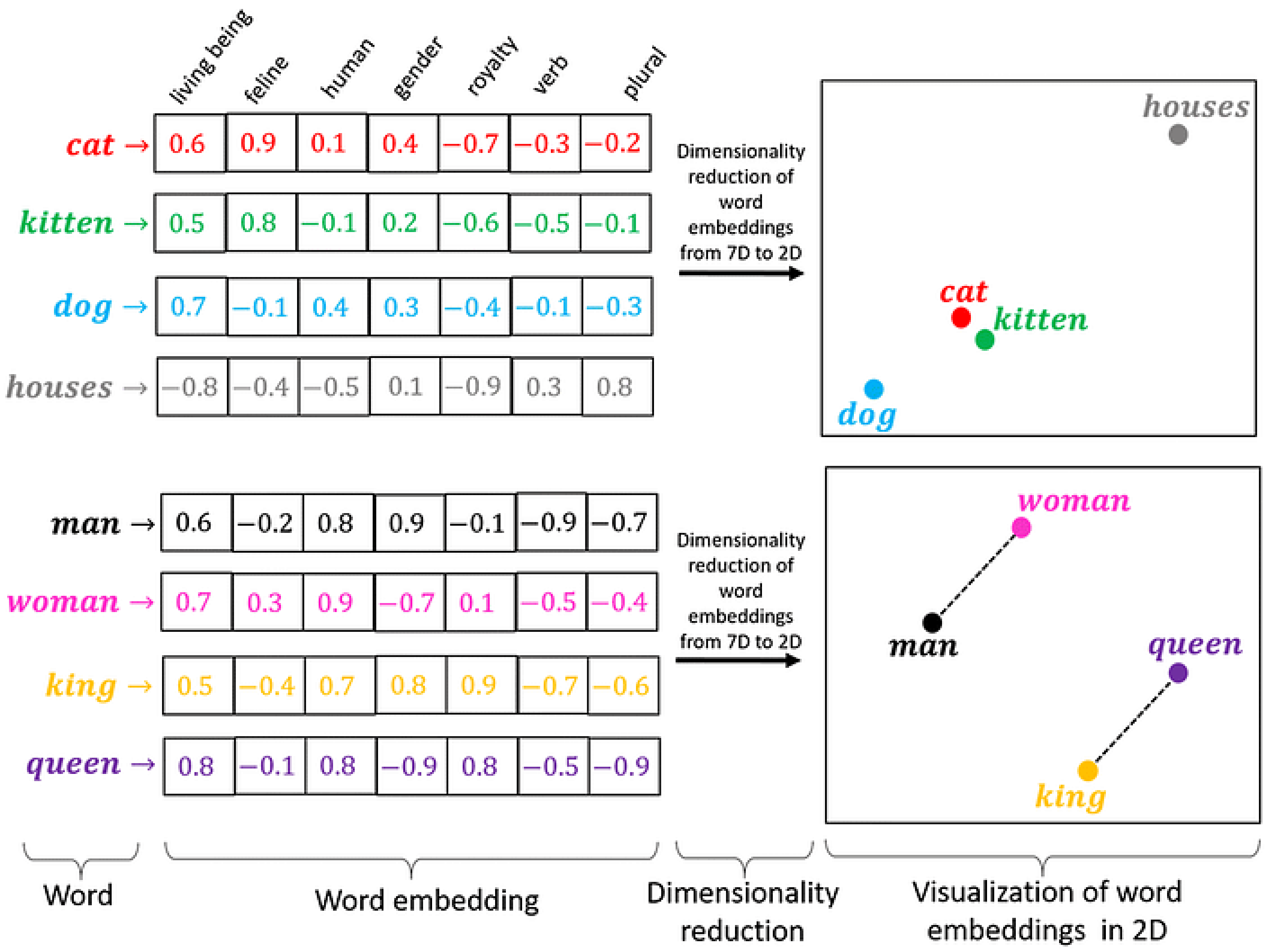


$$\begin{bmatrix} ? \\ ? \\ ? \end{bmatrix}$$

- Image
- Audio
- Video
- Text
- etc.

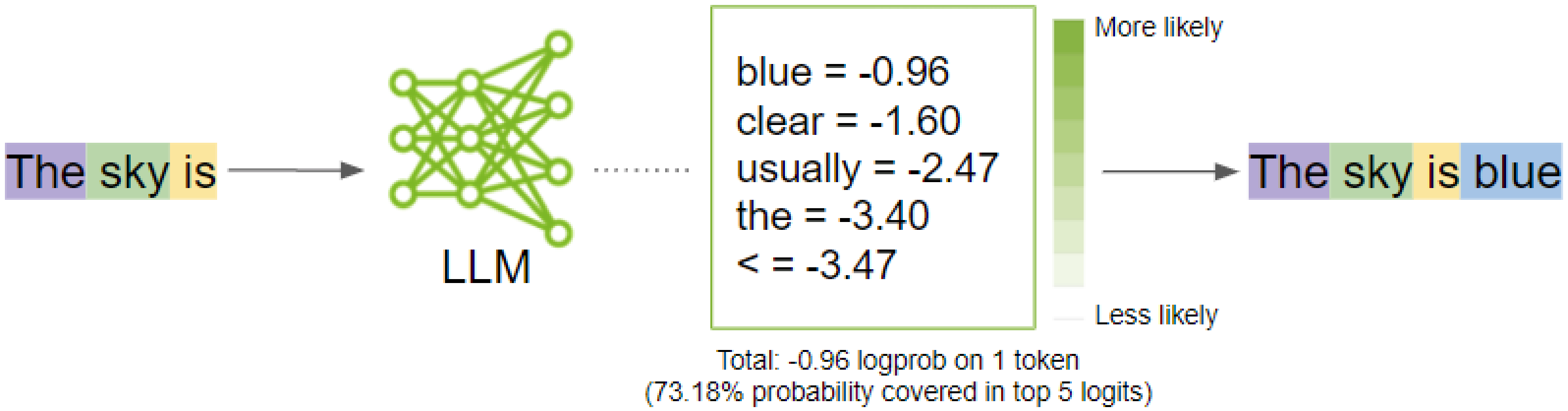


Word Embedding



Language Model

- Predict next token (word)



Encoder

- Transforms input data into a different (often lower-dimensional) representation.
  - This representation is called an embedding or a latent representation.

- Ex. Draw digit “1” in 10x10, 3x3, 3x1

Modality Representation

- Text:
  - Word embeddings (Word2Vec, GloVe, BERT), sentence embeddings.
- Images:
  - CNN features (ResNet, EfficientNet), Transformers (ViT).
- Audio:
  - Spectrograms, MFCCs, audio embeddings.
- Video:
  - 3D CNNs, temporal segment networks, video transformers.
- Sensor data:
  - time series representation.

Neural Network

Parinya Sanguansat

Timeline

Nobel prize 2024

Turing Award 2018




2022: ChatGPT
Stable
Diffusion

1958: Perceptron

1982: Hopfield
Network

1986: Boltzmann
Machines
BP MLP

1989: CNN

2014: GAN

2015: AlphaGO
YOLO

2021: DALL-E

1950

2024

1979: Neocognitron

1986: ID3

1992: SVM

2012: AlexNet

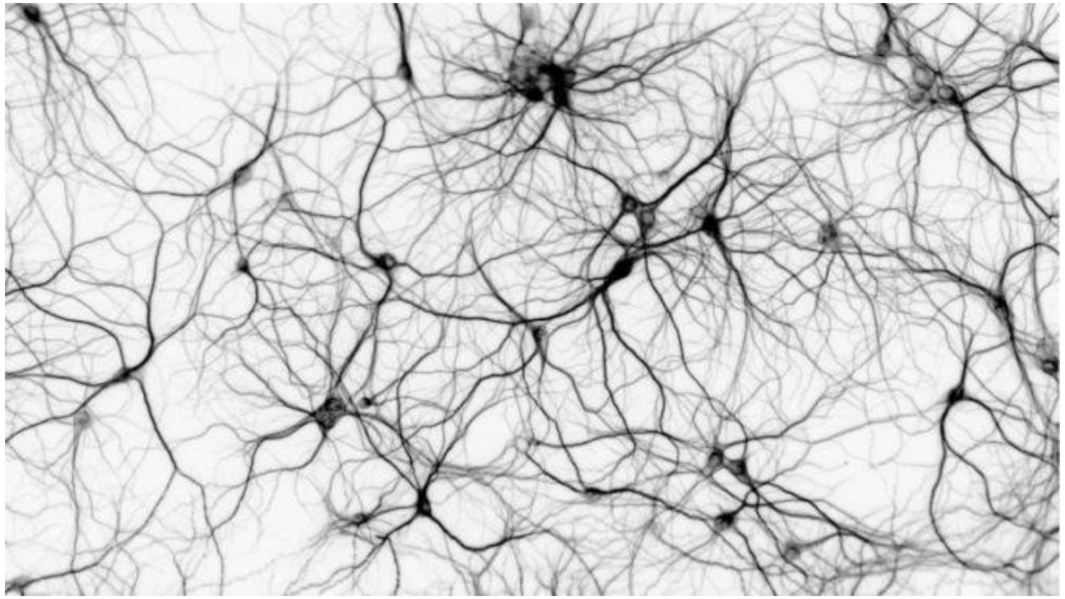
2018: BERT, GPT-1

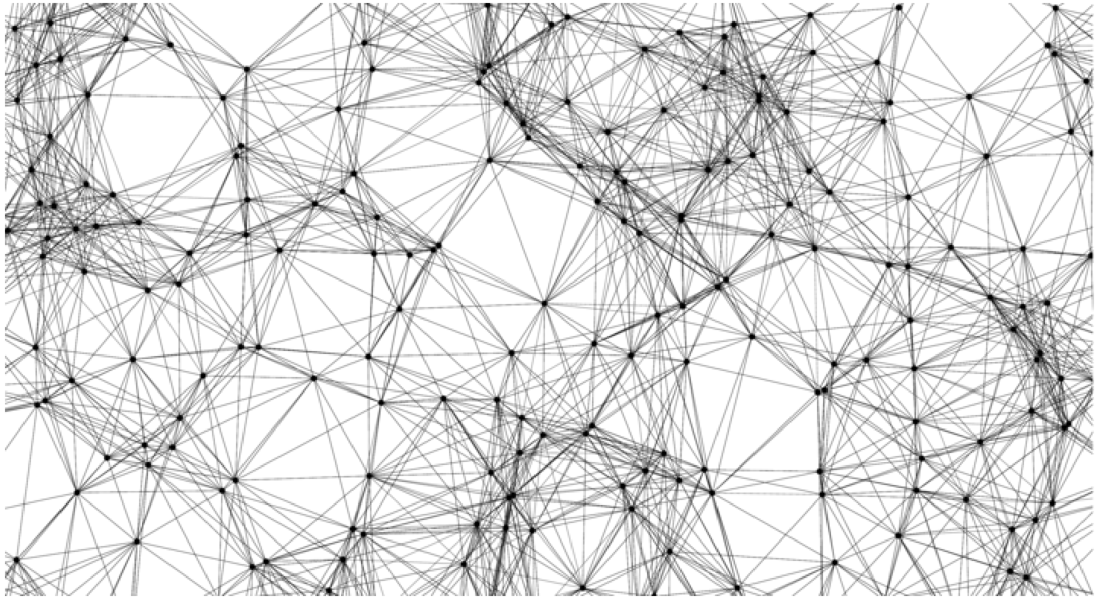
2017: Transformer




Artificial Neural Networks (ANN)

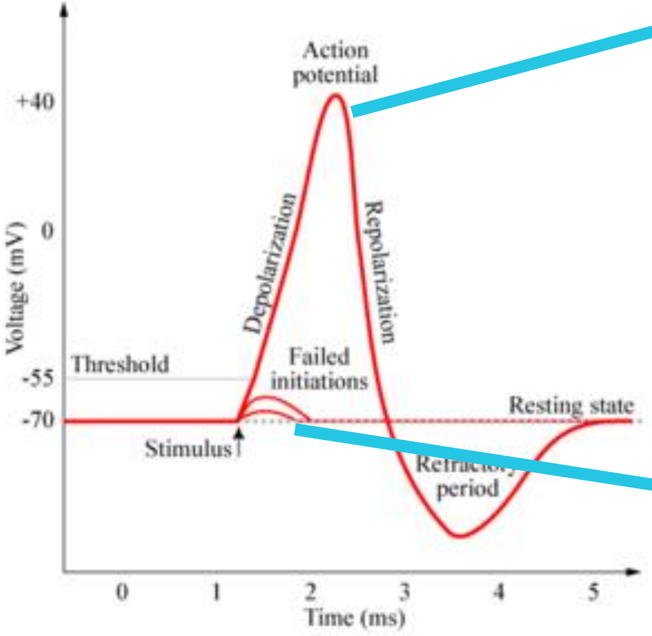
A neural network is an interconnection of neurons.

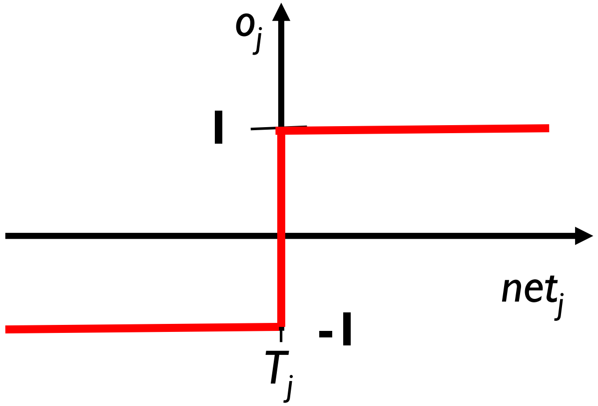




How neural net work?

- Activation

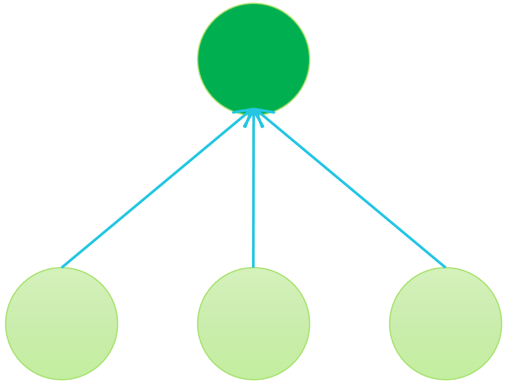




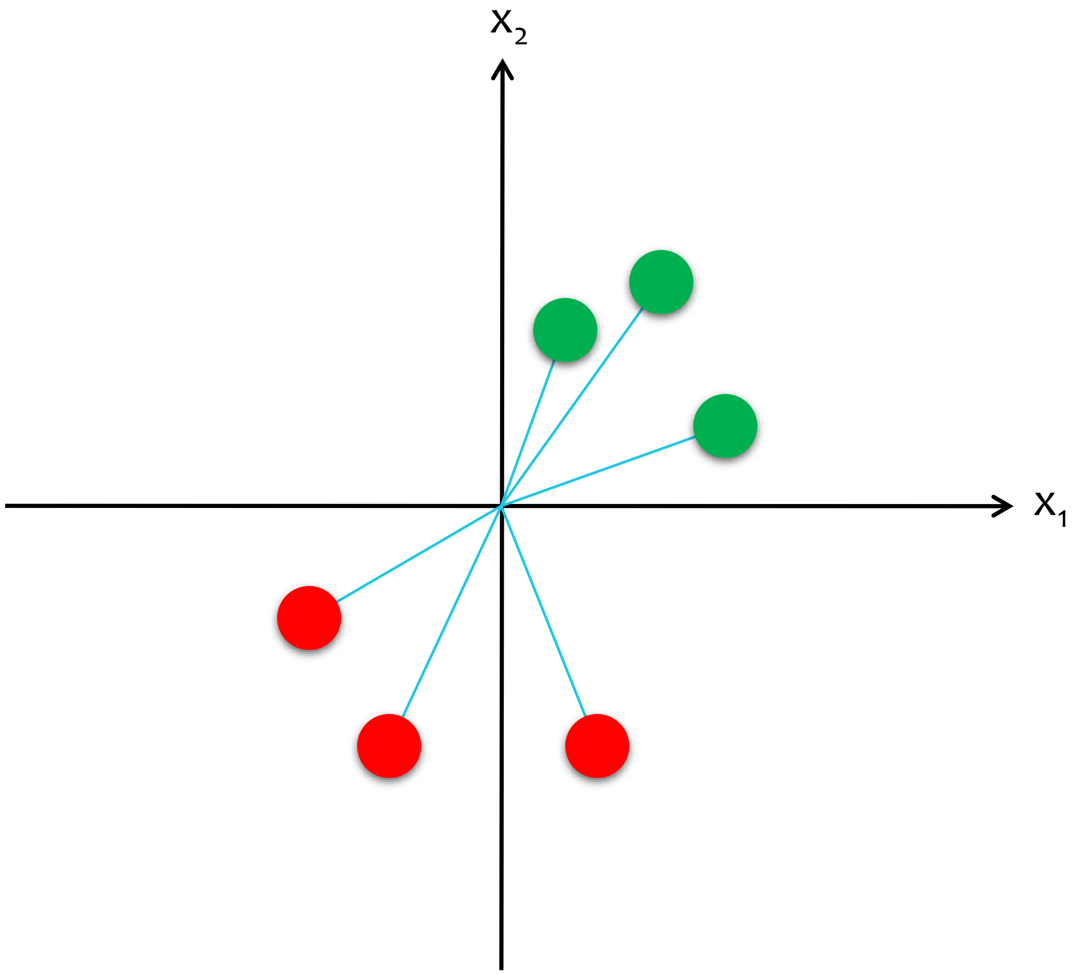
Perceptron

- Perceptron is the simplest algorithm of neural network
  - Supervised learning
  - Binary classifiers

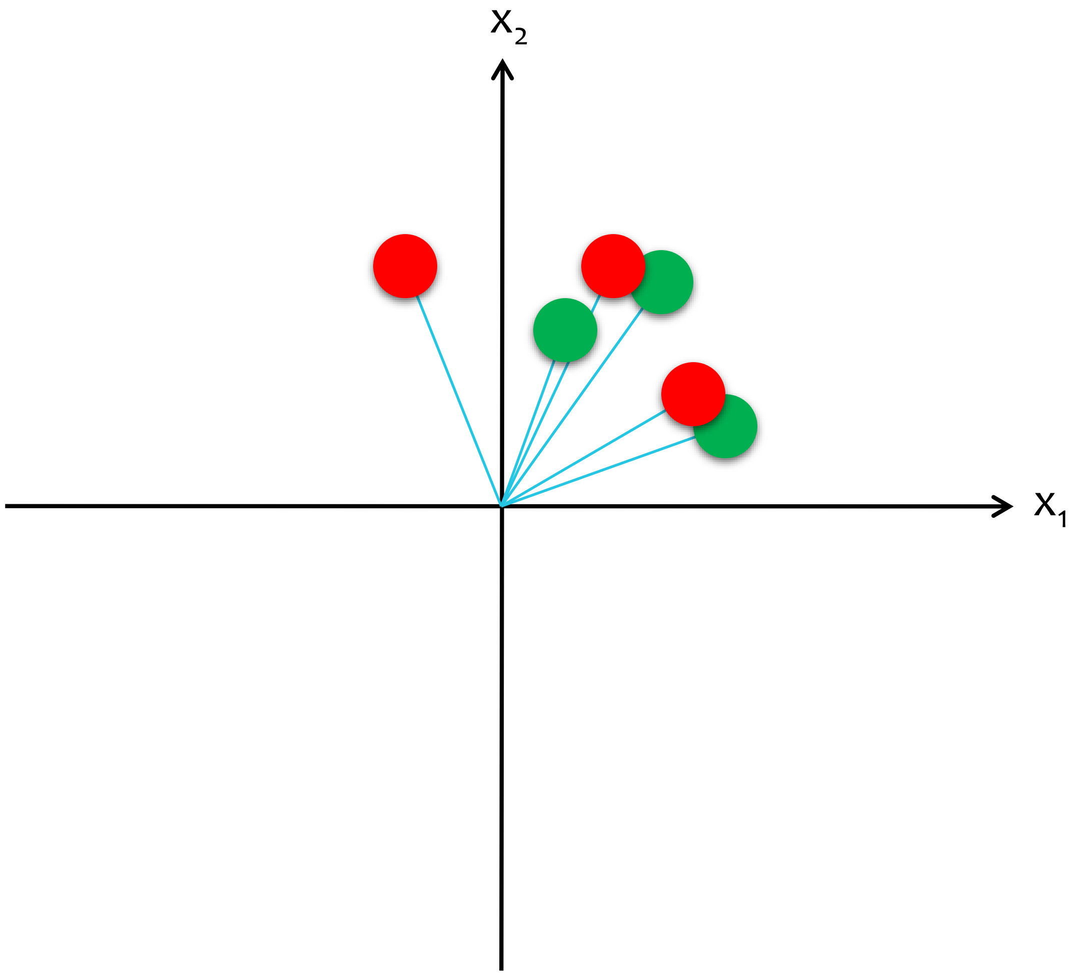
$$\mathbf{O} = f\left(\mathbf{X}_{N\times D}\mathbf{W}_{D\times 1} + b\right)$$



