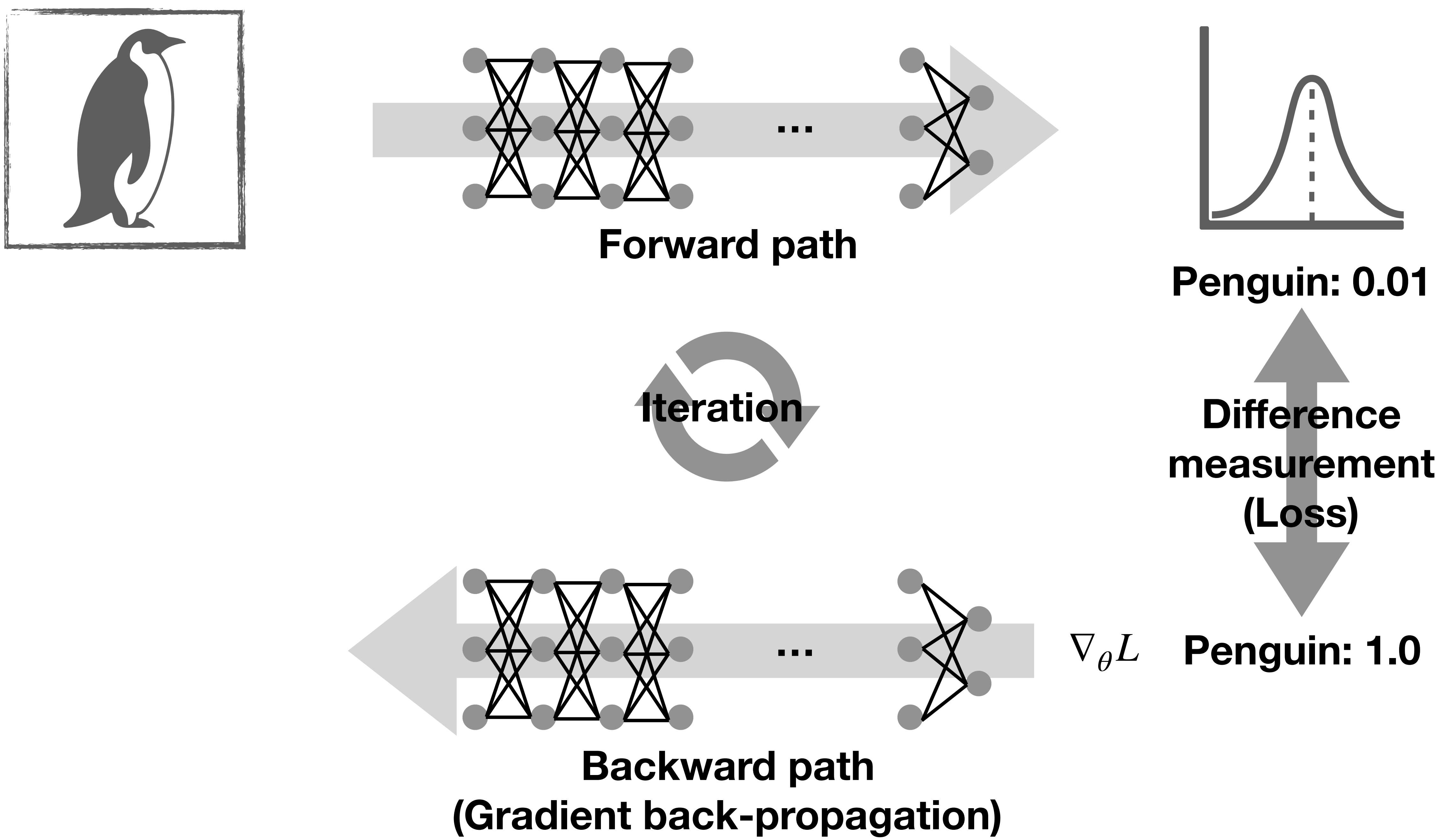
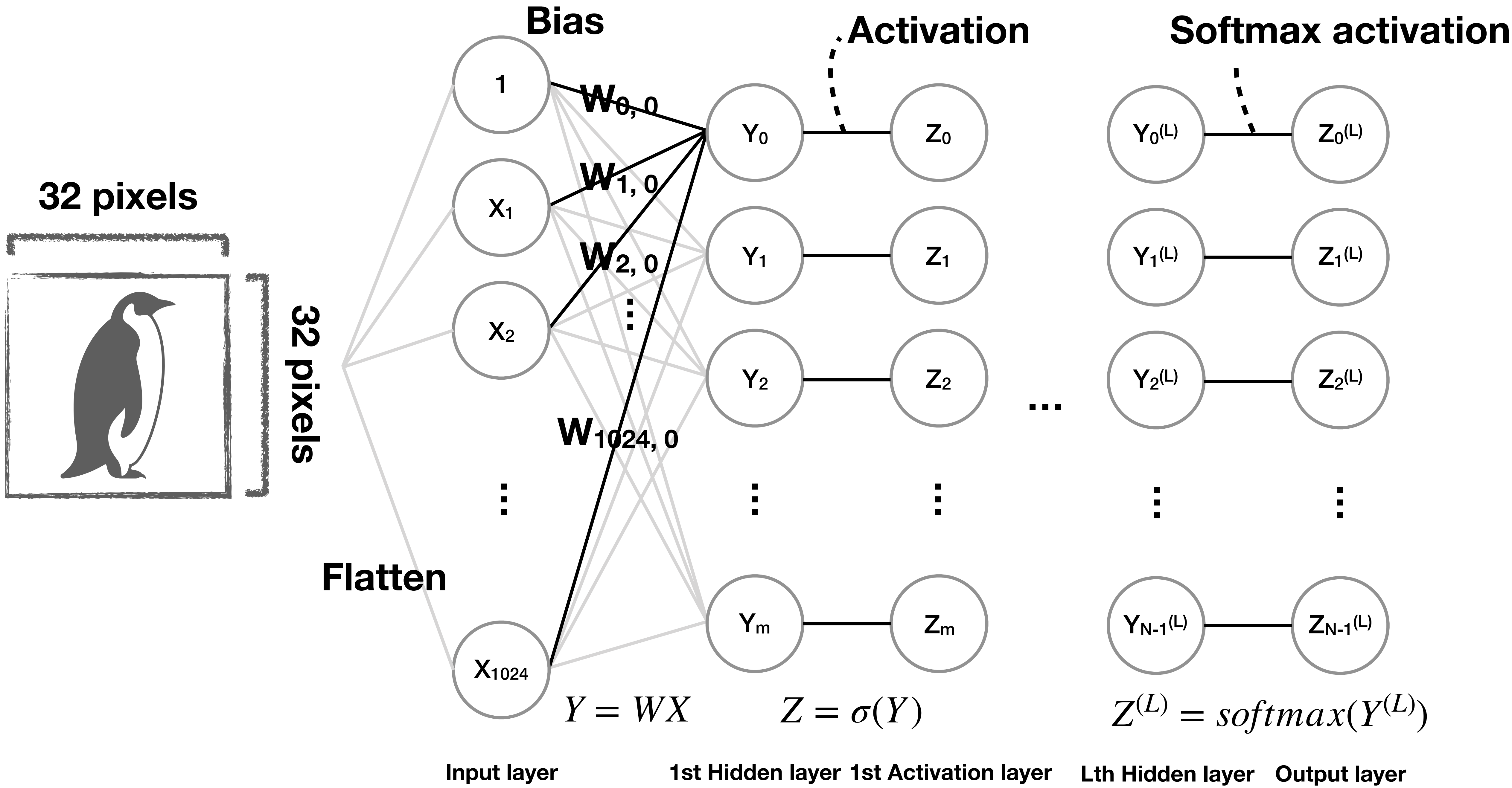
Classification Model

How does it work?



