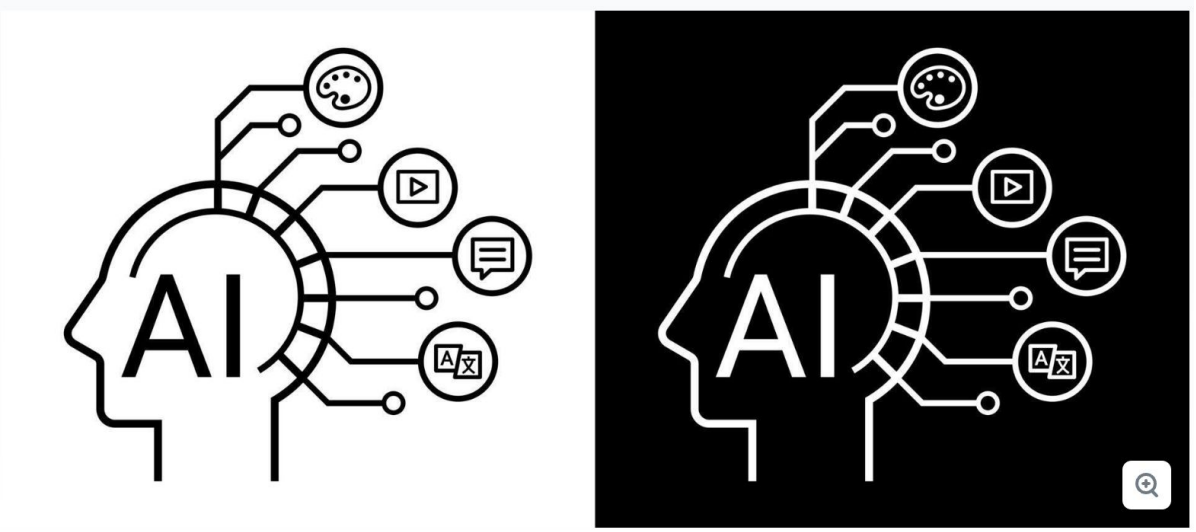
AIMS 5740
Generative Artificial Intelligence

(2026 Term 2)

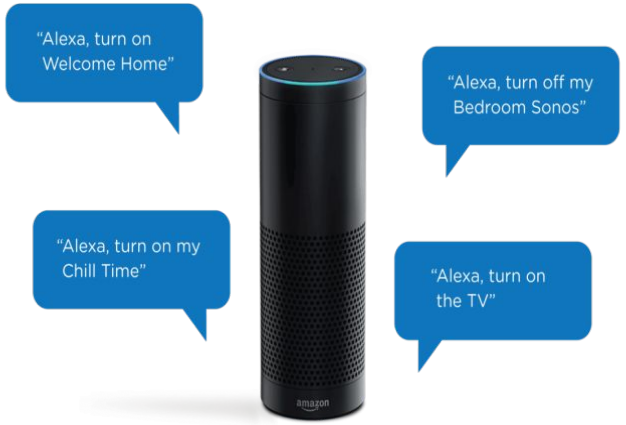


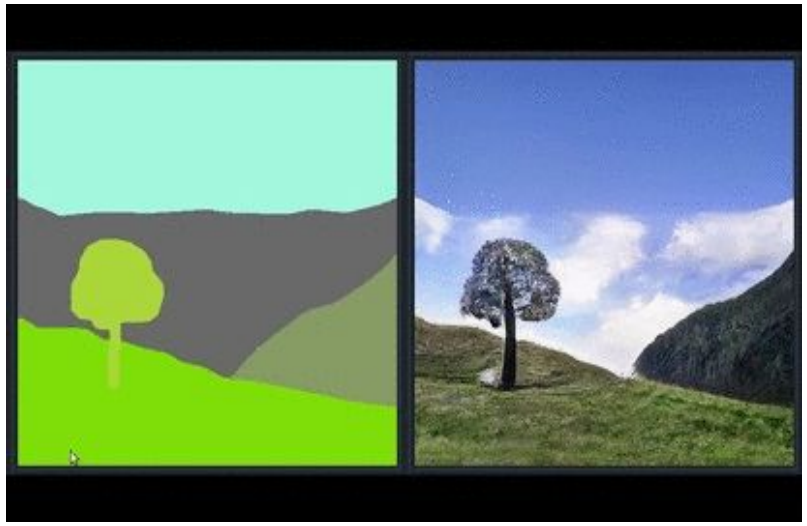
Computer Science & Engineering
The Chinese University of Hong Kong

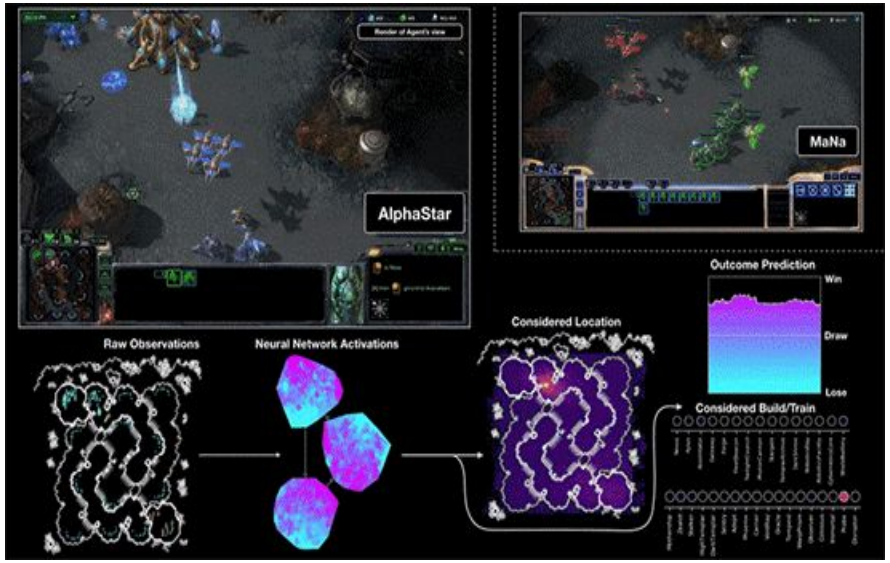
Announcement

- Please check the website and remember to register your Piazza account.
- Our first assignment will be released on Feb. 5.
- Our mid-term quiz is scheduled on Mar. 18.
- Each student will have 350 hours to use NV L4 on Colab.

AI Applications



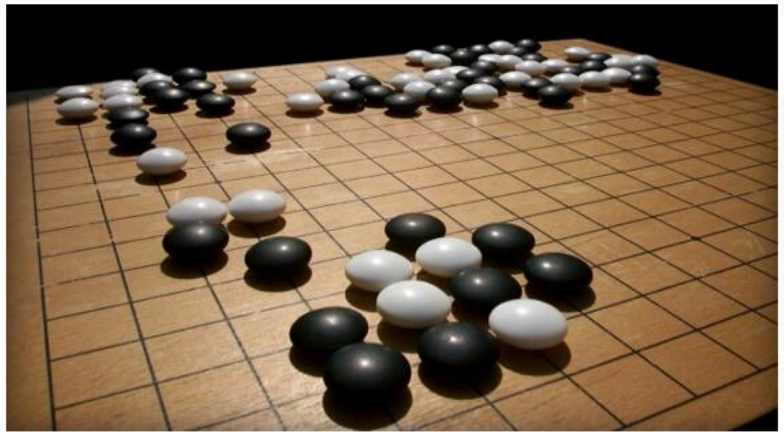




AI Applications

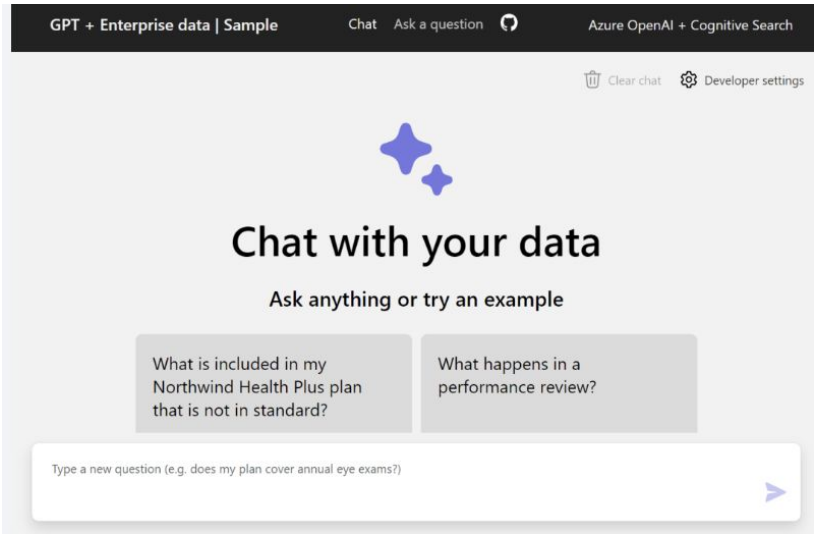
Google's DeepMind AI beats humans at the massively complex game Go

By Ryan Whitwam on January 27, 2016 at 4:00 pm | 11 Comments



AI systems are beating humans in reading comprehension

By Associated Press January 24, 2018 | 2:25pm

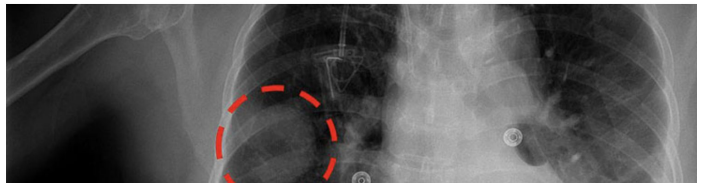


More Applications

Does AI Help or Hurt Human Radiologists' Performance? It Depends on the Doctor

New research shows radiologists and AI don't always work v together

By EKATERINA PESHEVA | March 19, 2024 | Research
4 min read



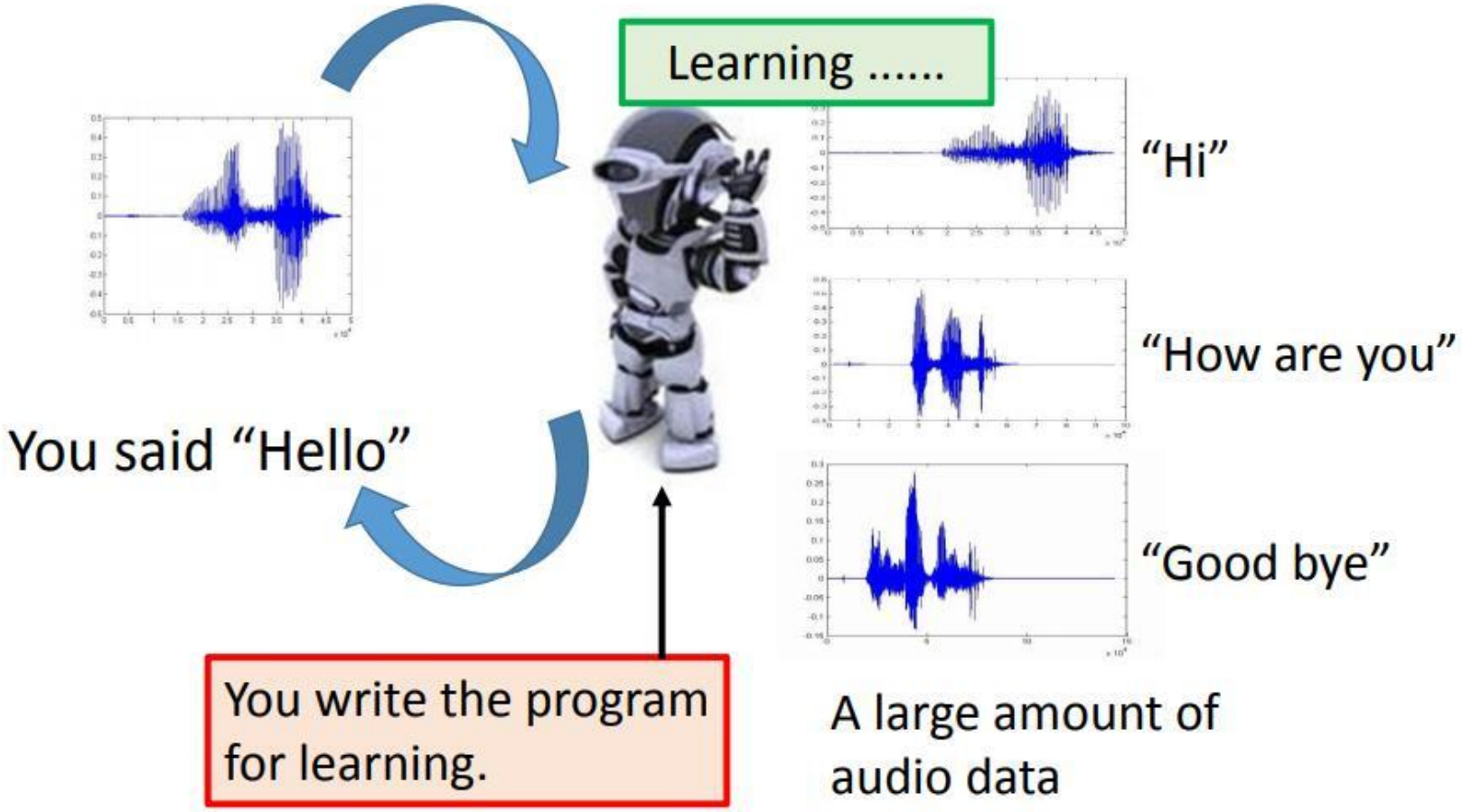
AI can pick stocks better than you can

Why wouldn't it?

BT Robert Armstrong
Published Fri, Jun 14, 2024 · 08:00 AM

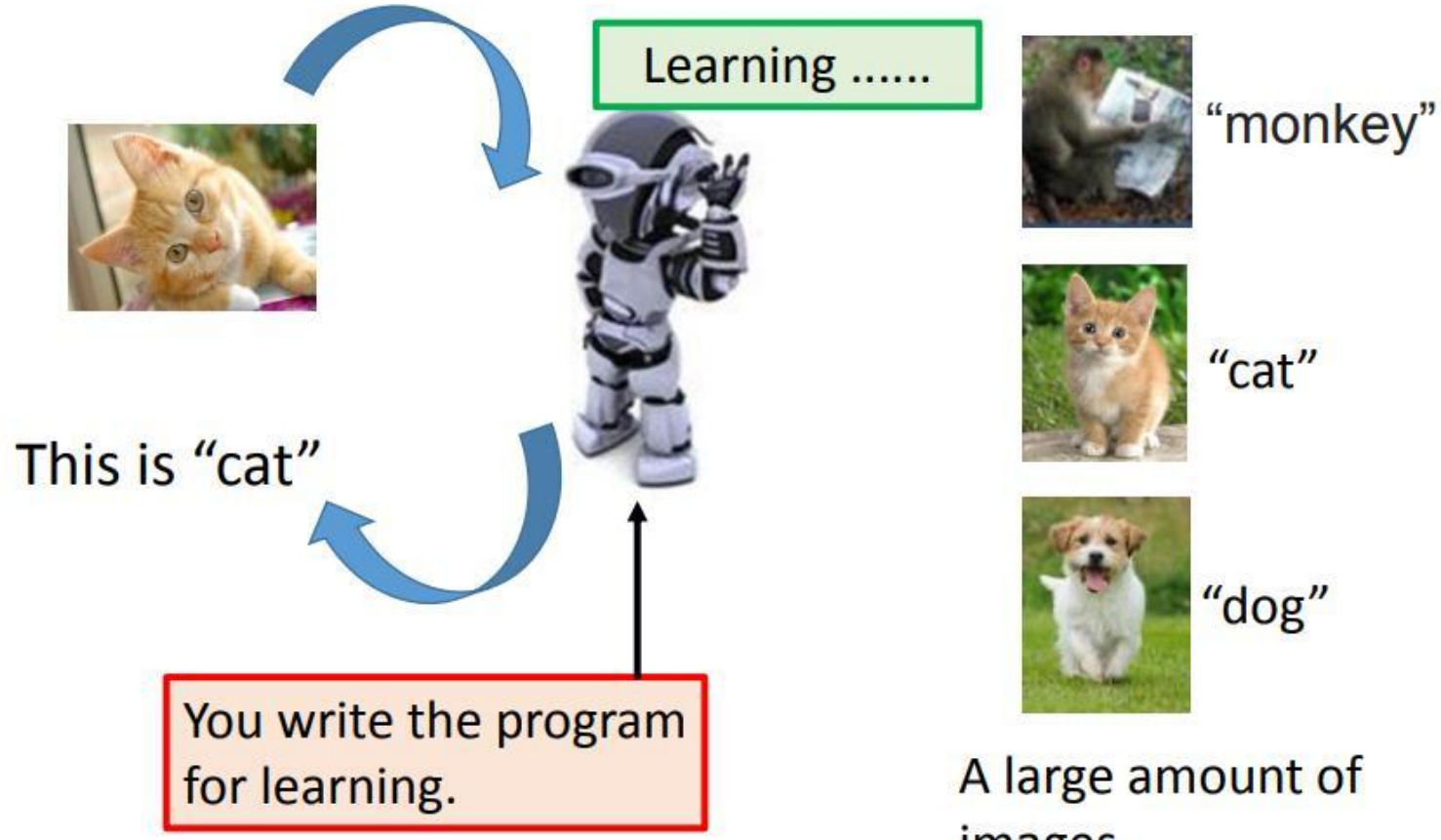
Artificial Intelligence Follow

What is Machine Learning?



What is Machine Learning?

Learning is not “memorization” – generalization.



Definition of Machine Learning

"Using a set of observations to uncover an underlying process"

- Yaser Abu-Mostafaz

"The study of computer programs that improve automatically with experience"

- Tom M. Mitchell

"Algorithms for inferring unknowns from knowns"

Machine learning = looking for a function f()

An example of bank credit card approval.

Input: **X** (customer application)

Output: y (good/bad customer)

Target function: $f : X \rightarrow Y$ (ideal credit approval formula)

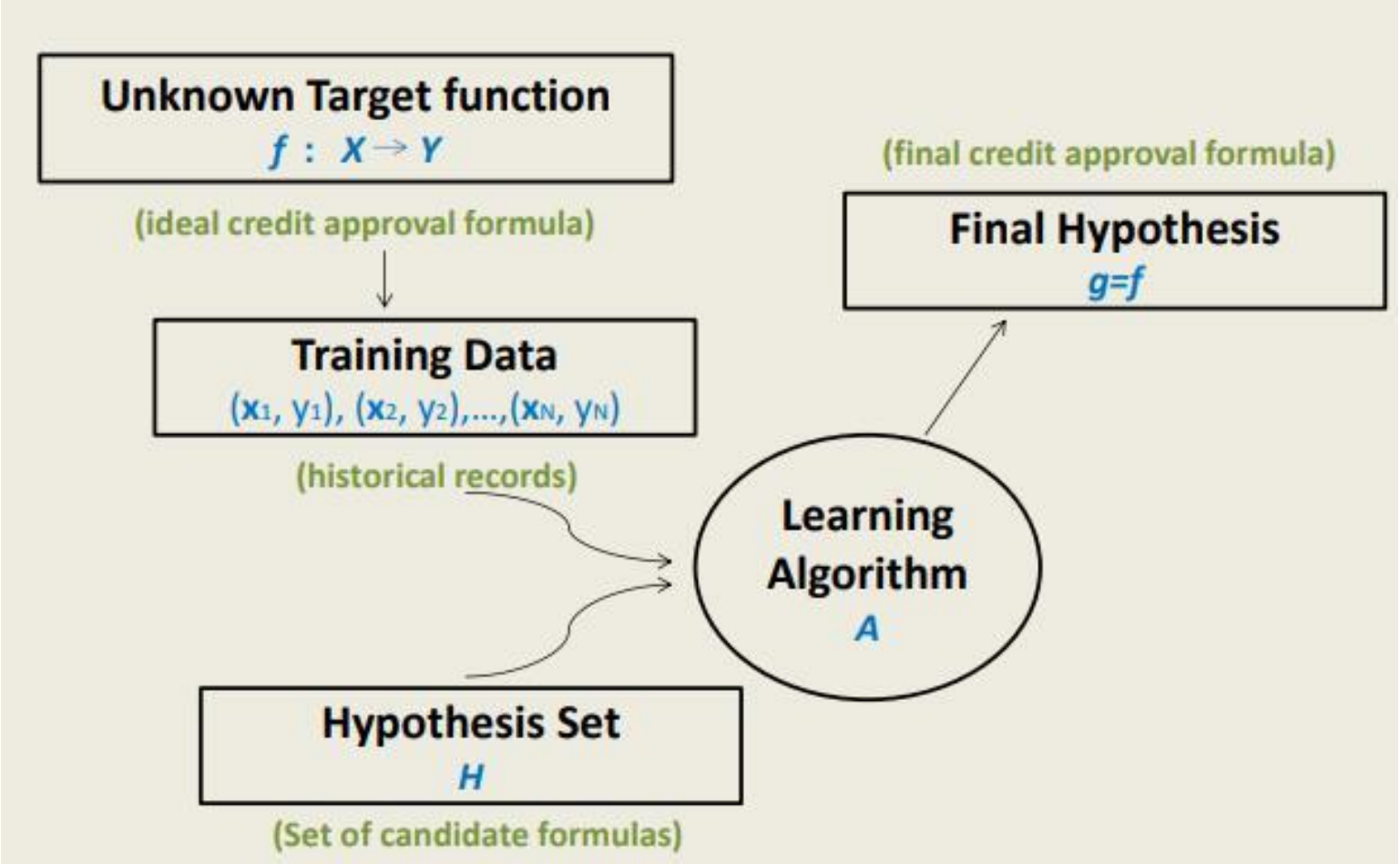
Data: $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)$ (historical records)

Hypothesis: $g : X \rightarrow Y$ (formula to be used)

Learning: find g that well approximates f

Components of Learning

An example of bank credit card approval.



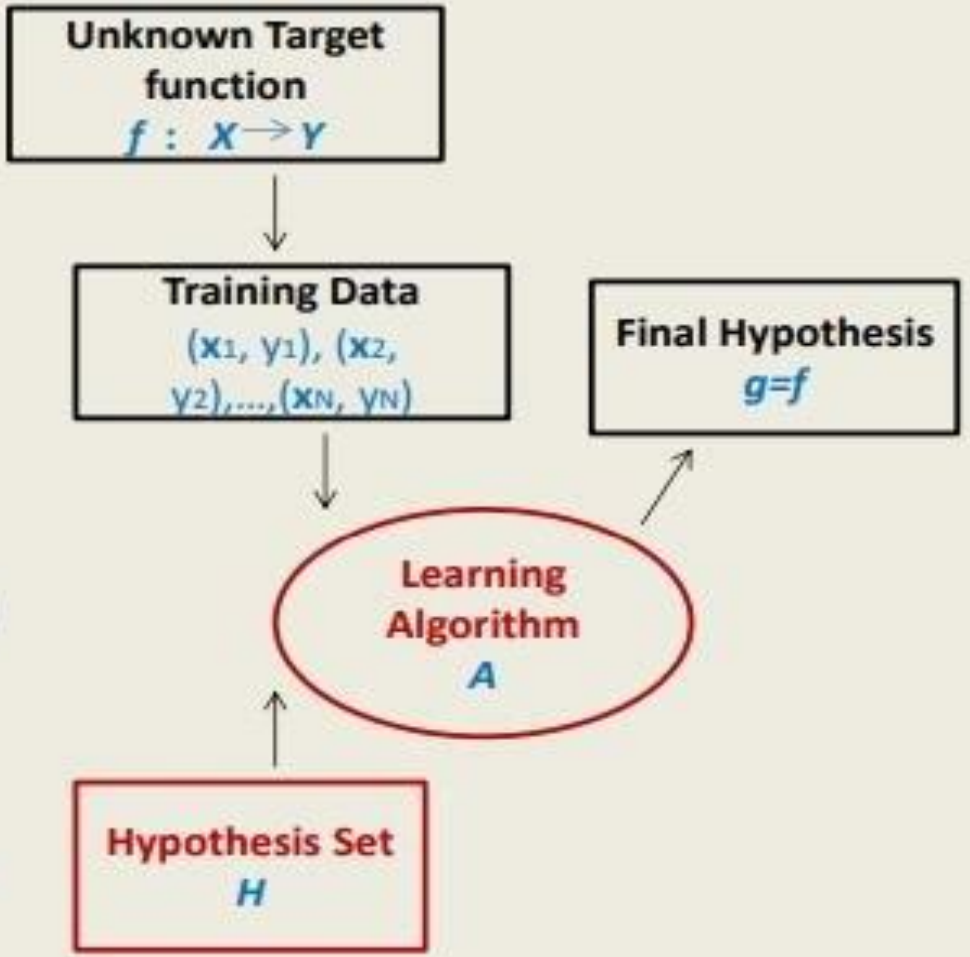
Solution Components

Solutions components of the learning problem:

- The hypothesis set
  $H = \{h\}$ $\quad g \in H$
- The learning algorithm

Together, they are referred to as the *learning model*.

A simple case – "perceptron"



The Perceptron

- For input, "customer attributes" $\mathbf{x} = \{x_1, \ldots, x_d\}$
  - Approve application if $\sum_{i=1}^{d} w_i x_i$ > threshold
  - Deny application if $\sum_{i=1}^{d} w_i x_i$ < threshold
- The formula $h \in H$ can be written as:

$$h(\mathbf{x}) = \text{sign}((\sum_{i=1}^{d} w_i x_i) - \text{threshold})$$

- Learning: given $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)$
  looking for the optimal $\mathbf{w} = \{w_1, \ldots, w_d\}$

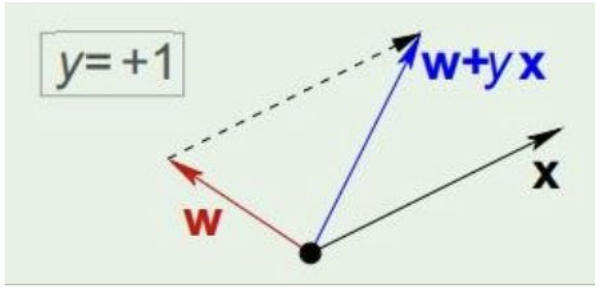
Simple Learning - PLA

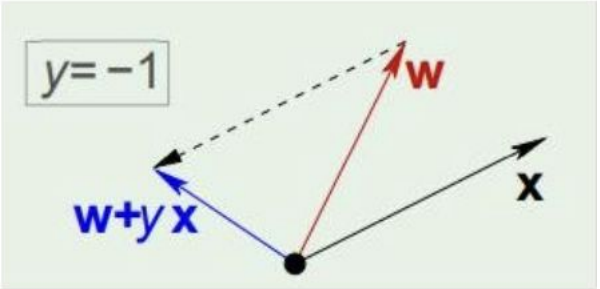
- Given $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)$ pick a misclassified point:

$$\text{sign}(\mathbf{w}^T \mathbf{x_i}) \neq y_i$$

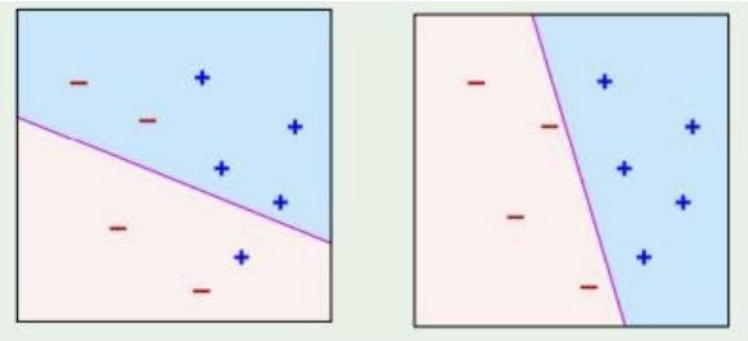
- Update the parameters by:

$$\mathbf{w}^{new} \leftarrow \mathbf{w}^{old} + \mathbf{x_i} y_i$$





- Assumption: “linearly separable”



Classification vs. Regression

- Classification: yes or no, Regression: credit line (amount)
- Given $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)$,$y_i \in \mathbf{R}$ and h() as:

$$h(\mathbf{x}) = \sum_{i=1}^{d} w_i x_i = \mathbf{w}^T \mathbf{x}$$

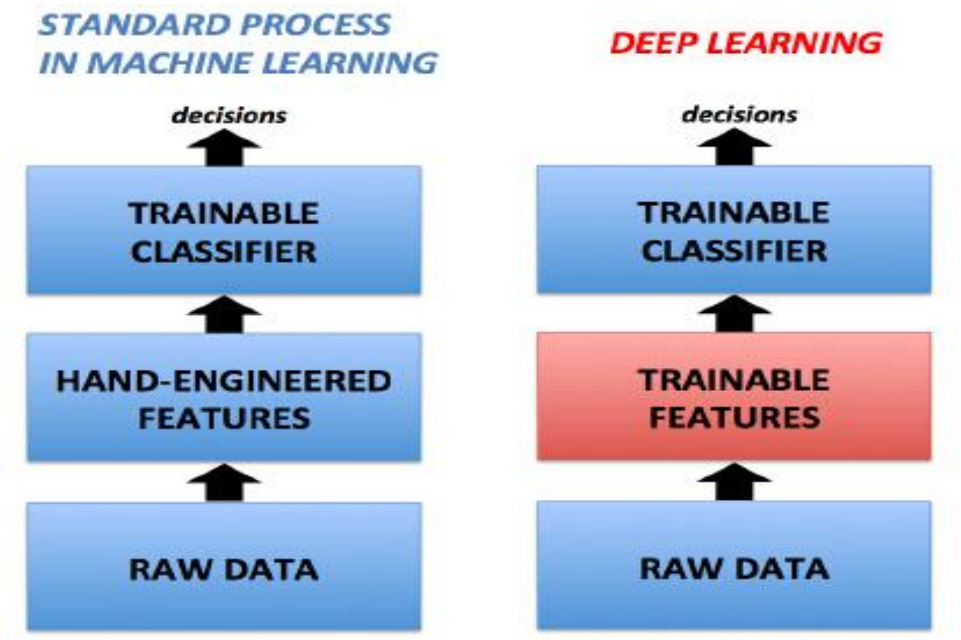
linear regression

- Loss function: square loss

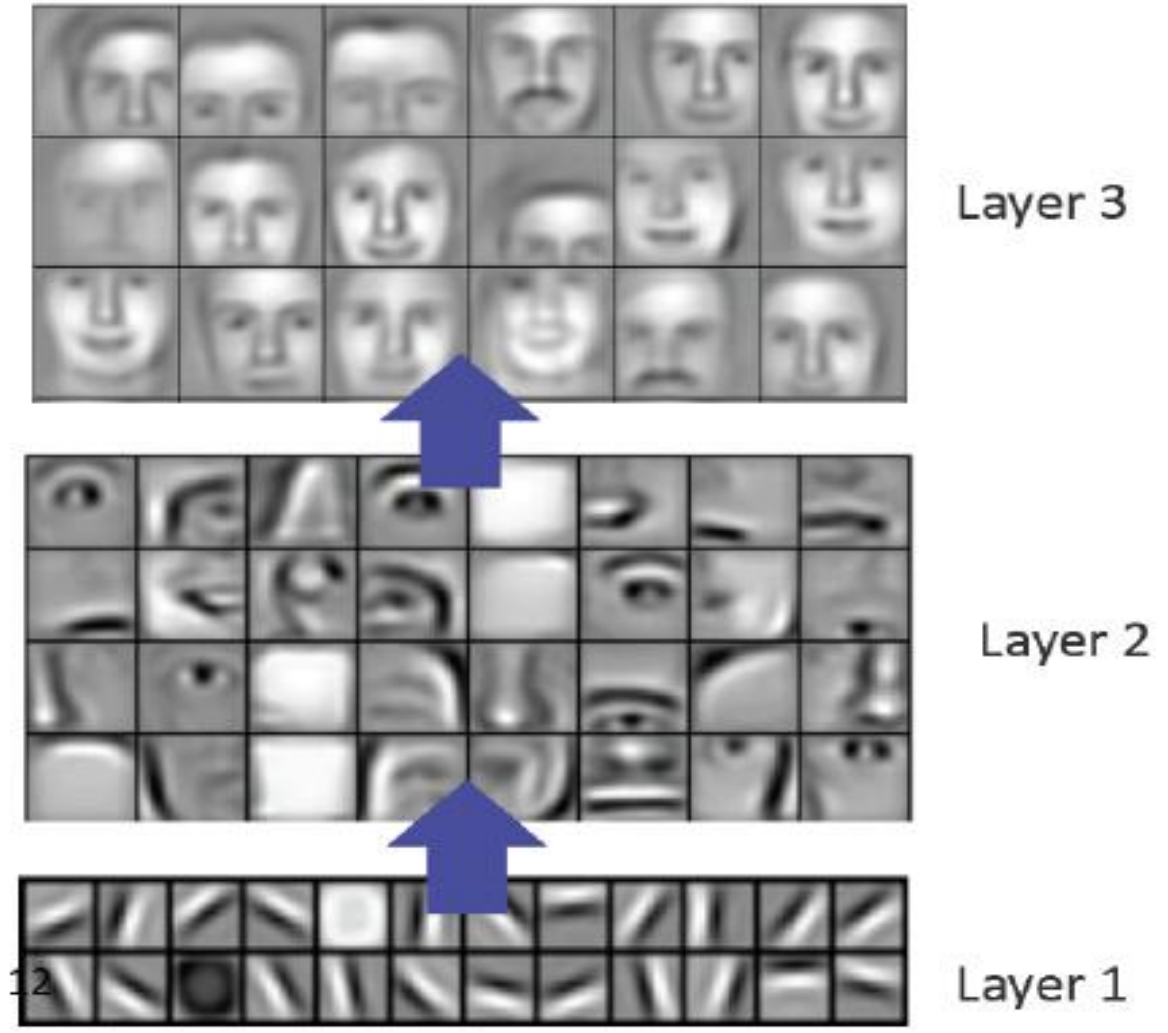
$$L(\mathbf{w}) = (h(\mathbf{x}) - f(\mathbf{x}))^2$$

What is Deep Learning?

Definition
A family of methods that uses deep architectures to learn high-level feature representations.



Example of Trainable Features

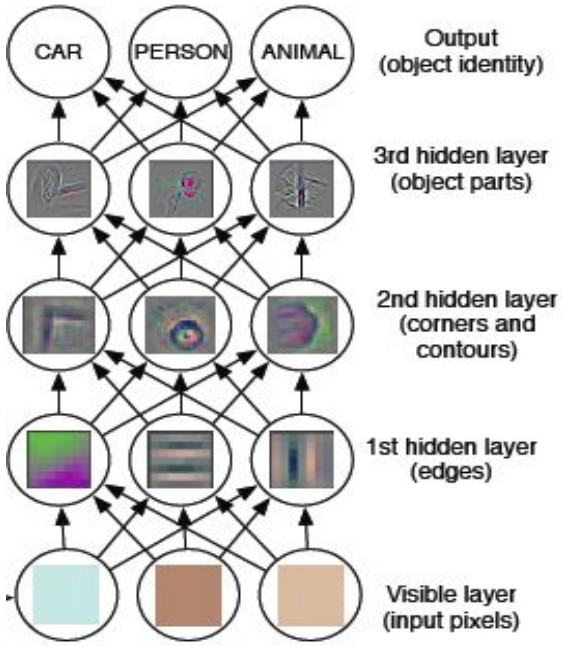


Why do we need “Deep”?

Why Deep Model?

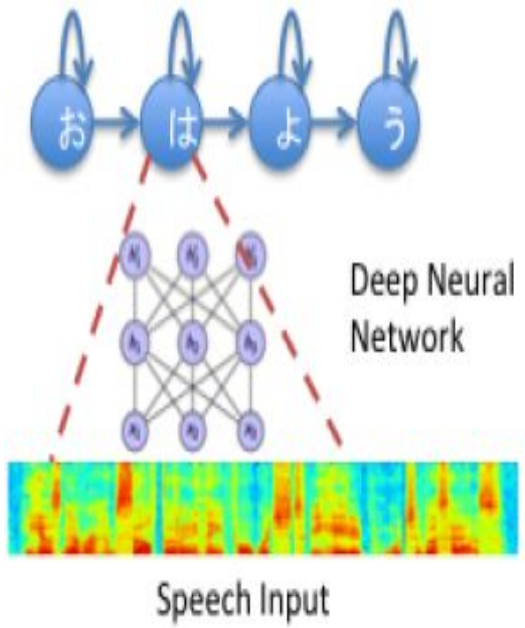
1) Fill in the gap between low-level feature and semantic meaning
2) Learn useful high-level abstraction with less variant/noise

object



pixels

sentence

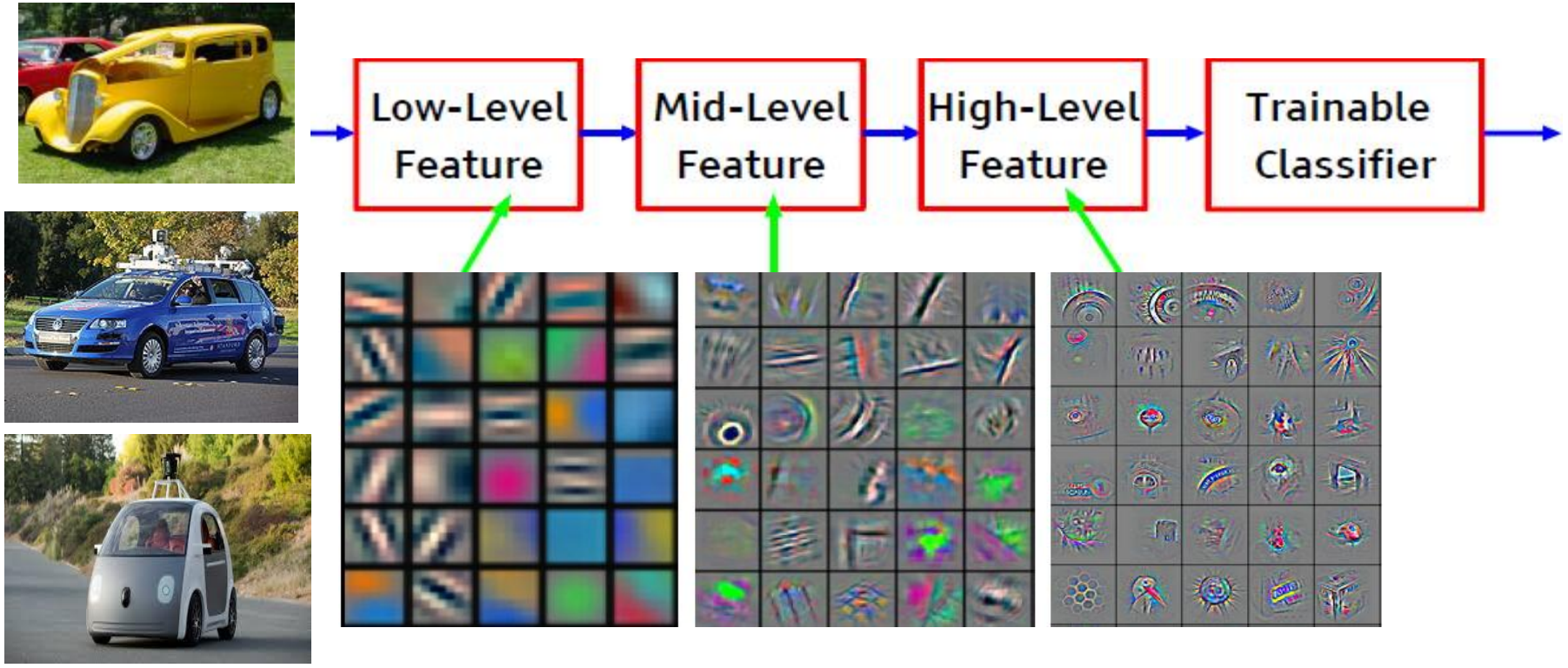


signals

Why do we need “Deep”?

Why Deep Model?

1) Fill in the gap between low-level feature and semantic meaning
2) Learn useful high-level abstractions with less variant/noise



Three Steps for Deep Learning

Step 1. A neural network is a function composed of simple functions (neurons)

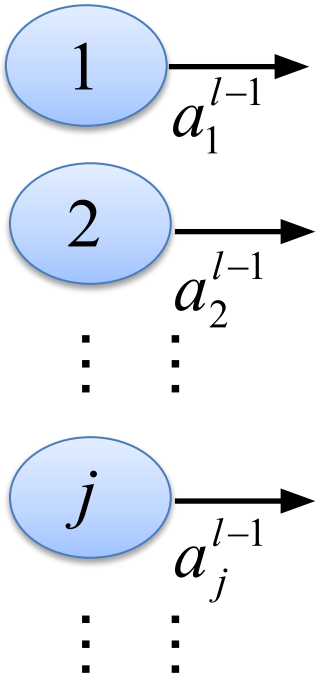
- Usually we design the network structure, and let machine find parameters from data

Step 2. Cost function evaluates how good a set of parameters is

- We design the cost function based on the task

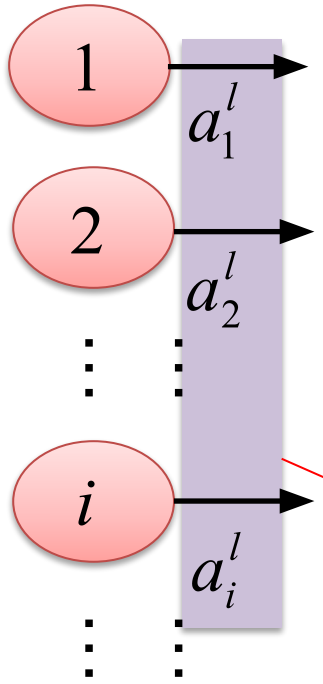
Step 3. Find the best function set (e.g. back propagation)

A Type of Neural Networks



Layer $l-1$

N_{l-1} nodes



Layer l

N_l nodes

Output of a neuron:

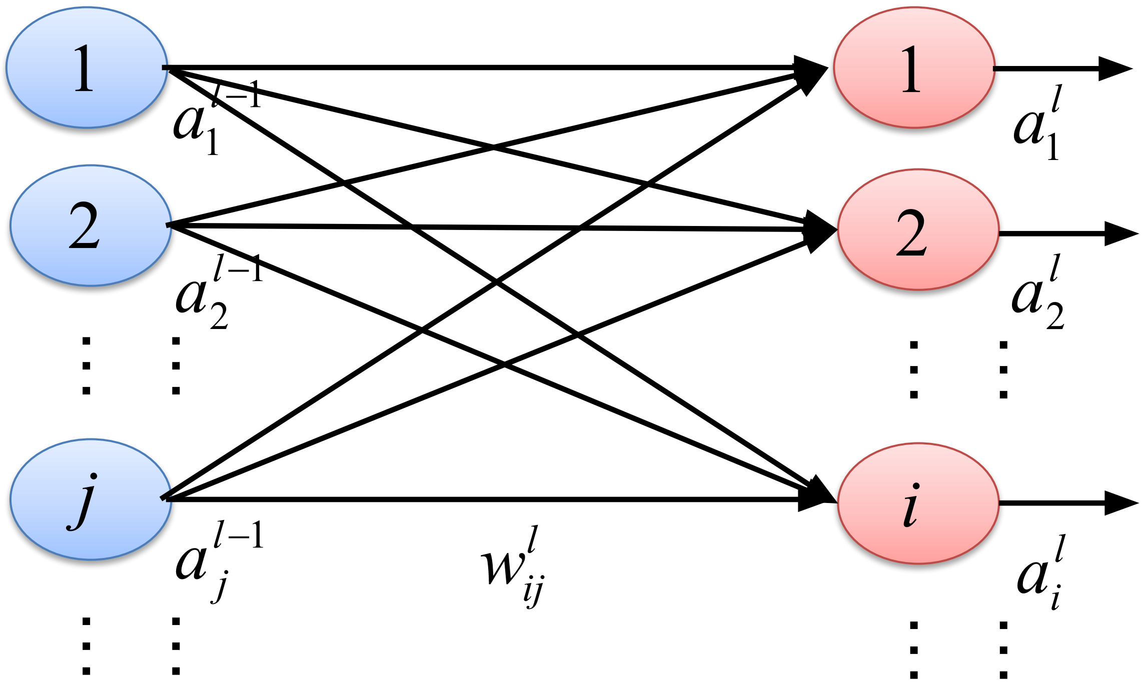
a_i^l → Layer l

a_i^l → Neuron i

Output of one layer:

a^l : a vector

Fully Connected Layer (1)



Layer $l-1$

N_{l-1} nodes

Layer l

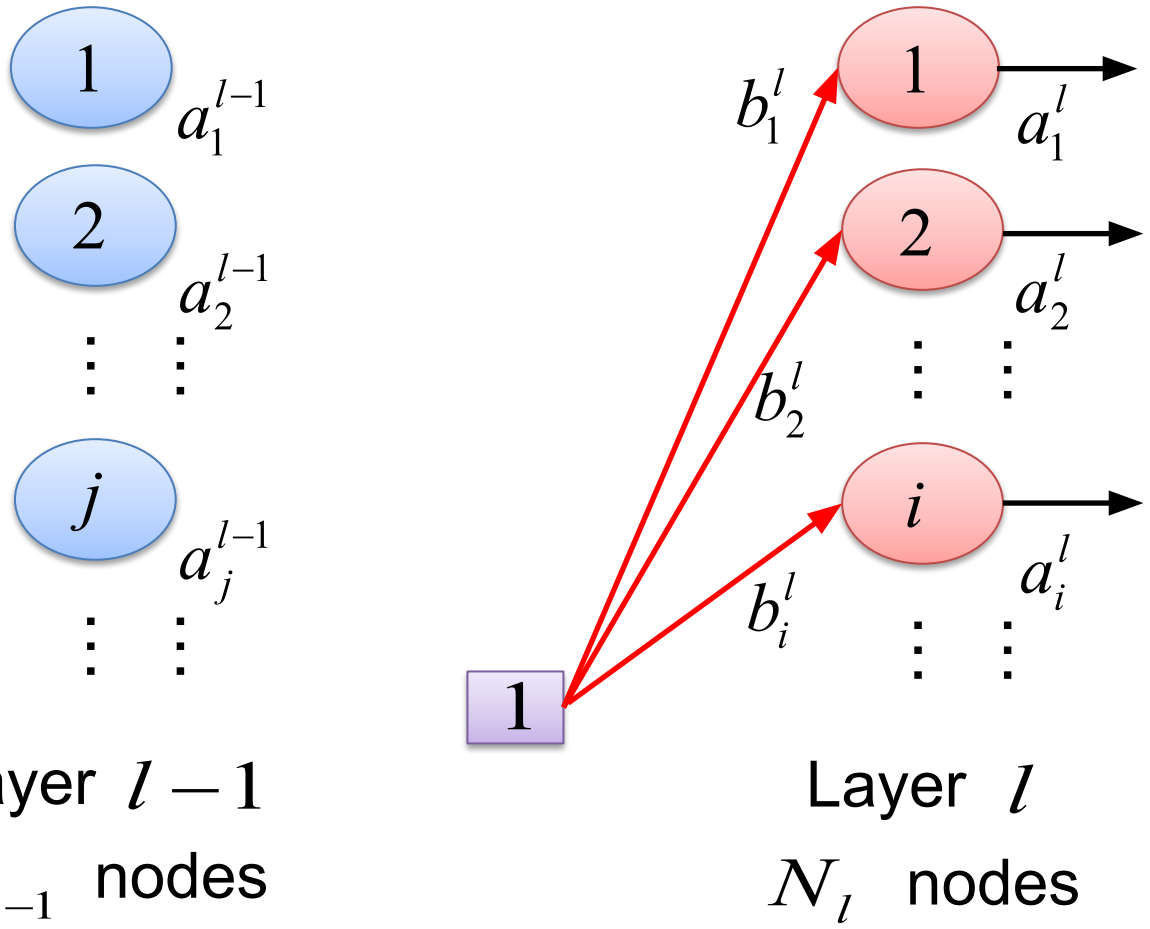
N_l nodes

w_{ij}^l → Layer $l-1$ to Layer l

from neuron j (Layer $l-1$)
to neuron i (Layer l)

$$W^l = \underbrace{\begin{bmatrix} w_{11}^l & w_{12}^l & \cdots \\ w_{21}^l & w_{22}^l & \\ \vdots & & \ddots \end{bmatrix}}_{\mathrm{N}_{l-1}} \Big\} \mathrm{N}_l$$

Fully Connected Layer (2)

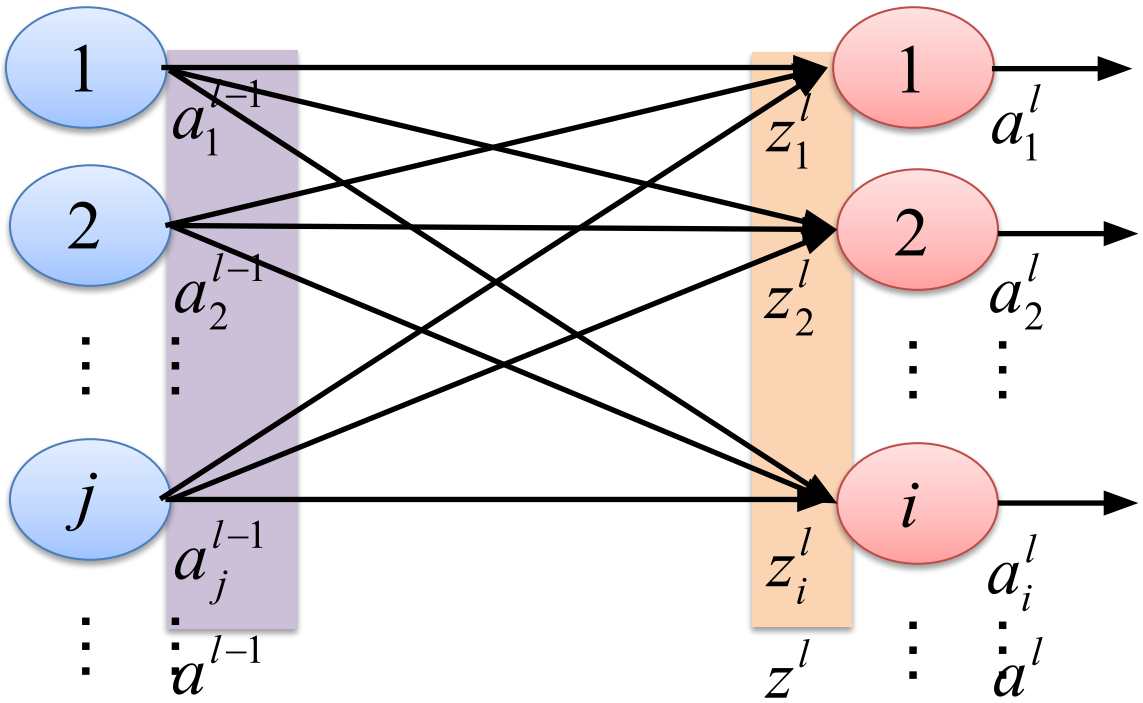


b_i^l : bias for neuron i at layer l

$$b^l = \begin{bmatrix} b_1^l \\ b_2^l \\ \vdots \\ b_i^l \\ \vdots \end{bmatrix}$$

bias for all neurons in layer l

Fully Connected Layer (3)



Layer $l-1$

N_{l-1} nodes

Layer l

N_l nodes

$$z_1^l = w_{11}^l a_1^{l-1} + w_{12}^l a_2^{l-1} + \boxtimes + b_1^l$$

$$z_2^l = w_{21}^l a_1^{l-1} + w_{22}^l a_2^{l-1} + \boxtimes + b_2^l$$

$$\vdots$$

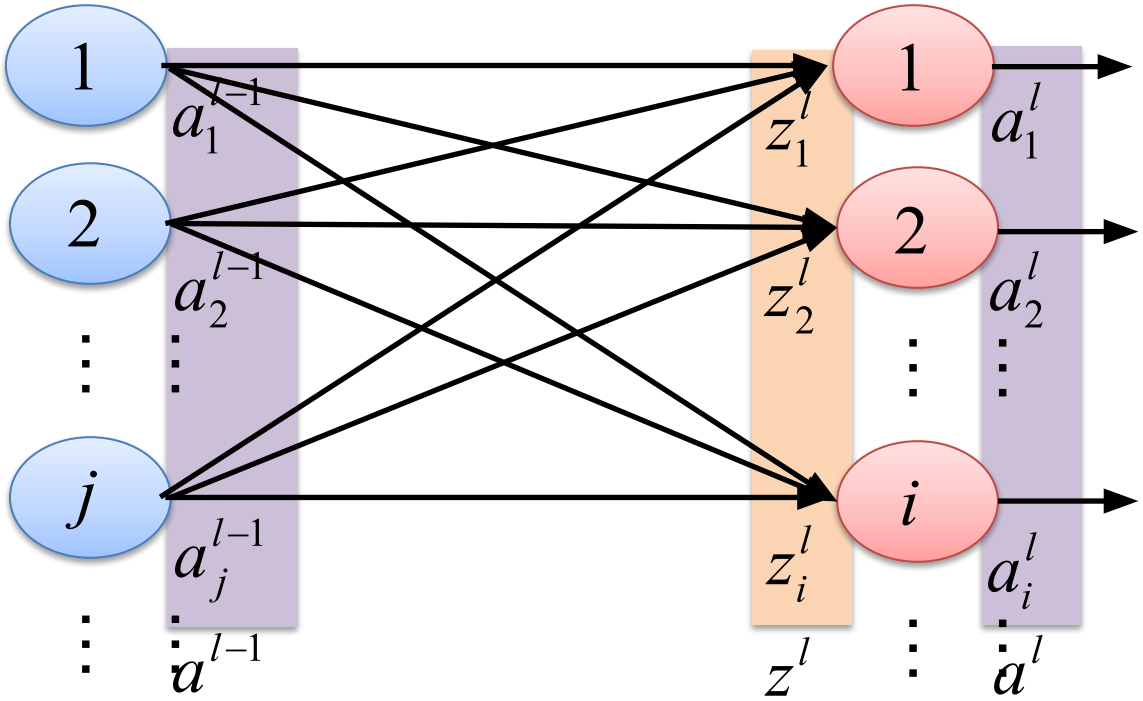
$$z_i^l = w_{i1}^l a_1^{l-1} + w_{i2}^l a_2^{l-1} + \boxtimes + b_i^l$$

$$\vdots$$

$$\begin{bmatrix} z_1^l \\ z_2^l \\ \boxtimes \\ z_i^l \\ \boxtimes \end{bmatrix} = \begin{bmatrix} w_{11}^l & w_{12}^l & \boxtimes \\ w_{21}^l & w_{22}^l & \\ \boxtimes & & \boxtimes \end{bmatrix} \begin{bmatrix} a_1^{l-1} \\ a_2^{l-1} \\ \boxtimes \\ a_i^{l-1} \\ \boxtimes \end{bmatrix} + \begin{bmatrix} b_1^l \\ b_2^l \\ \boxtimes \\ b_i^l \\ \boxtimes \end{bmatrix}$$

$$z^l = W^l a^{l-1} + b^l$$

Fully Connected Layer (4)



Layer $l-1$

N_{l-1} nodes

Layer l

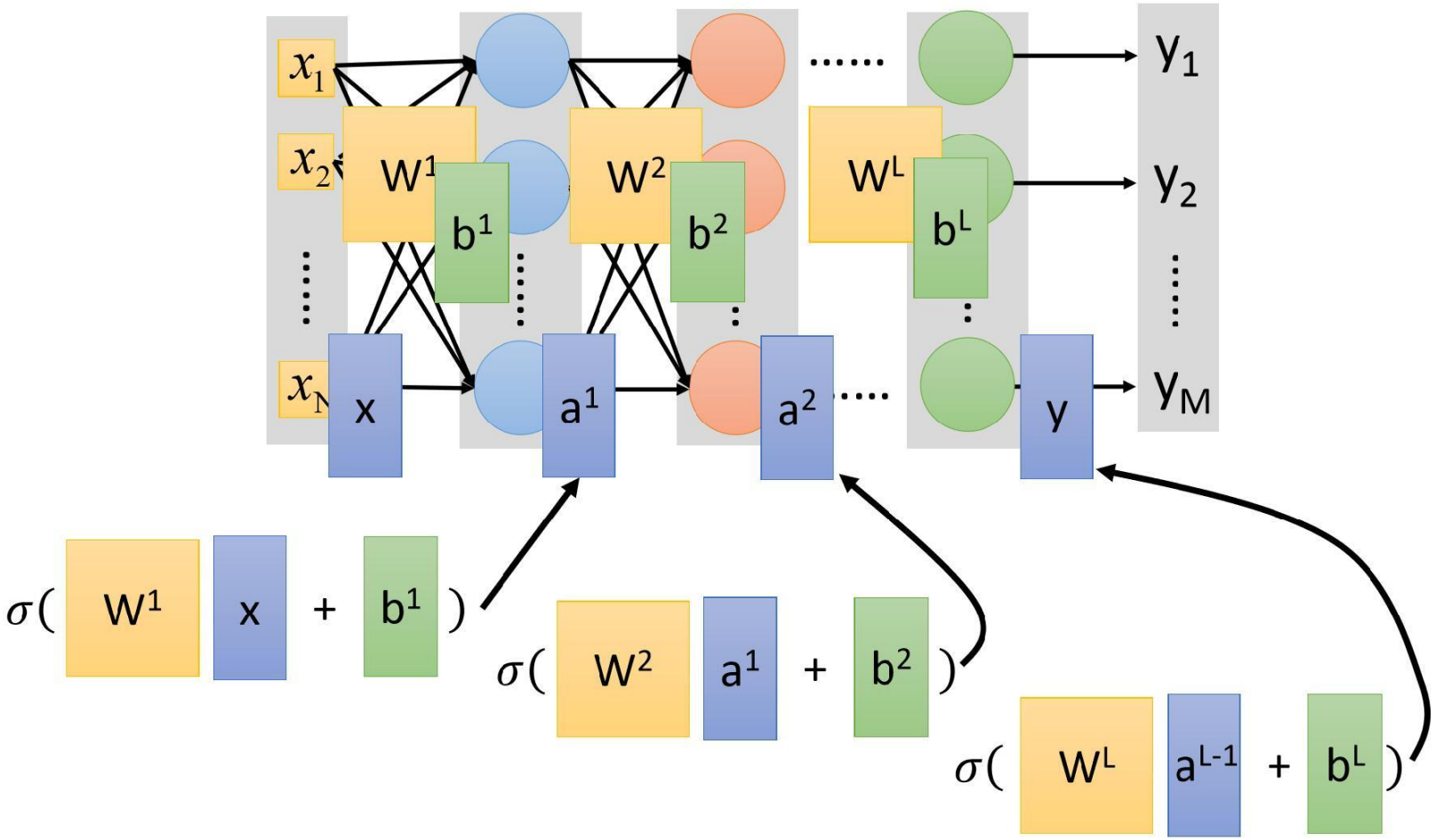
N_l nodes

$$z^l = W^l a^{l-1} + b^l$$

$$a^l = \sigma\left(z^l\right)$$

$$a^l = \sigma\left(W^l a^{l-1} + b^l\right)$$

Fully Connected Network



Three Steps for Deep Learning

Step 1. A neural network is a function composed of simple functions (neurons)

