Day 3

2023.10.23

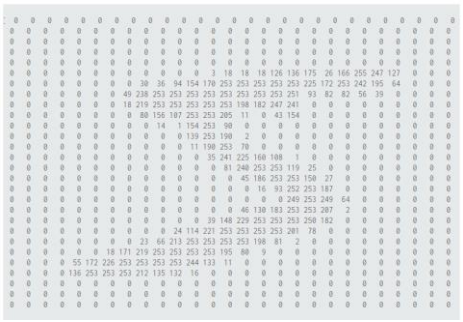
Image Classification

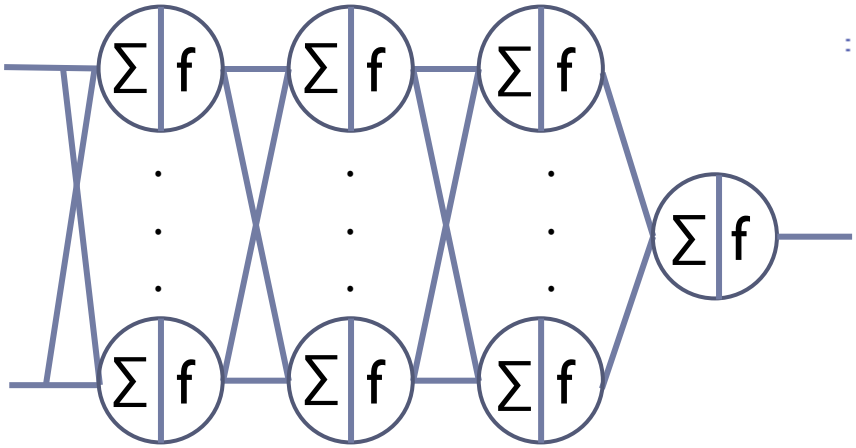
- **Mnist (image , label)**

Coding 전에 생각해 볼 것

RECAP

- 입력 데이터 : image(C , H , W)
- 출력 데이터 : class number (10개의 class)
- Optimizer: gradient descent method
- Loss function: cross entropy
- Model : Linear






```
CLASSES = {
    0: 'zero',
    1: 'one',
    2: 'two',
    3: 'three',
    4: 'four',
    5: 'five',
    6: 'six',
    7: 'seven',
    8: 'eight',
    9: 'nine'
}
```

x (input)

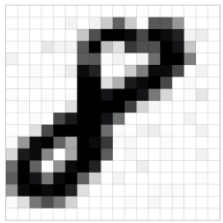
y (label)

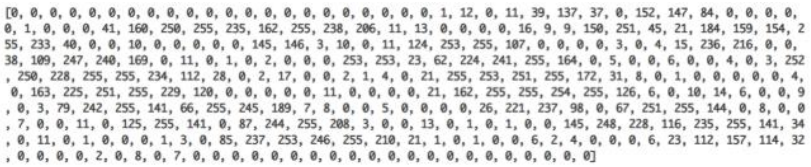
nn.Linear

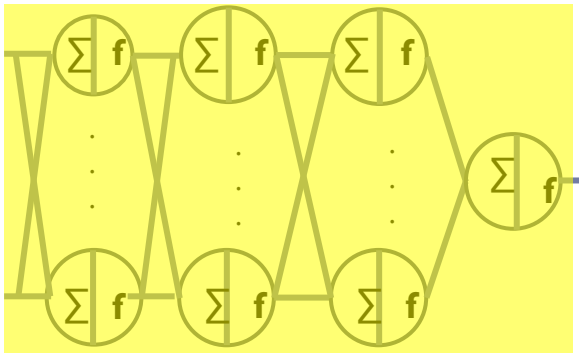
RECAP

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(784, 256)
        self.fc2 = nn.Linear(256, 128)
        self.fc3 = nn.Linear(128, 10)

    def forward(self, x):
        x = x.view(-1, 784)
        x = torch.sigmoid(self.fc1(x))
        x = torch.sigmoid(self.fc2(x))
        x = self.fc3(x)
        return x
```







1. Dataset

RECAP

- 입력 데이터
- 출력 데이터

```
trainset = datasets.MNIST(
    root      = '../data/',
    train     = True,
    download  = True,
    transform = transform
)
testset = datasets.MNIST(
    root      = '../data/',
    train     = False,
    download  = True,
    transform = transform
)
```