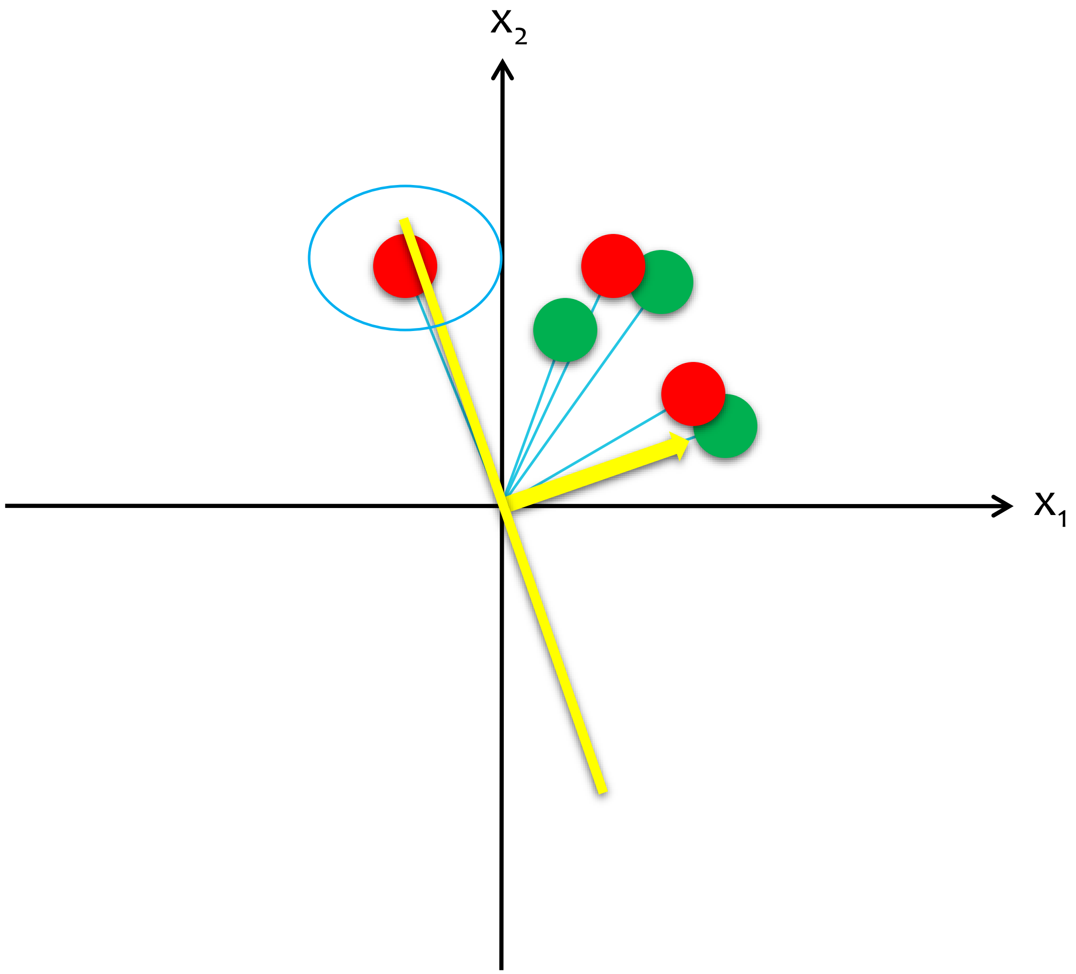
Perceptron: Step 0



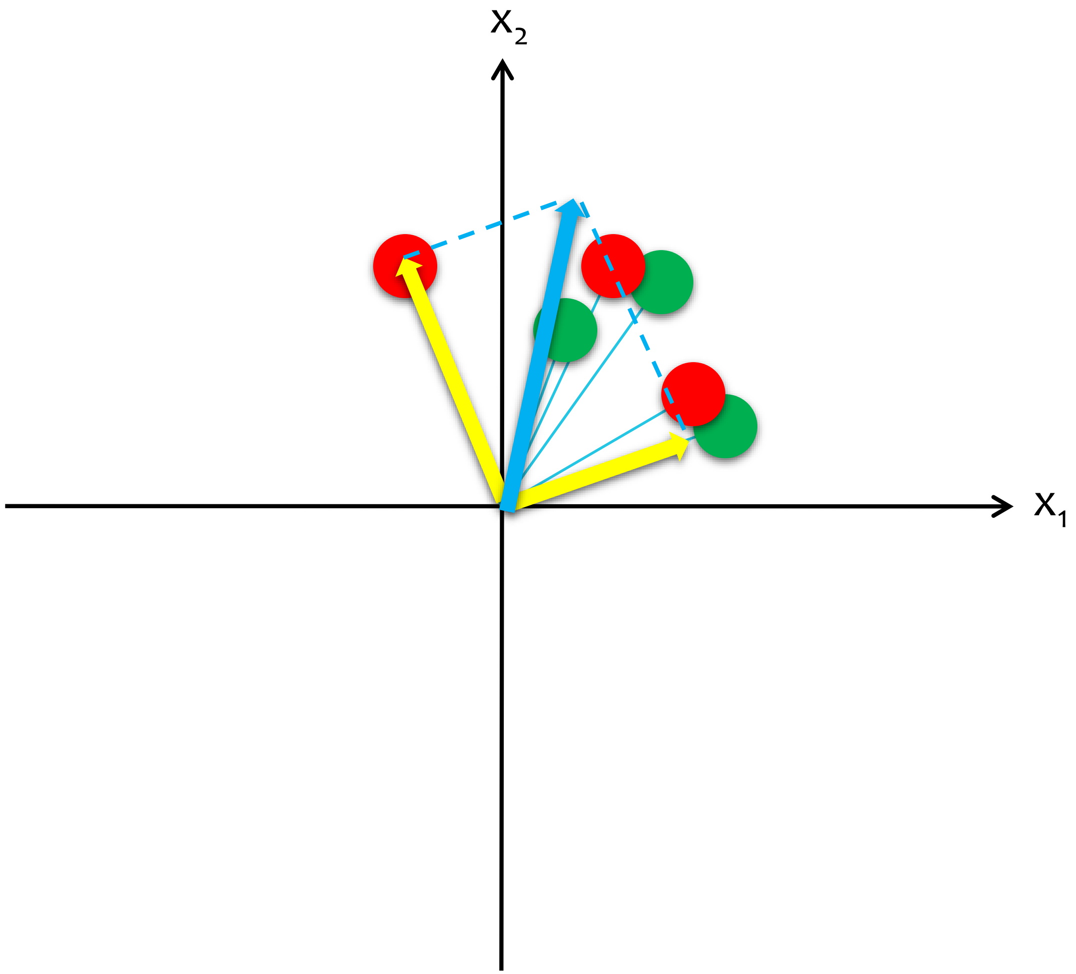
Perceptron: Step 1



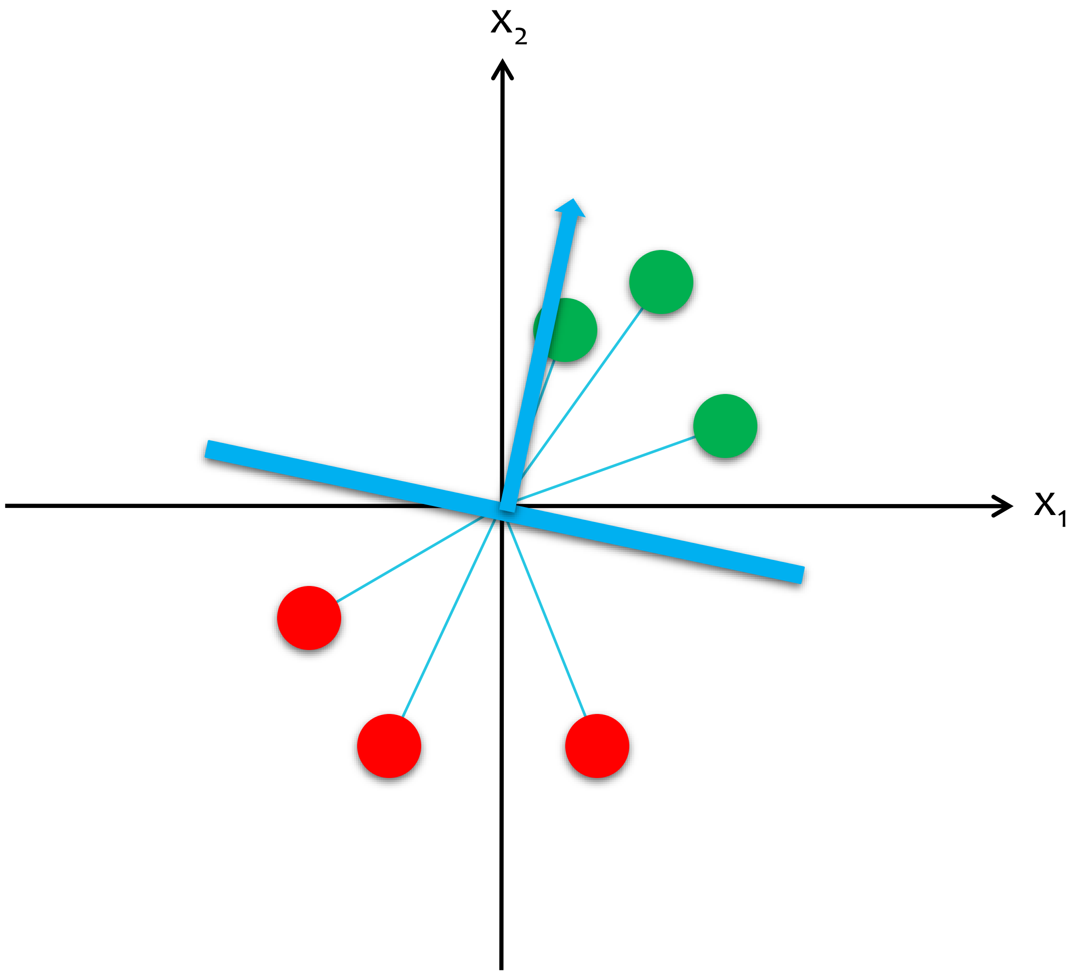
Perceptron: Step 2



Perceptron: Step 3

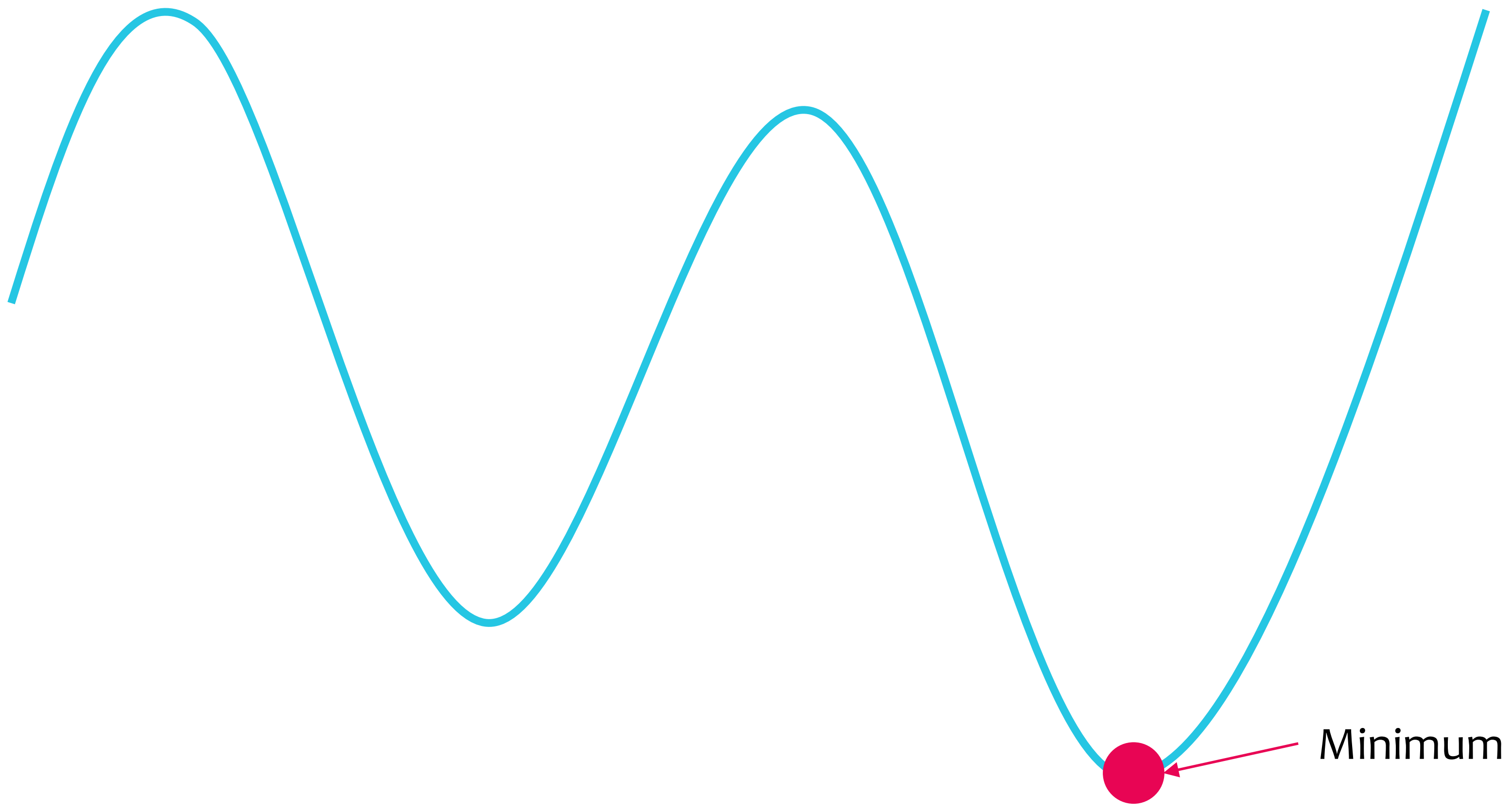


Perceptron: Step 4



How to find the optimal weights?

- If we can observe every points in the error function
  - We can find the minimum very easy



How to find the optimal weights?

- But we cannot observe that because we cannot collect all data

Mean Square Error (MSE)

- Loss = how to measure the fitness

$$E(\mathbf{W}) = \frac{1}{N}\sum_{i=1}^{N}\left(o_i^{true} - o_i^{predict}\right)^2$$

Stochastic Gradient Descent (SGD)

- After we have somewhere to start we need the direction
- Optimizer = how to update weights
  - Update in the opposite direction to gradient
- 1 epoch = all training samples
  - Gradient Descent (GD) use all training samples for each update
    - GD update weight only once in an epoch
  - SGD use mini-batch of training samples for each update
    - SGD update weight many times (up to batch size) in an epoch

$$\mathbf{W}' = \mathbf{W} - \eta \nabla E(\mathbf{W})$$

Learning rate = how much to believe in this direction

How to find the optimal weights?

- But we need somewhere to start
  - Initial by random values

Comments on Perceptron

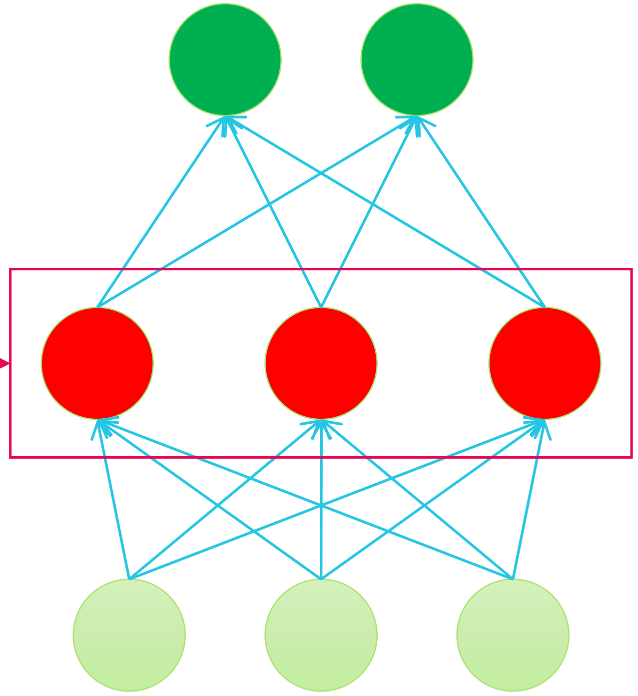
- Cannot solve nonlinear problem

- Nonlinear problem
  - No line can separate all data correctly
  - Loss will not converge
  - Real-world problems are often nonlinear

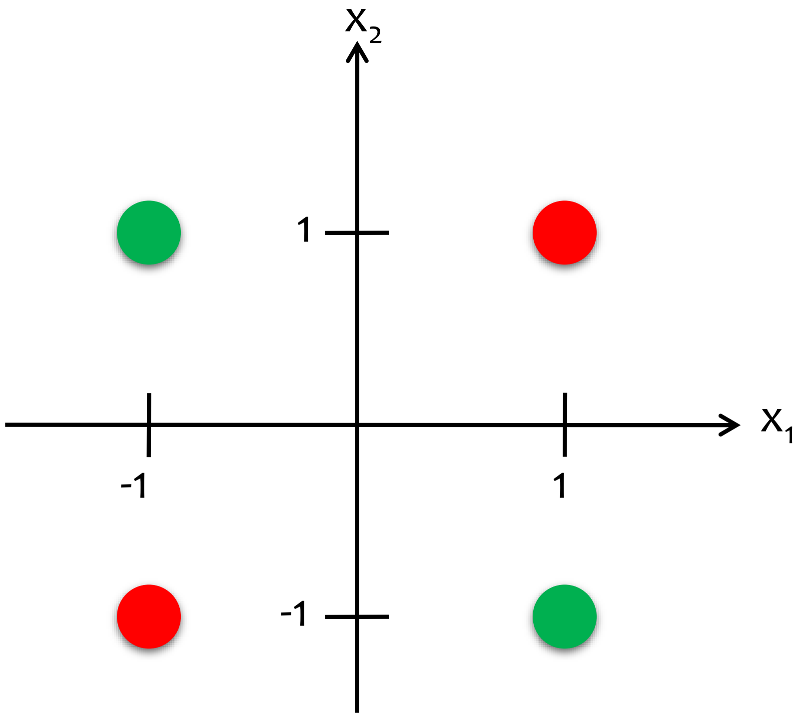
Multi-Layer Perceptron (MLP)

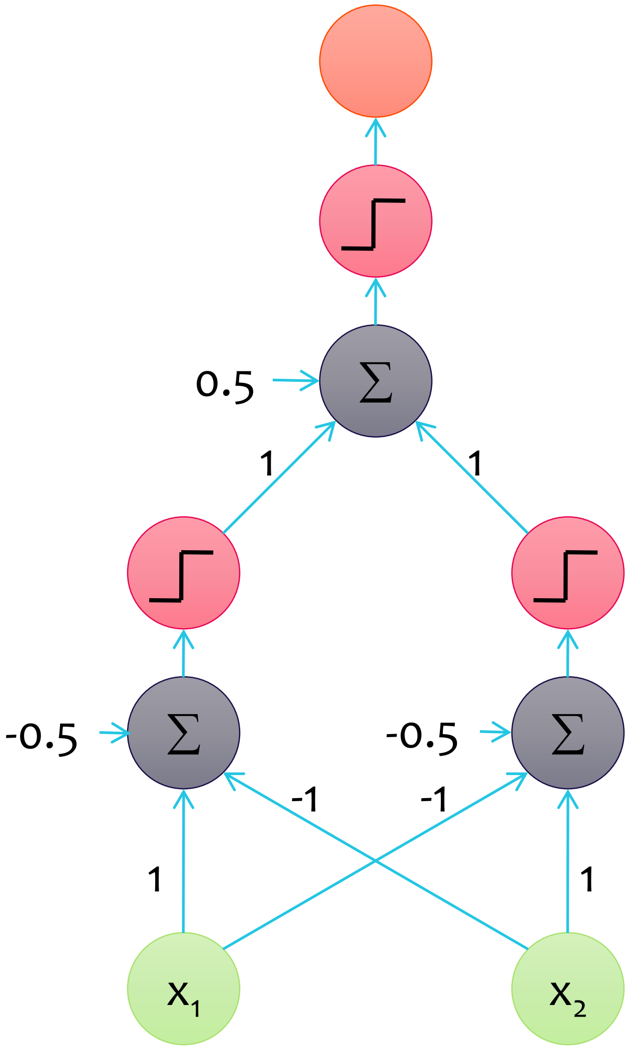
- Add hidden layers between input and output linear

$$\mathbf{O} = f\left(f\left(\mathbf{X}_{N\times D}\mathbf{W}^{1}_{D\times H_1} + b_1\right)\mathbf{W}^{2}_{H_1\times H_2} + b_2\right)$$