- Usually we design the network structure, and let machine find parameters from data

Step 2. Cost function evaluates how good a set of parameters is

- We design the cost function based on the task

Step 3. Find the best function set (e.g. back propagation)

Cost Function

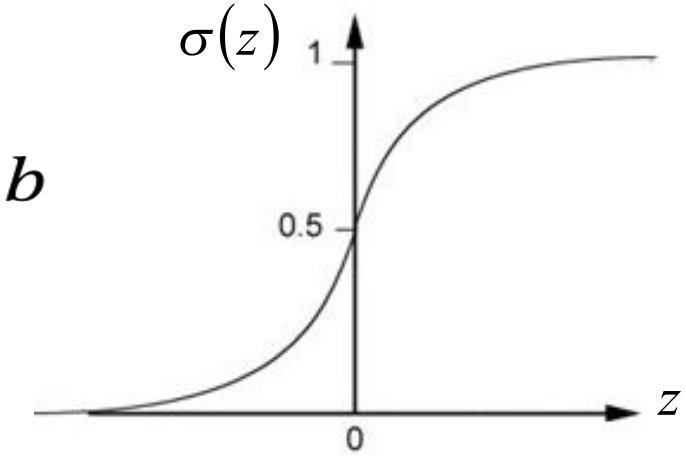
Function set: Including all different w and b

z	$\geq$	0	class 1
z	$<$	0	class 2

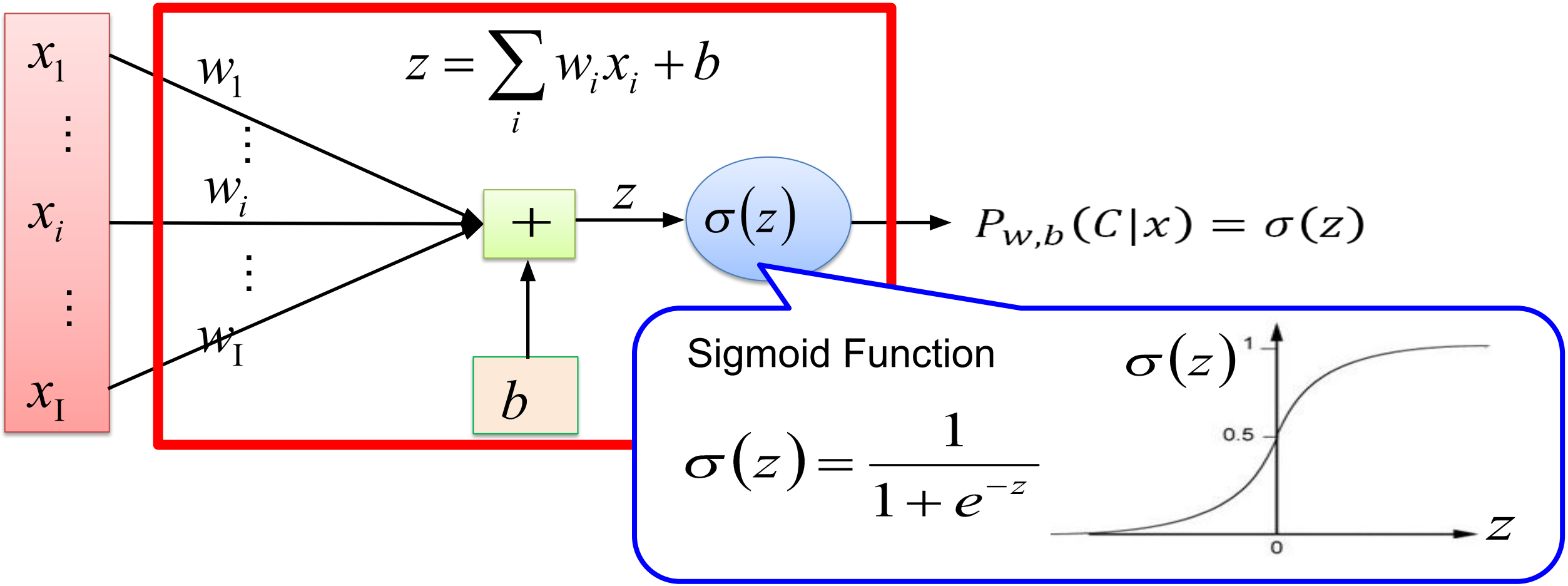
$$P_{w,b}(C|x) = \sigma(z)$$

$$z = w \cdot x + b = \sum_i w_i x_i + b$$

$$\sigma(z) = \frac{1}{1 + exp(-z)}$$



Cost Function



Cost Function - Cross Entropy

$$L(w,b) = f_{w,b}(x^1) f_{w,b}(x^2) \left(1 - f_{w,b}(x^3)\right) \cdots f_{w,b}(x^N)$$

$$-lnL(w,b) = lnf_{w,b}(x^1) + lnf_{w,b}(x^2) + ln\left(1 - f_{w,b}(x^3)\right) \cdots$$

$\hat{y}^n$: 1 for class 1, 0 for class 2

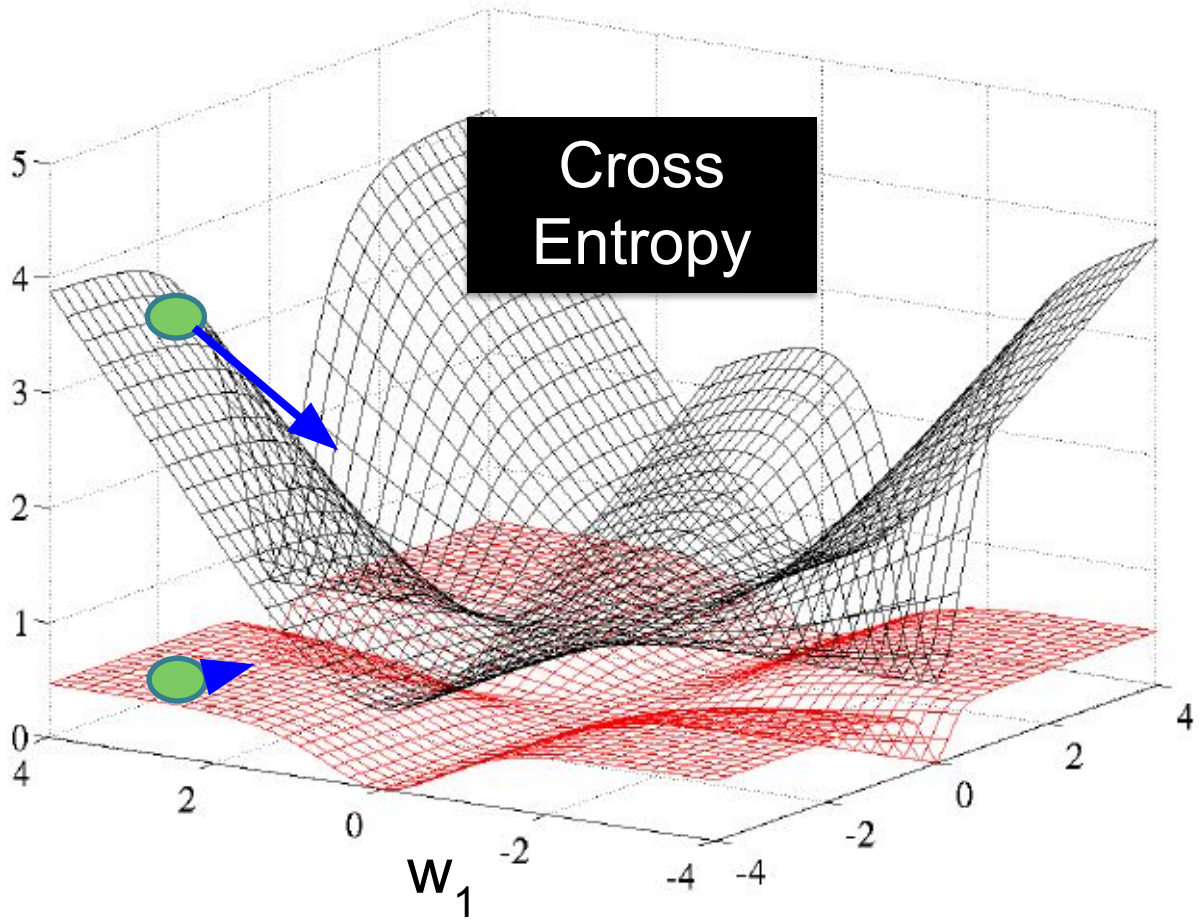
$$= \sum_n -\left[\hat{y}^n lnf_{w,b}(x^n) + (1 - \hat{y}^n) ln\left(1 - f_{w,b}(x^n)\right)\right]$$

Cross entropy:

$$l(f(x^n), \hat{y}^n) = -\left[\hat{y}^n lnf(x^n) + (1 - \hat{y}^n) ln\left(1 - f(x^n)\right)\right]$$

Cross Entropy vs. Square Error

Total Loss



Square Error

Three Steps for Deep Learning

Step 1. A neural network is a function composed of simple functions (neurons)

- Usually we design the network structure, and let machine find parameters from data

Step 2. Cost function evaluates how good a set of parameters is

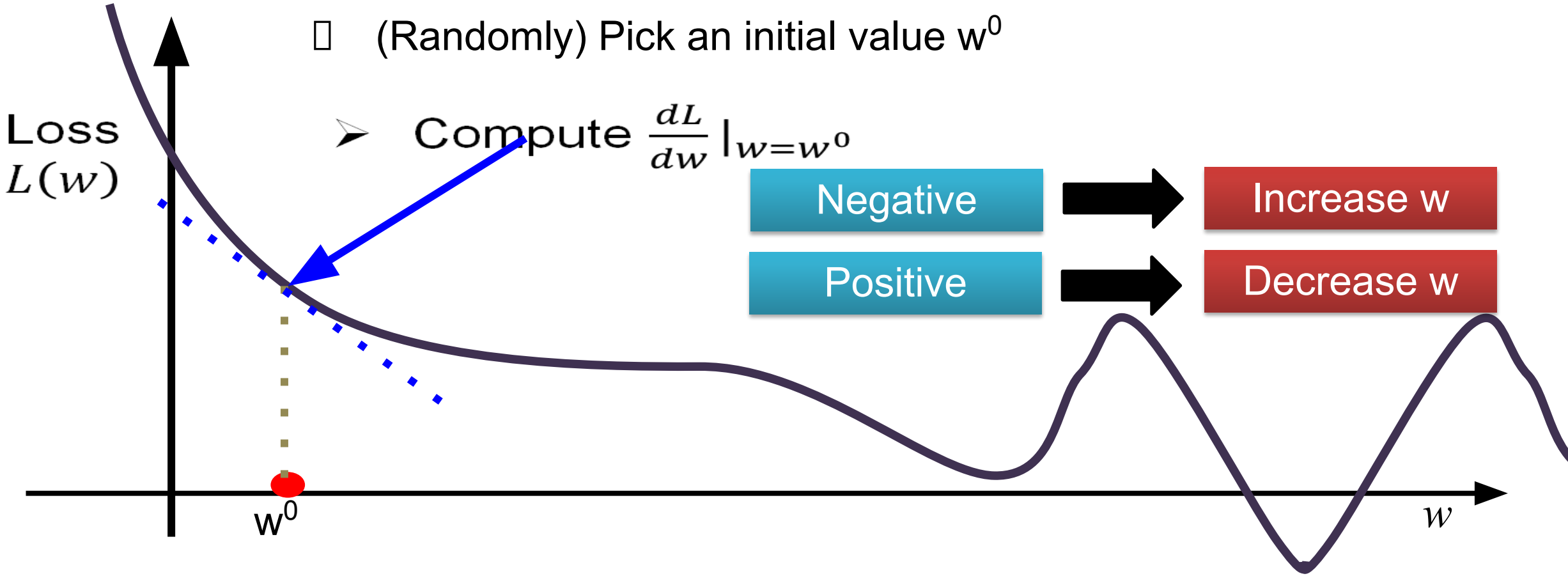
- We design the cost function based on the task

Step 3. Find the best function set (e.g. back propagation)

Gradient Descent

$$w^* = arg \min_w L(w)$$

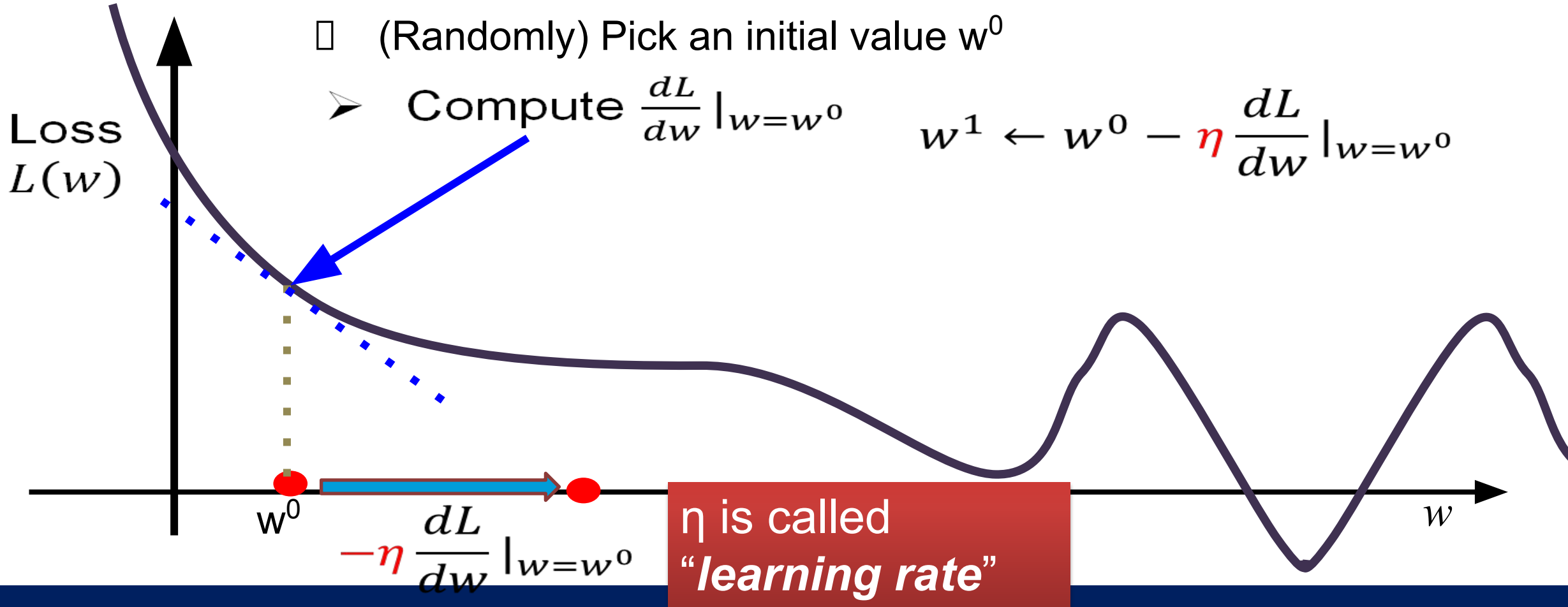
- Consider loss function $L(w)$ with one parameter w:



Gradient Descent

$$w^* = arg \min_w L(w)$$

- Consider loss function $L(w)$ with one parameter w:



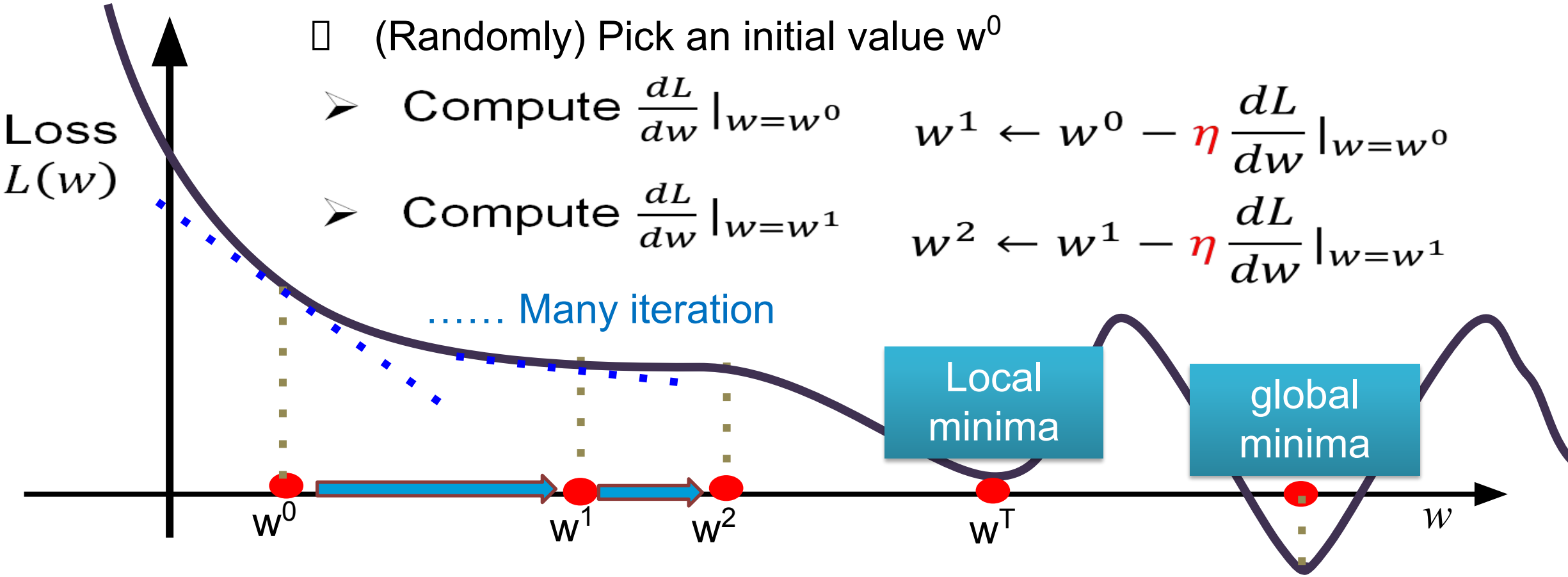
Gradient Descent

$$w^* = arg \min_w L(w)$$

- Consider loss function $L(w)$ with one parameter w:

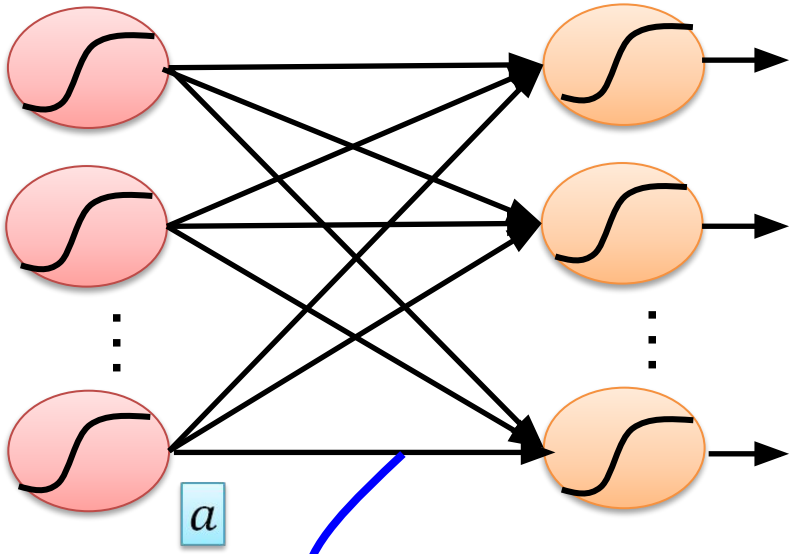
(Randomly) Pick an initial value w^0

- Compute $\frac{dL}{dw}|_{w=w^0}$ $\quad w^1 \leftarrow w^0 - \eta \frac{dL}{dw}|_{w=w^0}$
- Compute $\frac{dL}{dw}|_{w=w^1}$ $\quad w^2 \leftarrow w^1 - \eta \frac{dL}{dw}|_{w=w^1}$

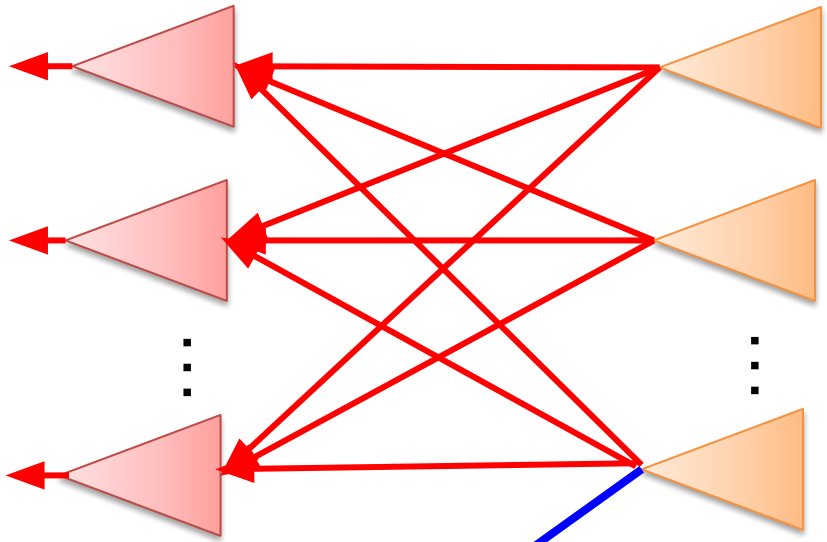


Backpropagation – Summary

Forward Pass



Backward Pass



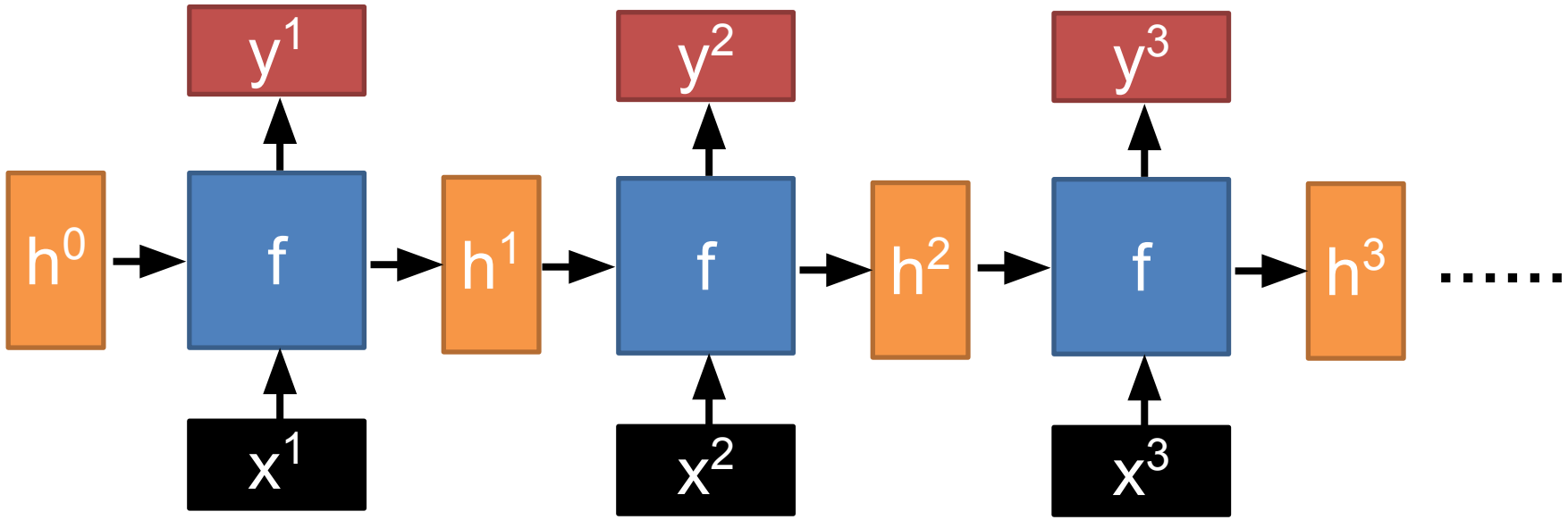
$$\frac{\partial z}{\partial w} = a \qquad \times \qquad \frac{\partial l}{\partial z} \qquad = \frac{\partial l}{\partial w}$$

for all w

Recurrent Neural Network

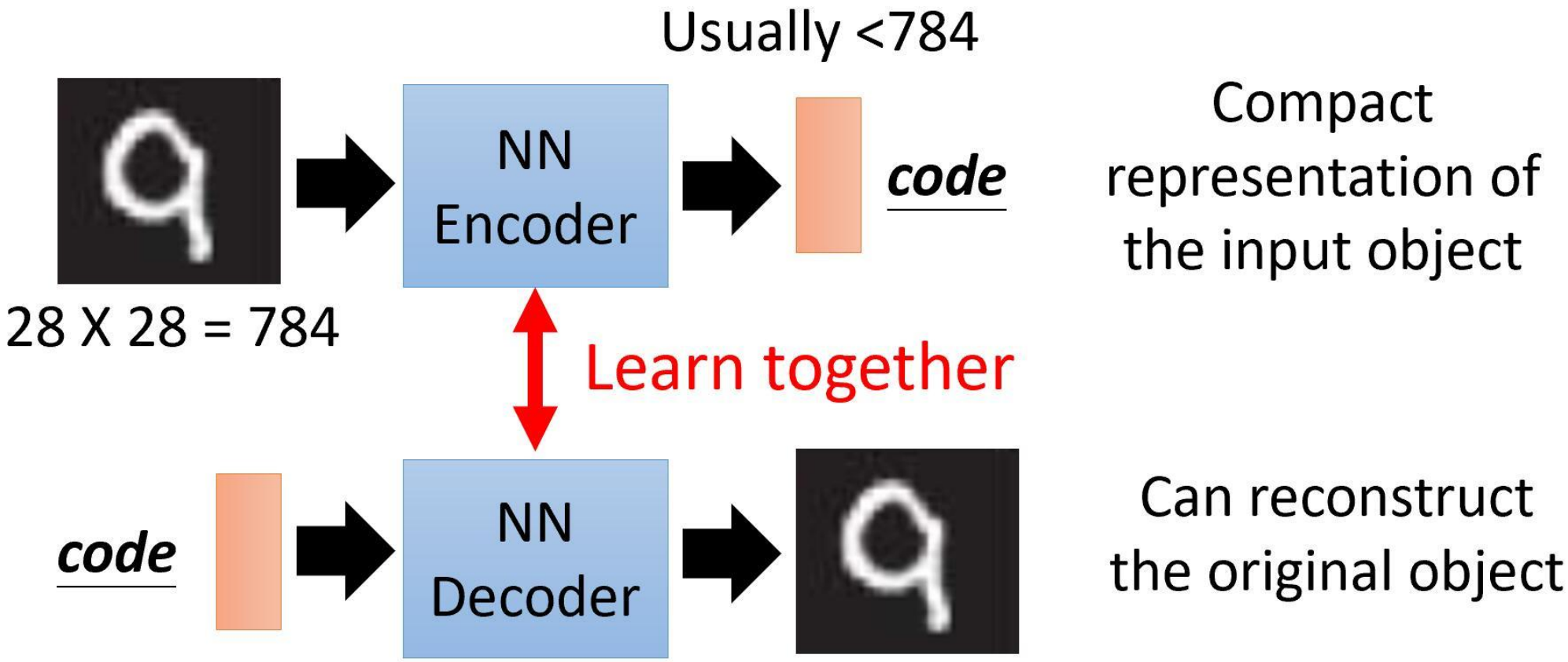
- Given function f: $h', y = f(h, x)$

h and h' are vectors with the same dimension

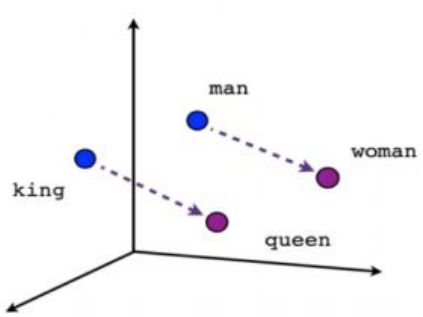


No matter how long the input/output sequence is, we only need one function f

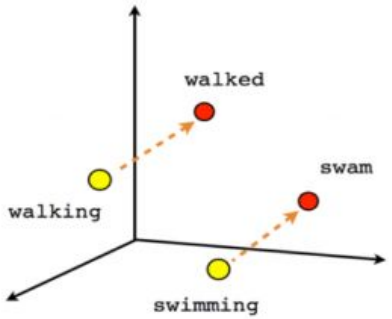
Auto Encoder



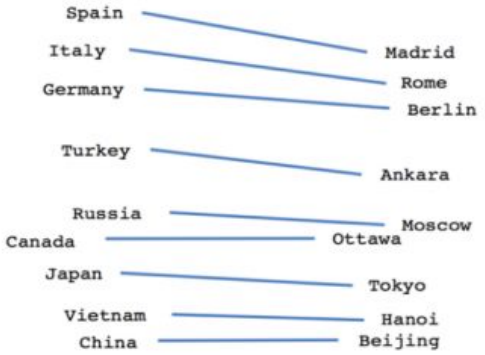
Text Models



Male-Female



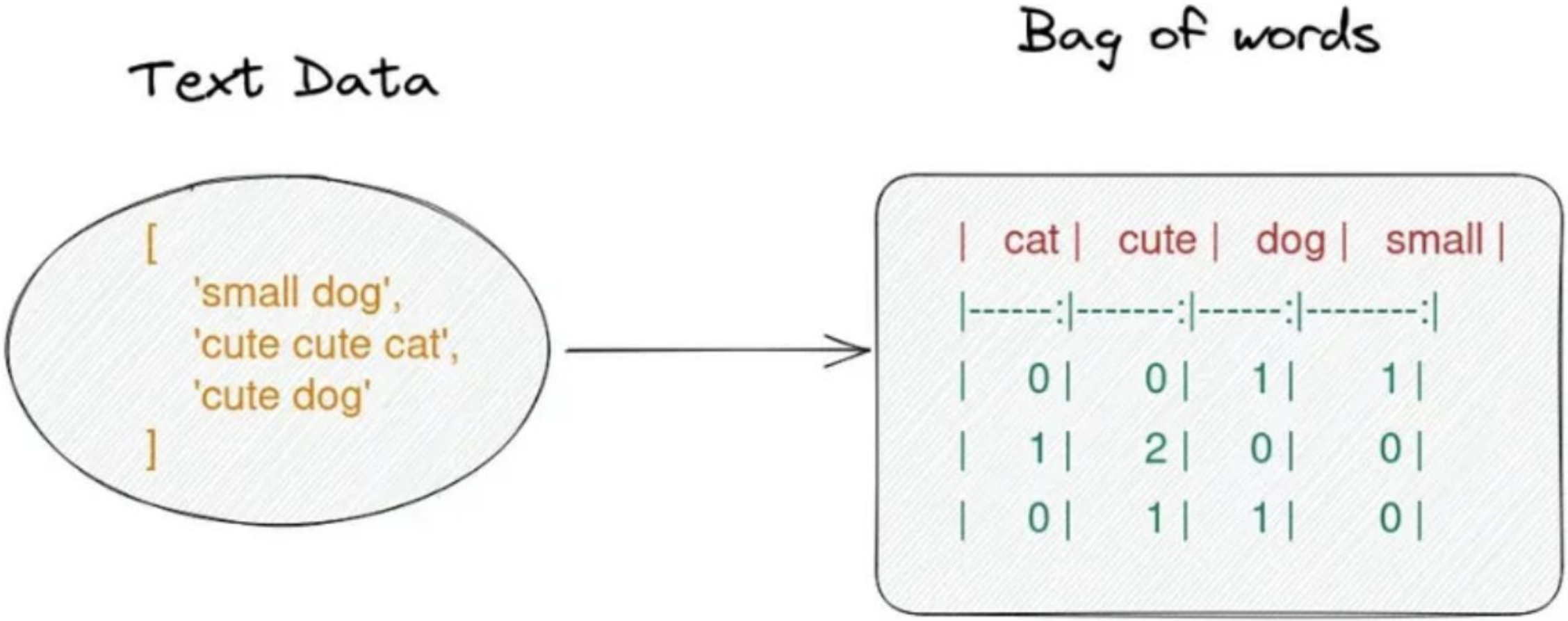
Verb tense



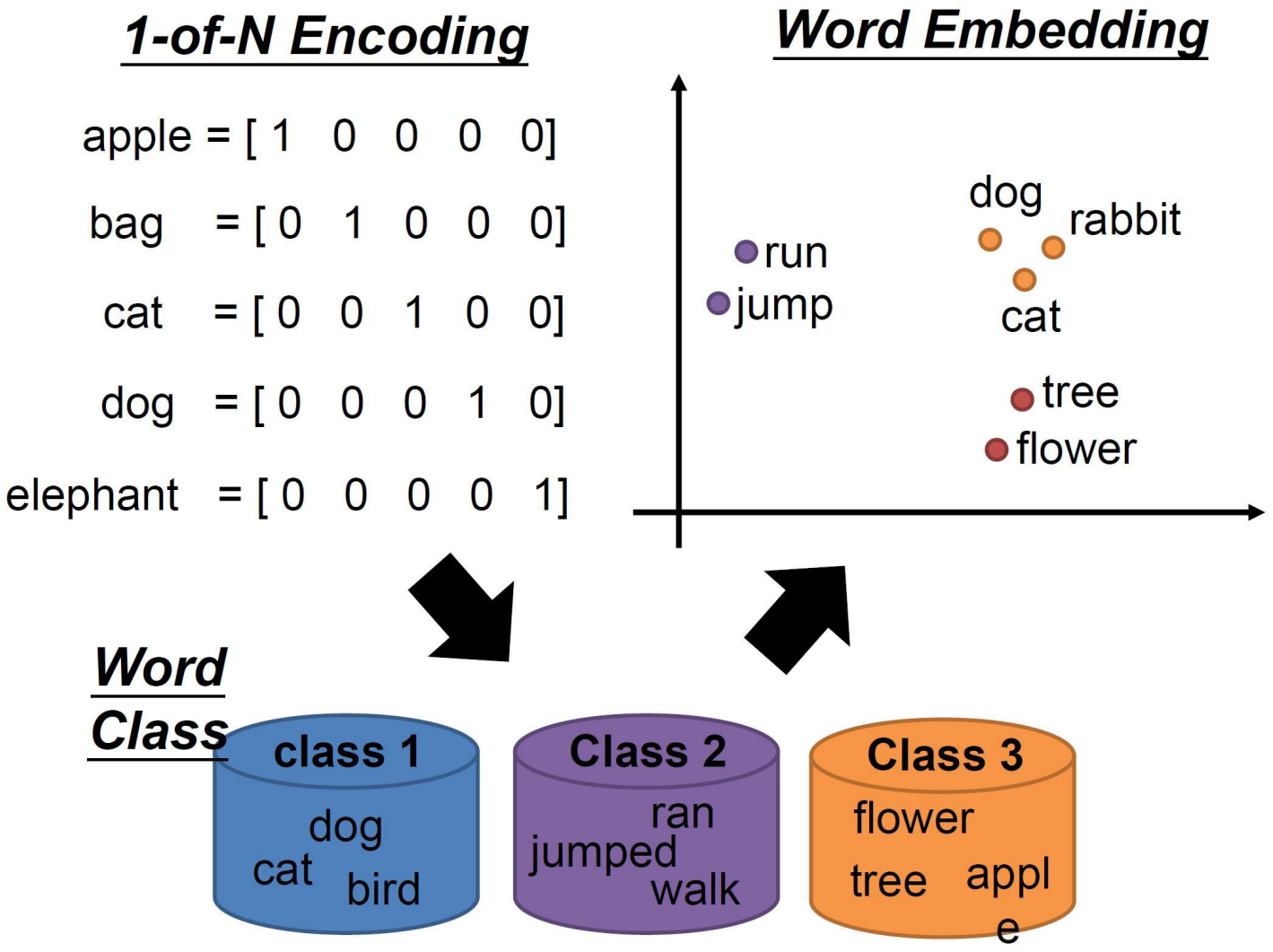
Country-Capital

Bag of Words (1)

- Bag of words - 1 of N encoding

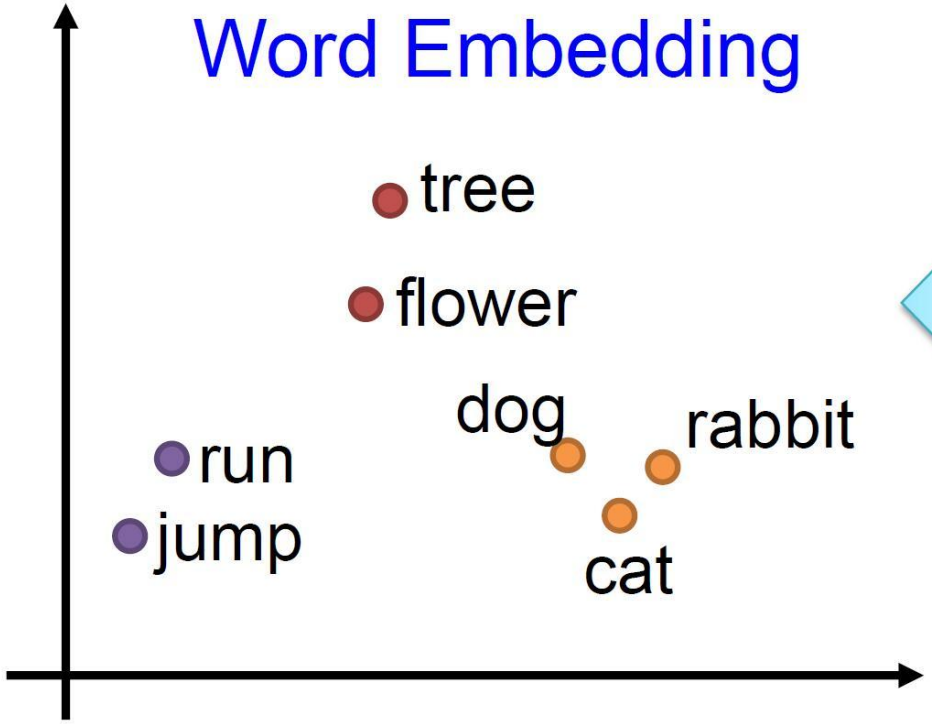


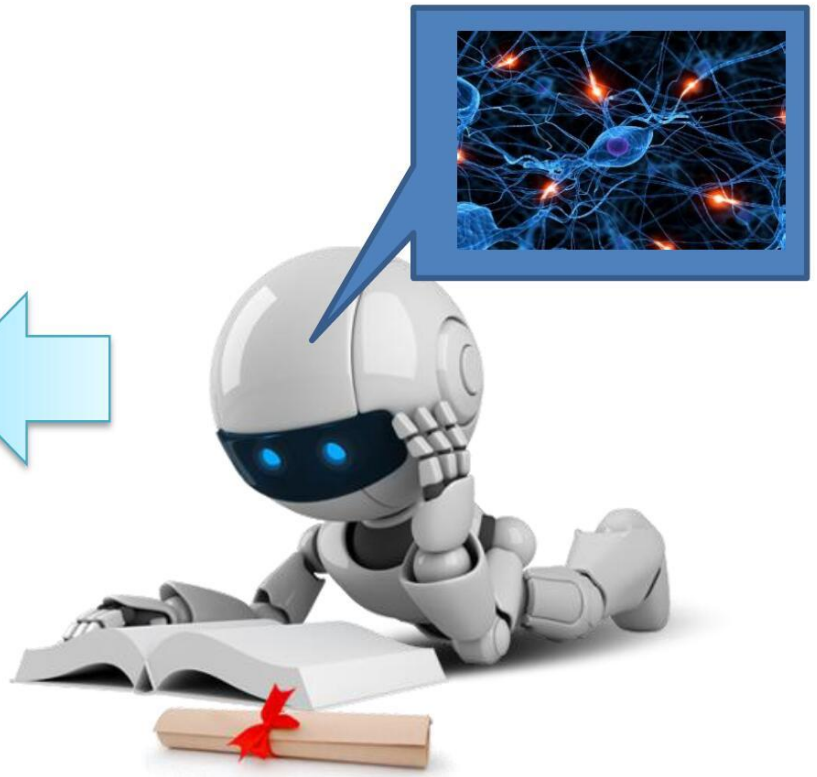
Drawbacks of BOW



Word Embedding (1)

- Machine learns the meaning of words from reading a lot of documents without supervision





Word Embedding (2)

- Machine learns the meaning of words from reading a lot of documents without supervision
- A word can be understood by its context

奥巴马、特朗普 are something very similar

You shall know a word by the company it keeps

奥巴马 120宣誓就職

特朗普 120宣誓就職

Word Embedding - Glove

- **Count based**
  - If two words w_i and w_j frequently co-occur, $V(w_i)$ and $V(w_j)$ would be close to each other
  - E.g. Glove Vector: http://nlp.stanford.edu/projects/glove/

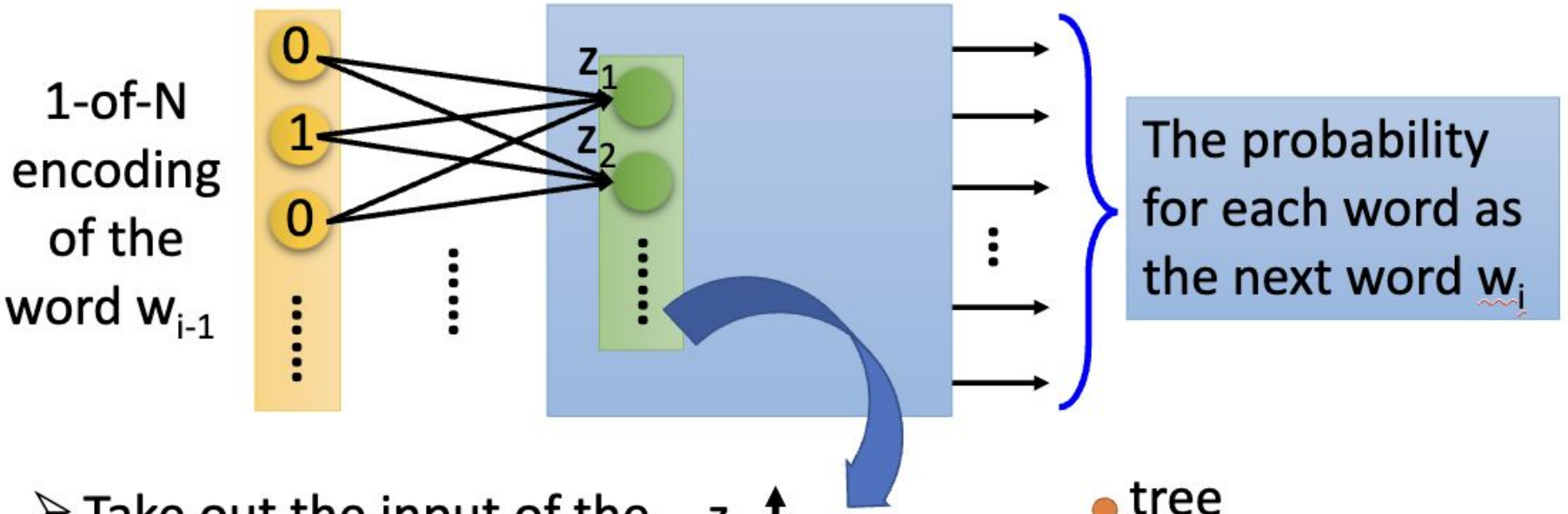
$$V(w_i) \cdot V(w_j) \longleftrightarrow N_{i,j}$$

Inner product

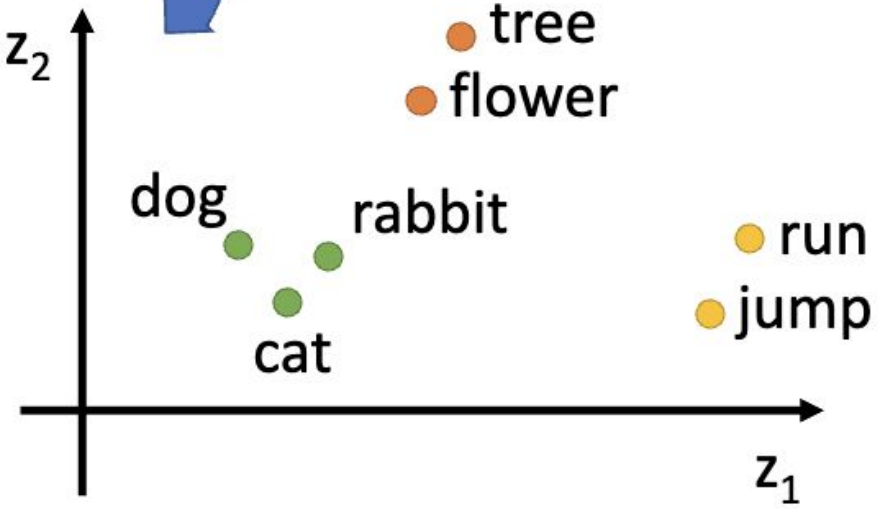
Number of times w_i and w_j in the same document

- **Perdition based**

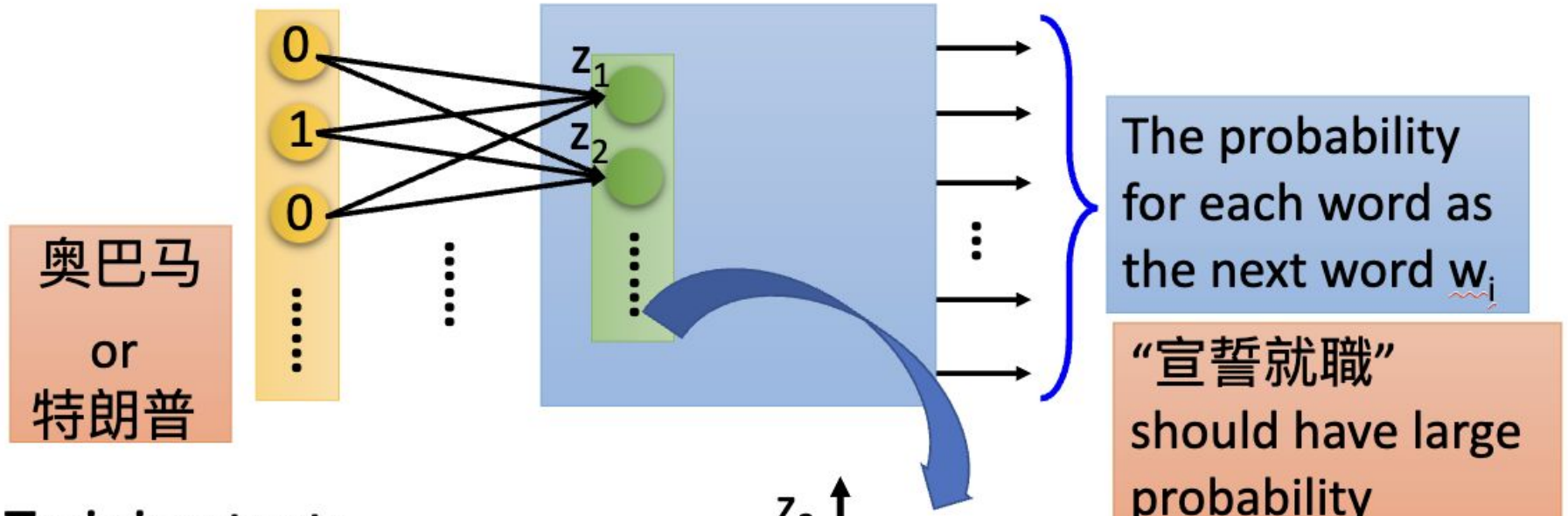
Prediction-based



- Take out the input of the neurons in the first layer
- Use it to represent a word w
- Word vector, word embedding feature: V(w)



Prediction-based



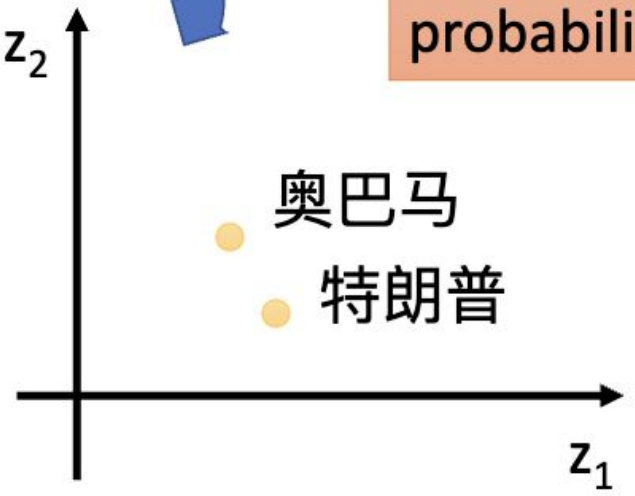
Training text:

..... 奥巴马 宣誓就職 .....

w_{i-1} w_i

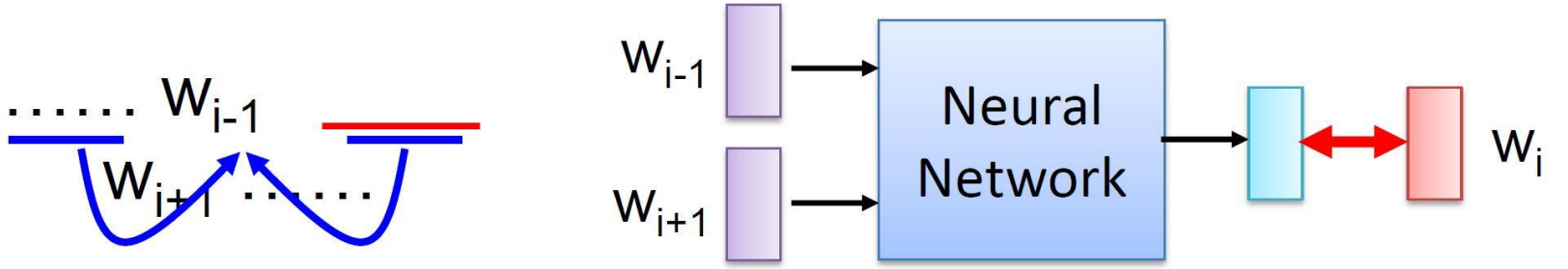
..... 特朗普 宣誓就職 .....

w_{i-1} w_i

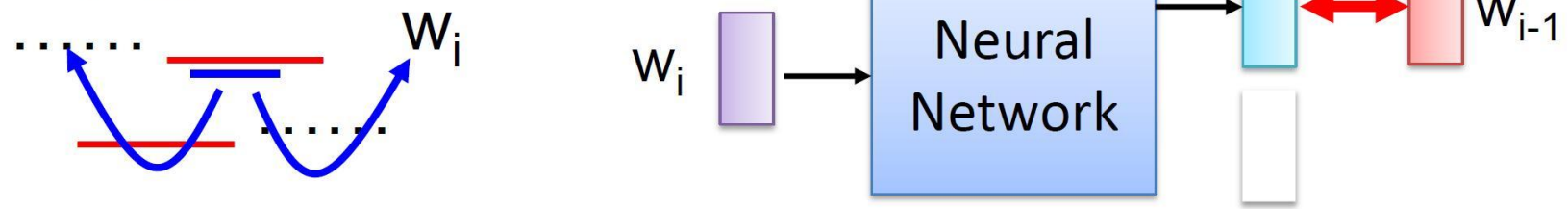


Word2Vec (1)

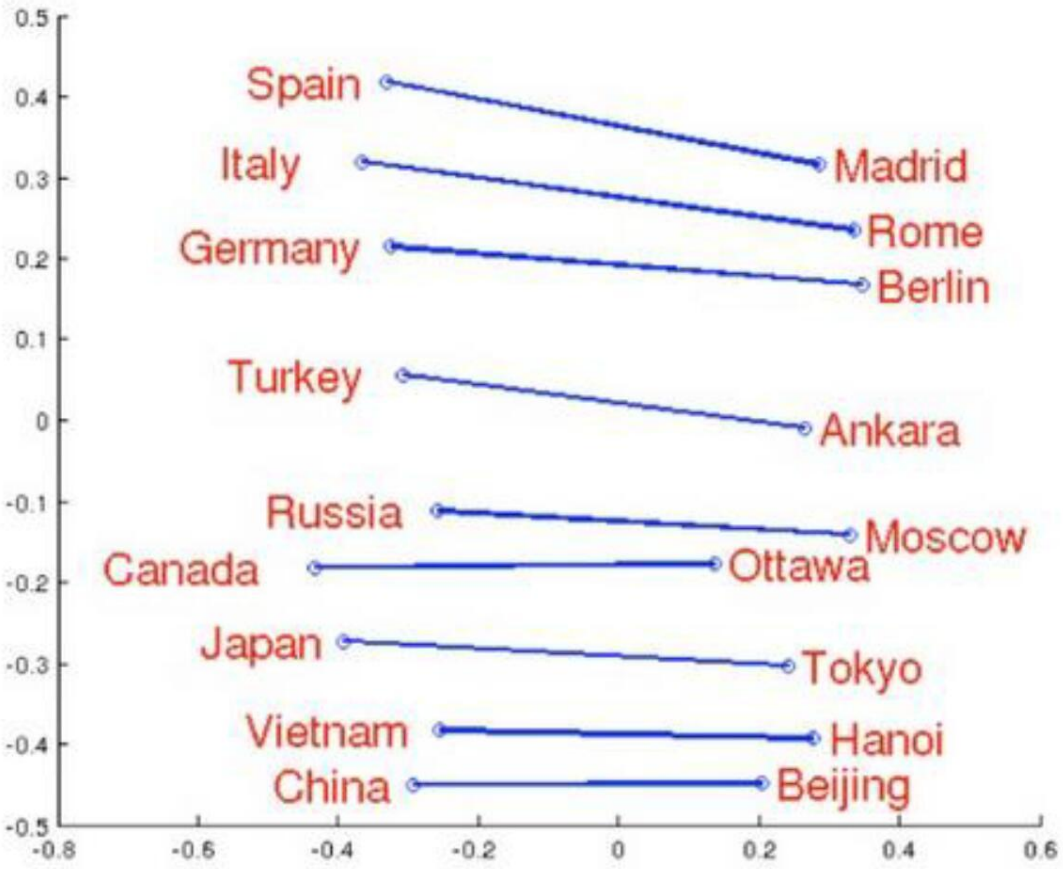
- Continuous bag of word (CBOW) model

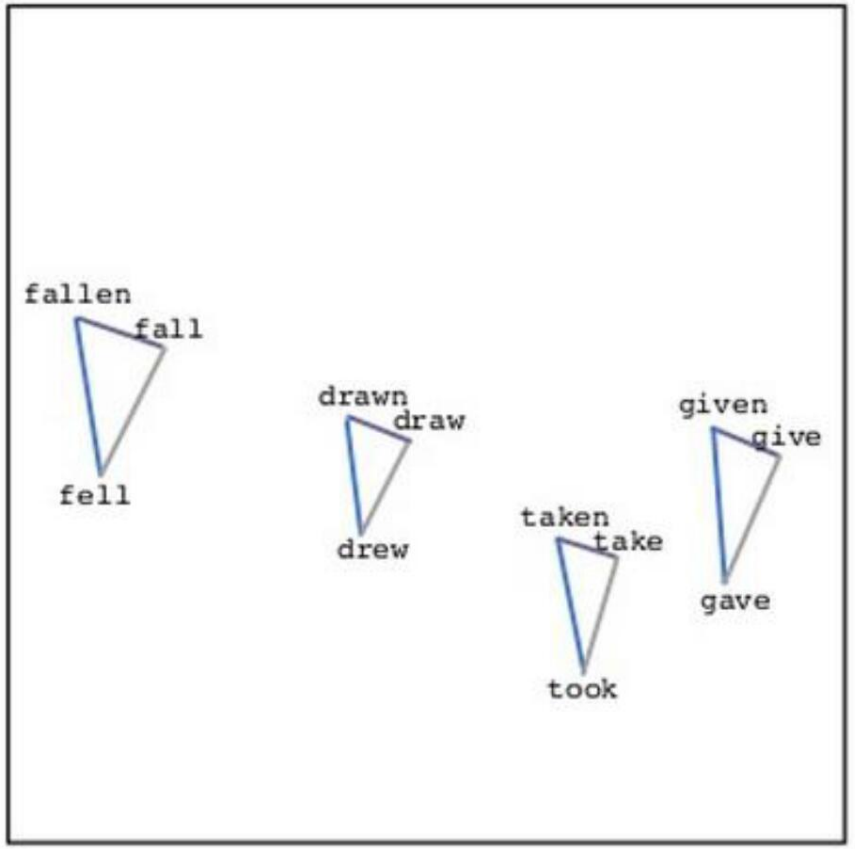


- Skip-gram



Word2Vec (2)





Source: http://www.slideshare.net/hustwj/cikm-keynotenov2014

Word2Vec (3)

- Characteristics

$$V(Germany) \approx V(Berlin) - V(Rome) + V(Italy)$$

$$V(hotter) - V(hot) \approx V(bigger) - V(big)$$
$$V(Rome) - V(Italy) \approx V(Berlin) - V(Germany)$$
$$V(king) - V(queen) \approx V(uncle) - V(aunt)$$

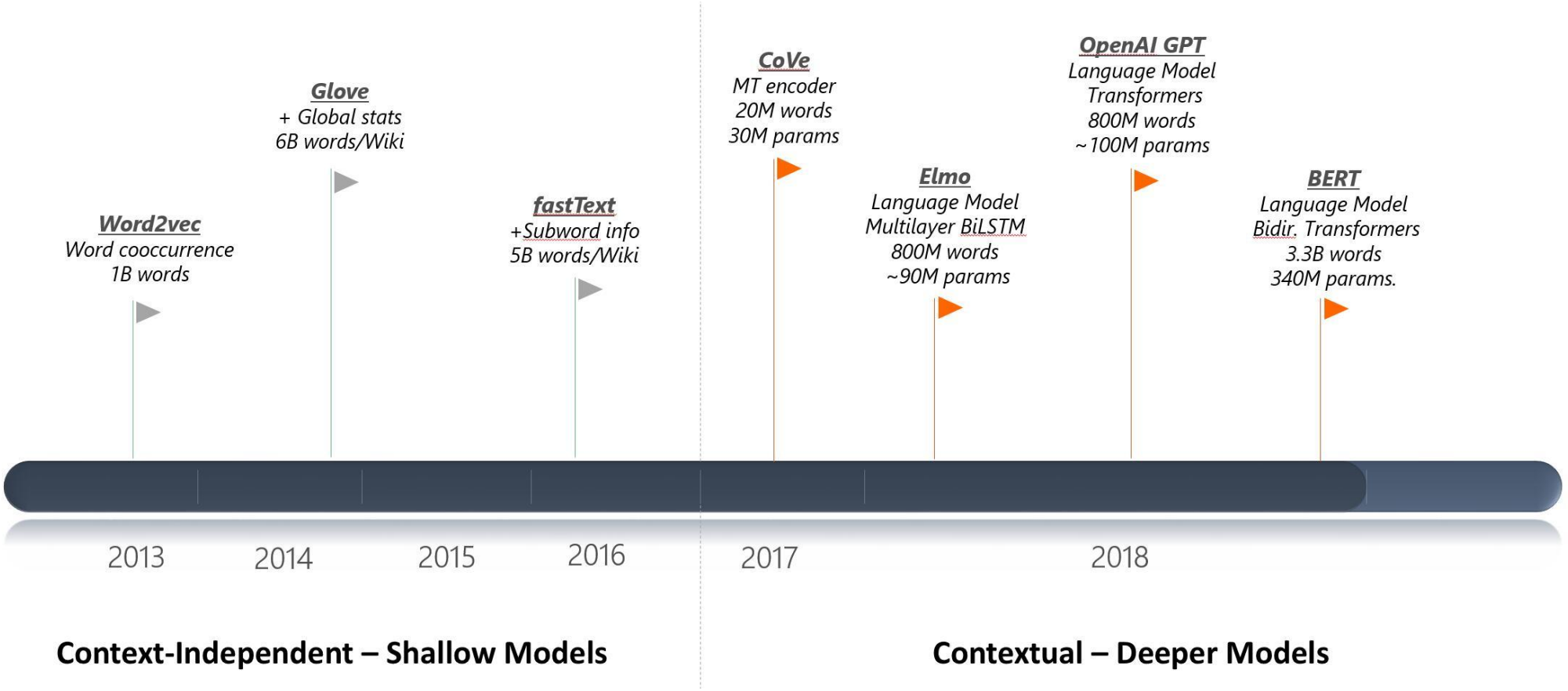
- Solving analogies

Rome : Italy = Berlin : ?