```
train_loader = data.DataLoader(
    dataset     = trainset,
    batch_size  = BATCH_SIZE, shuffle=True
)
test_loader = data.DataLoader(
    dataset     = testset,
    batch_size  = BATCH_SIZE, shuffle=True
)
```

2. Model

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(784, 256)
        self.fc2 = nn.Linear(256, 128)
        self.fc3 = nn.Linear(128, 10)

    def forward(self, x):
        x = x.view(-1, 784)
        x = torch.sigmoid(self.fc1(x))
        x = torch.sigmoid(self.fc2(x))
        x = self.fc3(x)
        return x
```

3. Optimizer

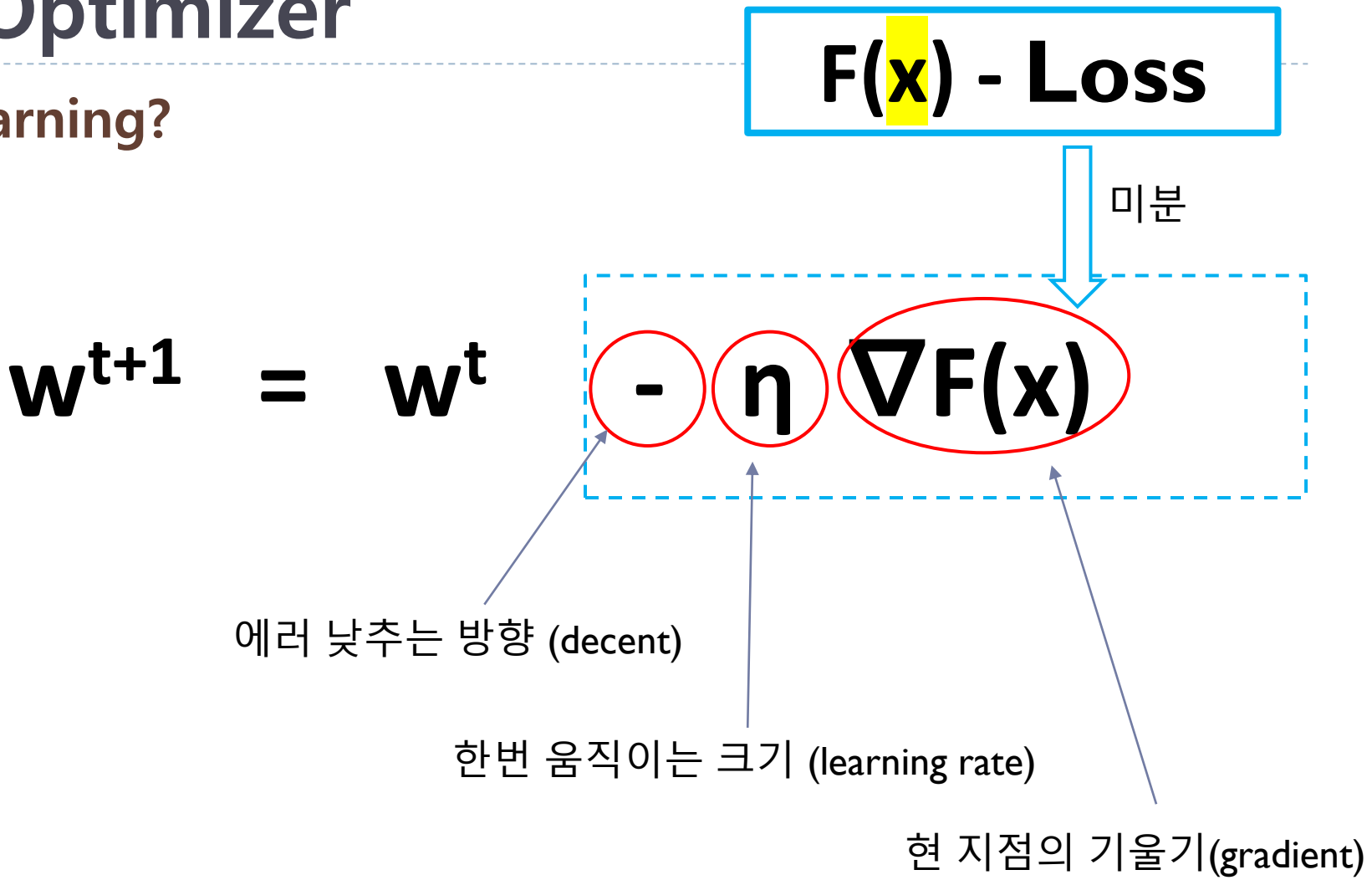
- **torch.optim**

Algorithms

`Adadelta`	Implements Adadelta algorithm.
`Adagrad`	Implements Adagrad algorithm.
`Adam`	Implements Adam algorithm.
`AdamW`	Implements AdamW algorithm.
	• • •
`RMSprop`	Implements RMSprop algorithm.
`Rprop`	Implements the resilient backpropagation algorithm.
`SGD`	Implements stochastic gradient descent (optionally with momentum).

3. Optimizer

- Learning?



3. Optimizer

- Learning?

