

Deep Learning Bootcamp

with PyTorch

Noel Shin

Contents

Part I. Classification Model

1. Multilayer perceptron
2. Convolutional neural networks
3. Residual networks
4. Densely connected convolutional networks
5. Attention modules - squeeze and excitation networks
6. Attention modules - gather-excite
7. Attention modules - convolutional block attention module

Contents

Part II. Time Series Data

8. Recurrent neural networks

9. Long short-term memory

10. seq2seq

11. seq2seq with attention

Contents

Part III. Generative Model

12. Generative adversarial networks
13. Deep convolutional adversarial networks
14. Conditional generative adversarial networks
15. Image-to-image translation with conditional adversarial networks
16. Unpaired image-to-image translation using cycle-consistent adversarial networks

Contents

Appendix

A. Python Class

Introduction

Goal of lecture

Getting used to

- read a PyTorch code of interest,**
- make a deep learning model using PyTorch.**

Prerequisite

**You need to have overall understanding of ‘class’ in Python.
If you are not familiar with class, you may refer to Appendix A.**

Class

- Let's make a basic class in Python.

What is PyTorch?

PyTorch is an **open-source machine learning library** for Python, based on **Torch**.

- **Tensor computation with strong GPU acceleration***
- Deep neural networks built on a tape-based **autograd system**

Why PyTorch?



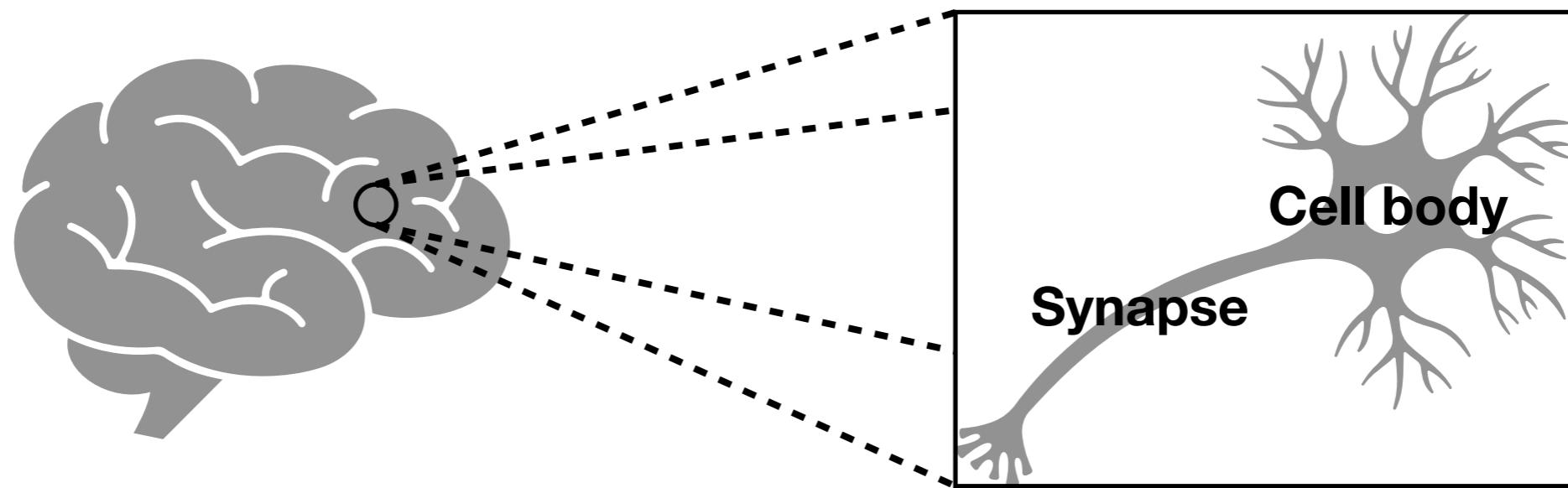
Keras



- A lot easier to learn than to learn TensorFlow.
- Available to customize a model. This is usually infeasible with Keras.
- Recently public codes for deep learning research papers are usually written with PyTorch.

Basics

What is deep learning?



Brain

Neuron

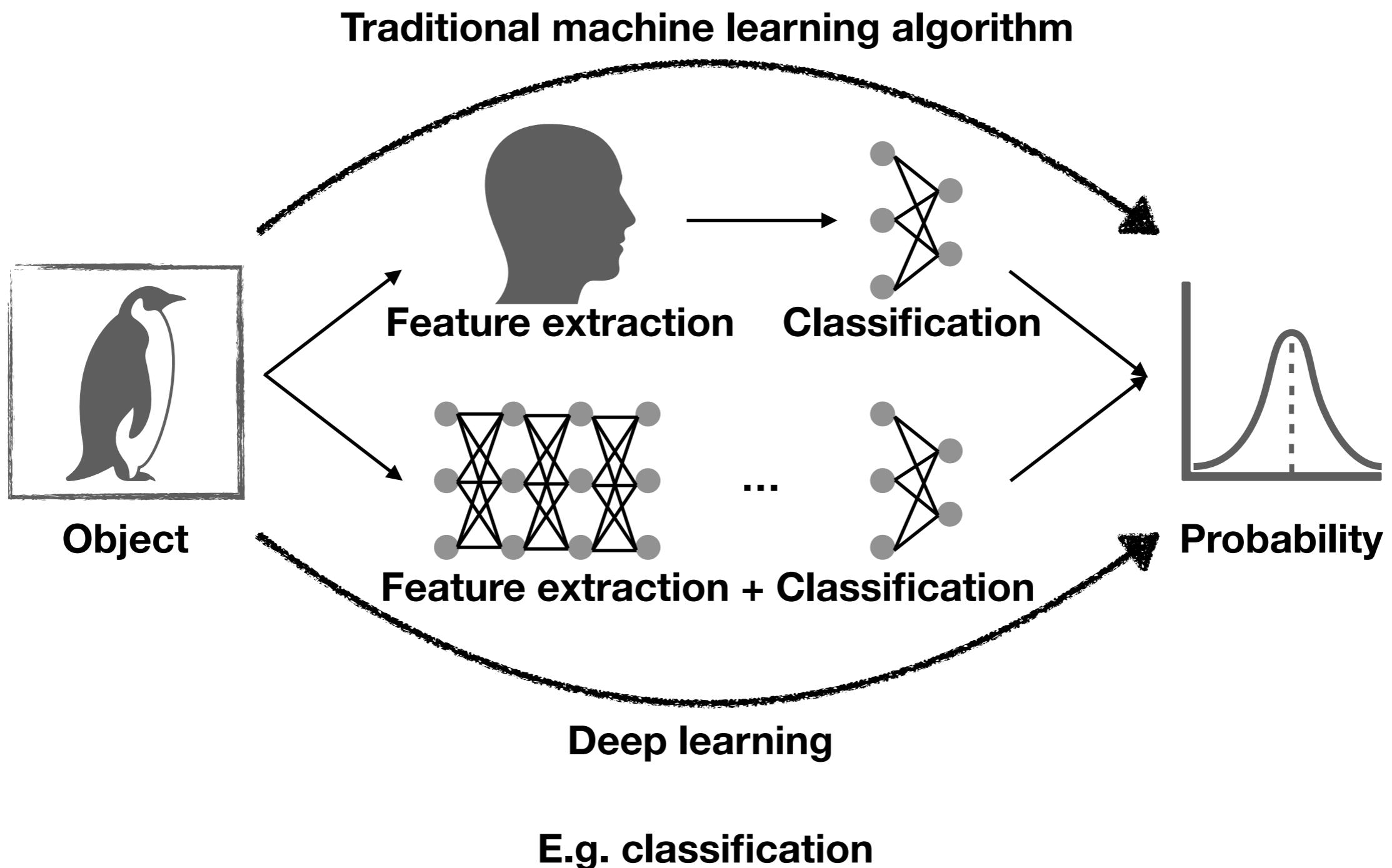


Model

Neural Networks

● : Node
— : Weight

What is deep learning?



How many buttons?



**Machine learning
researcher:**

“Find things having a circular shape and three holes.”

Deep learning researcher:

“I don’t care how you count them. Anyway the answer is four. Train yourself why the answer is four.”

How many buttons?



Machine learning model:

“Okay. Although they are circular, as they don’t have three holes, they are not button. The answer is 0.”

Deep learning model:

“Hmm... they don’t exactly match what I saw previously, somehow they look like ‘button’. Maybe the answer is 5?”

Part I

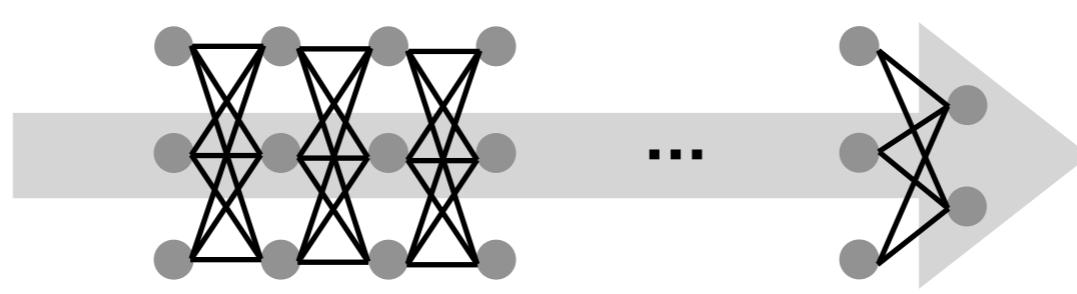
Classification Model

1. Multilayer perceptron

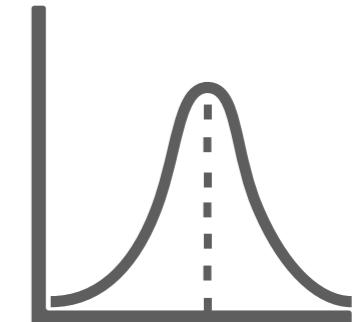
How does it work?



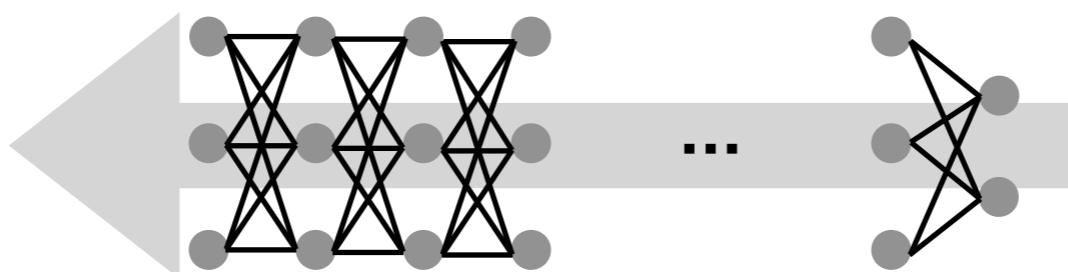
Input



Forward path



Penguin: 0.01



Backward path
(Gradient Back-propagation)