Compute $V(Berlin) - V(Rome) + V(Italy)$

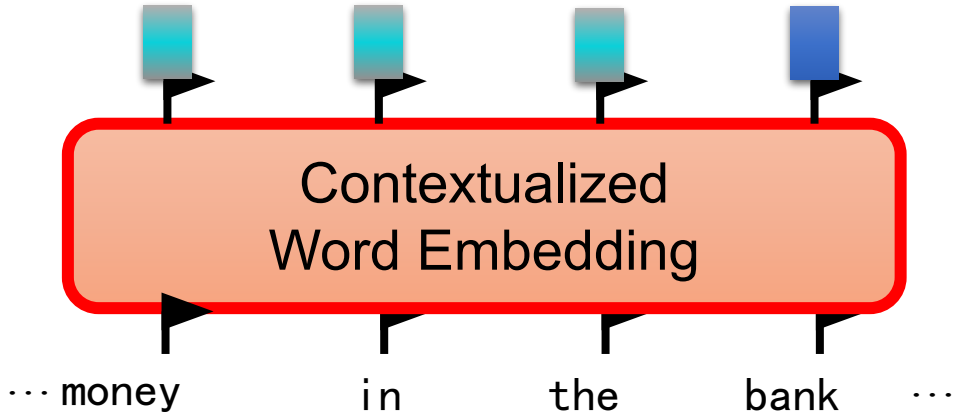
Find the word w with the closest V(w)

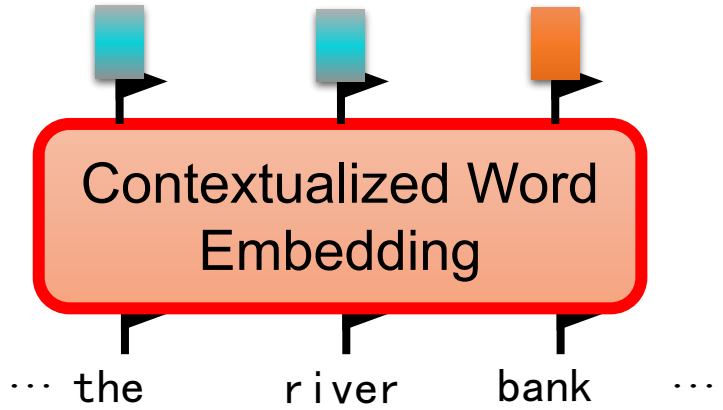
Language Models

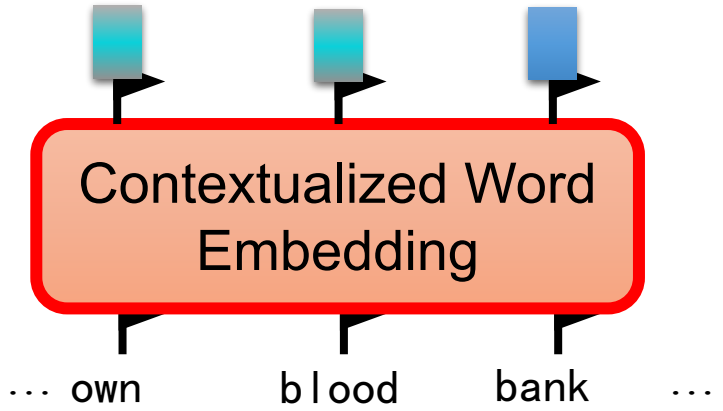


Good Enough?

Contextualized Word Embedding





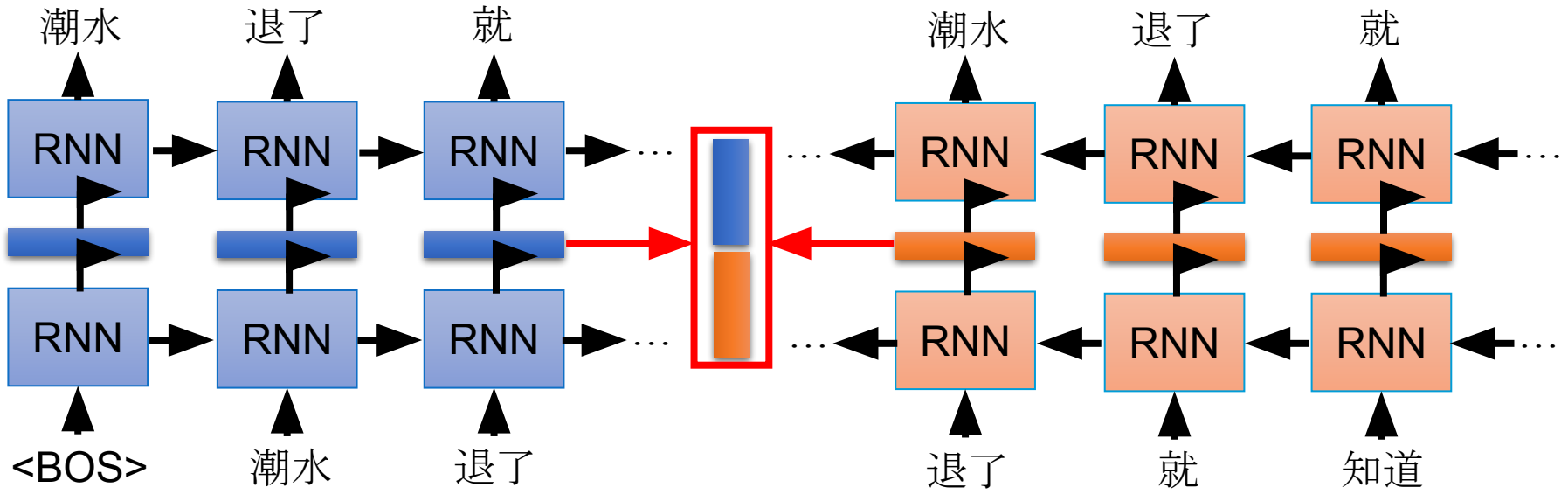


Embeddings from Language Model (ELMO)

https://arxiv.org/abs/1802.05365

- RNN-based language models (trained from lots of sentences)

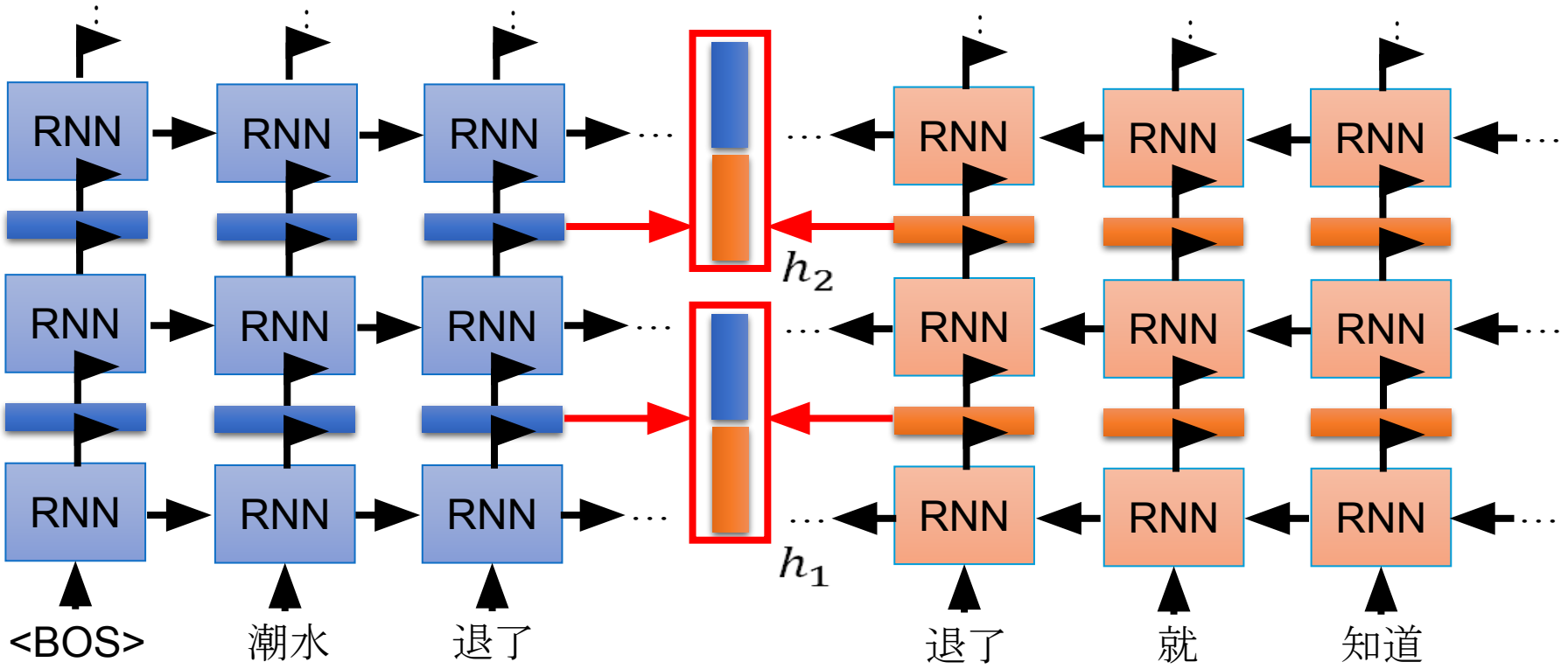
  e.g. given “潮水 退了 就 知道 誰 沒穿 褲子”



ELMO

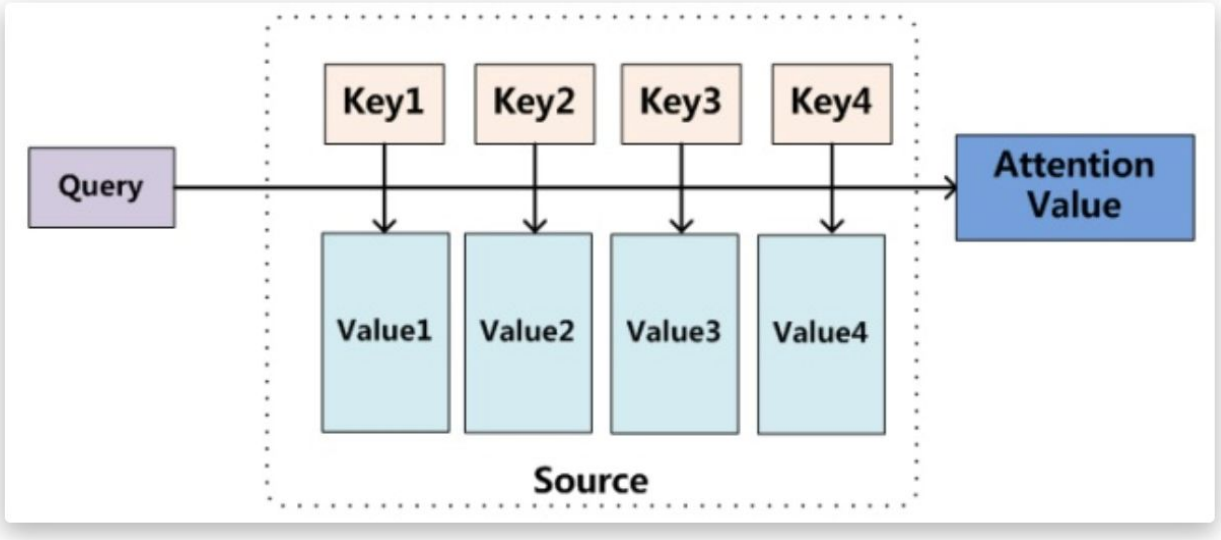
Each layer in deep LSTM can generate a latent representation.

Which one should we use???



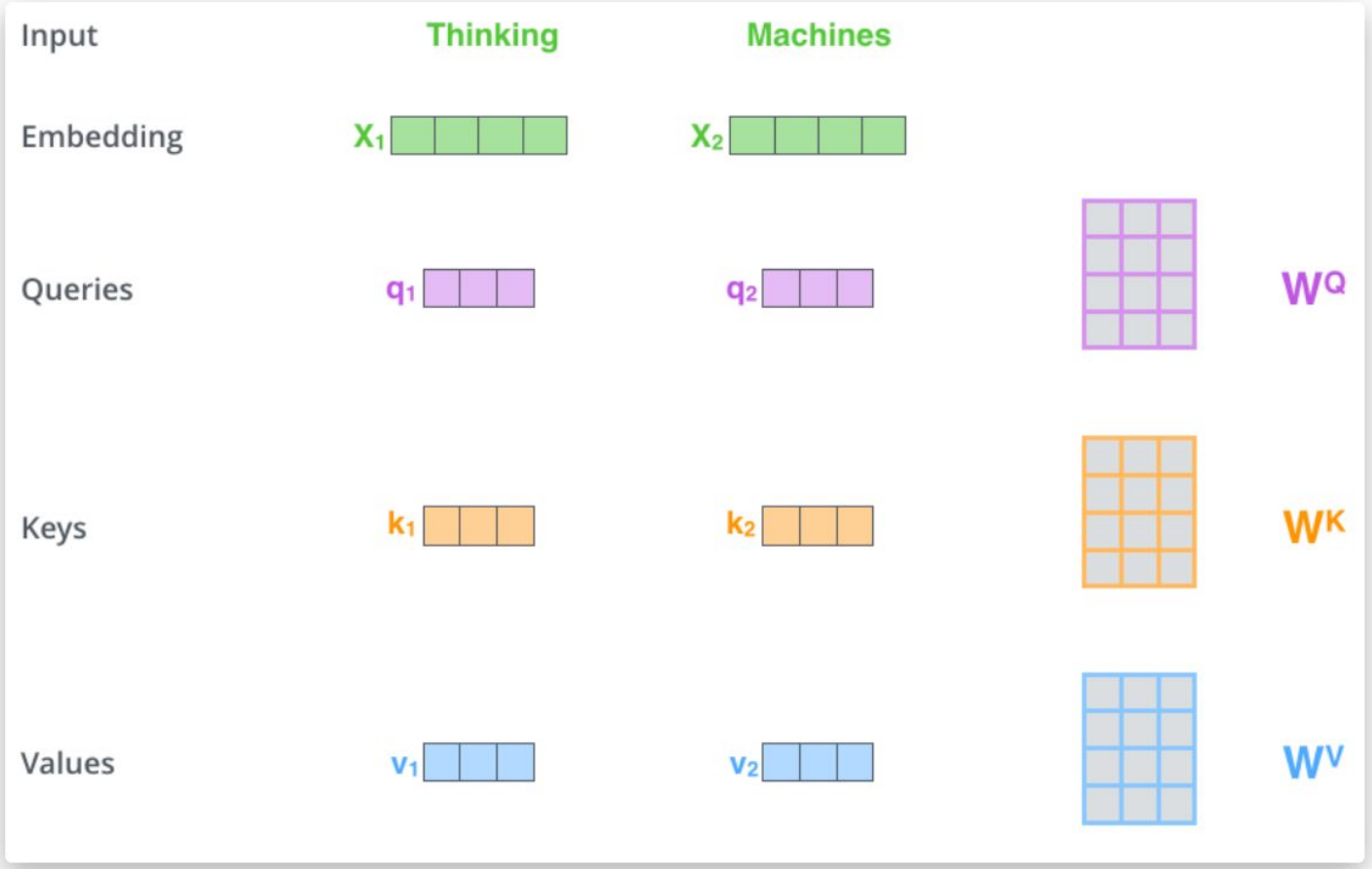
Self-Attention (1)

Self-Attention, also called intra Attention, is an attention mechanism relating different positions of a single sequence in order to compute a representation of the same sequence.

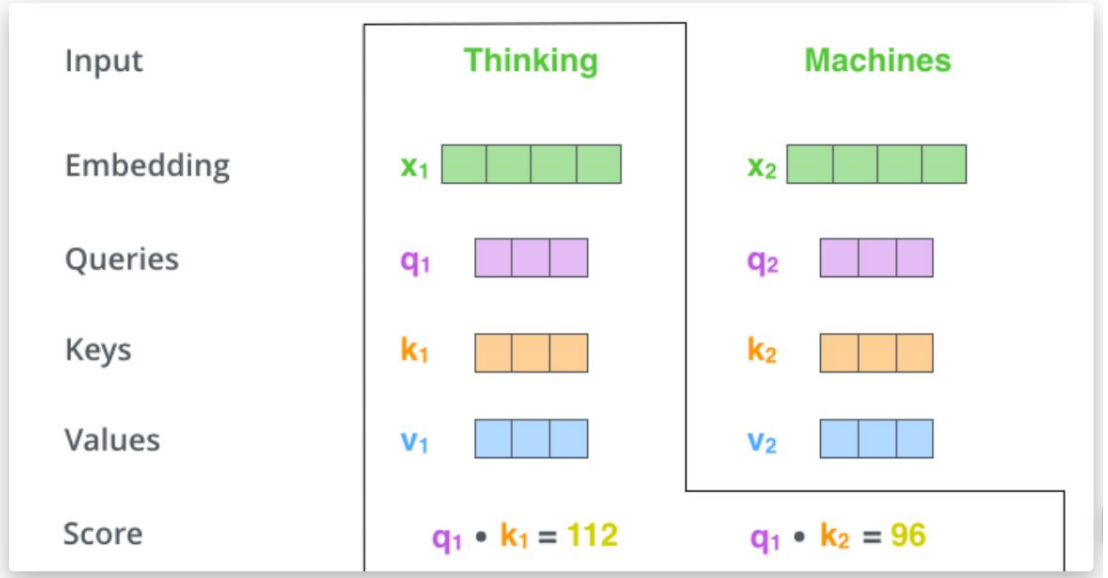


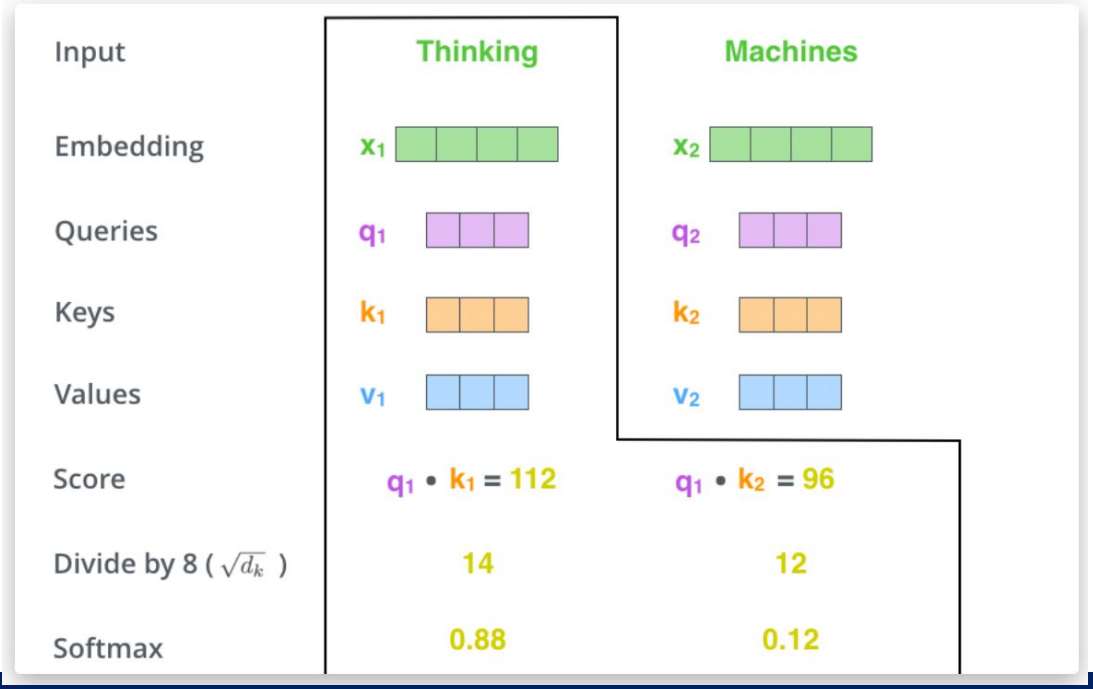
Self-Attention (2)

$$
\begin{aligned}
\mathrm{Q} &= \mathrm{W_q X} \\
\mathrm{K} &= \mathrm{W_k X} \\
\mathrm{V} &= \mathrm{W_v X}
\end{aligned}
$$

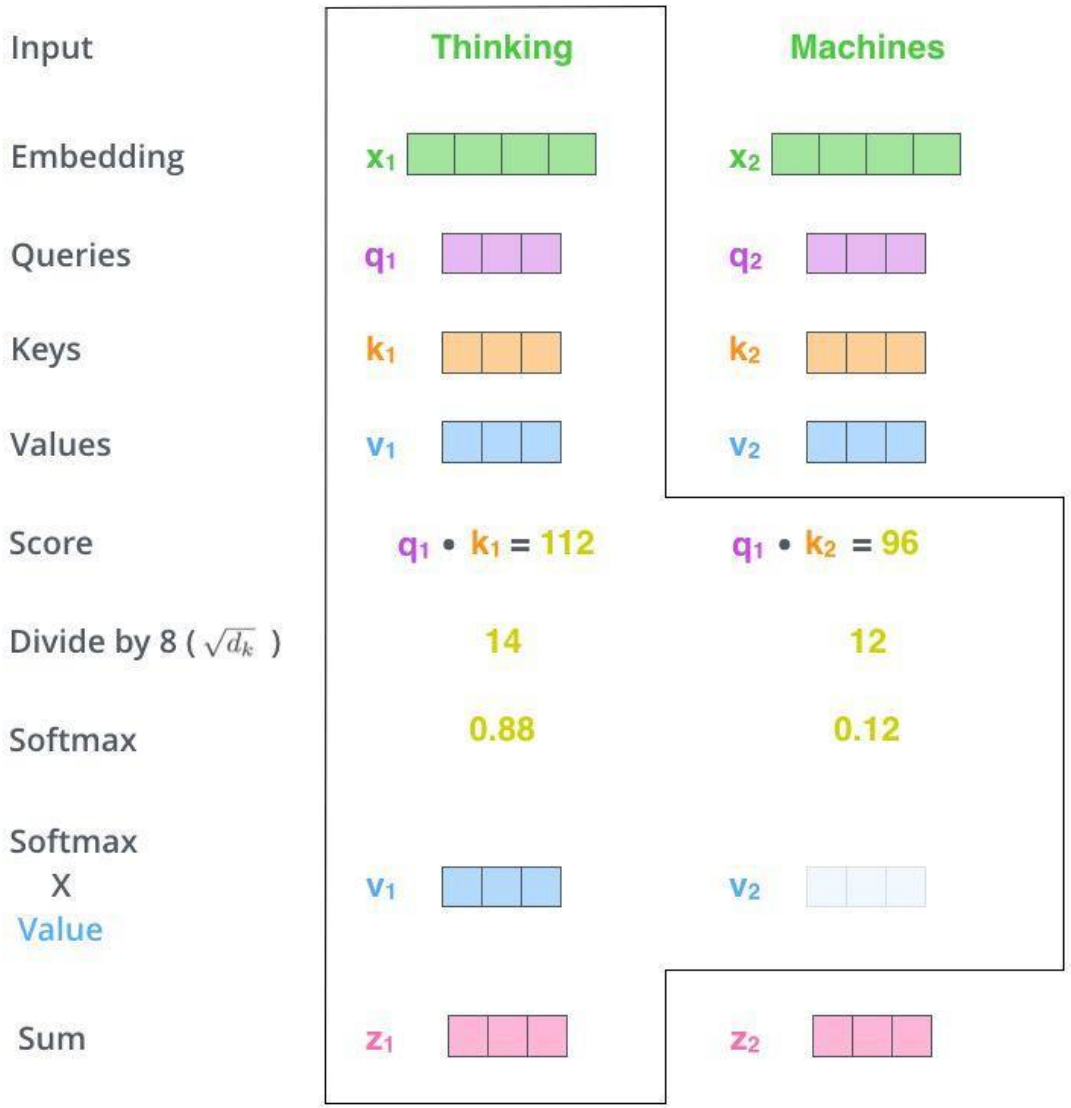


Self-Attention (3)

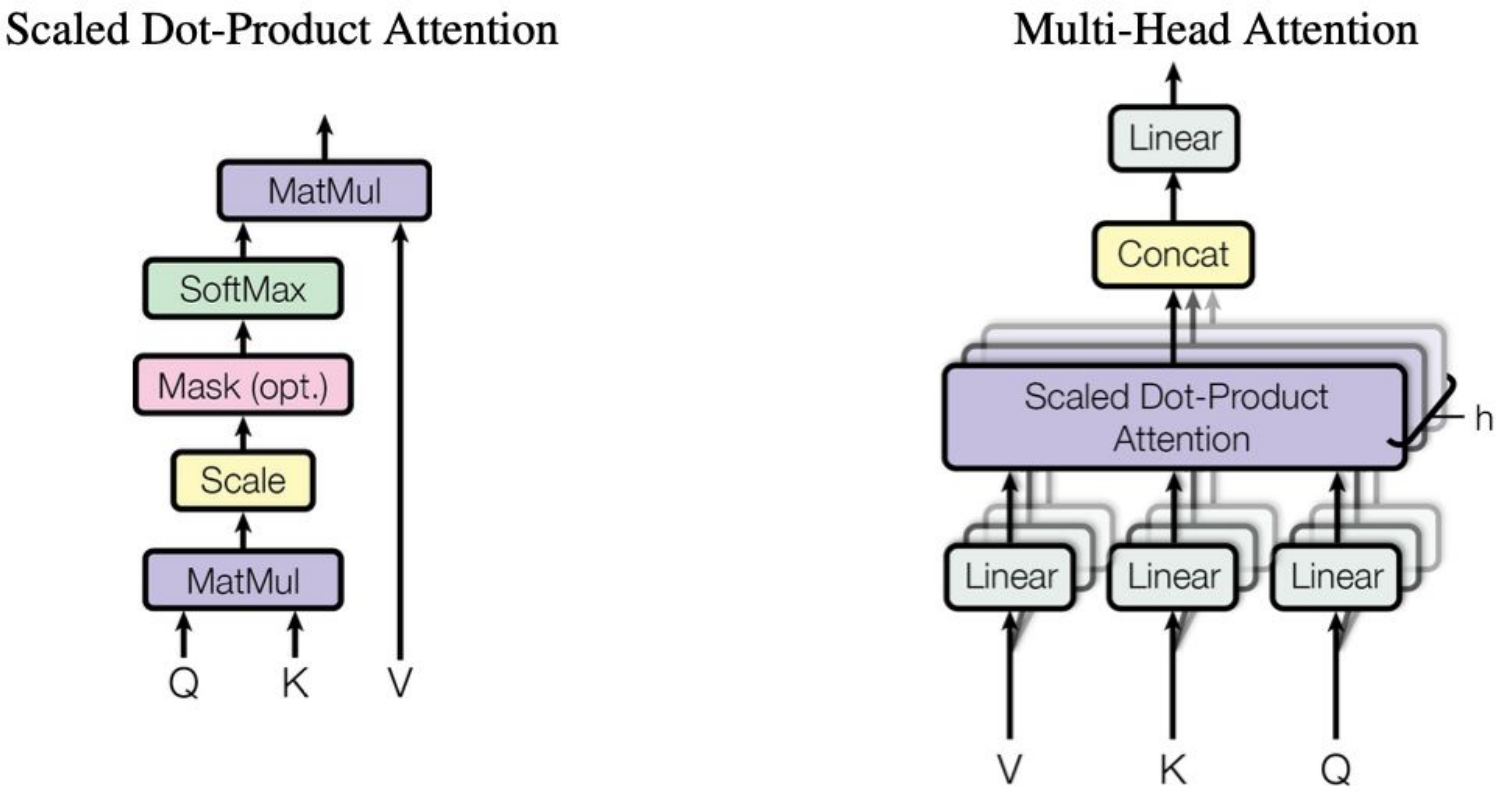




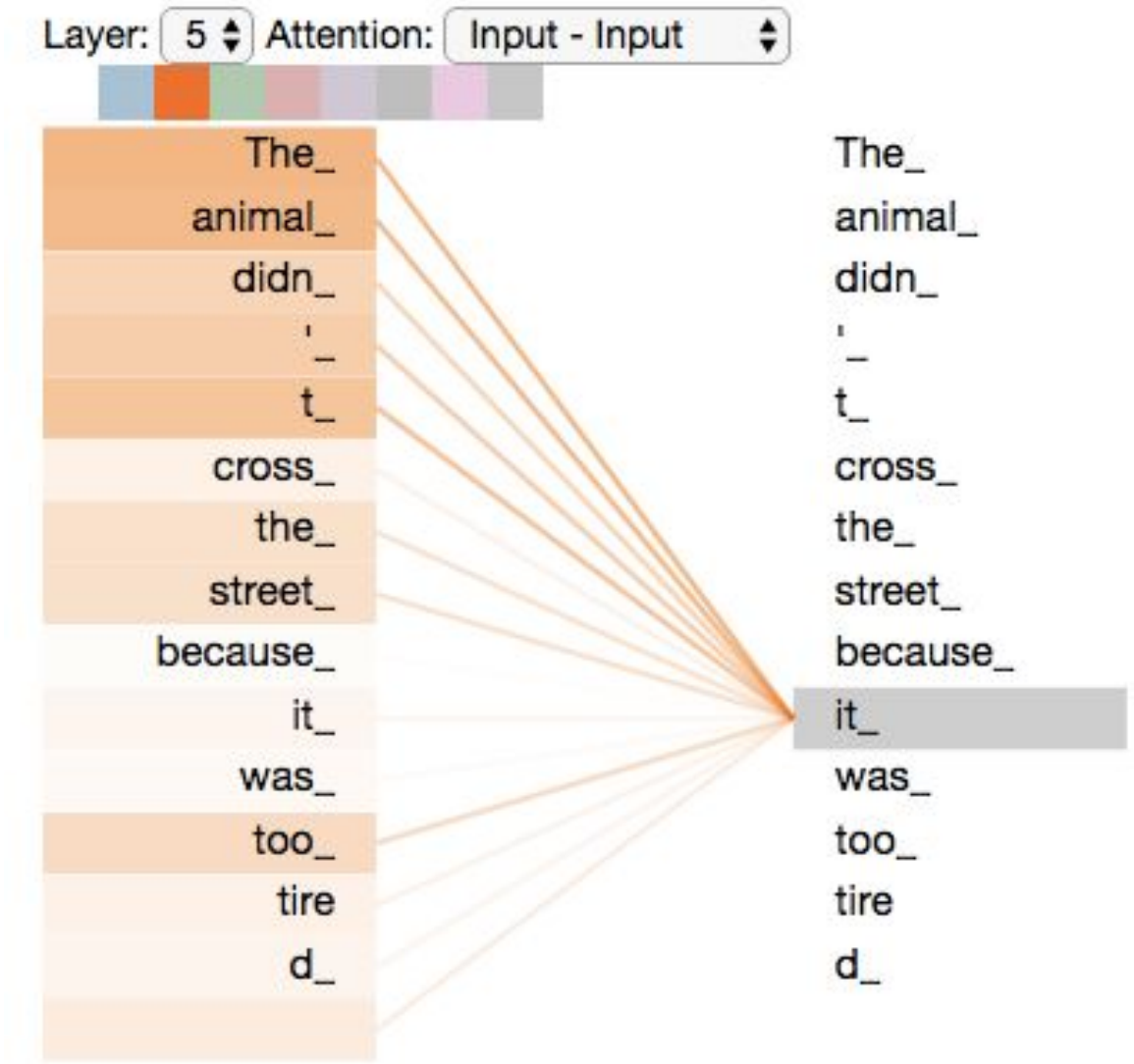
Self-Attention (4)



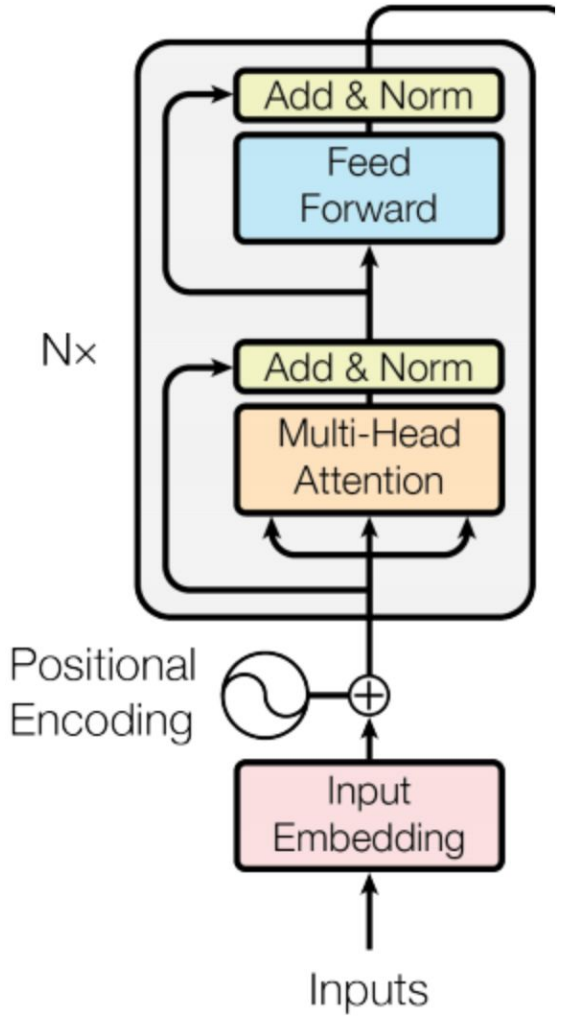
Multi-Head Attention



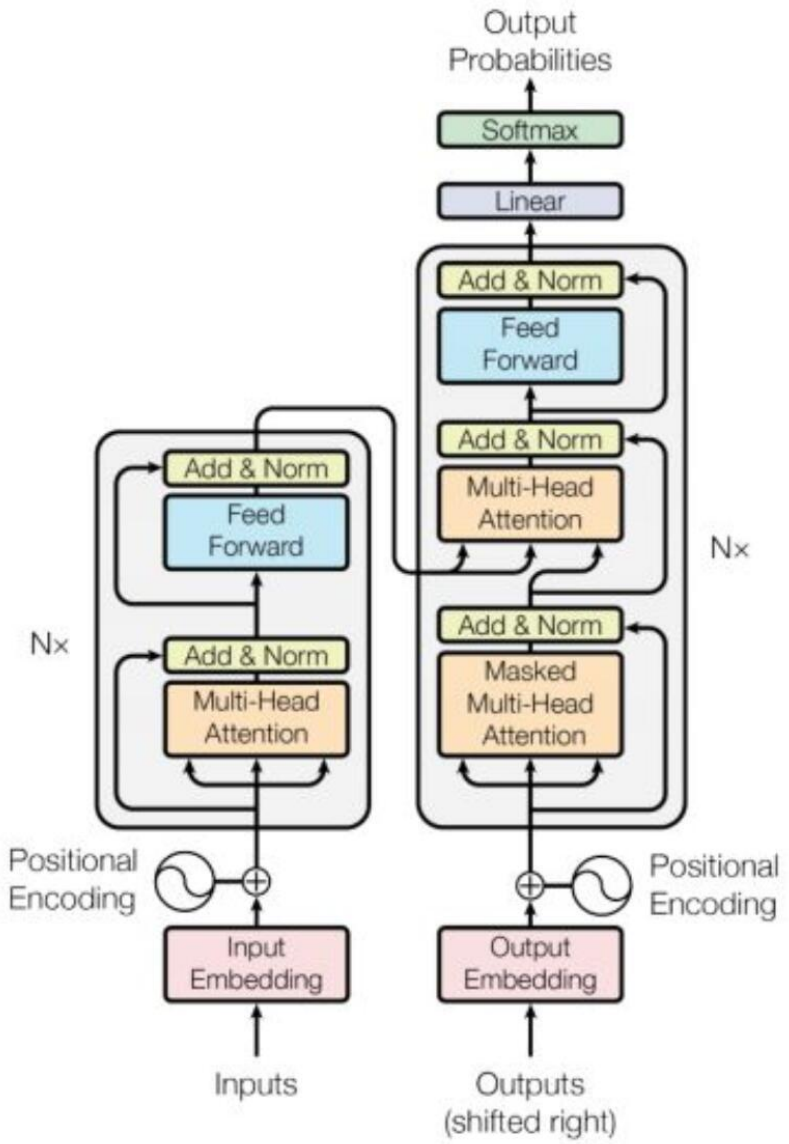
Attention Map



Transformer Encoder



Transformer



BERT

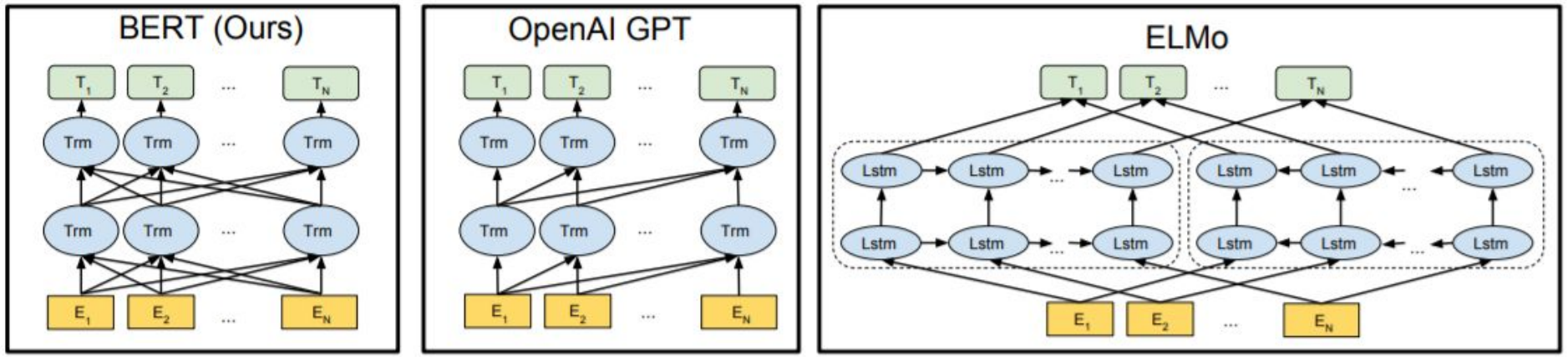
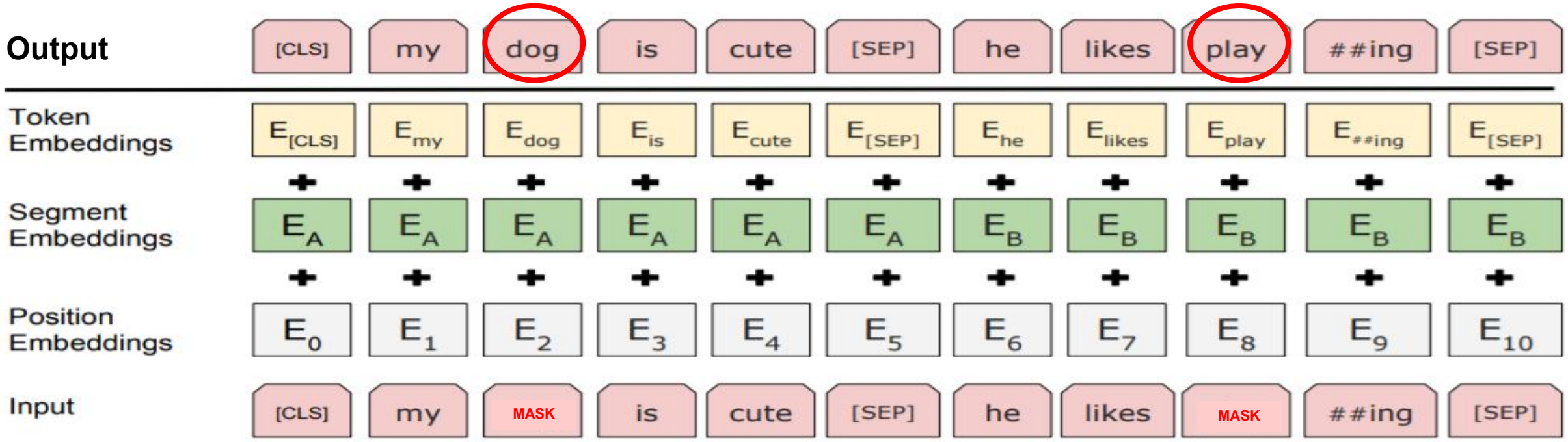


Figure 1: Differences in pre-training model architectures. BERT uses a bidirectional Transformer. OpenAI GPT uses a left-to-right Transformer. ELMo uses the concatenation of independently trained left-to-right and right-to-left LSTM to generate features for downstream tasks. Among three, only BERT representations are jointly conditioned on both left and right context in all layers.

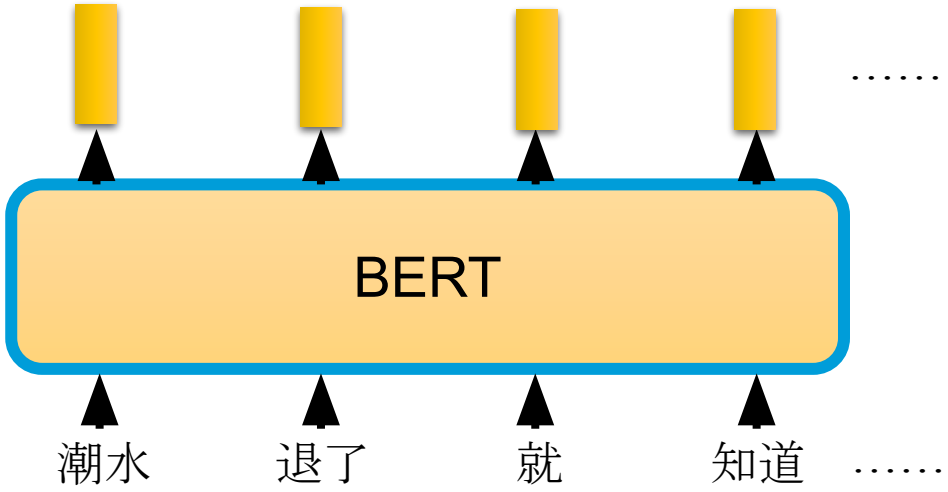
BERT - Mask LLM

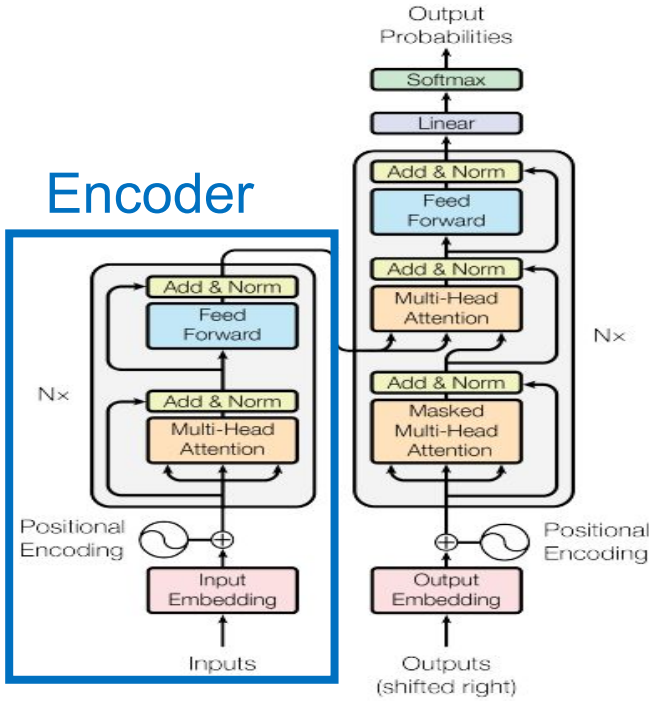


Bidirectional Encoder Representations from Transformers (BERT)

- BERT = Encoder of Transformer

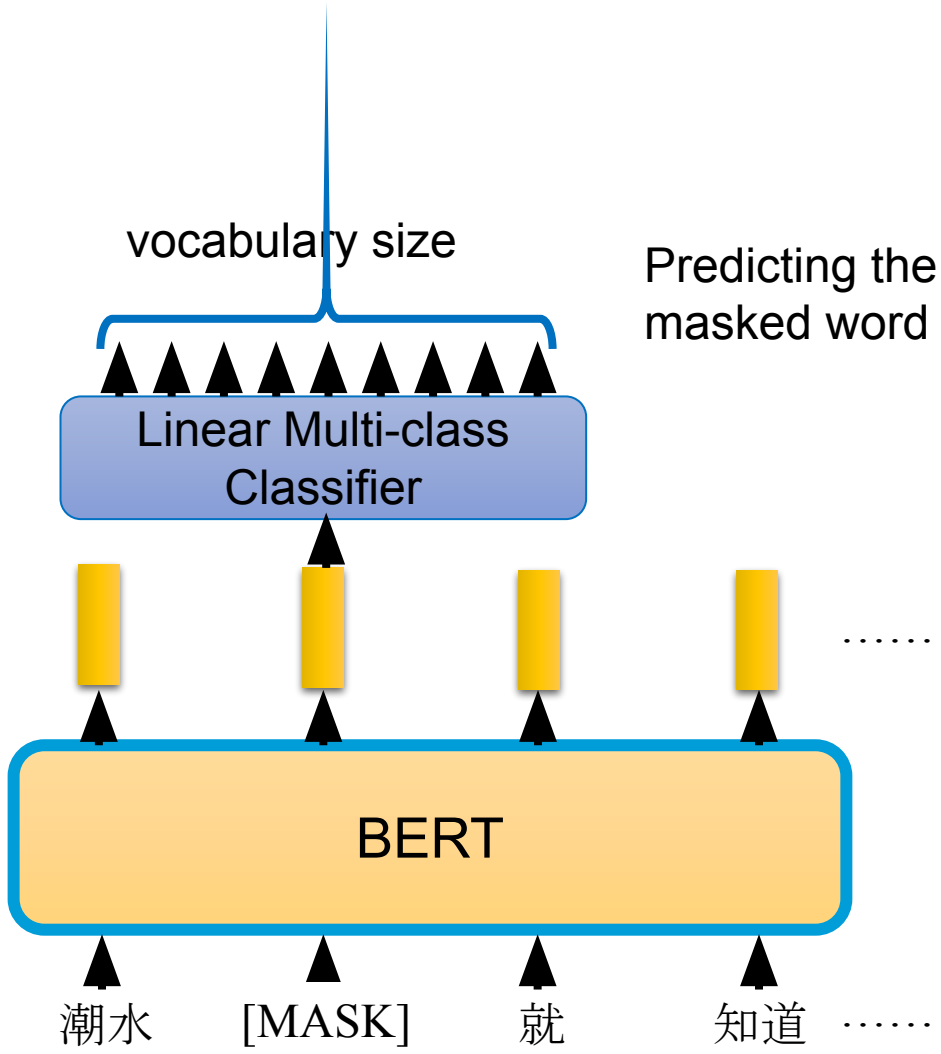
Learned from a large amount of text without annotation





Pre-Training of BERT (1)

- Approach 1:
  Masked LM

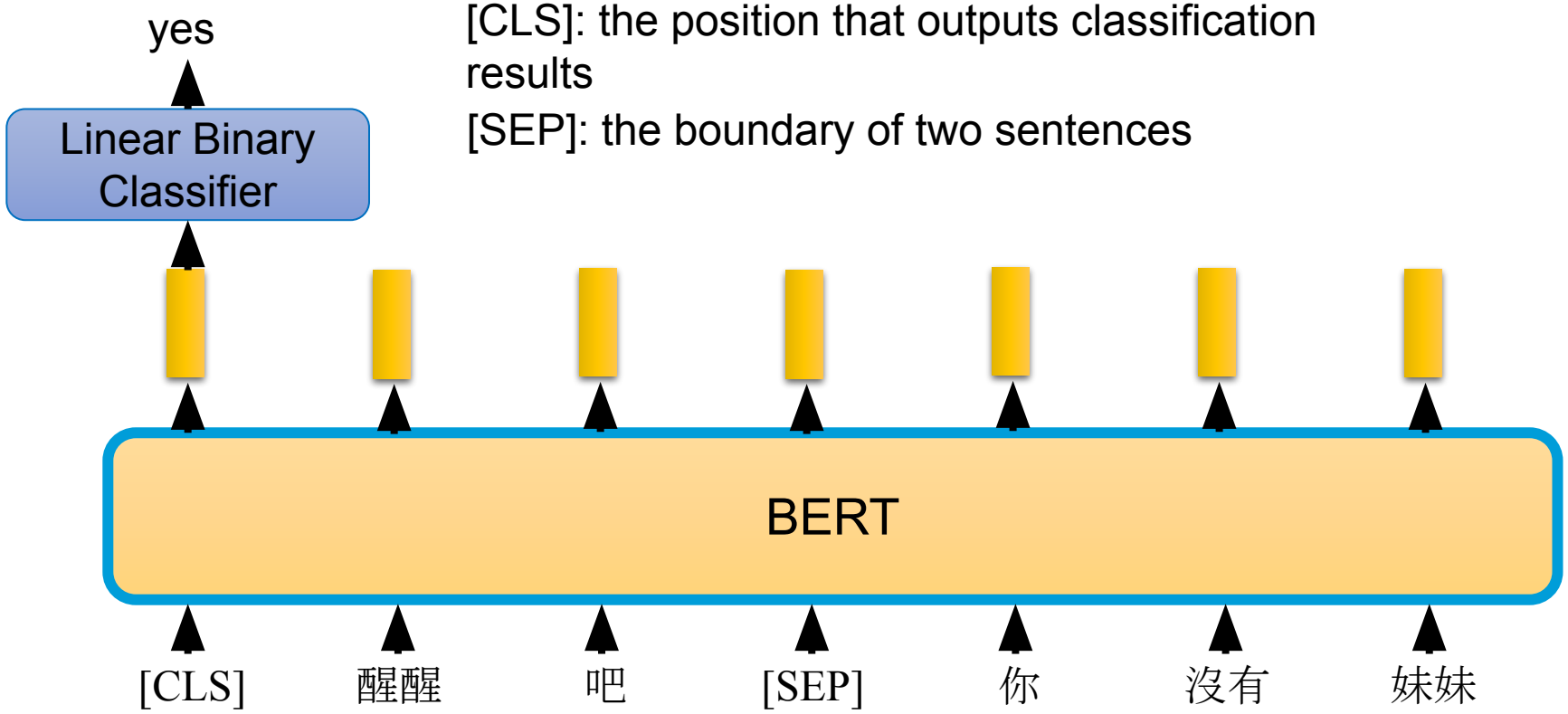


Pre-Training of BERT (2)

Approach 2: Next Sentence Prediction

[CLS]: the position that outputs classification results

[SEP]: the boundary of two sentences



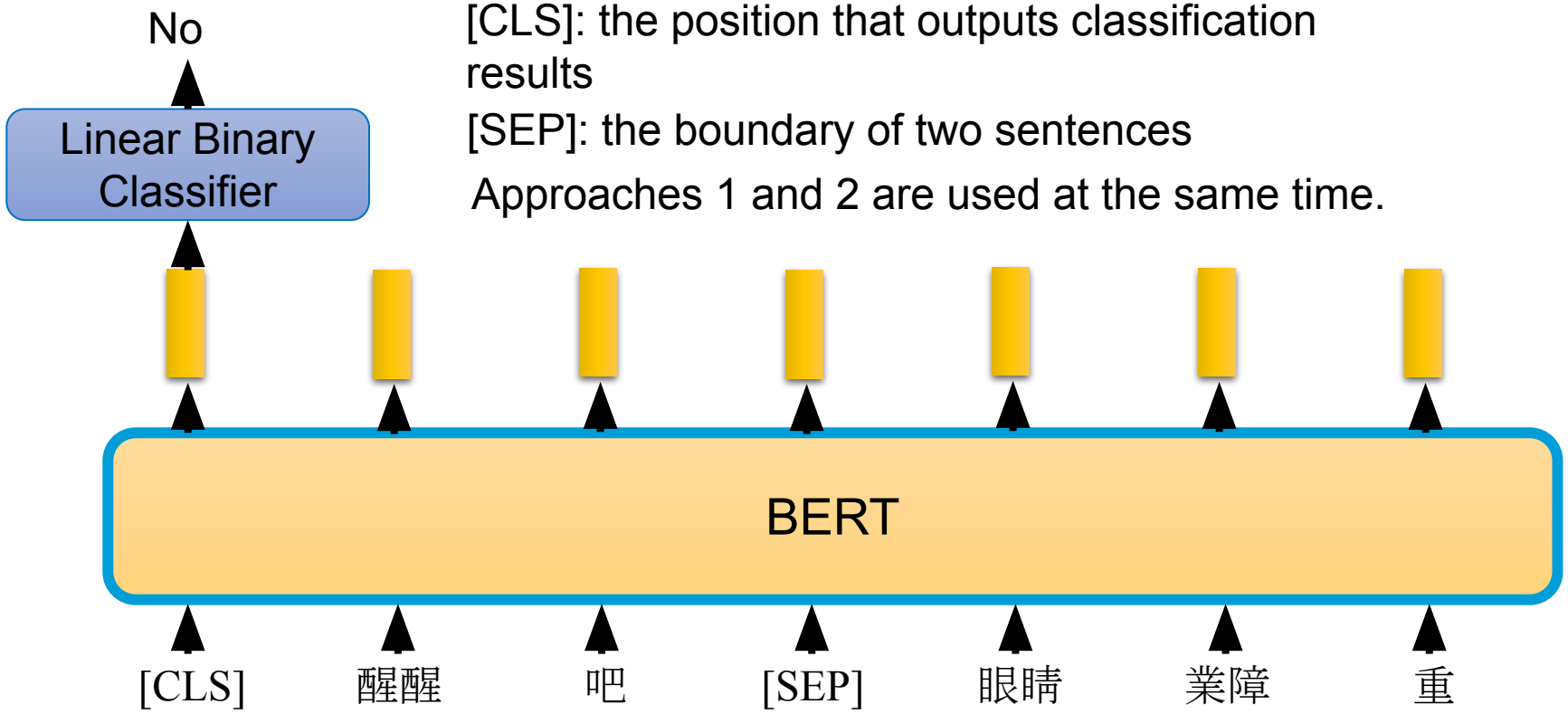
Pre-Training of BERT (3)

Approach 2: Next Sentence Prediction

[CLS]: the position that outputs classification results

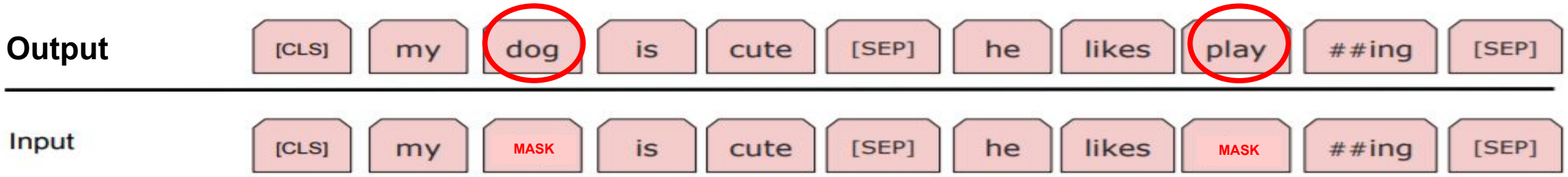
[SEP]: the boundary of two sentences

Approaches 1 and 2 are used at the same time.