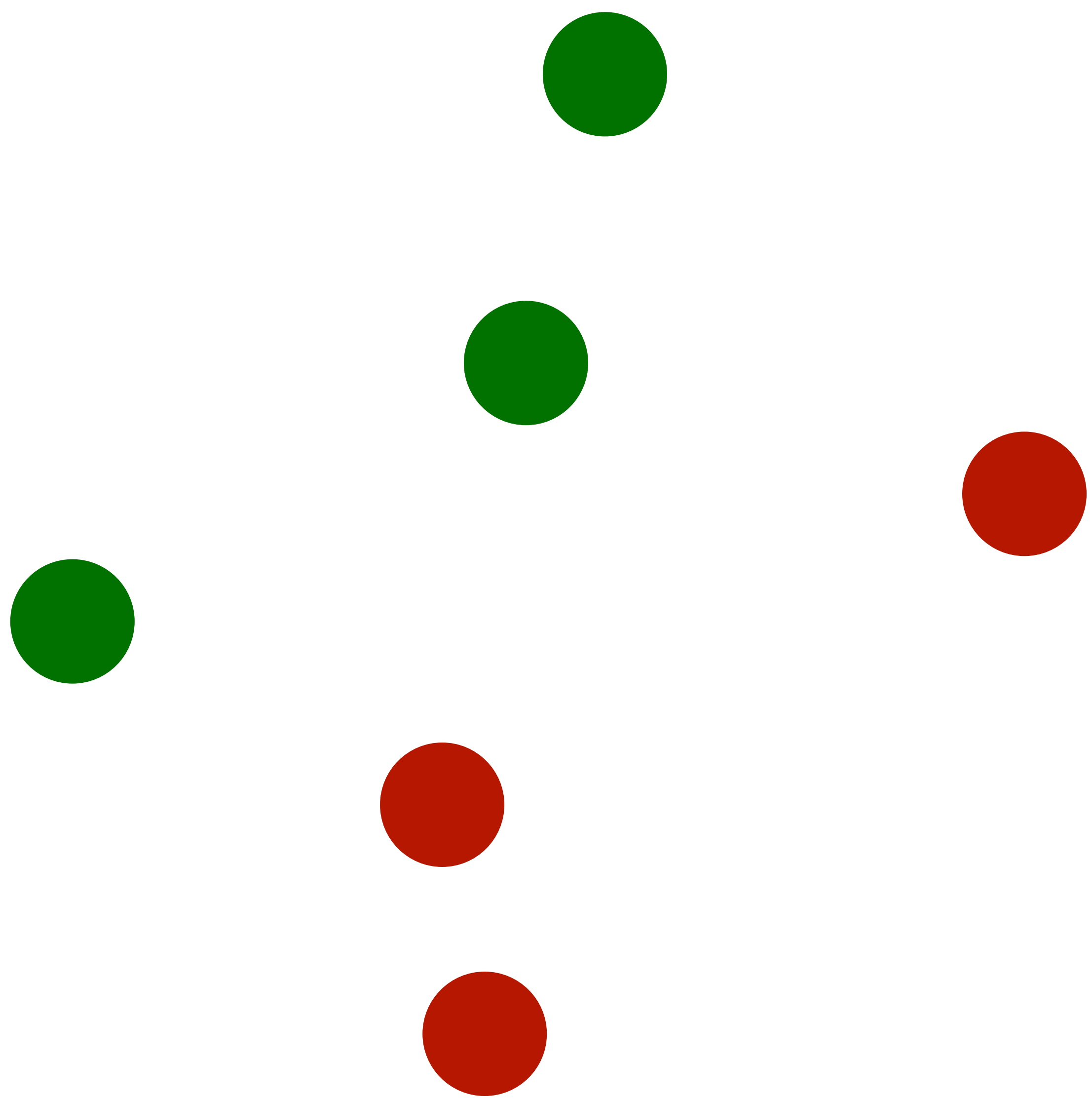
Forward path



Densely connected
(fully connected)

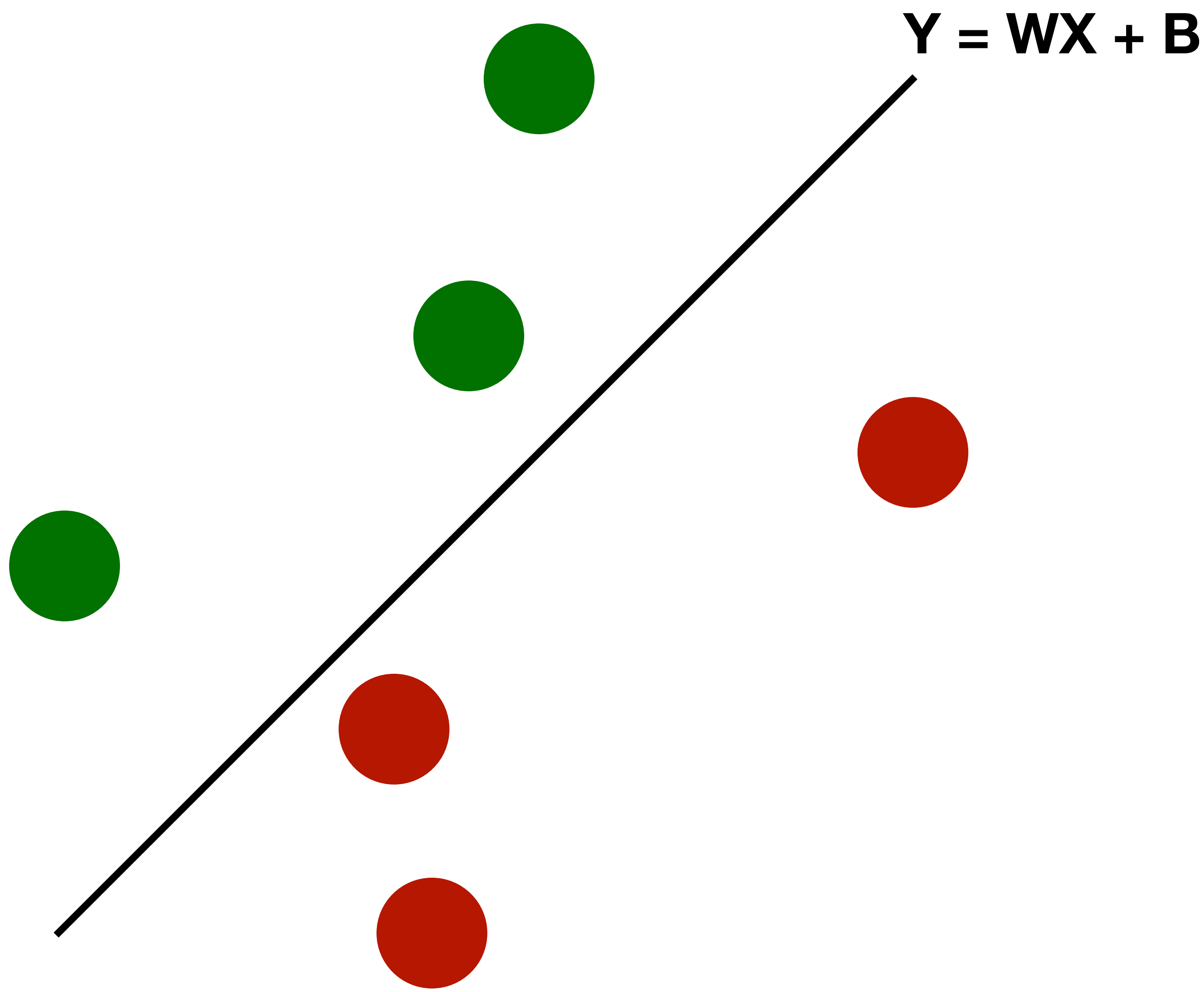
Forward path

- Why do we need an activation after a weight layer?