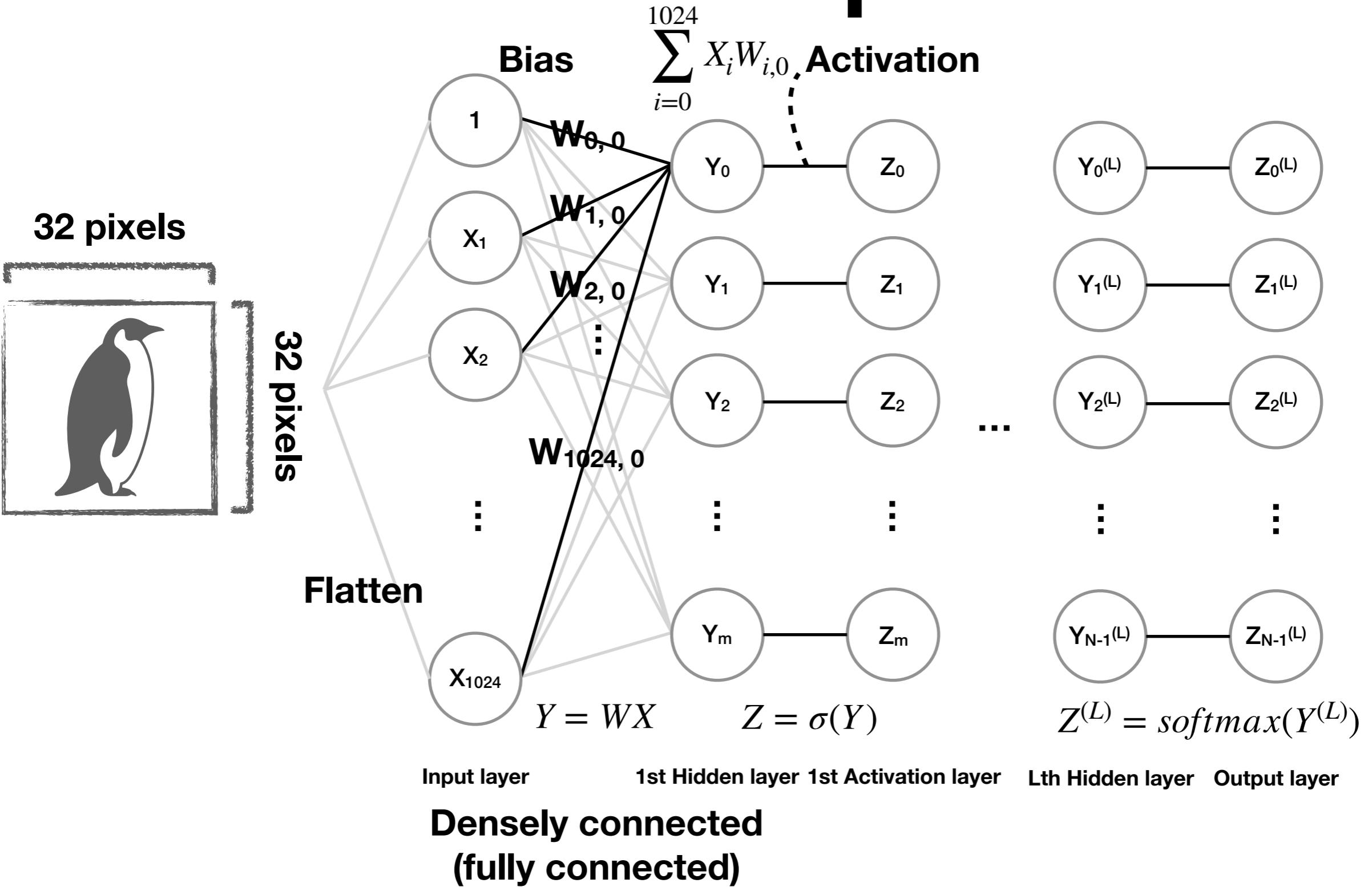
$$\nabla_{\theta} L$$

Penguin: 1.0

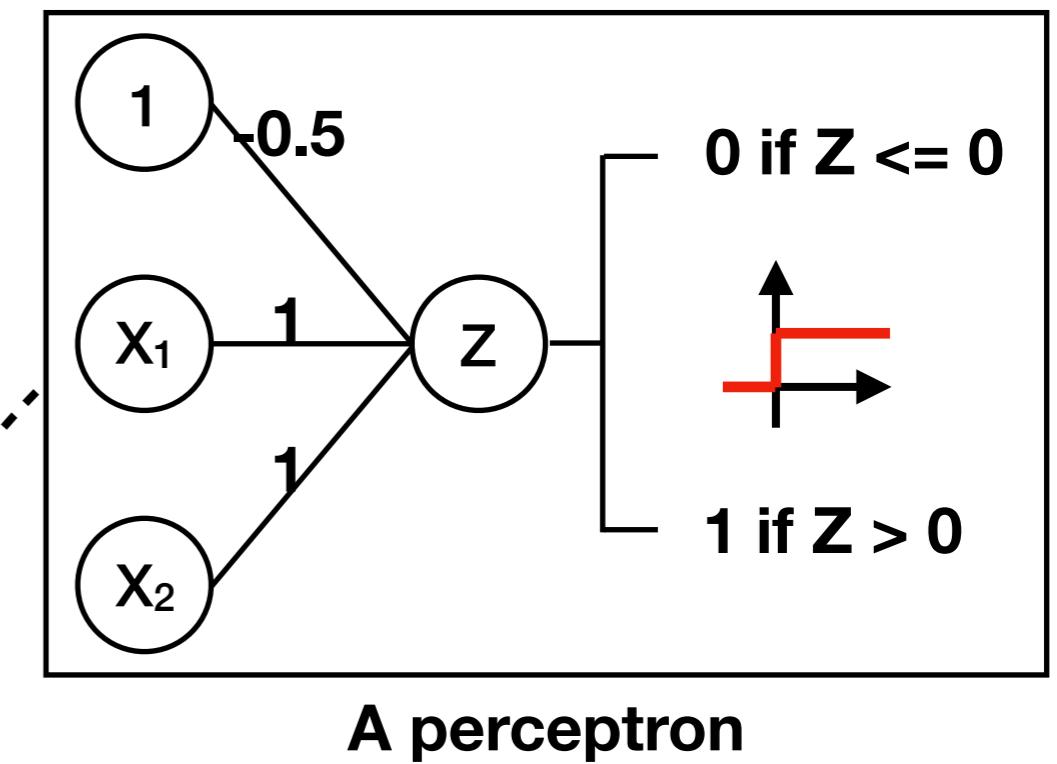
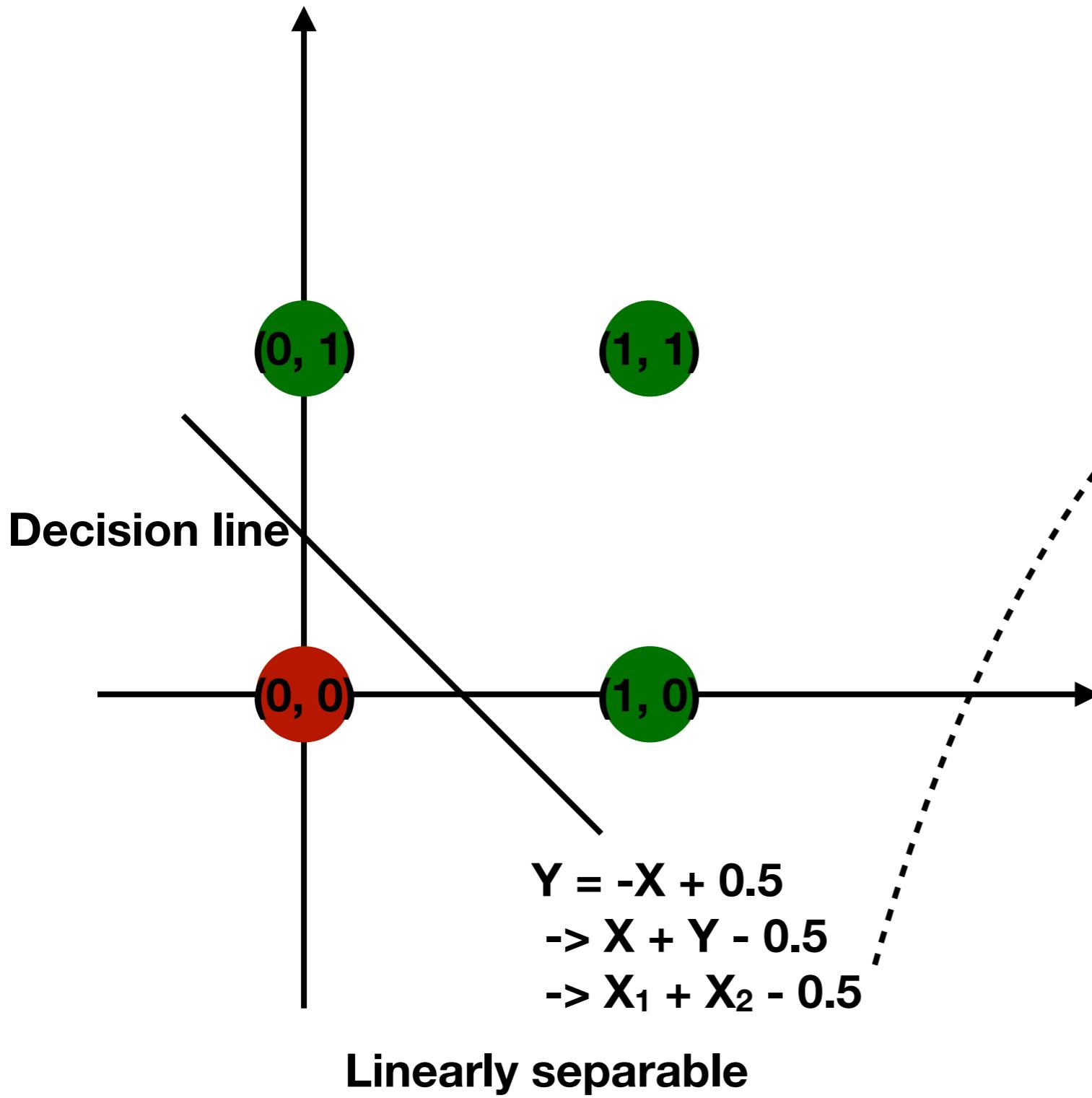
Difference
measurement
(Loss)

Forward path

Forward path



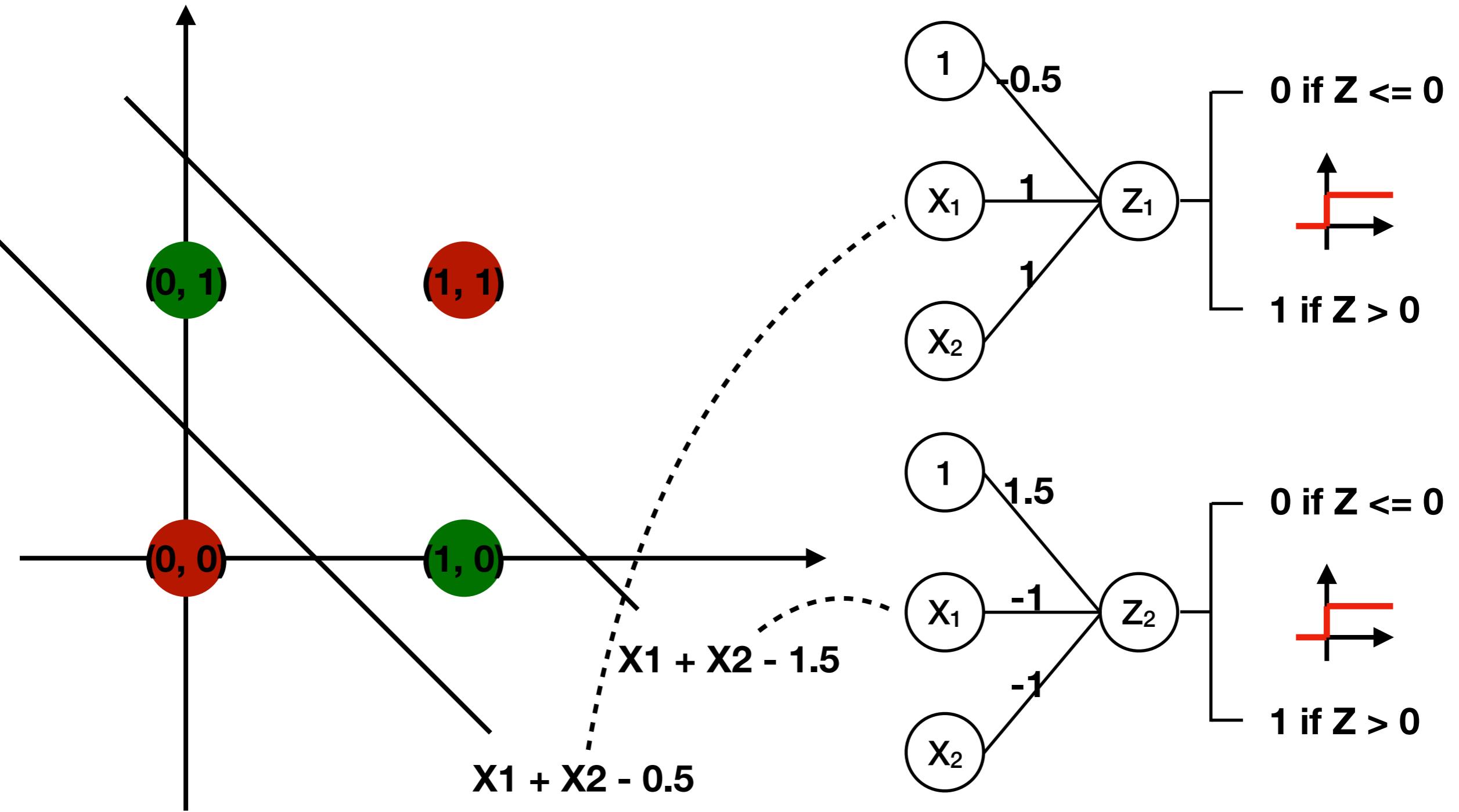
Forward path



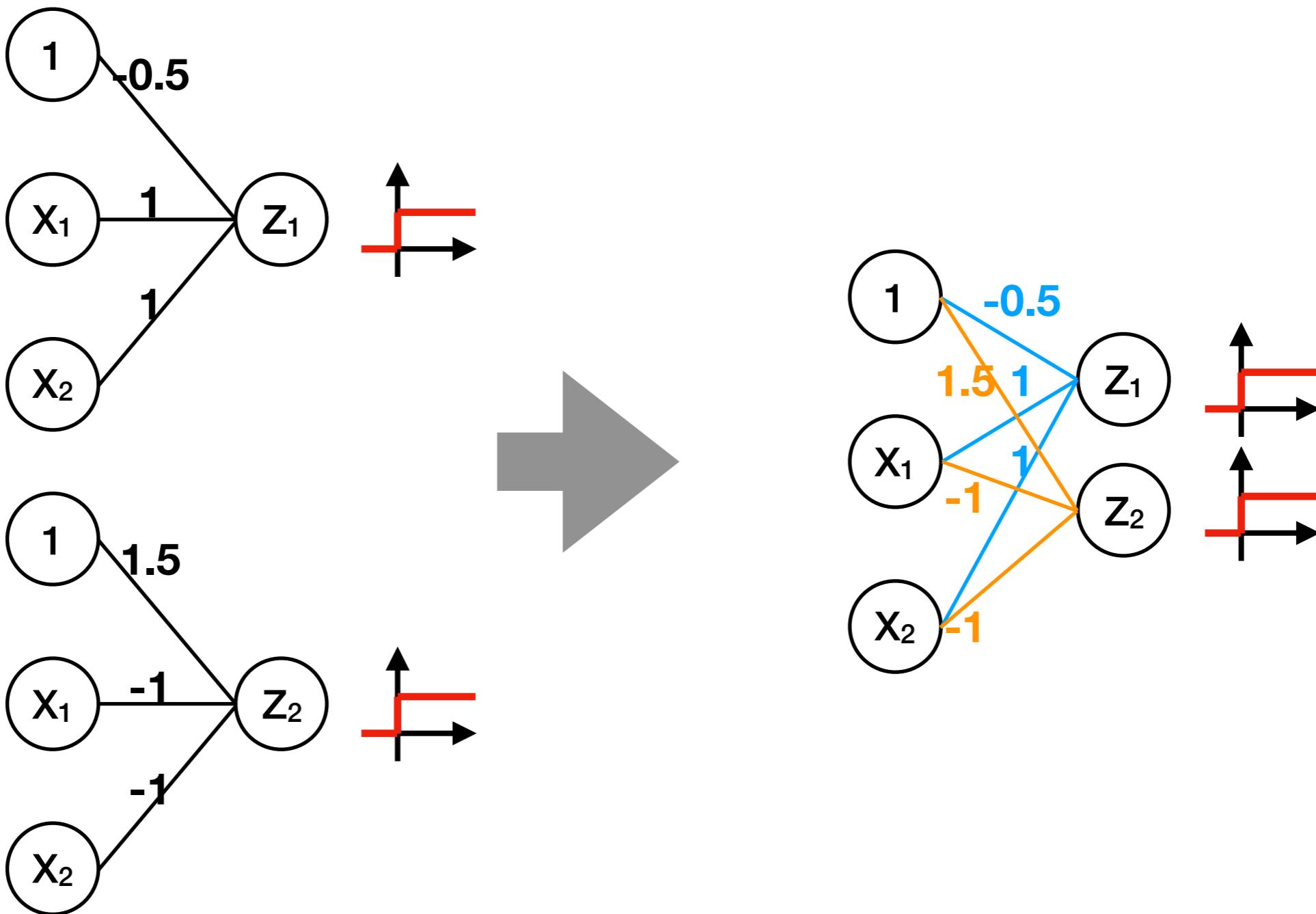
$(0, 0) \rightarrow -0.5 \rightarrow 0$
 $(0, 1) \rightarrow 0.5 \rightarrow 1$
 $(1, 0) \rightarrow 0.5 \rightarrow 1$
 $(1, 1) \rightarrow 1.5 \rightarrow 1$