BERT - Mask LLM

- Randomly Mask 15% of all tokens
  - Inspired by *Cloze* Task
- Issue:
  - Mask won't be in downstream tasks
  - Only predict 15%, converge slower



BERT - Mask LLM

- Issue: Mask won't be in downstream tasks
  - 80%: use mask
  - 10%: use random word
  - 10% use original word (bias representation towards actual word)
- Encoder doesn't know which word it will be asked to predict
  - Learn contextual representation of *every* word
- Random replacement only 1.5% overall

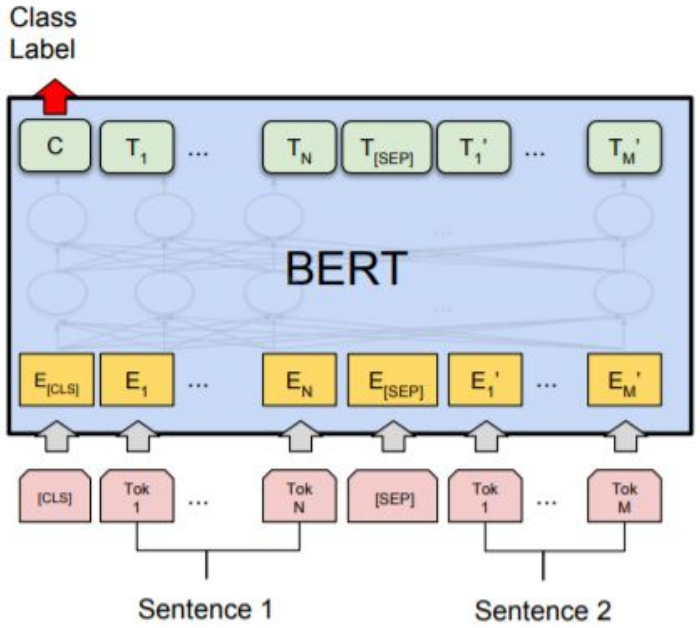
BERT - Next Sentence Prediction

- For MRC/NLI it's important to understand *relationship* between sentences
- 50% random replace actual next sentence

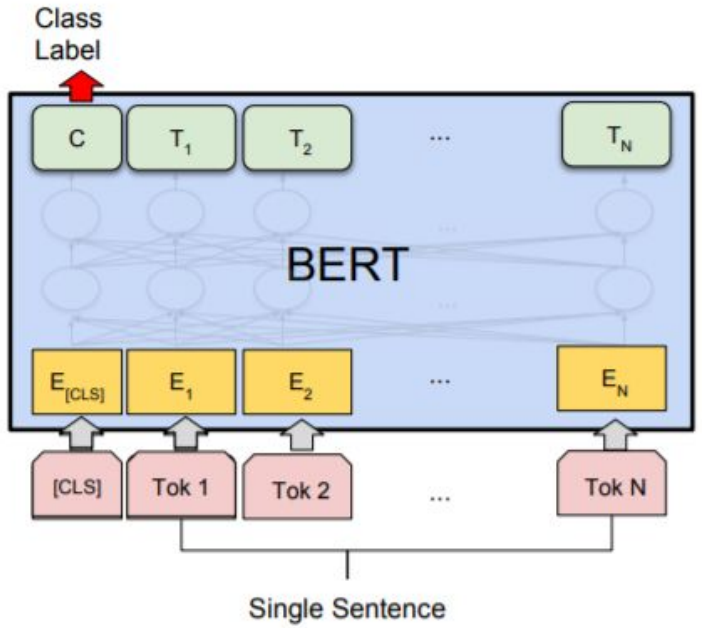
$$\text{Input} = \texttt{[CLS] the man went to [MASK] store [SEP]}$$
$$\texttt{he bought a gallon [MASK] milk [SEP]}$$
$$\text{Label} = \texttt{IsNext}$$

$$\text{Input} = \texttt{[CLS] the man [MASK] to the store [SEP]}$$
$$\texttt{penguin [MASK] are flight ##less birds [SEP]}$$
$$\text{Label} = \texttt{NotNext}$$

BERT - Fine Tuning

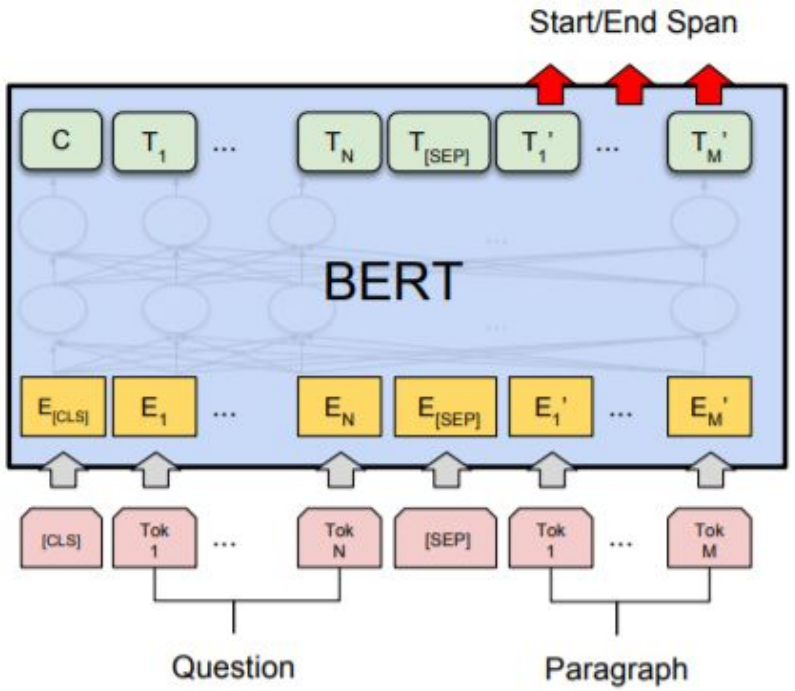


(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG

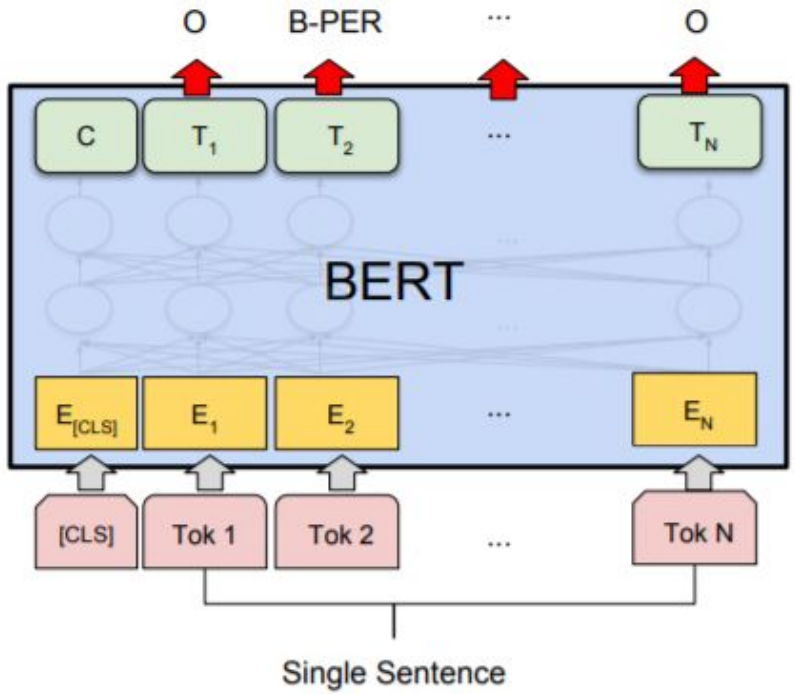


(b) Single Sentence Classification Tasks: SST-2, CoLA

BERT - Fine Tuning



(c) Question Answering Tasks:
SQuAD v1.1



(d) Single Sentence Tagging Tasks:
CoNLL-2003 NER

BERT - Fine Tuning

- **Input Question:**

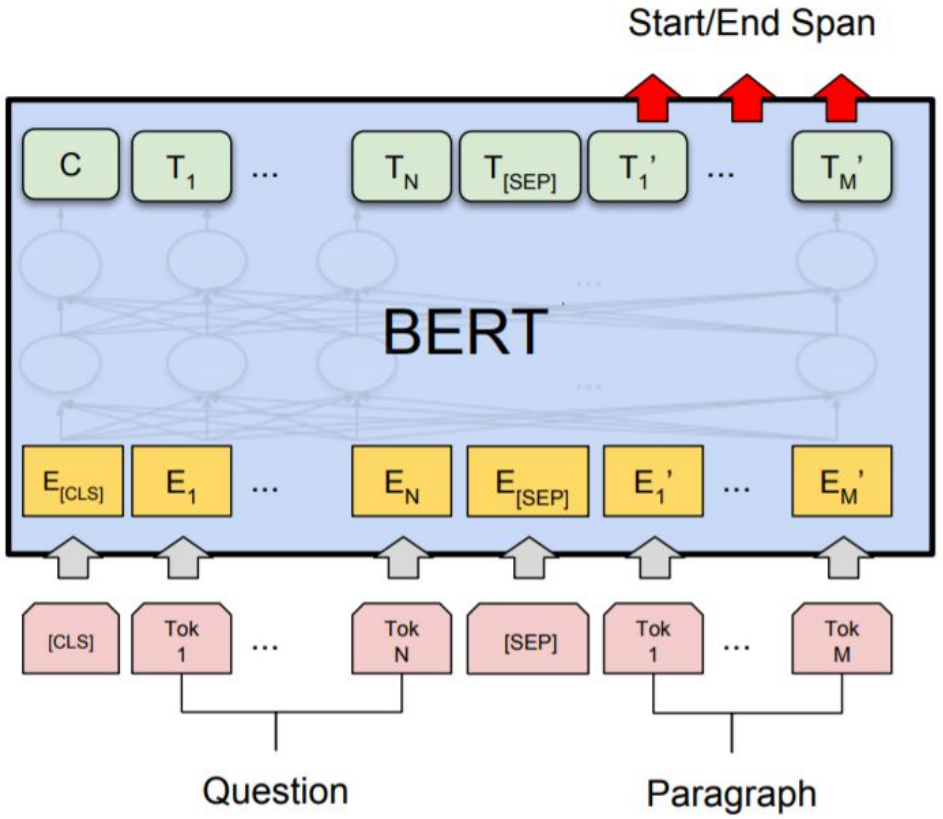
  Where do water droplets collide with ice crystals to form precipitation?

- **Input Paragraph:**

  ... Precipitation forms as smaller droplets coalesce via collision with other rain drops or ice crystals within a cloud. ...

- **Output Answer:**

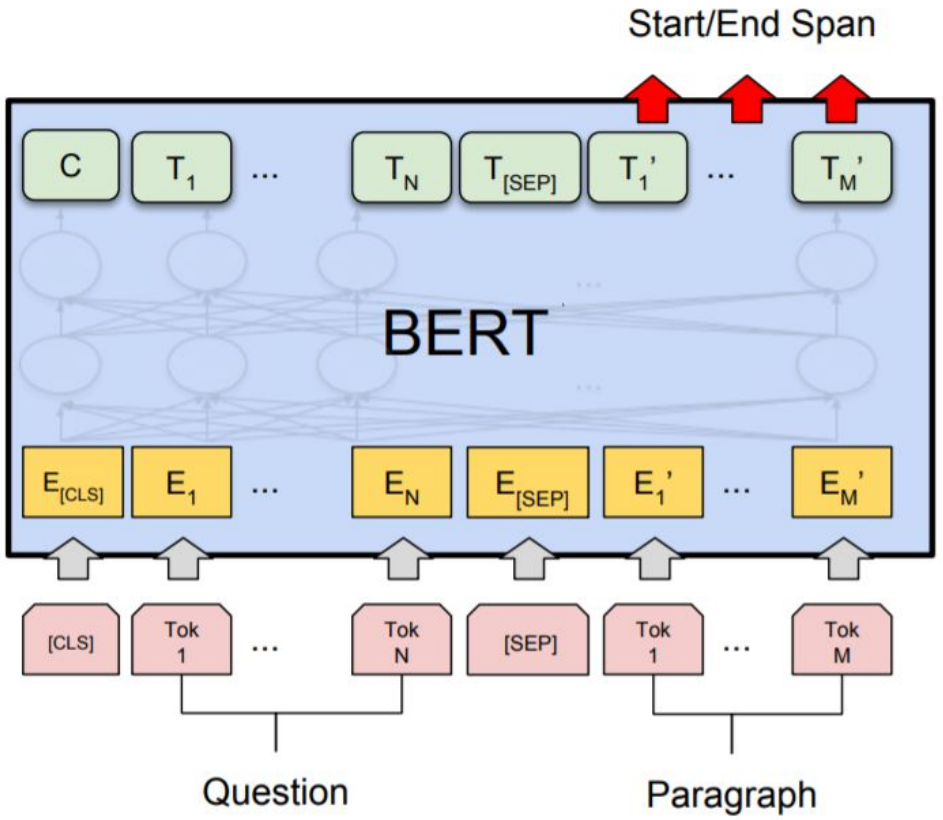
  within a cloud



BERT - Fine Tuning

- Start vector *S*
- End vector *E*

$$P_i = \frac{e^{S \cdot T_i}}{\sum_j e^{S \cdot T_j}}$$



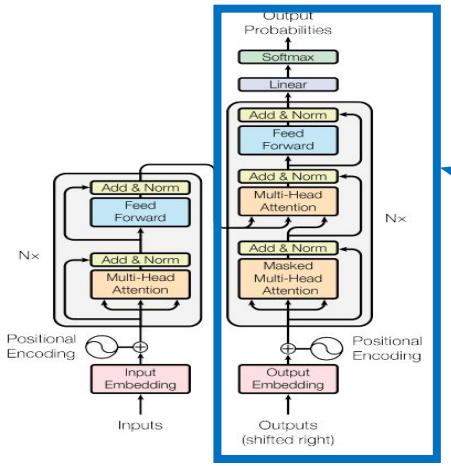
BERT - Training Details

- 3.3B word corpus * 40 epochs
- 256 sequence * 512 tokens / mini-batch
- Adam with warmup, linear decay
- *gelu* (Hendrycks and Gimpel, 2016) instead of *relu*
- 4 TPU Pods (16 TPUs) for 4 days (~1 year w/ 8x P100)

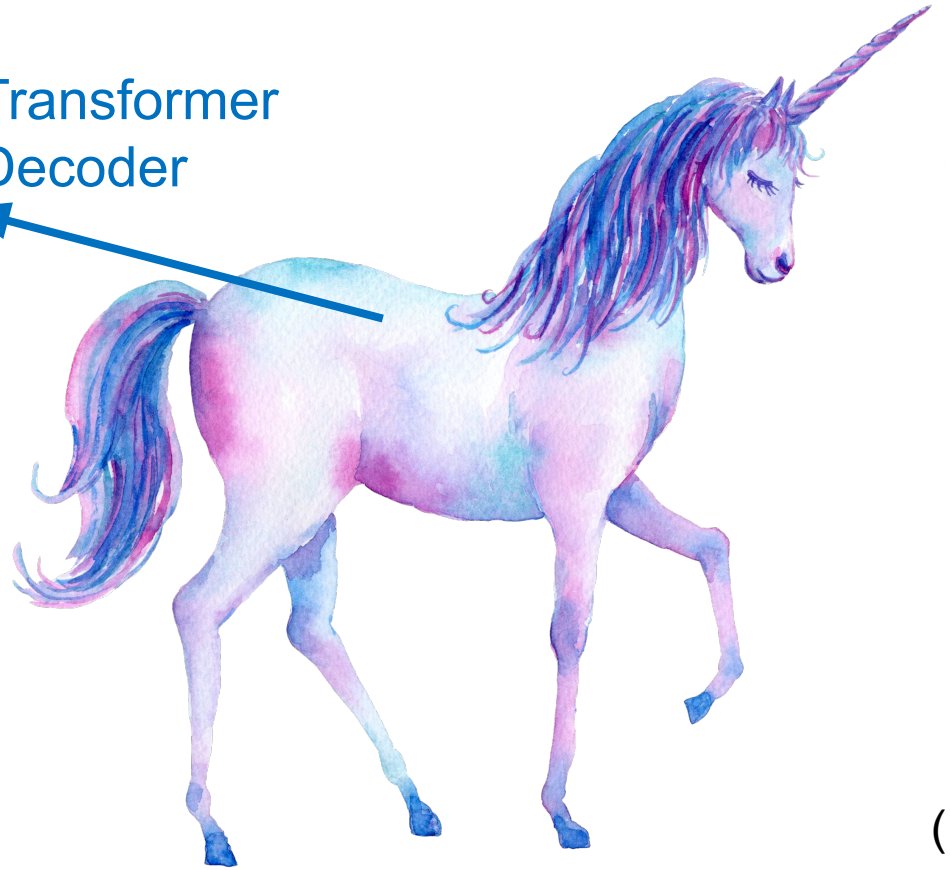
BERT - Results

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	**Average** -
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.9	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	88.1	91.3	45.4	80.0	82.3	56.0	75.2
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.1	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**91.1**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**81.9**

Generative Pre-trained Transformer (GPT)



Transformer Decoder



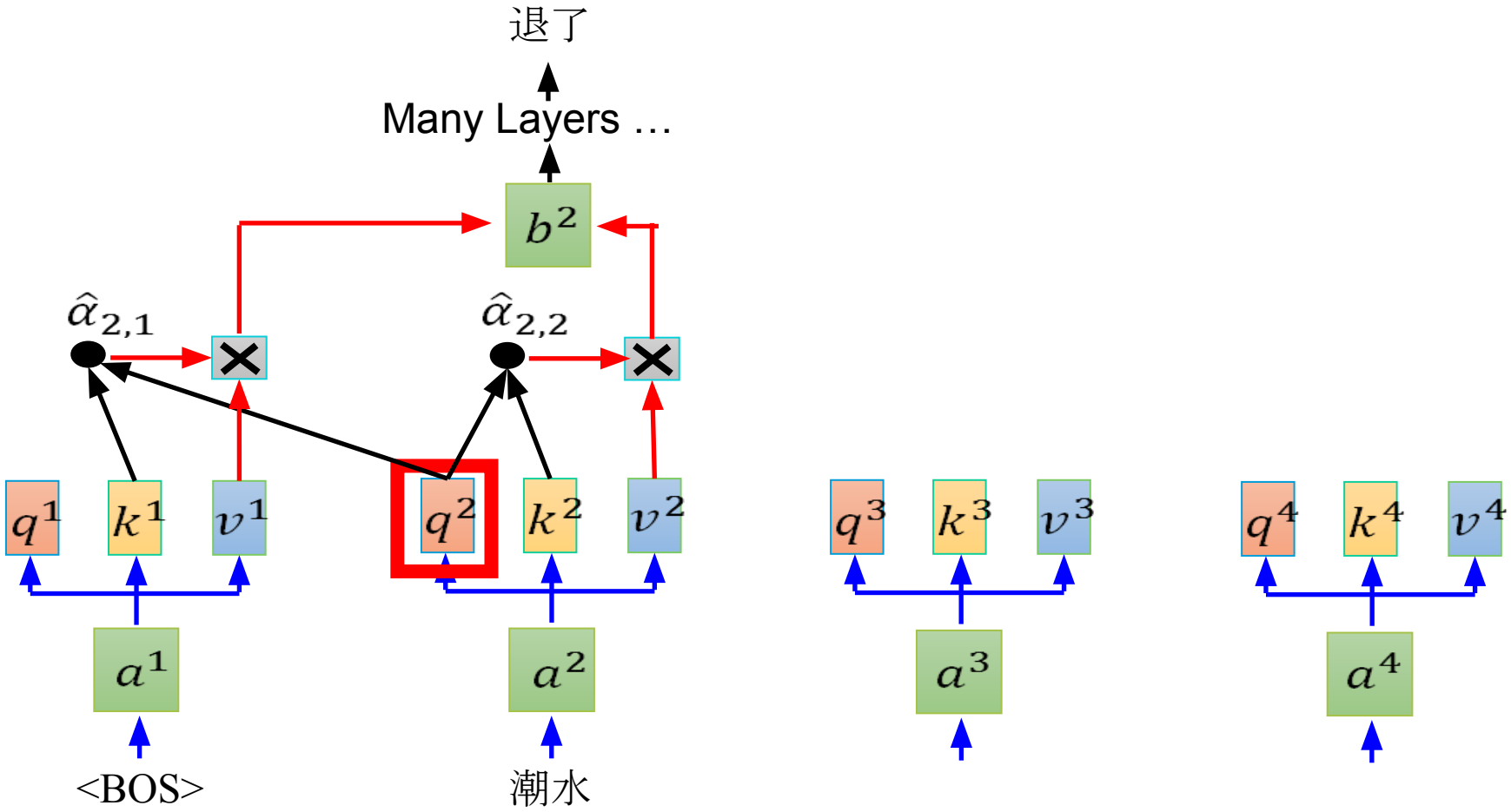
BERT (340M)

ELMO (94M)

GPT-2 (1542M)

Generative Pre-trained Transformer (GPT)

Training: Next token predication



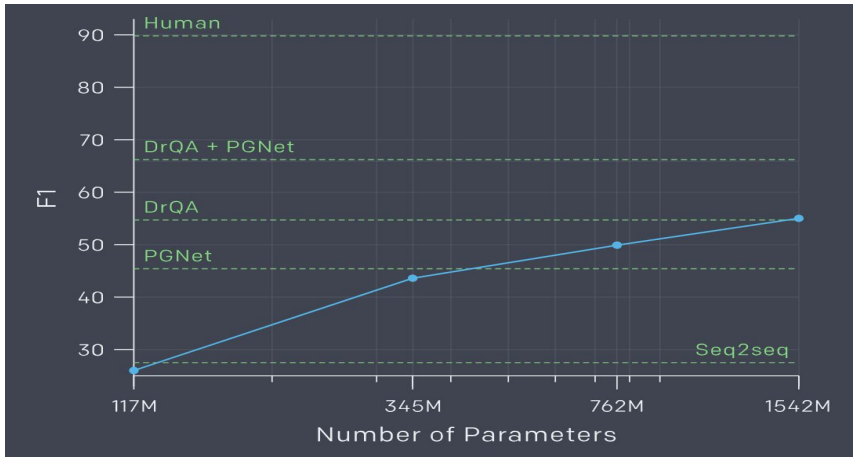
GPT

Zero-shot Learning?

- ***Reading Comprehension***

$$d_1, d_2, \cdots, d_N,$$
$$\text{"Q:"}, q_1, q_2, \cdots, q_N,$$
$$\text{"A:"}$$

CoQA



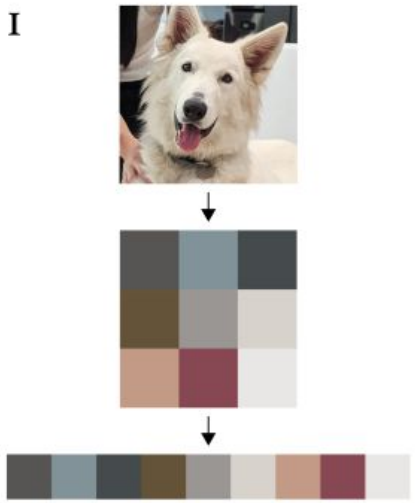
- ***Summarization*** $d_1, d_2, \cdots, d_N, \text{"TL;DR:"}$

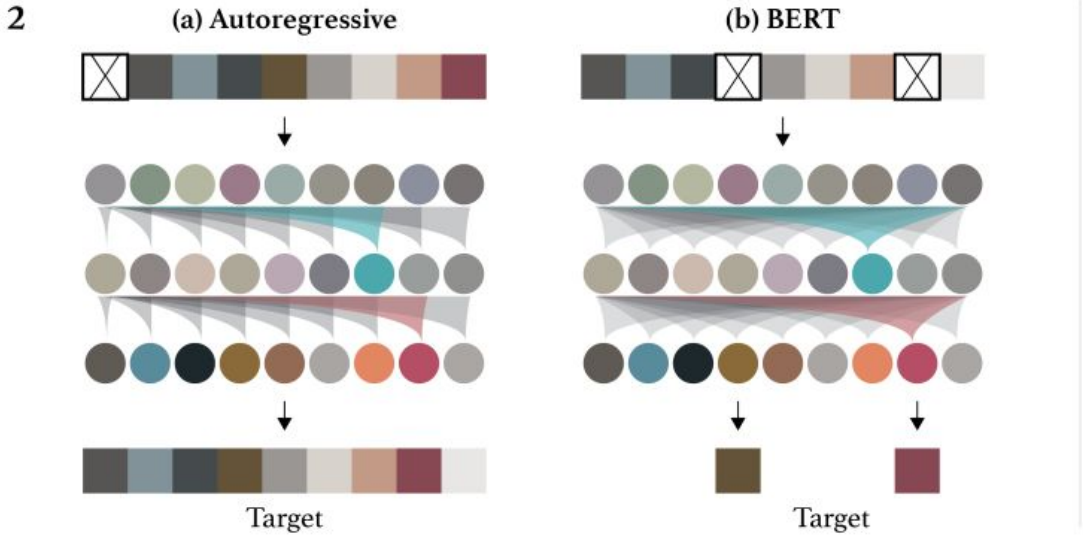
- ***Translation***

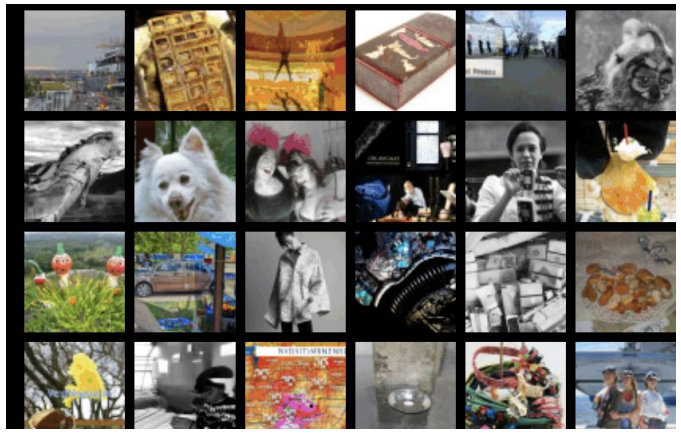
English sentence 1	=	French sentence 1
English sentence 2	=	French sentence 2
English sentence 3	=	

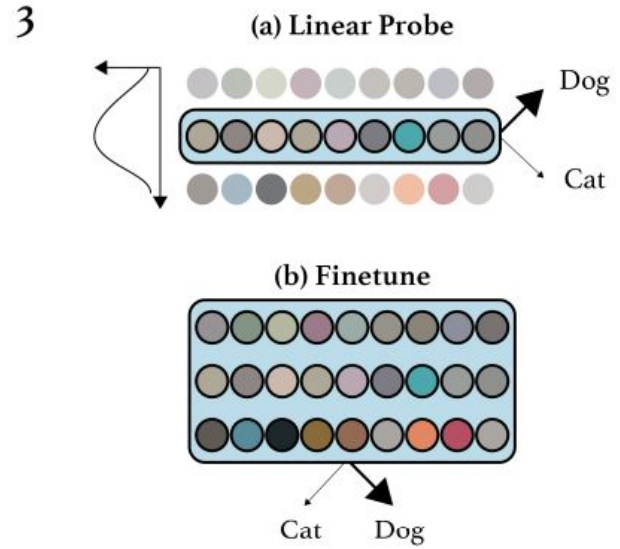
Image GPT?

Yes, we have

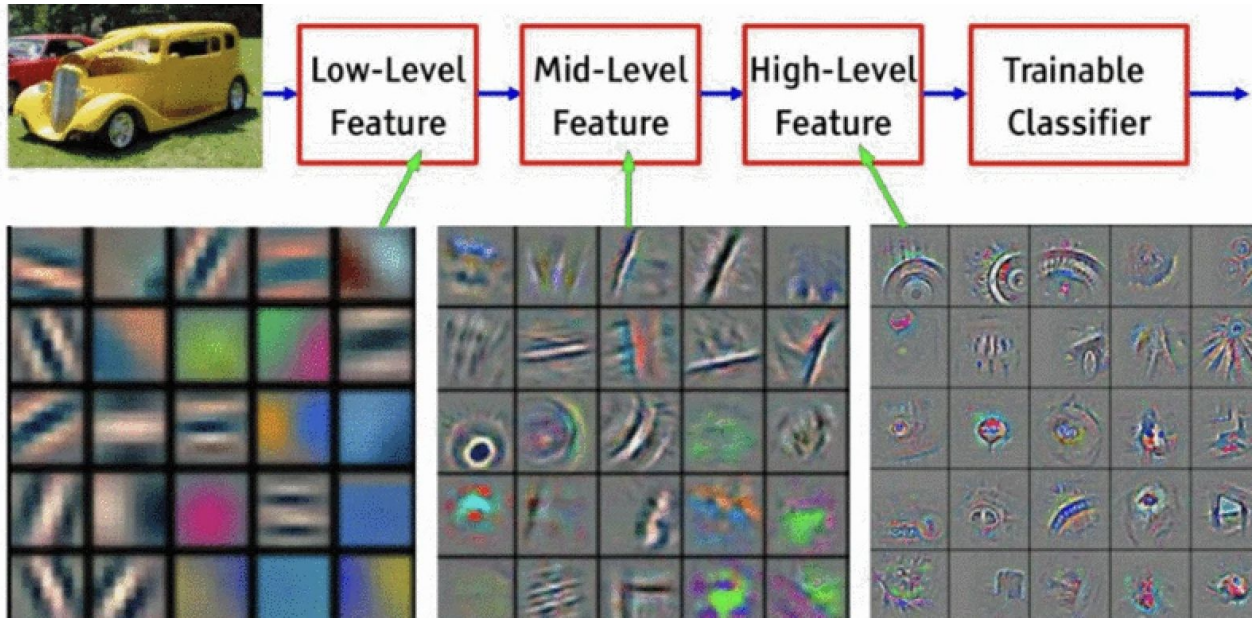




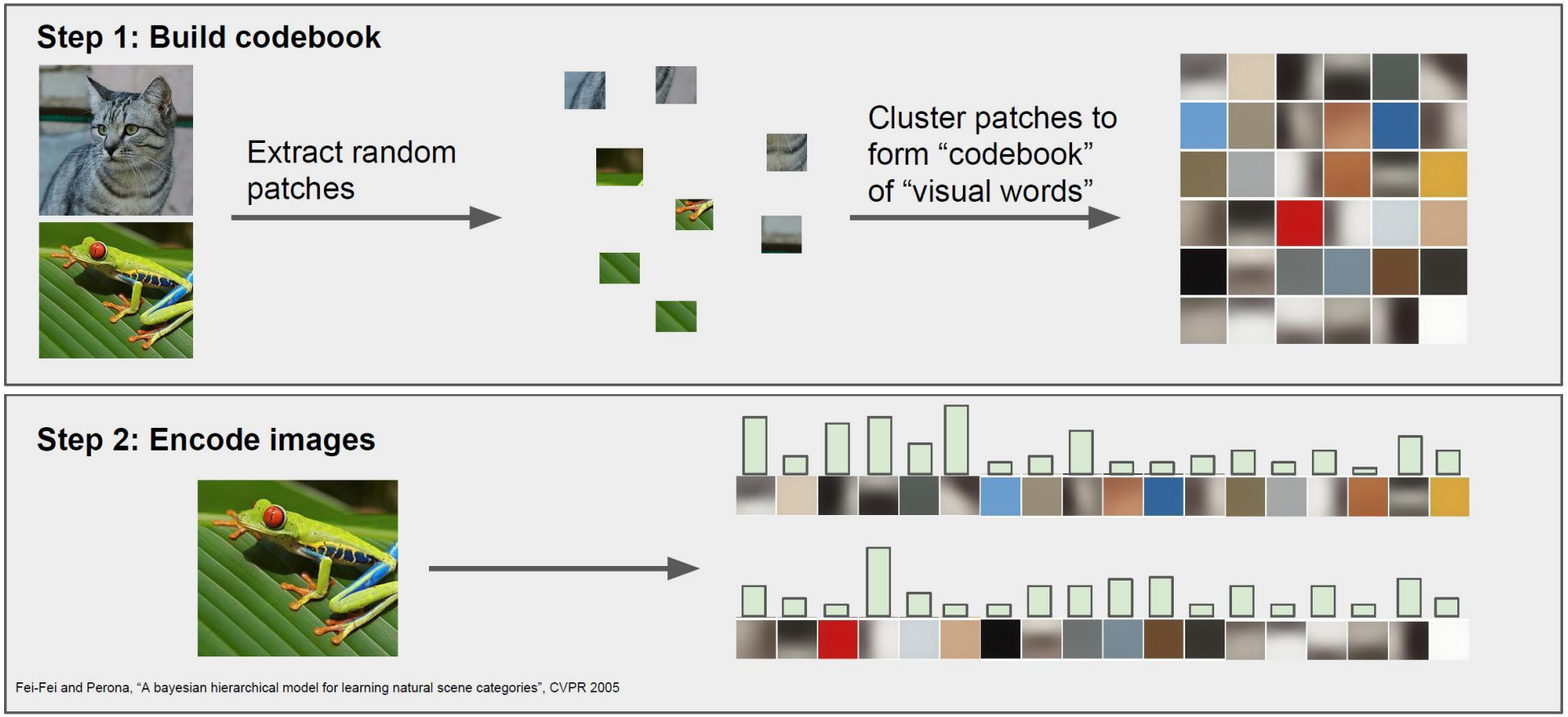




Convolutional Neural Network



Previous Feature Extraction



Fei-Fei and Perona, “A bayesian hierarchical model for learning natural scene categories”, CVPR 2005

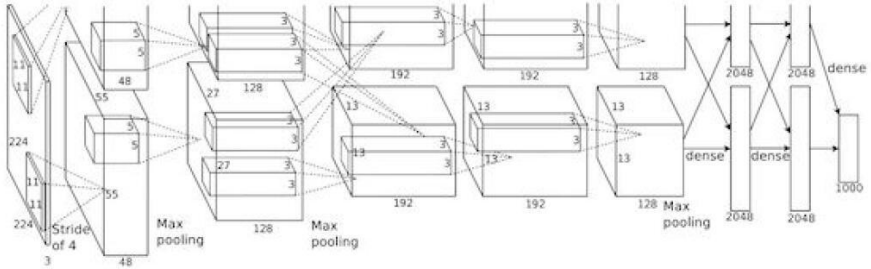
Feature Extraction with DL

Feature Extraction

f

10 numbers giving scores for classes

training

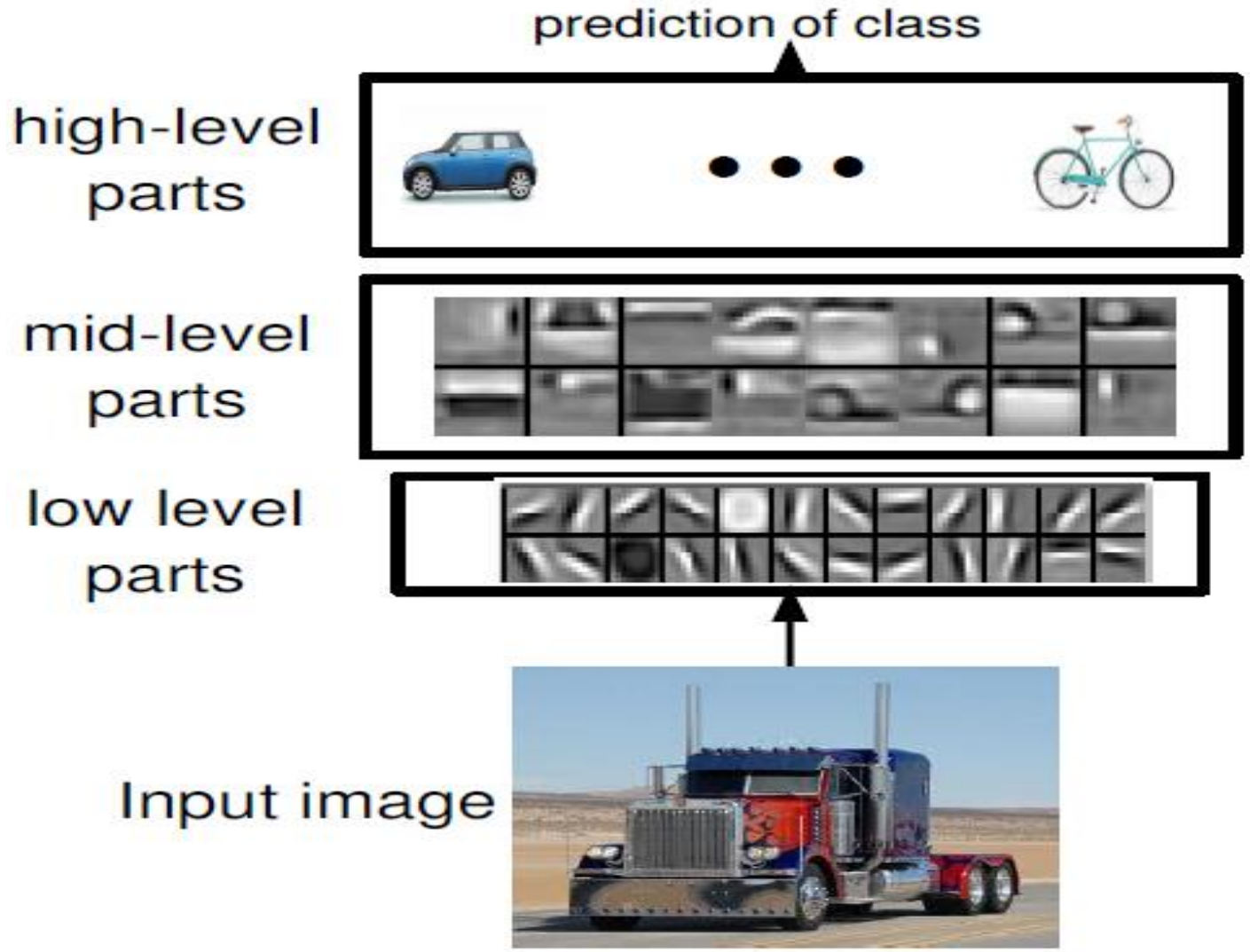


Krizhevsky, Sutskever, and Hinton, "Imagenet classification with deep convolutional neural networks", NIPS 2012. Figure copyright Krizhevsky, Sutskever, and Hinton, 2012. Reproduced with permission.

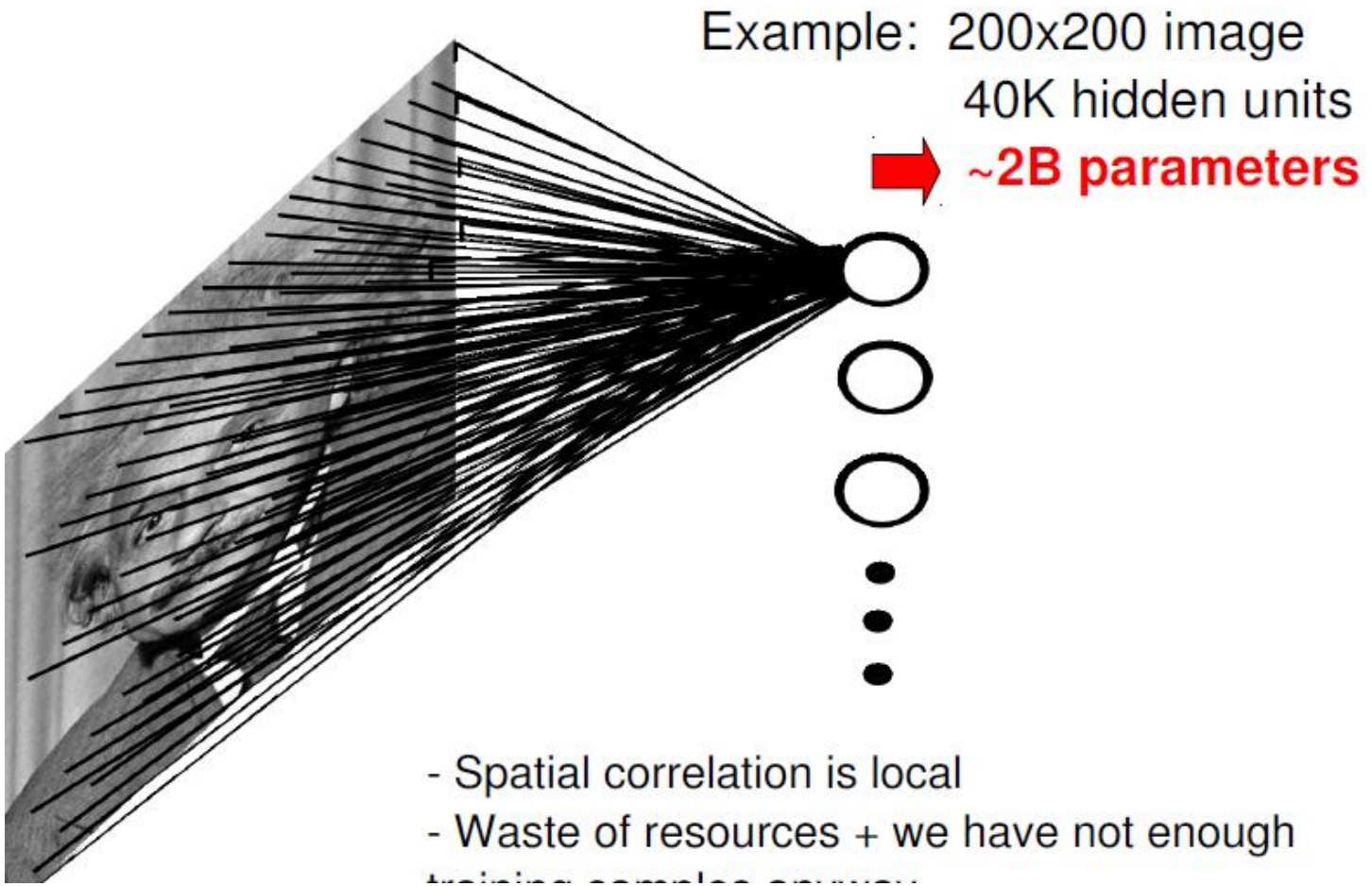
10 numbers giving scores for classes

training

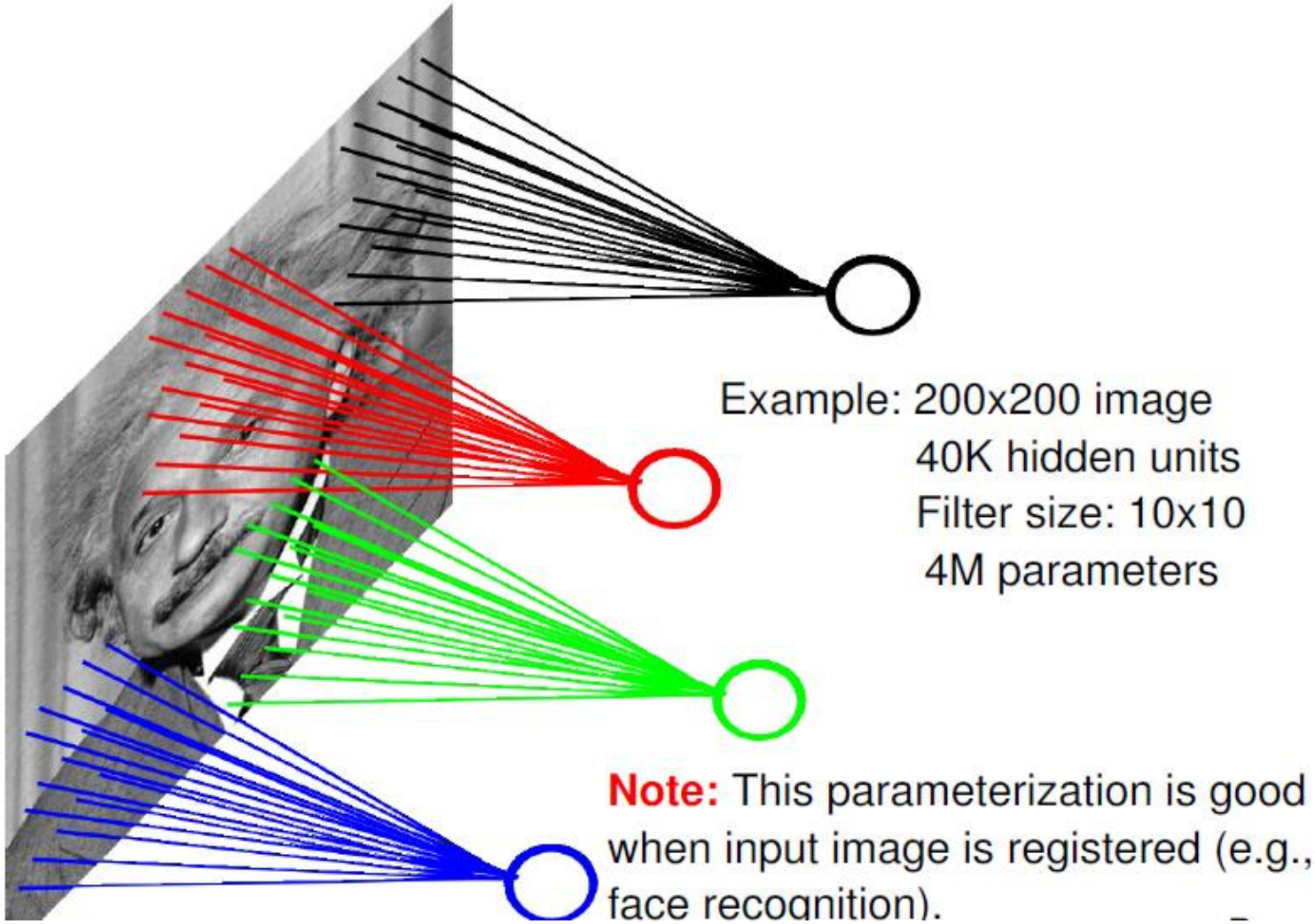
Convolutional Neural Networks



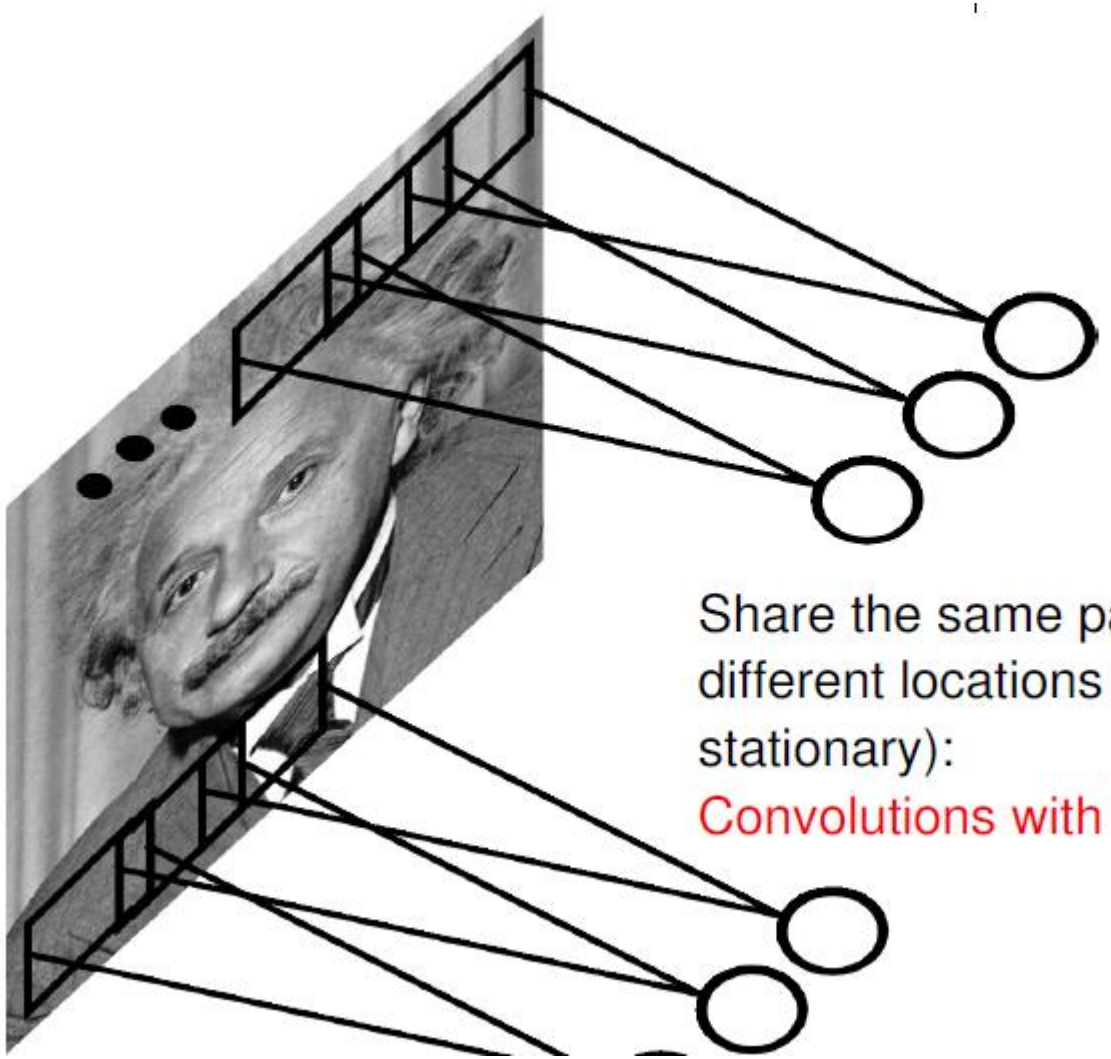
Fully Connected Layer



Locally Connected Layer



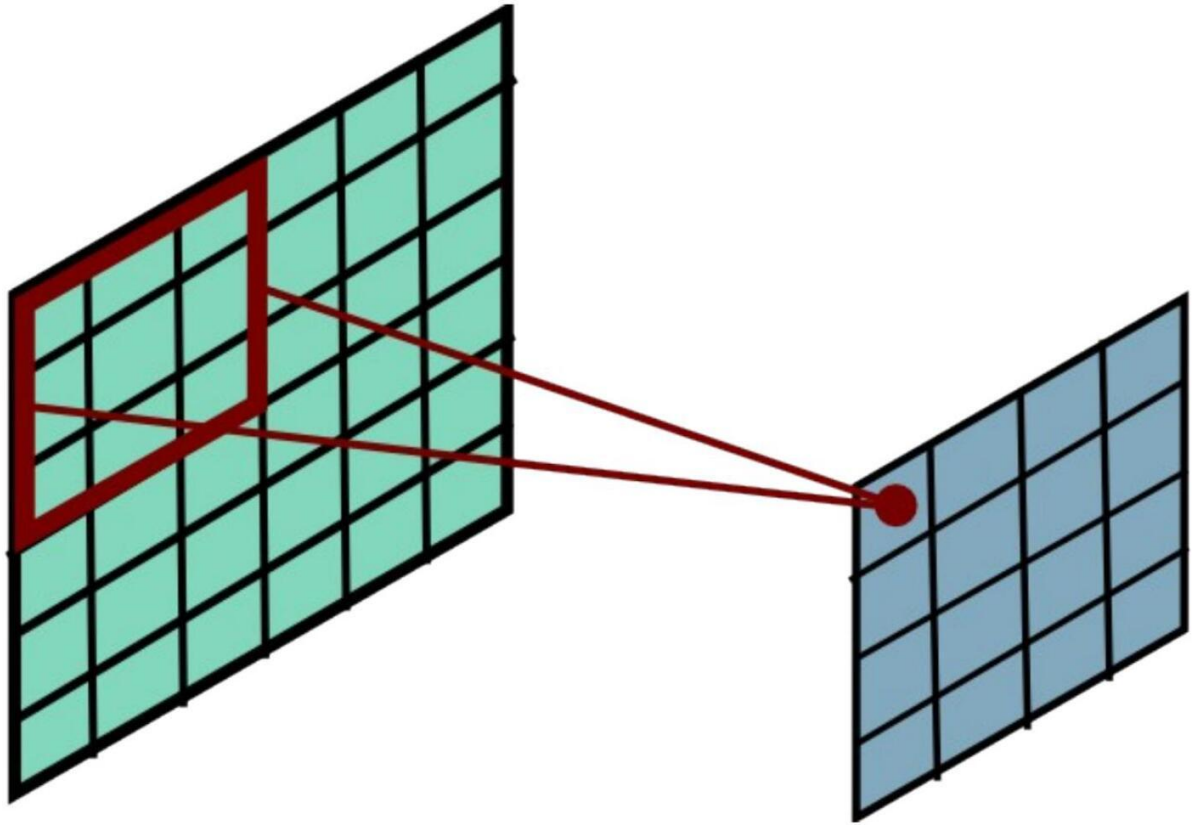
Convolutional Layer



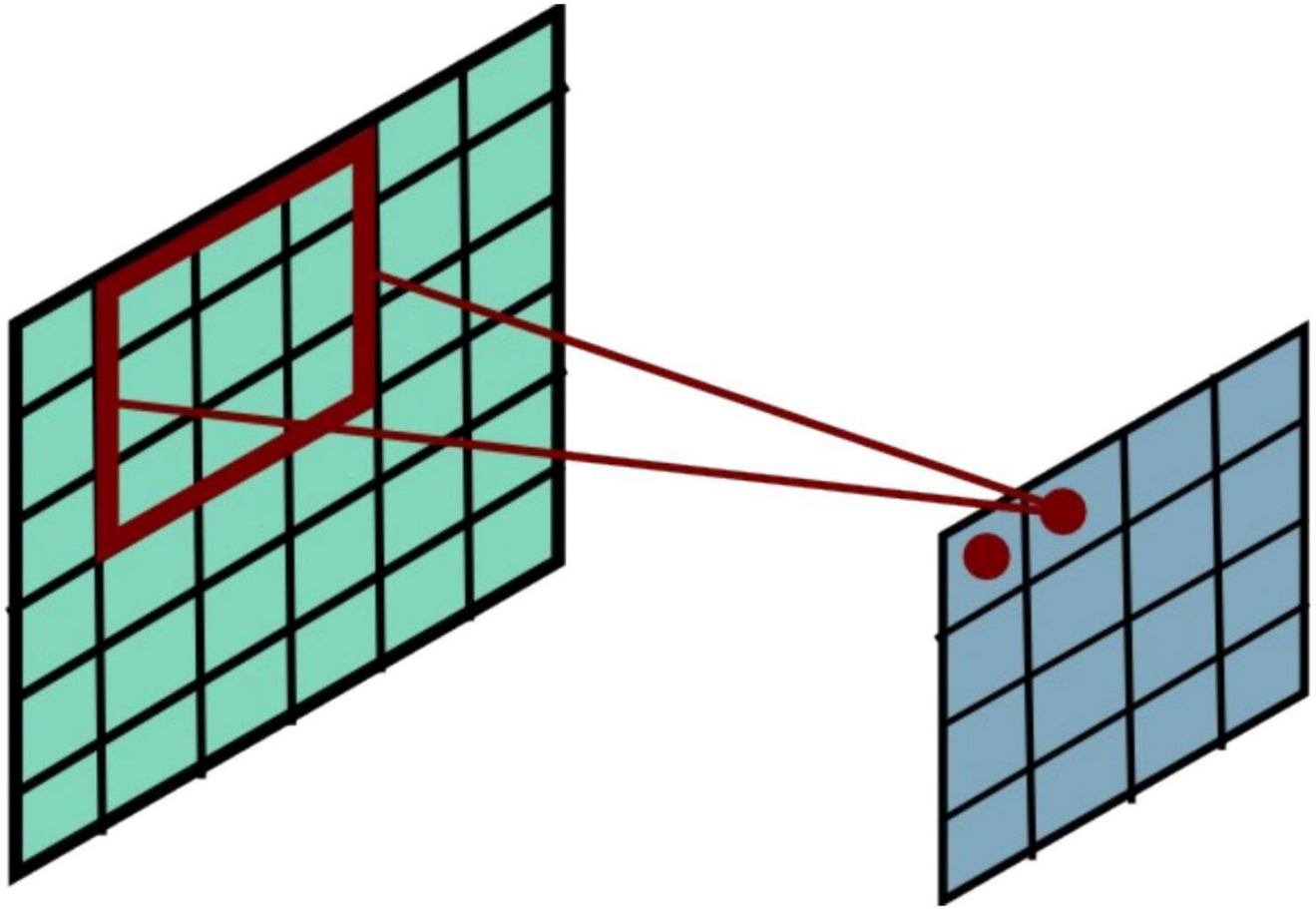
Share the same parameters across different locations (assuming input is stationary):