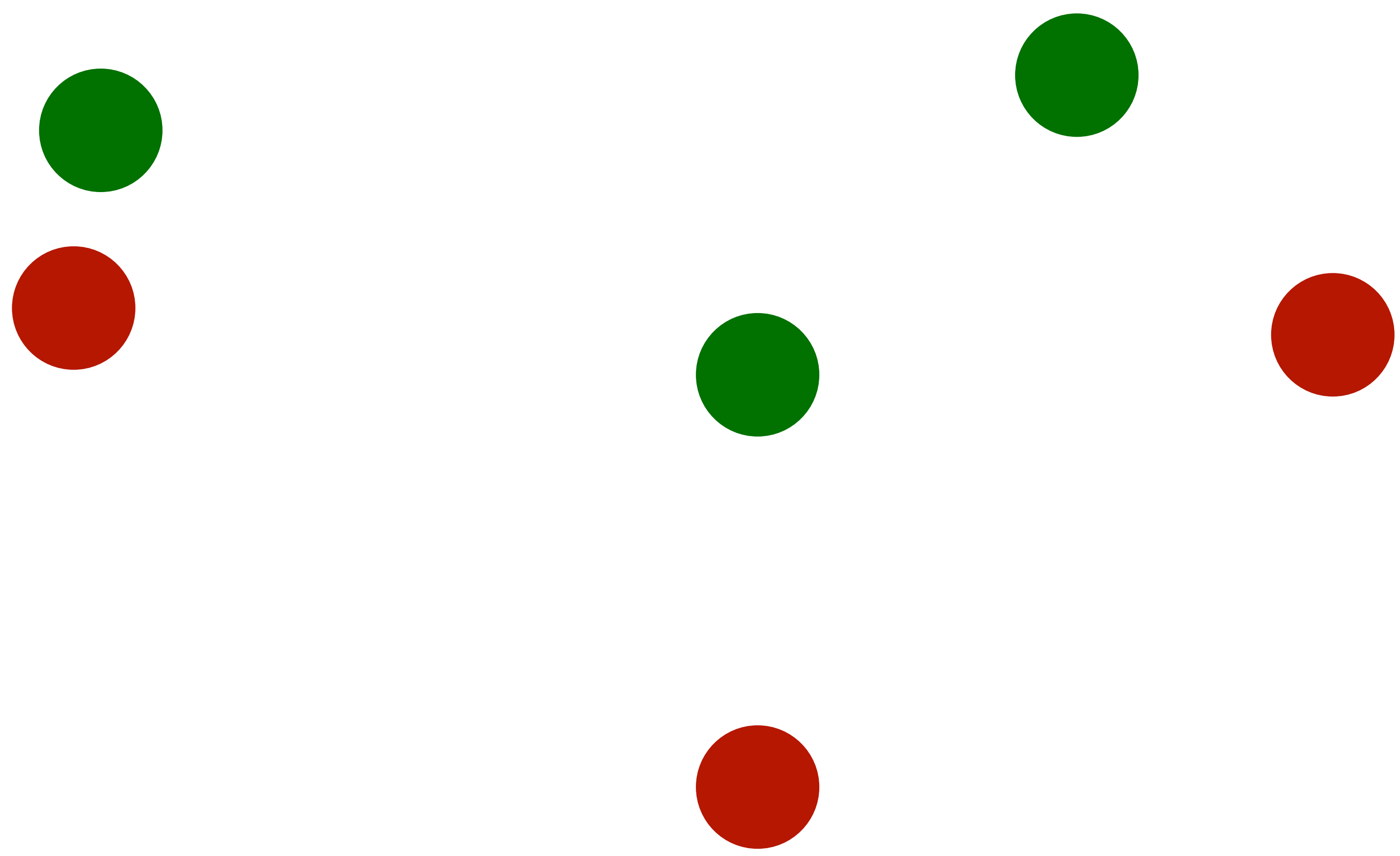
Forward path

- Why do we need an activation after a weight layer?



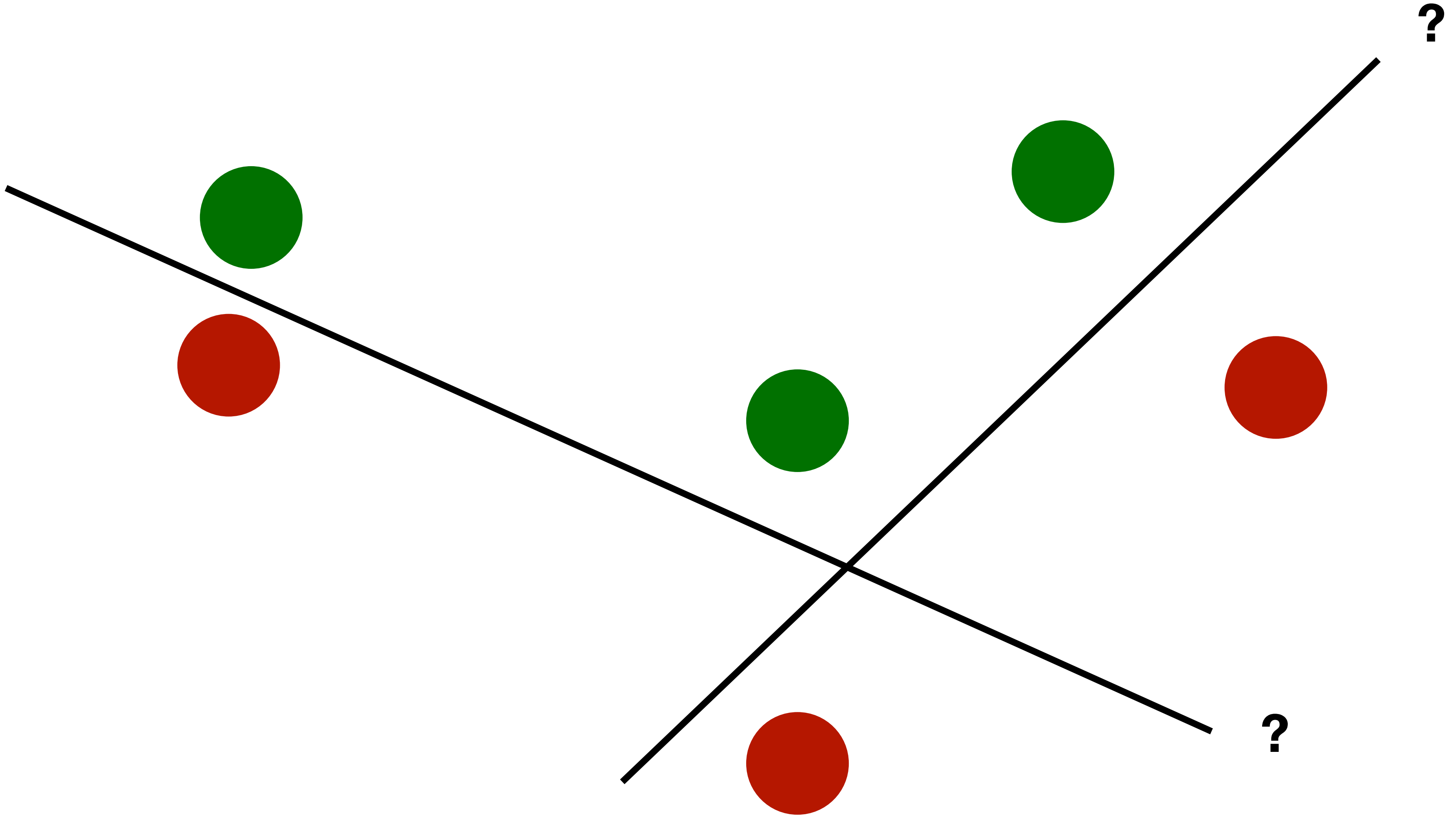
Forward path

- Why do we need an activation after a weight layer?



Forward path

- Why do we need an activation after a weight layer?



Forward path

- Why do we need an activation after a weight layer?

Without any activation function, it's impossible to express higher degree function than linear function

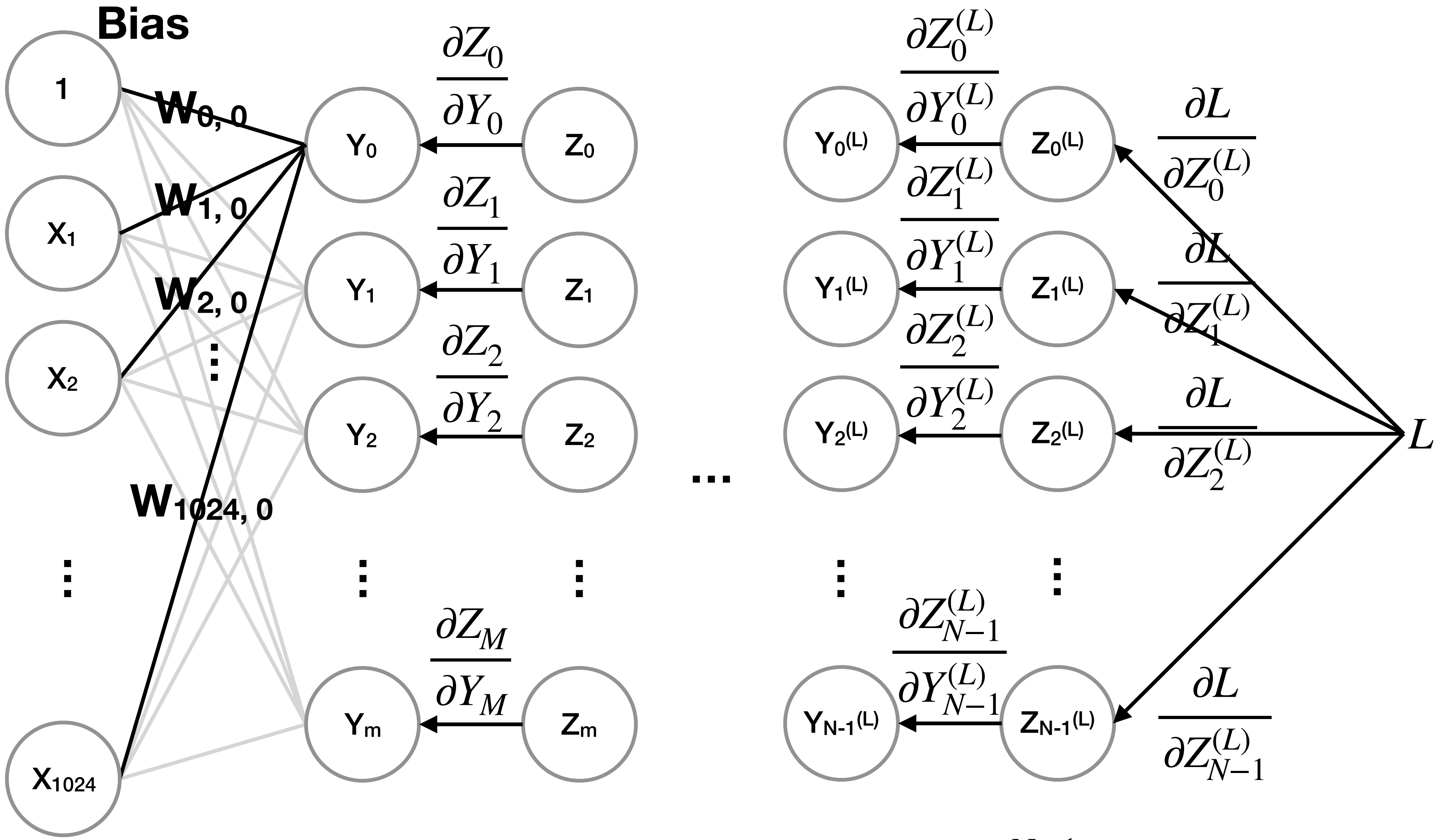
$$Y_1 = WX_1 + B$$
$$Y_2 = W(WX_1 + B) + B = WX_1 + B$$