Red circle : 0
Green circle : 1

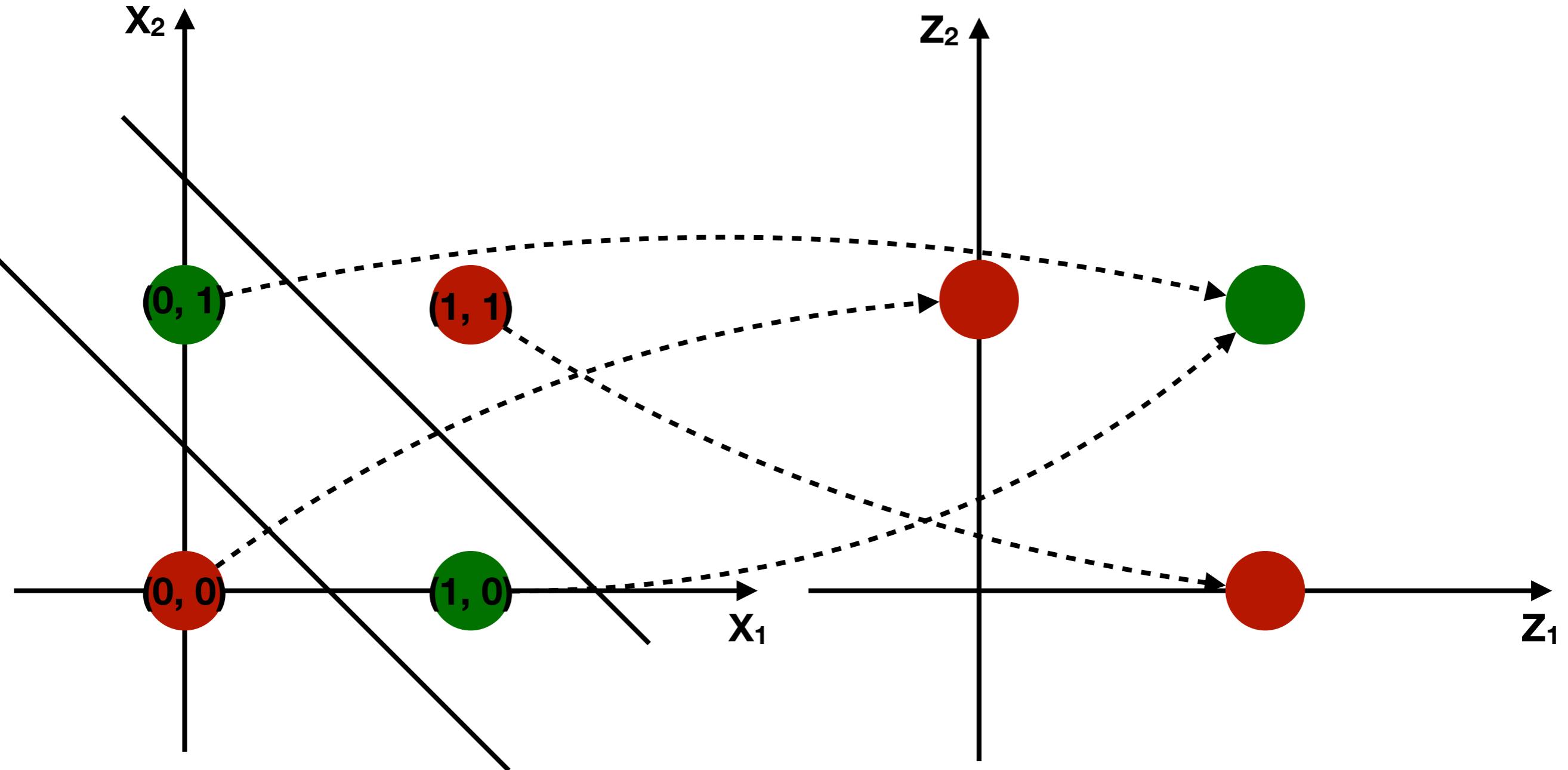
Forward path



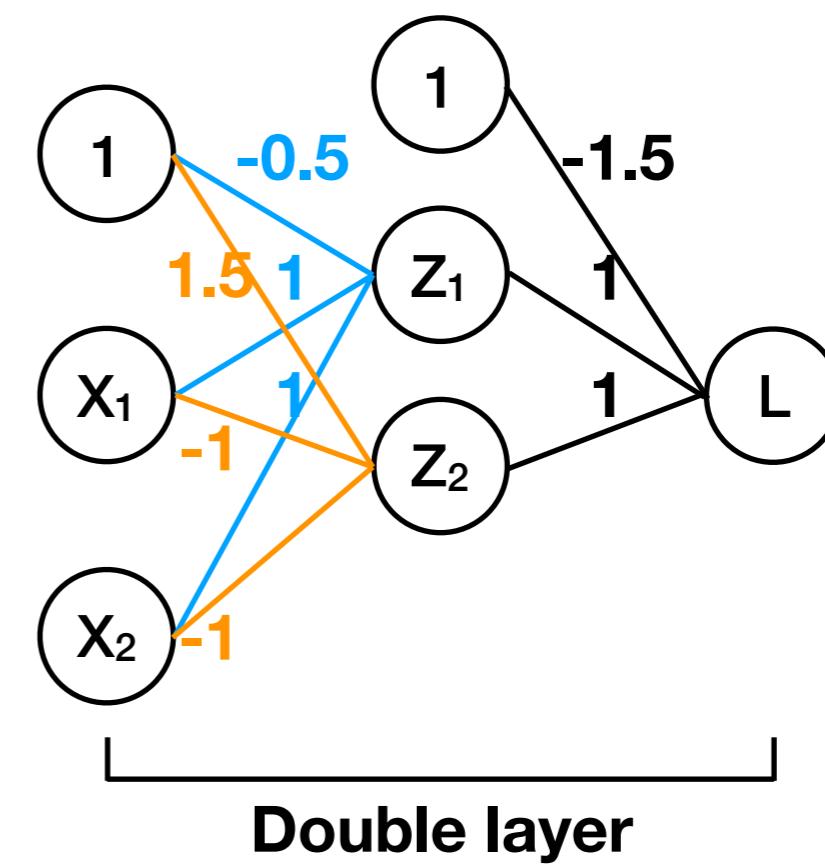
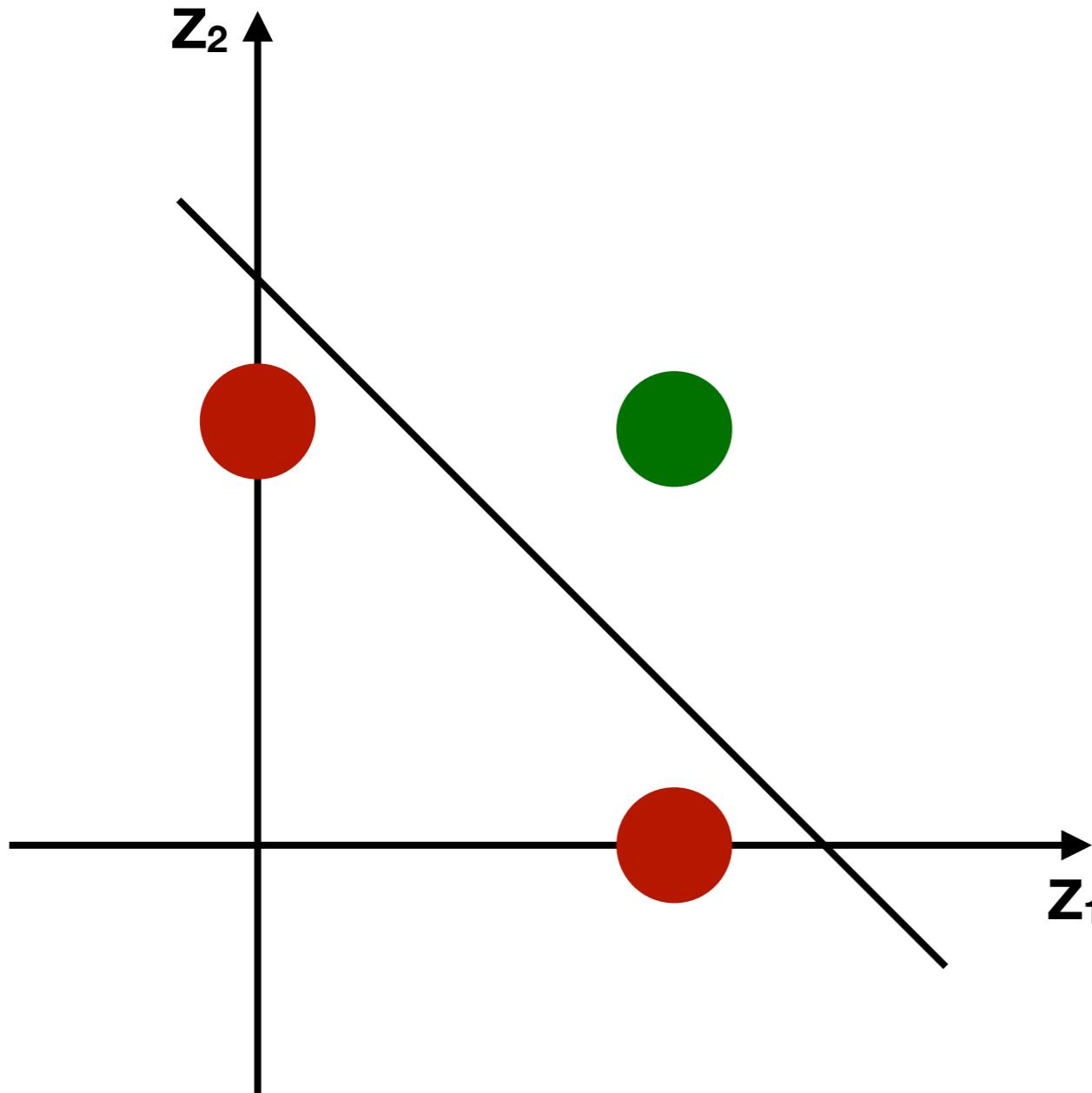
Forward path



Forward path



Forward path



Forward path

- Why do we need an activation function?

Without any activation function, it is infeasible to approximate higher degree function.

$$Y_1 = W X_1 + B$$

$$Y_2 = W(W X_1 + B) + B = W' X_1 + B' = W X_1 + B$$

$$Y_1 = W X_1 + B$$

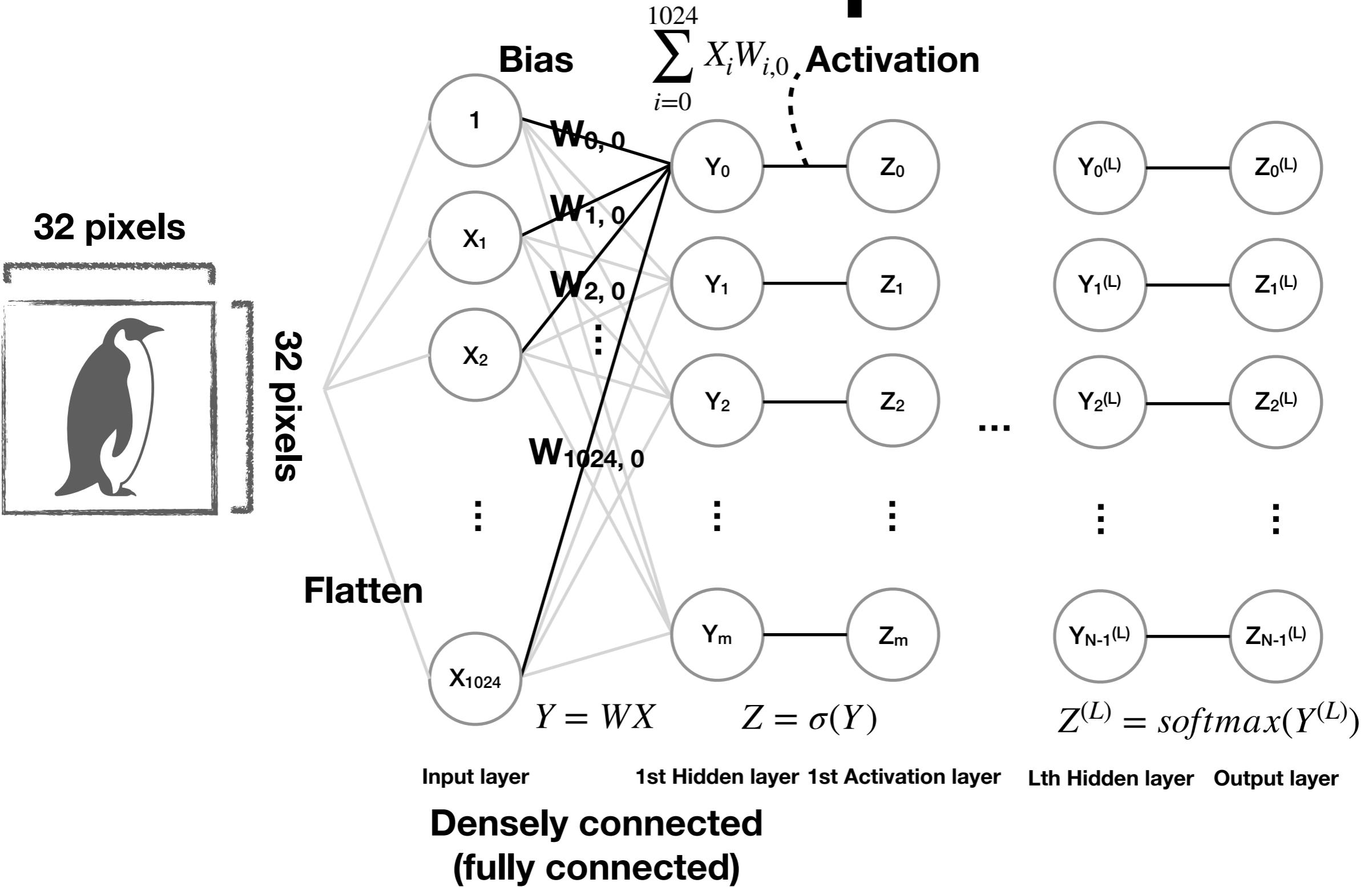
$$Z_1 = f(Y_1)$$

$$Y_2 = W Z_1 + B$$

where $f(\cdot)$ is non-linear activation function

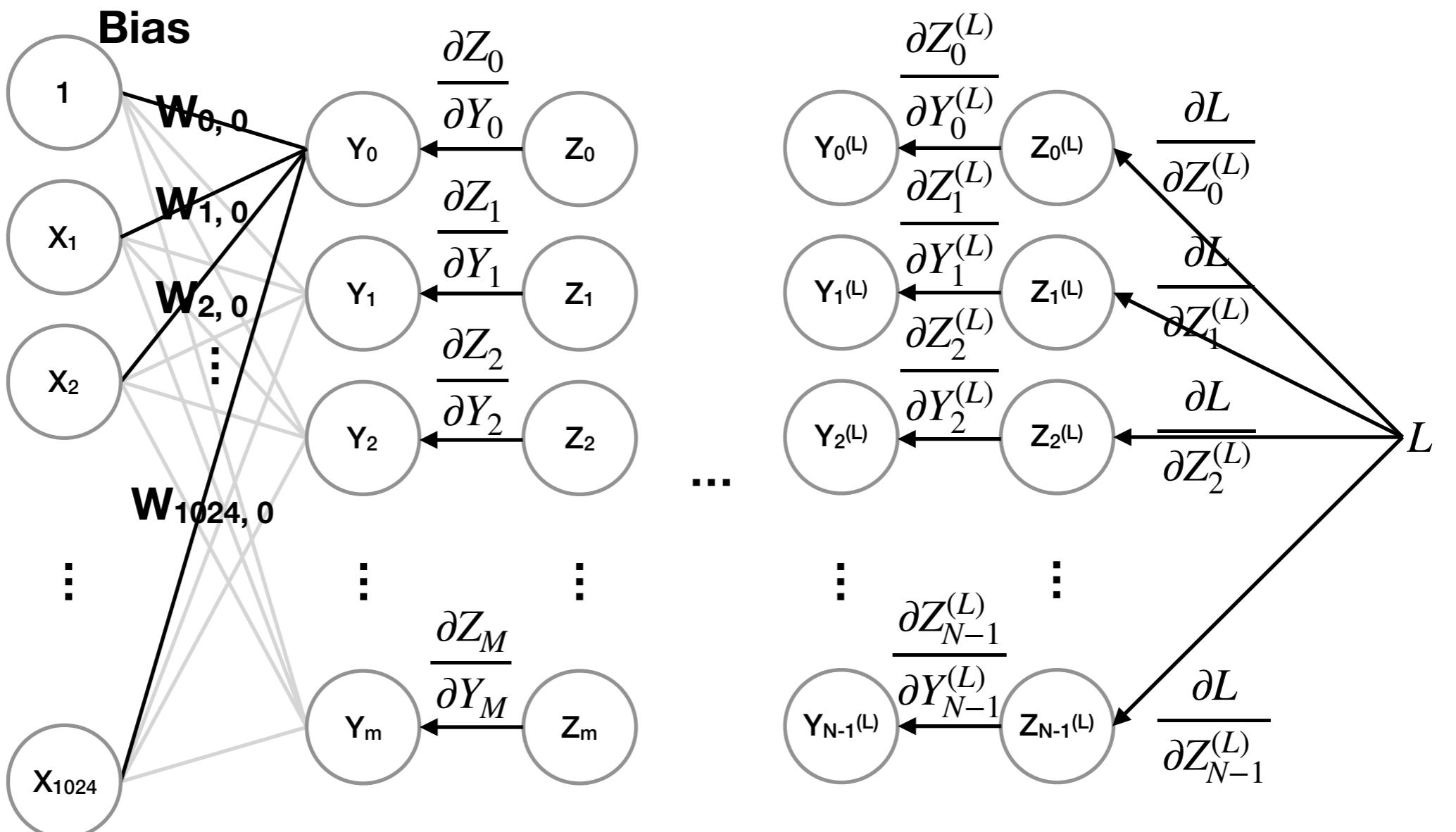
Thus we add an activation function after a weight layer.