Convolutions with learned kernels

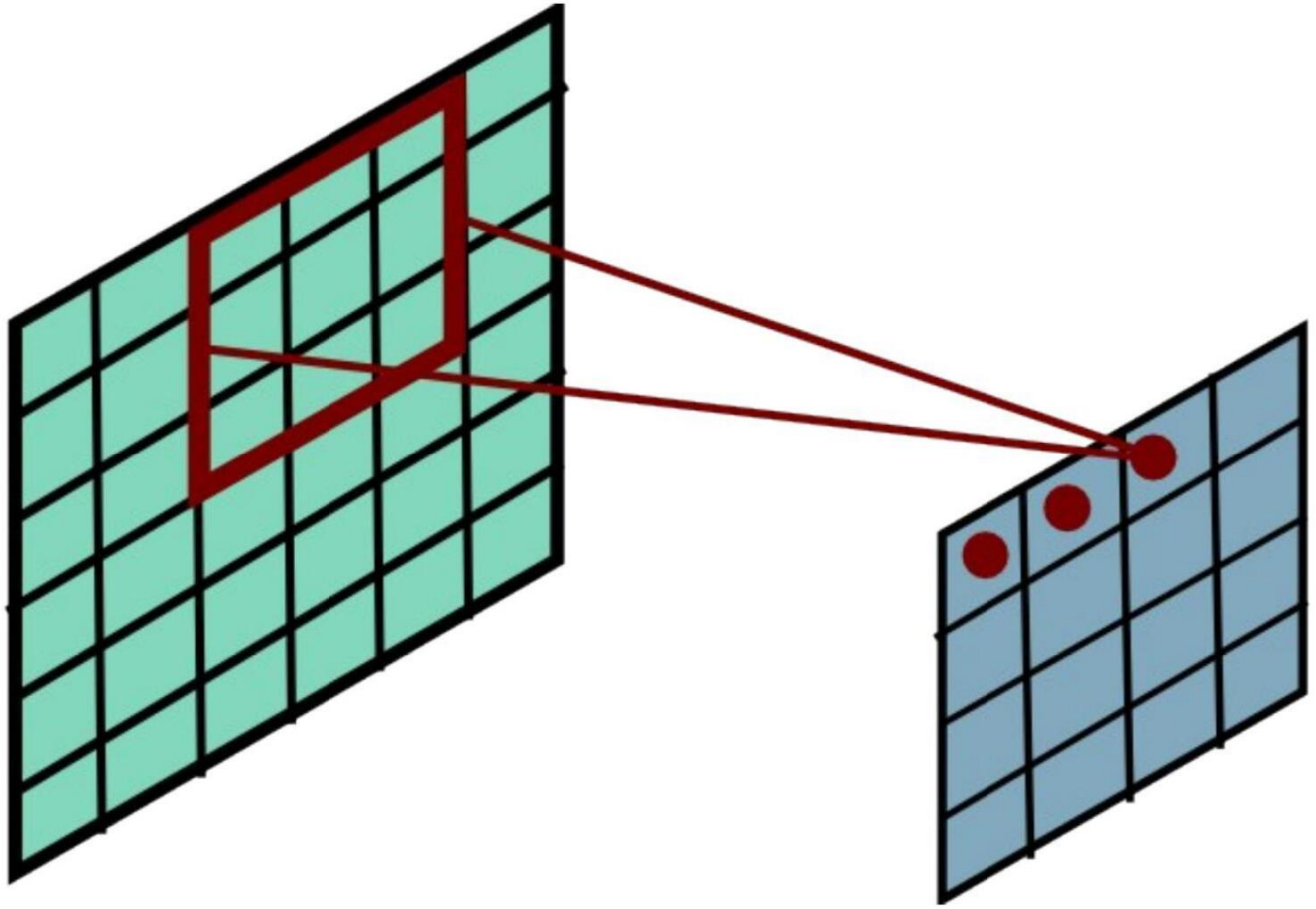
Convolutional Layer



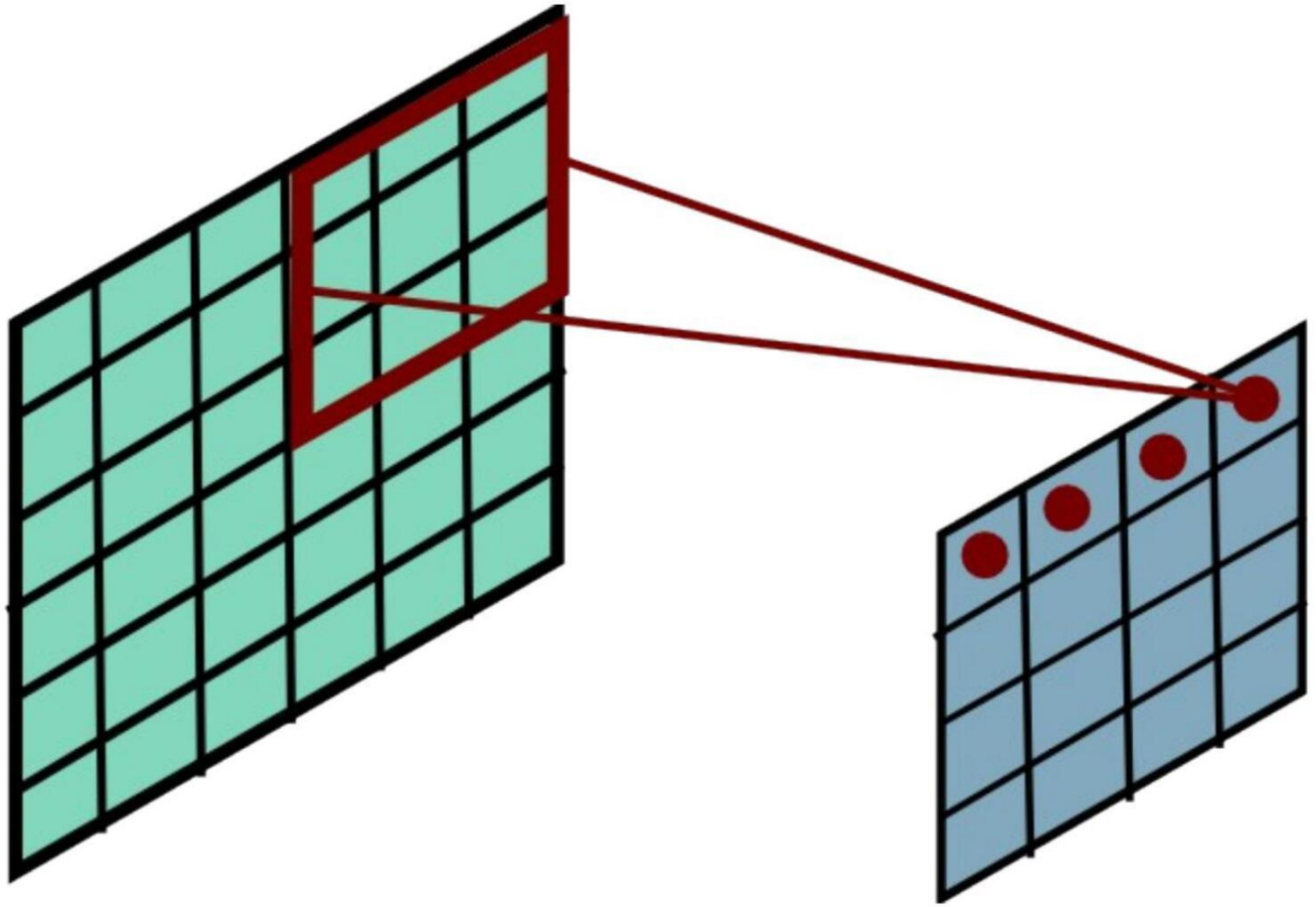
Convolutional Layer



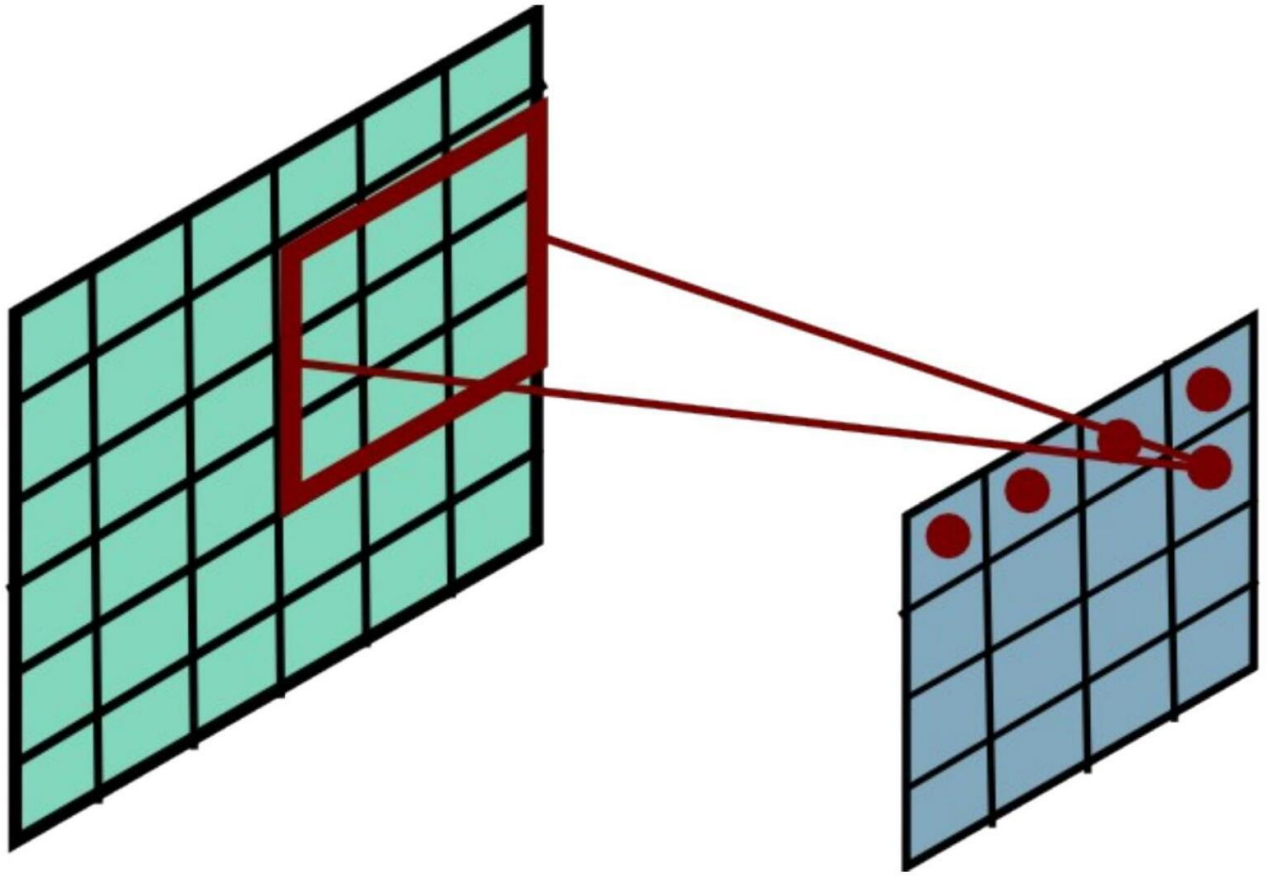
Convolutional Layer



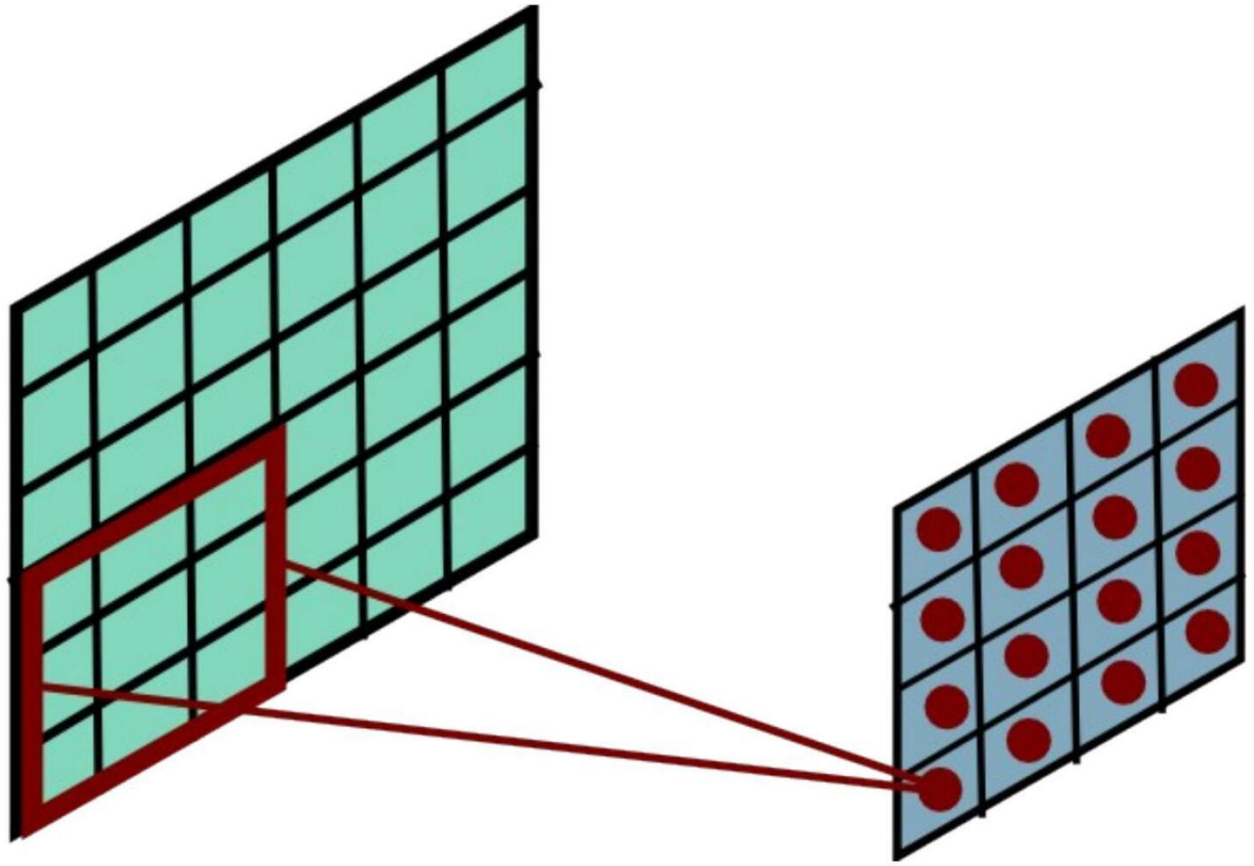
Convolutional Layer



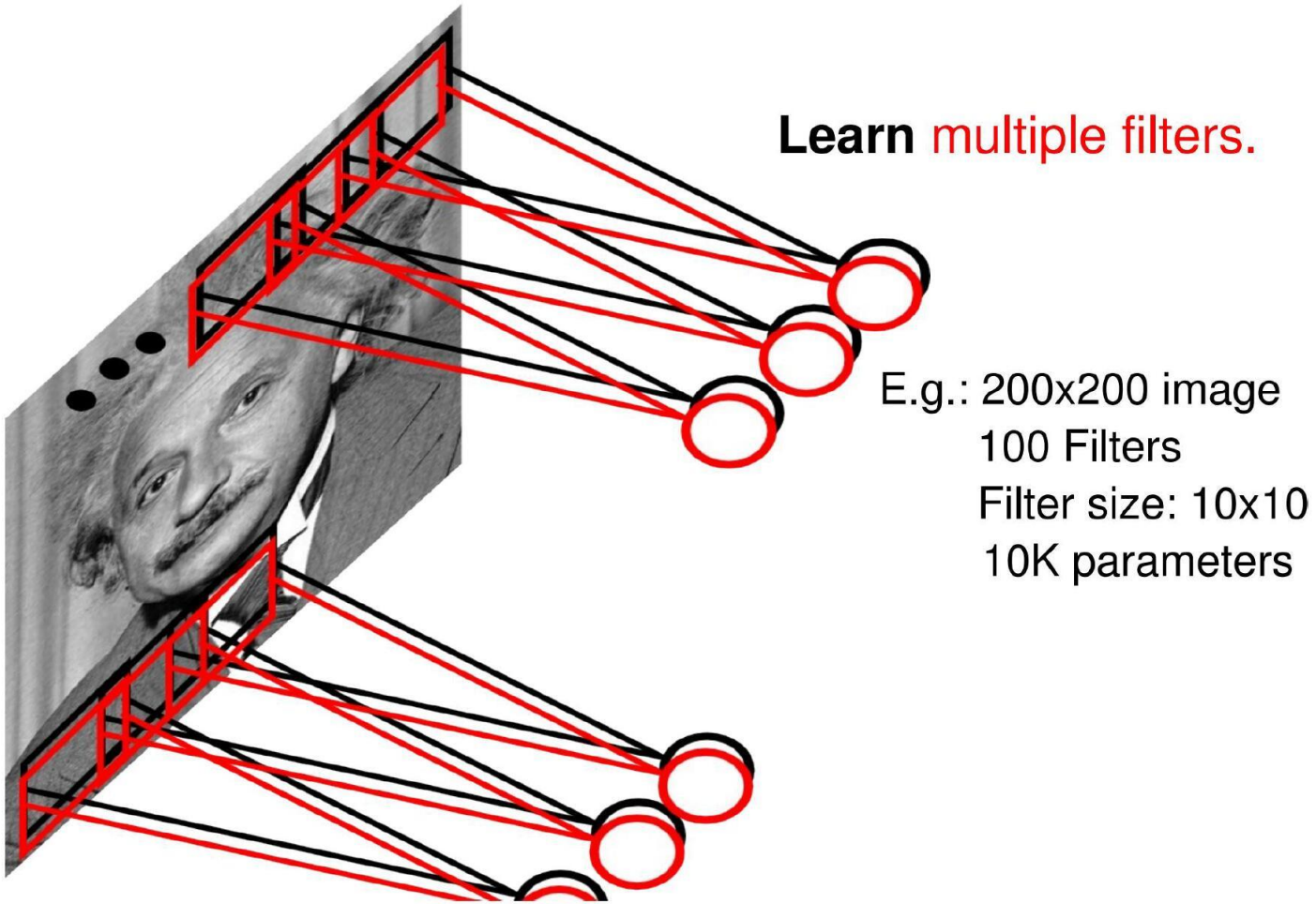
Convolutional Layer



Convolutional Layer



Convolutional Layer



Learn multiple filters.

E.g.: 200x200 image
100 Filters
Filter size: 10x10
10K parameters

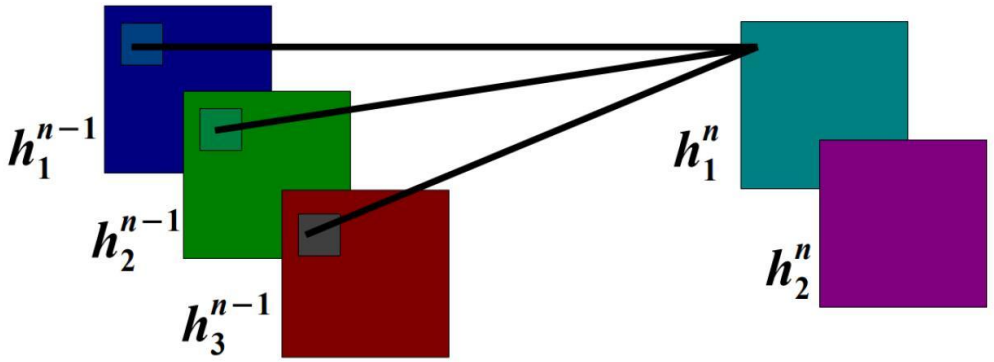
Convolutional Layer

$$h_j^n = max\left(0, \sum_{k=1}^{K} h_k^{n-1} * w_{kj}^n\right)$$

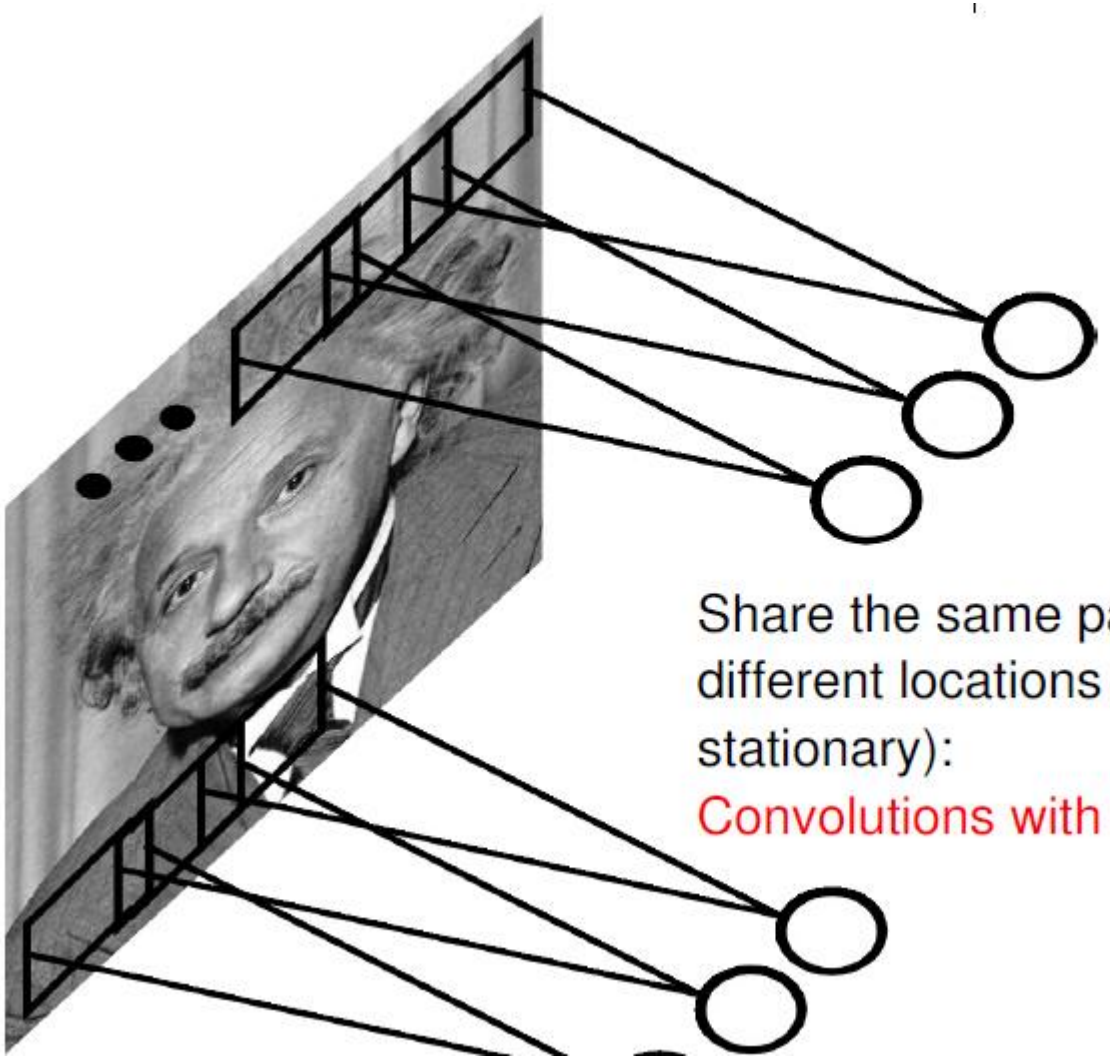
output feature map

input feature map

kernel



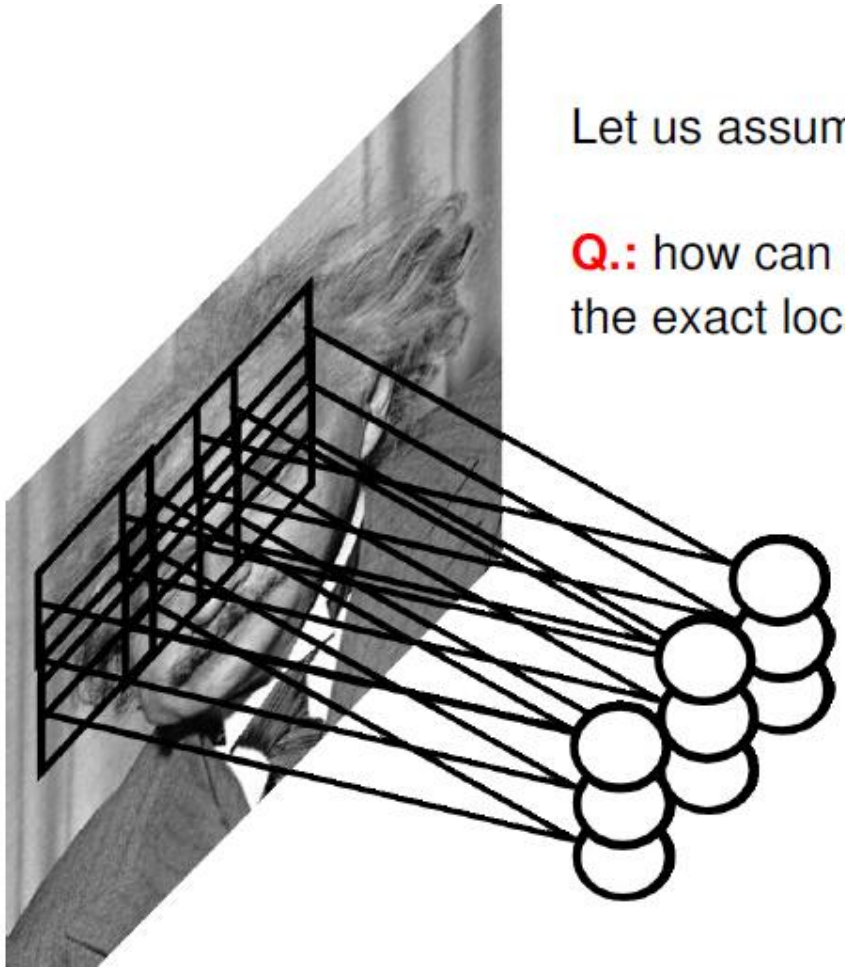
Convolutional Layer



Share the same parameters across different locations (assuming input is stationary):

Convolutions with learned kernels

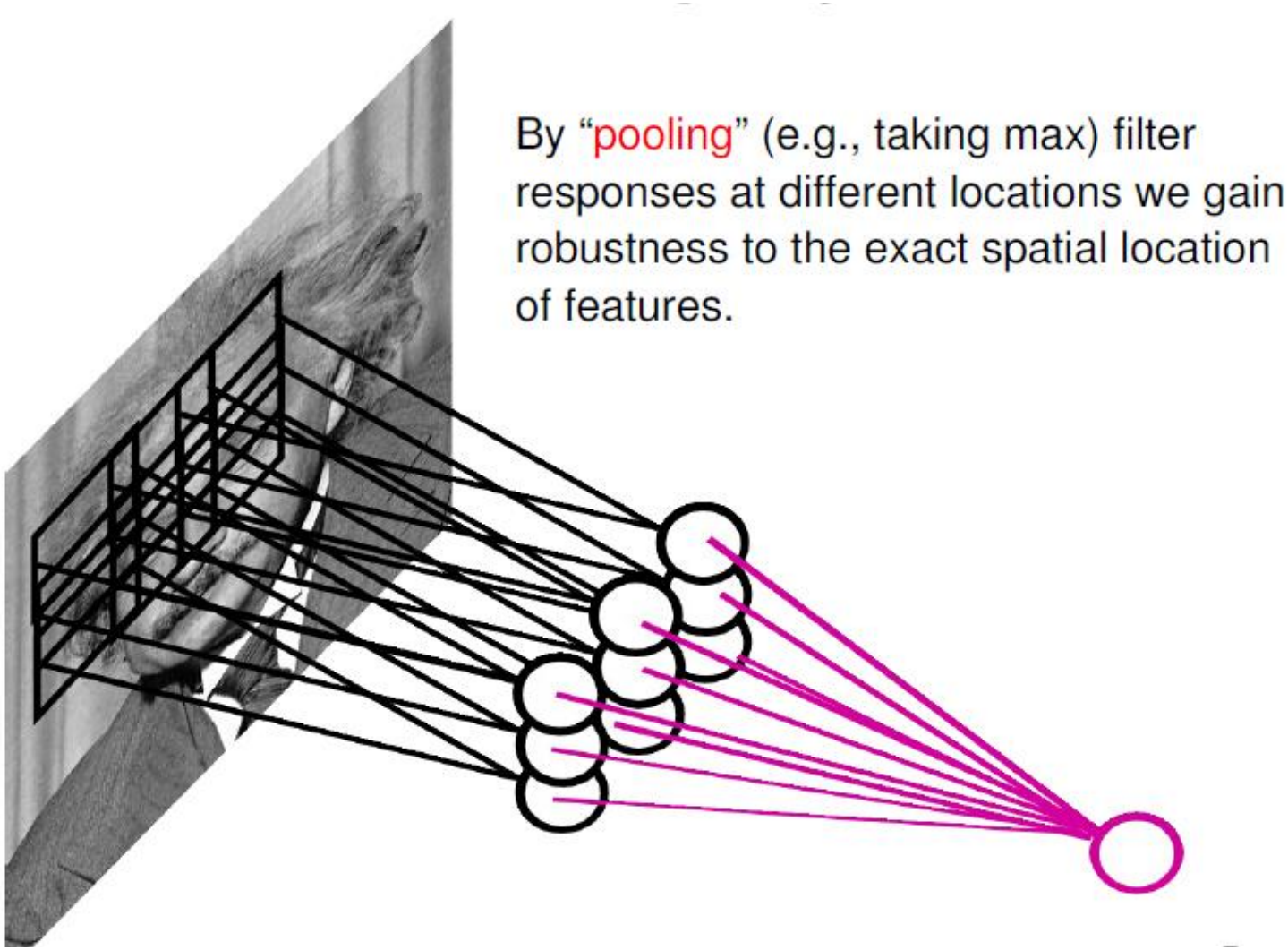
Convolutional Layer



Let us assume filter is an “eye” detector.

Q.: how can we make the detection robust to the exact location of the eye?

Convolutional Layer



By "pooling" (e.g., taking max) filter responses at different locations we gain robustness to the exact spatial location of features.

Pooling Layer

Max-pooling:

$$h_j^n(x, y) = max_{\bar{x} \in N(x),\, \bar{y} \in N(y)} h_j^{n-1}(\bar{x}, \bar{y})$$

Average-pooling:

$$h_j^n(x, y) = 1/K \sum_{\bar{x} \in N(x),\, \bar{y} \in N(y)} h_j^{n-1}(\bar{x}, \bar{y})$$

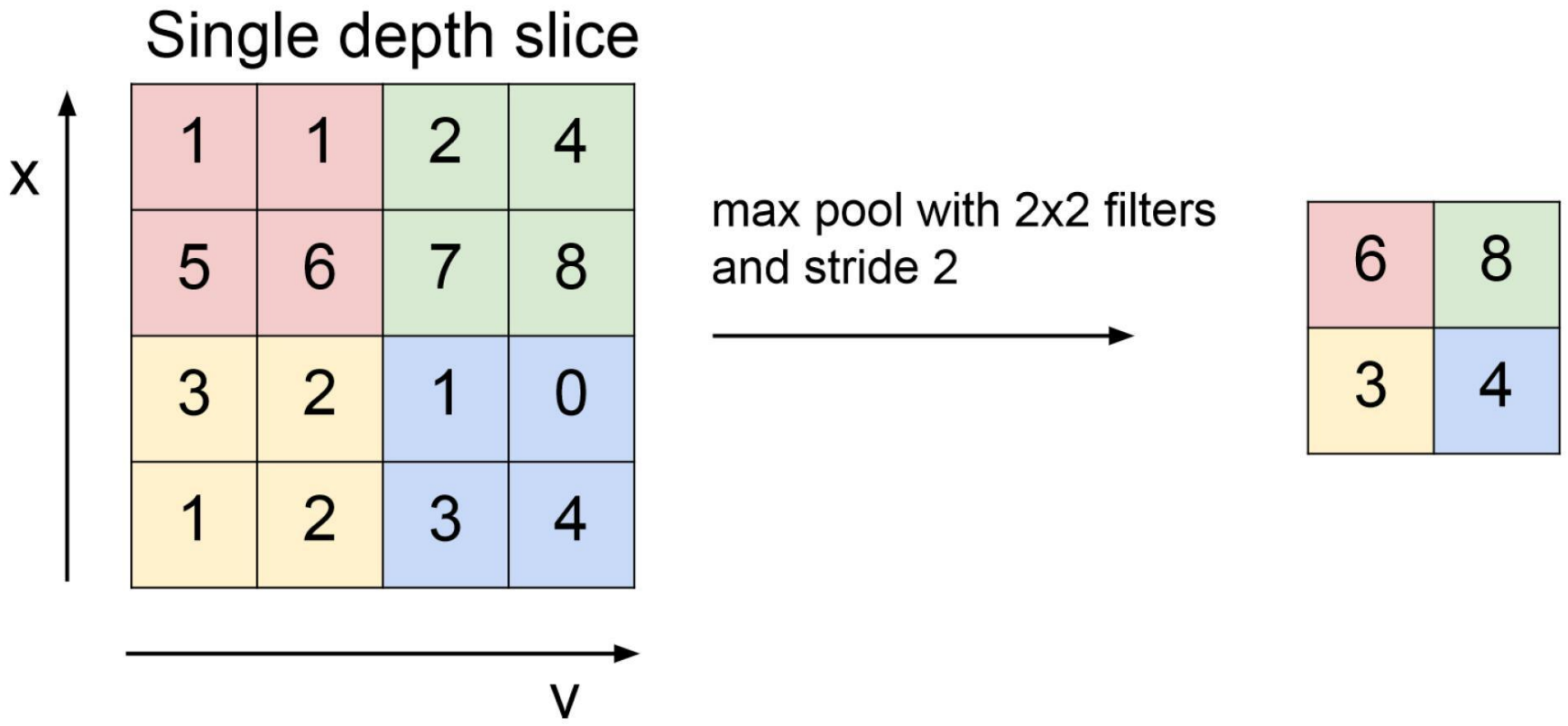
L2-pooling:

$$h_j^n(x, y) = \sqrt{\sum_{\bar{x} \in N(x),\, \bar{y} \in N(y)} h_j^{n-1}(\bar{x}, \bar{y})^2}$$

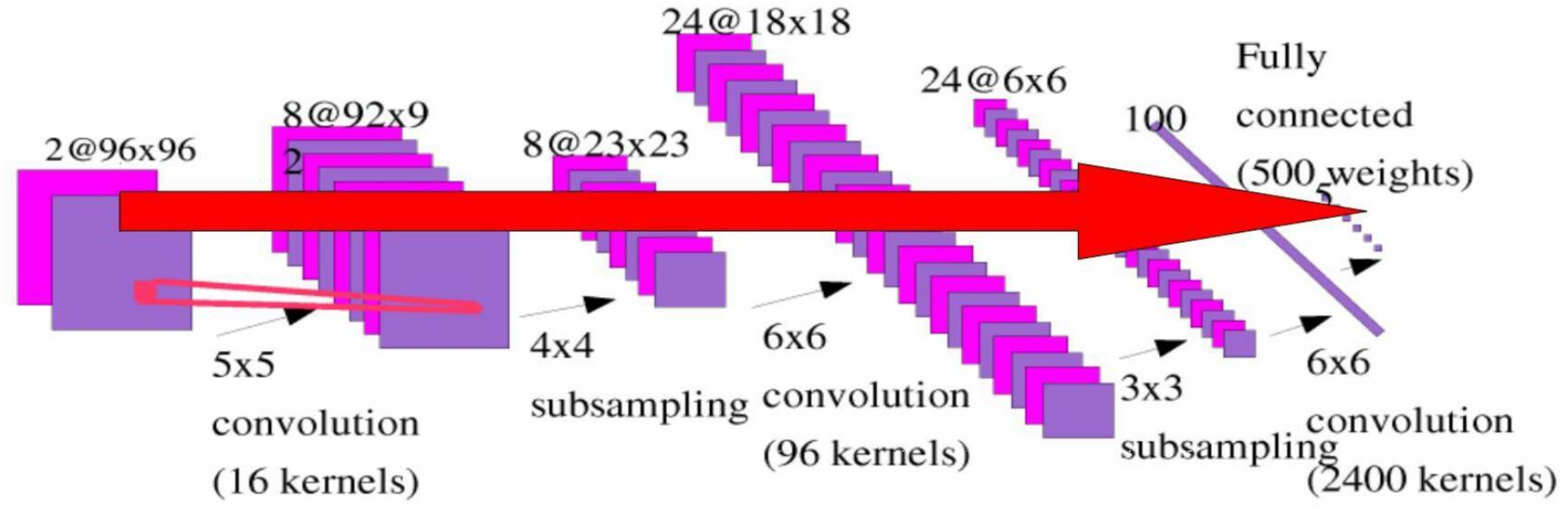
L2-pooling over features:

$$h_j^n(x, y) = \sqrt{\sum_{k \in N(j)} h_k^{n-1}(x, y)^2}$$

Pooling Layer

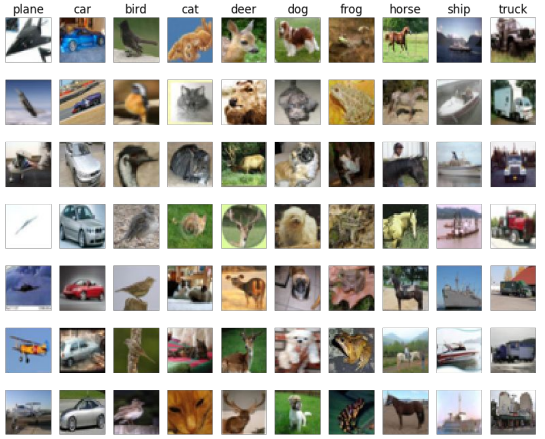


An Example of CNN



CNN: Image Classification

- CIFAR-10: https://www.cs.toronto.edu/~kriz/cifar.html



- Data download and import libararies

```
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

trainset = torchvision.datasets.CIFAR10(root='/home/CIFAR-10 Classifier Using CNN in PyTorch/da
                                        train=True,
                                        download=True,
                                        transform=transform)
trainloader = torch.utils.data.DataLoader(trainset,
                                          batch_size=4,
                                          shuffle=True)

testset = torchvision.datasets.CIFAR10(root='./data',
                                       train=False,
                                       download=True,
                                       transform=transform)
testloader = torch.utils.data.DataLoader(testset,
                                         batch_size=4,
                                         shuffle=False)

classes = ('plane', 'car', 'bird', 'cat', 'deer',
           'dog', 'frog', 'horse', 'ship', 'truck')
```

```
import torch
import torchvision
import torchvision.transforms as transforms
```

CNN: Image Classification

- Network architecture:

`Input > Conv (ReLU) > MaxPool > Conv (ReLU) > MaxPool > FC (ReLU) > FC (ReLU) > FC (SoftMax) > 10 outputs`

where:

`Conv` is a convolutional layer, `ReLU` is the activation function, `MaxPool` is a pooling layer, `FC` is a fully connected layer and `SoftMax` is the activation function of the output layer.

```
import torch.nn as nn
import torch.nn.functional as F

class Net(nn.Module):
    def __init__(self):
        super(Net, self).        ()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

net = Net()
```

CNN: Image Classification

- Loss function and optimizer:

```
import torch.optim as optim

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

- Train the network:

```
import os

model_directory_path = '/home/CIFAR-10 Classifier Using CNN in PyTorch/model/'
model_path = model_directory_path + 'cifar-10-cnn-model.pt'

if not os.path.exists(model_directory_path):
    os.makedirs(model_directory_path)

if os.path.isfile(model_path):
    # load trained model parameters from disk
    net.load_state_dict(torch.load(model_path))
    print('Loaded model parameters from disk.')
else:
    for epoch in range(2):  # loop over the dataset multiple times

        running_loss = 0.0
        for i, data in enumerate(trainloader, 0):
            # get the inputs
            inputs, labels = data

            # zero the parameter gradients
            optimizer.zero_grad()

            # forward + backward + optimize
            outputs = net(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            # print statistics
            running_loss += loss.item()
            if i % 2000 == 1999:    # print every 2000 mini-batches
                print('[%d, %5d] loss: %.3f' %
                      (epoch + 1, i + 1, running_loss / 2000))
                running_loss = 0.0
    print('Finished Training.')
    torch.save(net.state_dict(), model_path)
    print('Saved model parameters to disk.')
```

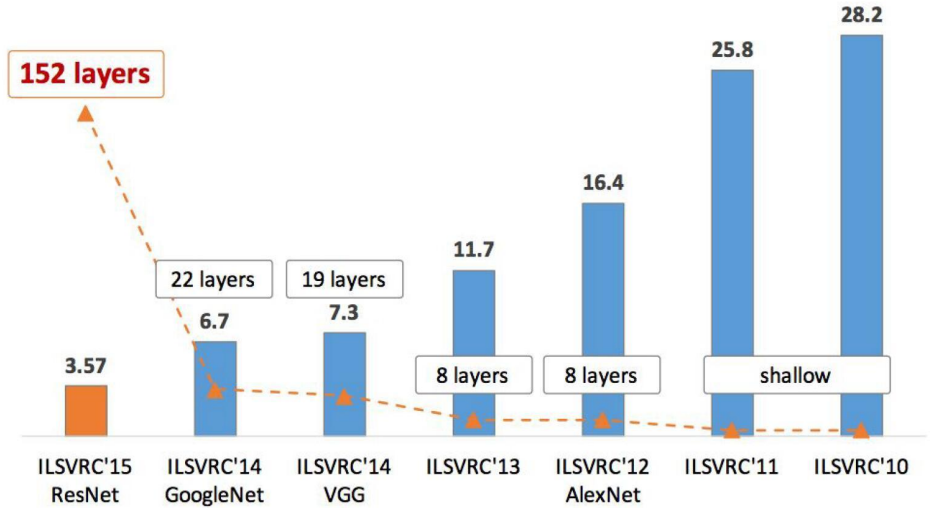
CNN: Image Classification

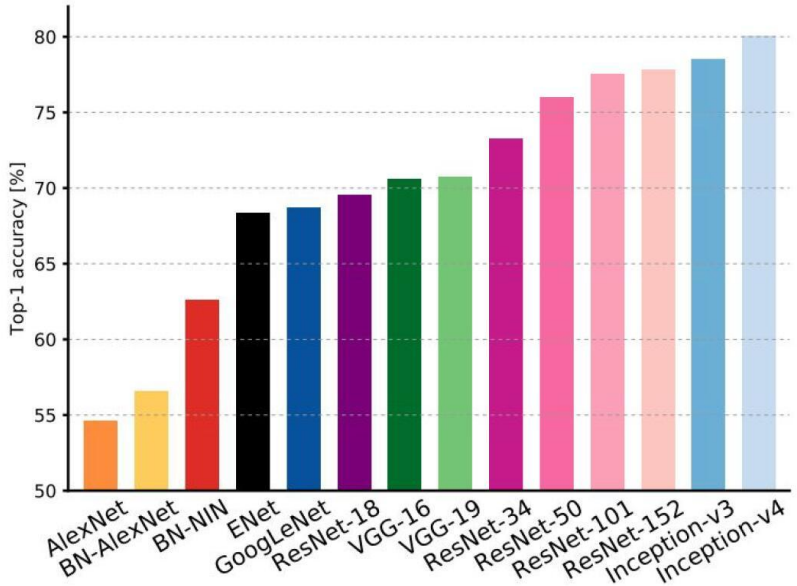
- Predict and test:

```
total_correct = 0
total_images = 0
confusion_matrix = np.zeros([10,10], int)
with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)
        total_images += labels.size(0)
        total_correct += (predicted == labels).sum().item()
        for i, l in enumerate(labels):
            confusion_matrix[l.item(), predicted[i].item()] += 1