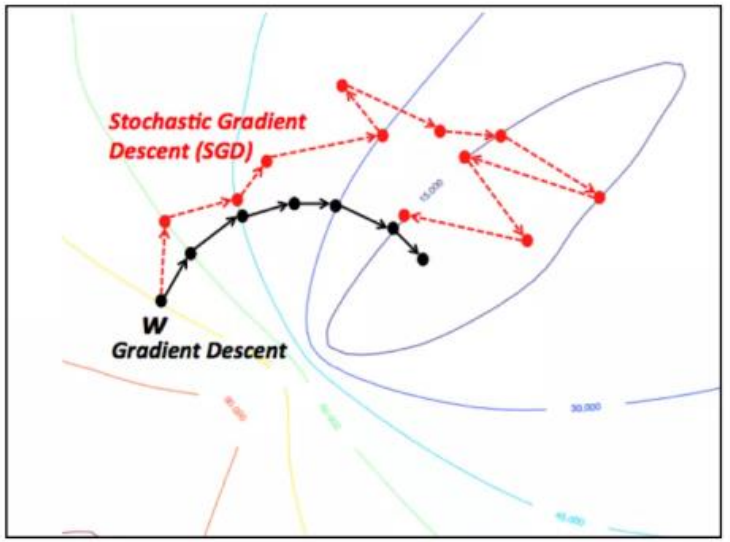
$$w^{t+1} = w^{t} - \eta \nabla F(x)$$

x 전부 => Gradient Decent

F(x) - Loss

x 나누어서 => Stochastic Gradient Decent

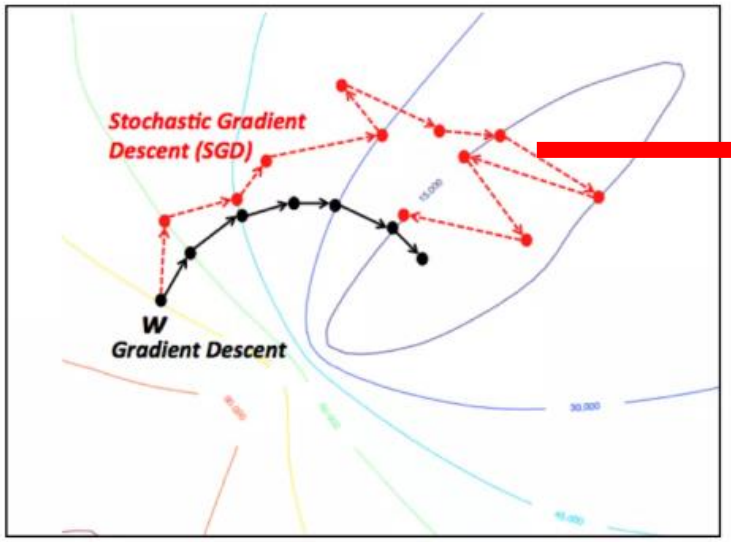
$F(x_1)$ - Loss



3. Optimizer

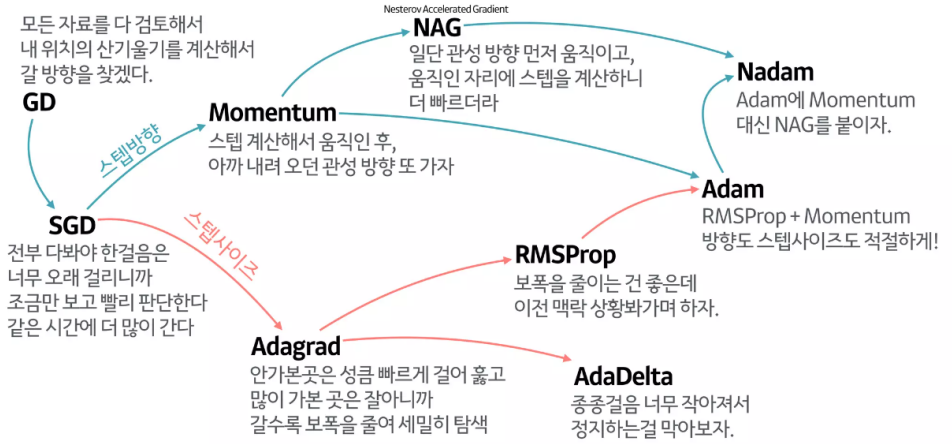
- Learning?

$$w^{t+1} = w^{t} - \eta \nabla F(x)$$



합쳐보면 ?

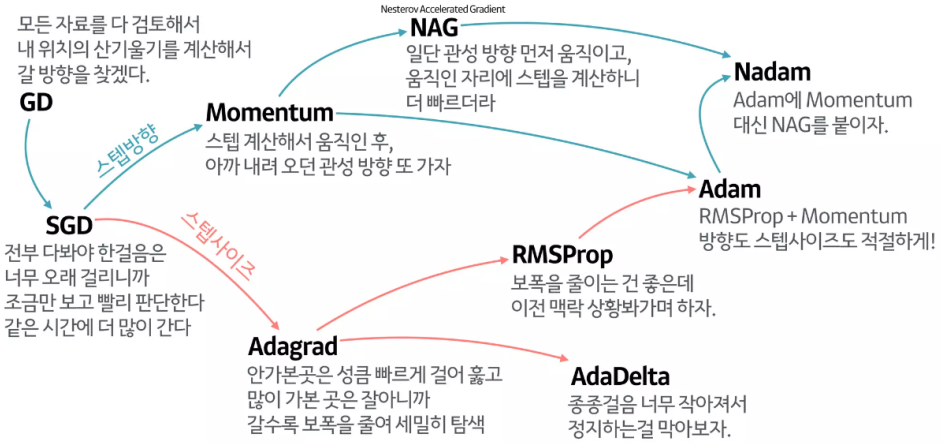
산 내려오는 작은 오솔길 잘찾기(Optimizer)의 발달 계보



https://www.slideshare.net/yongho/ss-79607172

3. Optimizer

산 내려오는 작은 오솔길 잘찾기(Optimizer)의 발달 계보



- **torch.optim**

```
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9)
```

```
optimizer = optim.Adam(model.parameters(), lr=0.01)
```

```
optimizer = optim.Adagrad(model.parameters(), lr=0.01)
```