$$Y_1 = WX_1 + B$$
$$Z_1 = f(Y_1)$$
$$Y_2 = WZ_1 + B$$

where f(.) is non-linear activation function

Thus we add an activation function after a weight layer.

Backward path



$$\frac{\partial L}{\partial W0,\,0} = \frac{\partial L}{\partial Y_0}\frac{\partial Y_0}{W_{0,\,0}}$$

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

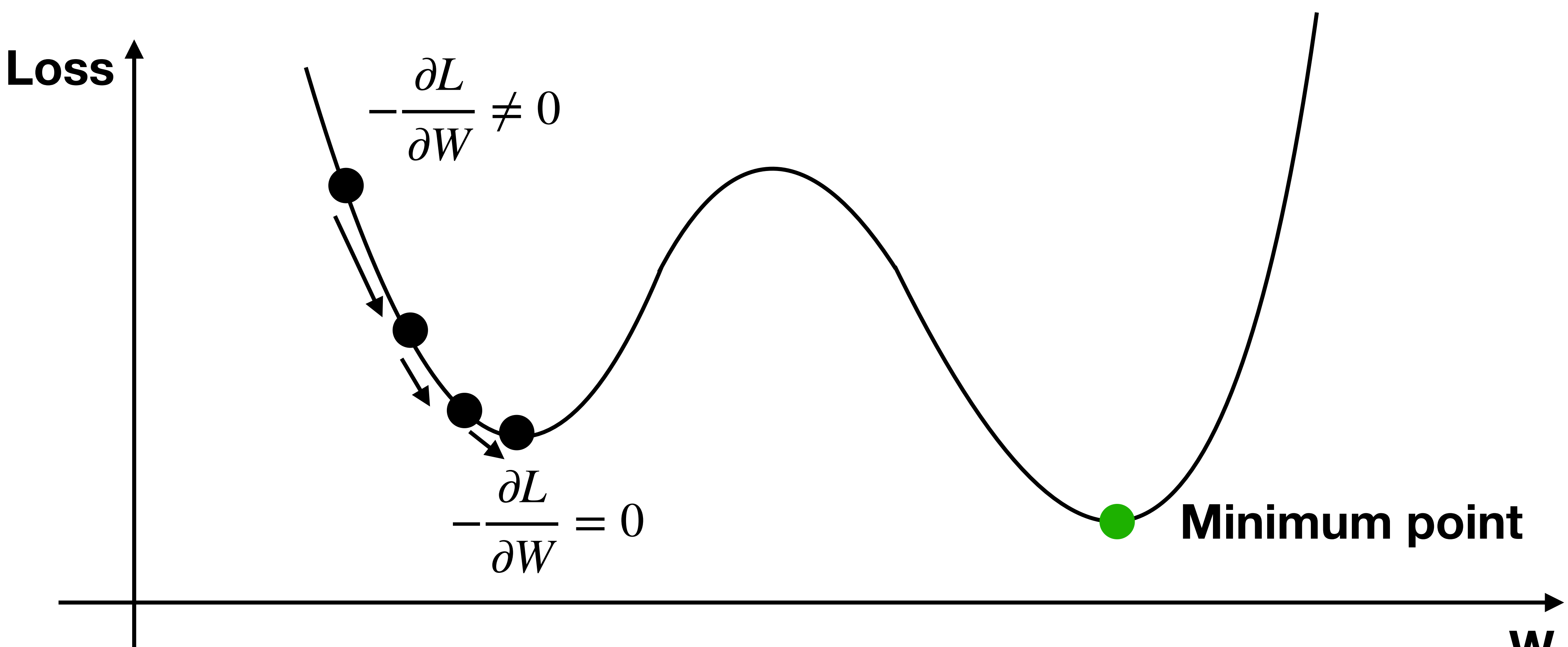
$$L = -\sum_{k=0}^{N-1} t_k logZ_k = -\,logZ_i$$

Cross-entropy loss

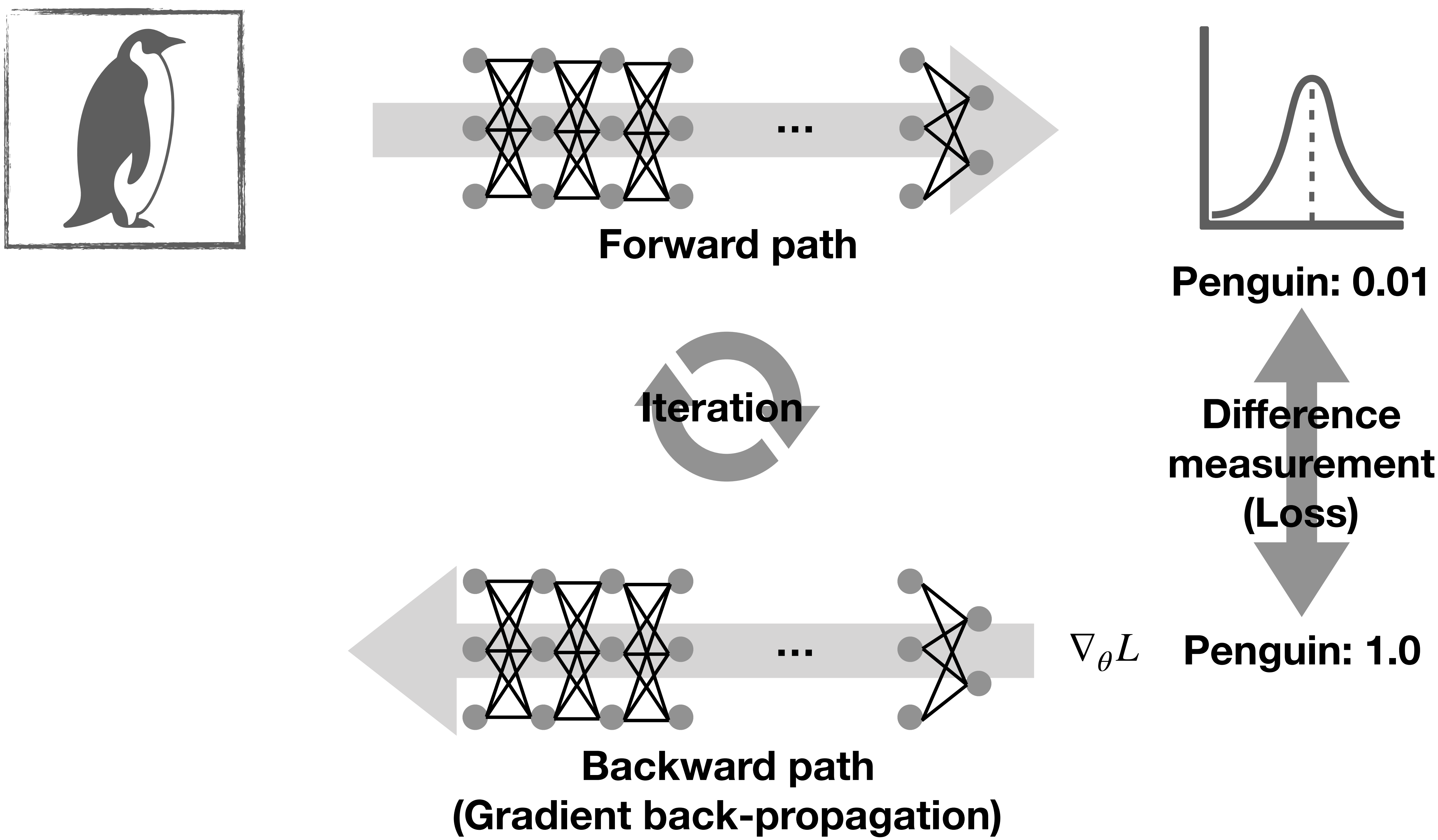
Backward path

- Gradient back-propagation 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



How does it work?