model_accuracy = total_correct / total_images * 100
print('Model accuracy on {0} test images: {1:.2f}%'.format(total_images, model_accuracy))
```

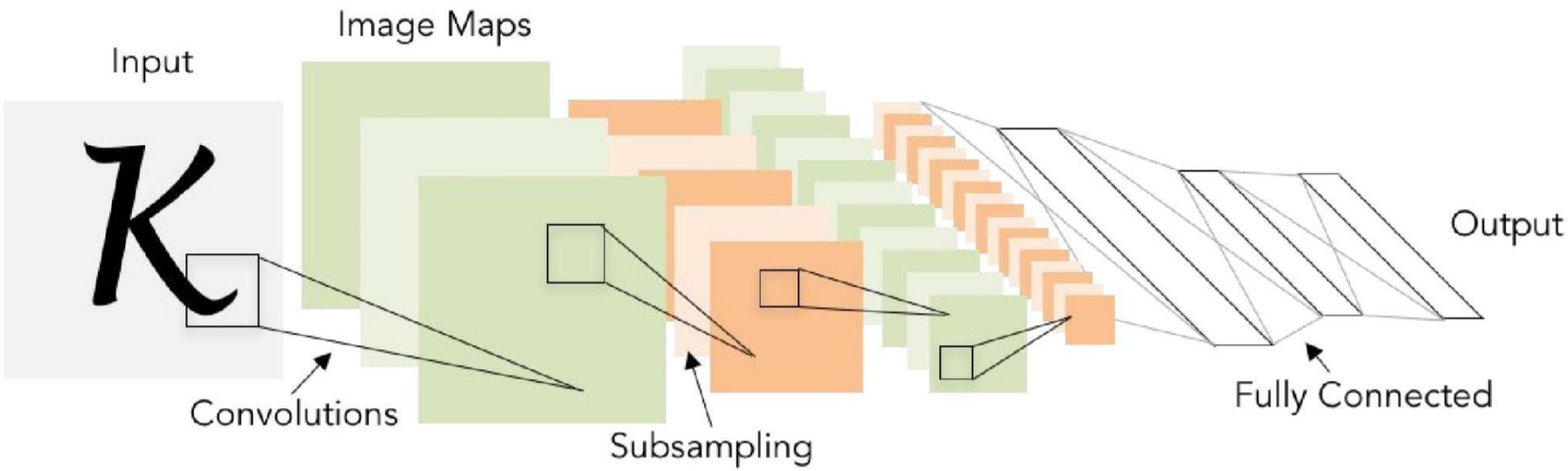
CNN Models History





Review: LeNet-5

[LeCun et al., 1998]

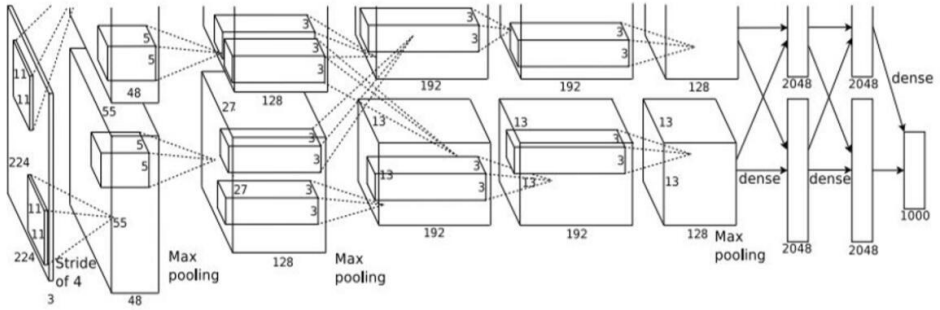


Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-FC-FC]

Review: AlextNet

Case Study: AlexNet

[Krizhevsky et al. 2012]



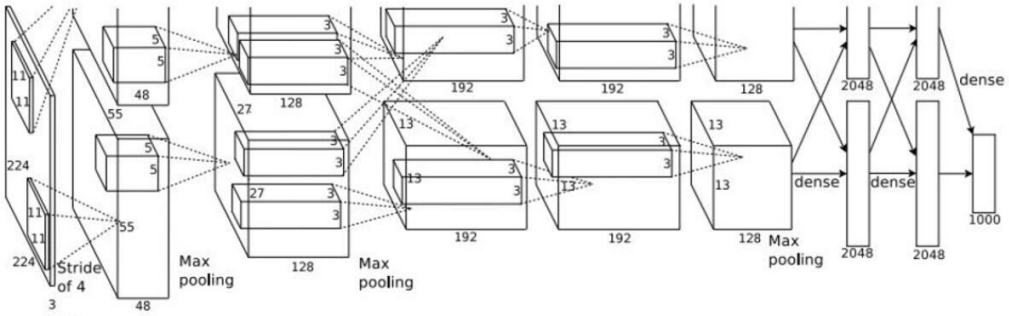
Architecture:

CONV1
MAX POOL1
NORM1
CONV2
MAX POOL2
NORM2
CONV3
CONV4
CONV5
Max POOL3
FC6
FC7
FC8

Review: AlextNet

Case Study: AlexNet

[Krizhevsky et al. 2012]

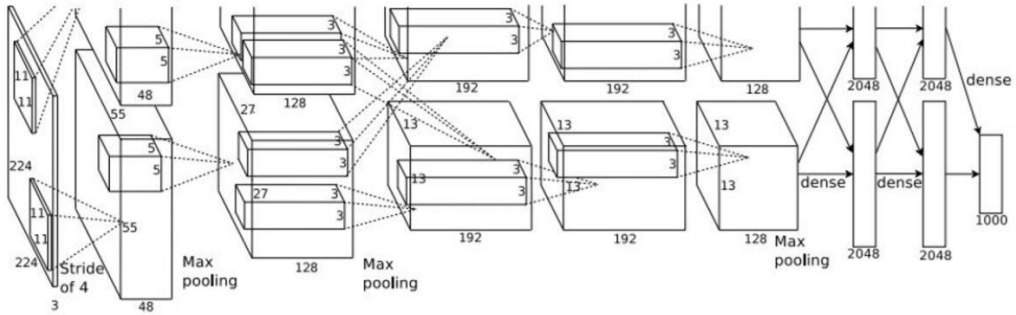


Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

Review: AlextNet

Case Study: AlexNet

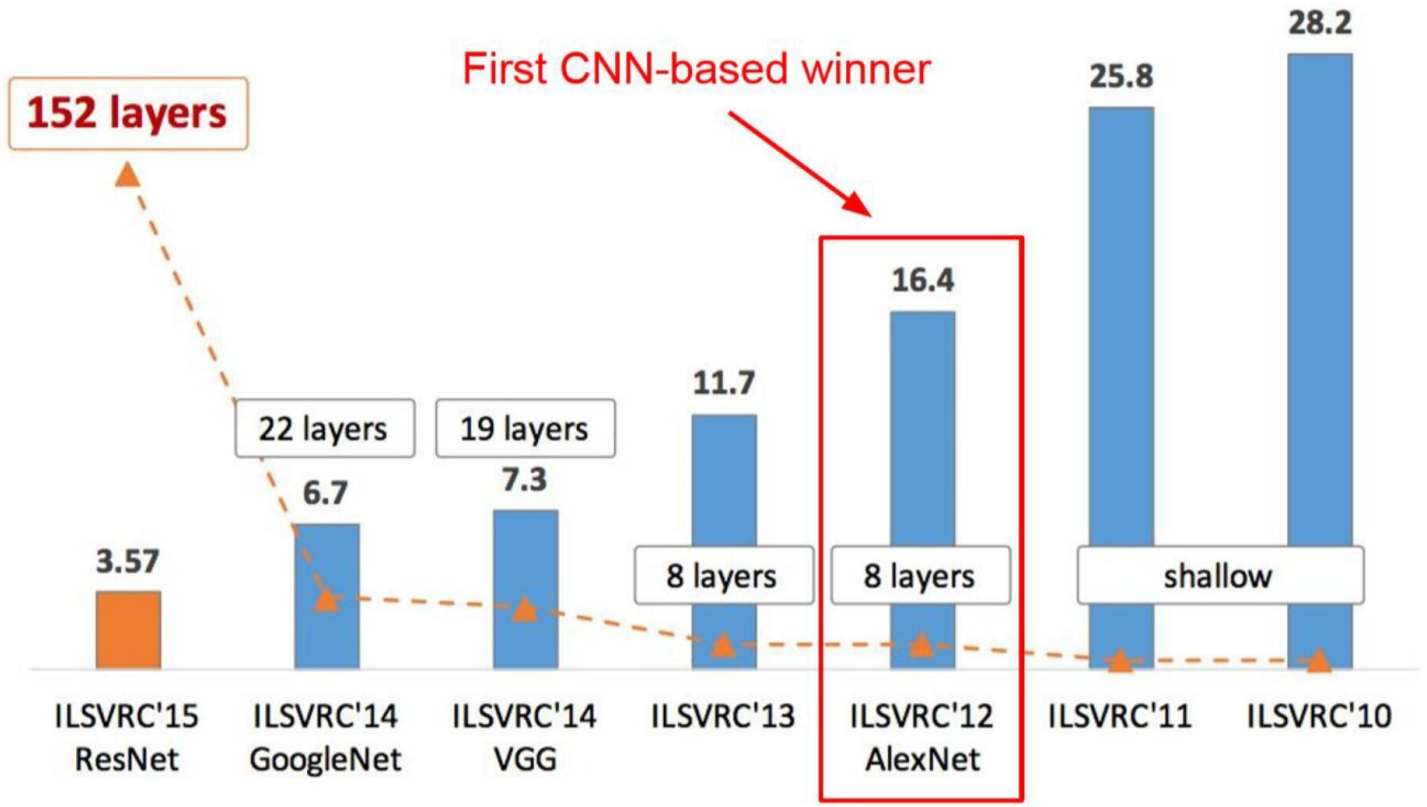
[Krizhevsky et al. 2012]



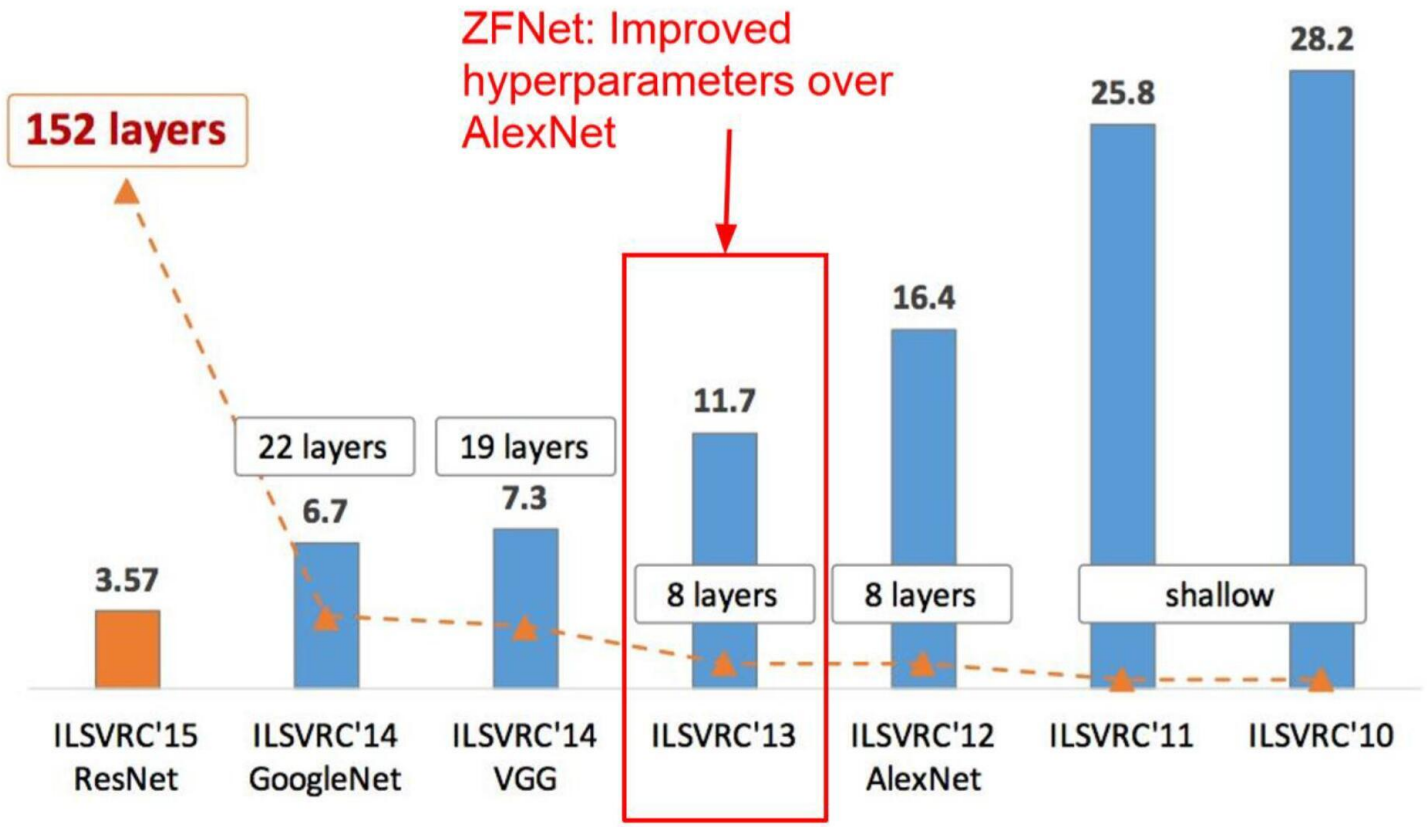
Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

Details/Retrospectives:
- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% -> 15.4%

Review: AlextNet



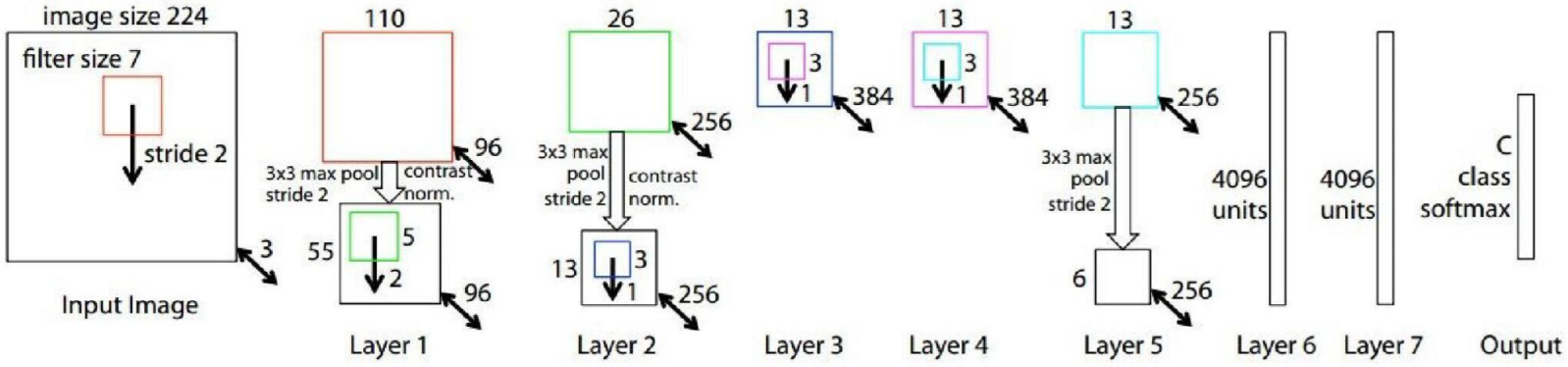
Review: AlextNet



Review: ZFNet

ZFNet

[Zeiler and Fergus, 2013]



TODO: remake figure

AlexNet but:
CONV1: change from (11x11 stride 4) to (7x7 stride 2)
CONV3,4,5: instead of 384, 384, 256 filters use 512, 1024, 512

Review: VGG

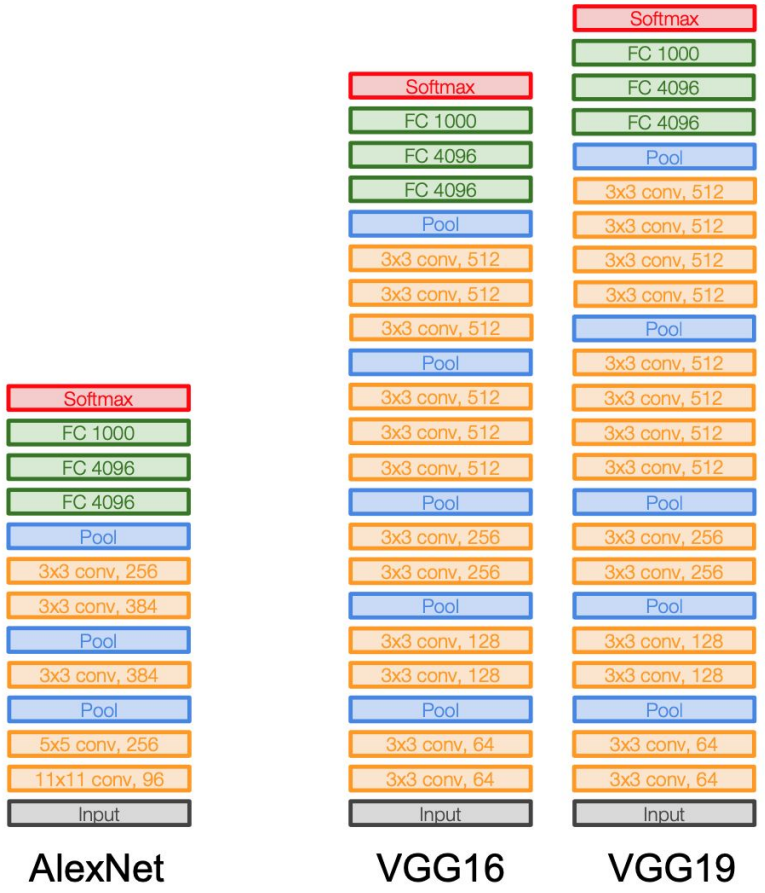
Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Q: Why use smaller filters? (3x3 conv)

Stack of three 3x3 conv (stride 1) layers has same **effective receptive field** as one 7x7 conv layer

Q: What is the effective receptive field of three 3x3 conv (stride 1) layers?



Review: VGG

Case Study: VGGNet

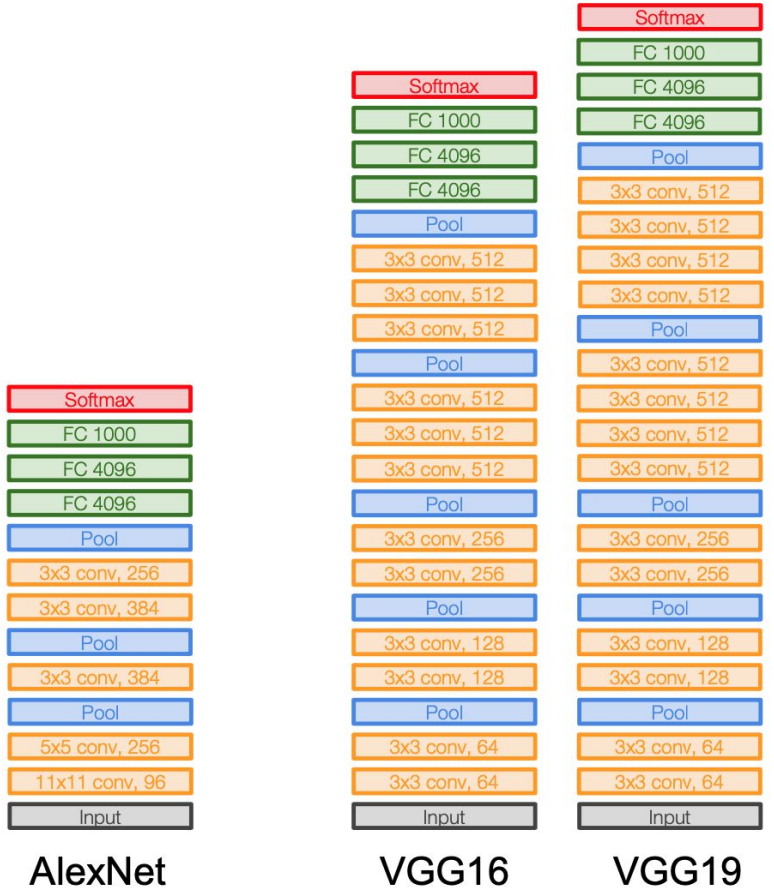
[Simonyan and Zisserman, 2014]

Small filters, Deeper networks

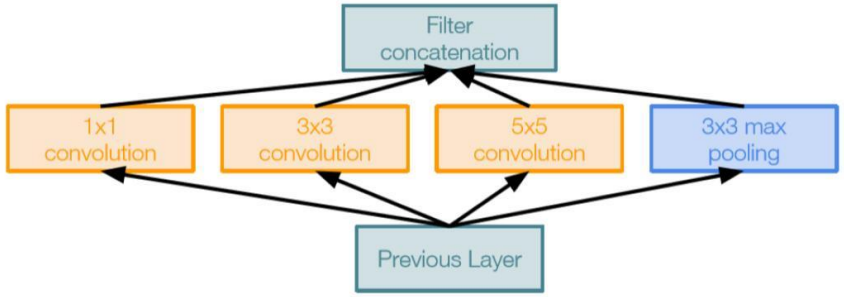
8 layers (AlexNet)
-> 16 - 19 layers (VGG16Net)

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

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



Review: GoogLeNet



Naive Inception module

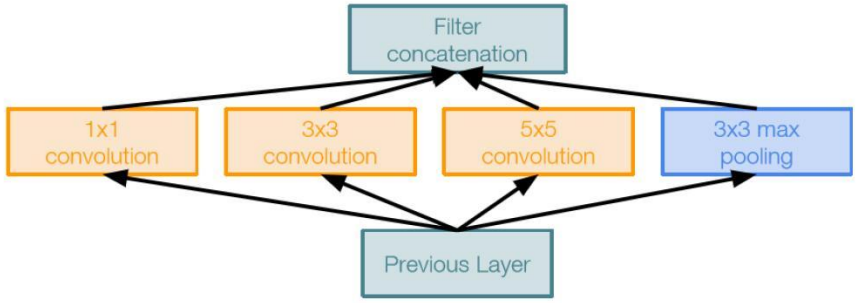
Apply parallel filter operations on the input from previous layer:

- Multiple receptive field sizes for convolution (1x1, 3x3, 5x5)
- Pooling operation (3x3)

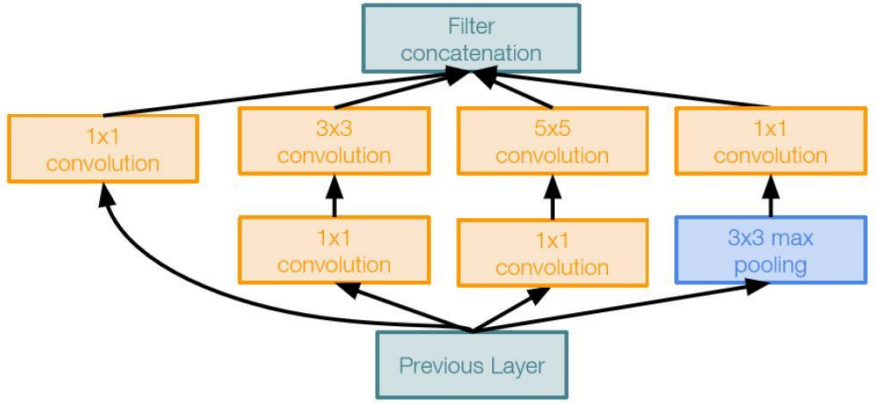
Concatenate all filter outputs together depth-wise

Review: GoogLeNet

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

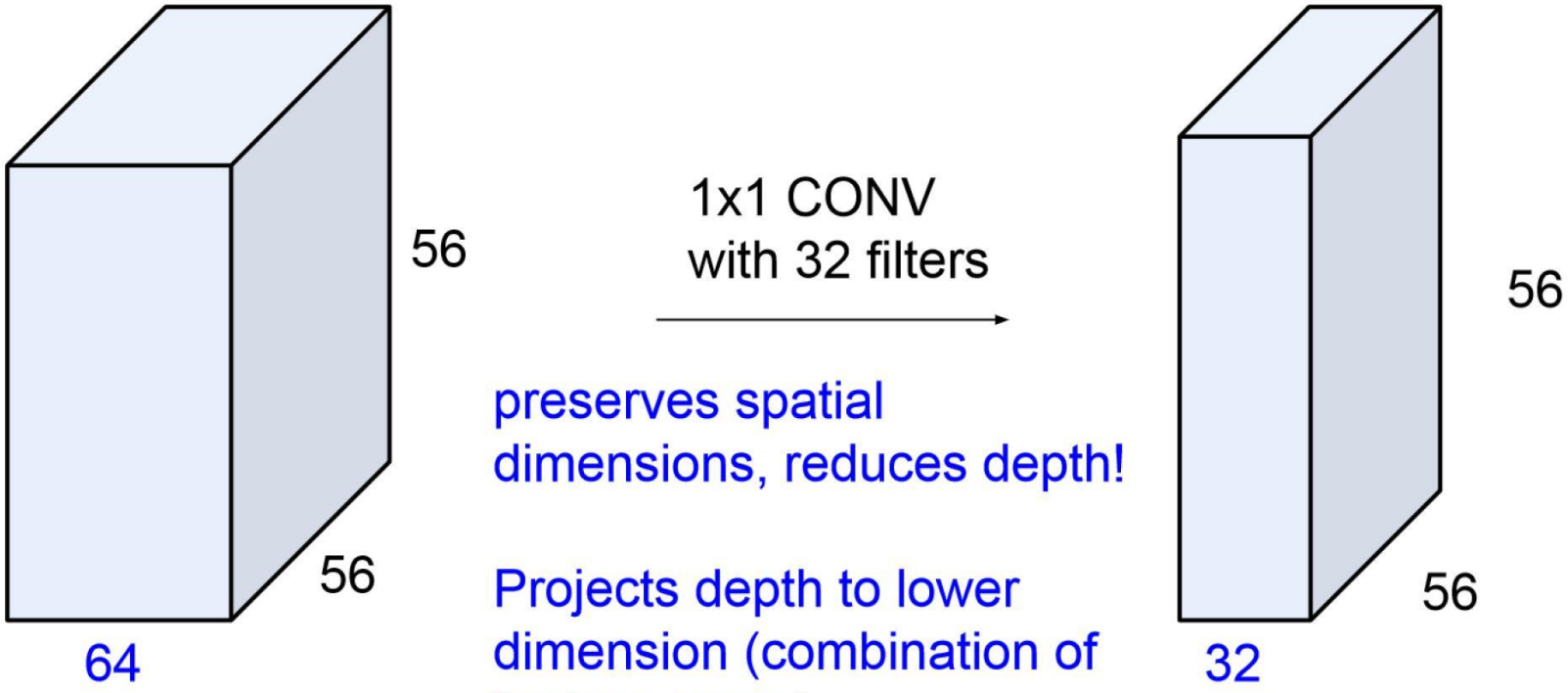


Naive Inception module



Inception module with dimension reduction

Review: GoogLeNet



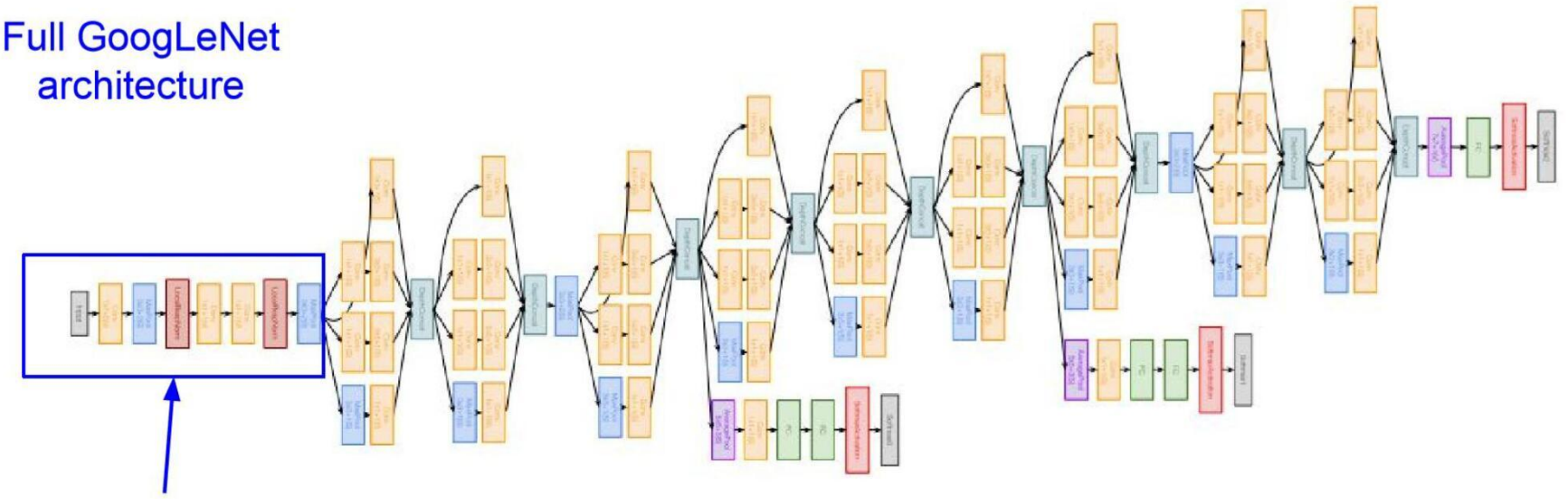
preserves spatial dimensions, reduces depth!

Projects depth to lower dimension (combination of feature maps)

Review: GoogLeNet

[Szegedy et al., 2014]

Full GoogLeNet architecture

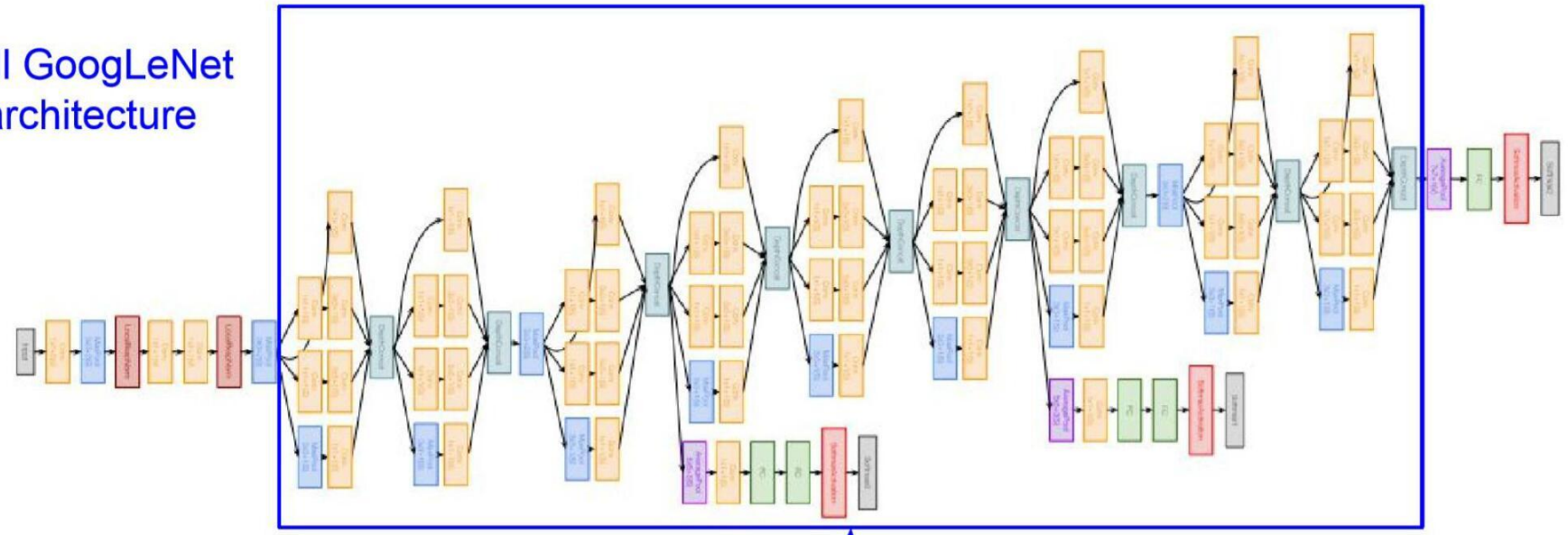


Stem Network: Conv-Pool- 2x Conv-Pool

Review: GoogLeNet

[Szegedy et al., 2014]

Full GoogLeNet architecture

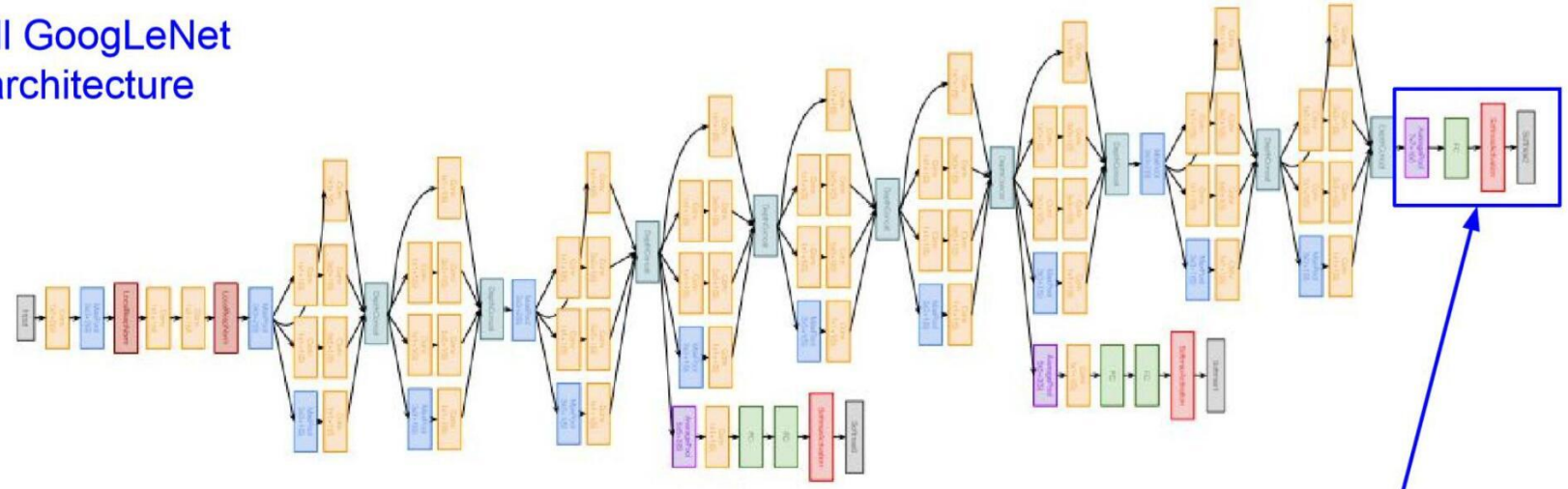


Stacked Inception Modules

Review: GoogLeNet

[Szegedy et al., 2014]

Full GoogLeNet architecture



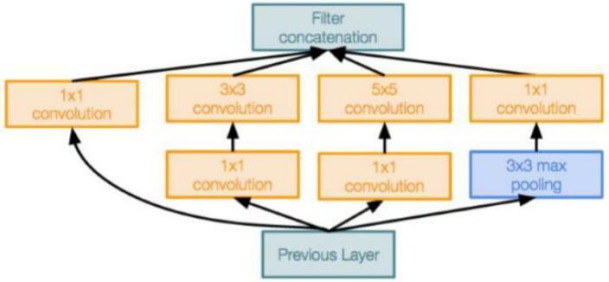
Classifier output
(removed expensive FC layers!)

Review: GoogLeNet

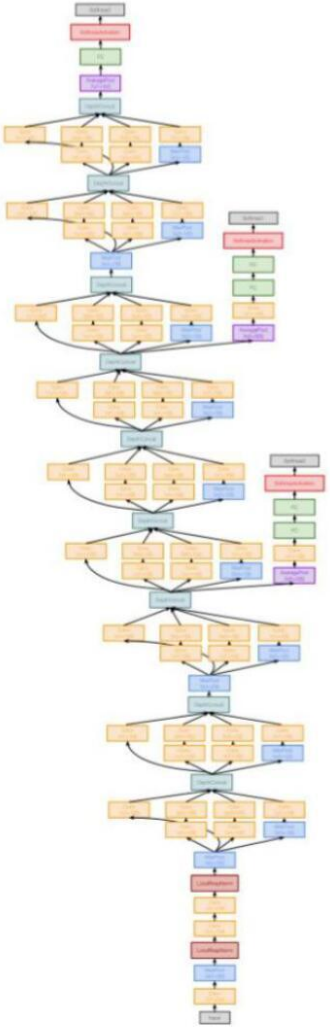
[Szegedy et al., 2014]

Deeper networks, with computational efficiency

- 22 layers
- Efficient “Inception” module
- No FC layers
- 12x less params than AlexNet
- ILSVRC’14 classification winner (6.7% top 5 error)



Inception module



Review: ResNet

What happens when we continue stacking deeper layers on a “plain” convolutional neural network?

Q: What’s strange about these training and test curves?
[Hint: look at the order of the curves]

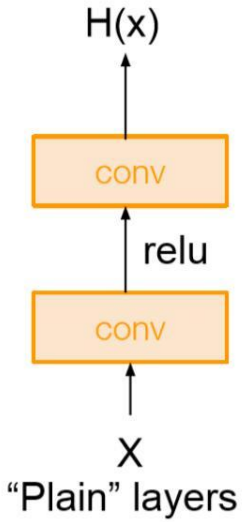
Review: ResNet

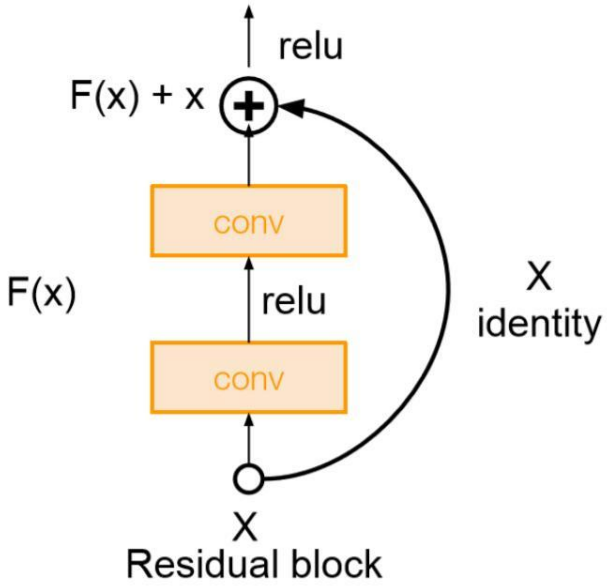
What happens when we continue stacking deeper layers on a “plain” convolutional neural network?

56-layer model performs worse on both training and test error
-> The deeper model performs worse, but it’s not caused by overfitting!

Review: ResNet

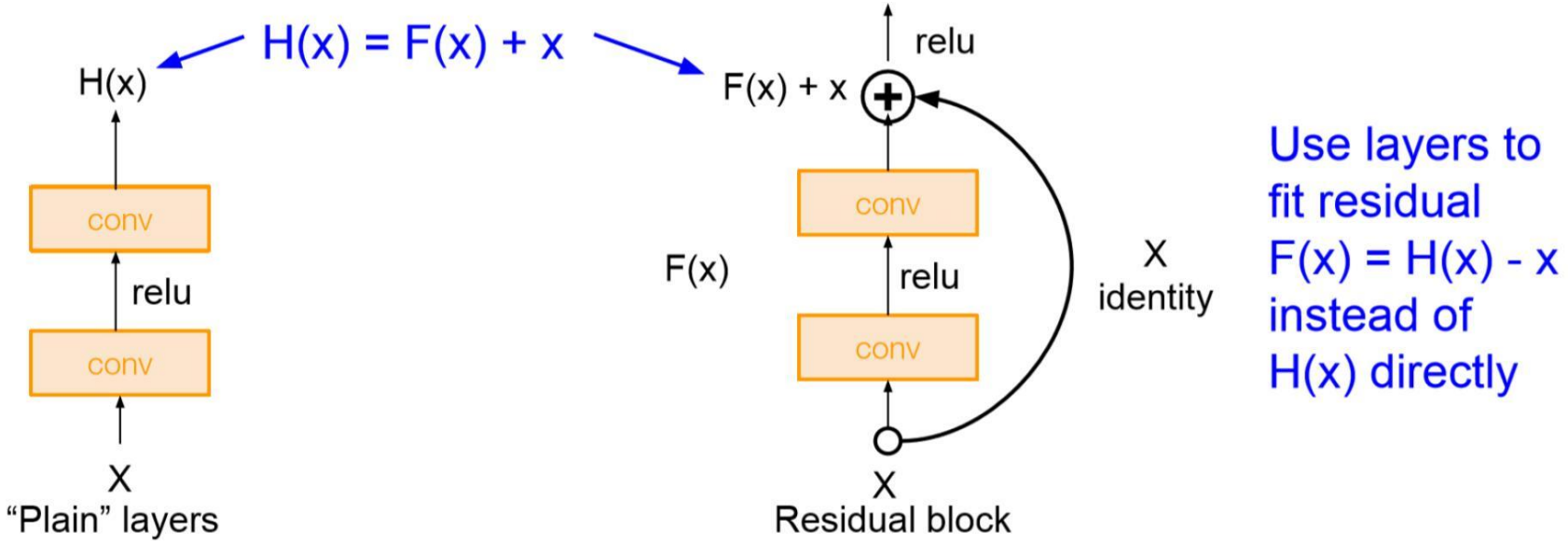
Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping





Review: ResNet

Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping

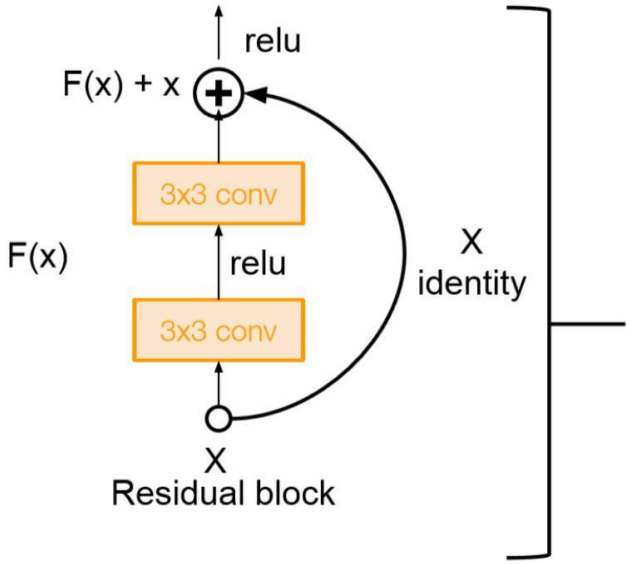


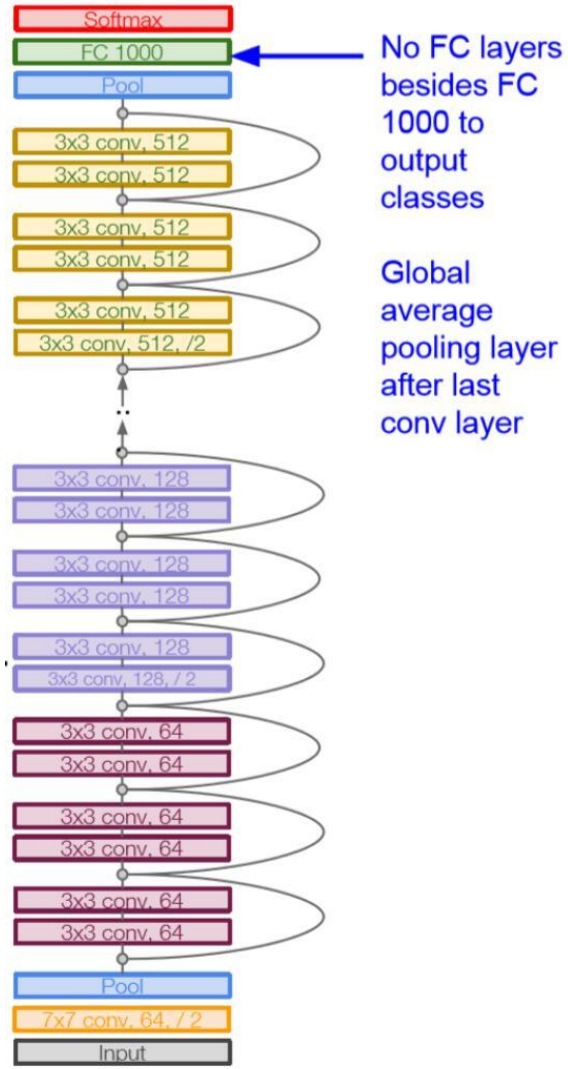
Review: ResNet

[He et al., 2015]

Full ResNet architecture:

- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)
- Additional conv layer at the beginning
- No FC layers at the end (only FC 1000 to output classes)

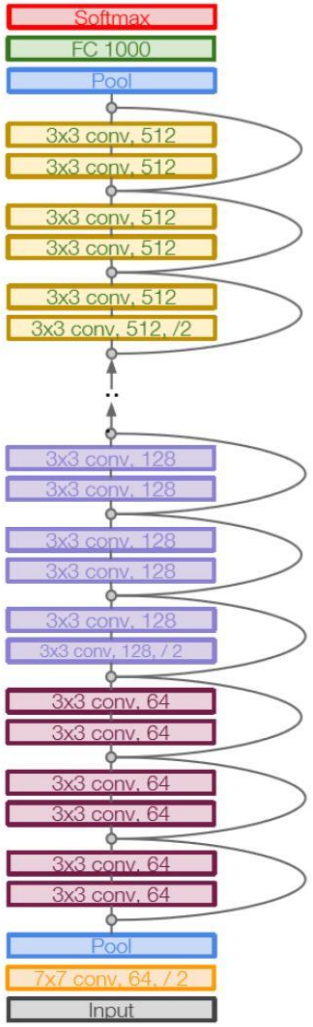




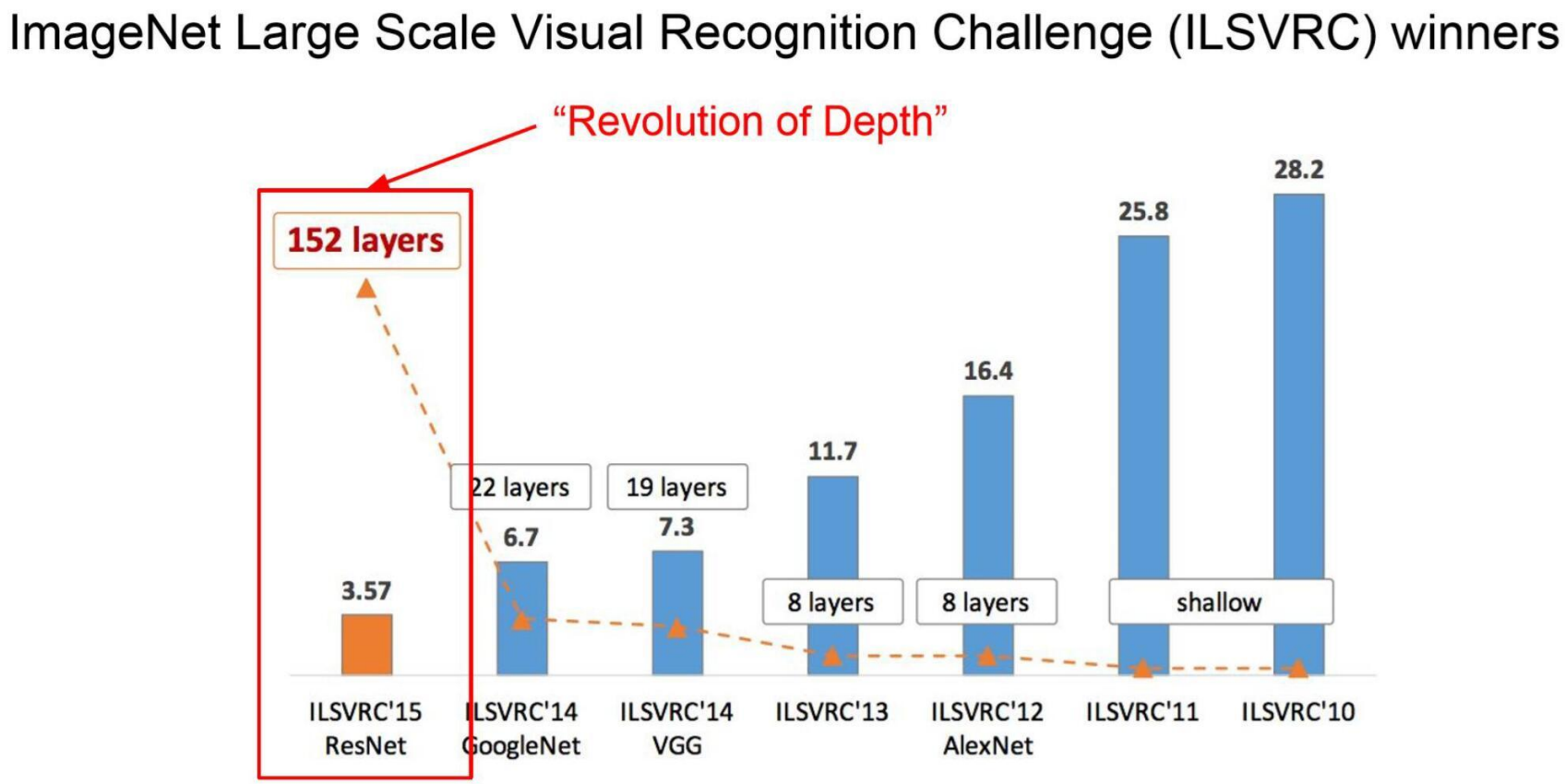
Review: ResNet

[He et al., 2015]

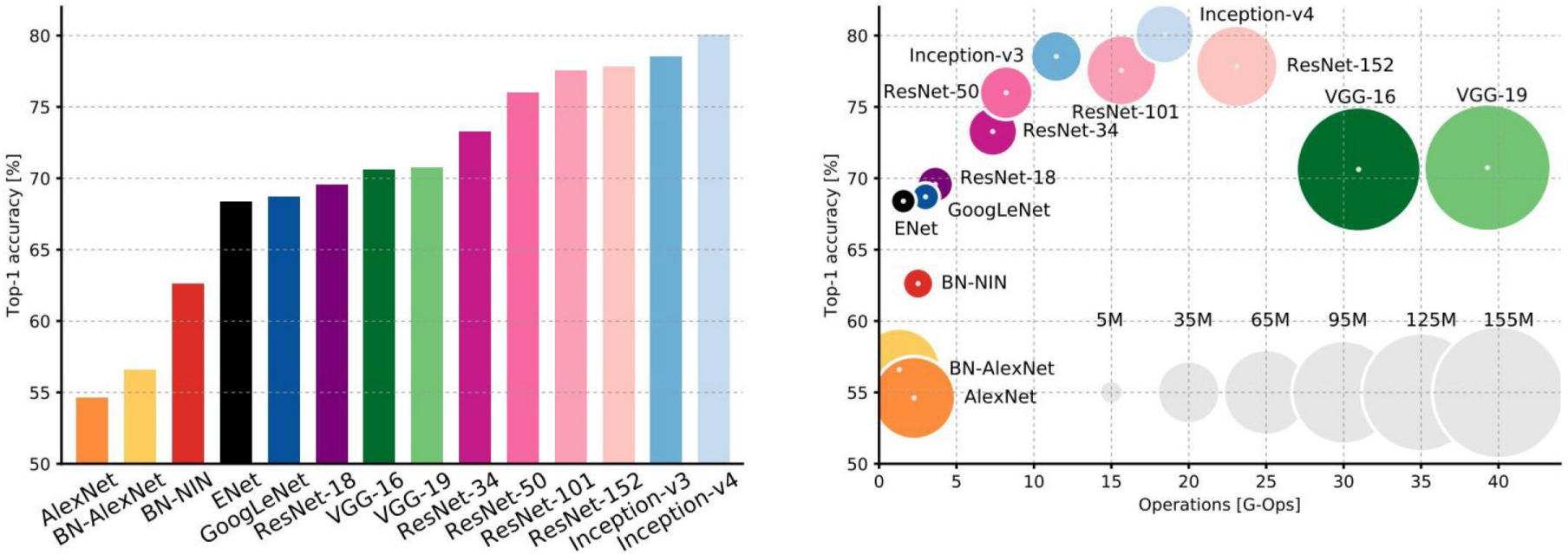
Total depths of 34, 50, 101, or
152 layers for ImageNet



Review: ResNet



Computational Comparisons

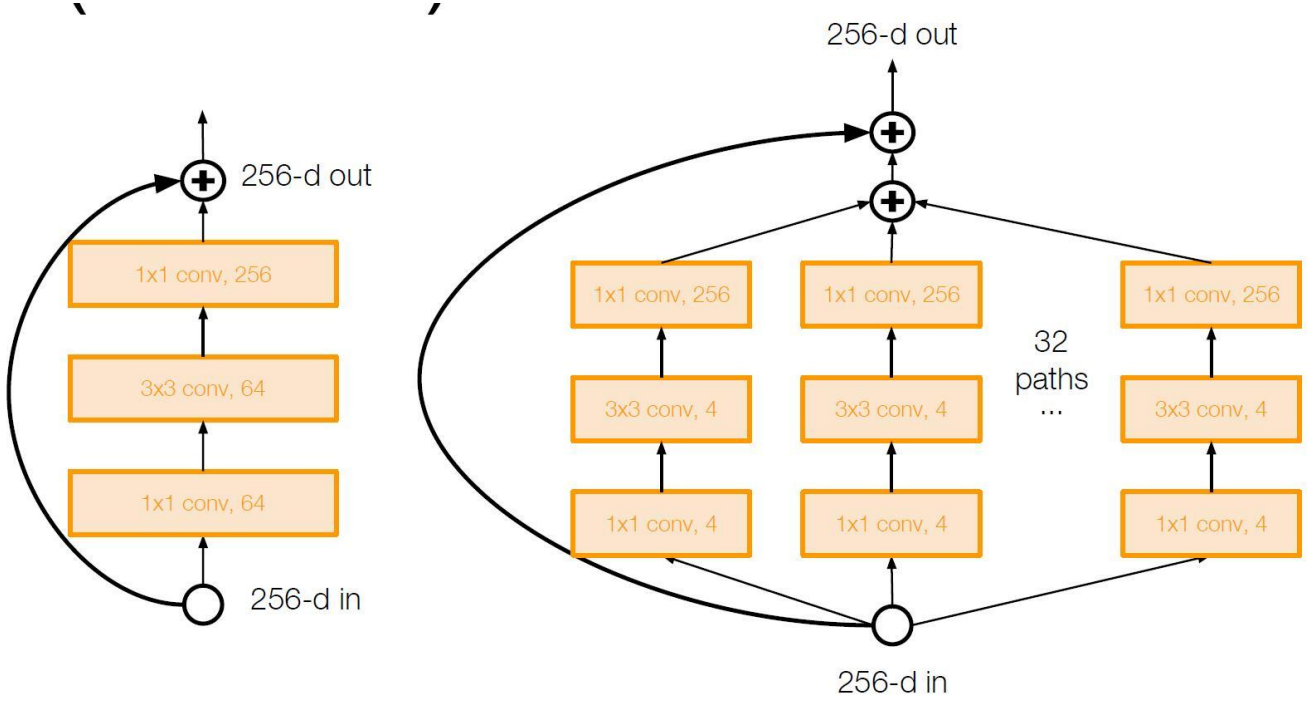


An Analysis of Deep Neural Network Models for Practical Applications, 2017.

ResNeXt

[Xie et al. 2016]

- Also from creators of ResNet
- Increases width of residual block through multiple parallel pathways (“cardinality”)
- Parallel pathways similar in spirit to Inception module



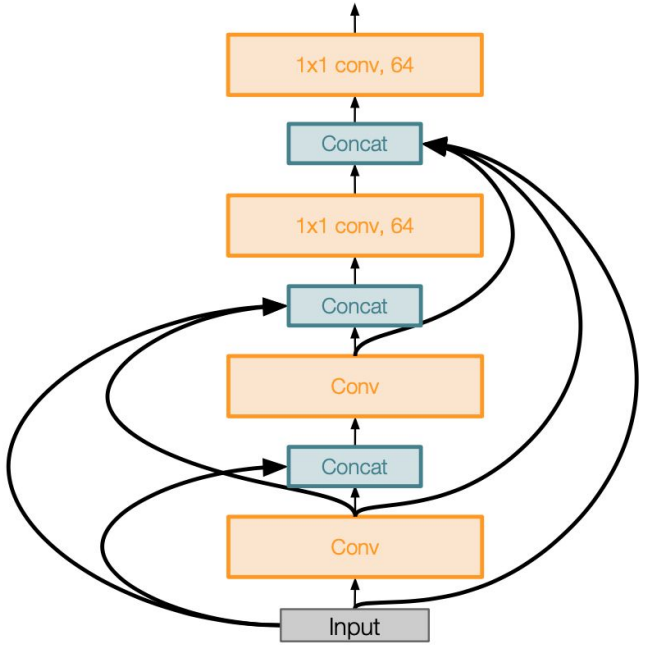
DenseNet

Beyond ResNets...

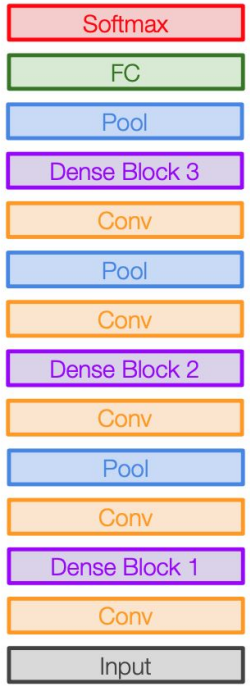
Densely Connected Convolutional Networks

[Huang et al. 2017]

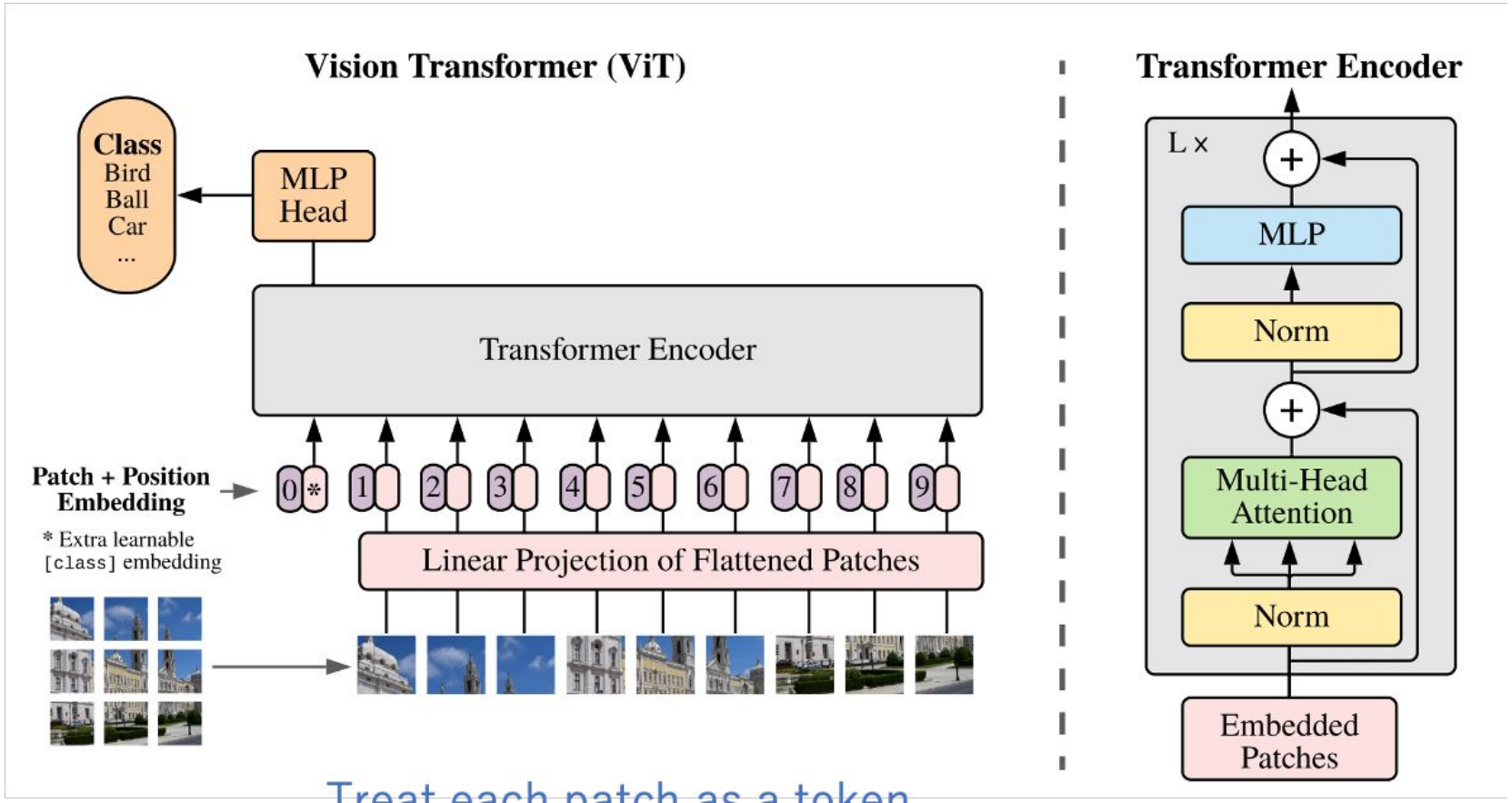
- Dense blocks where each layer is connected to every other layer in feedforward fashion
- Alleviates vanishing gradient, strengthens feature propagation, encourages feature reuse



Dense Block

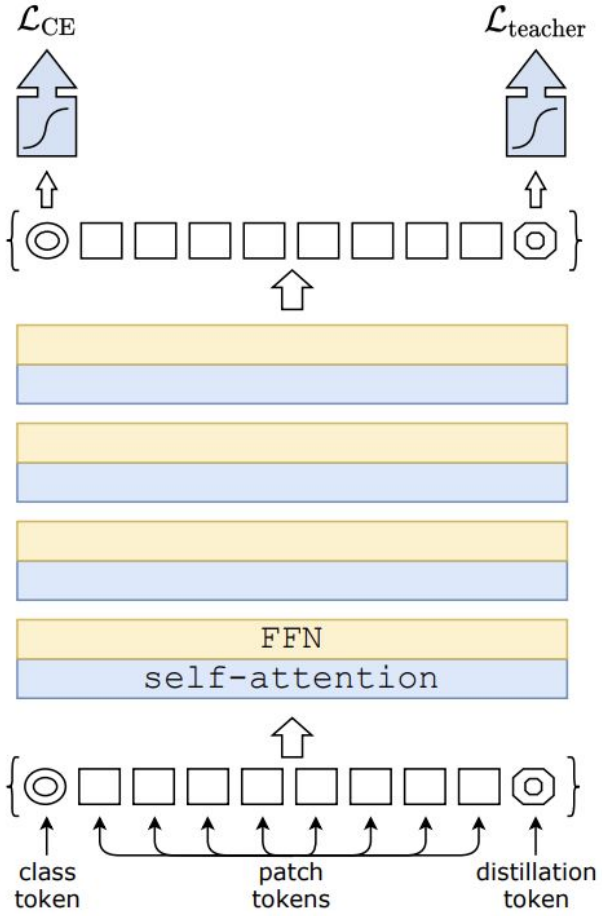


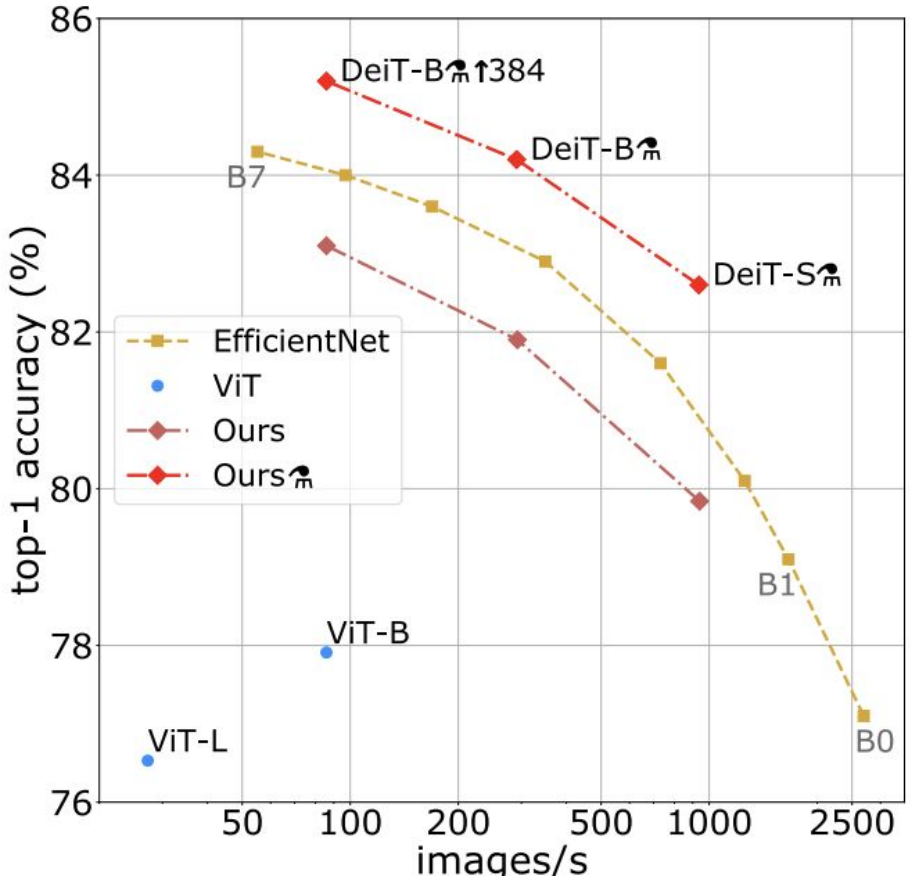
Vision Transformer



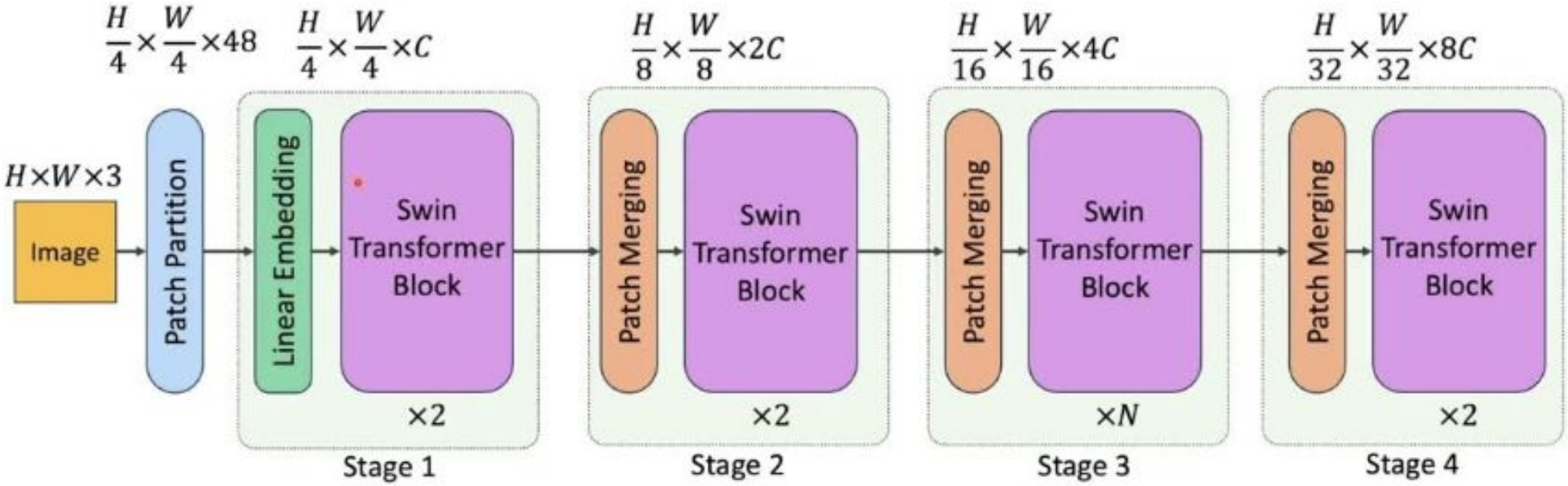
Treat each patch as a token (like a word) in NLP

DeiT: Data-efficient Image Transformers





Swin Transformer



Patch size: 4×4

- Using relative position embedding
- Each Swin transformer block contains a pair of alternative regular window and shifted window sub-blocks
- Down-sampling starts from stage 2

Text-to-Image/Video Generation

Video samples generated with ModelScope.

Text-to-Video Generation

Sora 2

Generation Models

Given training data, generate new samples from same distribution

Training data ~ $p_{data}(x)$

Generated samples ~ $p_{model}(x)$

Want to learn $p_{model}(x)$ similar to $p_{data}(x)$

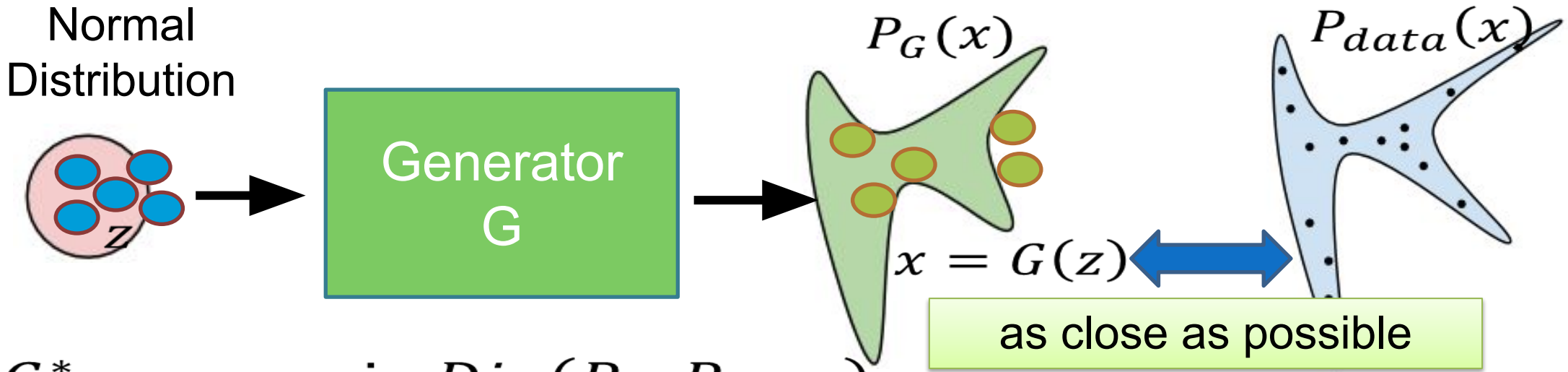
- A core problem in unsupervised learning
- Different flavors:

Explicit density estimation: explicitly define and solve for P_{model}

Implicit density estimation: learn model that can sample from P_{model}

Generative Adversarial Networks - GANs

- A generator G is a network. The network defines a probability distribution P_G



$$G^* = arg \min_G Div(P_G, P_{data})$$

Divergence between distributions P_G and P_{data}

How to compute the divergence?

Discriminator

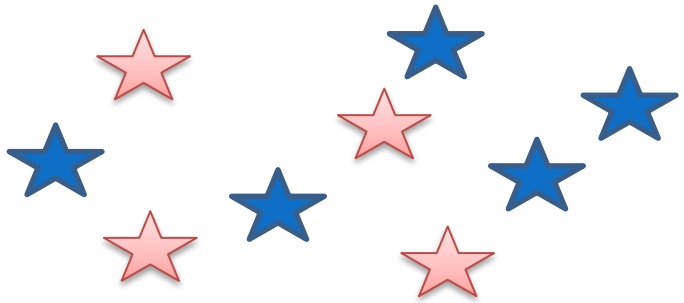
$$G^* = arg \min_G Div(P_G, P_{data})$$

★ : data sampled from P_{data}

☆ : data sampled from P_G

Training:

$$D^* = arg \max_D V(D, G)$$



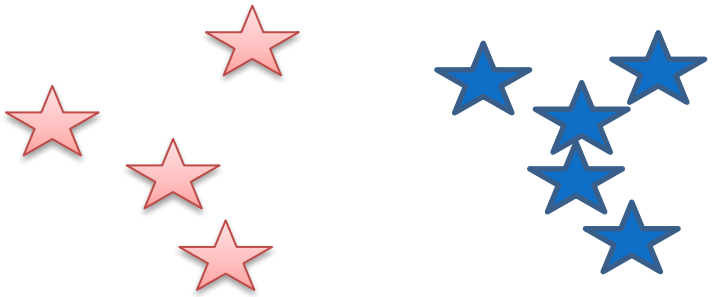
small divergence

train

Discriminator

hard to discriminate

(cannot make objective large)



large divergence

train

Discriminator

easy to discriminate

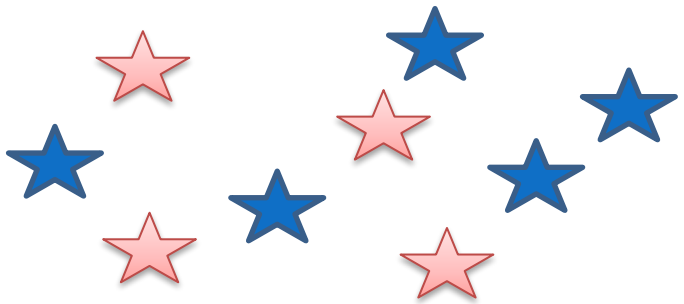
Discriminator

$$G^* = arg \min_G Div(P_G, P_{data})$$

: data sampled from P_{data}

: data sampled from P_G

Using the example objective function is exactly the same as training a binary classifier.



train

Discriminator

Sigmoid Output

Example Objective Function for D

$$V(G,D) = E_{x \sim P_{data}}[logD(x)] + E_{x \sim P_G}[log(1 - D(x))]$$

(G is fixed)

Training: $D^* = arg \max_D V(D,G)$

The maximum objective value is related to JS divergence.

Algorithm

$$G^* = arg \min_G \max_D V(G,D)$$

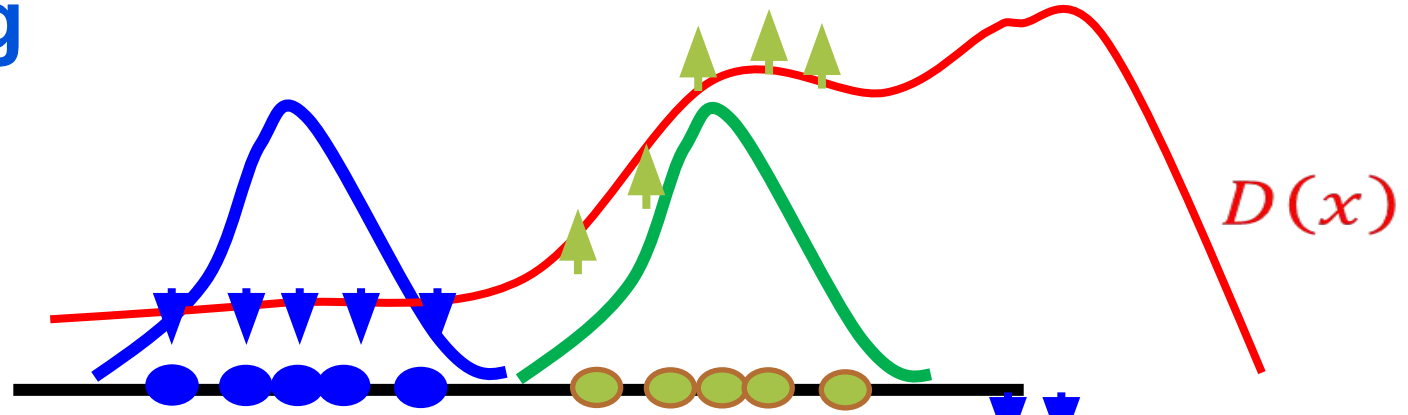
$$D^* = arg \max_D V(D,G)$$

The maximum objective value is related to JS divergence.

Step 1: Fix generator G, and update discriminator D

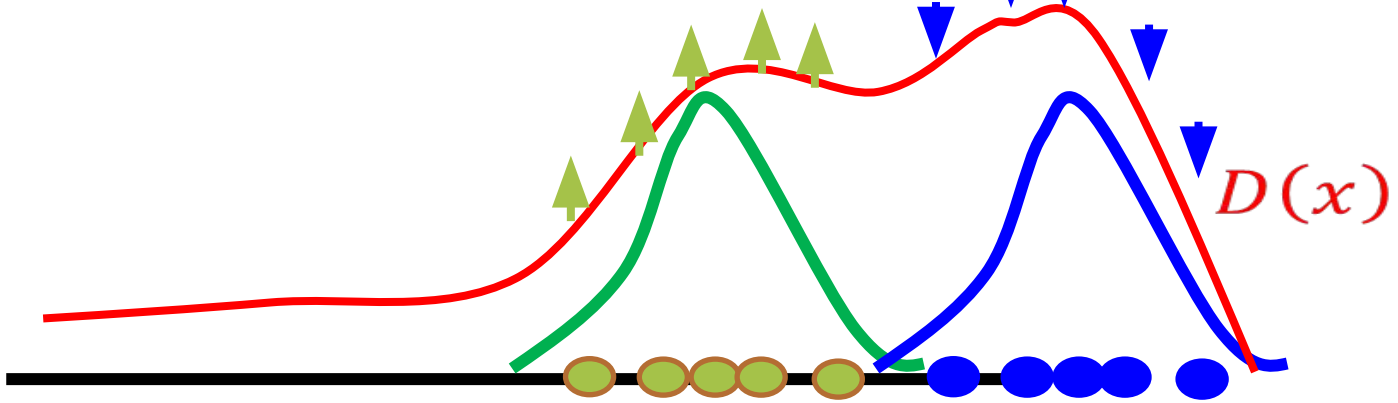
Step 2: Fix discriminator D, and update generator G

Discriminator - Training

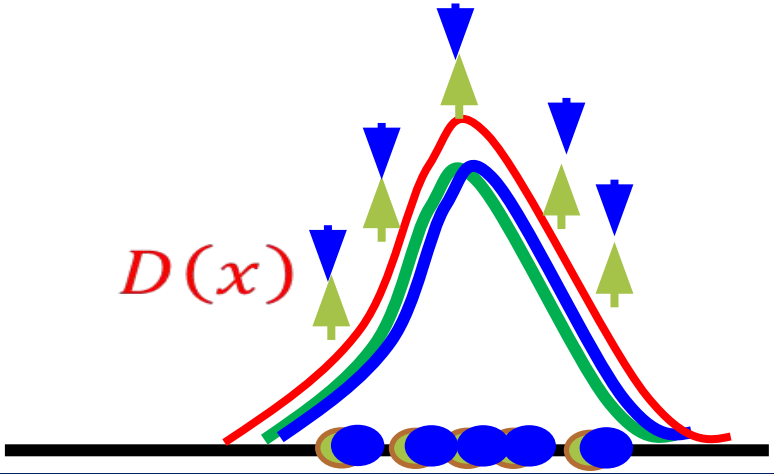


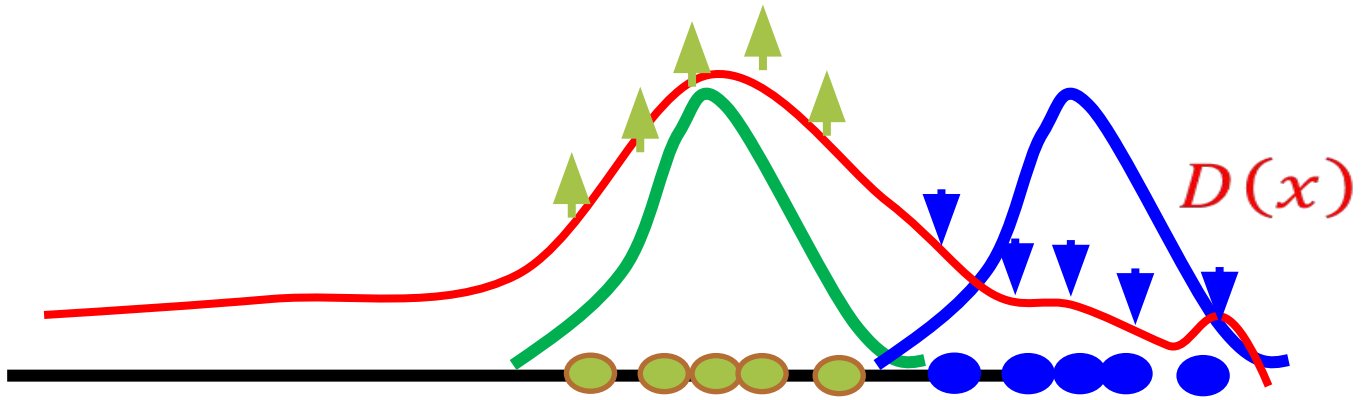
real

generated



In the end ……





Training GANs: Two-player Game

Putting it together: GAN training algorithm

for number of training iterations **do**

  for k steps **do**

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D_{\theta_d}(x^{(i)}) + \log(1 - D_{\theta_d}(G_{\theta_g}(z^{(i)}))) \right]$$

  end for

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by ascending its stochastic gradient (improved objective):

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(D_{\theta_d}(G_{\theta_g}(z^{(i)})))$$

end for

$$\max_D V(G,D)$$

$$\mathrm{JSD}(P \parallel Q) = \frac{1}{2}D(P \parallel M) + \frac{1}{2}D(Q \parallel M)$$

$$M = \frac{1}{2}(P+Q)$$

$$\max_D V(G,D) = V(G,D^*) \qquad D^*(x) = \frac{P_{data}(x)}{P_{data}(x) + P_G(x)}$$

$$= -2log2 + \int_x P_{data}(x) log \frac{P_{data}(x)}{\big(P_{data}(x) + P_G(x)\big)/2} dx$$