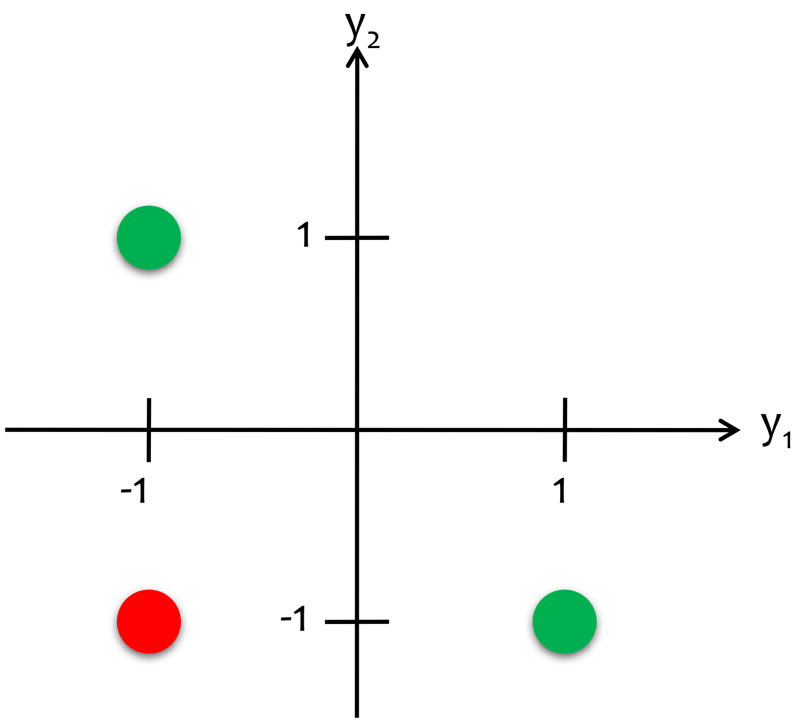


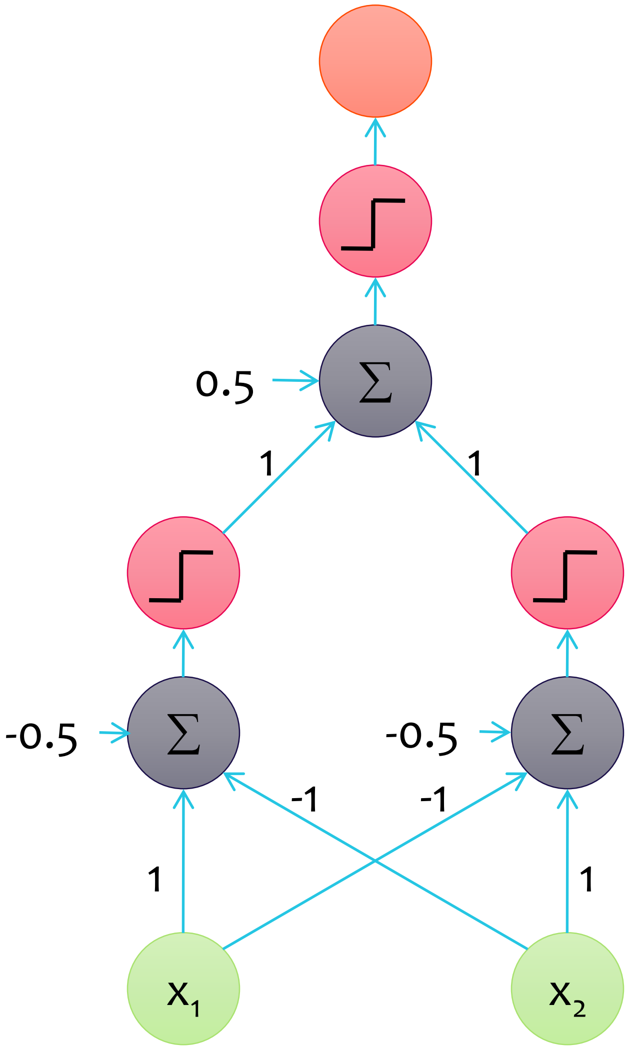
XOR with MLP



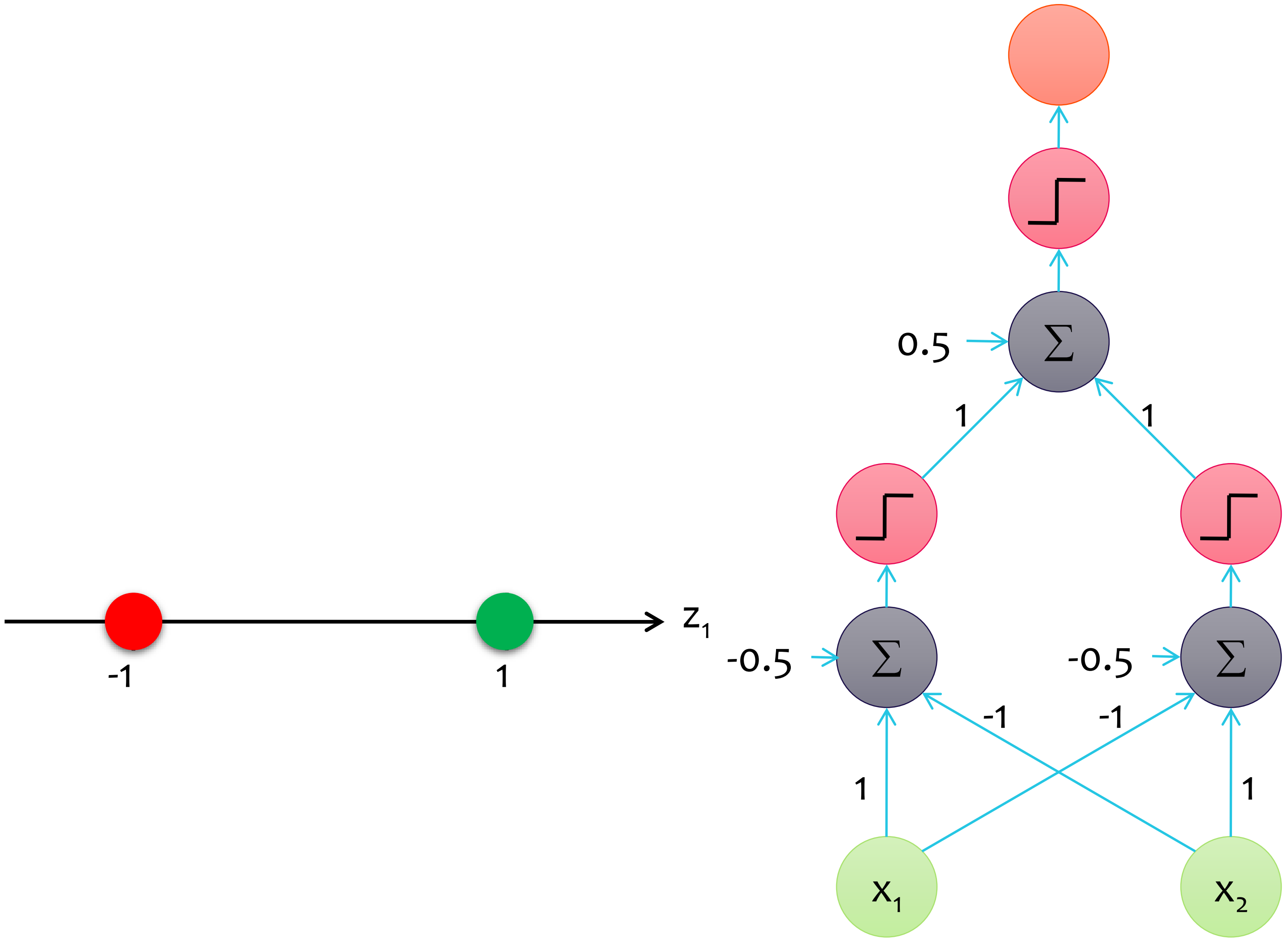


XOR with MLP

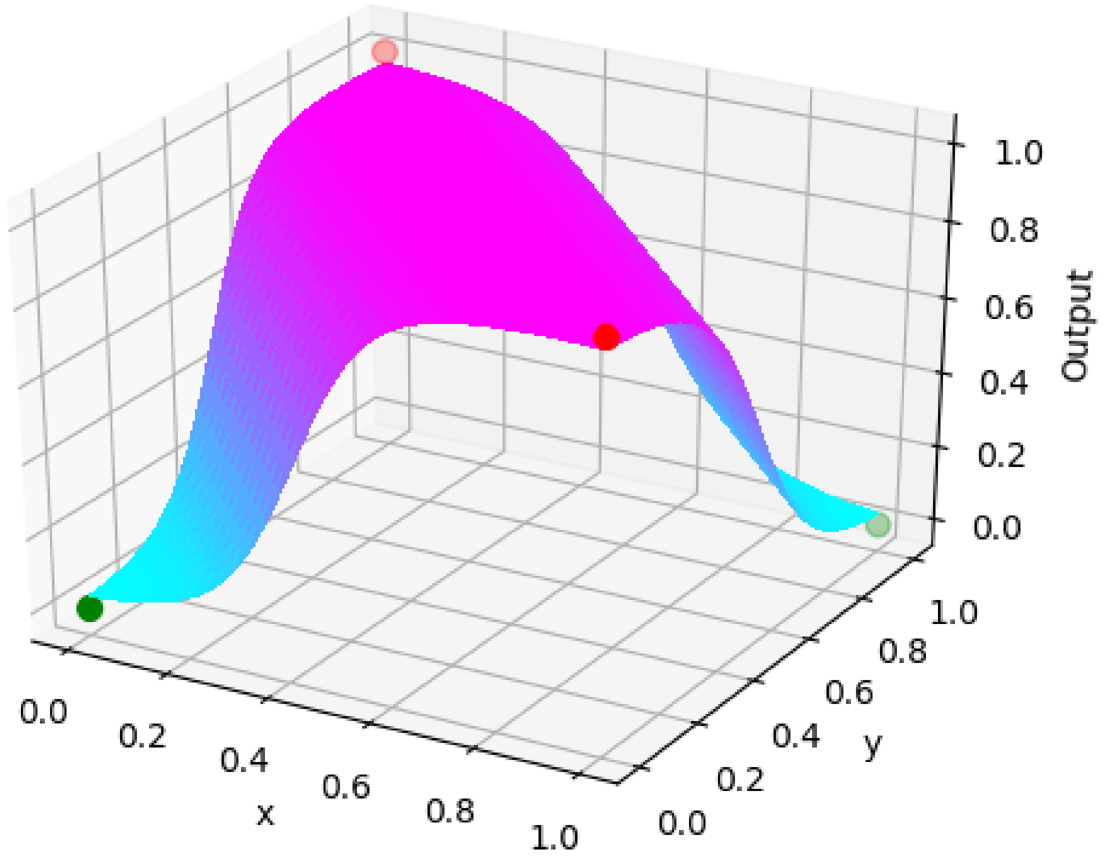




XOR with MLP



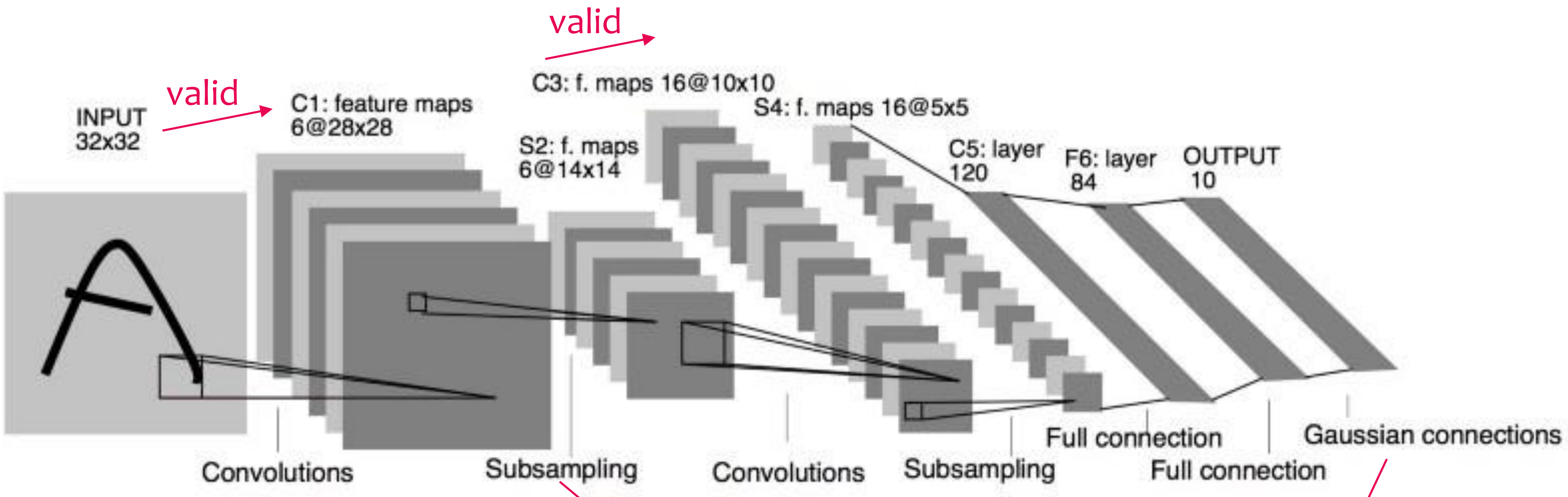
XOR with MLP



Convolutional Neural Network

Parinya Sanguansat

LeNet5: The First CNN

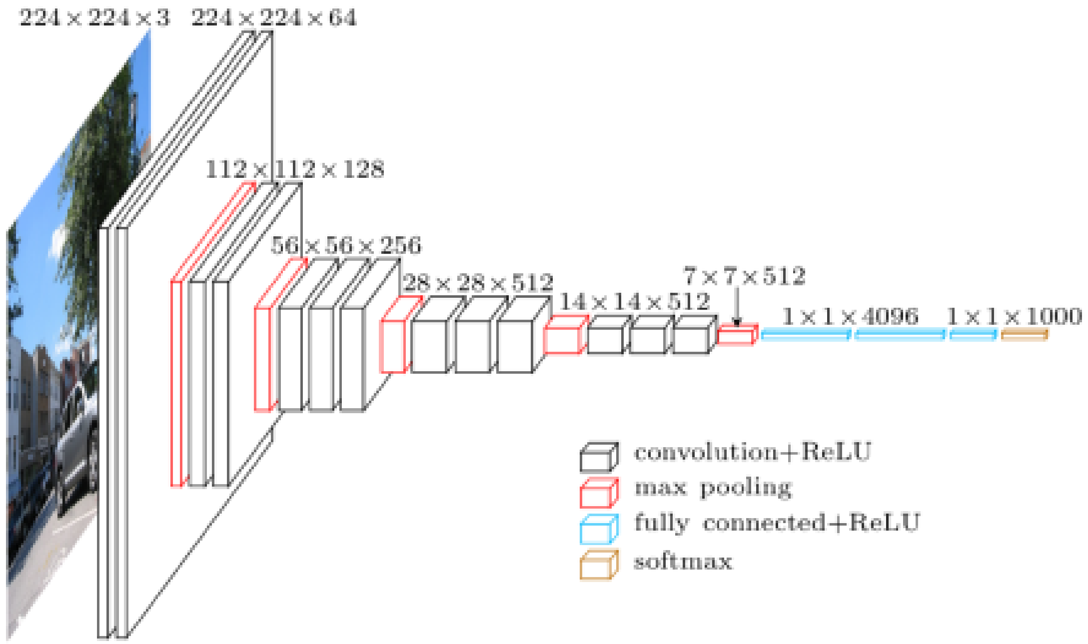


Now, replace with Max pooling

Now, replace with cross entropy

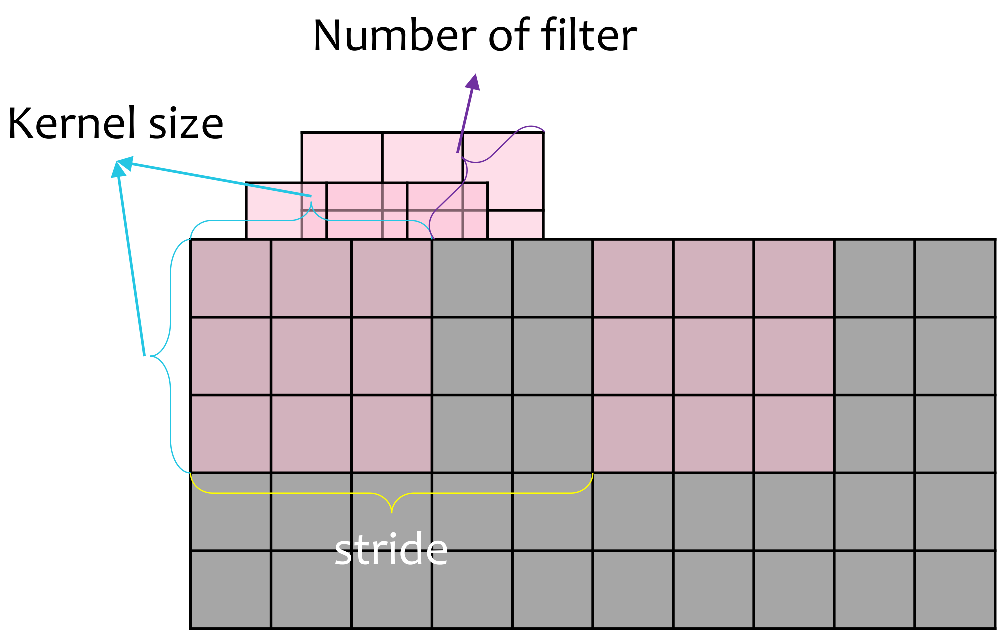
VGG

- Deeper network: VGG16
  - Better performance
  - More memory required
  - Longer training time



2D Convolutional layer

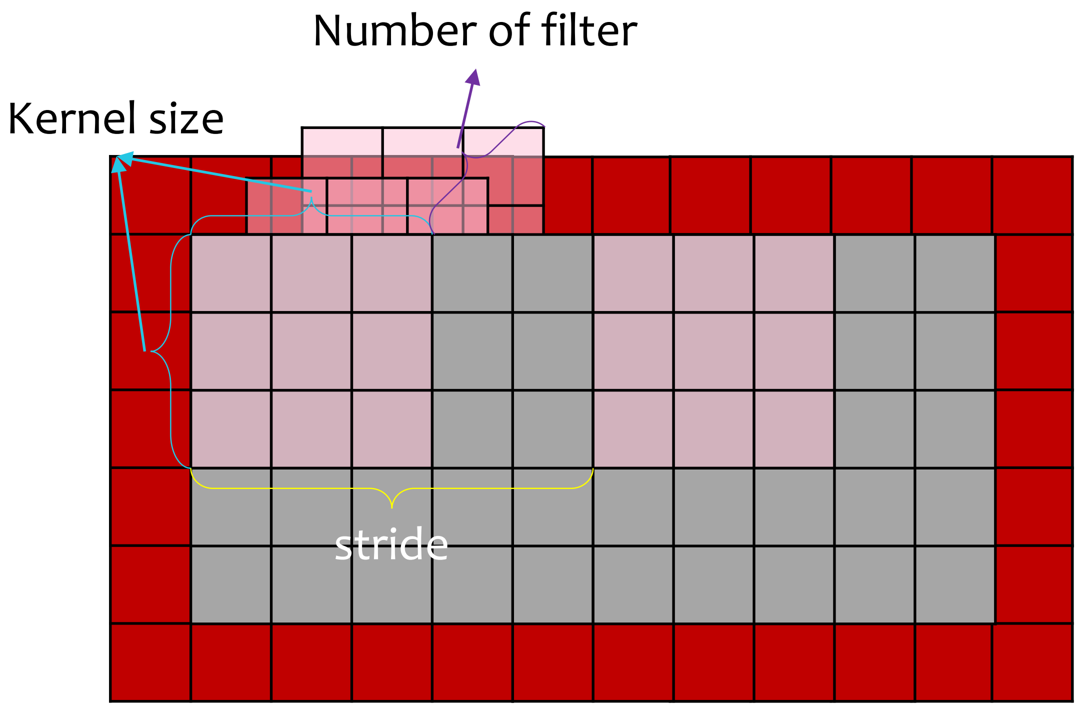
```
tf.keras.layers.Conv2D(3, (3, 3), strides=(5, 5), padding='valid')
```



Padding = Valid

2D Convolutional layer

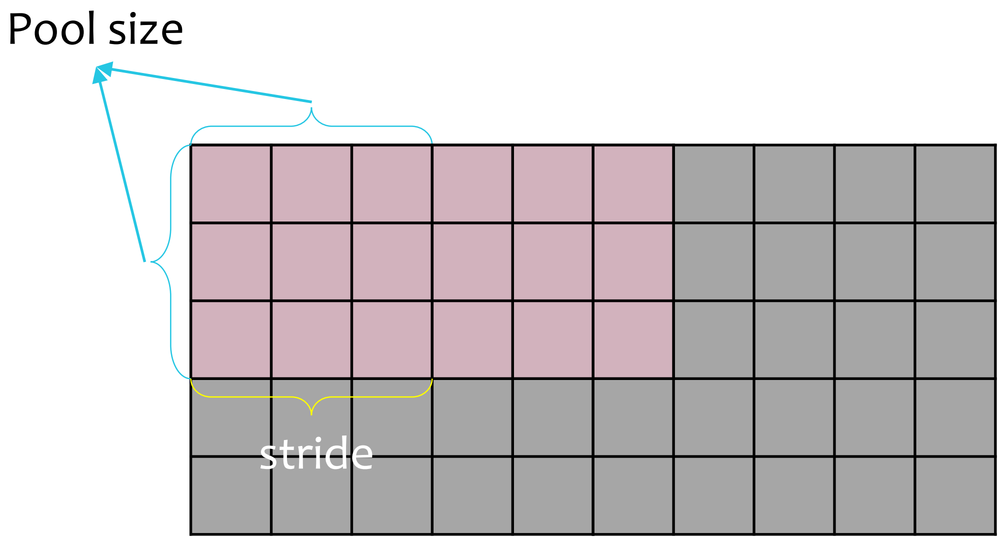
```
tf.keras.layers.Conv2D(3, (3, 3), strides=(5, 5), padding='same')
```