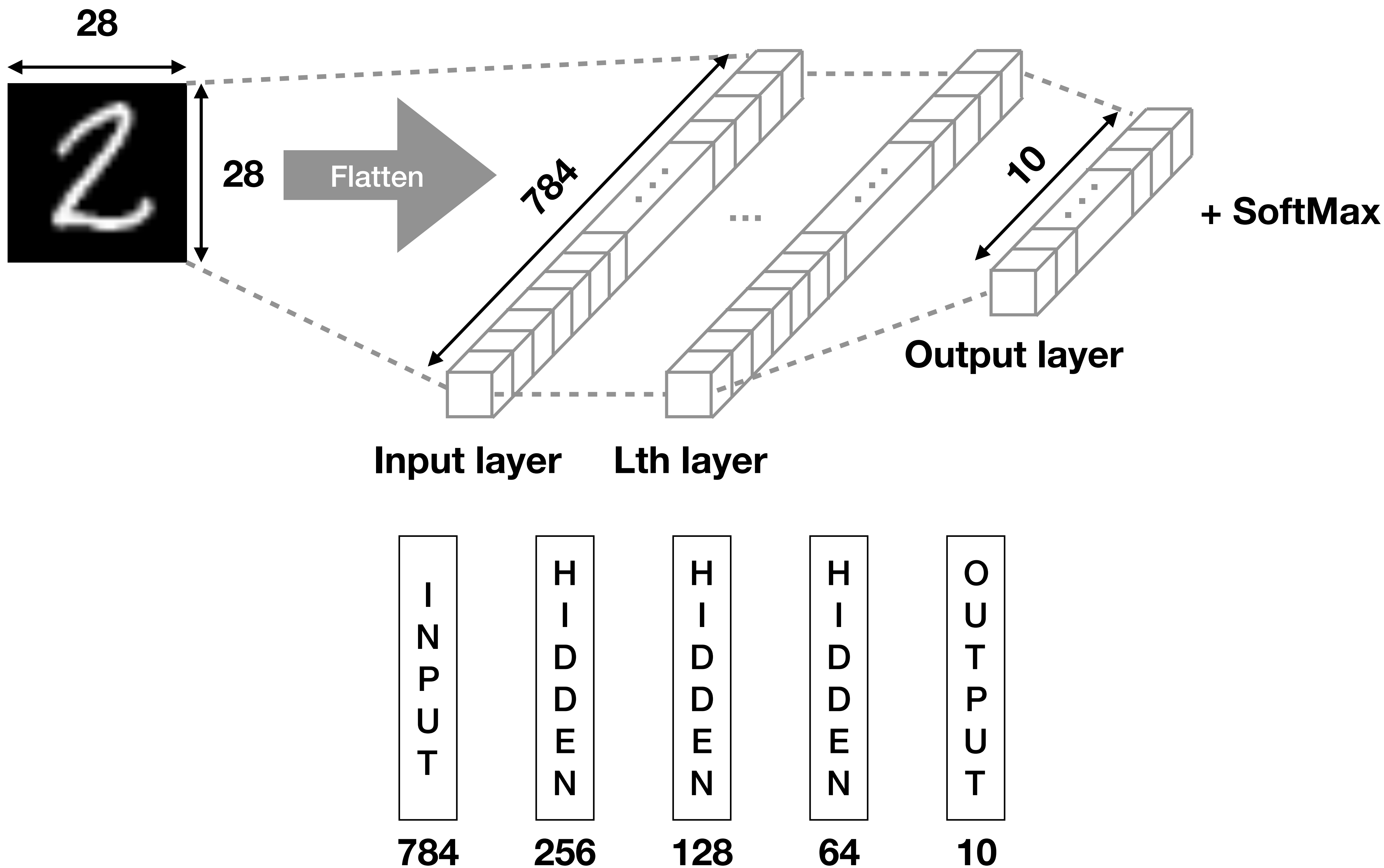
Practice

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.

Model architecture



Code snippet

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


class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=64, kernel_size=3, stride=2, padding=1)
        self.conv2 = nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3, stride=2, padding=1)
        self.conv3 = nn.Conv2d(in_channels=128, out_channels=256, kernel_size=3, stride=2, padding=1)
        self.fc = nn.Linear(in_features=4 * 4 * 256, out_features=10)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.relu(self.conv2(x))
        x = F.relu(self.conv3(x))
        x = x.view(x.shape[0], -1)
        x = self.fc(x)
        return x


class MLP(nn.Module):
    def __init__(self):
        super(MLP, self).__init__()
        self.linear1 = nn.Linear(in_features=28 * 28, out_features=256)
        self.linear2 = nn.Linear(in_features=256, out_features=128)
        self.linear3 = nn.Linear(in_features=128, out_features=64)
        self.linear4 = nn.Linear(in_features=64, out_features=10)