Forward path



Backward path

Backward path



$$\frac{\partial L}{\partial W_{0,0}} = \frac{\partial L}{\partial Y_0} \frac{\partial Y_0}{\partial W_{0,0}}$$

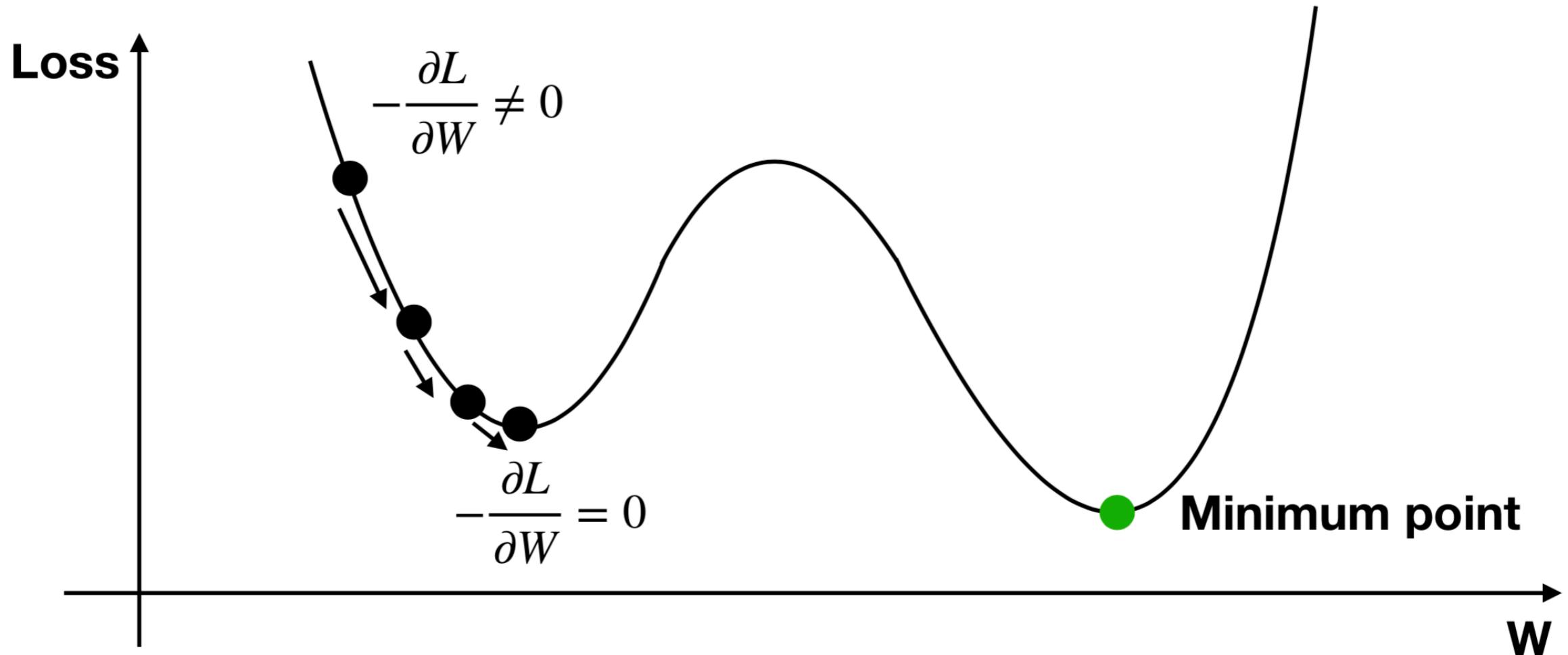
$$W_{0,0} = W_{0,0} - \rho \frac{\partial L}{\partial W_{0,0}}$$

$$L = - \sum_{k=0}^{N-1} t_k \log Z_k = - \log Z_i$$

Cross-entropy loss

Backward path

- Gradient back-propagation updates(changes) weights to have lower loss in a way loss decreases most fast.
- However, this does not guarantee the loss converges to its minimum value.



Practice

MNIST database

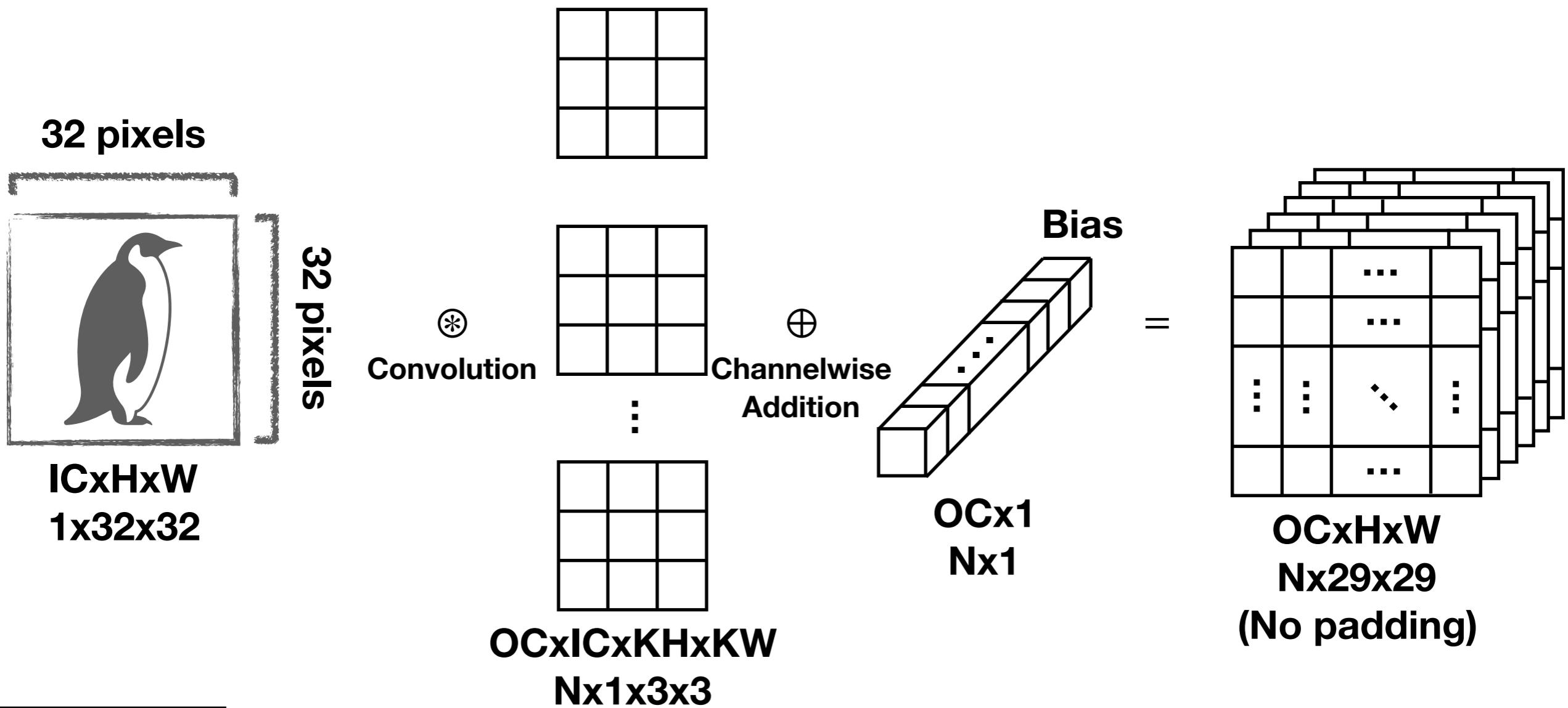


Training set: 60,000 images and labels
Test set: 10,000 images and labels

The **MNIST database (Modified National Institute of Standards and Technology database)** is a large database of handwritten digits that is commonly used for training various image processing systems.

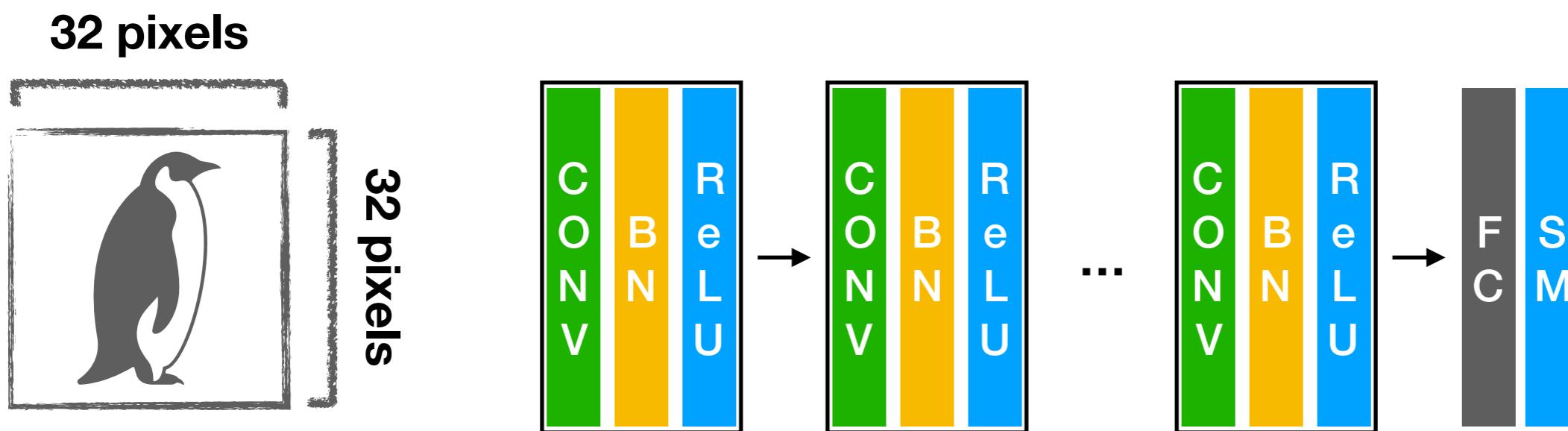
2. Convolutional neural networks

Convolution*



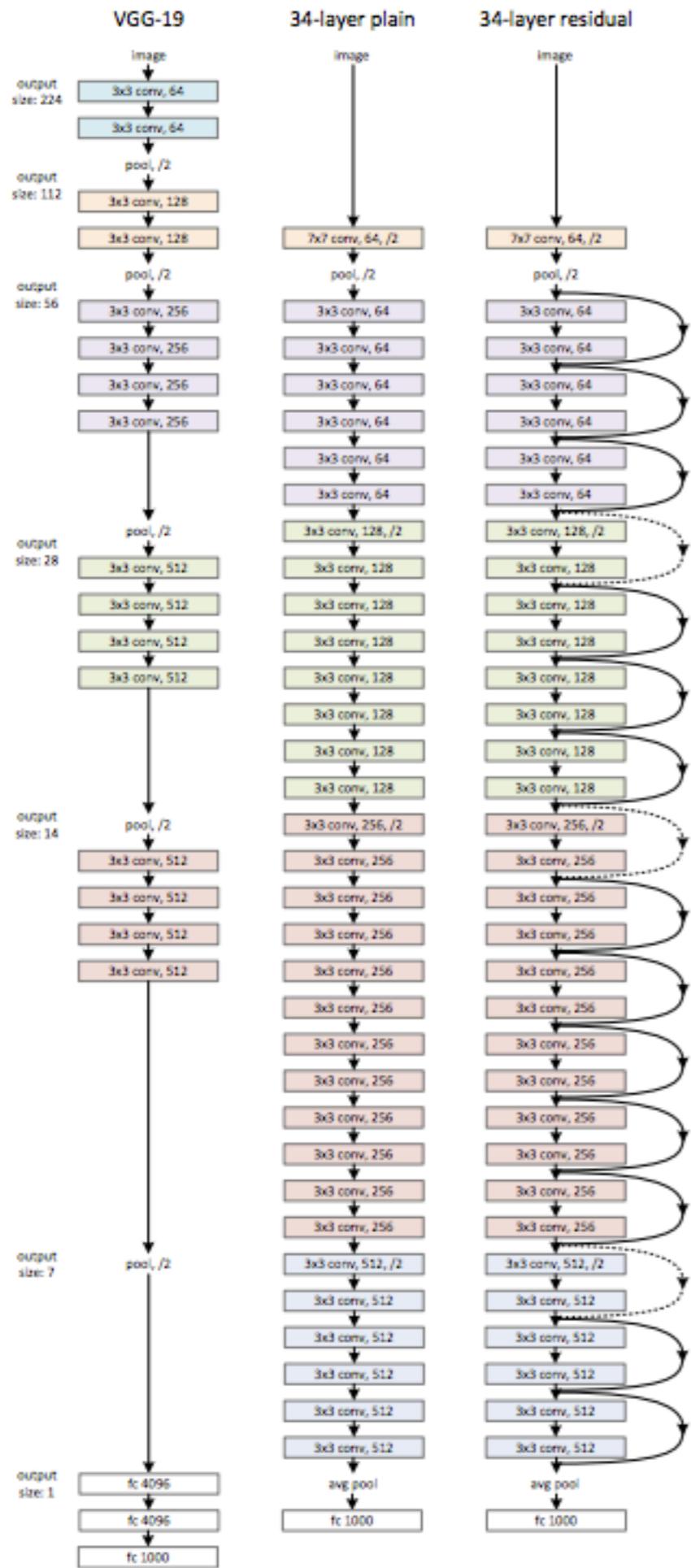
H: Height
W: Width
OC: Output Channel
IC: Input Channel
KH: Kernel Height
KW: Kernel Width

Convolutional neural network

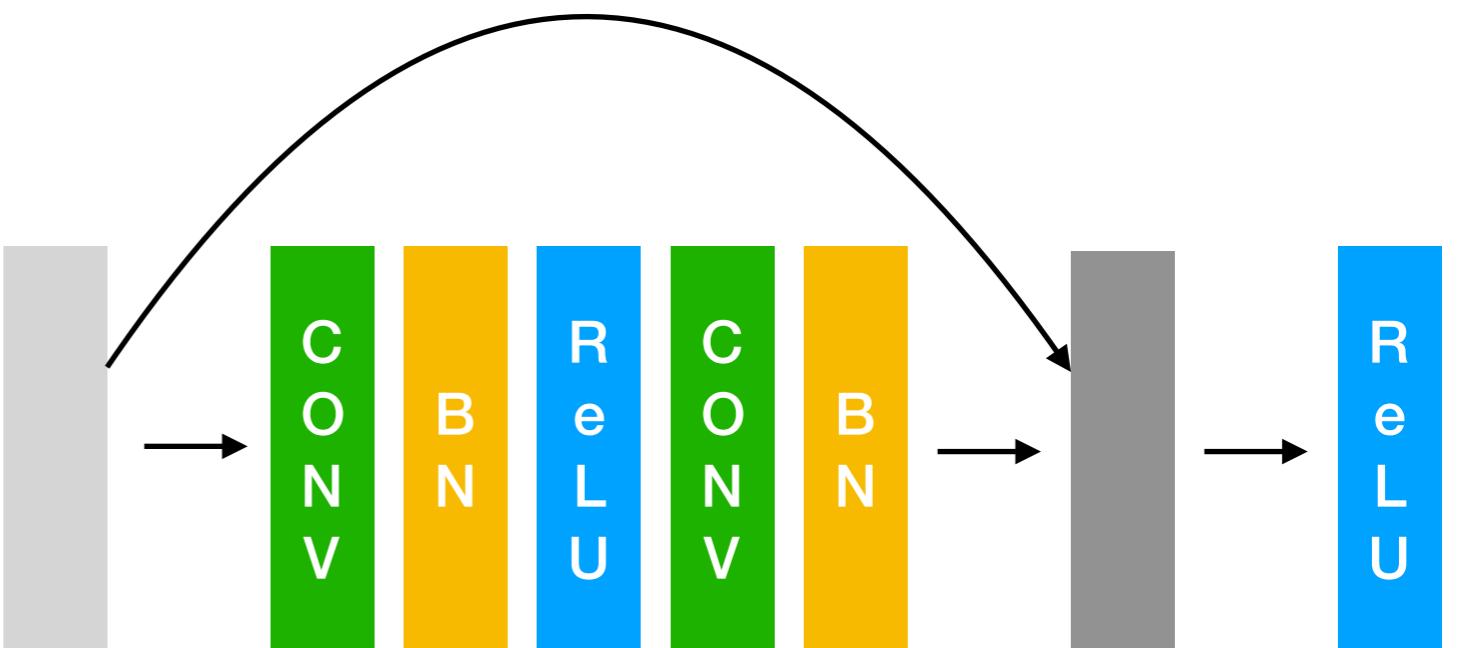


BN: Batch Normalization
CONV: CONVolution
FC: Fully Connected layer
ReLU: Rectified Linear Unit
SM: SoftMax activation layer

3. Residual Networks



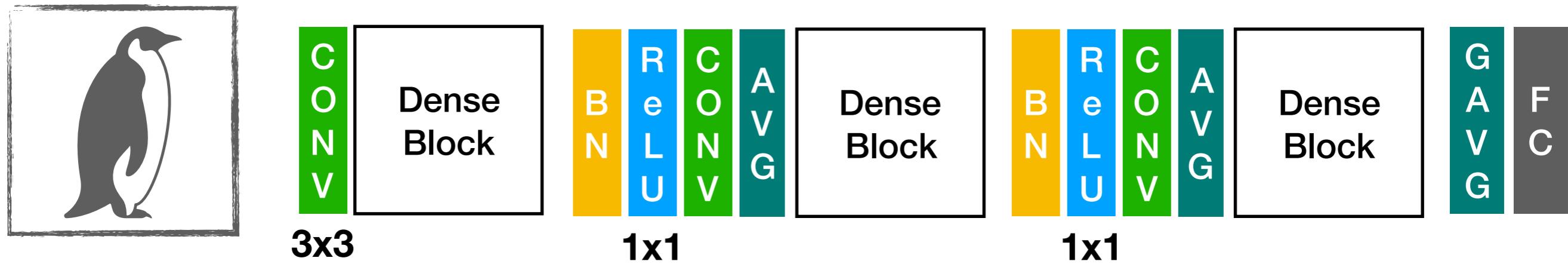
Element-wise summation



When feature map size changes, 1x1 convolution with stride 2 is applied such that the shortcut connection can be presented every two convolution layers.