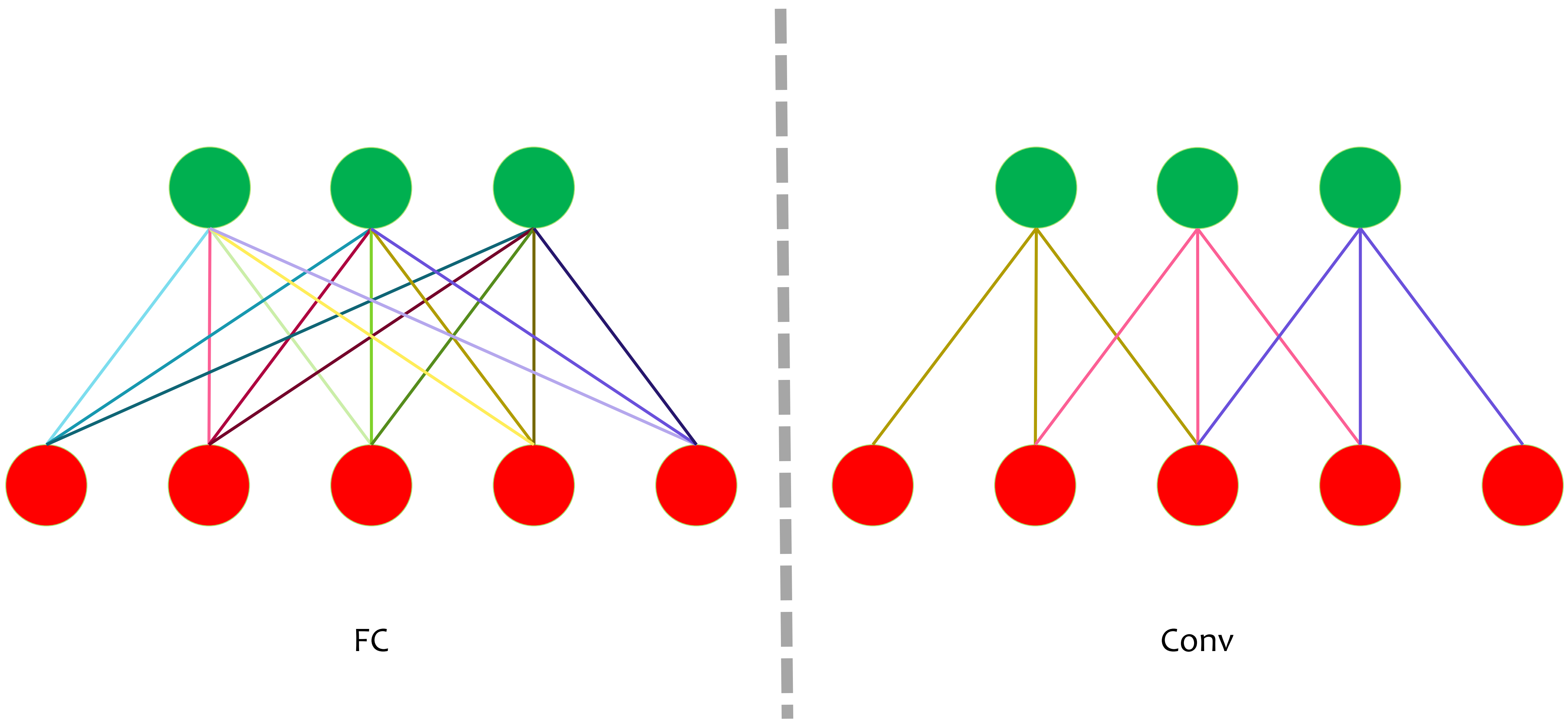
Padding = Same

2D Max Pooling layer

```
tf.keras.layers.MaxPool2D(pool_size=(3, 3), strides=None, padding='valid')
```

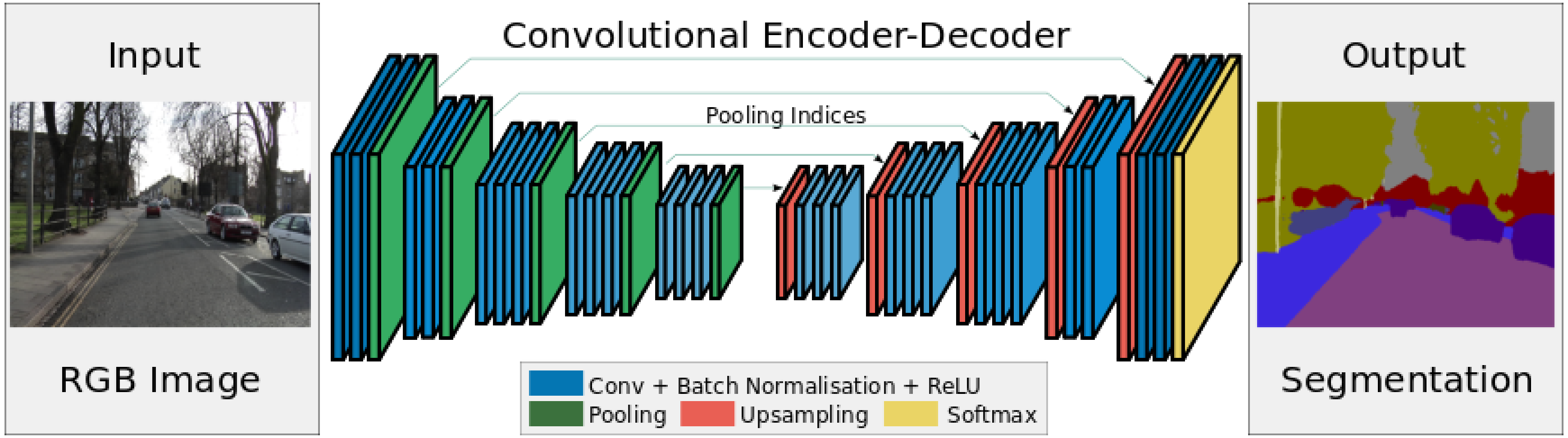


Convolutional vs Fully Connected

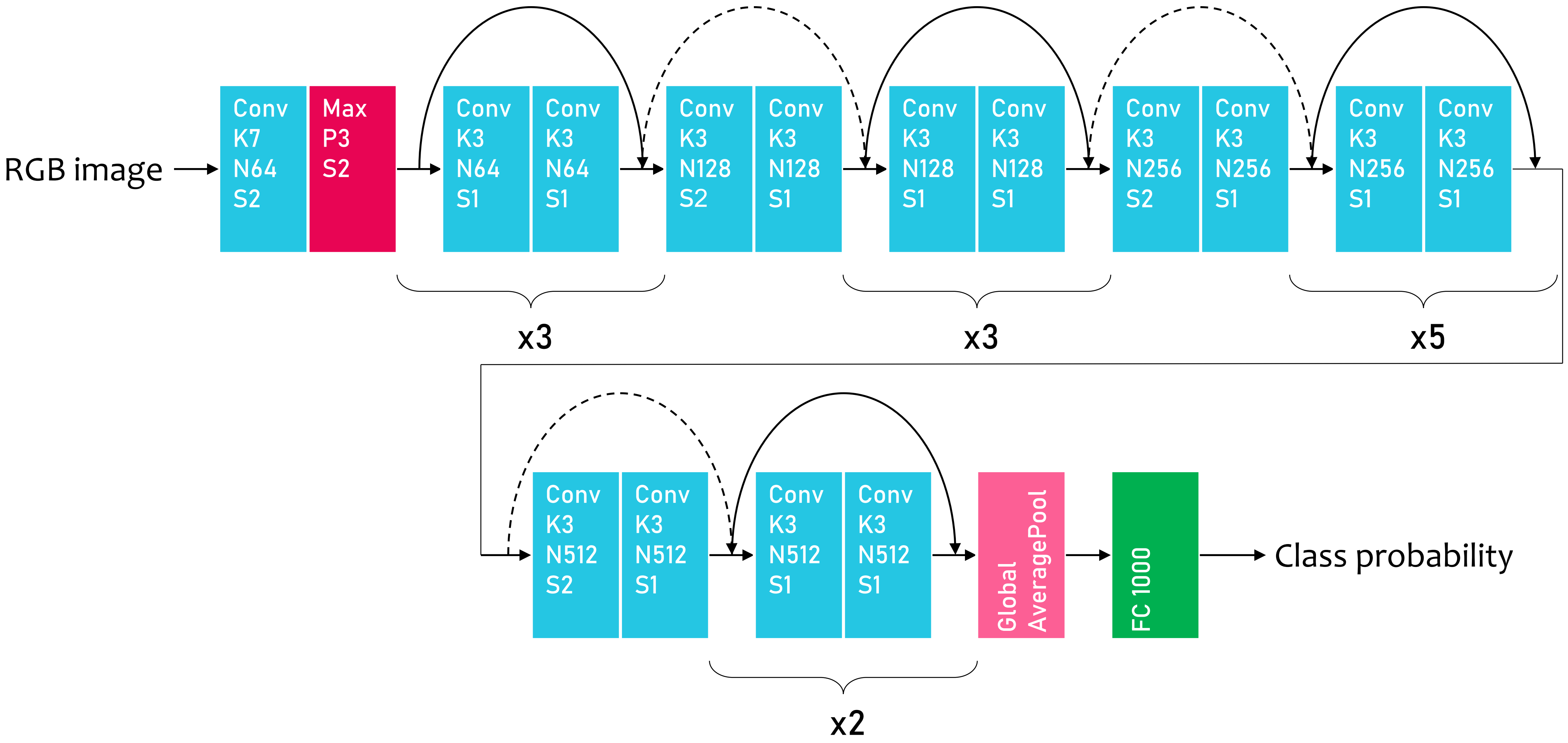


Semantic Segmentation

- Autoencoder with supervised learning
- SegNet



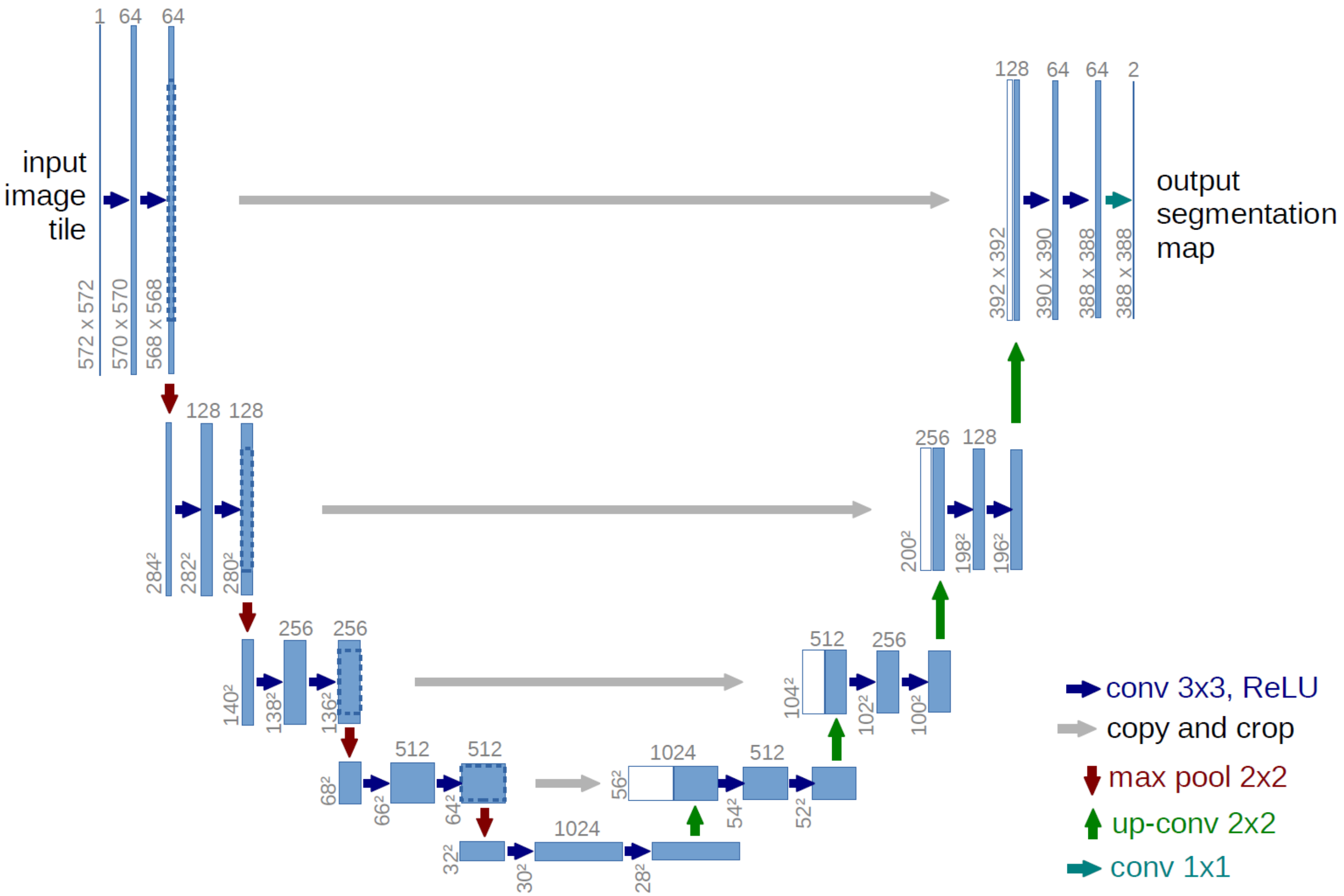
ResNet-34



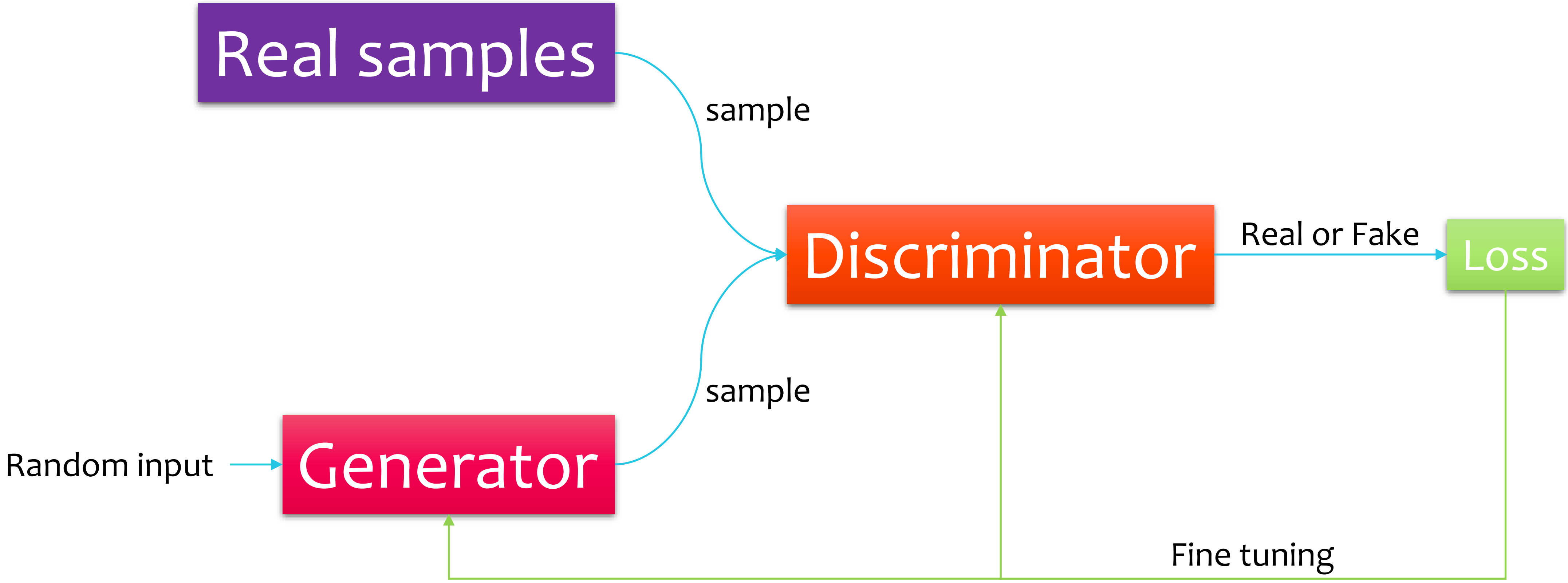
RGB image
Conv K7 N64 S2
Max P3 S2
Conv K3 N64 S1
Conv K3 N64 S1
x3
Conv K3 N128 S2
Conv K3 N128 S1
Conv K3 N128 S1
Conv K3 N128 S1
x3
Conv K3 N256 S2
Conv K3 N256 S1
Conv K3 N256 S1
Conv K3 N256 S1
x5
Conv K3 N512 S2
Conv K3 N512 S1
Conv K3 N512 S1
Conv K3 N512 S1
x2
Global AveragePool
FC 1000
Class probability

U-Net

- Concatenate encoder to decoder



Generative Adversarial Networks (GANs)



SRGAN

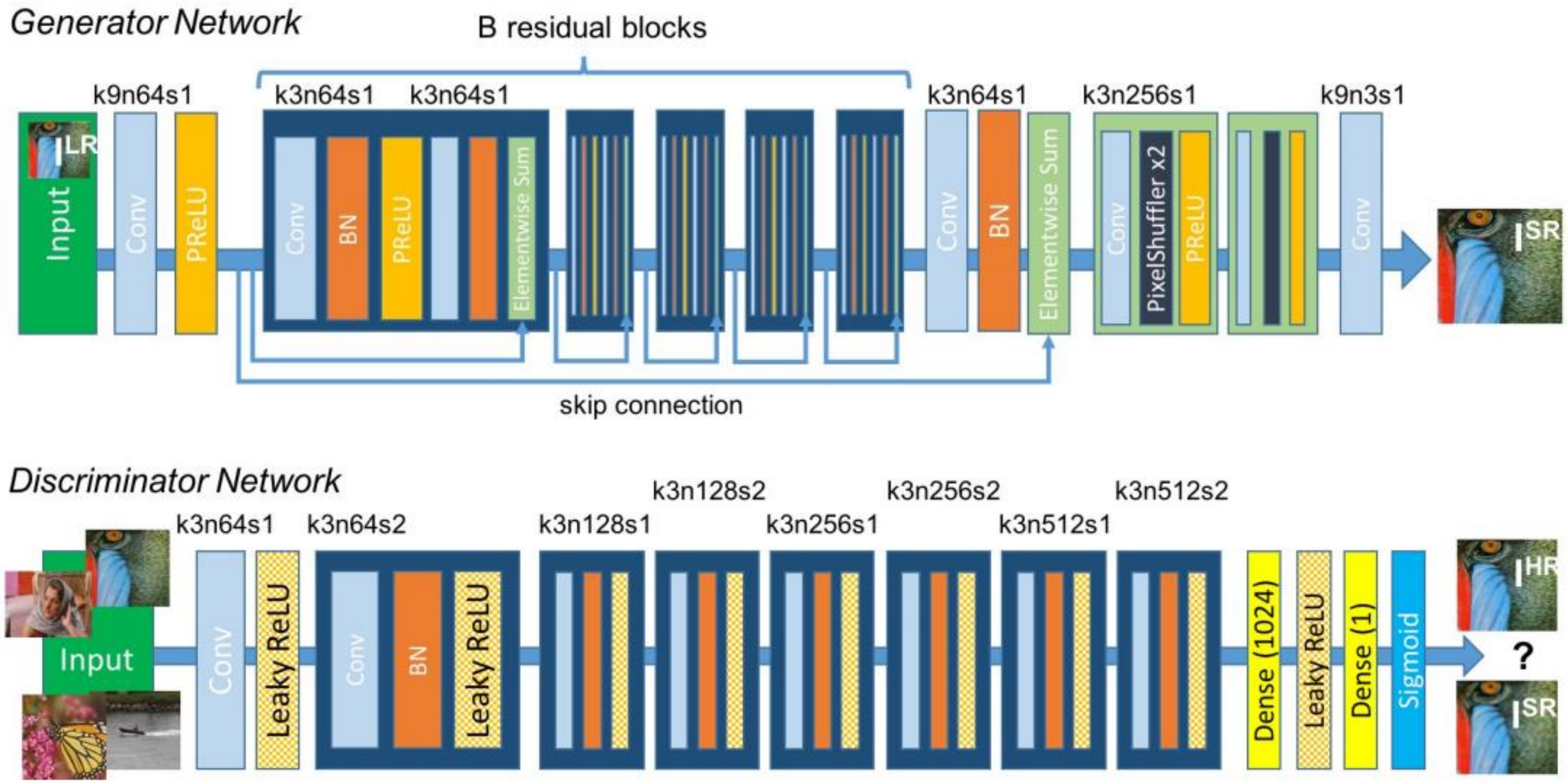
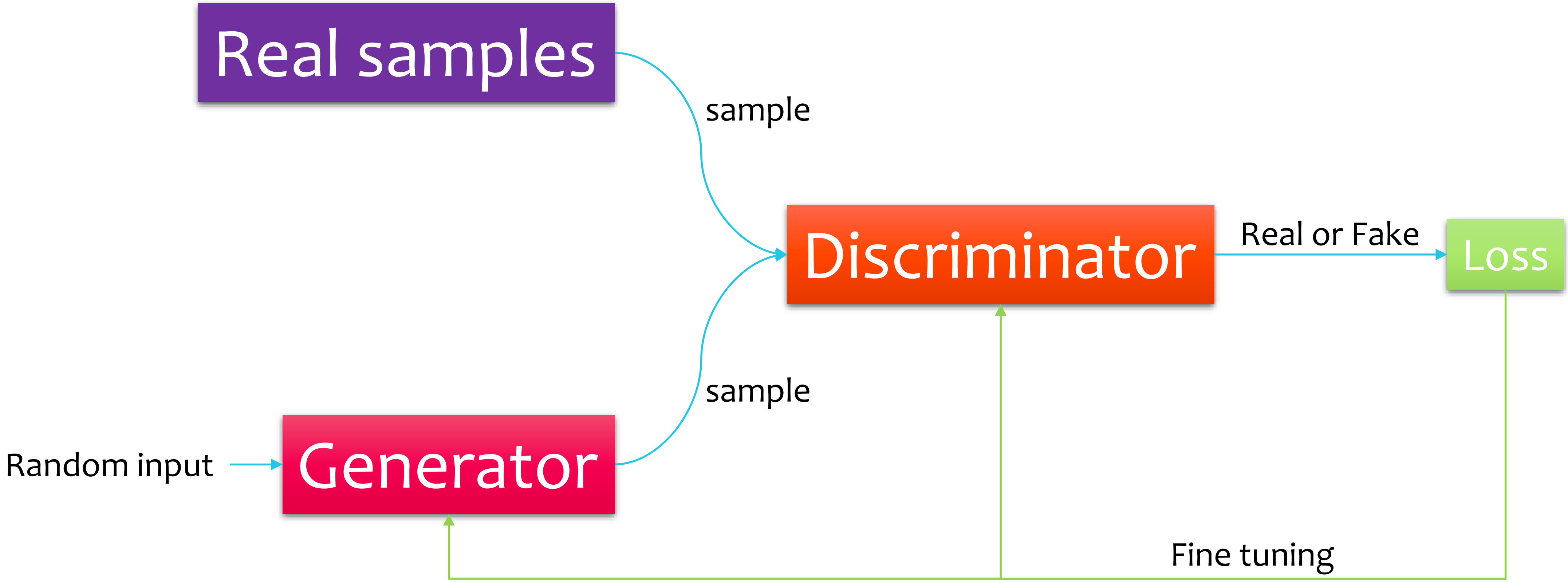


Figure 4: Architecture of Generator and Discriminator Network with corresponding kernel size (k), number of feature maps (n) and stride (s) indicated for each convolutional layer.