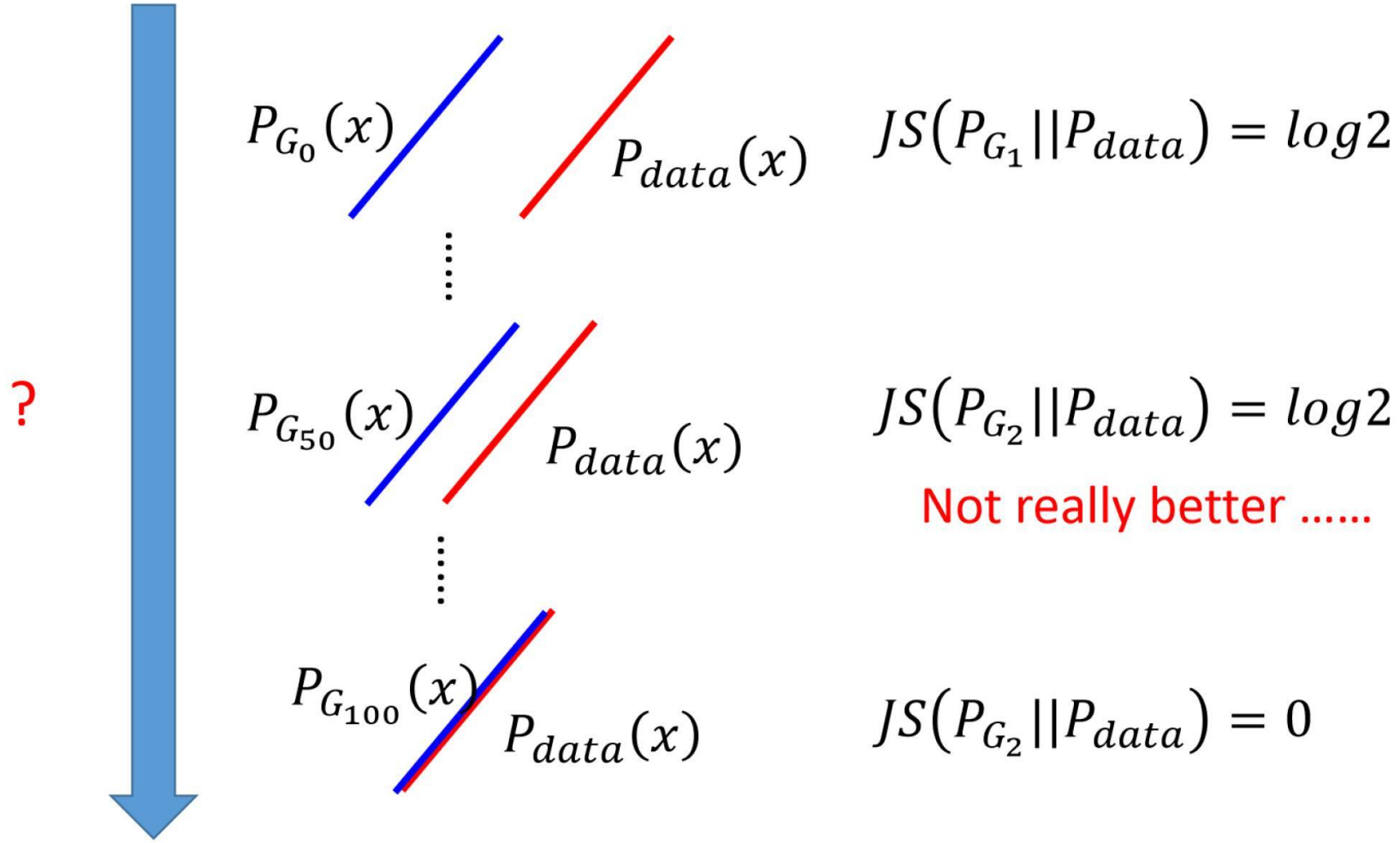
$$+ \int_x P_G(x) log \frac{P_G(x)}{\big(P_{data}(x) + P_G(x)\big)/2} dx$$

$$= -2\log 2 + \mathrm{KL}\left(\mathrm{P_{data}} || \frac{\mathrm{P_{data}} + \mathrm{P_G}}{2}\right) + \mathrm{KL}\left(\mathrm{P_G} || \frac{\mathrm{P_{data}} + \mathrm{P_G}}{2}\right)$$

$$= -2log2 + 2JSD(P_{data}||P_G)$$

Jensen–Shannon divergence

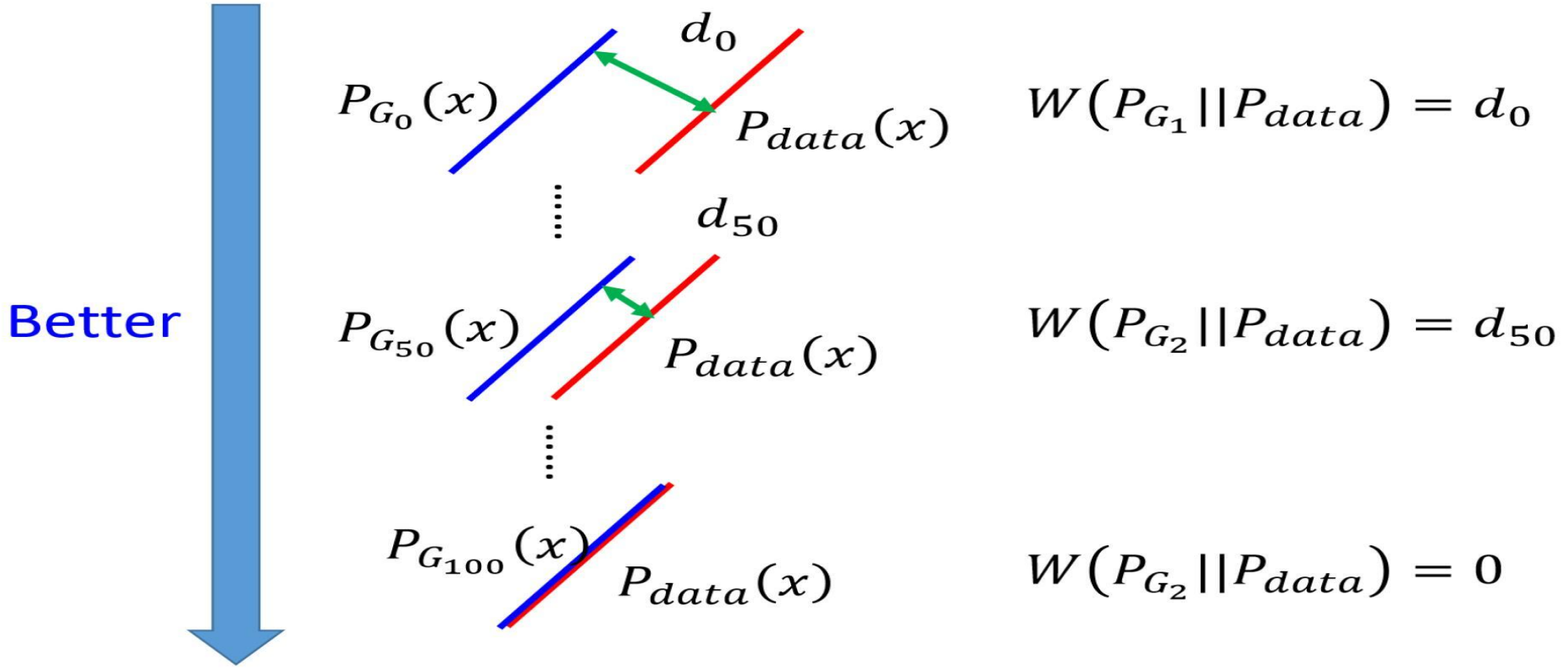
GANs: Hard to Train



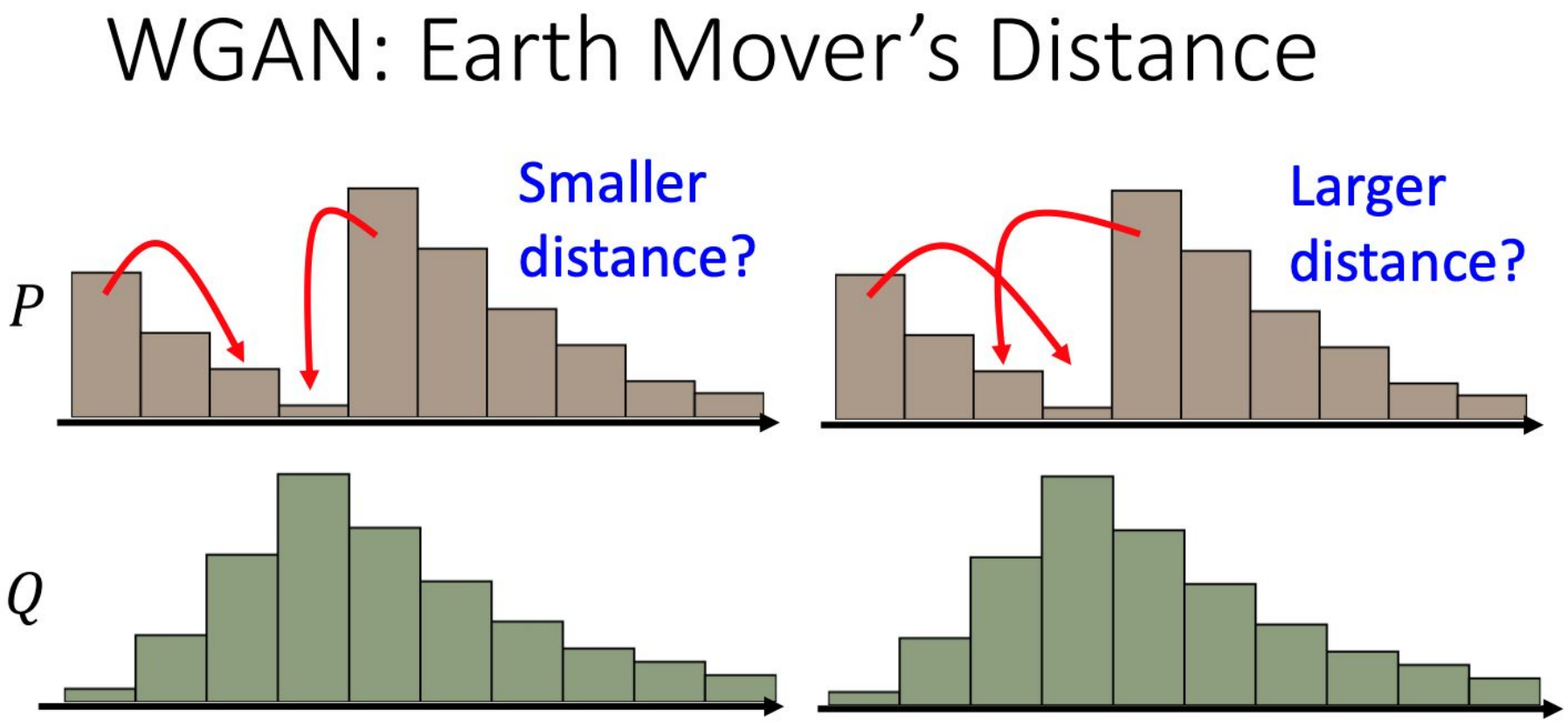
GANs: Hard to Train

WGAN

Using Wasserstein distance instead of JS divergence



WGAN - New Distance



There many possible "moving plans".

Using the "moving plan" with the smallest average distance to define the earth mover's distance.

WGAN - New Formula

[Martin Arjovsky, et al., arXiv, 2017]

WGAN

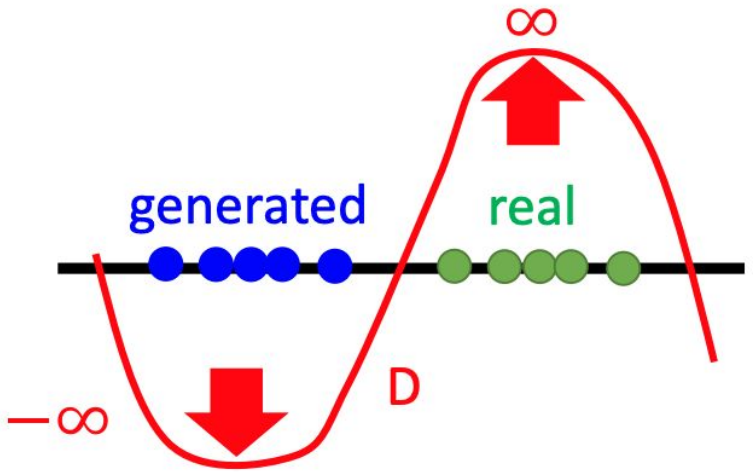
Evaluate wasserstein distance between P_{data} and P_G

$$V(G,D) = \max_{D \in 1-Lipschitz} \{E_{x \sim P_{data}}[D(x)] - E_{x \sim P_G}[D(x)]\}$$

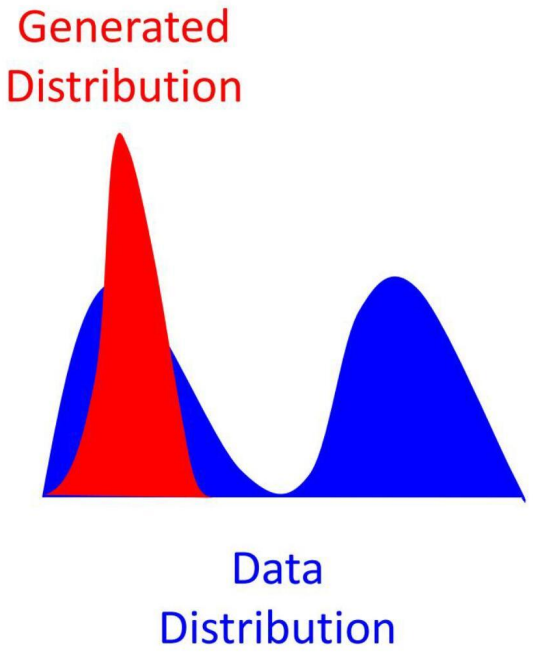
D has to be smooth enough.

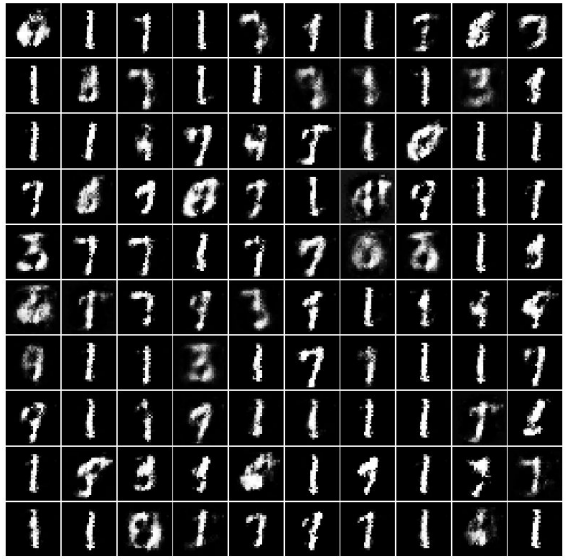
Without the constraint, the training of D will not converge.

Keeping the D smooth forces D(x) become ∞ and $-\infty$

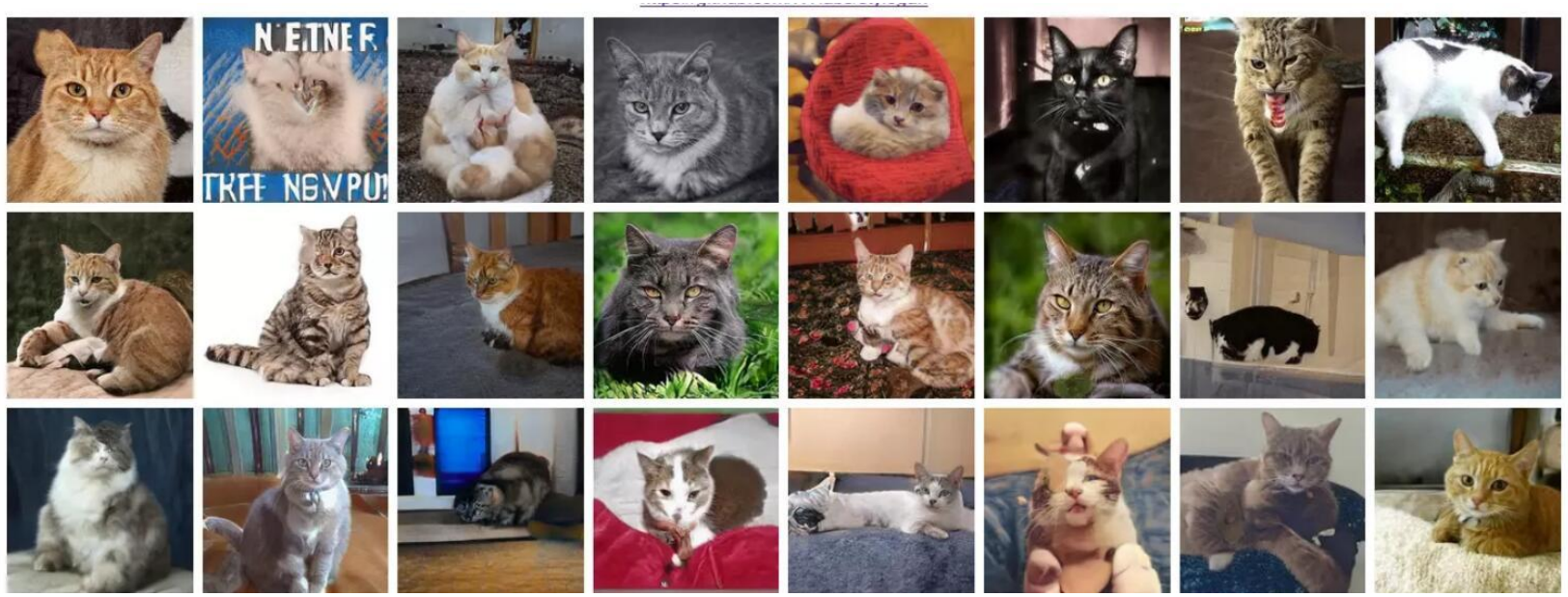


GANs: Mode Collapse



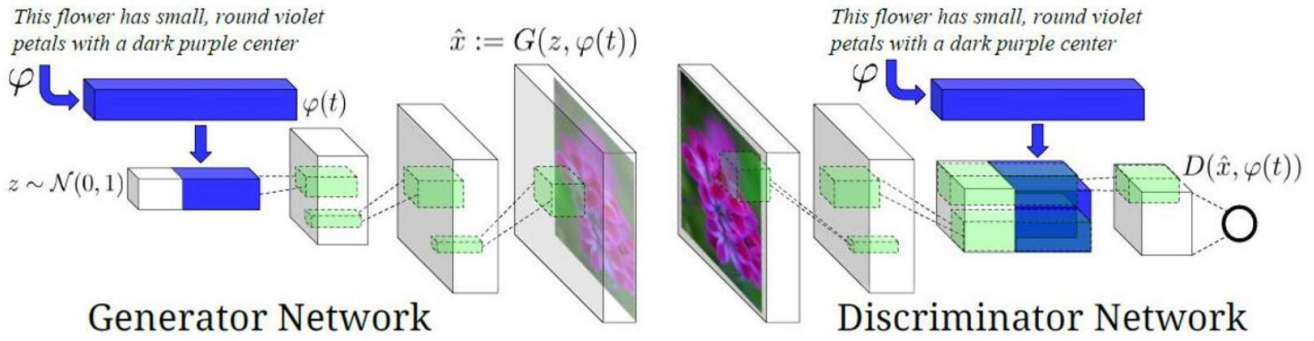


Generation with GANs



So Many GANs

Text to Image



Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, Honglak Lee, "Generative Adversarial Text-to-Image Synthesis", ICML 2016

Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaolei Huang, Xiaogang Wang, Dimitris Metaxas, "StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks", arXiv prepring, 2016

Scott Reed, Zeynep Akata, Santosh Mohan, Samuel Tenka, Bernt Schiele, Honglak Lee, "Learning What and Where to Draw", NIPS 2016

So Many GANs

Text to Image

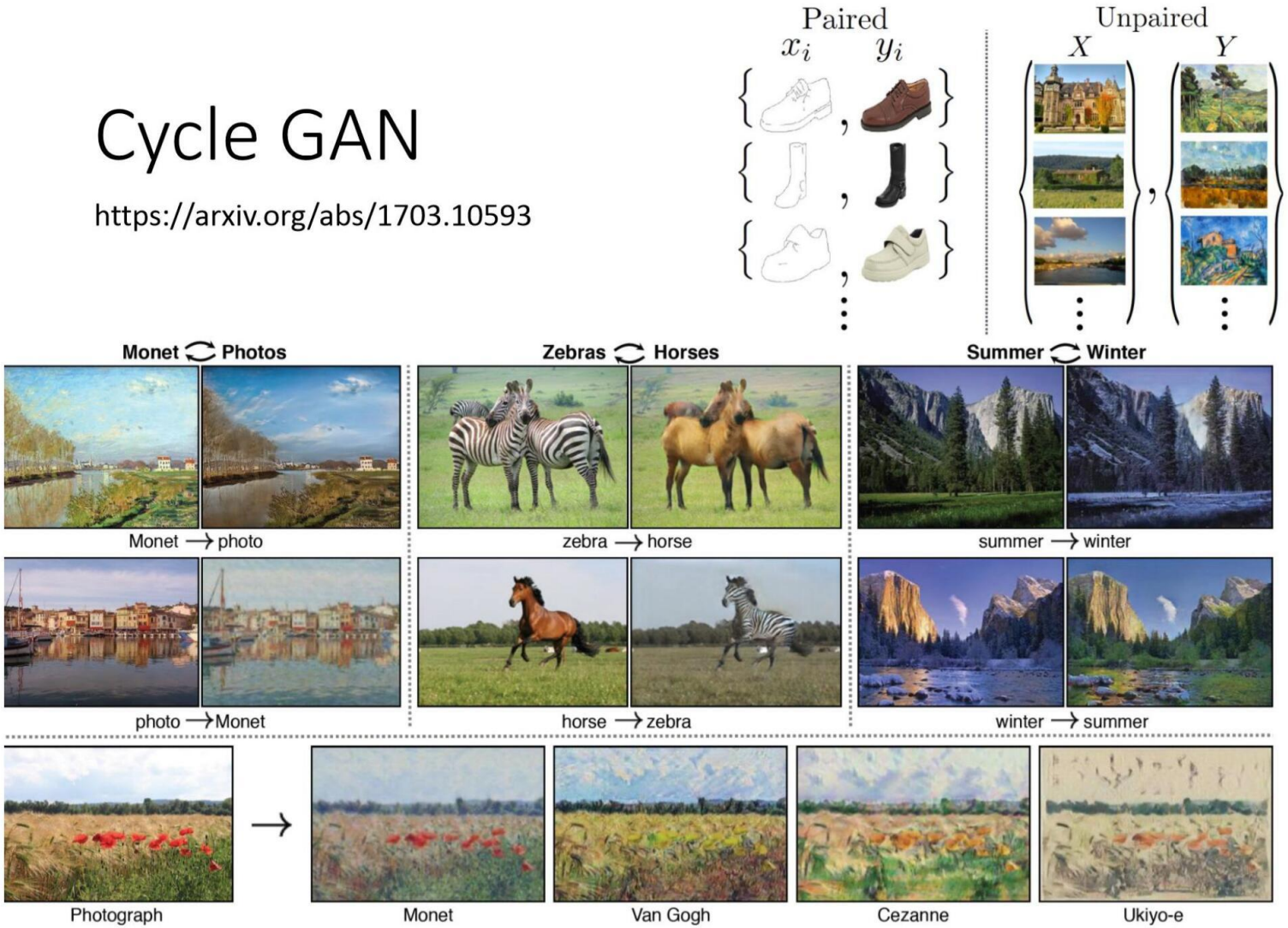
"red flower with black center"

Caption	Image
this flower has white petals and a yellow stamen	
the center is yellow surrounded by wavy dark purple petals	
this flower has lots of small round pink petals	

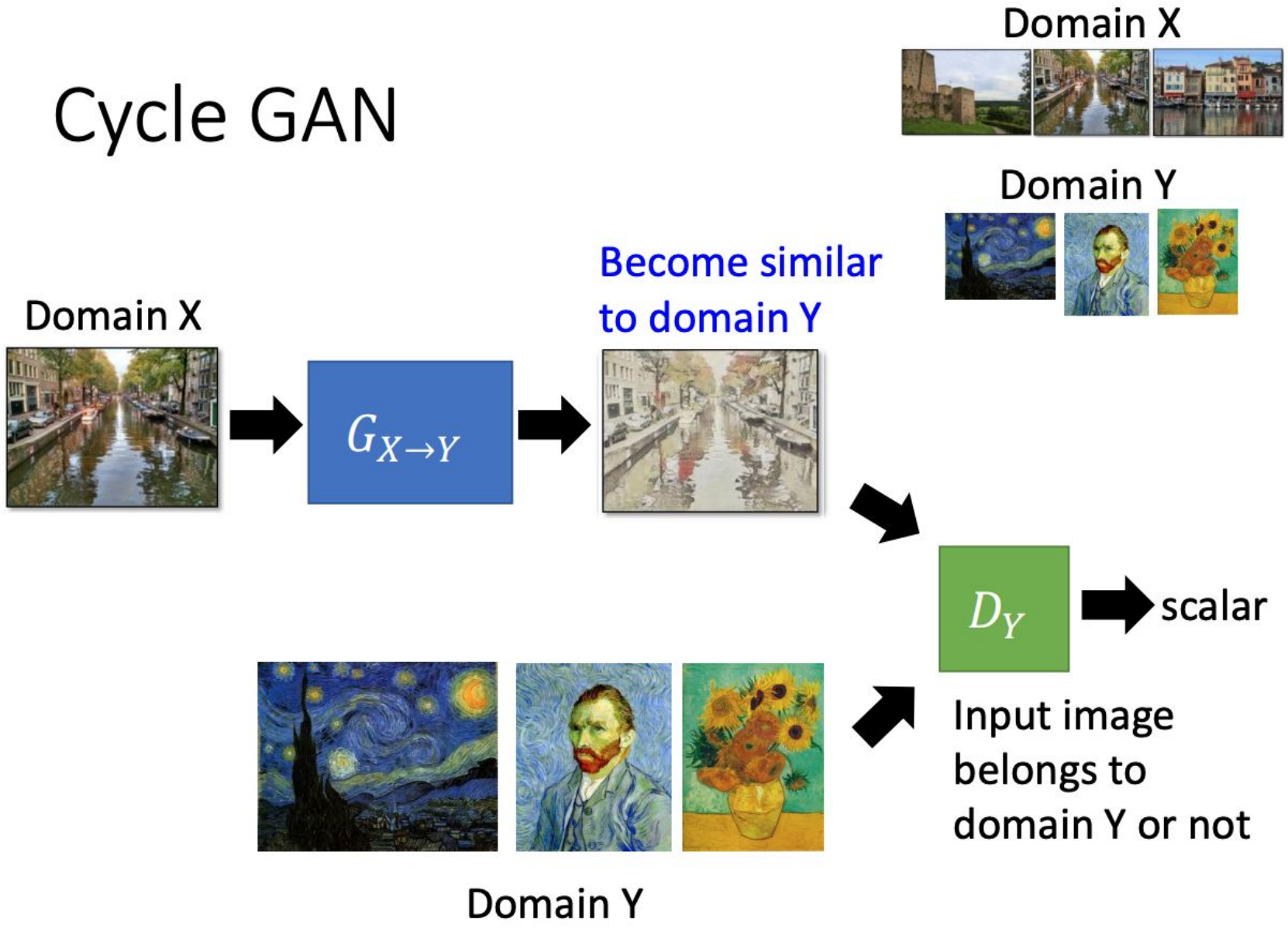
So Many GANs

Cycle GAN

https://arxiv.org/abs/1703.10593

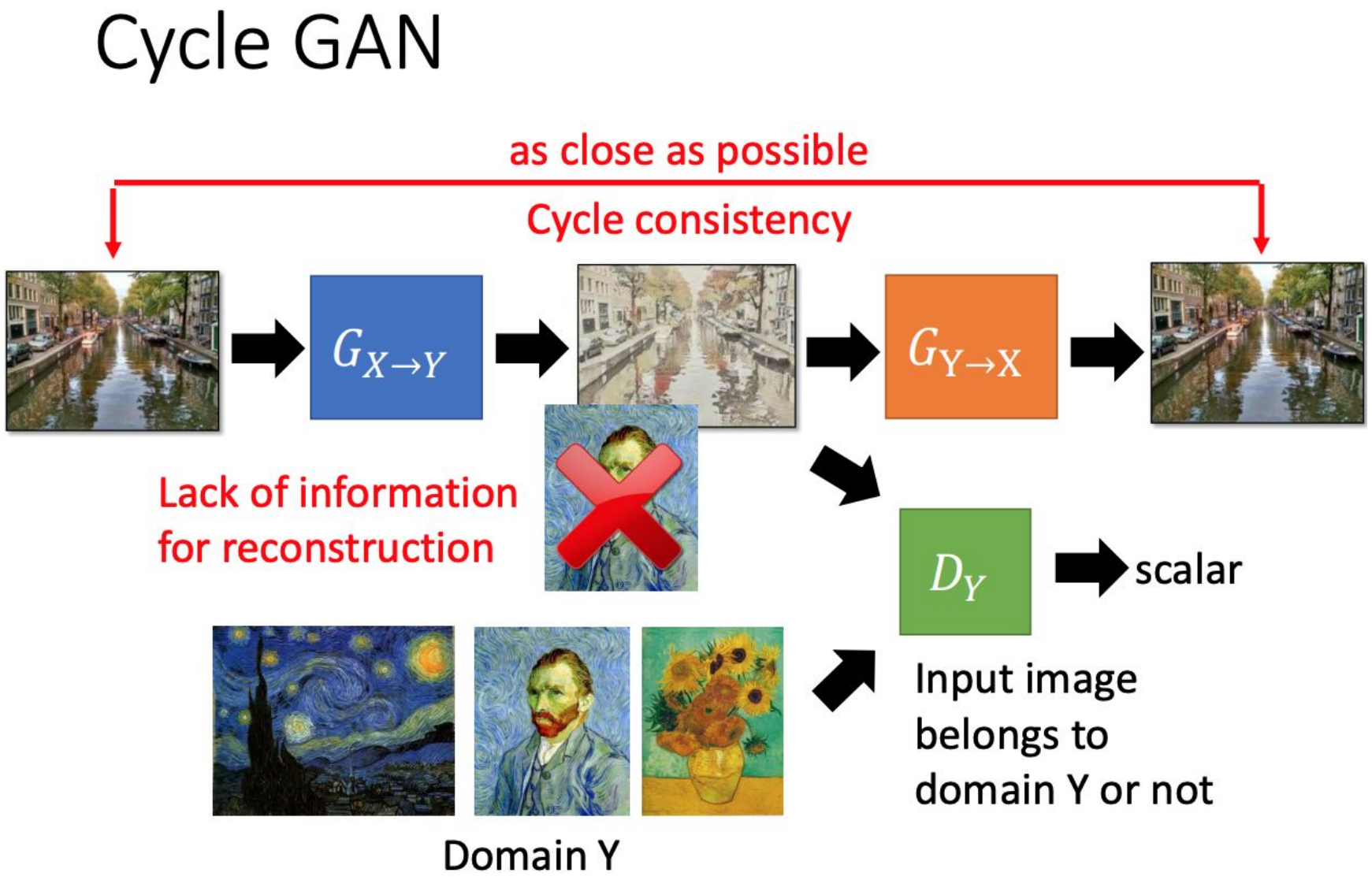


So Many GANs

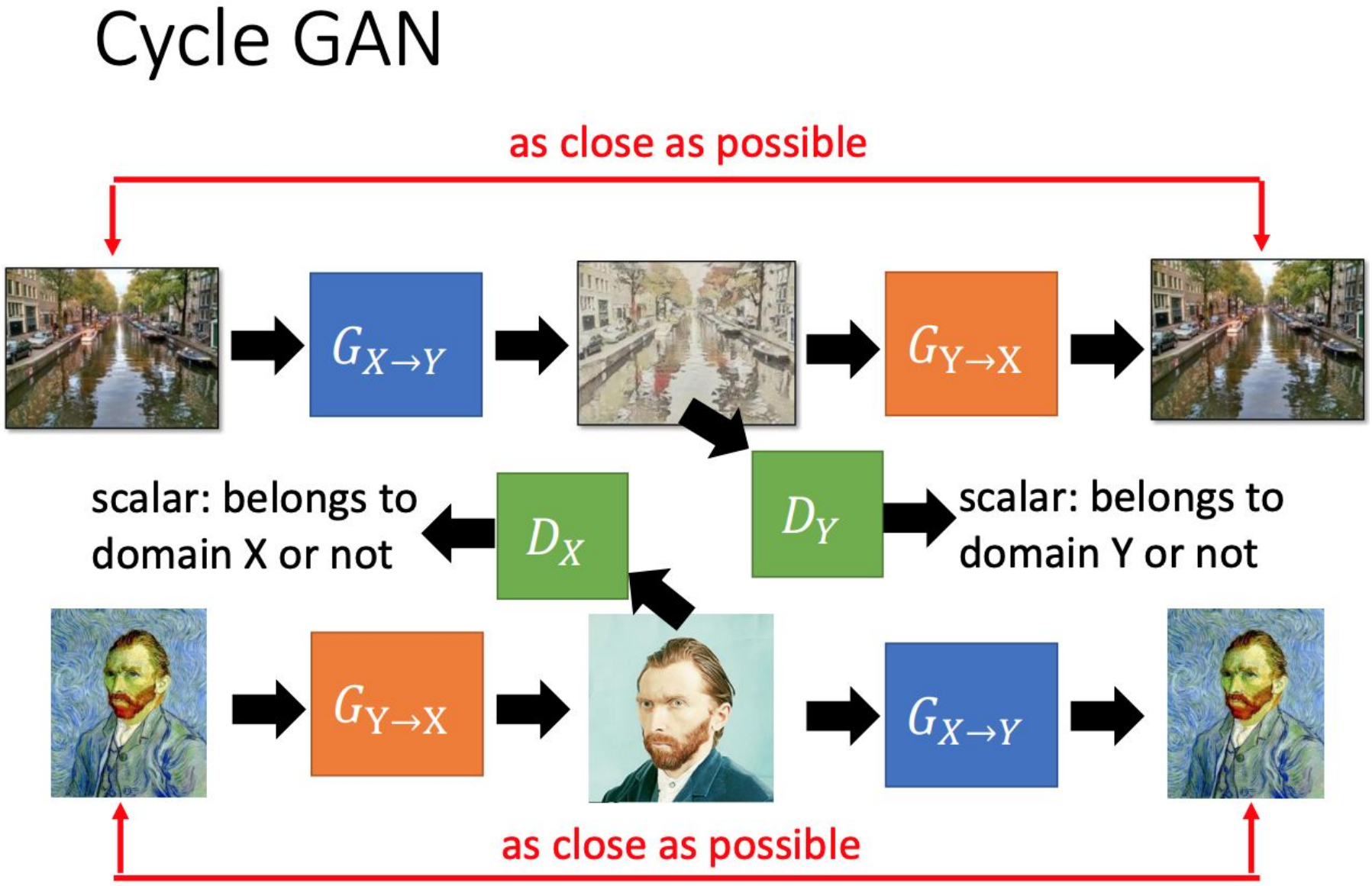

Cycle GAN
Domain X
Domain Y
Become similar to domain Y
Domain X
$G_{X \to Y}$
D_Y
scalar
Input image belongs to domain Y or not
Domain Y

So Many GANs

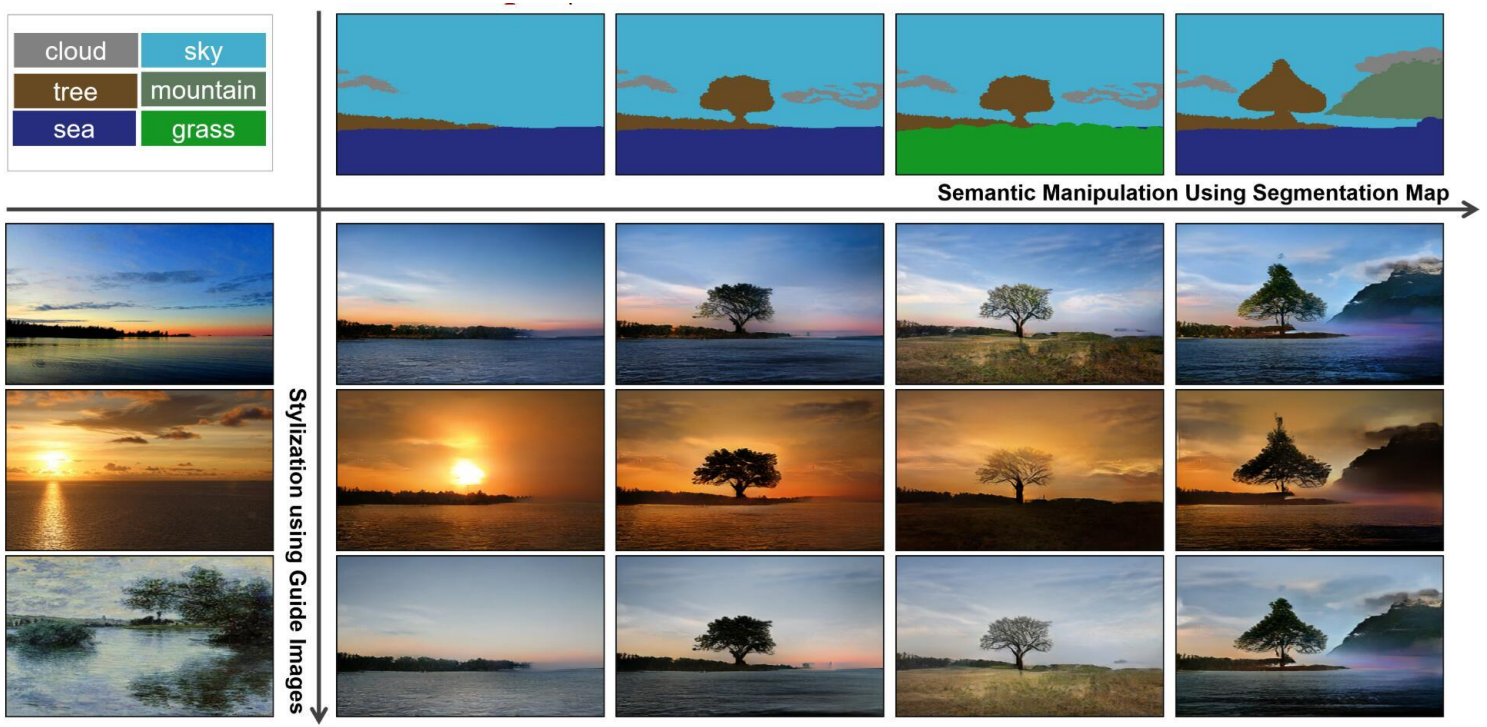
[Jun-Yan Zhu, et al., ICCV, 2017]



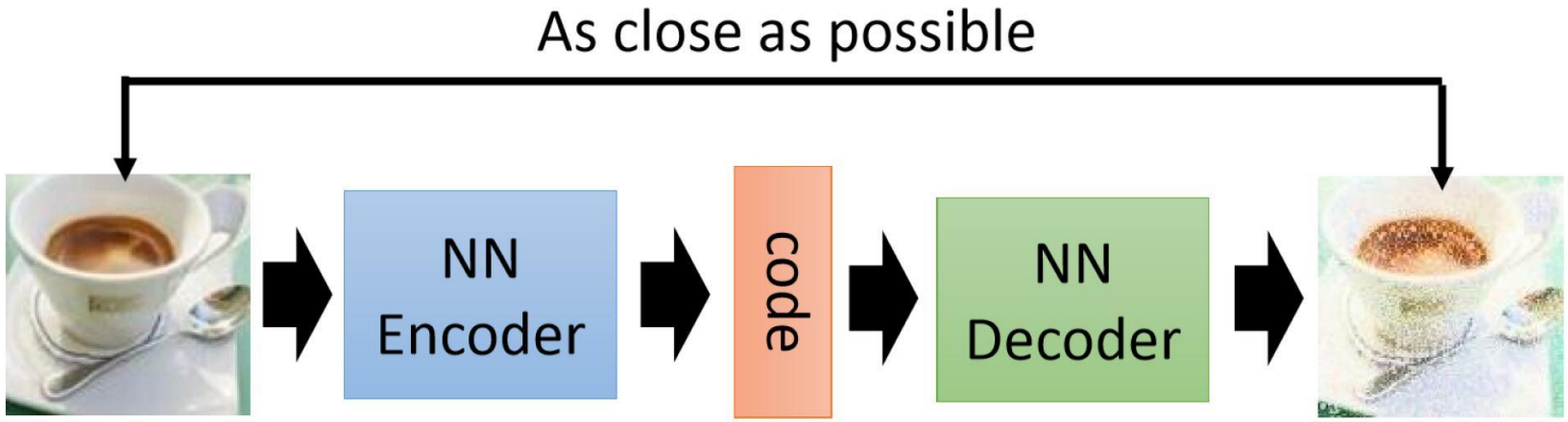
So Many GANs


Cycle GAN
as close as possible
$G_{X\to Y}$
$G_{Y\to X}$
D_X
D_Y
scalar: belongs to domain X or not
scalar: belongs to domain Y or not
$G_{Y\to X}$
$G_{X\to Y}$
as close as possible

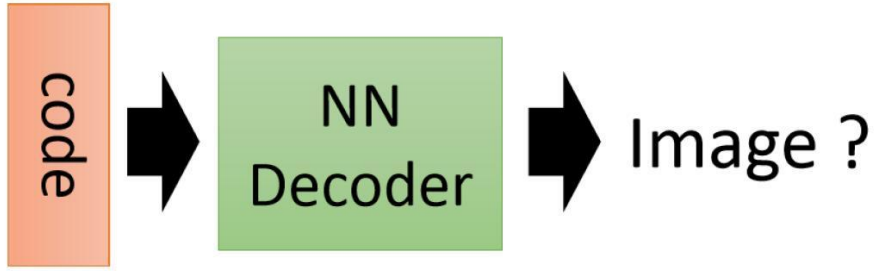
So Many GANs


cloud
sky
tree
mountain
sea
grass
Semantic Manipulation Using Segmentation Map
Stylization using Guide Images

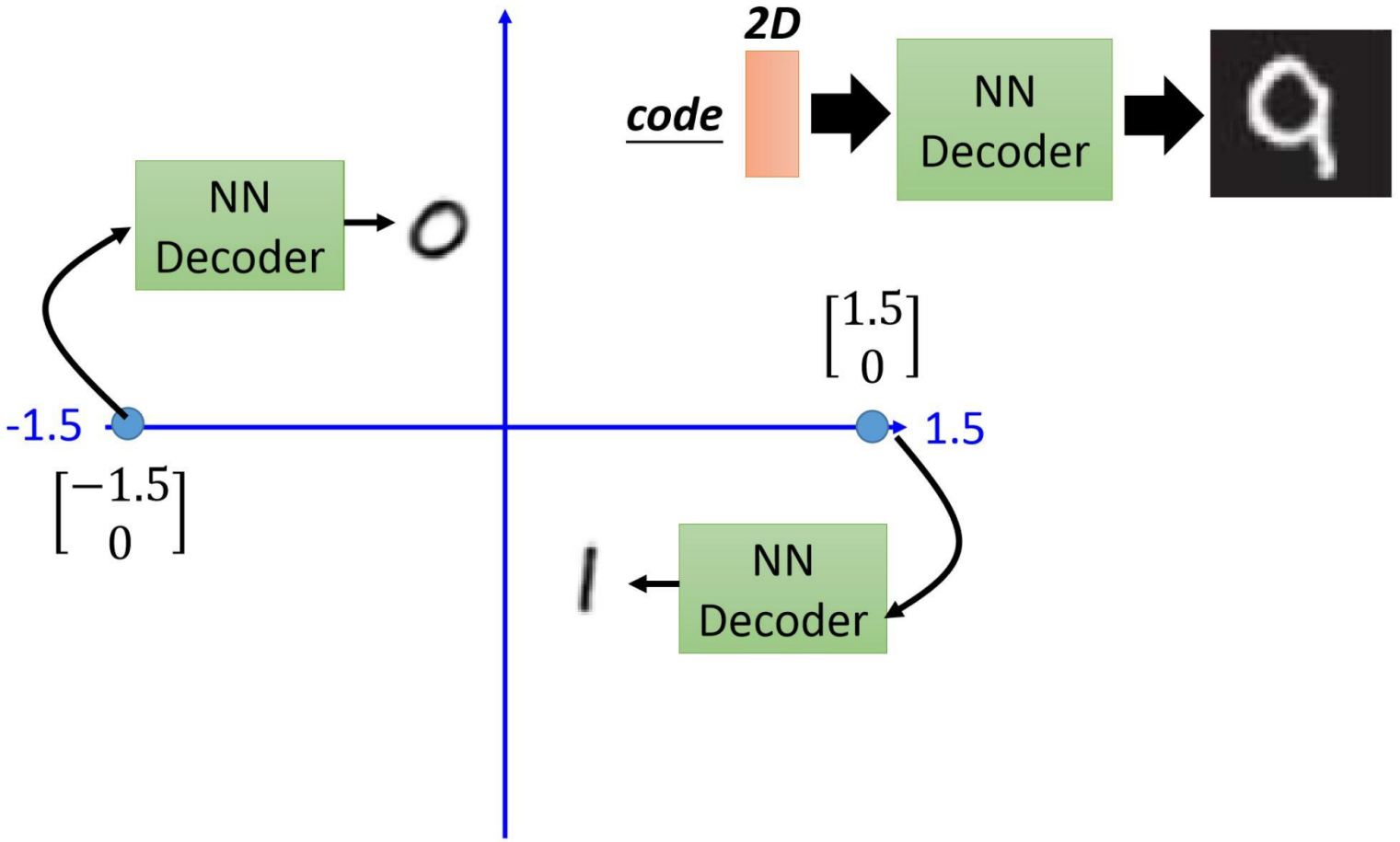
Review: Auto-Encoder



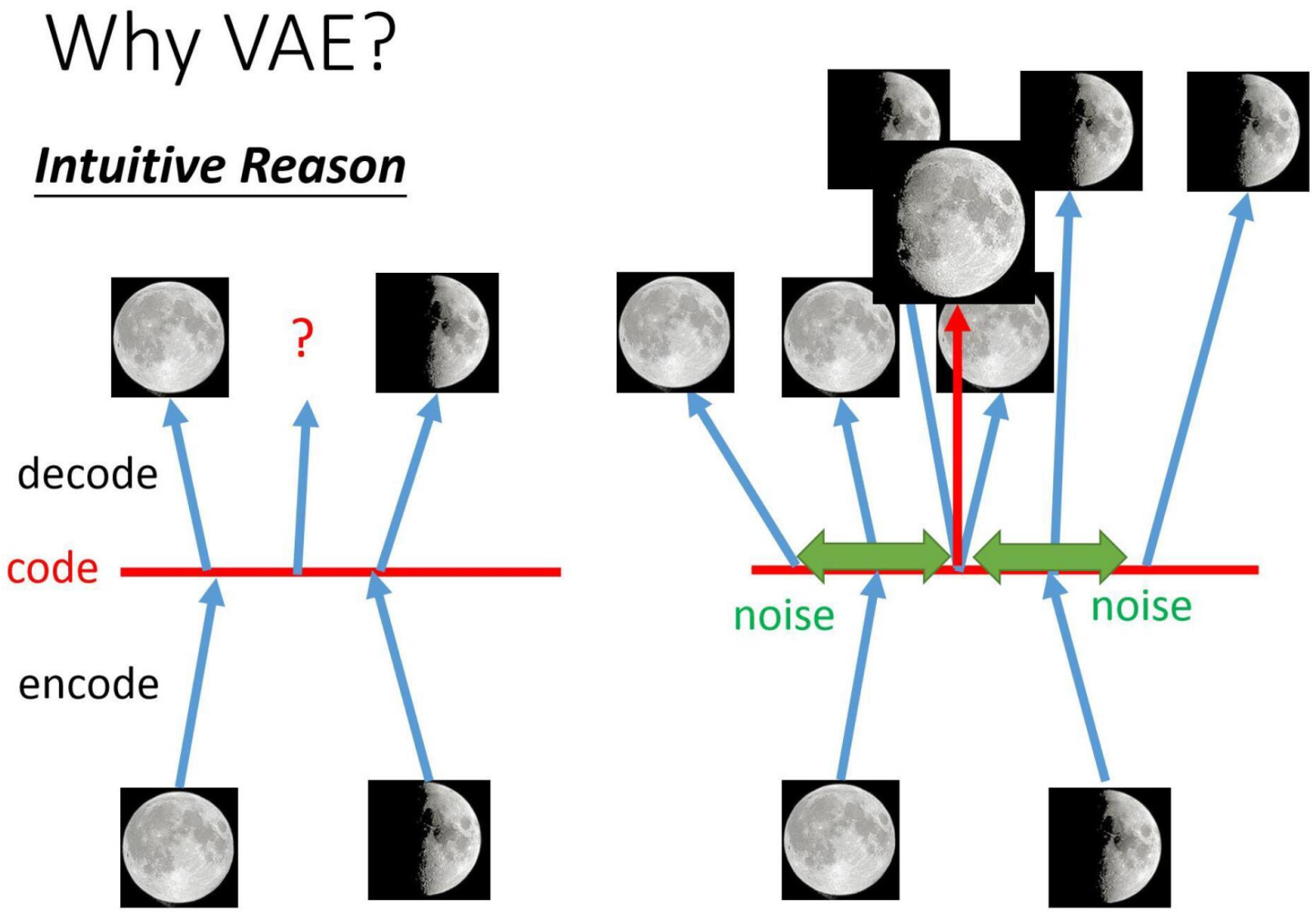
Randomly generate a vector as code



Review: Auto-Encoder



Review: Auto-Encoder



Variational Auto-Encoder

So far...

PixelCNNs define tractable density function, optimize likelihood of training data:

$$p_\theta(x) = \prod_{i=1}^{n} p_\theta(x_i | x_1, ..., x_{i-1})$$

VAEs define intractable density function with latent **z**:

$$p_\theta(x) = \int p_\theta(z) p_\theta(x|z) dz$$

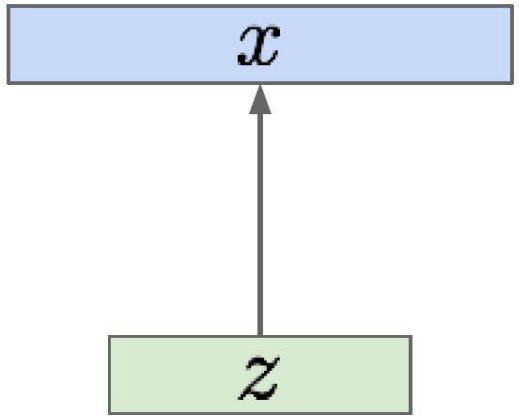
Cannot optimize directly, derive and optimize lower bound on likelihood instead

Variational Auto-Encoder

Assume training data $\{x^{(i)}\}_{i=1}^{N}$ is generated from underlying unobserved (latent) representation **z**

Sample from true conditional $p_{\theta^*}(x \mid z^{(i)})$

Sample from true prior $p_{\theta^*}(z)$



Intuition (remember from autoencoders!): **x** is an image, **z** is latent factors used to generate **x:** attributes, orientation, etc.

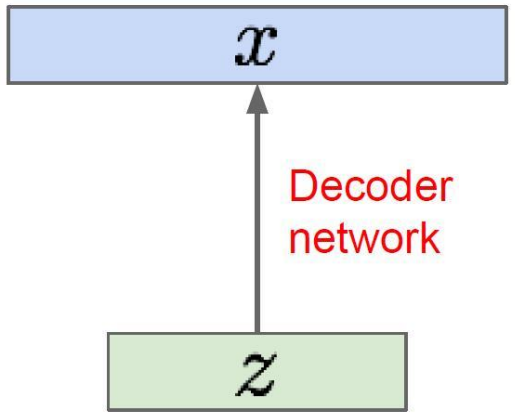
Variational Auto-Encoder

Sample from true conditional

$p_{\theta^*}(x \mid z^{(i)})$

Sample from true prior

$p_{\theta^*}(z)$



We want to estimate the true parameters θ^* of this generative model.

How should we represent this model?

Choose prior p(z) to be simple, e.g. Gaussian.

Conditional p(x|z) is complex (generates image) => represent with neural network

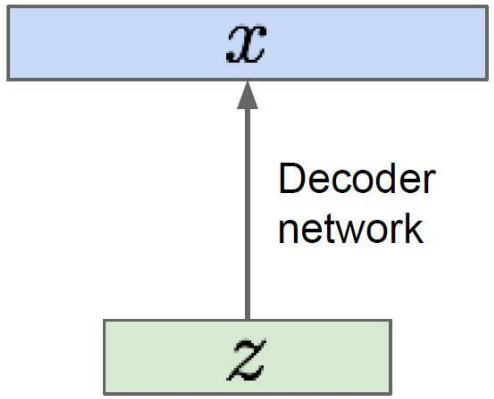
Variational Auto-Encoder

Sample from true conditional

$p_{\theta^*}(x \mid z^{(i)})$

Sample from true prior

$p_{\theta^*}(z)$



We want to estimate the true parameters θ^* of this generative model.

How to train the model?

Remember strategy for training generative models from FVBNs. Learn model parameters to maximize likelihood of training data

$$p_\theta(x) = \int p_\theta(z) p_\theta(x|z) dz$$

Now with latent z

VAE: Intractability

Data likelihood: $p_\theta(x) = \int p_\theta(z) p_\theta(x|z) dz$

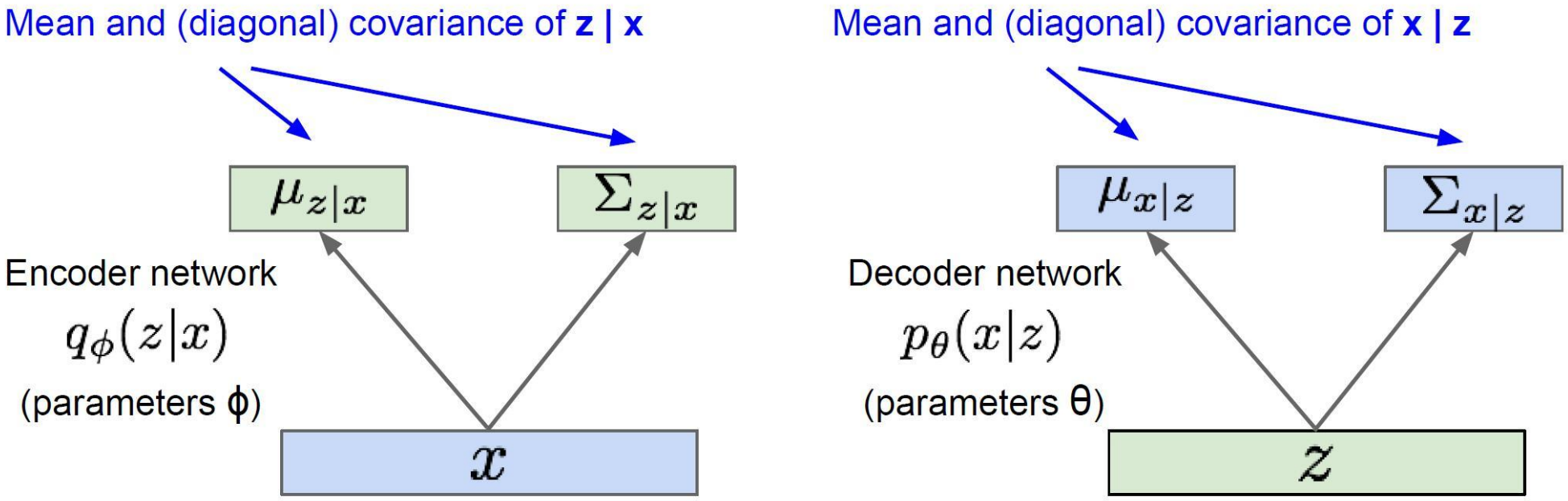
Posterior density also intractable: $p_\theta(z|x) = p_\theta(x|z) p_\theta(z) / p_\theta(x)$

Solution: In addition to decoder network modeling $p_\theta(x|z)$, define additional encoder network $q_\phi(z|x)$ that approximates $p_\theta(z|x)$

Will see that this allows us to derive a lower bound on the data likelihood that is tractable, which we can optimize

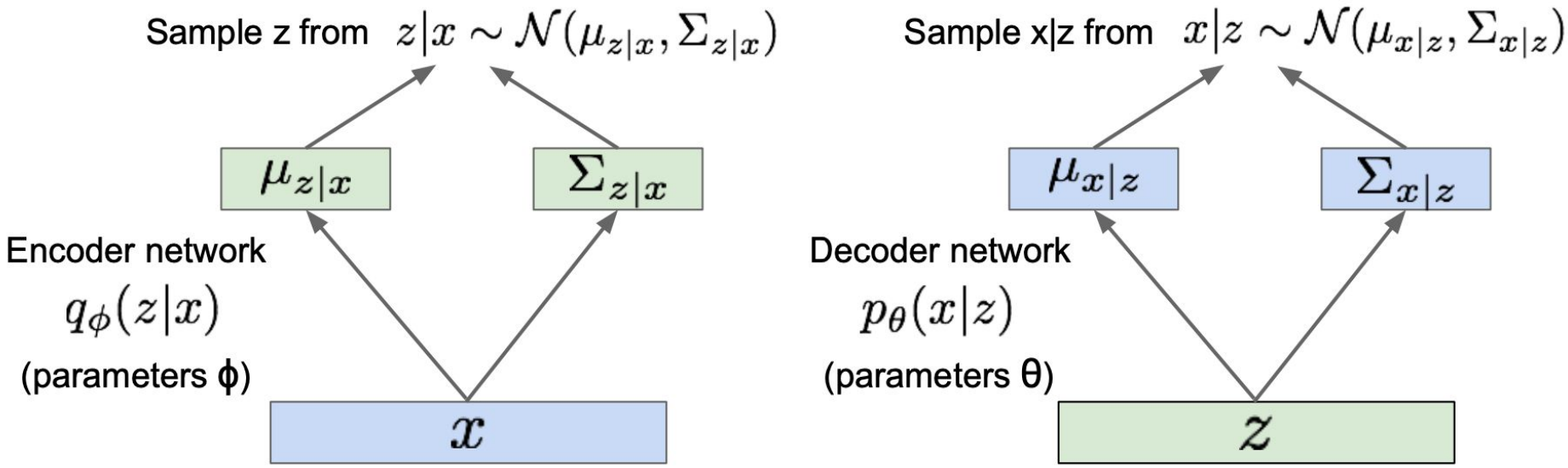
VAE: Probabilistic

Since we're modeling probabilistic generation of data, encoder and decoder networks are probabilistic



VAE: Probabilistic

Since we're modeling probabilistic generation of data, encoder and decoder networks are probabilistic



Encoder and decoder networks also called "recognition"/"inference" and "generation" networks

Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

VAE: Formulation

Now equipped with our encoder and decoder networks, let's work out the (log) data likelihood:

$$\log p_\theta(x^{(i)}) = \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z)$$

$$= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\right] \quad \text{(Bayes' Rule)}$$

$$= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\frac{q_\phi(z \mid x^{(i)})}{q_\phi(z \mid x^{(i)})}\right] \quad \text{(Multiply by constant)}$$

$$= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z)}\right] + \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z \mid x^{(i)})}\right] \quad \text{(Logarithms)}$$

$$= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z)) + D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))$$

Decoder network gives $p_\theta(x|z)$, can compute estimate of this term through sampling. (Sampling differentiable through reparam. trick, see paper.)

This KL term (between Gaussians for encoder and z prior) has nice closed-form solution!

$p_\theta(z|x)$ intractable (saw earlier), can't compute this KL term :(But we know KL divergence always >= 0.

VAE: Formulation

Now equipped with our encoder and decoder networks, let's work out the (log) data likelihood:

$$\log p_\theta(x^{(i)}) = \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z)$$

$$= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\right] \quad \text{(Bayes' Rule)}$$

$$= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\frac{q_\phi(z \mid x^{(i)})}{q_\phi(z \mid x^{(i)})}\right] \quad \text{(Multiply by constant)}$$

$$= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z)}\right] + \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z \mid x^{(i)})}\right] \quad \text{(Logarithms)}$$

$$= \underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)},\theta,\phi)} + \underbrace{D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))}_{\geq 0}$$

Reconstruct the input data

Make approximate posterior distribution close to prior

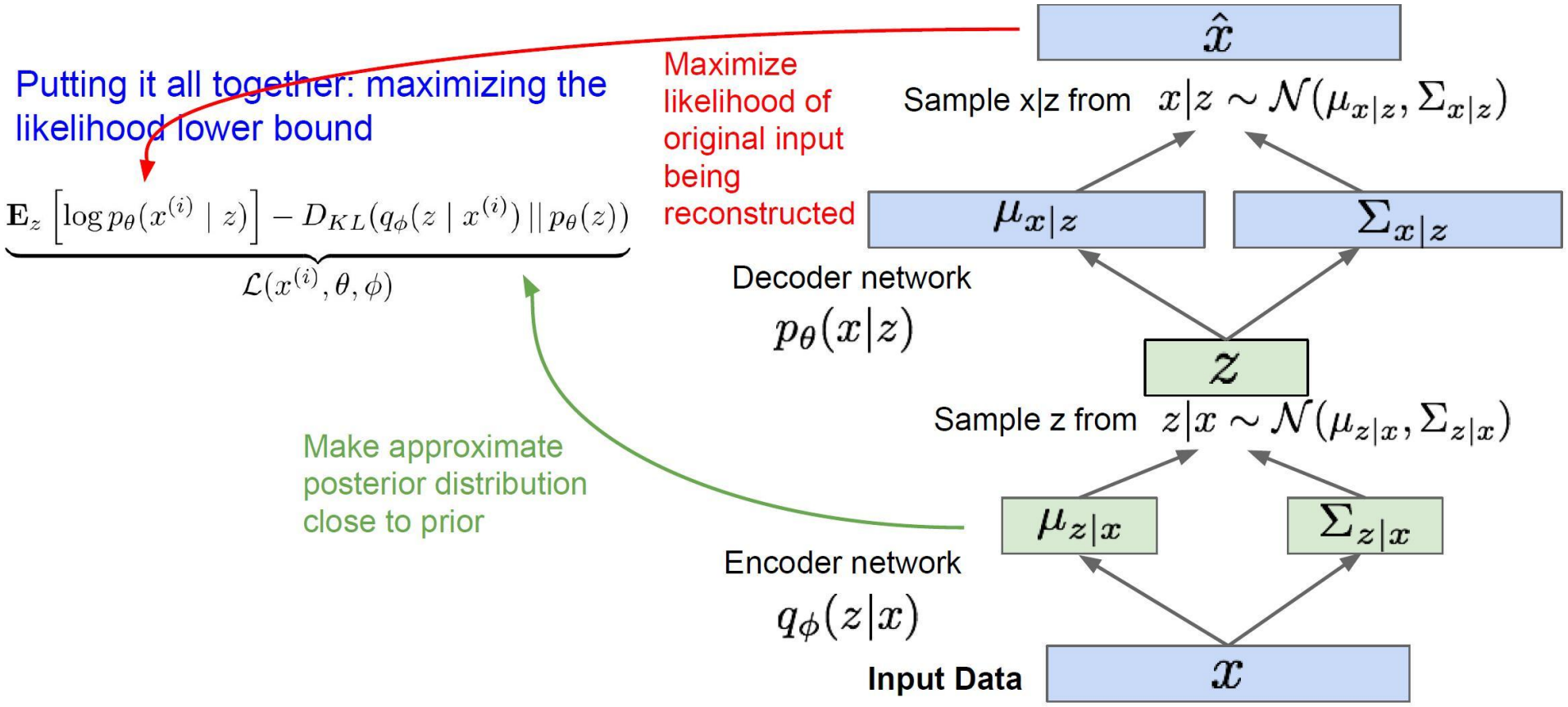
$$\log p_\theta(x^{(i)}) \geq \mathcal{L}(x^{(i)},\theta,\phi)$$

Variational lower bound ("ELBO")

$$\theta^*, \phi^* = \arg\max_{\theta,\phi} \sum_{i=1}^{N} \mathcal{L}(x^{(i)},\theta,\phi)$$

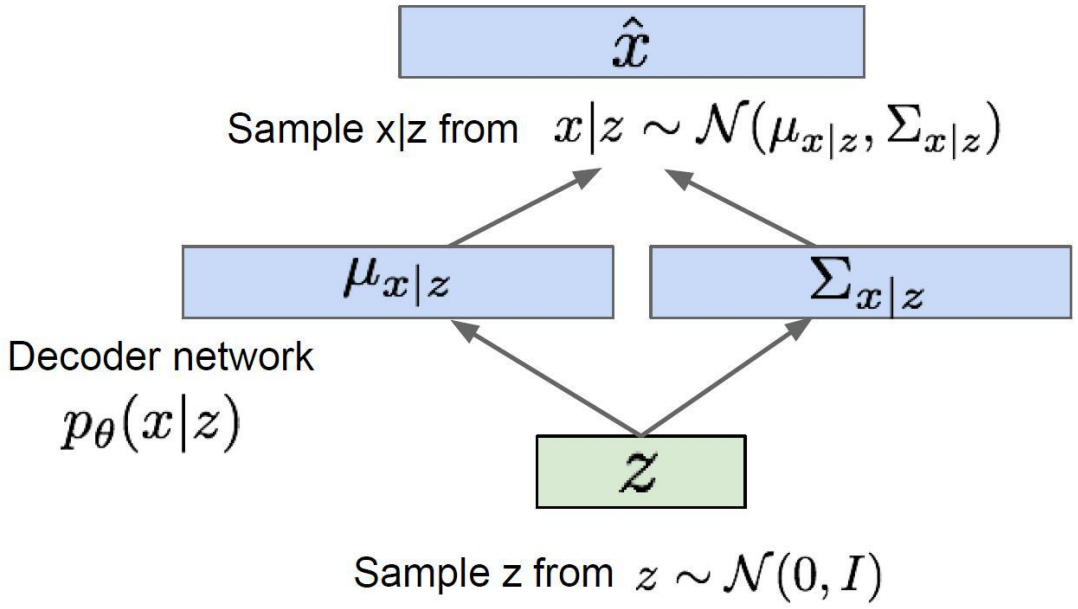
Training: Maximize lower bound

VAE: Put it Together



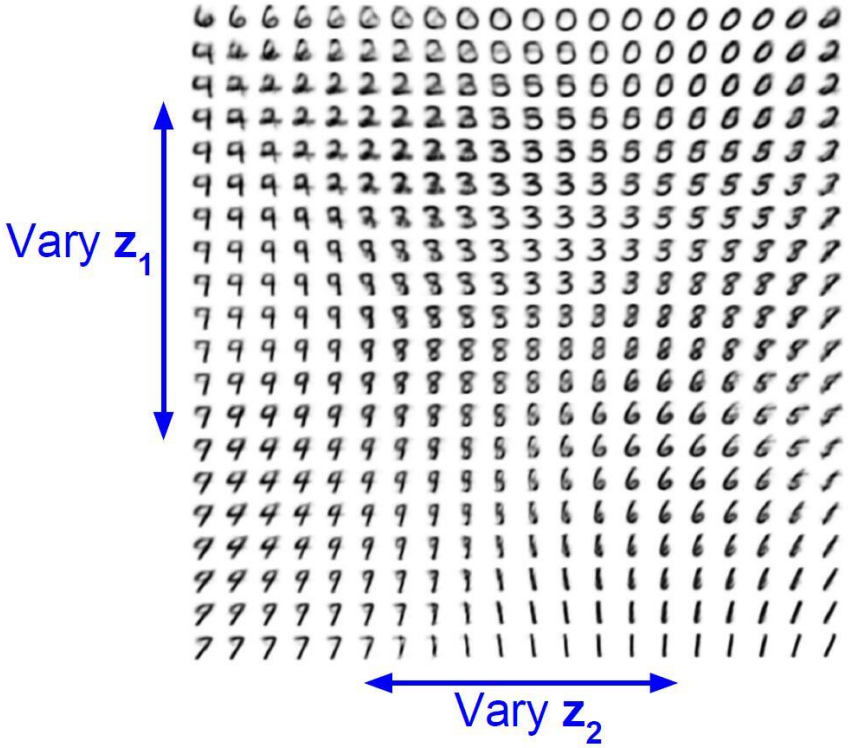
VAE: Generation

Use decoder network. Now sample z from prior!



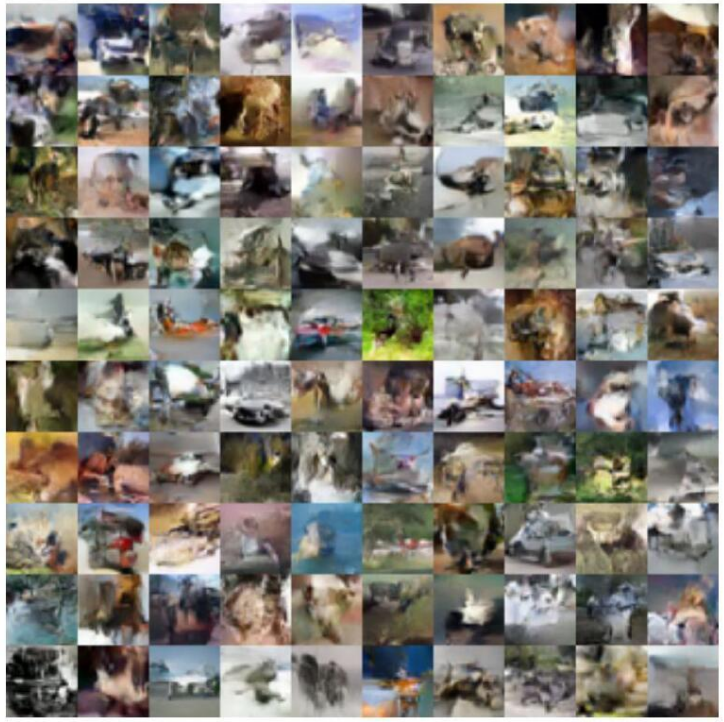
Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Data manifold for 2-d **z**

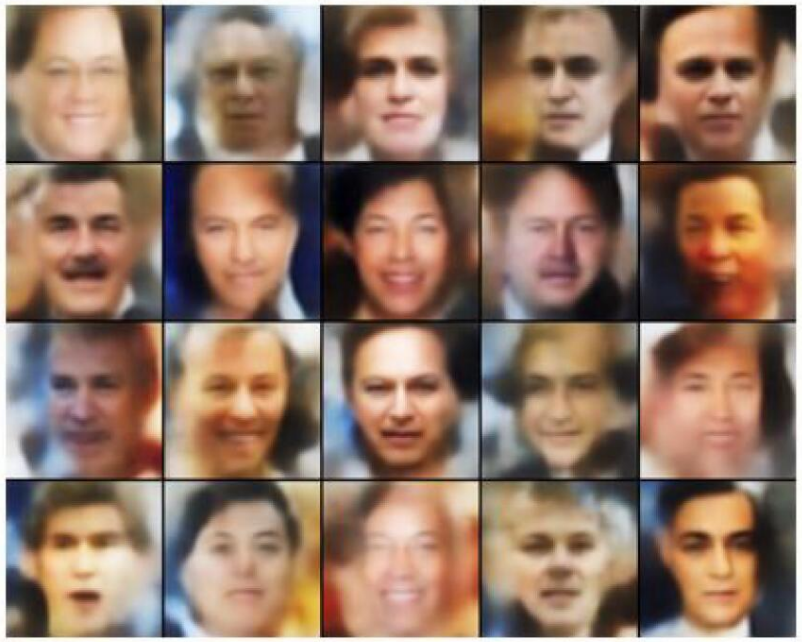


VAE: Generation

Blurry results.



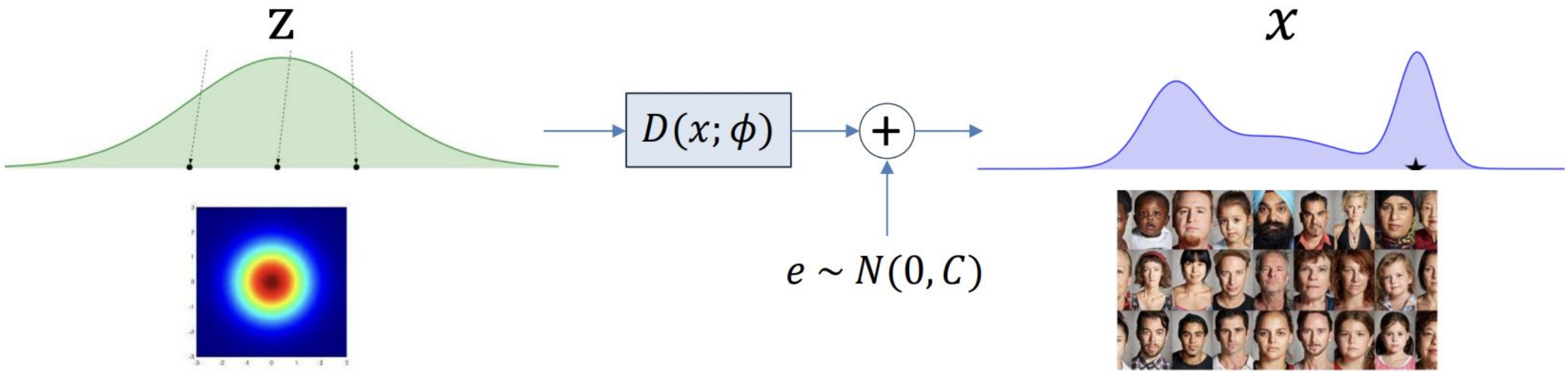
32x32 CIFAR-10



Labeled Faces in the Wild

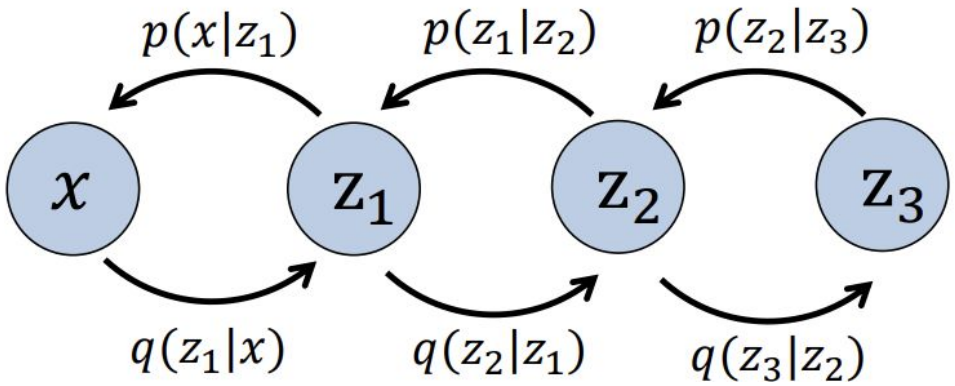
Limitations of VAEs

- Decoder must transform a standard Gaussian all the way to the target distribution in **one-step**
  - Often too large a gap
  - Blurry results are generated



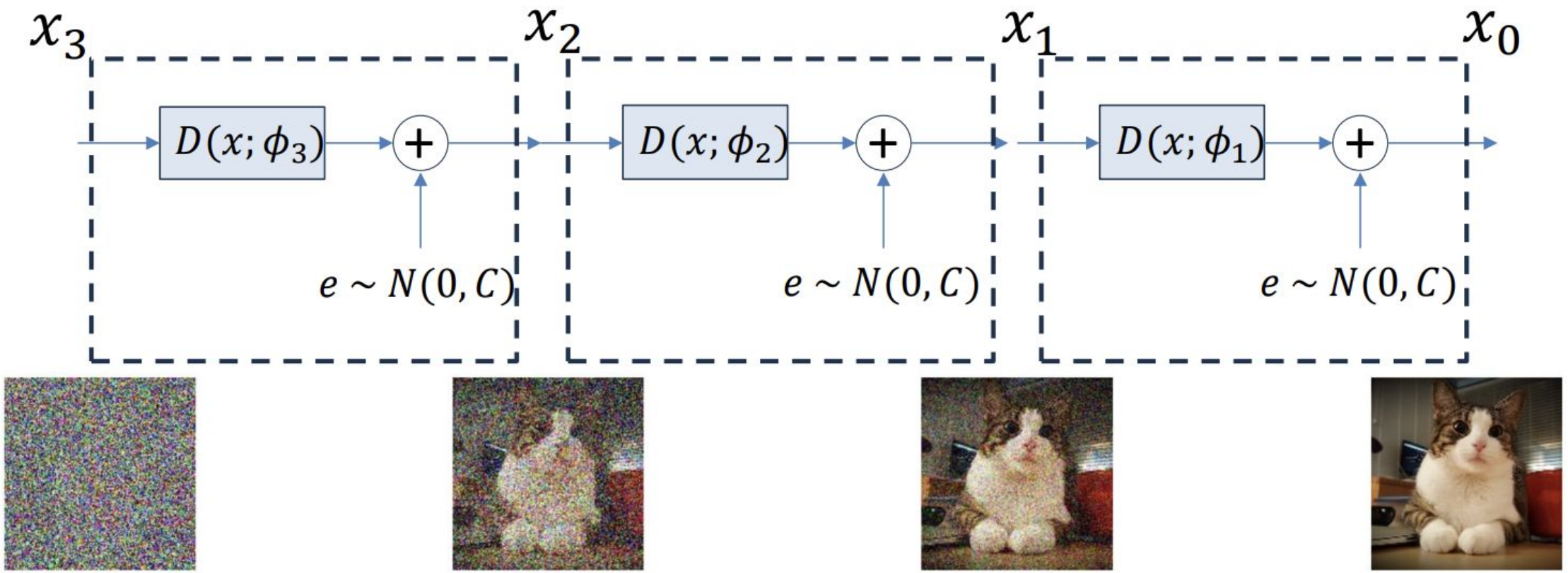
Solution: Hierarchical VAEs

- Hierarchical VAEs – Stacking VAEs on top of each other
  - Multiple (T) intermediate latent
  - Joint distribution $p(\boldsymbol{x}, \boldsymbol{z}_{1:T}) = p(\boldsymbol{z}_T) p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z}_1) \prod_{t=2}^{T} p_{\boldsymbol{\theta}}(\boldsymbol{z}_{t-1} \mid \boldsymbol{z}_t)$