https://www.slideshare.net/yongho/ss-79607172

4. Model

- **torchvision.models**

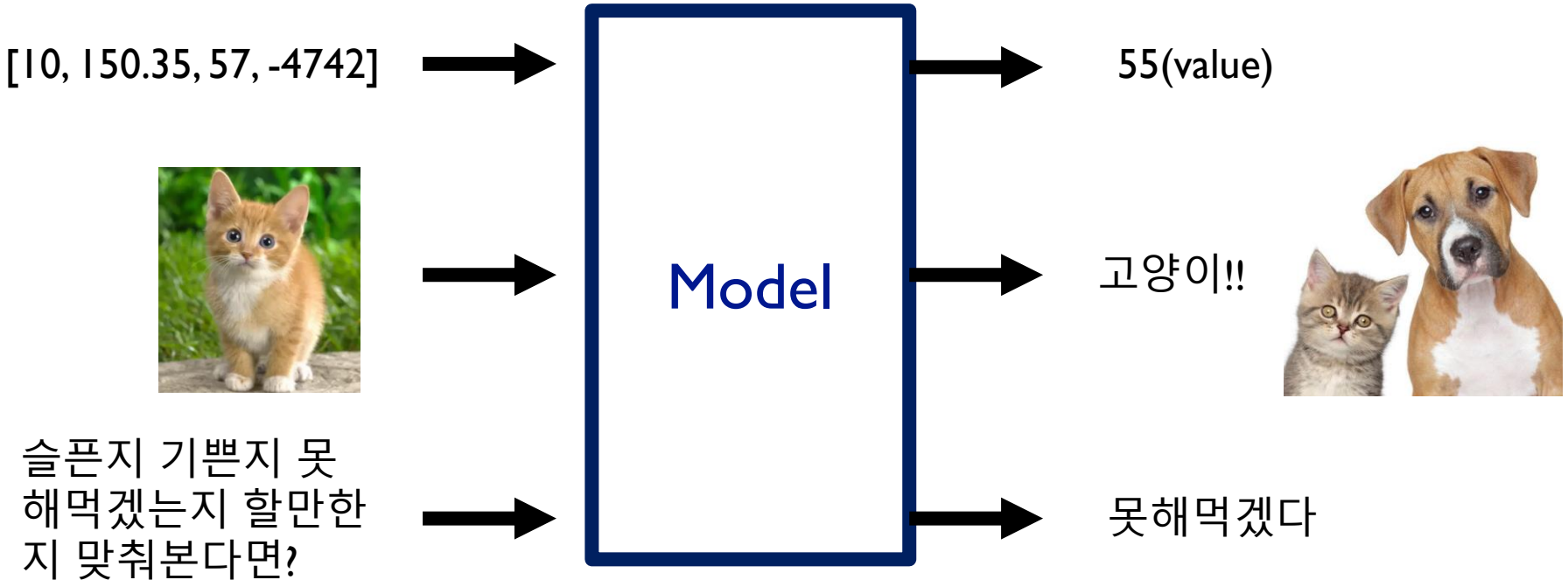
Classification

The following classification models are available, with or without pre-trained weights:

- AlexNet
- ConvNeXt
- DenseNet
- EfficientNet
- EfficientNetV2
- GoogLeNet
- Inception V3
- MaxVit
- MNASNet
- MobileNet V2
- MobileNet V3
- RegNet
- ResNet
- ResNeXt
- ShuffleNet V2
- SqueezeNet
- SwinTransformer
- VGG
- VisionTransformer
- Wide ResNet

Model에게 원하는 것?

- Regression -> nn.MSELoss()
- Binary classification -> nn.BCELoss()
- Multi class classification -> nn.CrossEntropyLoss()

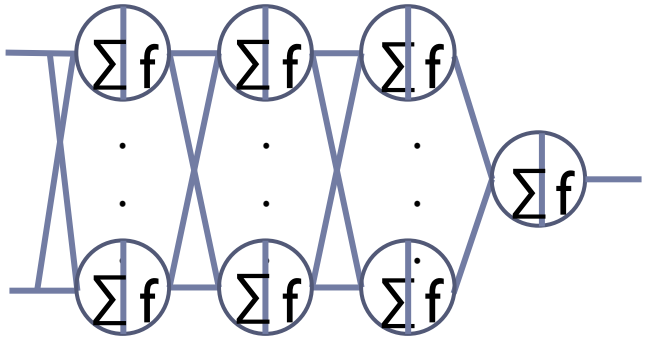


Coding 전에 생각해 볼 것

- 입력 데이터 : excel, csv
- 출력 데이터 : 숫자
- Optimizer: gradient descent method
- Loss function: cross entropy
- Model : Linear

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income
0	-114.31	34.19	15.0	5612.0	1283.0	1015.0	472.0	1.4936
1	-114.47	34.40	19.0	7650.0	1901.0	1129.0	463.0	1.8200
2	-114.56	33.69	17.0	720.0	174.0	333.0	117.0	1.6509
3	-114.57	33.64	14.0	1501.0	337.0	515.0	226.0	3.1917
4	-114.57	33.57	20.0	1454.0	326.0	624.0	262.0	1.9250
5	-114.58	33.63	29.0	1387.0	236.0	671.0	239.0	3.3438
6	-114.58	33.61	25.0	2907.0	680.0	1841.0	633.0	2.6768
7	-114.59	34.83	41.0	812.0	168.0	375.0	158.0	1.7083
8	-114.59	33.61	34.0	4789.0	1175.0	3134.0	1056.0	2.1782
9	-114.60	34.83	46.0	1497.0	309.0	787.0	271.0	2.1908

x (input)



median_house_value
66900.0
80100.0
85700.0
73400.0
65500.0
74000.0
82400.0
48500.0
58400.0
48100.0

y (label)

Question and Answer