    def forward(self, x):
        x = F.relu(self.linear1(x))
        x = F.relu(self.linear2(x))
        x = F.relu(self.linear3(x))
        x = self.linear4(x)
        return x
```

For a full code, please visit
https://github.com/NoelShin/Deep-Learning-Bootcamp-with-PyTorch

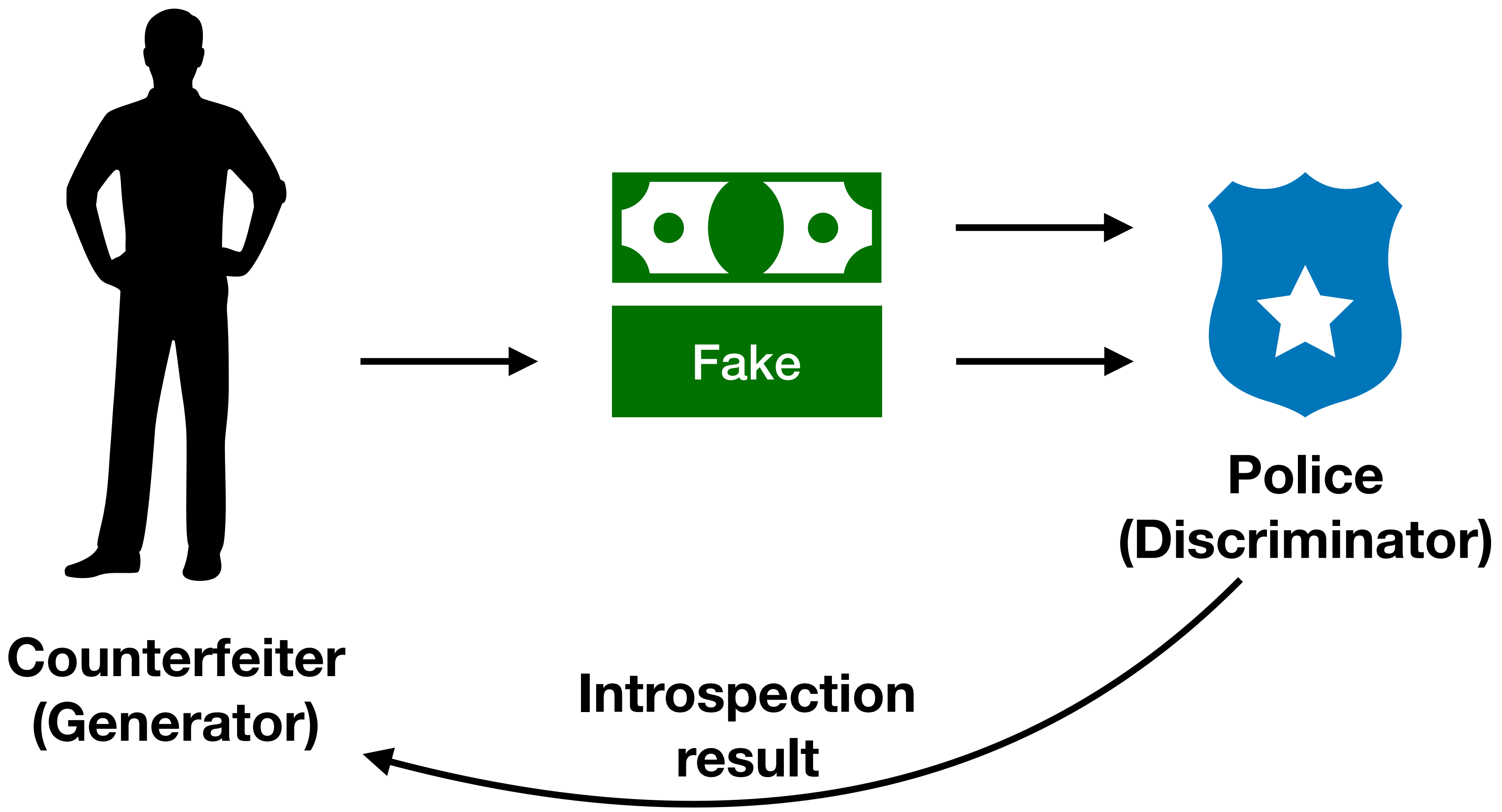
Generative Model

Various generative models

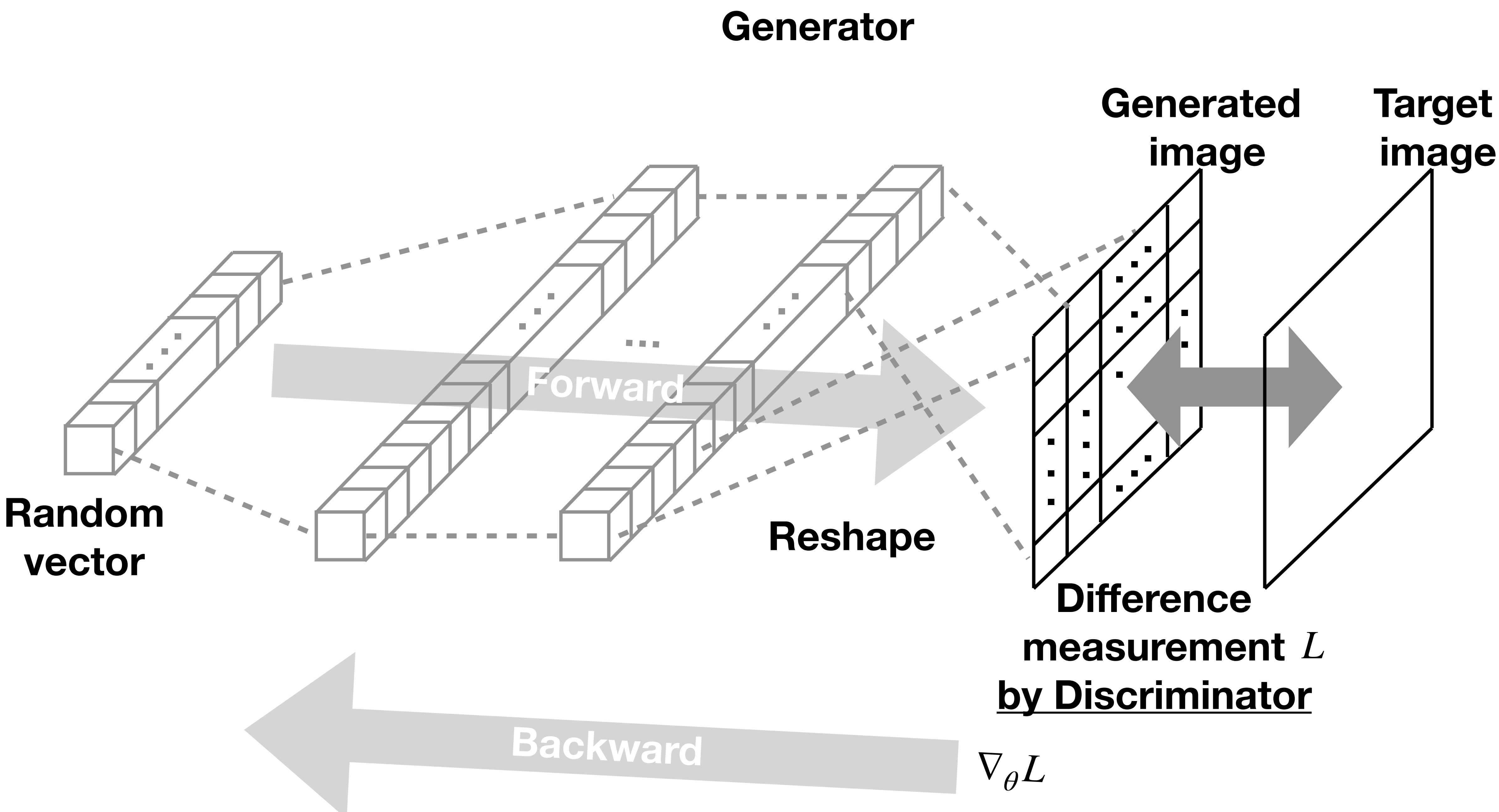
- Hidden Markov Model (HMM)