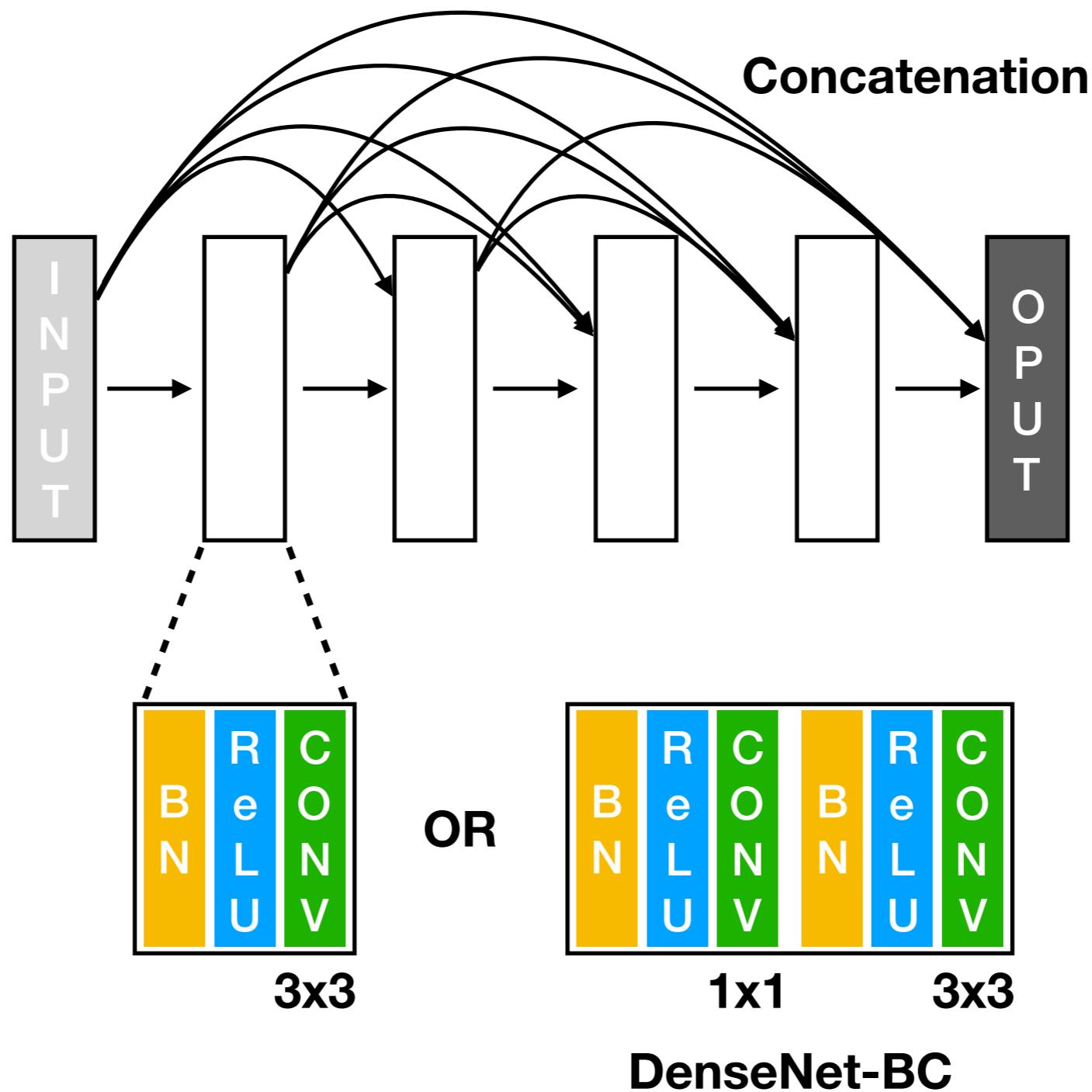
4. Dense Networks

Dense Network



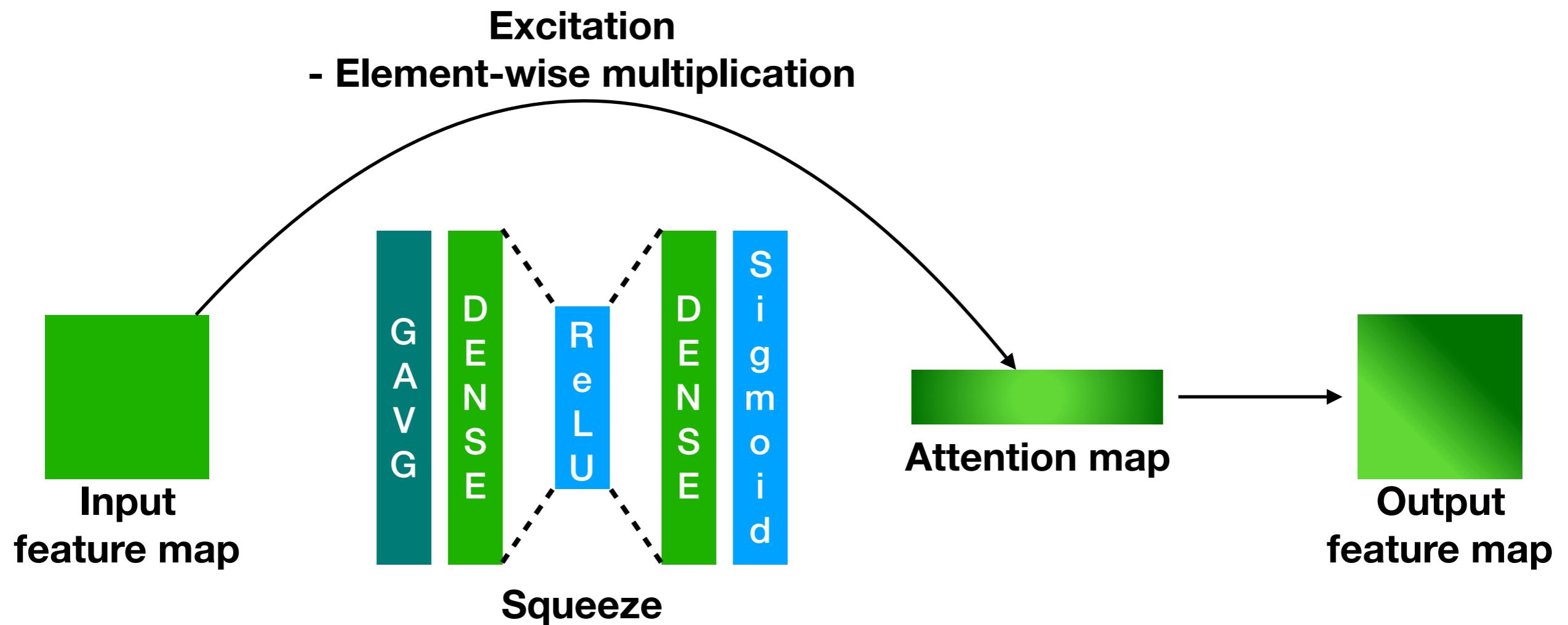
Avg: AVerage pooling
BN: BatchNormalization layer
Conv: Convolutional layer
FC: Fully-Connected layer
GAvg: Global AVerage pooling
ReLU: Rectified Linear Unit

Dense Block



5. Squeeze-and- Excitemen^t Networks

Squeeze-and-Excitation Block



DENSE: Fully-connected layer
GAVG: Global AVerage pooling
ReLU: Rectified Linear Unit

Part II

Time Series Data

Part III

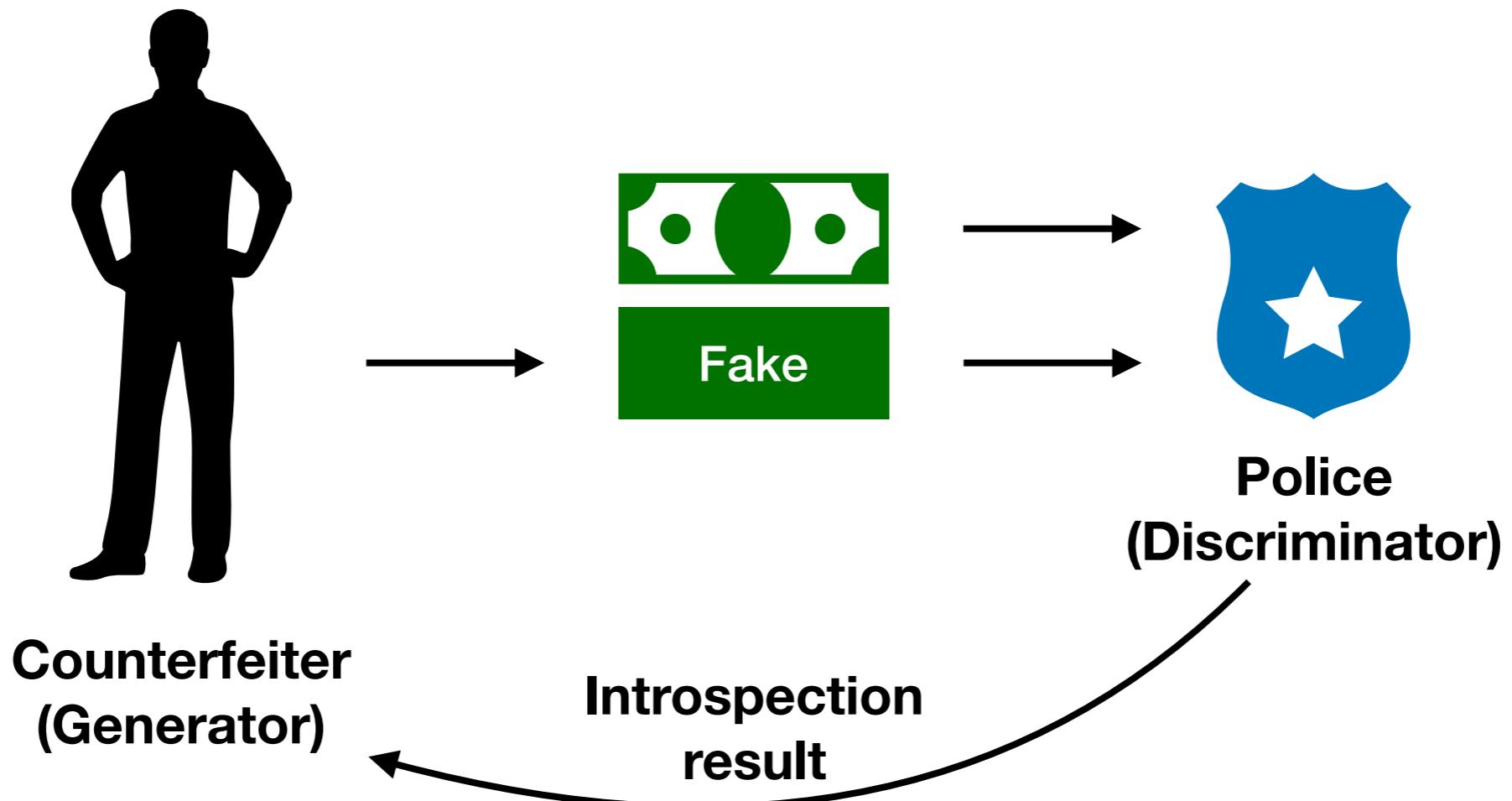
Generative Model

Various generative models

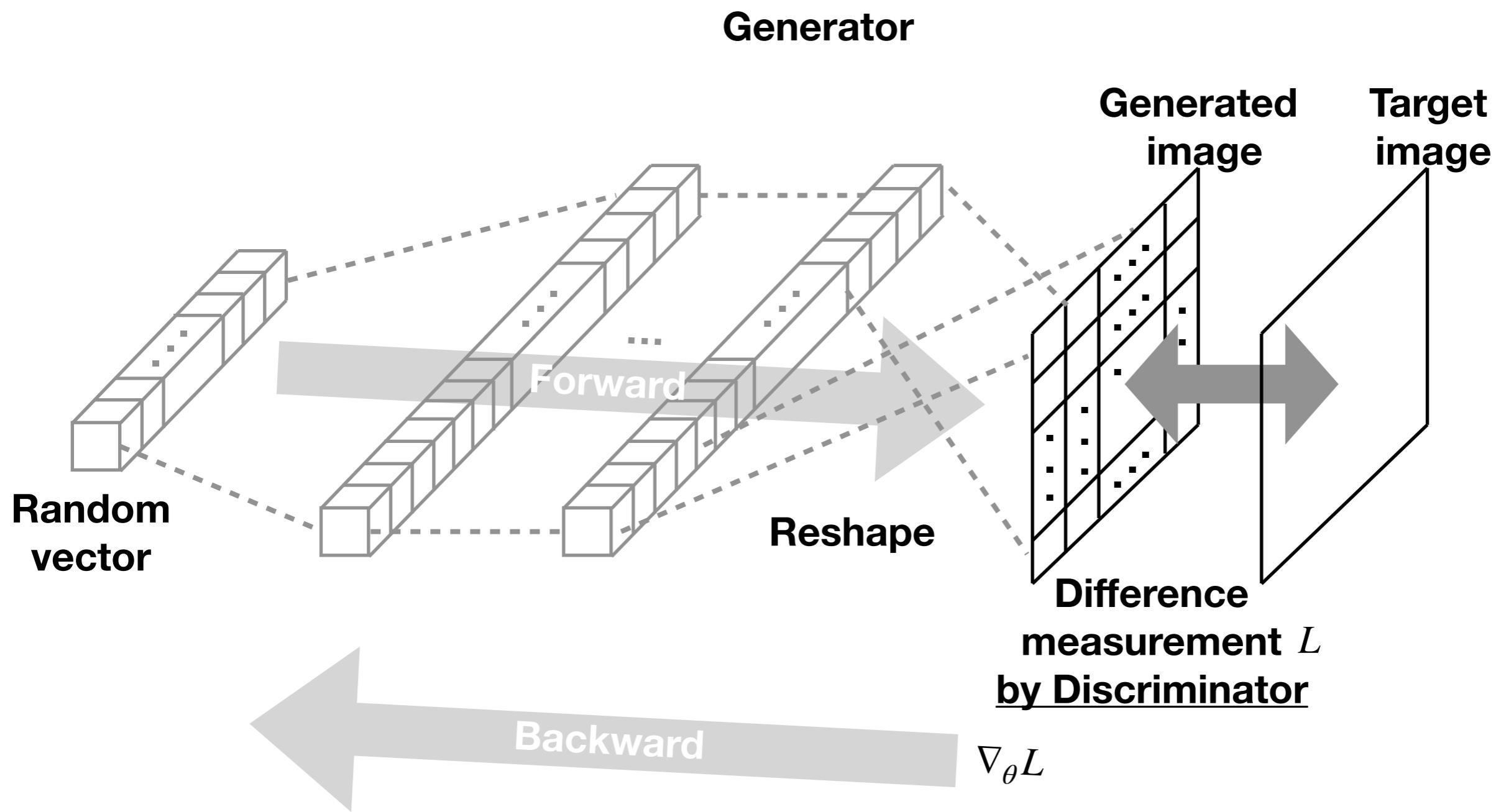
- Hidden Markov Model (HMM)
- Restricted Boltzmann Machine (RBM)
- Variational Auto-Encoder (VAE)
- Recurrent Neural Network (RNN)
- **Generative Adversarial Network (GAN)**

GAN

What is GAN?

