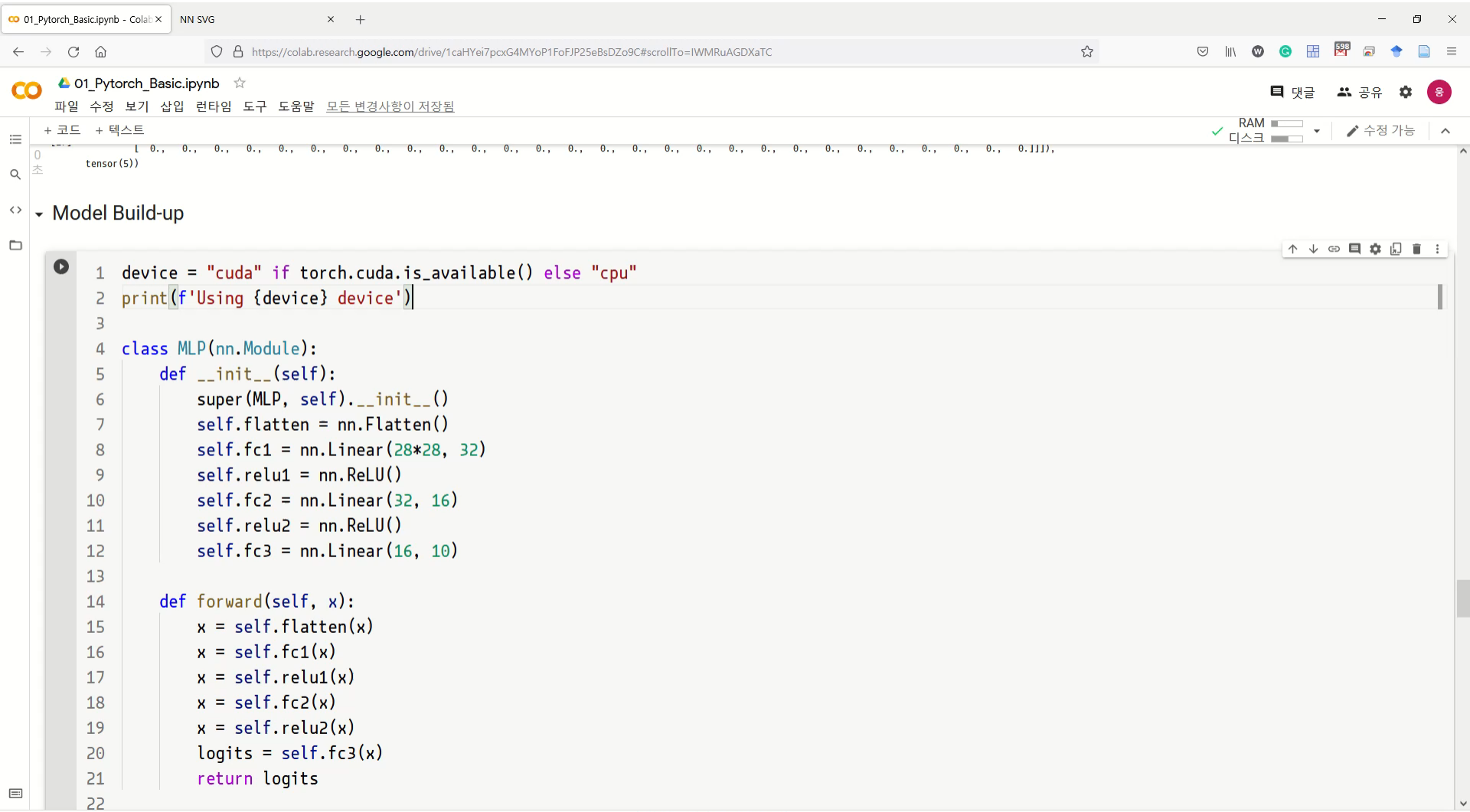
# Model Build-up



# Model Build-up

## ❑ Class 기반 Model 생성

```
device = "cuda" if torch.cuda.is_available() else "cpu"
print(f'Using {device} device')

class MLP(nn.Module):
    def __init__(self):
        super(MLP, self).__init__()
        self.flatten = nn.Flatten()
        self.fc1 = nn.Linear(28*28, 32)
        self.relu1 = nn.ReLU()
        self.fc2 = nn.Linear(32, 16)
        self.relu2 = nn.ReLU()
        self.fc3 = nn.Linear(16, 10)

    def forward(self, x):
        x = self.flatten(x)
        x = self.fc1(x)
        x = self.relu1(x)
        x = self.fc2(x)
        x = self.relu2(x)
        logits = self.fc3(x)
        return logits

model = MLP().to(device)
```

- 모델 파라미터 Handling

*model_name*.state_dict()