Generative Adversarial Networks (GANs)

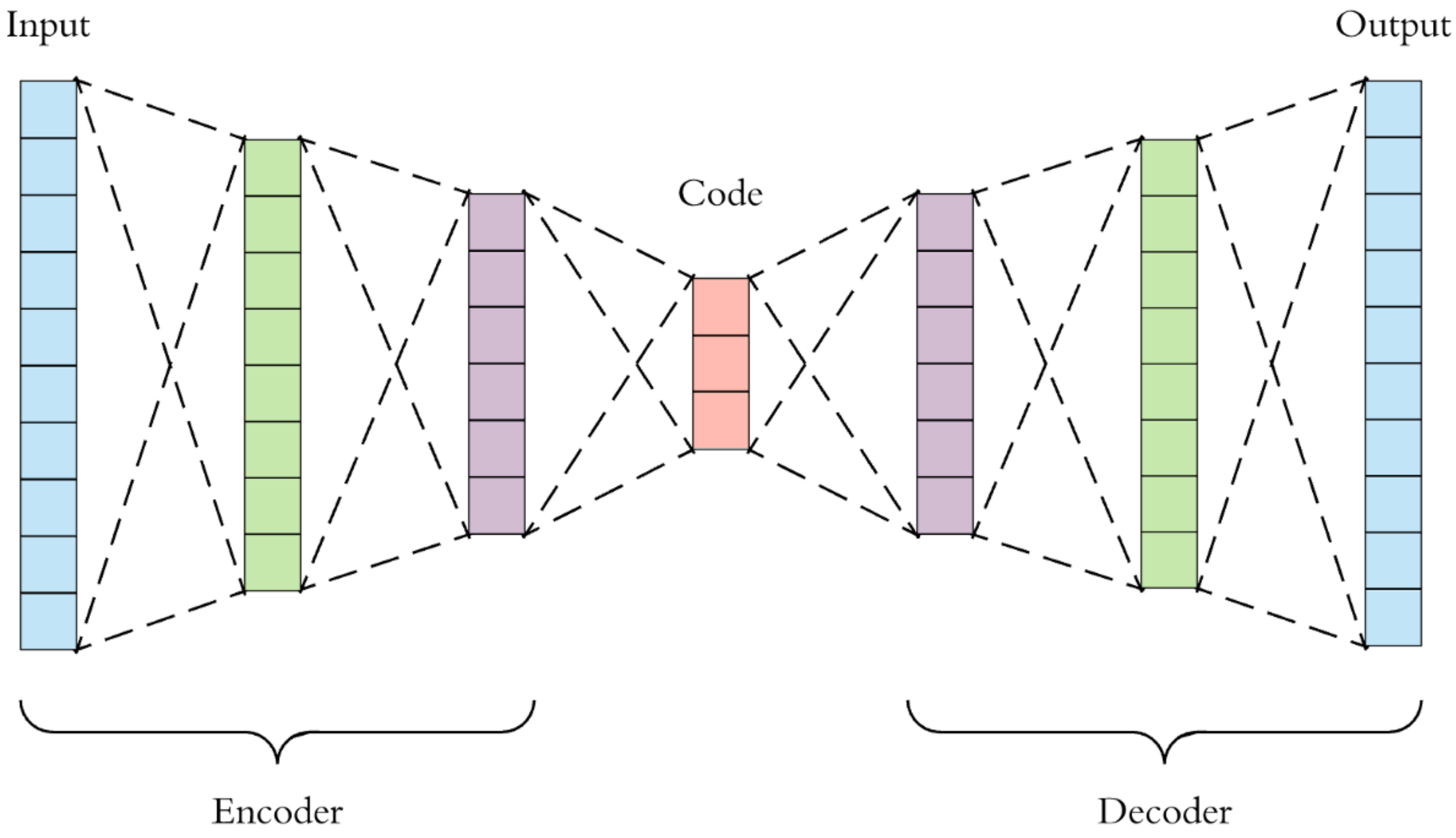


Variational Autoencoders

Parinya Sanguansat

Autoencoder

- Unsupervised: use only x not y !!!

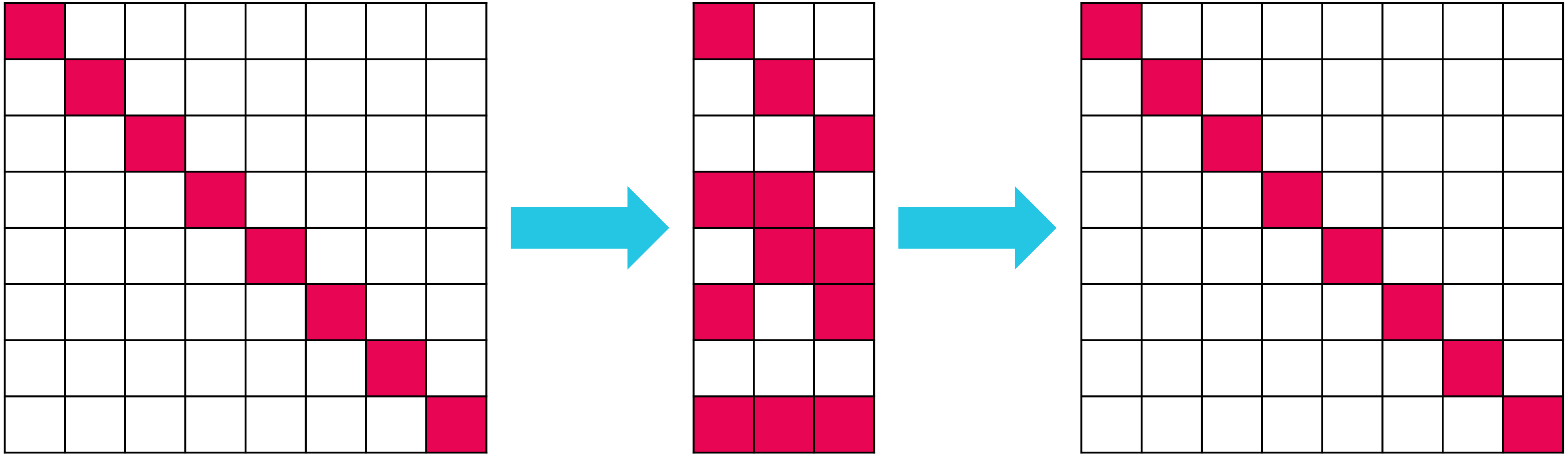


Autoencoder

```
print(X)
```

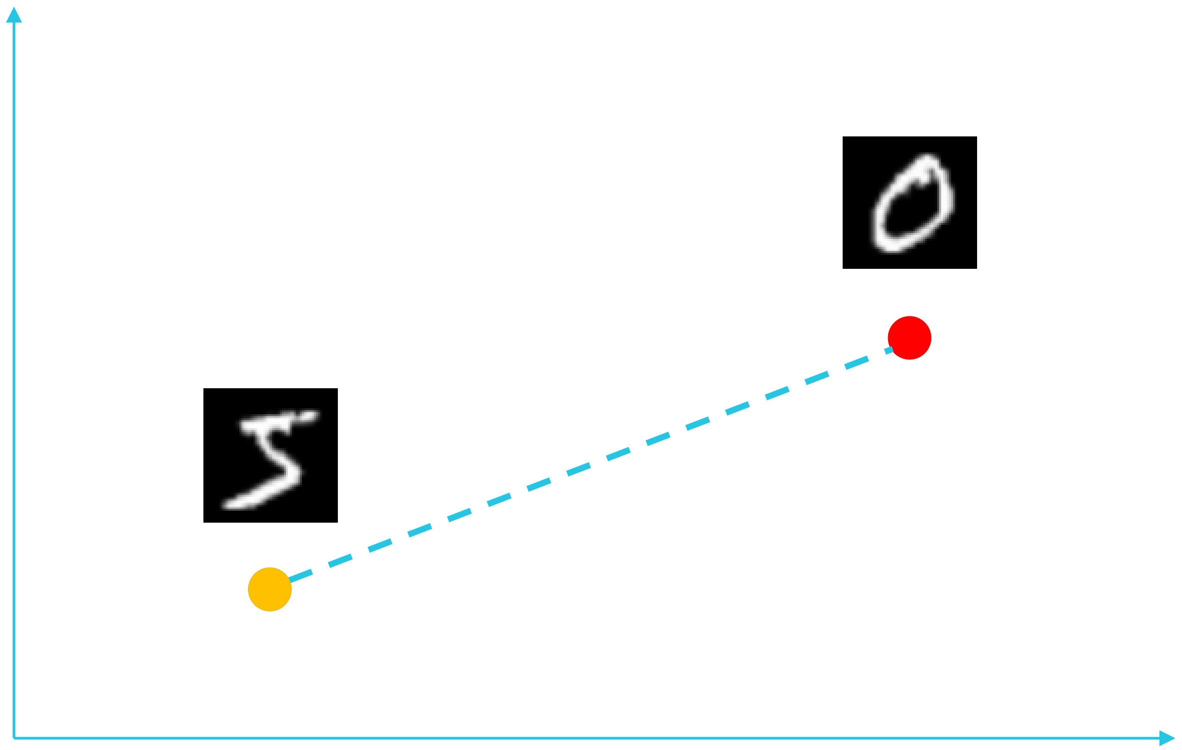
```
np.round(encoder(X))
```

```
Z = np.round(model(X))
print(Z)
```



What does it mean?

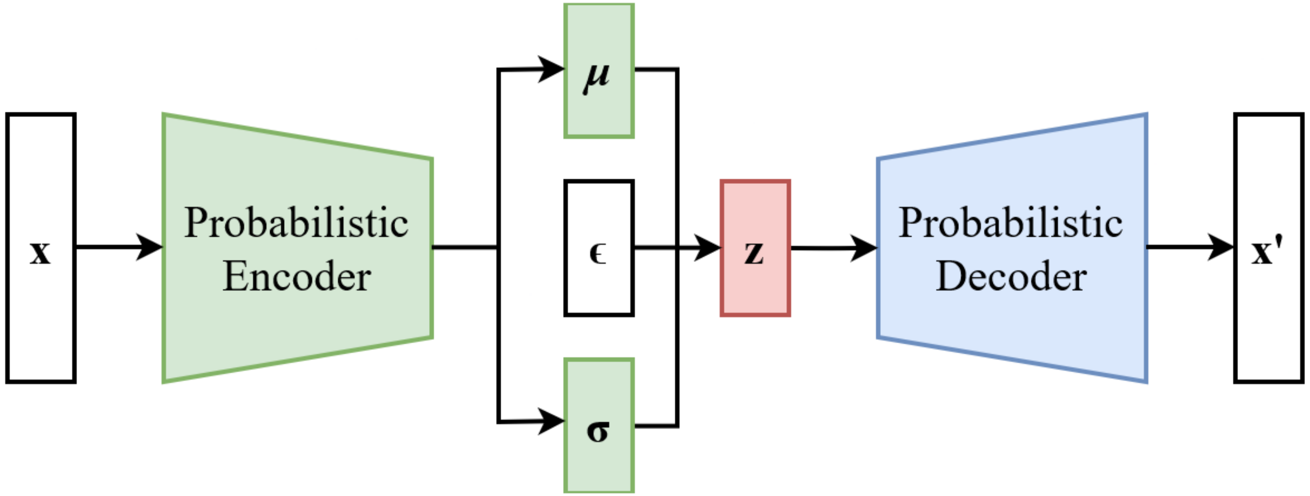
Limitation of Autoencoder



Latent space

VAE (Variational Autoencoder)

Mapping input into a distribution instead of mapping the it into a fixed vector

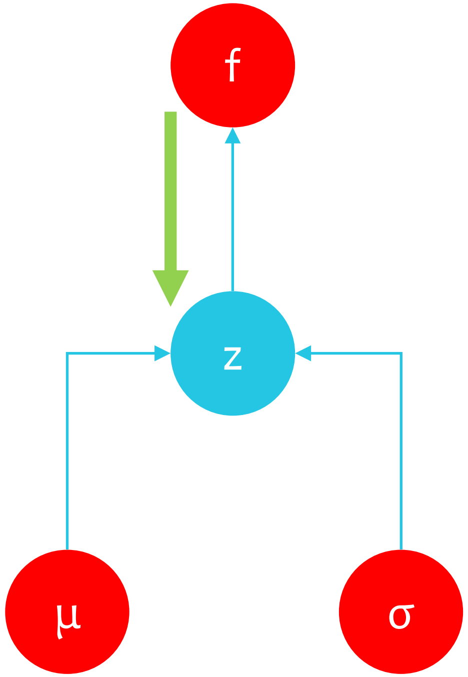


Loss = Reconstruction loss + KL Divergence

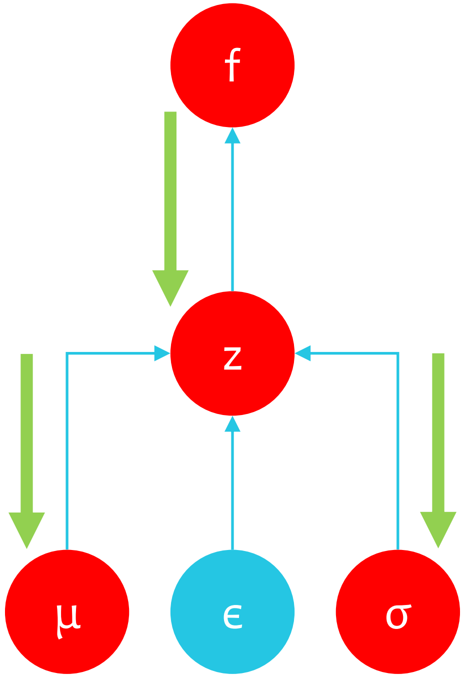
Binary cross entropy, MSE, … force to normal distribution

Reparameterization trick

Backpropagation work with deterministic node, not for stochastic node



$z \sim \mathcal{N}(\mu, \sigma)$



$z = \mu + \epsilon\sigma$

Kullback-Leibler (KL) divergence

How Q divergence from P

Want to know the difference $P(x) - Q(x)$

Easier with log $logP(x) - logQ(x)$

Many values $\mathbb{E}[logP(x) - logQ(x)] = \sum_{x \in \mathcal{X}} P(x)[logP(x) - logQ(x)]$

$$D_{KL}(P||Q) = \sum_{x \in \mathcal{X}} P(x)\left[log\frac{P(x)}{Q(x)}\right]$$

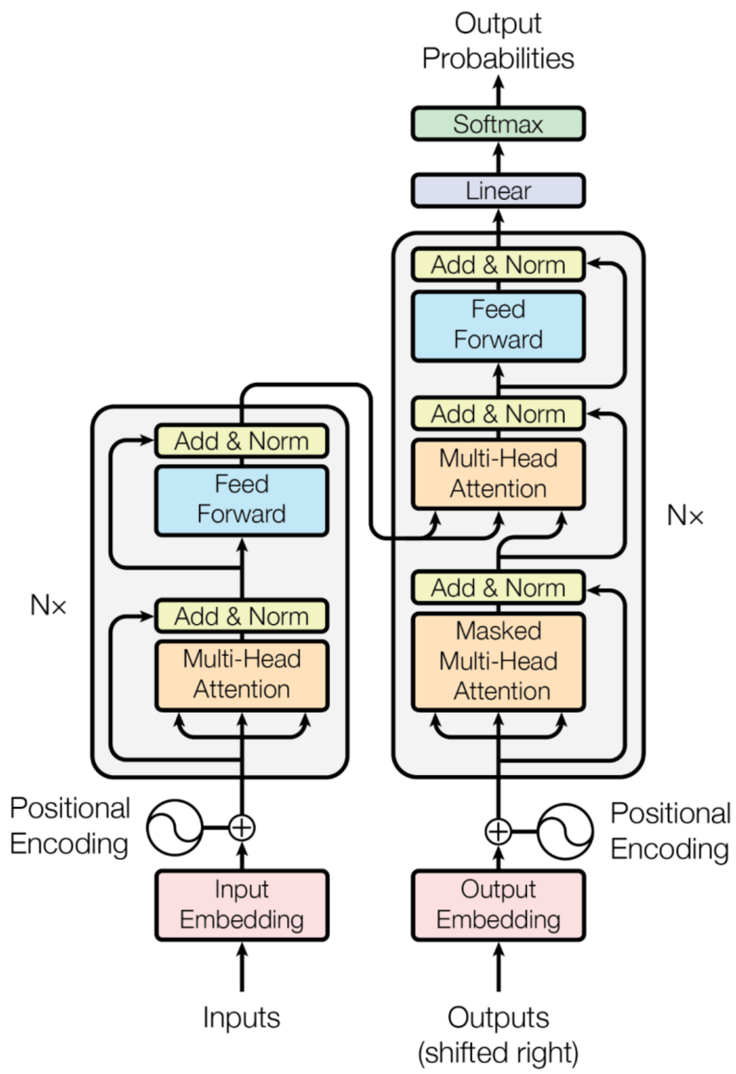
Continuous values $D_{KL}(p||q) = \int p(x)\left[log\frac{p(x)}{q(x)}\right]dx$

Transformer

Parinya Sanguansat

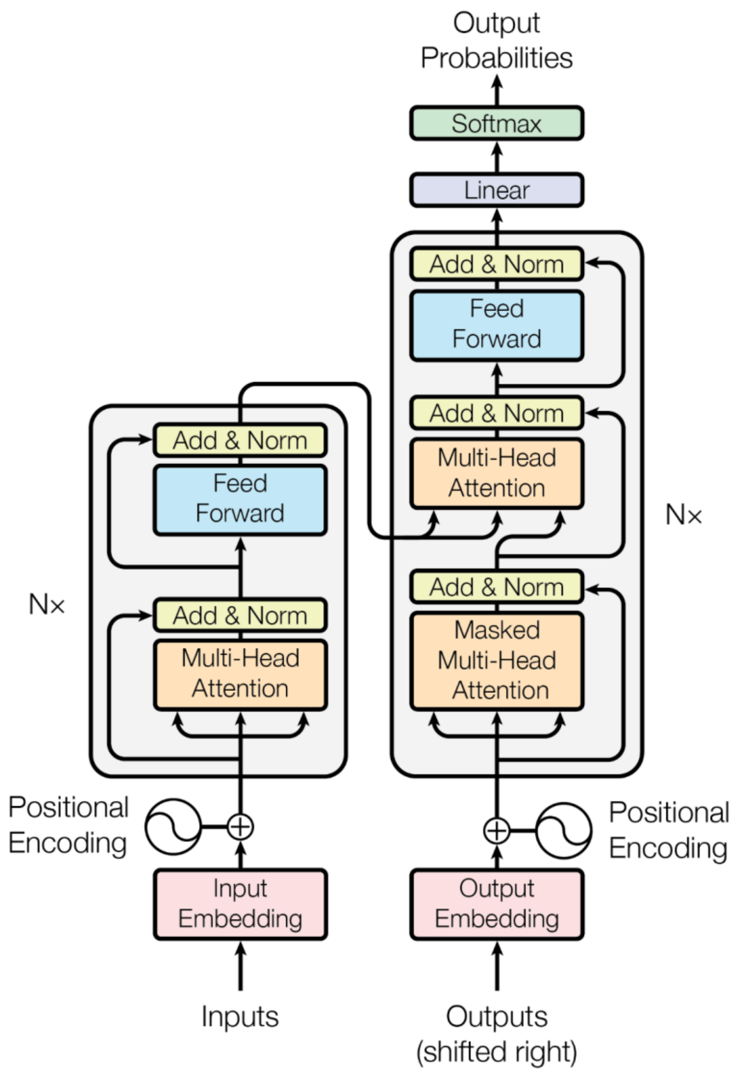
Attention

- Original design for machine translation
- Google



The Transformer

- Original design for machine translation
- Google



https://arxiv.org/pdf/1706.03762.pdf

Basic Retrieval Mechanism

- Components
  - Query
  - Key
  - Value

Q

match?

Key	Value
K1	V1
K2	V2
:	:
Kn	Vn

return

- Find a key that matches with query
- Return only one value of that key

Word Embedding

- Basic usage in Embedding layer

I eat fish at school.

A school of fish.