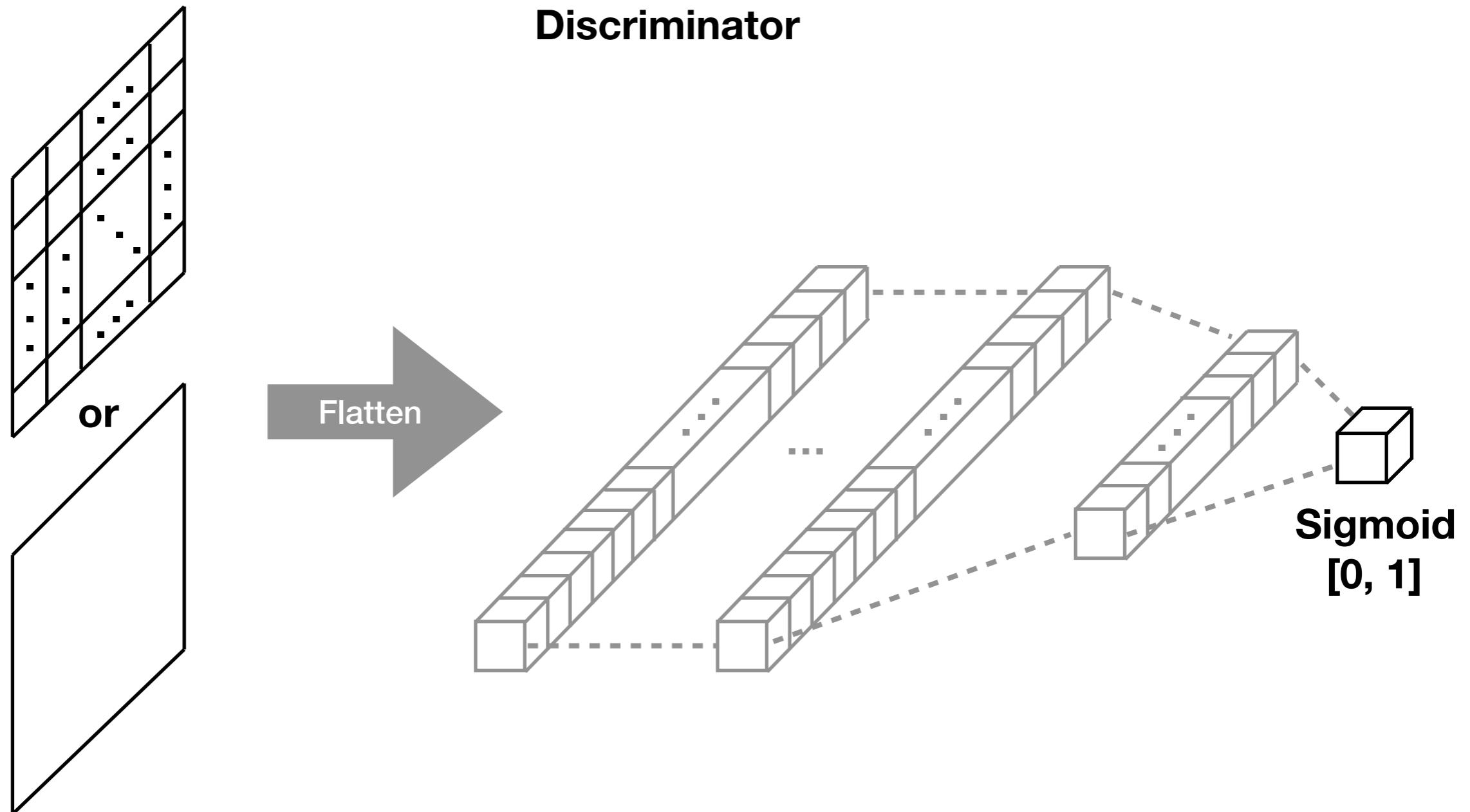


How does GAN work?



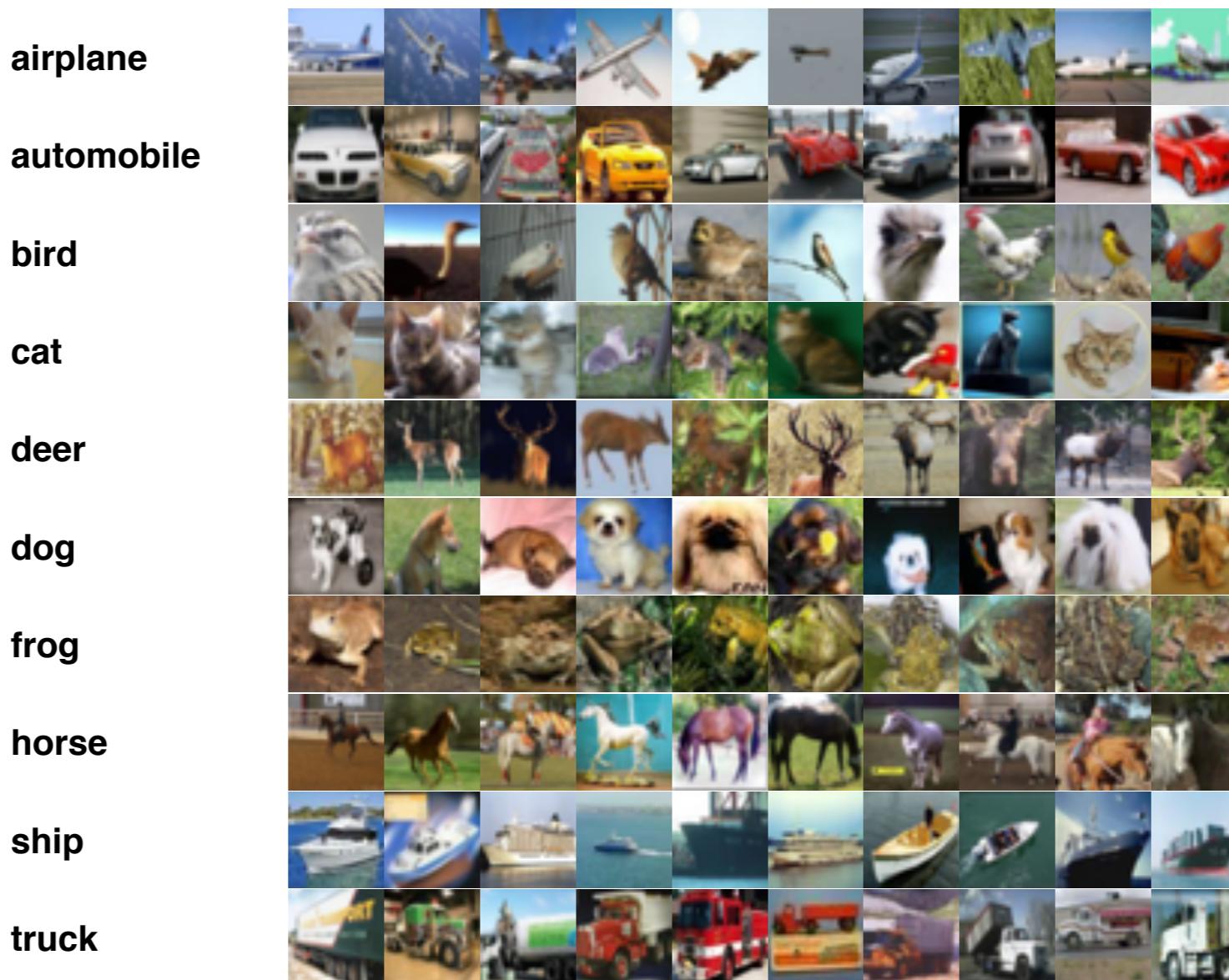
E.g. image generation model

How does GAN work?



Practice

CIFAR10

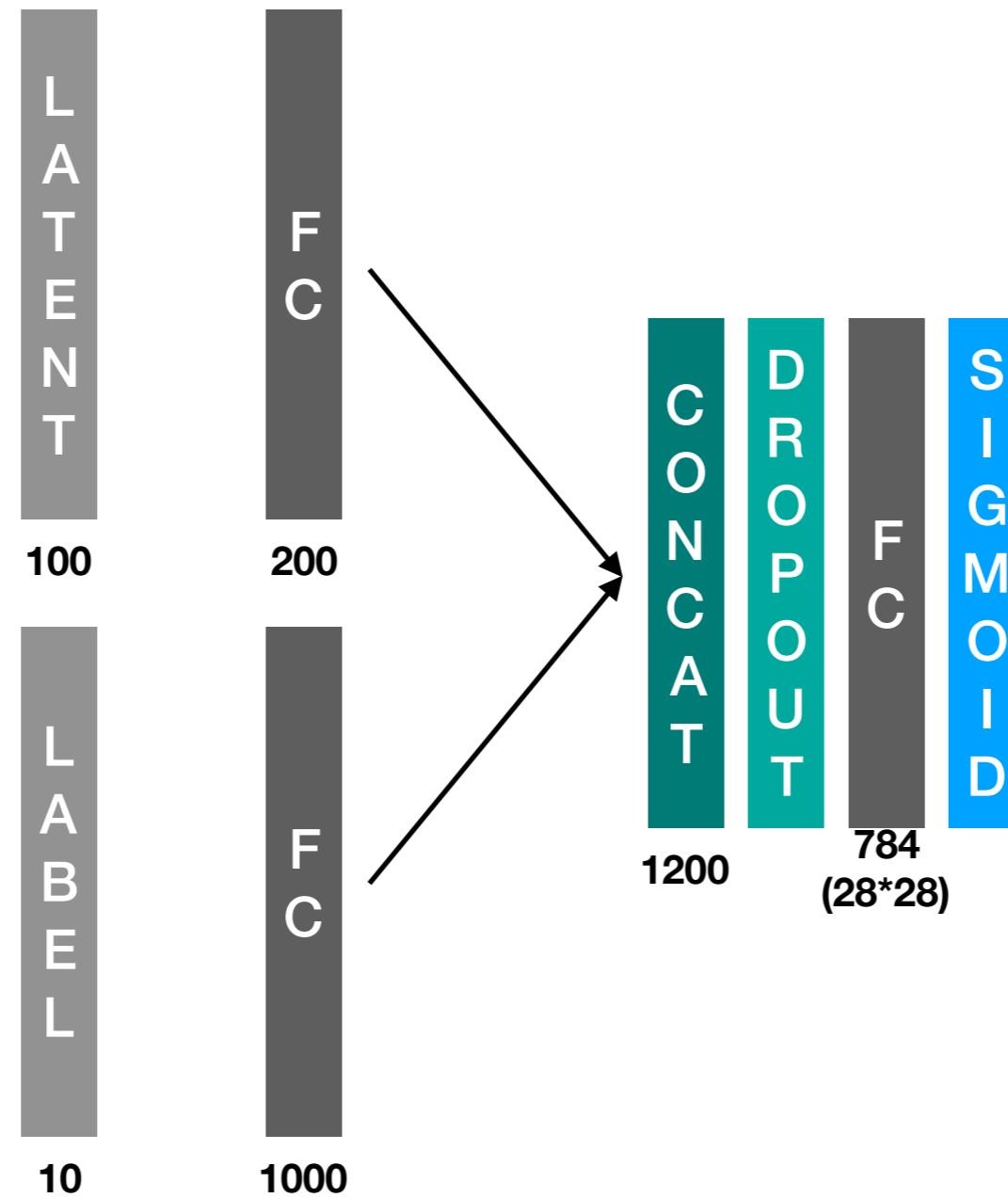


Credit. [Learning Multiple Layers of Features from Tiny Images](#), Alex Krizhevsky, 2009.

The CIFAR-10 dataset consists of 60,000 32x32 color images in 10 classes, with 6,000 images per class. There are 50,000 training images and 10,000 test images.

Conditional GAN

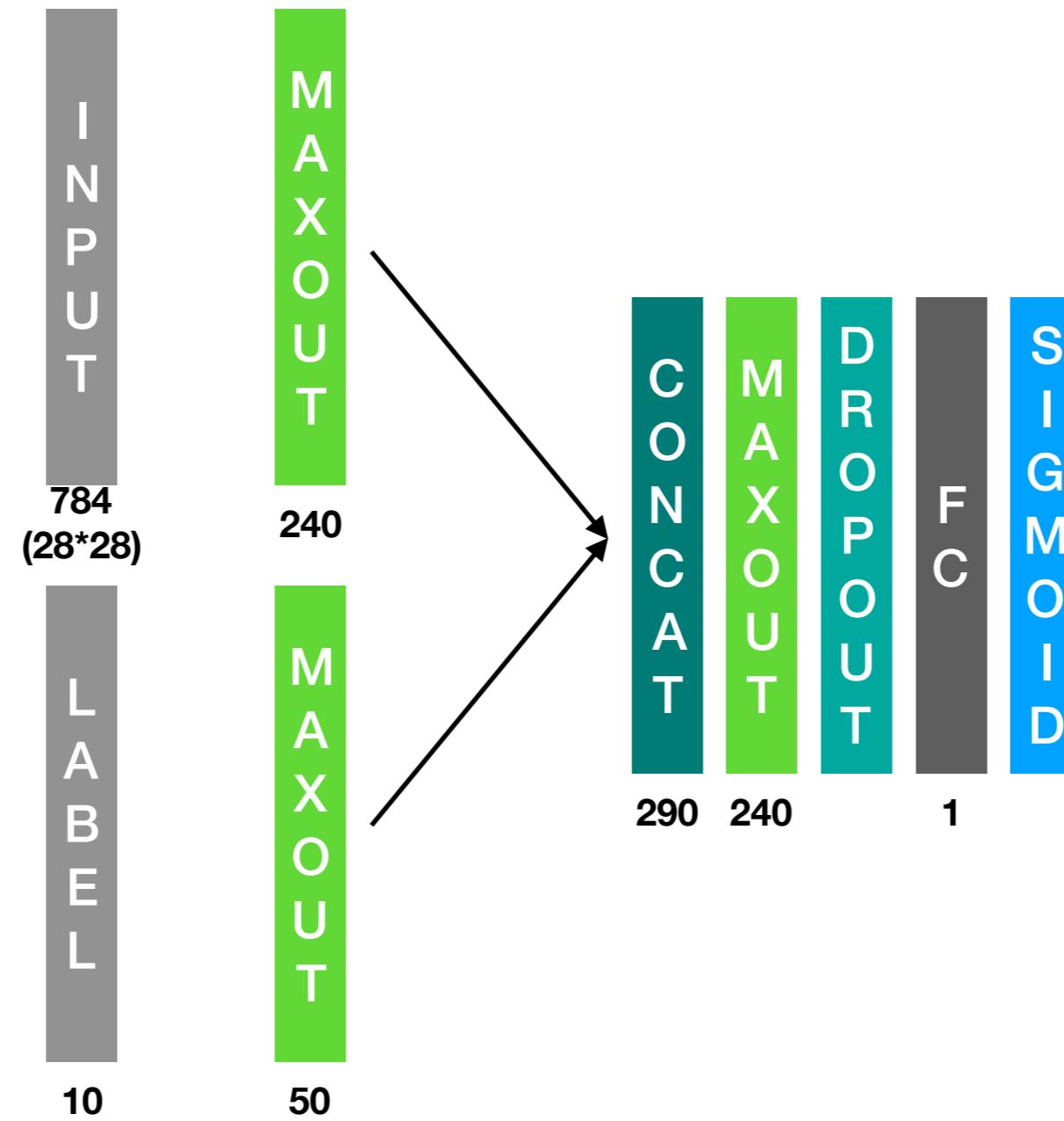
Generator



FC: Fully Connected layer

Note. This model is for MNIST dataset. LABEL is one-hot encoded label. Dropout rate is 0.5.