  - Posterior $q_{\boldsymbol{\phi}}(\boldsymbol{z}_{1:T} \mid \boldsymbol{x}) = q_{\boldsymbol{\phi}}(\boldsymbol{z}_1 \mid \boldsymbol{x}) \prod_{t=2}^{T} q_{\boldsymbol{\phi}}(\boldsymbol{z}_t \mid \boldsymbol{z}_{t-1})$
- Better likelihood achieved!



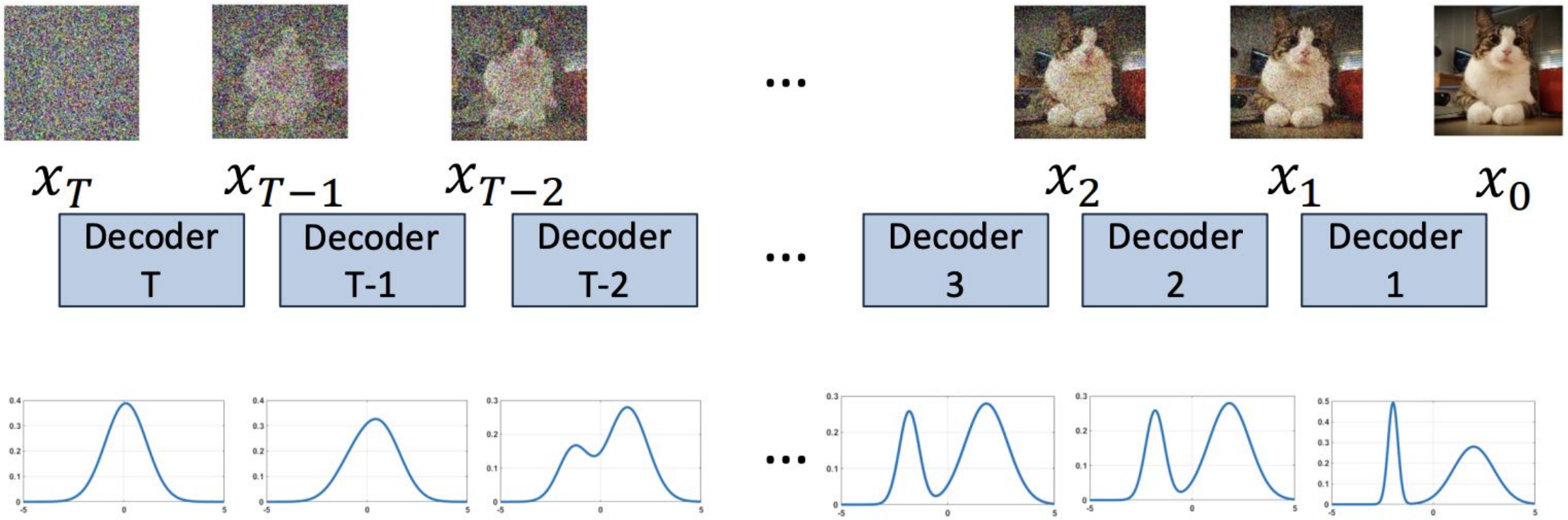
Stacking VAEs

- Each step, the decoder removes part of the noise
- Provides a seed model closer to final distribution



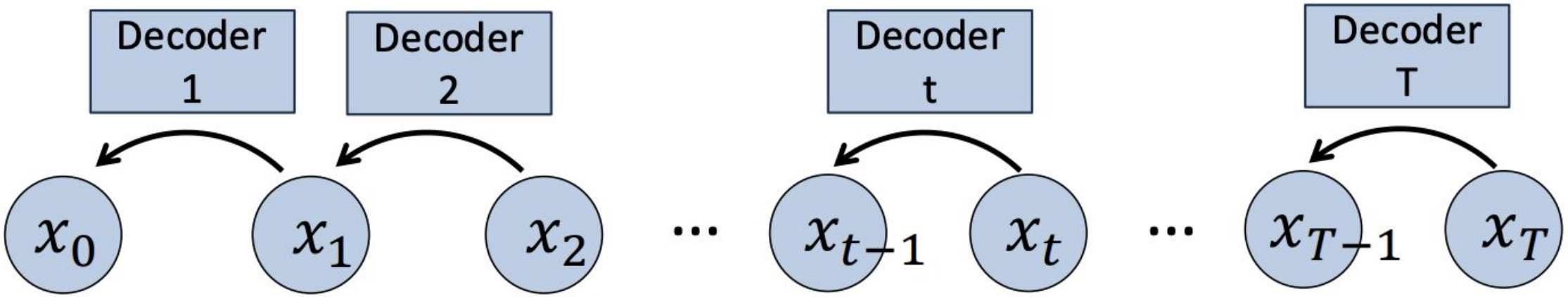
Stacking VAEs

- We can have many many steps (in total T)…
- Each step incrementally recovers the final distribution



Diffusion Models are Stacking VAEs

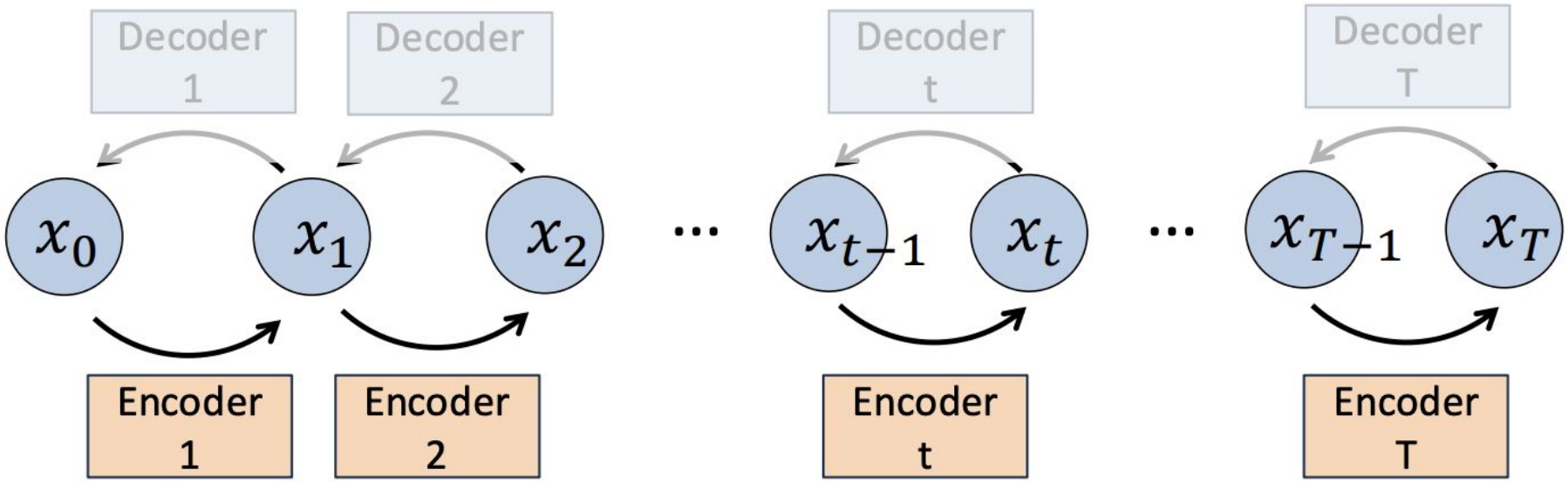
- Diffusion models are special cases of Stacking VAEs



- The reverse denoising process is the stack of decoders
- What about encoders?

Diffusion Models are Stacking VAEs

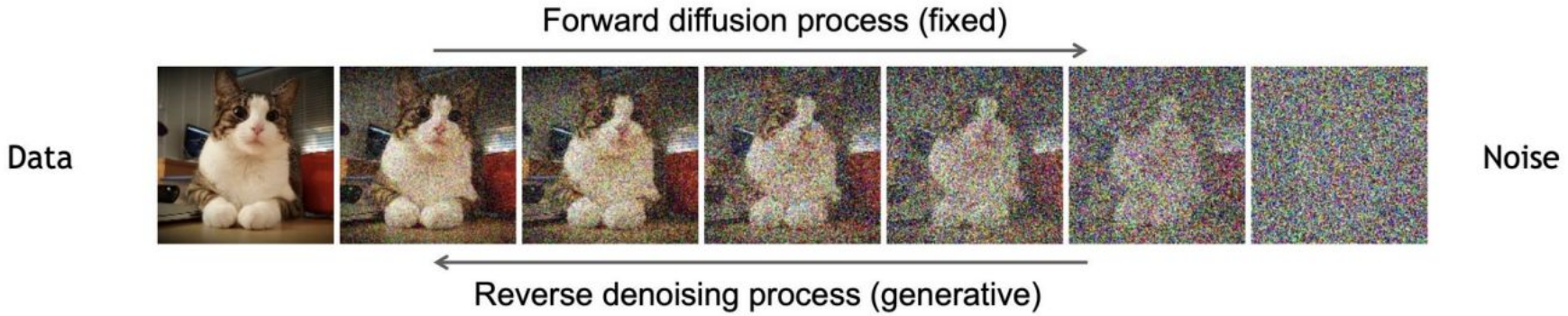
- Diffusion models are special case of Stacking VAEs



- In VAEs, encoders are learned with KL-divergence between the posterior and the prior
- Suffers from the 'posterior-collapse' issue
- Diffusion models use **fixed inference encoders**

Diffusion Process

- Diffusion models have two processes
- **Forward diffusion process** gradually adds noise to input
- **Reverse denoising process** learns to generate data by denoising

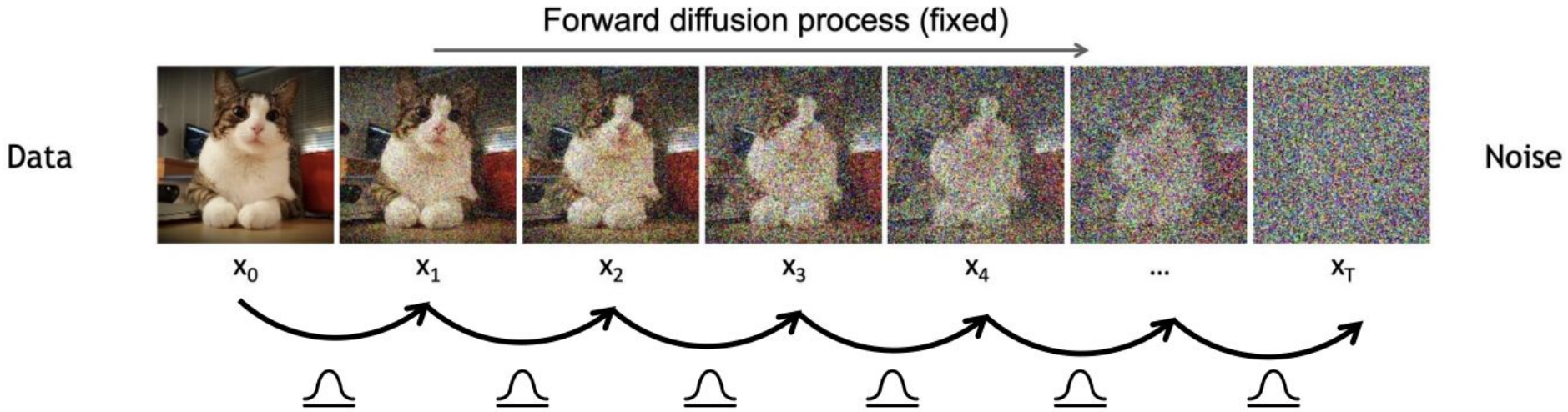


Diffusion Process (2)

- Forward diffusion process is stacking **fixed** VAE encoders
  - gradually adding Gaussian noise according to schedule β_t

$$q\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}\right)=\mathcal{N}\left(\mathbf{x_t} ; \sqrt{1-\beta_t} \mathbf{x_{t-1}}, \beta_t \mathbf{I}\right)$$

$$q\left(\mathbf{x}_{1: T} \mid \mathbf{x}_0\right)=\prod_{t=1}^T q\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}\right)$$

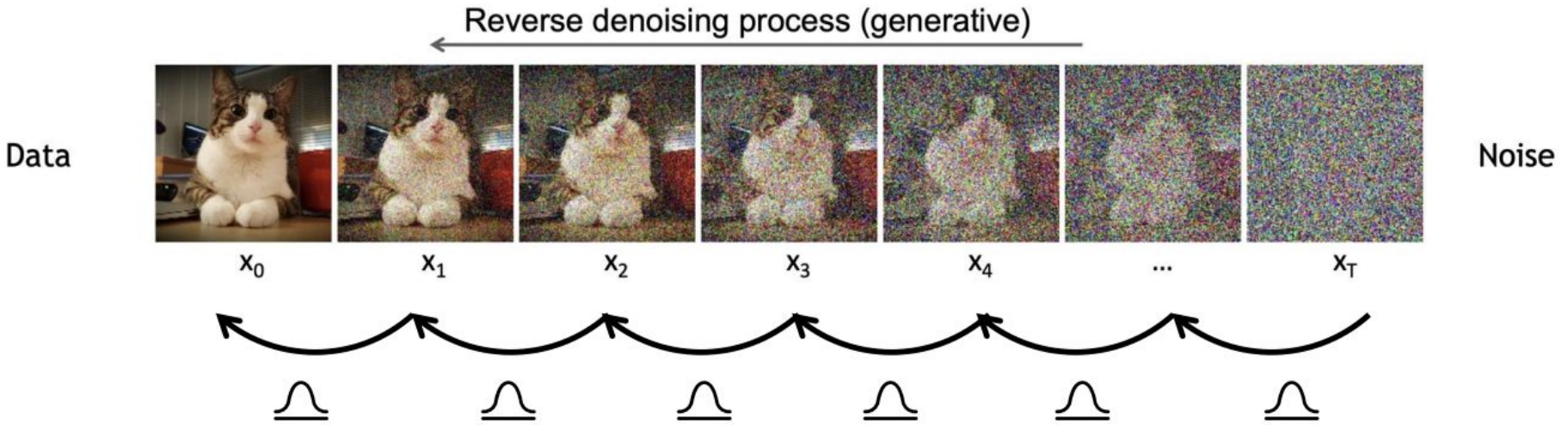


Diffusion Process (3)

- Reverse diffusion process is stacking **learnable** VAE decoders
  - Predicting the mean and std of added Gaussian Noise

$$p(\mathbf{x}_T) = \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I}) \qquad p_\theta(\mathbf{x}_{0:T}) = p(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$$

$$p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) = \mathcal{N}\left(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \sigma_t^2 \mathbf{I}\right)$$



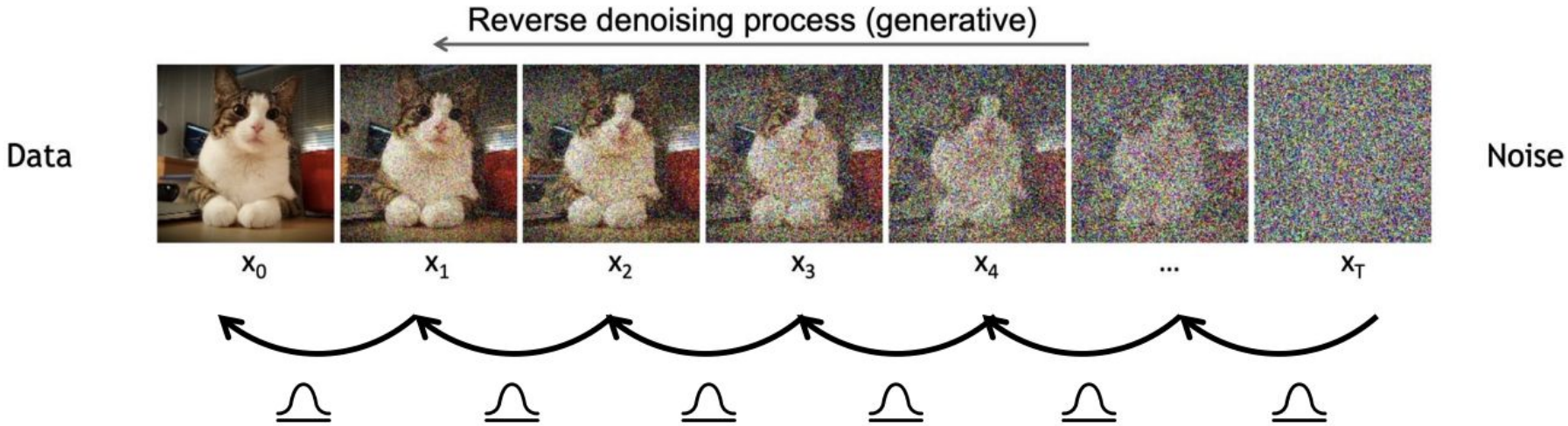
Diffusion Process (4)

- Reverse diffusion process is stacking **learnable** VAE decoders
  - Predicting the mean and std of added Gaussian Noise

$$p\left(\mathbf{x}_T\right) = \mathcal{N}\left(\mathbf{x}_T; \mathbf{0}, \mathbf{I}\right) \qquad p_\theta\left(\mathbf{x}_{0:T}\right) = p\left(\mathbf{x}_T\right) \prod_{t=1}^{T} p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right)$$

$$p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right) = \mathcal{N}\left(\mathbf{x}_{t-1}; \underbrace{\mu_\theta\left(\mathbf{x}_t, t\right), \sigma_t^2 \mathbf{I}}\right)$$

Trainable Network, Shared Across All Timesteps



Learning the Denoising Model (1)

- Denoising models are trained with variational upper bound (negative ELBO), as VAEs

$$\mathbb{E}_{q(\mathbf{x}_0)}\left[-\log p_\theta\left(\mathbf{x}_0\right)\right] \leq \mathbb{E}_{q(\mathbf{x}_0) q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)}\left[-\log \frac{p_\theta\left(\mathbf{x}_{0:T}\right)}{q\left(\mathbf{x}_{1:T} \mid \mathbf{x}_0\right)}\right] =: L$$

- which derives to:

$$L = \mathbb{E}_q[\underbrace{D_{\mathrm{KL}}\left(q\left(\mathbf{x}_T \mid \mathbf{x}_0\right) \| p\left(\mathbf{x}_T\right)\right)}_{L_T} + \sum_{t>1} \underbrace{D_{\mathrm{KL}}\left(q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0\right) \| p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right)\right)}_{L_{t-1}} \underbrace{-\log p_\theta\left(\mathbf{x}_0 \mid \mathbf{x}_1\right)}_{L_0}]$$

constant

Scaling

- tractable posterior distribution (closed-form)

$$q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0\right) = \mathcal{N}\left(\mathbf{x}_{t-1} ; \tilde{\mu}_t\left(\mathbf{x}_t, \mathbf{x}_0\right), \tilde{\beta}_t \mathbf{I}\right)$$

$$\text{where } \tilde{\mu}_t\left(\mathbf{x}_t, \mathbf{x}_0\right) := \frac{\sqrt{\bar{\alpha}_{t-1}} \beta_t}{1-\bar{\alpha}_t} \mathbf{x}_0 + \frac{\sqrt{1-\beta_t}\left(1-\bar{\alpha}_{t-1}\right)}{1-\bar{\alpha}_t} \mathbf{x}_t \text{ and } \tilde{\beta}_t := \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t} \beta_t$$

Learning the Denoising Model (2)

- Denoising models are trained with variational upper bound (negative ELBO), as VAEs

$$\mathbb{E}_{q(\mathbf{x}_0)}\left[-\log p_\theta\left(\mathbf{x}_0\right)\right] \leq \mathbb{E}_{q(\mathbf{x}_0) q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)}\left[-\log \frac{p_\theta\left(\mathbf{x}_{0:T}\right)}{q\left(\mathbf{x}_{1:T} \mid \mathbf{x}_0\right)}\right] =: L$$

- which derives to:

$$L = \mathbb{E}_q[\underbrace{D_{\mathrm{KL}}\left(q\left(\mathbf{x}_T \mid \mathbf{x}_0\right) \| p\left(\mathbf{x}_T\right)\right)}_{L_T} + \sum_{t>1} \underbrace{D_{\mathrm{KL}}\left(q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0\right) \| p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right)\right)}_{L_{t-1}} \underbrace{-\log p_\theta\left(\mathbf{x}_0 \mid \mathbf{x}_1\right)}_{L_0}]$$

- tractable posterior distribution (closed-form)

$$q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0\right) = \mathcal{N}\left(\mathbf{x}_{t-1} ; \tilde{\mu}_t\left(\mathbf{x}_t, \mathbf{x}_0\right), \tilde{\beta}_t \mathbf{I}\right)$$

$$\text{where } \tilde{\mu}_t\left(\mathbf{x}_t, \mathbf{x}_0\right) := \frac{\sqrt{\bar{\alpha}_{t-1}} \beta_t}{1-\bar{\alpha}_t} \mathbf{x}_0 + \frac{\sqrt{1-\beta_t}\left(1-\bar{\alpha}_{t-1}\right)}{1-\bar{\alpha}_t} \mathbf{x}_t \text{ and } \tilde{\beta}_t := \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t} \beta_t$$

Parameterizing the Denoising Model

- KL divergence has a simple form between Gaussians

$$L_{t-1} = D_{\mathrm{KL}}\left(q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0\right) \| p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right)\right) = \mathbb{E}_q\left[\frac{1}{2\sigma_t^2}\left\|\tilde{\mu}_t\left(\mathbf{x}_t, \mathbf{x}_0\right) - \mu_\theta\left(\mathbf{x}_t, t\right)\right\|^2\right] + C$$

- Recall that: $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{(1-\bar{\alpha}_t)}\epsilon$

$$\tilde{\mu}_t\left(\mathbf{x}_t, \mathbf{x}_0\right) = \frac{1}{\sqrt{1-\beta_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon\right)$$

- Trainable network predicts the noise mean

$$\mu_\theta\left(\mathbf{x}_t, t\right) = \frac{1}{\sqrt{1-\beta_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\boxed{\epsilon_\theta\left(\mathbf{x}_t, t\right)}\right)$$

Simplified Training Objective

$$L_{t-1} = \mathbb{E}_{\mathbf{x}_0 \sim q(\mathbf{x}_0), \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})} \left[\underbrace{\frac{\beta_t^2}{2\sigma_t^2 \left(1 - \beta_t\right)\left(1 - \bar{\alpha}_t\right)}}_{\lambda_t} \left\| \epsilon - \epsilon_\theta \left(\sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon, t \right) \right\|^2 \right]$$

- λ_t ensures the weighting for correct maximum likelihood estimation

- In DDPM, this is further simplified to:

$$L_{\text{simple}} = \mathbb{E}_{\mathbf{x}_0 \sim q(\mathbf{x}_0), \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), t \sim \mathcal{U}(1, T)} [\| \epsilon - \epsilon_\theta (\underbrace{\sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon}_{\mathbf{x}_t}, t) \|^2]$$

Summary: Training and Sampling

Algorithm 1 Training

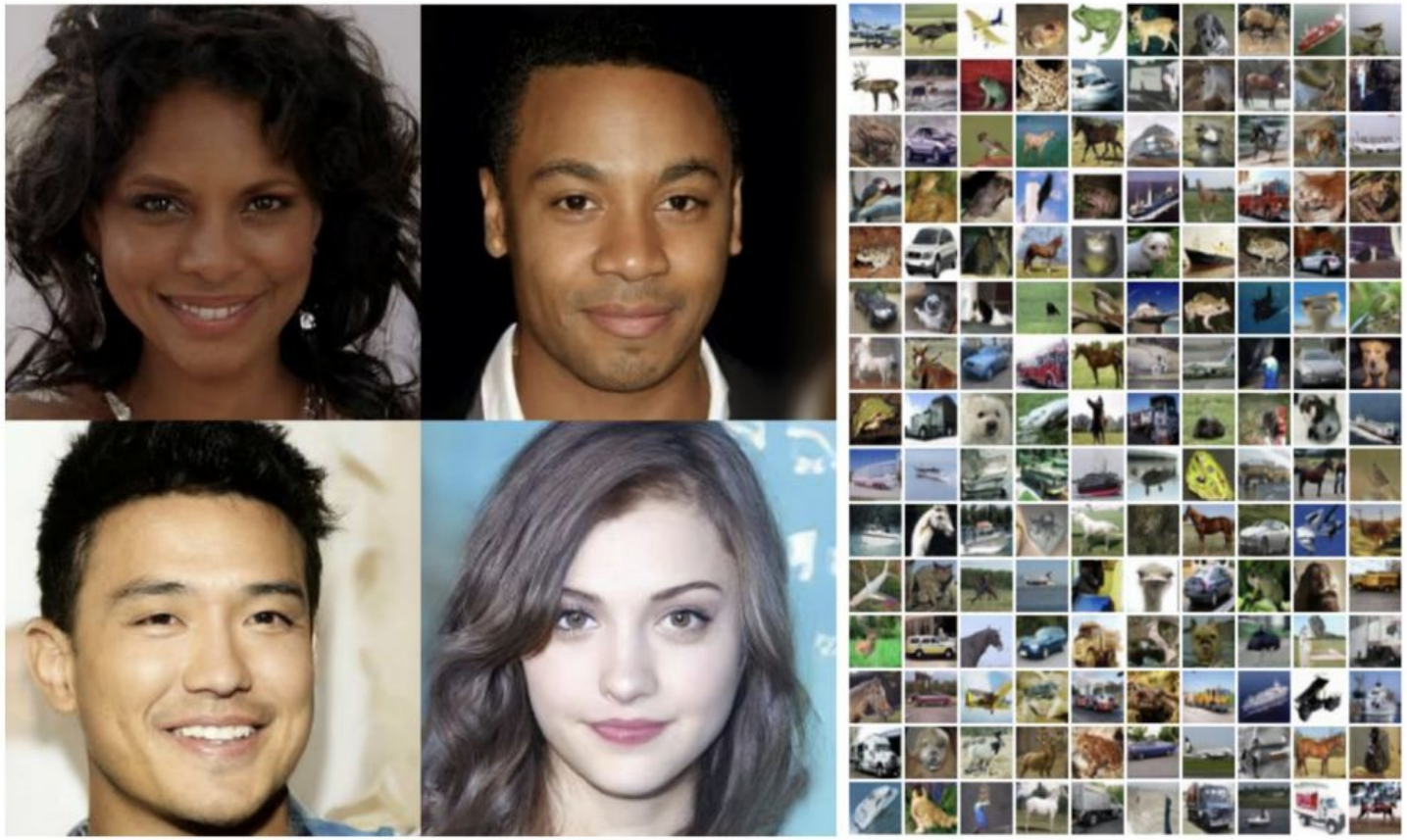
1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \dots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0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

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \dots, 1$ **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** $\mathbf{x}_0$

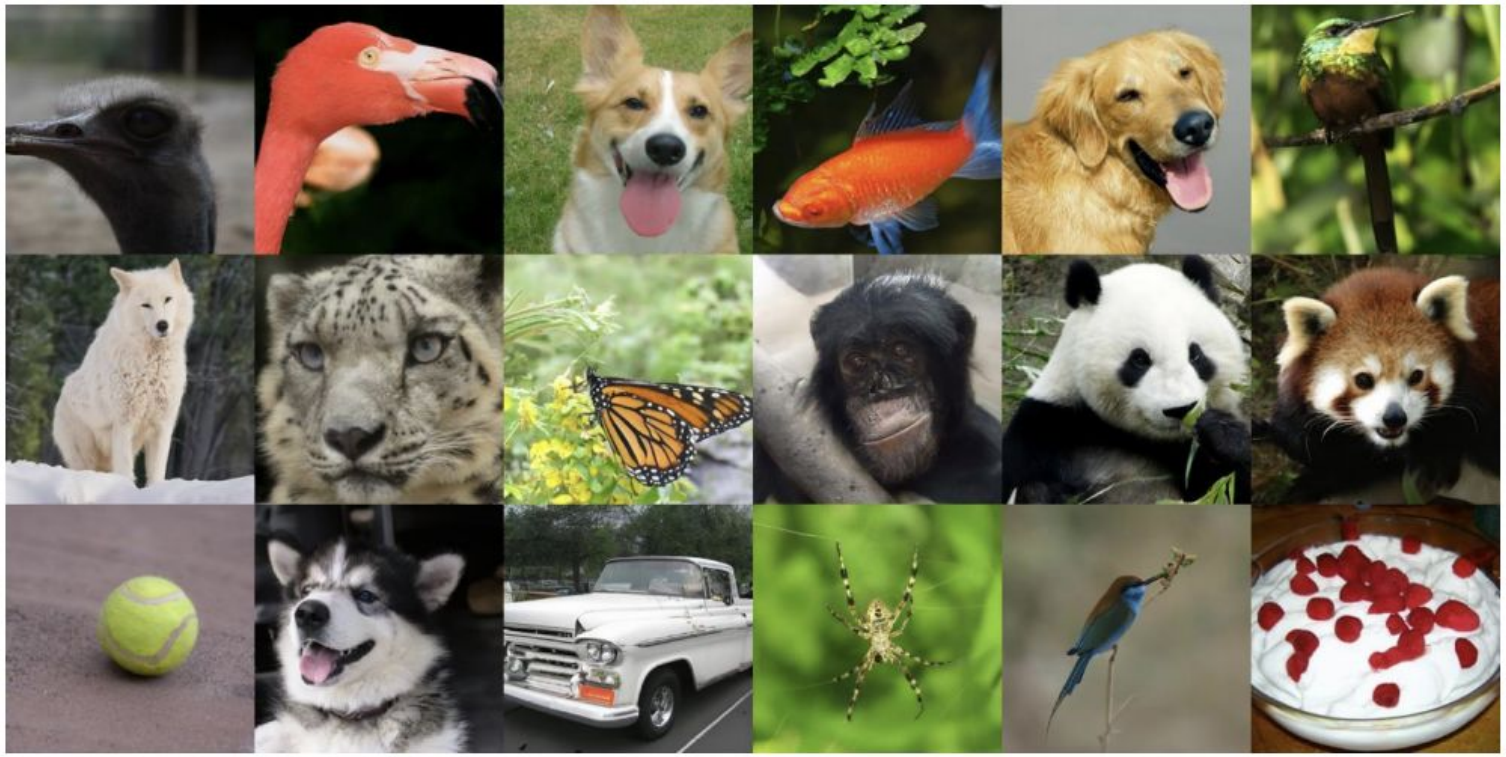
Denoising Diffusion Probabilistic Models (DDPM)

- Training diffusion models on raw images with a U-Net model



Diffusion Models Beat GANs

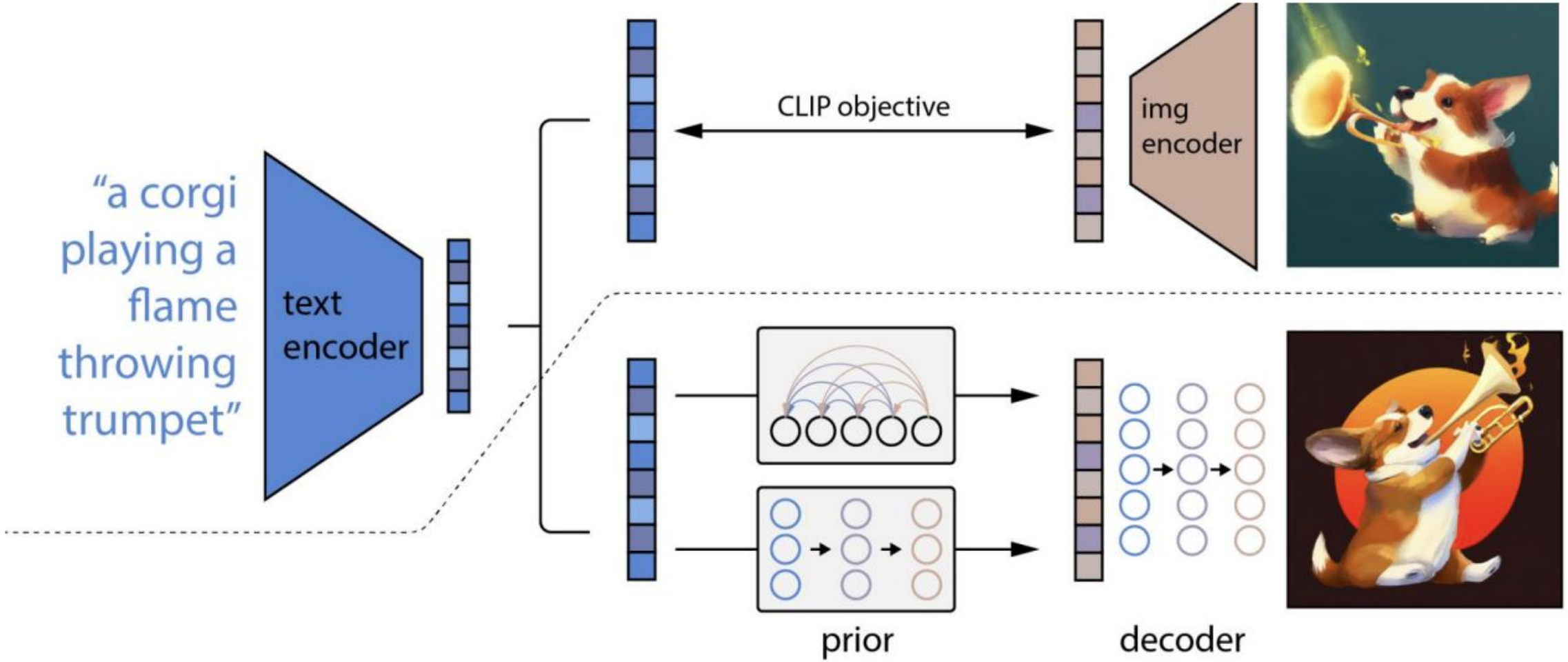
- Larger denoising model with sophisticated design
  - Adaptive group normalization
  - Attention layers in U-Net



Stable Diffusion

- Large-scale text-conditional LDMs
  - With VAEs trained also on larger datasets

DALL-E/DALL-E-2



DALL-E-3

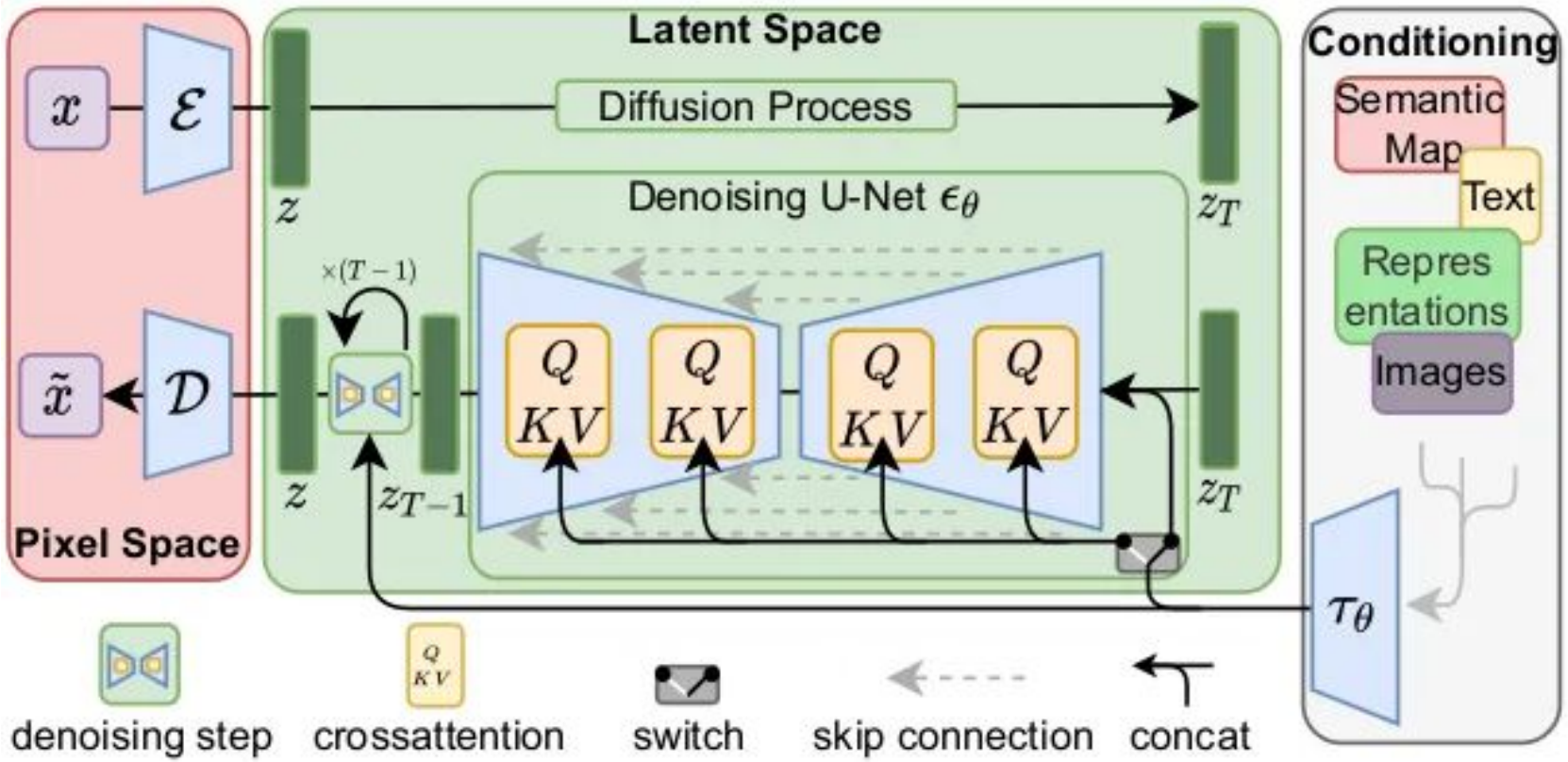
DALL·E 3 makes notable improvements over DALL·E 2,
even when given the same prompt.

DALL·E 2 · An expressive oil painting of a basketball player dunking, depicted as an explosion of a nebula.

DALL·E 3 · An expressive oil painting of a basketball player dunking, depicted as an explosion of a nebula.

Diffusion Transformer

Using diffusion transformer for video generation.



Diffusion Language Model?

Yes, we have