- Restricted Boltzmann Machine (RBM)
- Variational Auto-Encoder (VAE)
- Recurrent Neural Network (RNN)
- **Generative Adversarial Network (GAN)**

GAN

What is GAN?

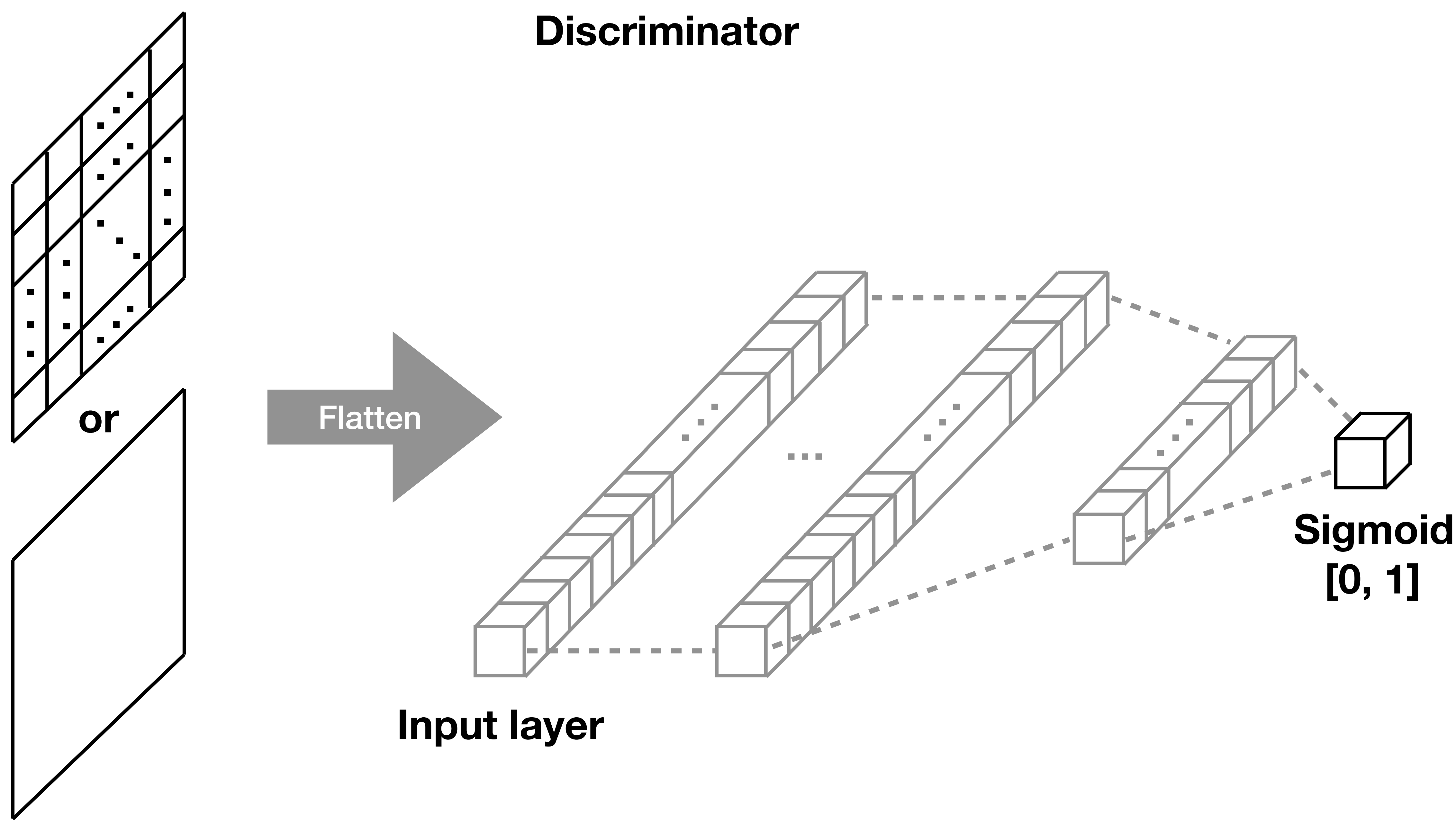

Fake
Police
(Discriminator)
Counterfeiter
(Generator)
Introspection
result

How does GAN work?



E.g. image generation model

How does GAN work?



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.

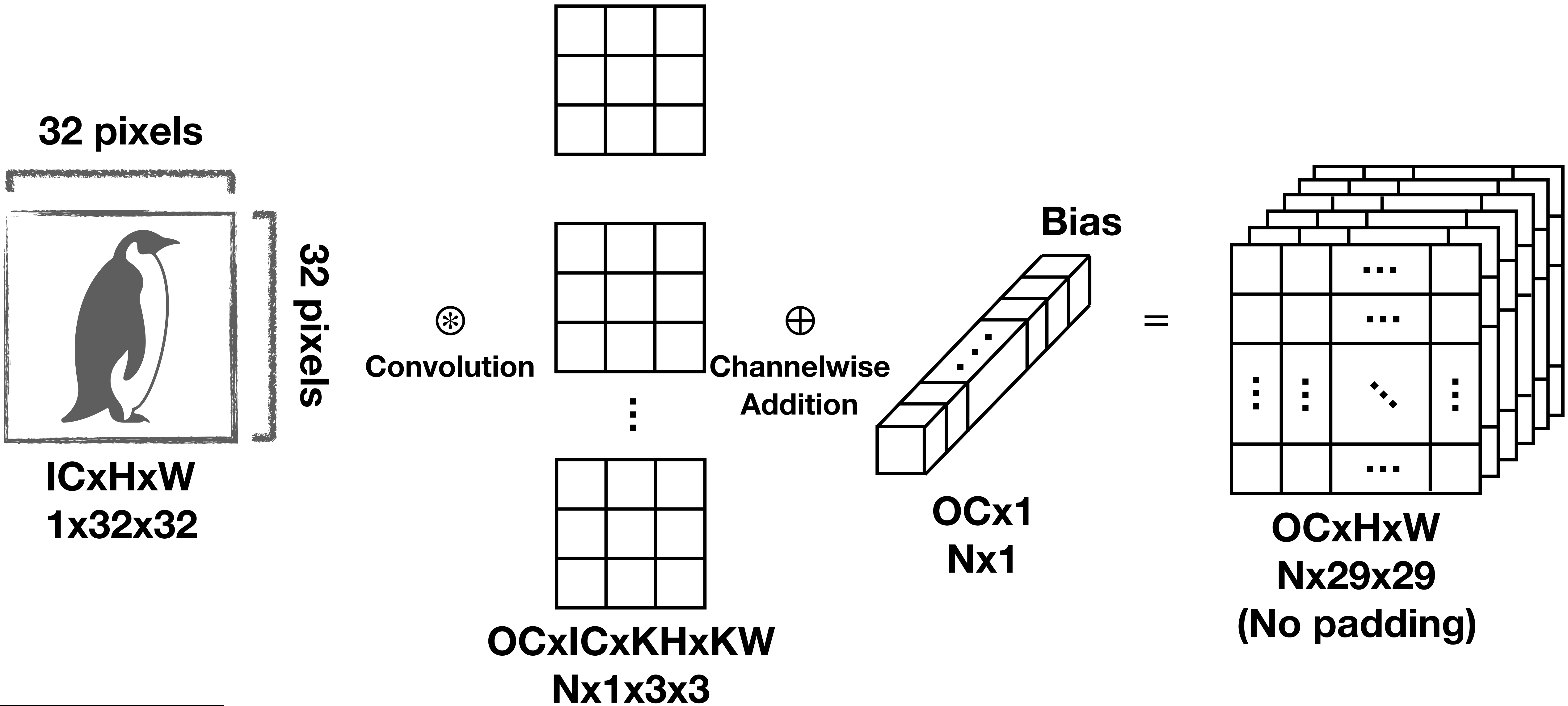
Code snippet