Must unique

Key	Value
fish	[0.71, 0.59, 0.40, 0.70, 0.62]
school	[0.60, 0.51, 0.39, 0.86, 0.61]
:	:
eat	[0.17, 0.58, 0.09, 0.54, 0.34]

Attention

- Components
  - Query
  - Key
  - Value

$$Attention(\mathbf{q},\mathbf{k},\mathbf{v}) = \sum_i f(\mathbf{q},\mathbf{k}_i) \cdot \mathbf{v}_i$$

Similarity function

- Find the similarity of keys and query
- Return the combination of values

Examples of Similarity functions

- Dot-product: $\mathbf{q}^T \mathbf{k}_i$
- Scaled Dot-Product: $\dfrac{\mathbf{q}^T \mathbf{k}_i}{\sqrt{d}}$
- General Dot-Product: $\mathbf{q}^T \mathbf{W} \mathbf{k}_i$
- Additive Similarity: $\mathbf{w}_q^T \mathbf{q} + \mathbf{w}_k^T \mathbf{k}_i$
- Kernel: $Kernel(\mathbf{q}, \mathbf{k}_i)$

Example

$$\frac{\mathbf{q}^T \mathbf{k}_i}{\sqrt{d}}$$

Query	Key	Value
[0.30, 0.29, 0.33]	[0.55, 0.24, 0.10]	[0.71, 0.59, 0.40, 0.70, 0.62]
	[0.87, 0.17, 0.46]	[0.60, 0.51, 0.39, 0.86, 0.61]
	[0.98, 0.01, 0.61]	[0.17, 0.58, 0.09, 0.54, 0.34]

0.154
0.267
0.288

Σ = 0.71

Example

$\frac{\mathbf{q}^T \mathbf{k}_i}{\sqrt{d}}$

Query	Key	Value
[0.30, 0.29, 0.33]	[0.55, 0.24, 0.10]	x[0.71, 0.59, 0.40, 0.70, 0.62]
	[0.87, 0.17, 0.46]	x[0.60, 0.51, 0.39, 0.86, 0.61]
	[0.98, 0.01, 0.61]	x[0.17, 0.58, 0.09, 0.54, 0.34]

0.154
0.267
0.288

Softmax

$$Attention(\mathbf{q}, \mathbf{k}, \mathbf{v}) = \sum_i \text{softmax}\left(f\left(\mathbf{q}, \mathbf{k}_i\right)\right) \cdot \mathbf{v}_i$$

$$\left.\begin{array}{ll} 0.31 & \text{x}[0.71, 0.59, 0.40, 0.70, 0.62] \\ 0.34 & \text{x}[0.60, 0.51, 0.39, 0.86, 0.61] \\ 0.35 & \text{x}[0.17, 0.58, 0.09, 0.54, 0.34] \end{array}\right\} + \rightarrow [0.48, 0.56, 0.29, 0.70, 0.52]$$

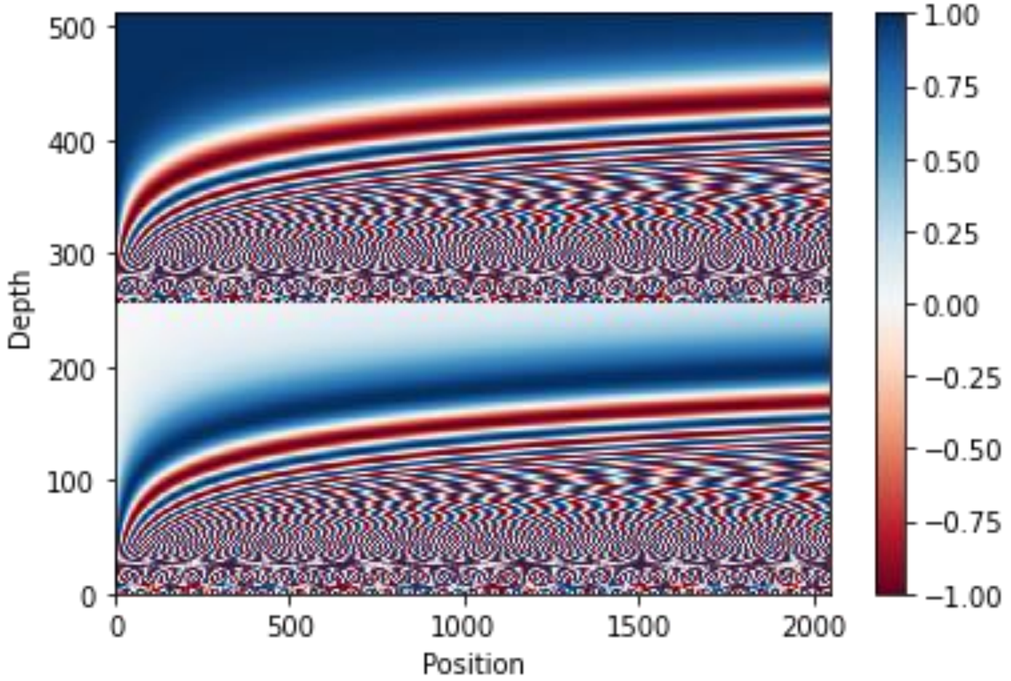
$\Sigma = 1$

Positional Encoding

- No sequential order in attention layer
- Adding the information about the position into embedding vector
- Original one:

$$PE_{(pos,2i)} = \sin\left(pos / 10000^{2i/d_{\text{model}}} \right)$$

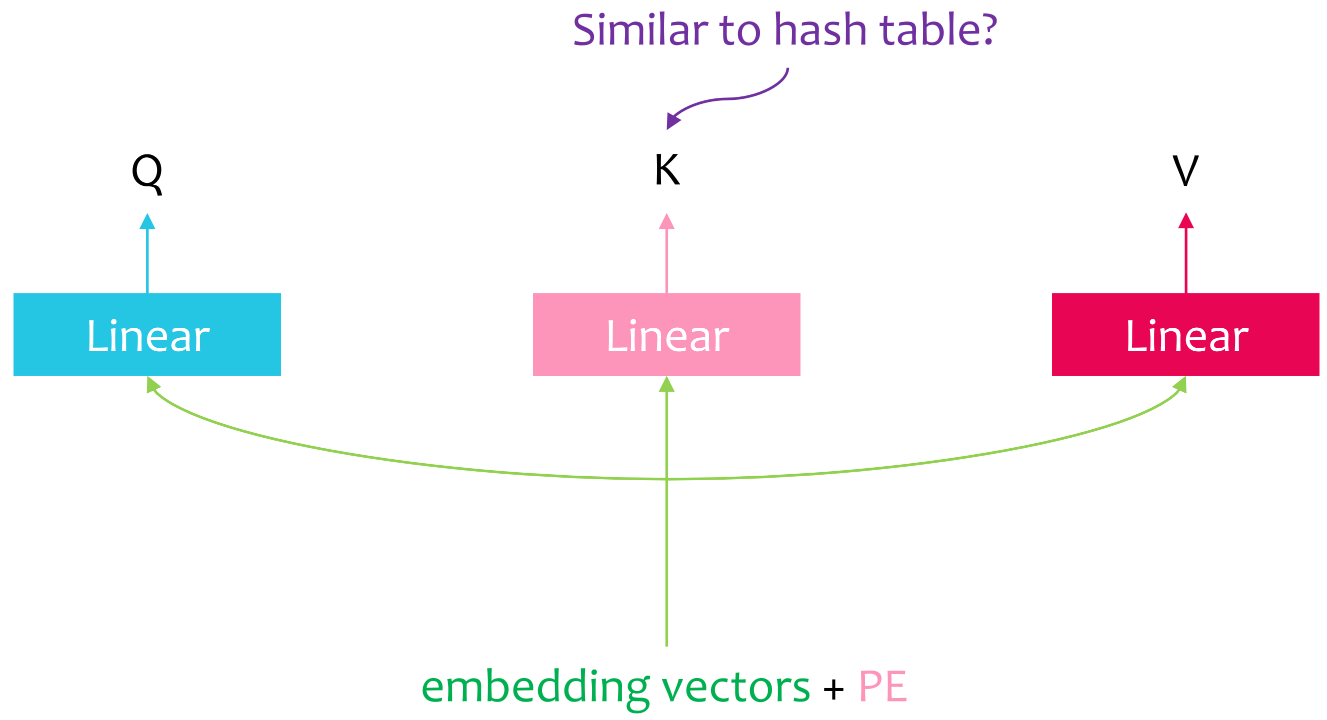
$$PE_{(pos,2i+1)} = \cos\left(pos / 10000^{2i/d_{\text{model}}} \right)$$



https://www.tensorflow.org/text/tutorials/transformer

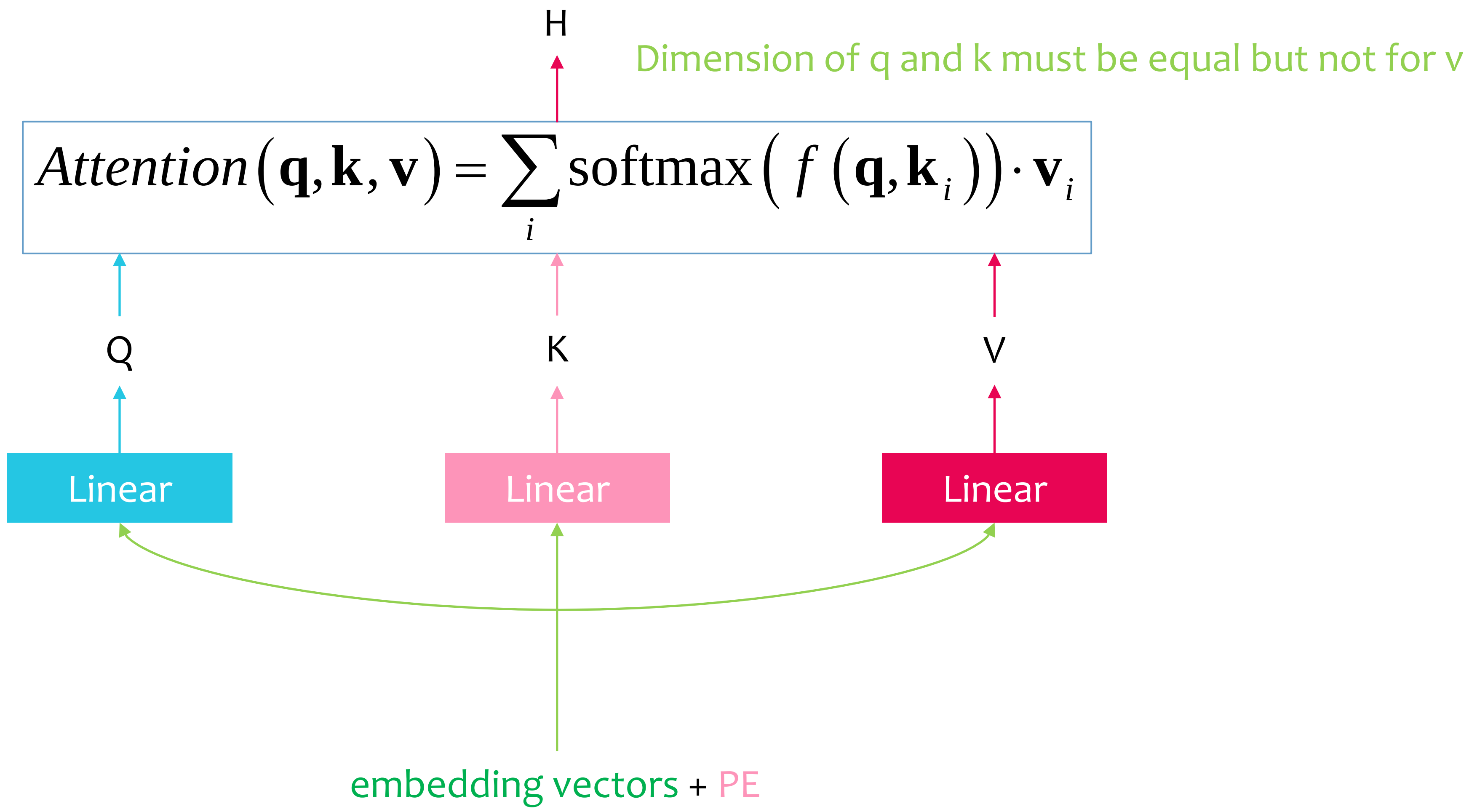
Q, K, V

- We have only embedding vectors + PE



Self-Attention

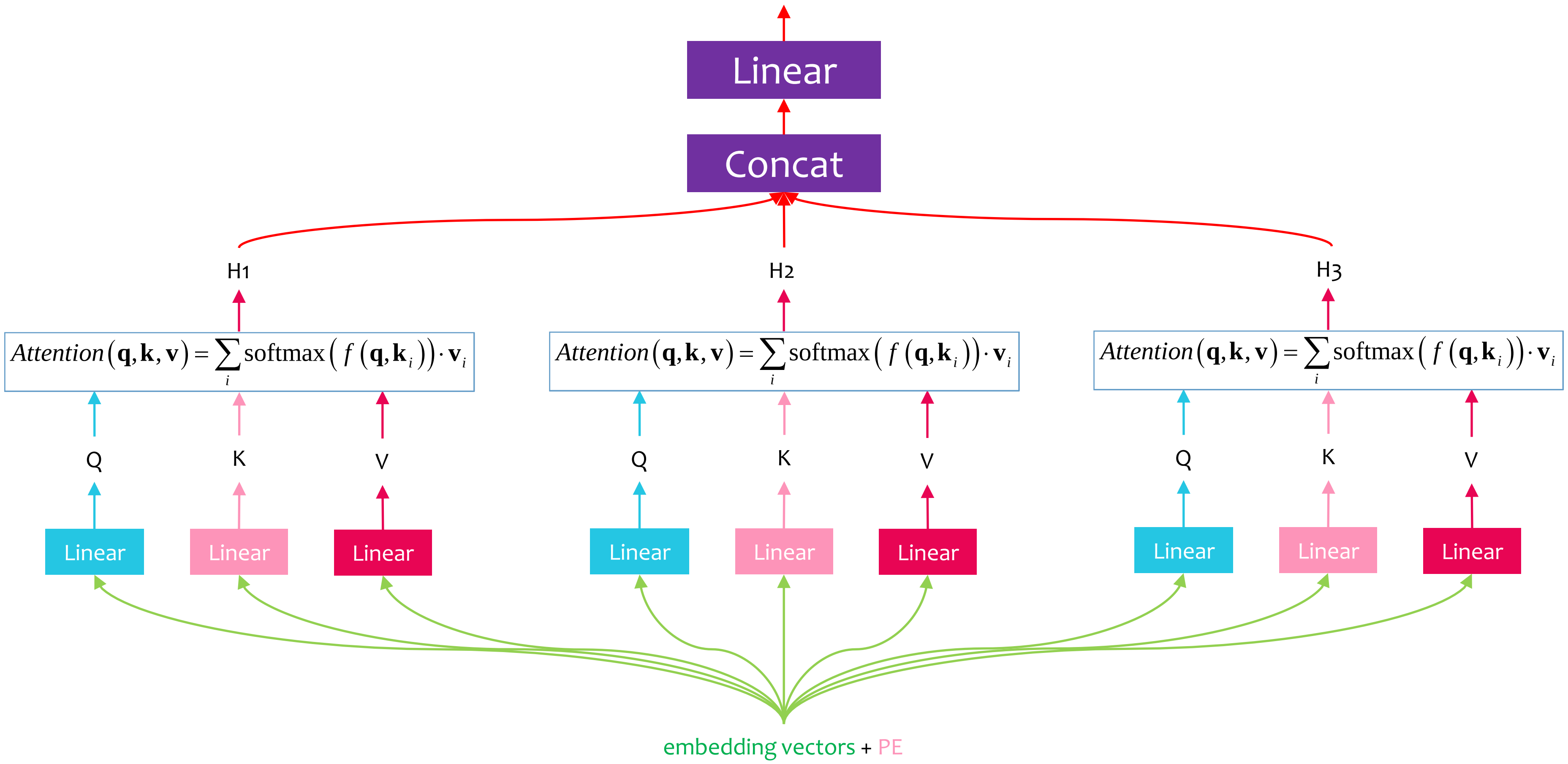
- Self-attention (intra-attention) is an attention mechanism relating different positions of a single sequence in order to compute a representation of the sequence.



$$Attention(\mathbf{q},\mathbf{k},\mathbf{v}) = \sum_i \mathrm{softmax}\left(f\left(\mathbf{q},\mathbf{k}_i\right)\right)\cdot \mathbf{v}_i$$

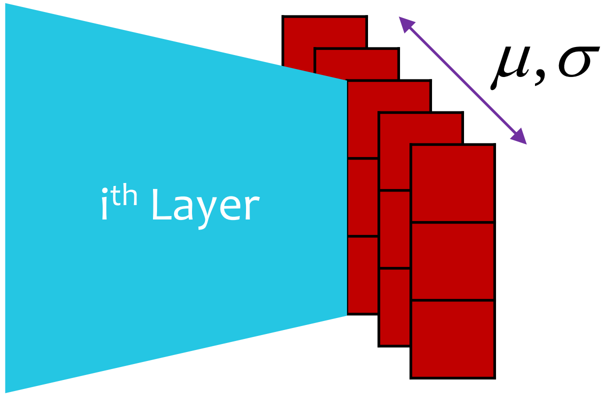
Multihead Self-Attention: MSA

- MSA Block (original #head = 8)

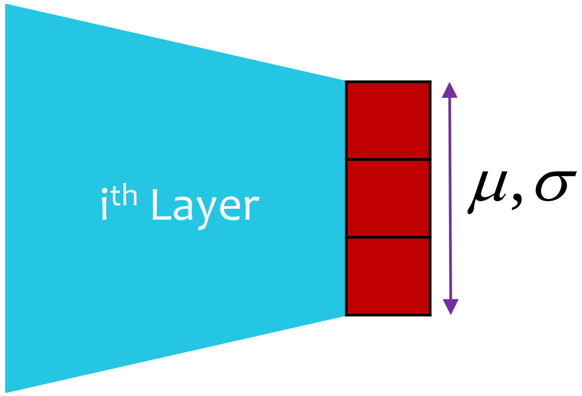


Layer Normalization

Batch Normalization



Layer Normalization



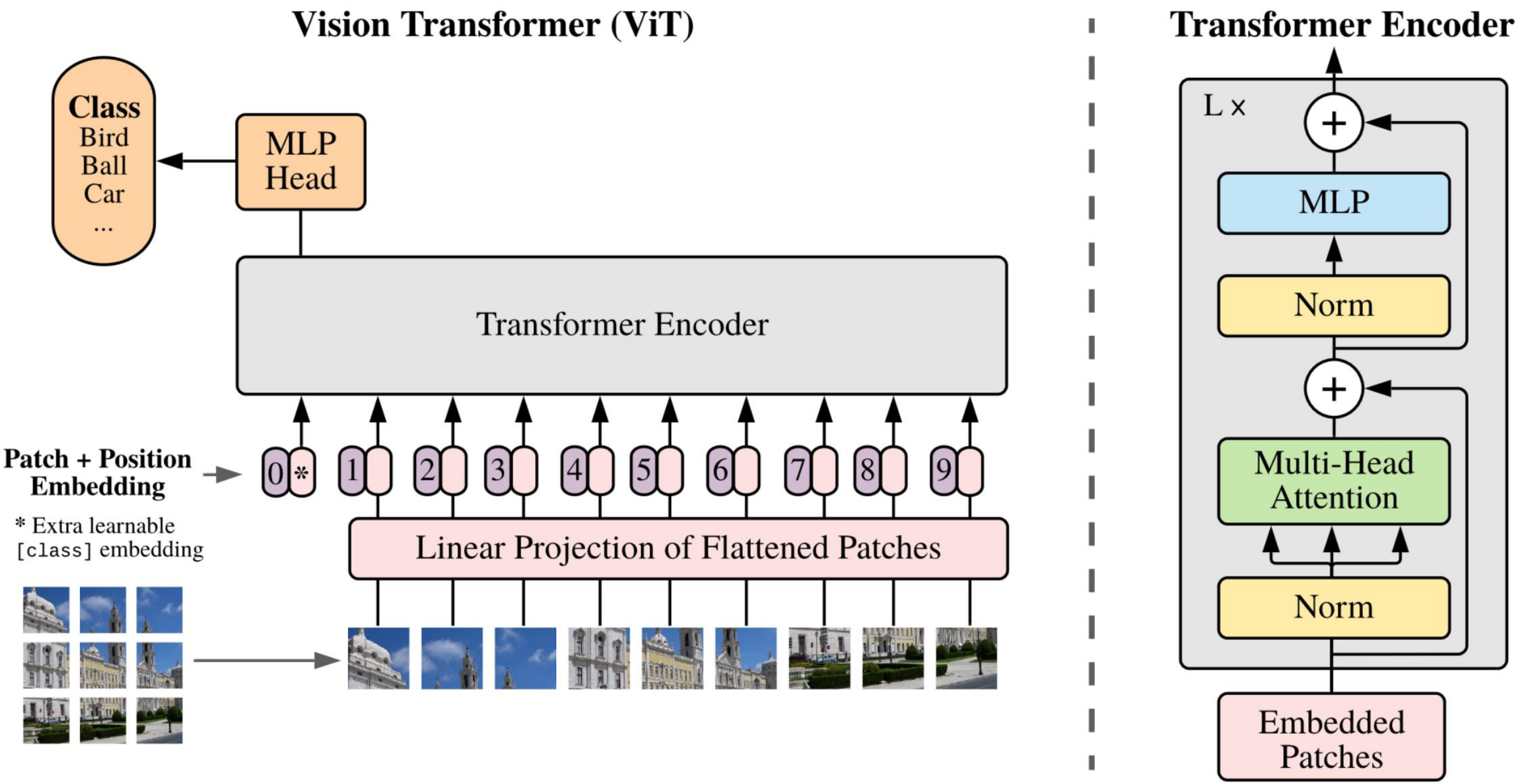
AN IMAGE IS WORTH 16X16 WORDS:
TRANSFORMERS FOR IMAGE RECOGNITION AT SCALE

Vision Transformer: ViT (2020)