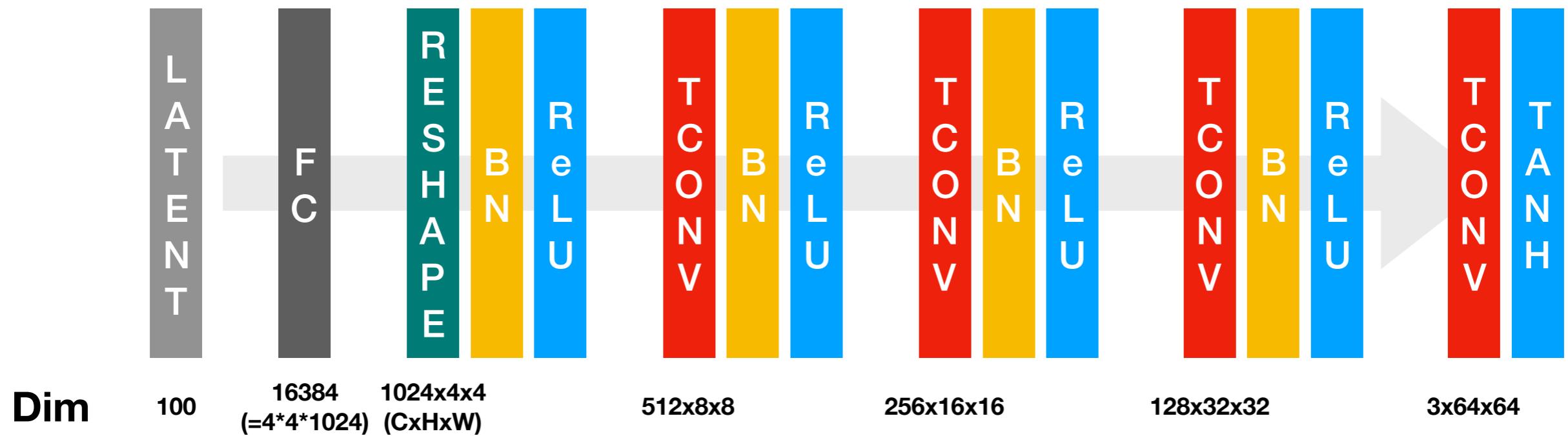
Discriminator



Note. This model is for MNIST dataset. LABEL is one-hot encoded label. Dropout rate is 0.5. MAXOUT layer includes dropout layer implicitly with rate 0.5. MAXOUT parameter k is set to 5 except for the last MAXOUT set to be 4.

DCGAN

Generator



BN: Batch Normalization

C: Channel

FC: Fully Connected layer

H: Height

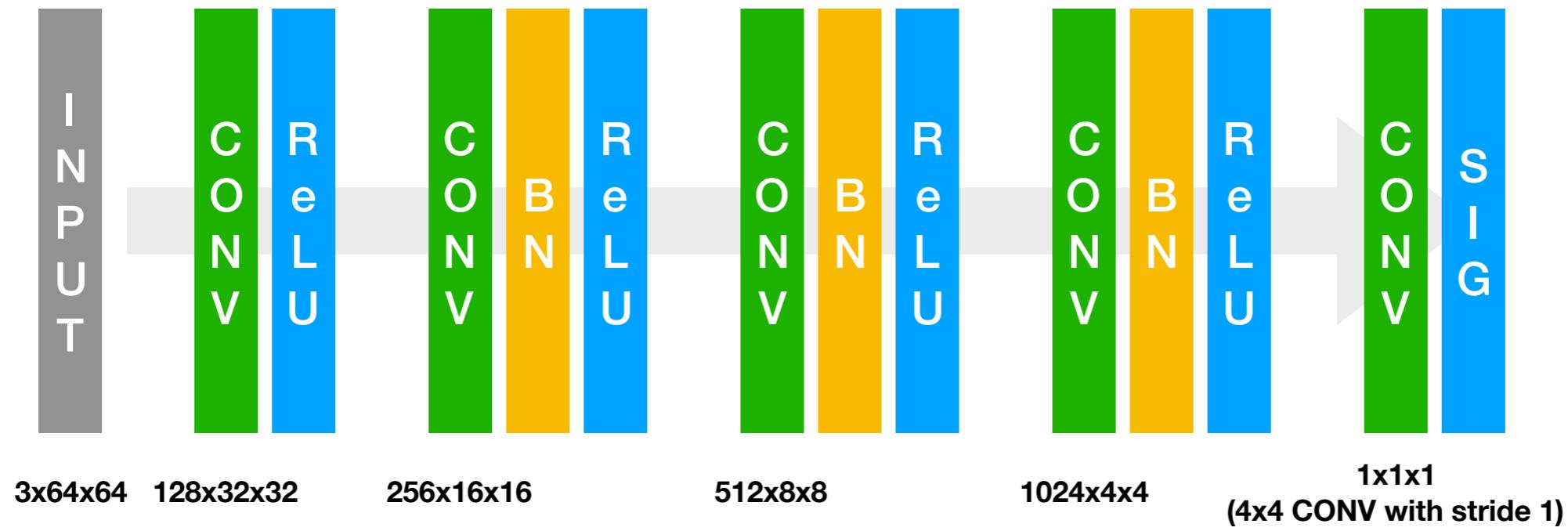
ReLU: Rectified Linear Unit

TCONV: 5x5 Transposed CONVolution (stride 2)

W: Width

Note. This is for the case of LSUN dataset. The number of TCONV layer can be varied with your target dataset.

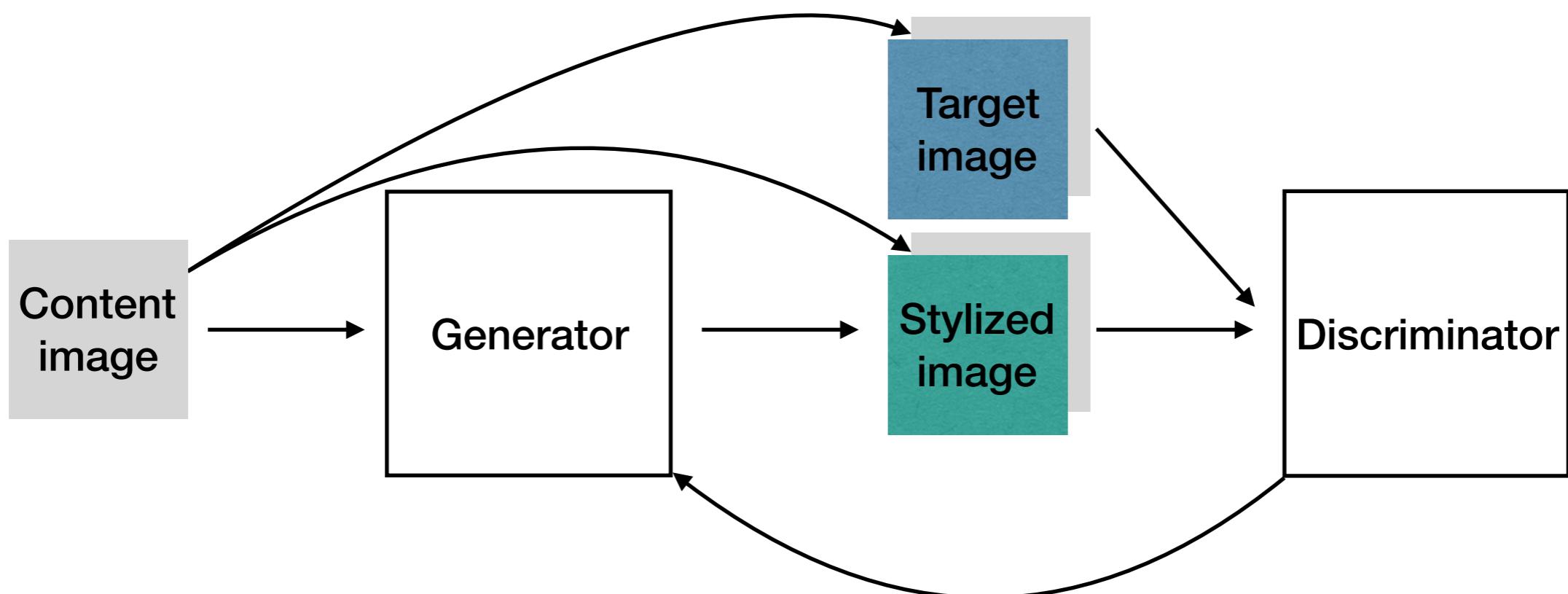
Discriminator



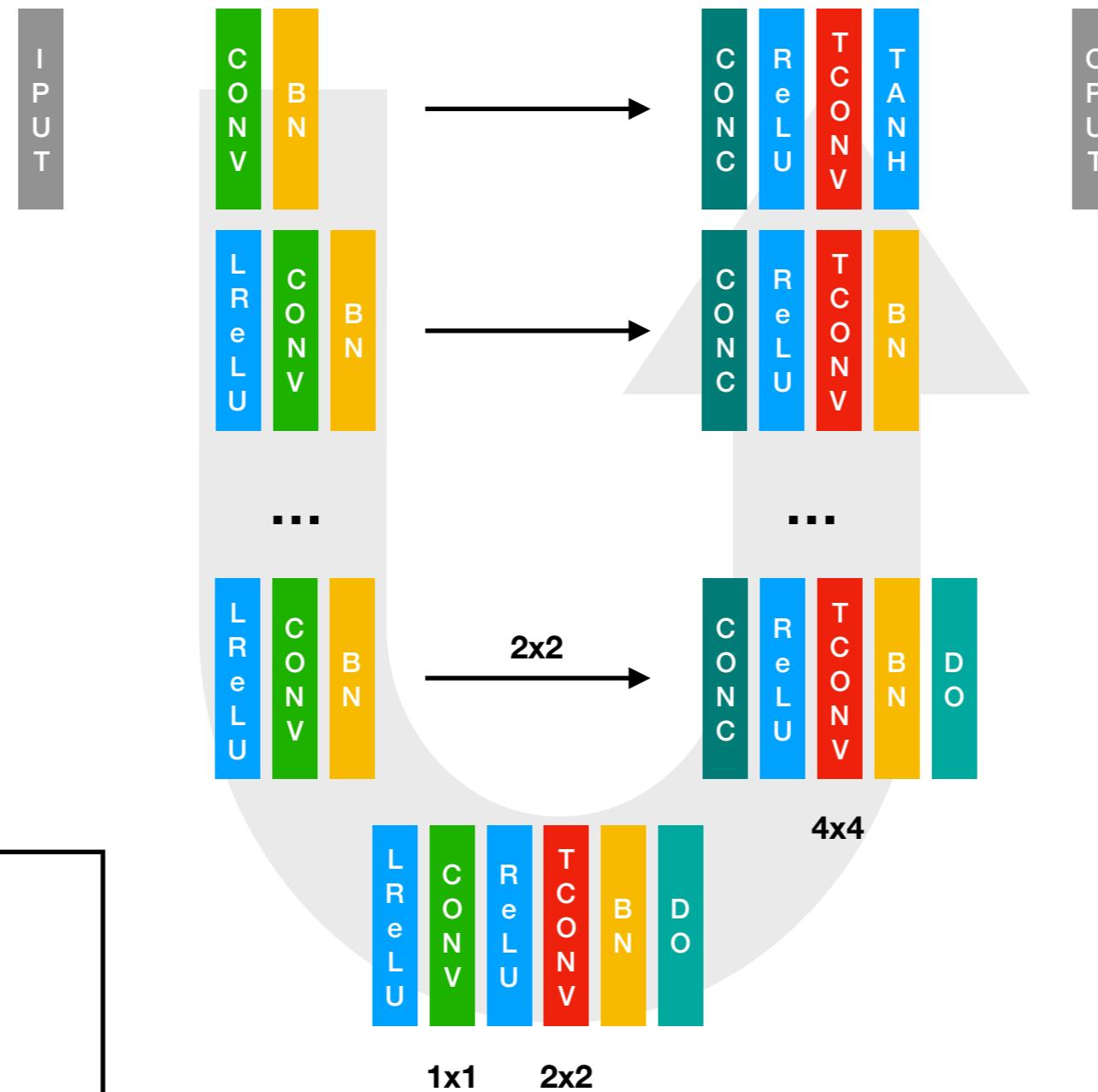
BN: Batch Normalization
C: Channel
H: Height
ReLU: Rectified Linear Unit
CONV: 5x5 CONVolution (stride 2)
W: Width

pix2pix

What is pix2pix?



Generator



BN: Batch Normalization

CONC: CONCatenation

CONV: 4x4 CONVolution (stride 2)

DO: DropOut ($p = 0.5$)

INPUT: InPUT

LReLU: Leaky ReLU with slope 0.2

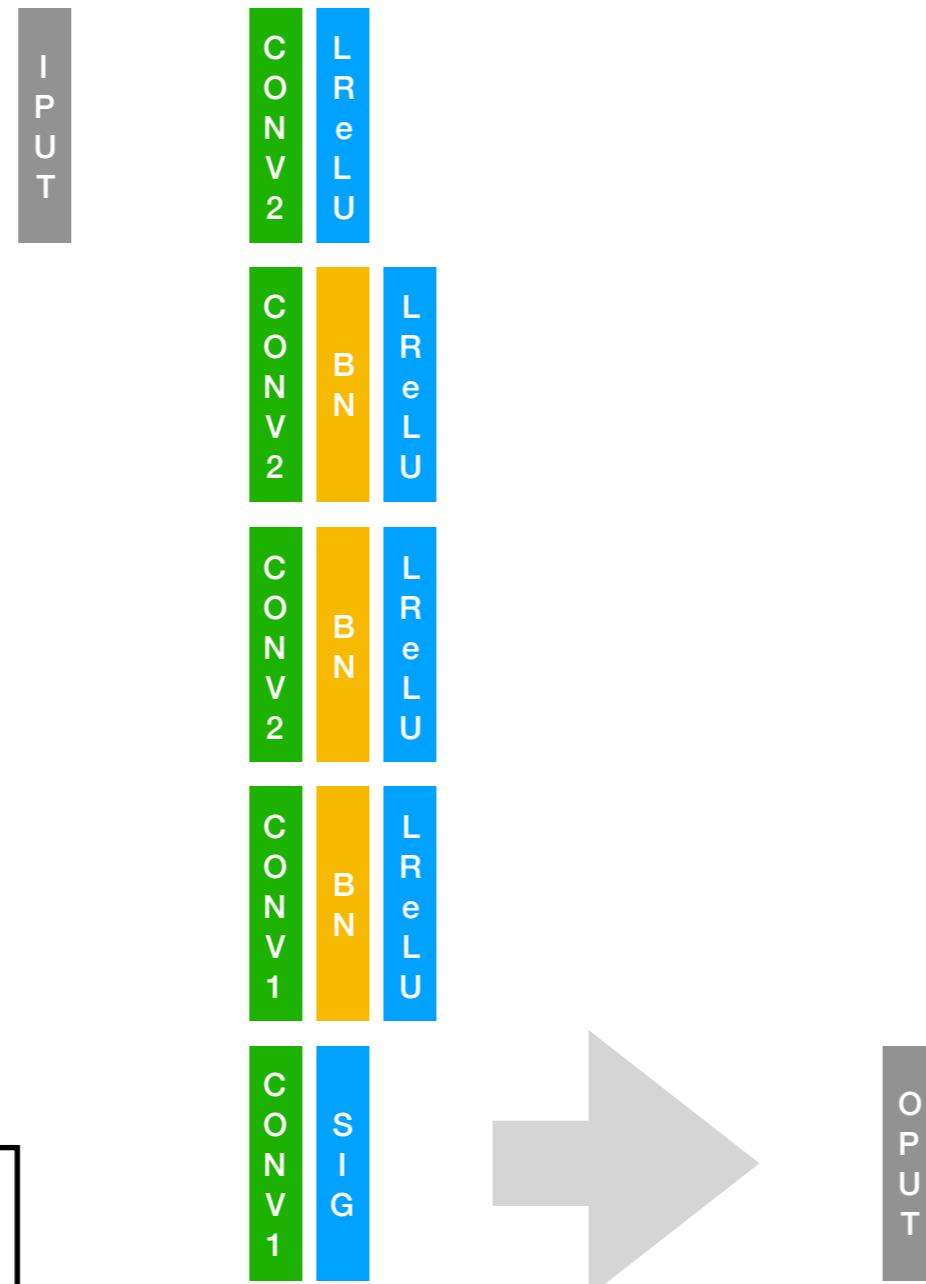
OPUT: OutPUT

ReLU: Rectified Linear Unit

TCONV: 4x4 Transposed CONV (stride 2)

Note. Dropout is applied where feature map size is 2x2, 4x4, and 8x8 in the decoder part.

Discriminator



BN: Batch Normalization
CONV1: 4x4 CONVolution (stride 1)
CONV2: 4x4 CONVolution (stride 2)
INPUT: InPUT
LReLU: Leaky ReLU with slope 0.2
OUTPUT: OutPUT

1 channel output

Note. This is for the case of 70x70 receptive field. Layers should be varied to change receptive field size.