```
import torch.nn as nn


class Generator(nn.Module):
    def __init__(self):
        super(Generator, self).__init__()
        model = [nn.Linear(in_features=100, out_features=128), nn.ReLU(inplace=True)]
        model += [nn.Linear(in_features=128, out_features=256), nn.ReLU(inplace=True)]
        model += [nn.Linear(in_features=256, out_features=28 * 28), nn.Sigmoid()]
        self.model = nn.Sequential(*model)

        # "The generator nets used a mixture of rectifier linear activations and sigmoid activations, while the
        #  discriminator net used maxout activations." - Generative Adversarial Networks

    def forward(self, x):
        return self.model(x)


class Discriminator(nn.Module):
    def __init__(self):
        super(Discriminator, self).__init__()
        model = [Maxout(28 * 28, 256, dropout=False, k=5)]
        model += [Maxout(256, 128, dropout=True, k=5)]
        model += [nn.Linear(128, 1), nn.Sigmoid()]
        self.model = nn.Sequential(*model)

    def forward(self, x):
        return self.model(x)
```

For a full code, please visit
https://github.com/NoelShin/Deep-Learning-Bootcamp-with-PyTorch

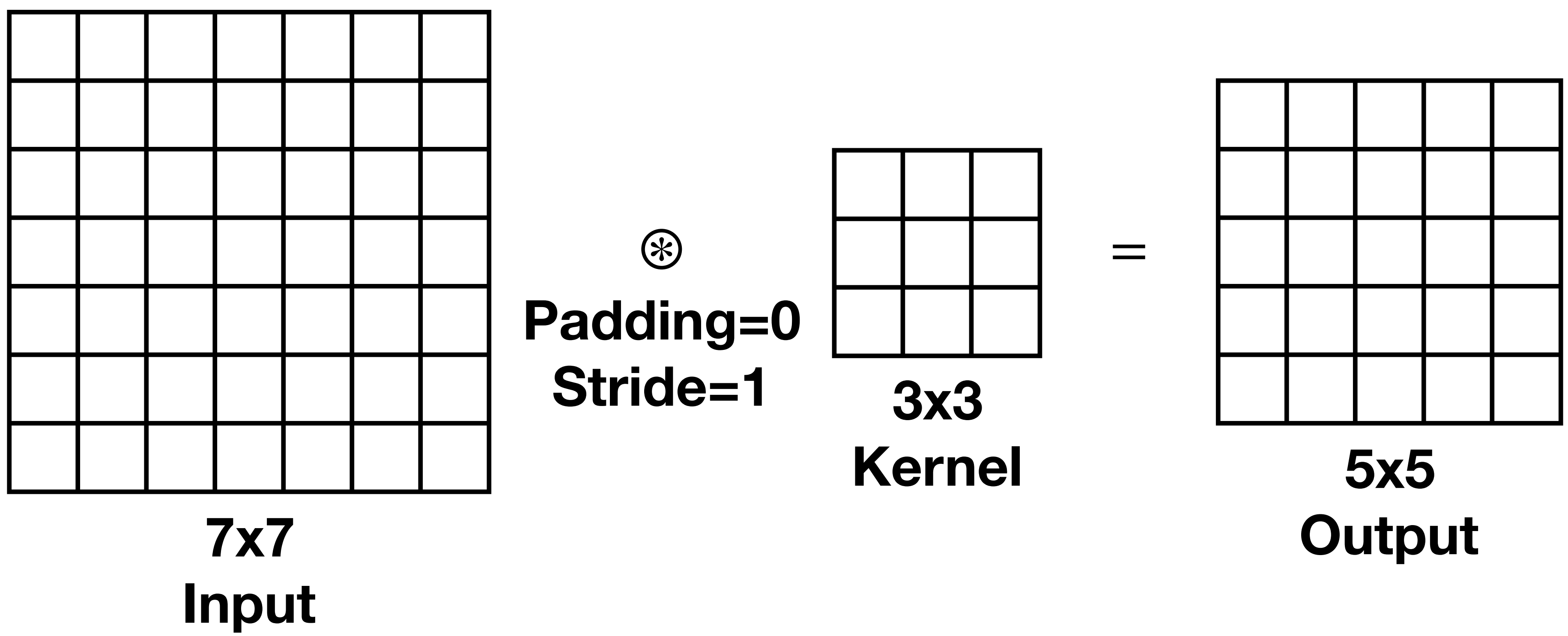
Appendix

Convolution



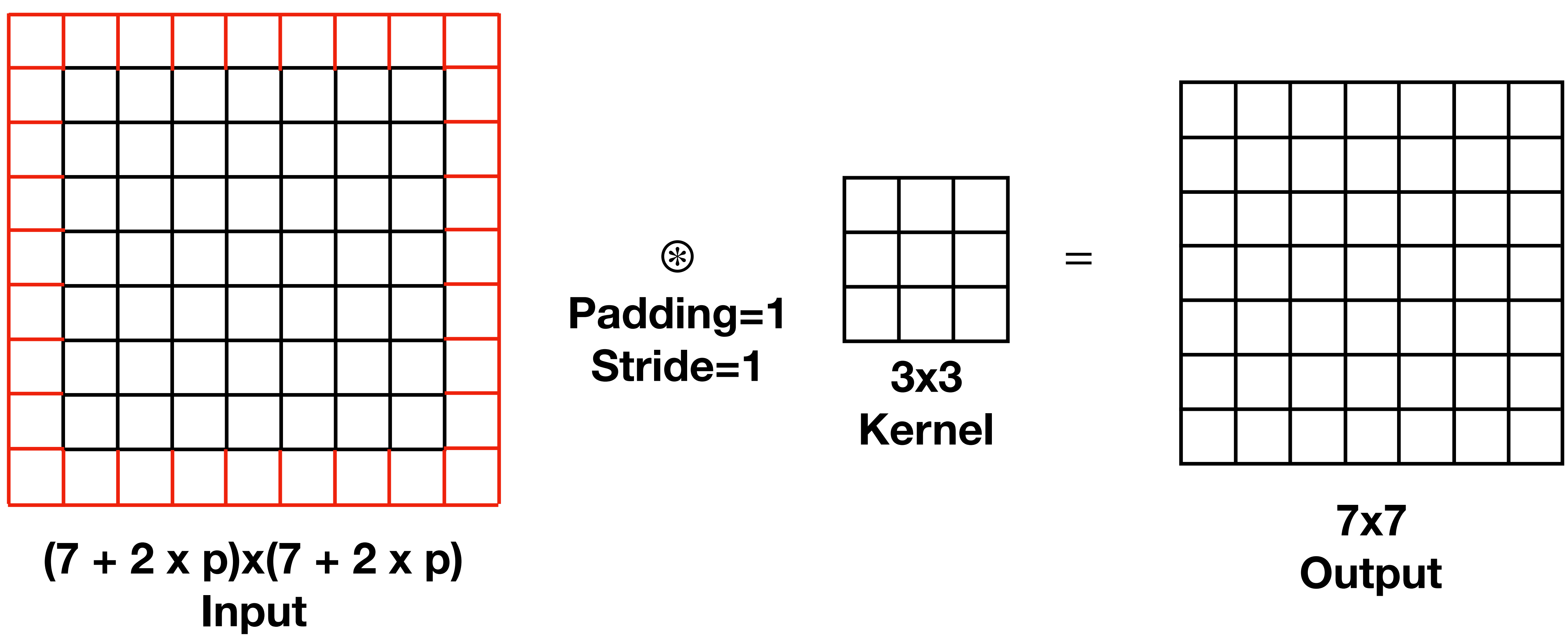
H: Height
W: Width
OC: Output Channel
IC: Input Channel
KH: Kernel Height
KW: Kernel Width

Convolution



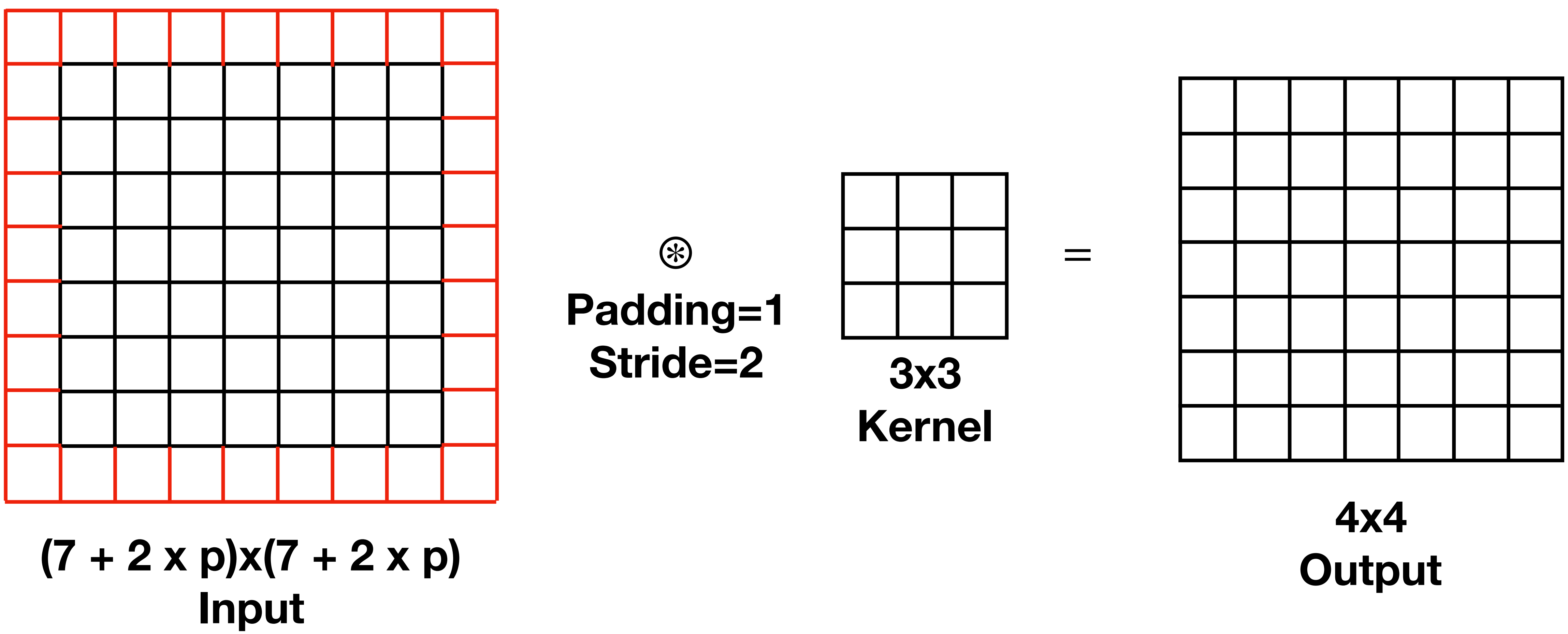
$$o = i - k + 1$$

Convolution



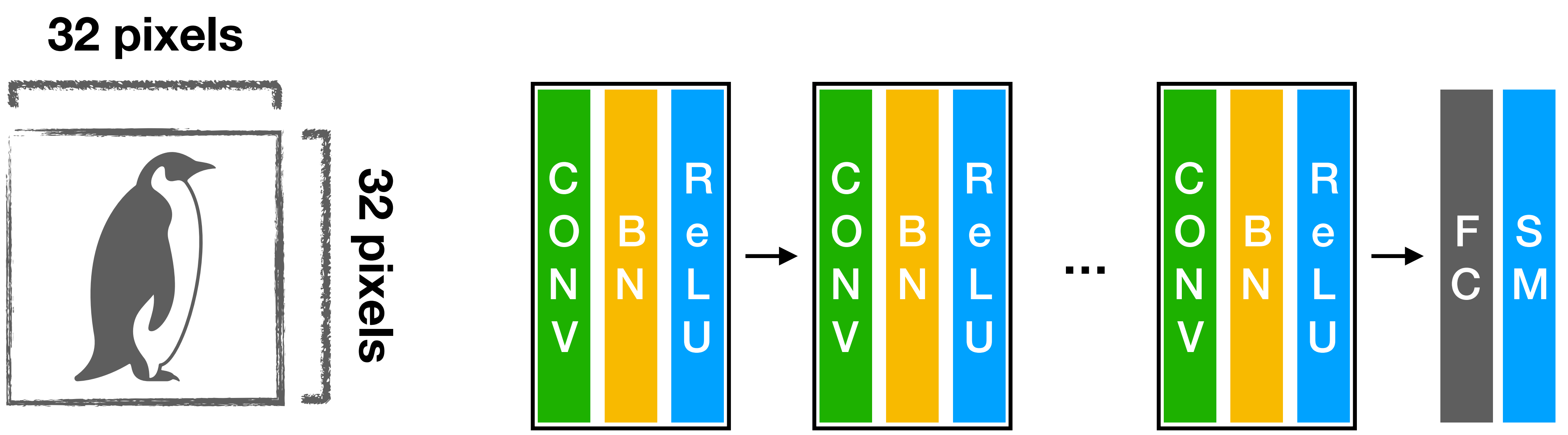
$$o = i - k + 2p + 1$$

Convolution



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

Convolutional neural network



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