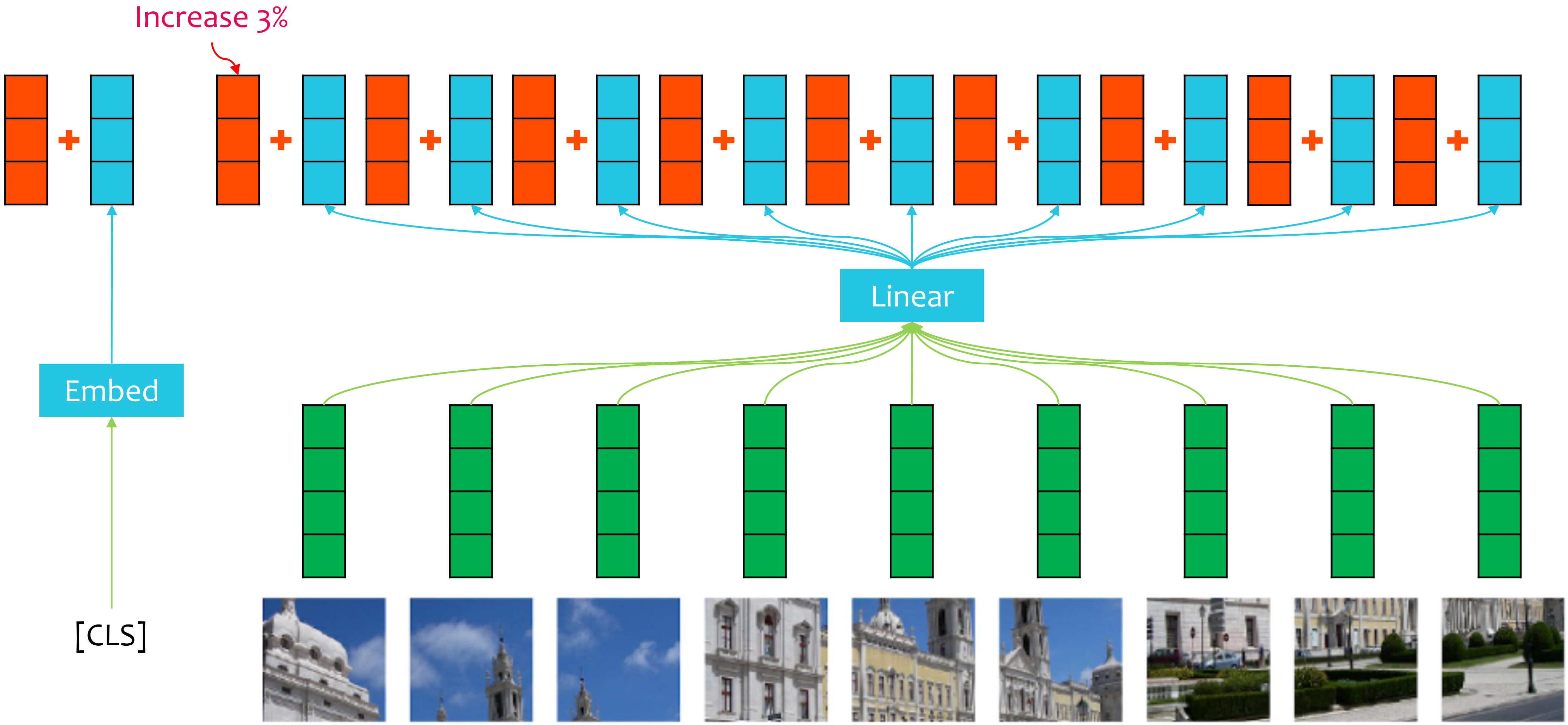
Vision Transformer

- Naïve way is to use pixel unit but it is high computational cost
- ViT use patch (without overlapping): 16 x 16
- Use [class] token (similar to BERT)

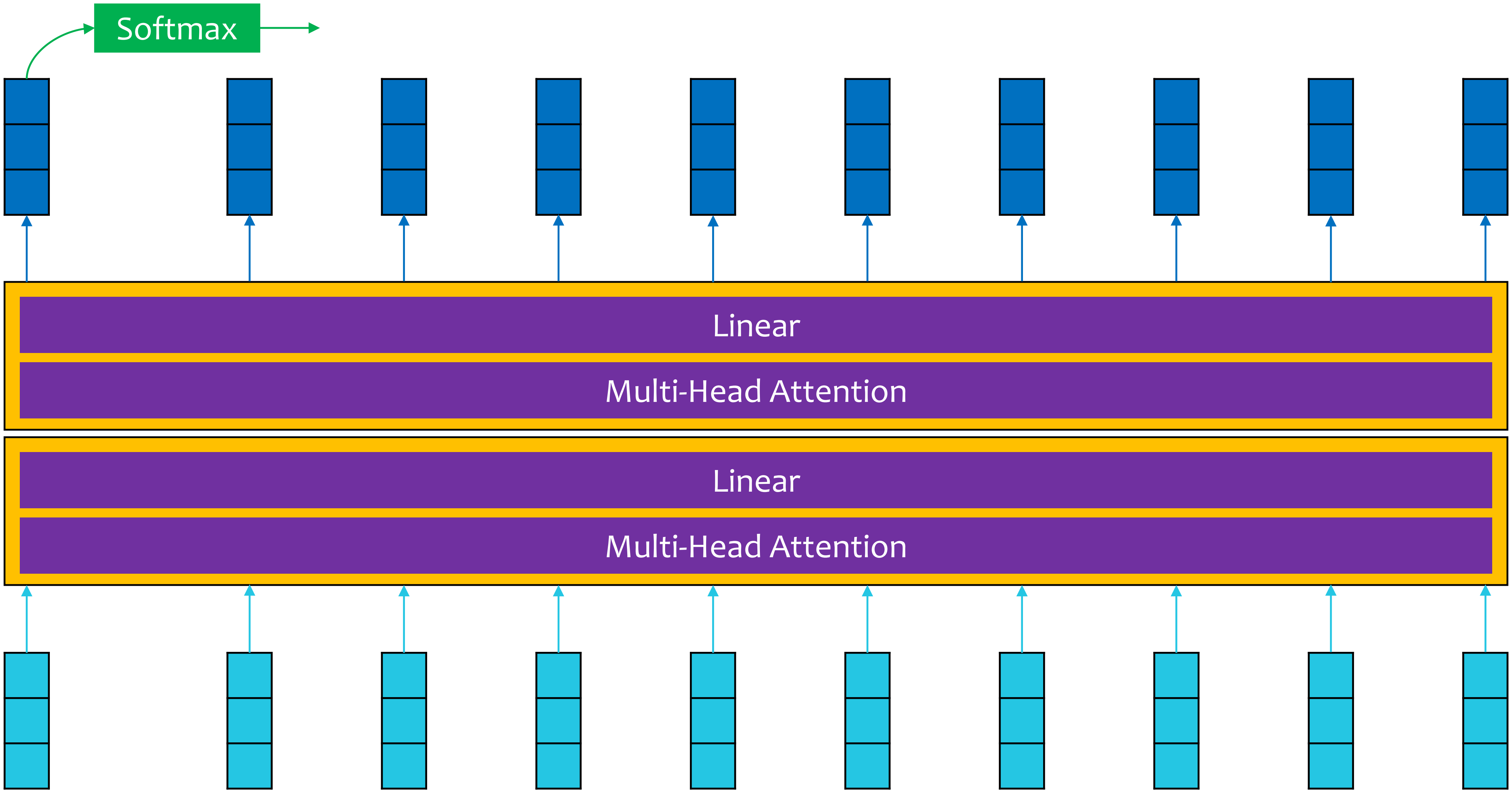
ViT


Vision Transformer (ViT)
Class
Bird
Ball
Car
...
MLP
Head
Transformer Encoder
Patch + Position
Embedding
* Extra learnable
[class] embedding
0
*
1
2
3
4
5
6
7
8
9
Linear Projection of Flattened Patches
Transformer Encoder
L ×
MLP
Norm
Multi-Head
Attention
Norm
Embedded
Patches

ViT



ViT



Softmax
Linear
Multi-Head Attention
Linear
Multi-Head Attention

ViT Performance

- Pretrained on
  - ImageNet (1.3 M): ViT is slightly worse than ResNet
  - ImageNet-22k (14 M): ViT is comparable to ResNet
  - JFT (300 M): ViT is slightly better than ResNet

Google only

- Scaling Vision Transformers (2021 by Google)

Improve ImageNet top-1 from 88.55% to 90.45%
By using JFT 3B!!!

- Model soups (2022 by Google)

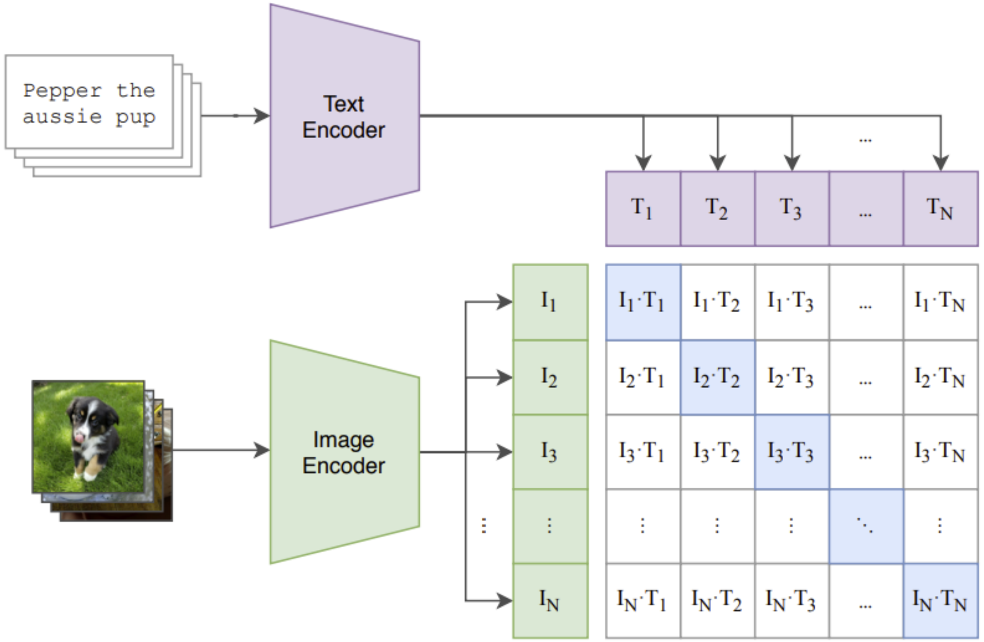
Improve ImageNet top-1 from 90.45% to 90.94%
By using JFT 3B and ensemble!!!

Vision-Language models

Parinya Sanguansat

CLIP

- Contrastive Language–Image Pre-training from OpenAI
  - Trained by 400 million image-text pairs from internet
- Contrastive Learning: Positive and Negative pairs
- Zero-shot Classification

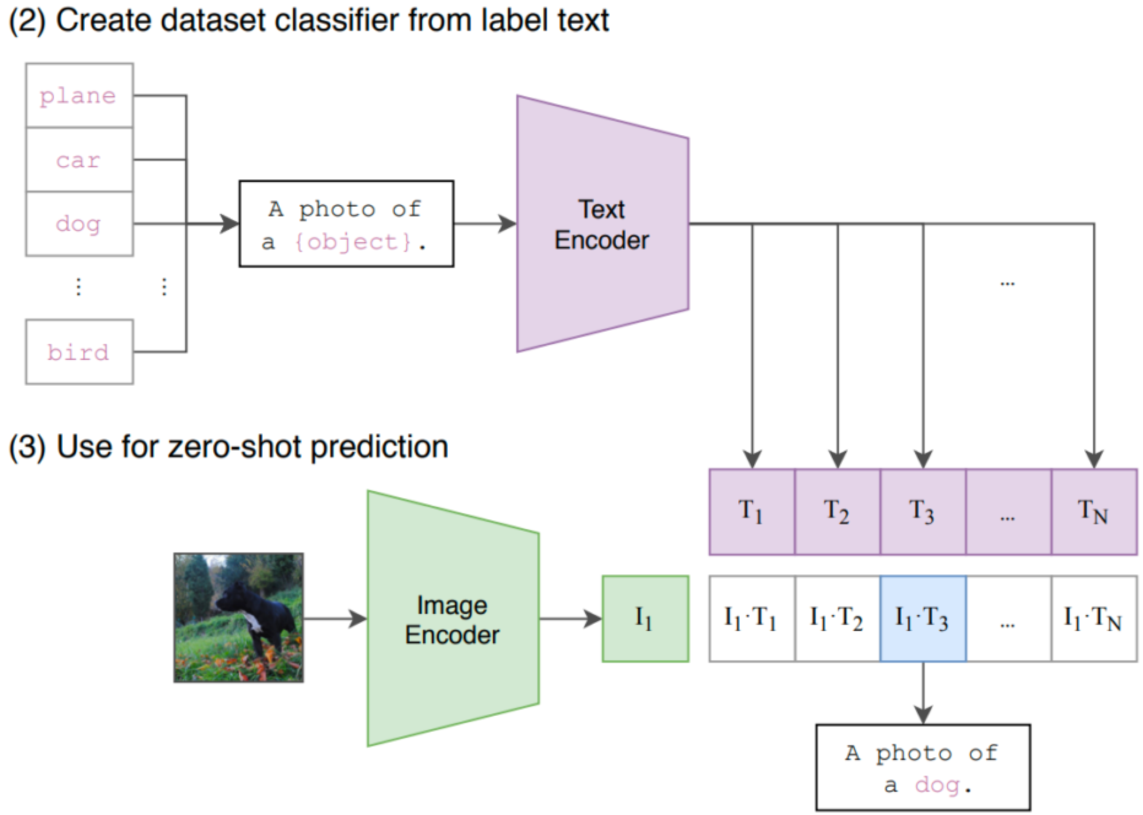


$$\mathcal{L}_k = -log \frac{e^{sim\,(I_k,T_k)/\tau}}{\sum_{j=0}^{N} e^{sim\,(I_k,T_j)/\tau}}$$

Lower τ: faster convergence but overfitting
Higher τ: slower convergence but more generalize

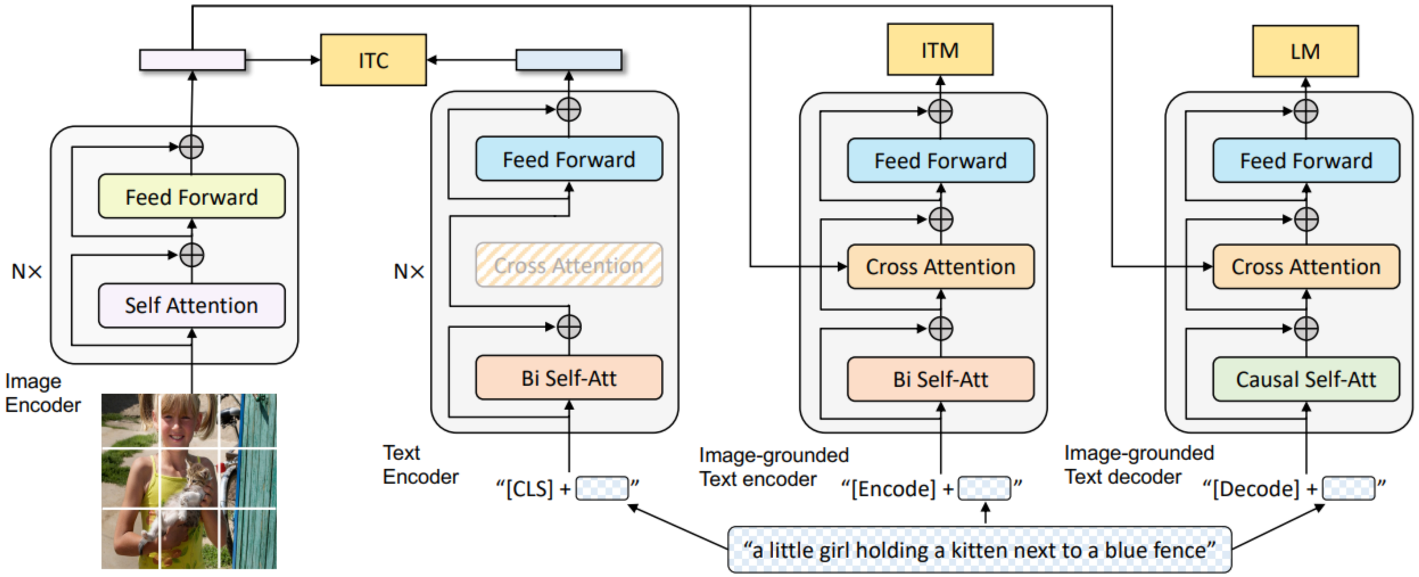
Zero-shot classification

- Classify data into classes that it has never explicitly been trained on.
- Input: 1 image + n label text
- Dynamic Categories



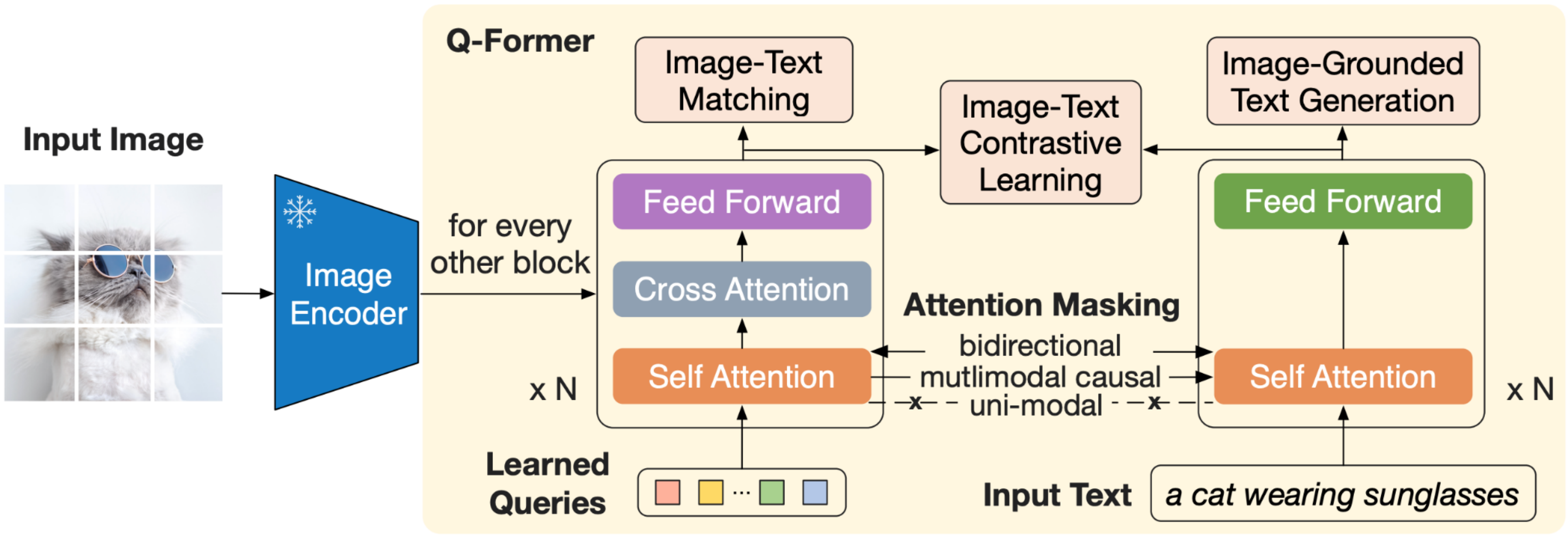
BLIP

- CapFit: address the noisy internet data
  - captioner model to generate synthetic captions
  - Filter model to remove noisy image-text pairs
- Image captioning



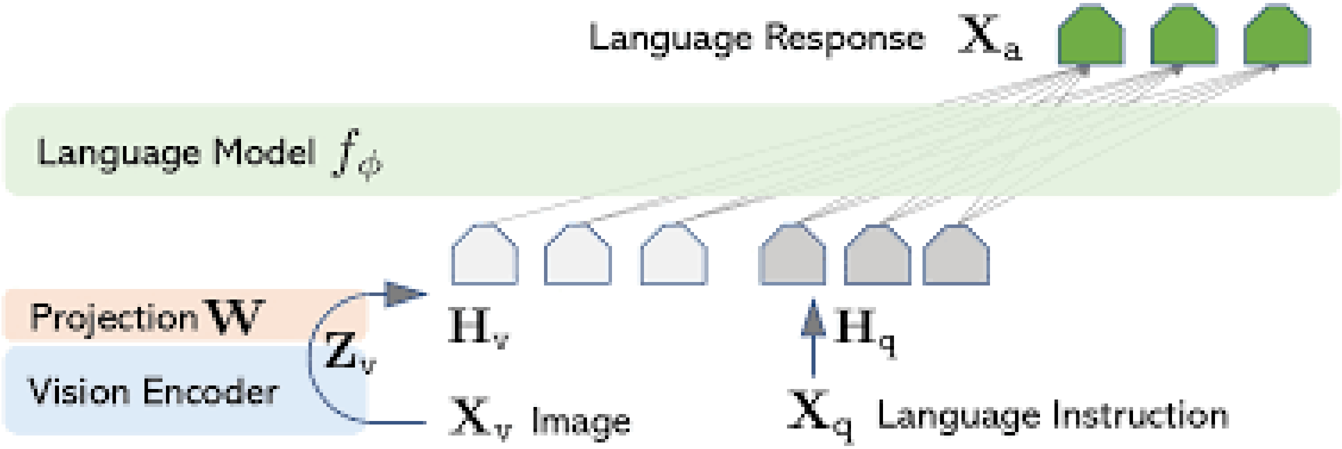
BLIP 2

- Q-former: extract relevant visual information from a frozen image encoder and prepare it for a frozen LLM
- Better than BLIP



LLaVA

- Large Language and Vision Assistant
  - project the image into the same embedding space as the text embeddings of LLM
- Conversation about the image