Practice

CycleGAN

Appendix

A. Python class

I assume you know about Python function. **Class is usually used for storing related data or handling a certain type of data using functions defined in the class.** For example, calculator has four basic functions - addition, subtraction, multiplication, and division. By making a class equipped with above functions, we can calculate numbers easily.

```
class Calculator:
    def __init__(self): # __init__ function is called "constructor" as every time you make an instance of Calculator,
                        # codes written in the constructor will be implemented.
        self.value = 0

    def add(self, n):
        self.value += n
        print("After addition:", self.value)

    def subtract(self, n):
        self.value -= n
        print("After subtraction:", self.value)

    def multiply(self, n):
        self.value *= n
        print("After multiplication:", self.value)

    def divide(self, n):
        if n == 0:
            print("You can't divide with 0.")
            return

        else:
            self.value /= n
        print("After division", self.value)

    def reset(self):
        self.value = 0
        print("reset value to", self.value)

print("Calculator a...")
a = Calculator() # Make an instance of Calculator class. You need parenthesis after class name to do this.

a.add(5) # Call add function defined in Calculator class.
a.divide(0)
a.divide(5)

print()
print("Calculator b...")
b = Calculator() # Make another instance of Calculator class. This does not share value with calculator a.

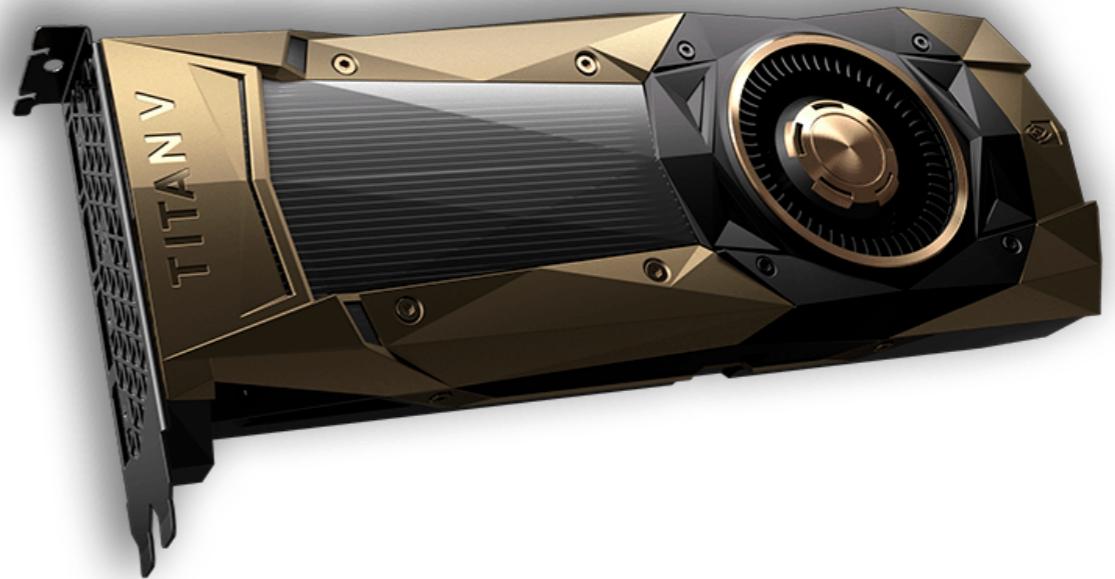
b.add(1)
b.multiply(100)
b.reset()
```

CPU vs. GPU



credit. [hohardware.com](https://www.hothardware.com)

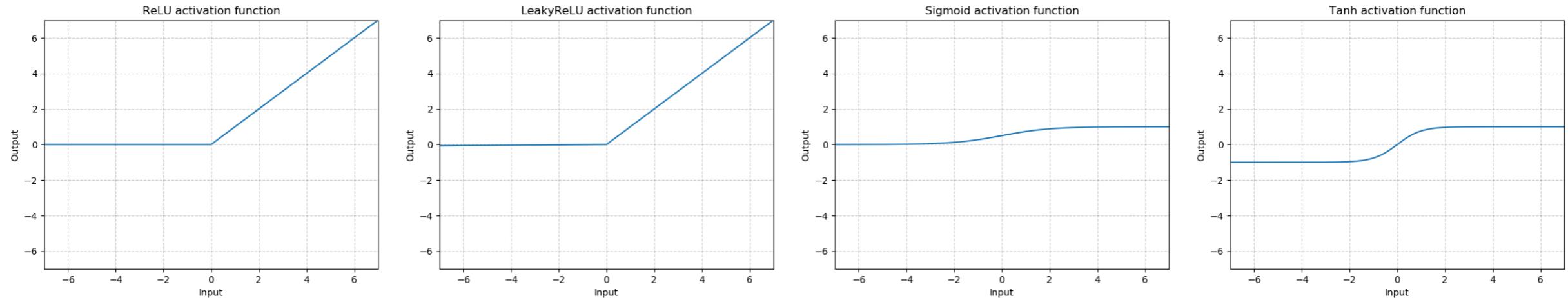
A central processing unit (CPU), also called a central processor or main processor, is the electronic circuitry within a computer that carries out the instructions of a computer program by performing the basic arithmetic, logic, controlling, and input/output operations specified by the instructions.



credit. NVIDIA

A graphics processing unit (GPU) is a specialized electronic circuit designed to rapidly manipulate and alter memory to accelerate the creation of images in a frame buffer intended for output to a display device.

Activation functions

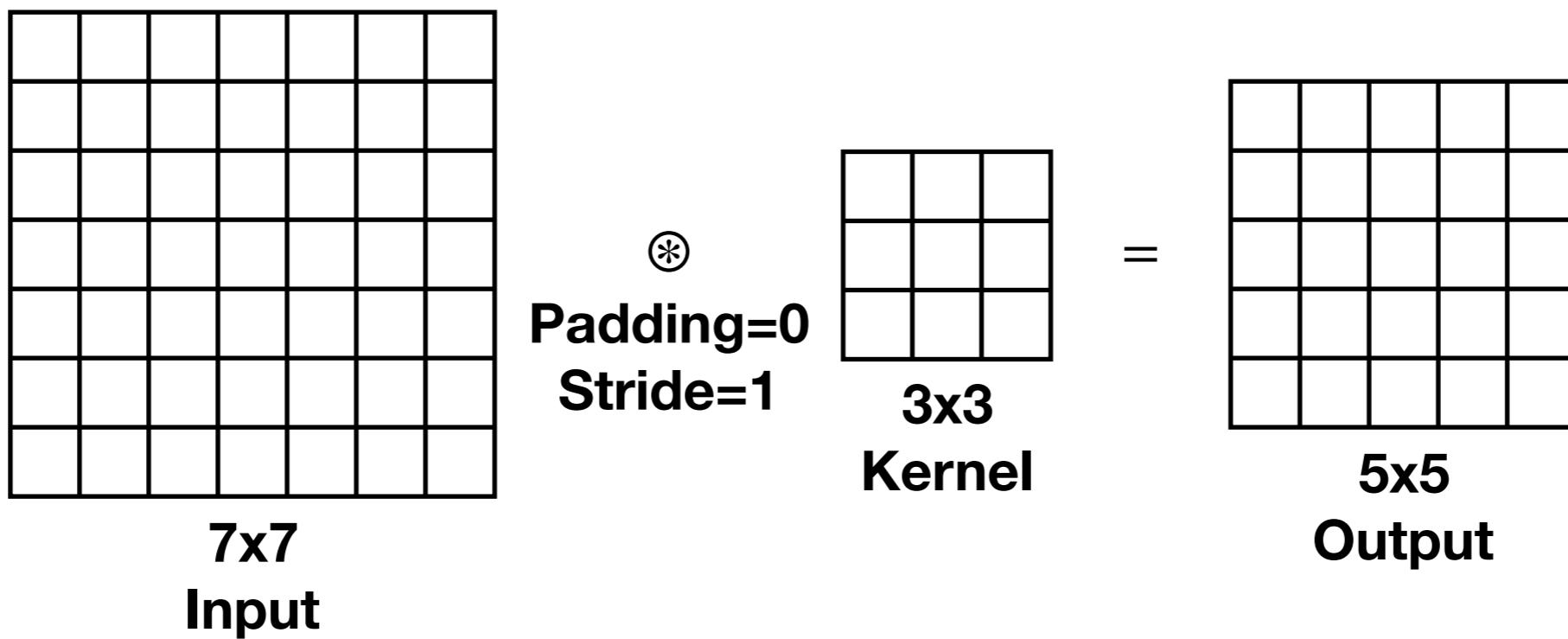


- ReLU (Rectified linear unit) : $\max(0, x)$
- Leaky ReLU : $\max(0, x) + \text{negative slope} \times \min(0, x)$
- Hyperbolic tangent (tanh):
$$\frac{e^x - e^{-x}}{e^x + e^{-x}}$$
- Logistic sigmoid :
$$\frac{1}{1 + e^{-x}} = \frac{e^x}{e^x + 1} = \frac{1}{2} + \frac{1}{2}\tanh\left(\frac{x}{2}\right)$$

Loss functions

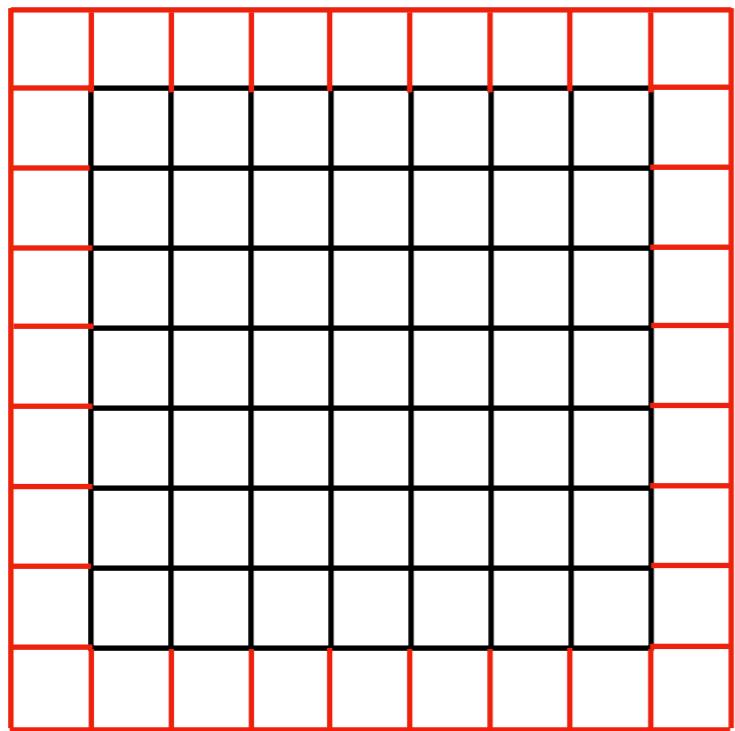
- Binary cross entropy loss (BCE) : $-\log \hat{p}(y_c | x) - \sum_{k=1, k \neq c}^C \log[1 - \hat{p}(y_k | x)]$
- Cross entropy loss (CE) : $\mathbb{E}_{p(y|x)}[-\log \hat{p}(y | x)] = - \sum_{c=1}^C p(y_c | x) \log \hat{p}(y_c | x) = -\log \hat{p}(y_c | x)$
- Mean squared error loss (MSE) : $\frac{1}{MN} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} (x_{i,j} - \hat{x}_{i,j})^2$
- Mean absolute error loss (MAE) : $\frac{1}{MN} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} |x_{i,j} - \hat{x}_{i,j}|$

Convolution



$$o = i - k + 1$$

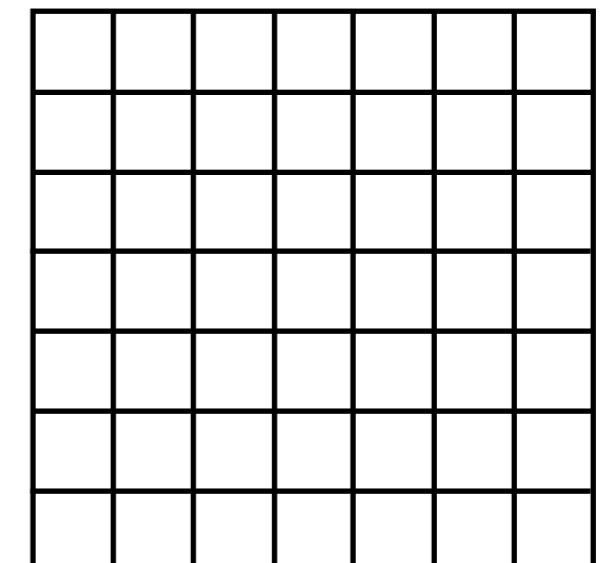
Convolution



$(7 + 2 \times p) \times (7 + 2 \times p)$
Input

\circledast
Padding=1
Stride=1

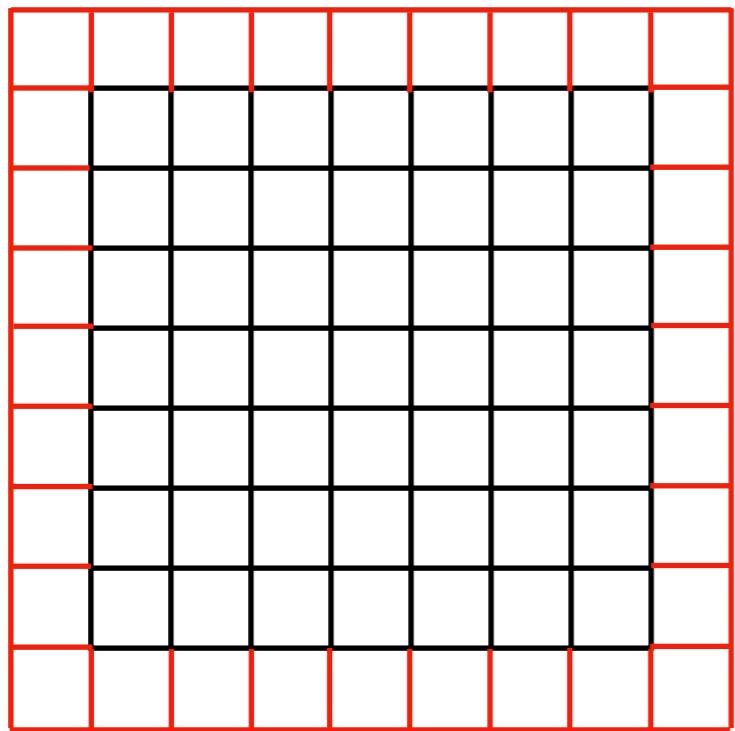
3x3 Kernel



**7x7
Output**

$$o = i - k + 2p + 1$$

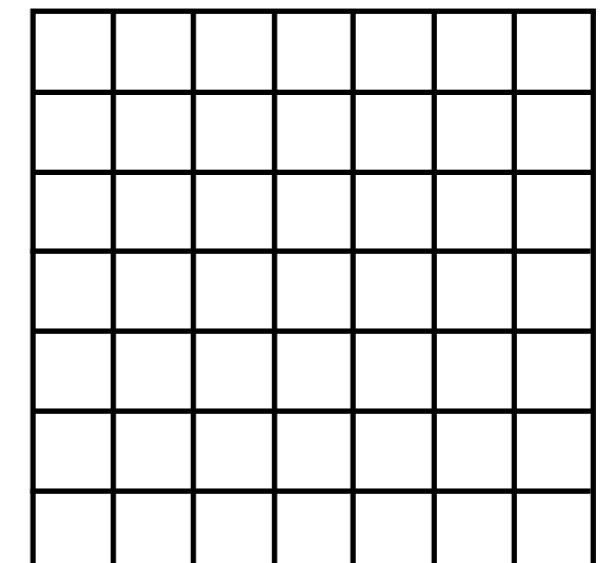
Convolution



$(7 + 2 \times p) \times (7 + 2 \times p)$
Input

\circledast
Padding=1
Stride=2

3x3 Kernel



4x4 Output

$$o = \lfloor \frac{i - k + 2p}{s} \rfloor + 1$$