BLIP 2 vs LLaVA

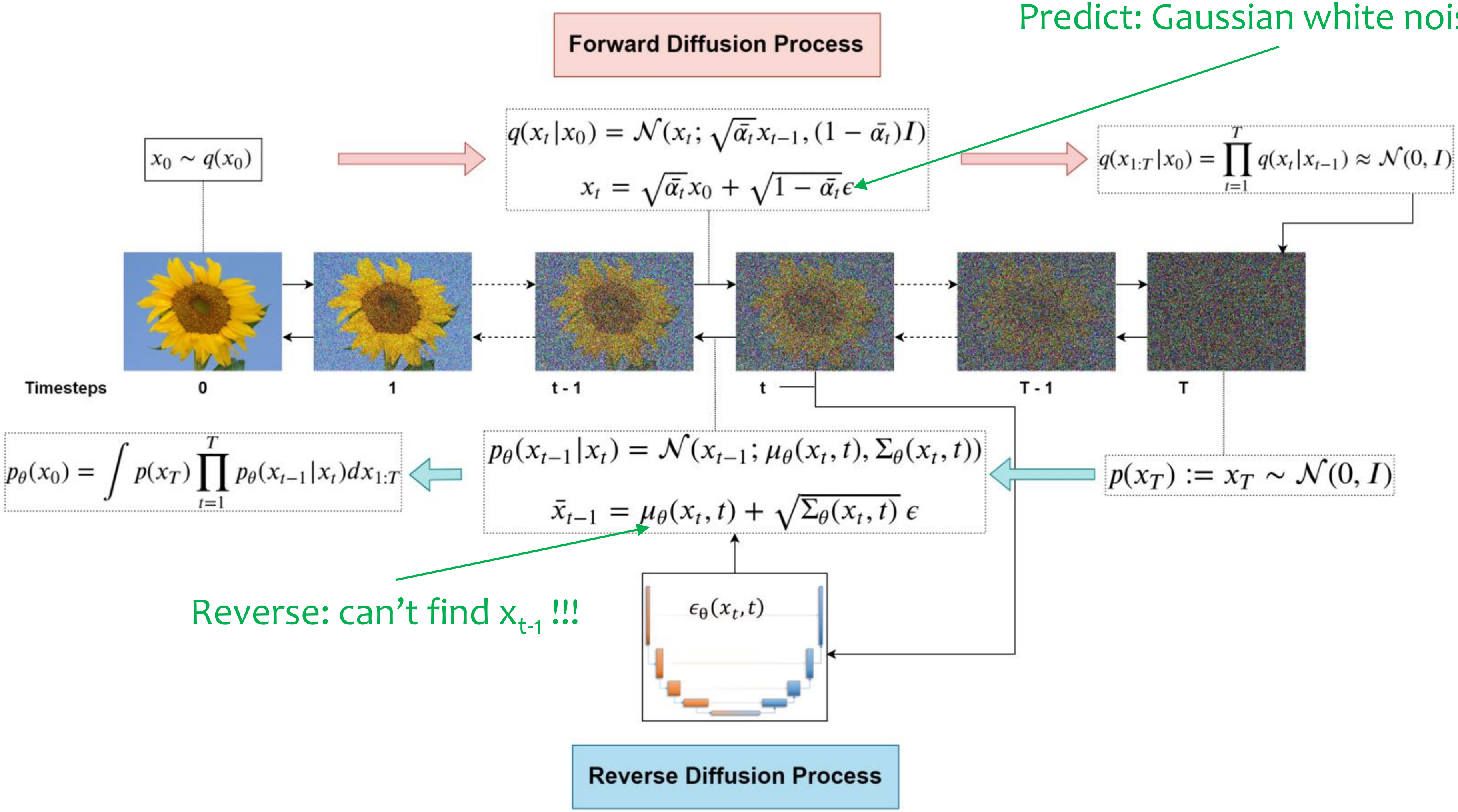
	BLIP 2	LLaVA
Input	Image (and optional text prompt)	Image and text prompt
LLM interactor	Q-former (Transformer)	Projection Layer (MLP)
Input to LLM	Fix tokens	Variable length
LLM training	Frozen	Trainable
Visual Question Answering (VQA)	** (pure)	** (conversational)
Image Captioning	**	*
Image-Text Retrieval	**	*
Interactive image-based chatbots	*	***
Zero-Shot Image-to-Text Generation	**	*
Changing the LLM	Train only Q-former	LLM need to be retrained

Diffusion Model

Parinya Sanguansat

Denoising Diffusion Probabilistic Models

Predict: Gaussian white noise !!!



Reverse: can't find x_{t-1} !!!

The model is trained to predict stationary white noise, but the prediction itself is not stationary because it's conditioned on a specific image and timestep.

https://learnopencv.com/denoising-diffusion-probabilistic-models/

How to generate noisy image

$$x_t \sim \mathcal{N}(\sqrt{\alpha_t} x_{t-1}, (1-\alpha_t)\boldsymbol{I})$$

$$x_t = \sqrt{\alpha_t} x_{t-1} + \sqrt{1-\alpha_t}\epsilon_{t-1}$$

reparameterization trick

$$x_{t-1} = \sqrt{\alpha_{t-1}} x_{t-2} + \sqrt{1-\alpha_{t-1}}\epsilon_{t-2}$$

$$x_t = \sqrt{\alpha_t}\left(\sqrt{\alpha_{t-1}} x_{t-2} + \sqrt{1-\alpha_{t-1}}\epsilon_{t-2}\right) + \sqrt{1-\alpha_t}\epsilon_{t-1}$$

$$x_t = \sqrt{\alpha_t}\sqrt{\alpha_{t-1}} x_{t-2} + \sqrt{\alpha_t(1-\alpha_{t-1})}\epsilon_{t-2} + \sqrt{1-\alpha_t}\epsilon_{t-1}$$

$$\epsilon_{t-1}, \epsilon_{t-2} \sim \mathcal{N}(0, \boldsymbol{I})$$

Mean = $\sqrt{\alpha_t}\sqrt{\alpha_{t-1}} x_{t-2} + 0 \qquad + 0 \qquad = \sqrt{\alpha_t \alpha_{t-1}} x_{t-2}$

Var = $0^2 \qquad + \sqrt{\alpha_t(1-\alpha_{t-1})}^2 \cdot 1^2 + \sqrt{1-\alpha_t}^2 \cdot 1^2 \qquad = 1 - \alpha_t \alpha_{t-1}$

$\vdots$

$$x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon \qquad \bar{\alpha}_t = \alpha_t \alpha_{t-1} \dots \alpha_1$$

How to reverse diffusion

$$x_{t-1} \sim \mathcal{N}\left(\tilde{\mu}(x_t,t), \tilde{\beta}_t \boldsymbol{I}\right)$$

Bayes' rule

Markov

$$\mathcal{N}\left(\tilde{\mu}(x_t,t), \tilde{\beta}_t \boldsymbol{I}\right) = q(x_{t-1}|x_t,x_0) = q(x_t|x_{t-1},x_0)\frac{q(x_{t-1}|x_0)}{q(x_t|x_0)} = q(x_t|x_{t-1})\frac{q(x_{t-1}|x_0)}{q(x_t|x_0)}$$

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{\alpha_t}x_{t-1}, (1-\alpha_t)\boldsymbol{I}) \qquad x_t = \sqrt{\alpha_t}x_{t-1} + \sqrt{1-\alpha_t}\epsilon_{t-1}$$

$$q(x_t|x_0) = \mathcal{N}\left(x_t; \sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)\boldsymbol{I}\right) \qquad x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon_t$$

$$q(x_{t-1}|x_0) = \mathcal{N}\left(x_{t-1}; \sqrt{\bar{\alpha}_{t-1}}x_0, (1-\bar{\alpha}_{t-1})\boldsymbol{I}\right) \qquad x_{t-1} = \sqrt{\bar{\alpha}_{t-1}}x_0 + \sqrt{1-\bar{\alpha}_{t-1}}\epsilon$$

Gaussian

$$q(x_t|x_{t-1})\frac{q(x_{t-1}|x_0)}{q(x_t|x_0)} \propto exp\left(-\frac{1}{2}\left(\frac{\left(x_t-\sqrt{\alpha_t}x_{t-1}\right)^2}{1-\alpha_t} + \frac{\left(x_{t-1}-\sqrt{\bar{\alpha}_t}x_0\right)^2}{1-\bar{\alpha}_{t-1}} - \frac{\left(x_t-\sqrt{\bar{\alpha}_{t-1}}x_0\right)^2}{1-\bar{\alpha}_t}\right)\right)$$

$$= exp\left(-\frac{1}{2}\left(\left(\frac{\alpha_t}{1-\alpha_t} + \frac{1}{1-\bar{\alpha}_{t-1}}\right)x_{t-1}^2\right) - 2x_{t-1}\left(\frac{2\sqrt{\alpha_t}}{1-\alpha_t}x_t + \frac{2\sqrt{\bar{\alpha}_{t-1}}}{1-\bar{\alpha}_{t-1}}x_0\right) + C(x_t,x_0)\right)$$

completing the square $\frac{(x-\mu)^2}{\sigma}$

$$\tilde{\mu}(x_t,t) = \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}x_t + \frac{\sqrt{\bar{\alpha}_{t-1}},(1-\alpha_t)}{1-\bar{\alpha}_t}x_0 = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_t\right)$$

Training and Sampling

$$\tilde{\mu}(x_t, t) = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_t\right)$$

$$\boxed{L_t^{simple} = MSE\big(\epsilon_t, \epsilon_\theta(x_t, t)\big)}$$

$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t, t)\right)$$

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \mathrm{Uniform}(\{1, \ldots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$
6: **until** converged

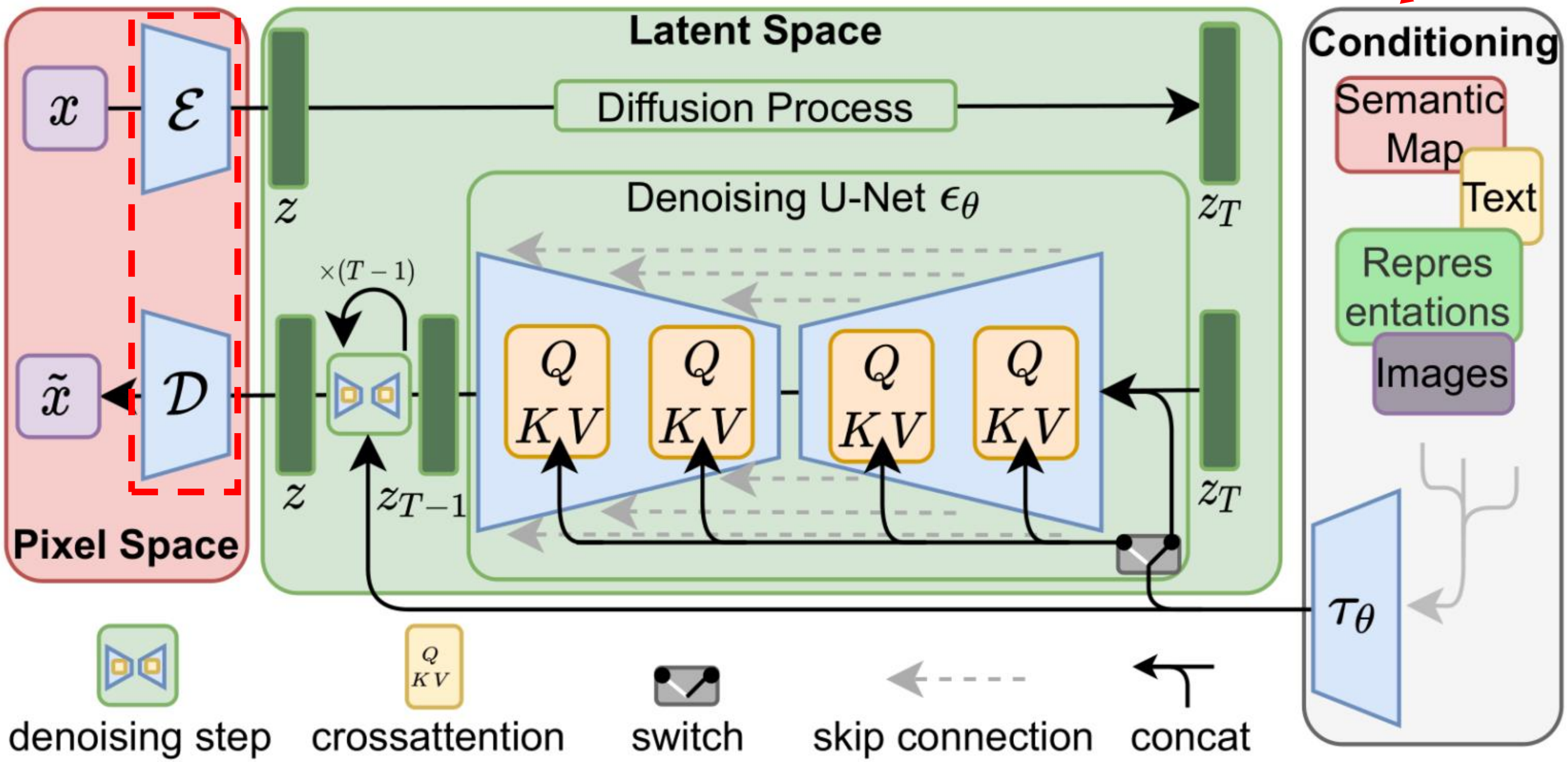
Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \ldots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

Stable Diffusion

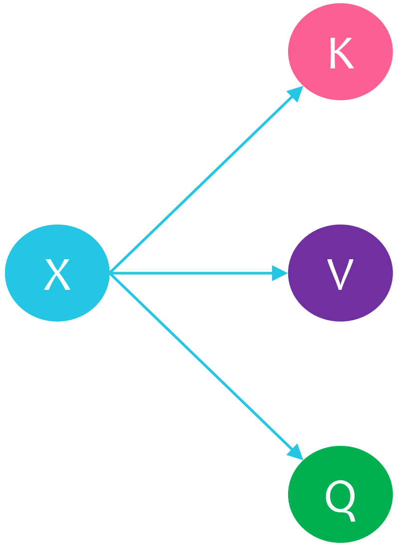
DDPM := Pixel Space with unconditional

VAE

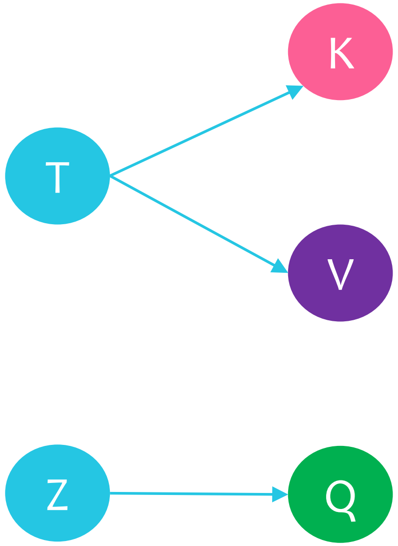


CLIP (text encoder)

Cross Attention in Stable Diffusion



Self Attention



Cross Attention

Training Optimization

Parinya Sanguansat

Optimization Techniques

- Data Preprocessing and Augmentation
- Optimizer and Hyperparameter tuning
- Loss Functions
- Transfer Learning and Fine-tuning
- Distributed Training
- Optimization for heterogeneous Hardware

Image Data Augmentation

- **Geometric Transformations:**
  - **Rotation:** Rotating images by various angles.
  - **Flipping:** Mirroring images horizontally or vertically.
  - **Scaling:** Resizing images, zooming in or out.
  - **Translation:** Shifting images horizontally or vertically.
  - **Cropping:** Extracting random or central portions of images.
  - **Shearing:** Distorting the shape of images.
- **Color Space Transformations:**
  - **Brightness Adjustment:** Altering the overall brightness of images.
  - **Contrast Adjustment:** Modifying the difference between light and dark areas.
  - **Saturation Adjustment:** Changing the intensity of colors.
  - **Color Jittering:** Randomly varying brightness, contrast, and saturation.
- **Noise Injection:**
  - **Gaussian Noise:** Adding random noise with a Gaussian distribution.
  - **Salt-and-Pepper Noise:** Introducing random black and white pixels.
- **Kernel Filters:**
  - Applying blurring or sharpening filters.
- **Random Erasing:**
  - Randomly masking out rectangular regions of images.
- **Mixup and CutMix:**
  - Combining pixels from different images to create new samples.

Text Data Augmentation

- **Synonym Replacement:**
  - Replacing words with their synonyms.
- **Random Insertion/Deletion:**
  - Adding or removing words randomly.
- **Word Shuffling:**
  - Rearranging the order of words in a sentence.
- **Back Translation:**
  - Translating text to another language and back.
- **Text generation:**
  - Using models to create new sentences that have the same meaning.

Audio Data Augmentation

- **Noise Injection:**
  - Adding background noise or white noise.
- **Time Stretching:**
  - Speeding up or slowing down audio.
- **Pitch Shifting:**
  - Changing the pitch of audio signals.
- **Time Shifting:**
  - Shifting audio signals forward or backward in time.
- **Volume Adjustment:**
  - Increasing or decreasing the volume of audio.

Optimizers

Adagrad (2011)

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{v_{t+1}} + \epsilon} \nabla J(\theta_t)$$

$$v_{t+1} = v_t + \left(\nabla J(\theta_t)\right)^2$$

accumulate

SGD (1950s)

$$\theta_{t+1} = \theta_t - \eta \nabla J(\theta_t)$$

Adadelta (Dec 2012)

$$\theta_{t+1} = \theta_t \boxed{- \frac{\sqrt{u_t + \epsilon}}{\sqrt{v_{t+1}} + \epsilon} \nabla J(\theta_t)}$$

$$u_{t+1} = \gamma u_t + (1-\gamma)(\Delta\theta_t)^2$$

RMSprop (2012)

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{v_{t+1}} + \epsilon} \nabla J(\theta_t)$$

$$v_{t+1} = \beta v_t + (1-\beta)\left(\nabla J(\theta_t)\right)^2$$

Momentum (1980s)

$$\theta_{t+1} = \theta_t - \eta m_{t+1}$$

$$m_{t+1} = \alpha m_t + (1-\alpha)\nabla J(\theta_t)$$

$$\hat{v}_{t+1} = \frac{v_{t+1}}{1-\beta^{t+1}}$$

Bias correction

$$\hat{m}_{t+1} = \frac{m_{t+1}}{1-\alpha^{t+1}}$$

Adam (2014)

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_{t+1}} + \epsilon} \hat{m}_{t+1}$$

Optimizers

- **Stochastic Gradient Descent (SGD):**
  - A basic but widely used optimizer. It updates parameters based on the gradient of the loss function calculated on a single or small batch of training examples.
  - While simple, it can be prone to oscillations and slow convergence.
- **Momentum:**
  - An extension of SGD that adds a "momentum" term to the parameter updates. This helps to accelerate convergence and reduce oscillations by incorporating information from previous updates.
- **Adagrad (Adaptive Gradient Algorithm):**
  - This algorithm adapts the learning rate to the parameters, performing larger updates for infrequent parameters, and smaller updates for frequent parameters.
- **RMSprop (Root Mean Square Propagation):**
  - An adaptive learning rate optimizer that adjusts the learning rate for each parameter based on the magnitude of recent gradients.
  - It performs well in many scenarios, particularly when dealing with noisy or sparse gradients.
- **Adadelta (Adaptive Delta):**
  - An extension of Adagrad that seeks to reduce Adagrad's aggressively decreasing learning rates.
- **Adam (Adaptive Moment Estimation):**
  - A popular and effective optimizer that combines the advantages of momentum and RMSprop.
  - It adapts the learning rate for each parameter and is generally robust and efficient.

Hyperparameters

Learning Rate:

- **Effect:**
  - Controls the step size during parameter updates.
  - A high learning rate can lead to rapid but unstable convergence, potentially overshooting the optimal solution.
  - A low learning rate can result in slow convergence, requiring more training time.
  - Finding the right balance is essential for efficient learning.
- **Impact:**
  - Directly affects how quickly and accurately the model learns.

Hyperparameters

Batch Size:

- **Effect:**
  - Determines the number of training examples used in each iteration.
  - Small batch sizes introduce more noise into the gradient estimation, which can help the model escape local minima but may also lead to instability.
  - Large batch sizes provide more stable gradients but may require more memory and can lead to slower convergence.
- **Impact:**
  - Influences training speed, memory usage, and model stability.

Hyperparameters

Number of Epochs:

- **Effect:**
  - Specifies the number of times the entire training dataset is passed through the model.
  - Too few epochs can result in underfitting, where the model fails to learn the underlying patterns.
  - Too many epochs can lead to overfitting, where the model memorizes the training data and performs poorly on unseen data.
- **Impact:**
  - Determines how well the model learns the training data and its ability to generalize.

Hyperparameters

Number of Hidden Units/Layers (Neural Networks):

- **Effect:**
  - Controls the complexity of the neural network.
  - More hidden units and layers allow the model to learn more complex patterns but increase the risk of overfitting.
  - Fewer hidden units and layers may limit the model's ability to capture complex relationships.
- **Impact:**
  - Affects the model's capacity to learn complex relationships.

Hyperparameters

Regularization Strength (e.g., L1, L2):

- **Effect:**
  - Helps prevent overfitting by adding a penalty term to the loss function.
  - Higher regularization strength reduces model complexity but can also lead to underfitting if it's too strong.
- **Impact:**
  - Controls the model's complexity and its ability to generalize.

Hyperparameters

Momentum:

- **Effect:**
  - Accelerates gradient descent by adding a momentum term that incorporates information from previous updates.
  - Helps the model overcome local minima and converge faster.
- **Impact:**
  - Improves convergence speed and stability.