*model_name*.*layer_name*.weight.data

# Model Customizing

01_Pytorch_Basic.ipynb

## Model Customizing

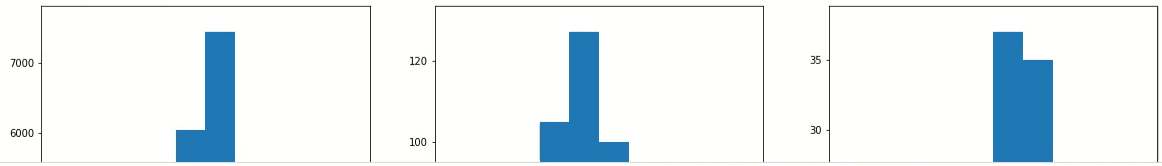
```
1 nn.init.xavier_normal_(model.fc1.weight)
2 nn.init.xavier_normal_(model.fc2.weight)
3 nn.init.xavier_normal_(model.fc3.weight)
```

```
Parameter containing:
tensor([[-3.0111e-01,  2.6399e-01,  5.5594e-02,  5.0123e-01, -1.6500e-01,  3.2857e-01, -6.1399e-01, -2.4941e-01, -7.1969e-02,  5.3716e-01, -2.0893e-01,  1.6986e-01,  2.4920e-02,  8.0231e-02, -3.2333e-01, -2.1658e-01],
        [ 2.5054e-01,  3.3832e-01,  2.2806e-01, -5.1387e-01, -1.5078e-01,  5.7204e-01,  1.4394e-01, -3.6062e-01,  2.3568e-01,  2.0382e-02,  9.6566e-03, -2.5667e-01, -2.7122e-02, -1.3801e-02,  7.5170e-02,  1.8423e-02],
        [ 3.0947e-02,  4.3716e-01, -3.2194e-01,  1.2892e-01,  4.3450e-01,  5.6060e-01, -3.7897e-01,  1.2094e-01,  2.9929e-01,  1.3558e-01,  1.9421e-01, -1.2224e-01, -3.4993e-01, -6.6392e-02, -5.0445e-01,  1.5313e-01],
        [ 1.3258e-02, -1.4364e-01,  1.6445e-01,  1.9393e-01,  2.8240e-01,  4.4780e-01, -1.3247e-01,  1.6790e-01,  5.0652e-02,  1.4477e-01, -1.2008e-01,  1.6248e-01,  3.4264e-01,  2.1732e-01, -6.5484e-02, -8.0802e-02],
        [ 4.7867e-01, -4.1669e-01,  2.4278e-02, -1.8108e-01,  3.5155e-01, -3.5653e-01,  7.8443e-01,  2.6168e-02,  1.8573e-01, -1.1255e-02,  4.4363e-01, -5.4933e-03,  2.1771e-01,  1.1453e-01,  1.3705e-01, -3.3701e-01],
        [-6.9377e-02, -4.8643e-03,  2.3746e-02,  8.8345e-02,  5.5763e-01,  6.0439e-01, -6.4256e-02, -6.9260e-01, -4.2262e-01, -3.4029e-01,  1.0045e-01,  1.5683e-01,  1.6218e-01, -2.6361e-01, -2.4386e-01, -2.2995e-01],
        [ 4.3846e-01, -3.3726e-03, -2.5488e-01,  1.6125e-01, -9.8897e-02,  2.9629e-01, -4.2134e-01,  1.9665e-01,  2.5928e-01, -2.0588e-04, -4.4740e-01,  2.8933e-01,  1.2472e-01, -7.4707e-02,  6.1438e-02, -5.8703e-02],
        [-7.3802e-01, -2.9539e-01,  2.6276e-01, -7.4484e-04, -3.1436e-01, -1.0018e+00, -2.4689e-01, -1.9656e-01,  3.8416e-01, -1.1696e-01,  1.7008e-01,  3.2269e-01,  5.2680e-01,  1.2714e-01, -2.6407e-02, -2.5601e-01],
        [-1.4705e-01,  1.7863e-01,  2.9889e-01, -1.3583e-03,  7.0985e-01,  6.2741e-02, -1.8675e-01,  2.8790e-01,  3.7941e-01, -5.2014e-01, -1.8100e-01, -2.5525e-01, -5.0758e-01,  4.4134e-02,  6.4788e-01,  2.0863e-02],
        [ 8.9196e-02,  2.5781e-01, -3.0809e-01,  5.7889e-01,  1.0903e-01,  6.8309e-02, -6.2974e-02, -4.5238e-01, -8.1845e-02, -4.9449e-01,  2.6834e-01,  3.0744e-01,  3.7816e-01,  1.7629e-01,  9.7783e-02, -1.7214e-01]], device='cuda:0', requires_grad=True)
```

[41]

```
1 fig, axs = plt.subplots(1, 3, figsize=(20, 10))
2 axs[0].hist(model.fc1.weight.data.view(-1).to('cpu').numpy())
3 axs[1].hist(model.fc2.weight.data.view(-1).to('cpu').numpy())
4 axs[2].hist(model.fc3.weight.data.view(-1).to('cpu').numpy())
```

```
(array([ 1.,  2.,  6., 17., 24., 37., 35., 21., 14.,  3.]),
 array([-1.0018213 , -0.823196  , -0.64457077, -0.46594548, -0.28732023,
        -0.10869497,  0.06993029,  0.24855556,  0.42718083,  0.60580605,
         0.78443134], dtype=float32),
 <a list of 10 Patch objects>)
```




0초 오전 6:42에 완료됨



VLSI & System Lab.

# Model Customizing

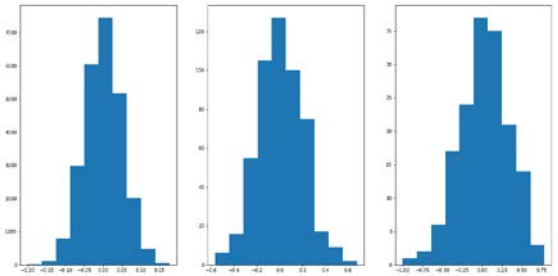
## ❑ 모델 가중치 초기화

```
1 torch.nn.init.xavier_normal_(model.fc1.weight)
2 torch.nn.init.xavier_normal_(model.fc2.weight)
3 torch.nn.init.xavier_normal_(model.fc3.weight)
```

dir(내부를 확인하고자 하는 변수)

ex) dir(torch.nn.init)

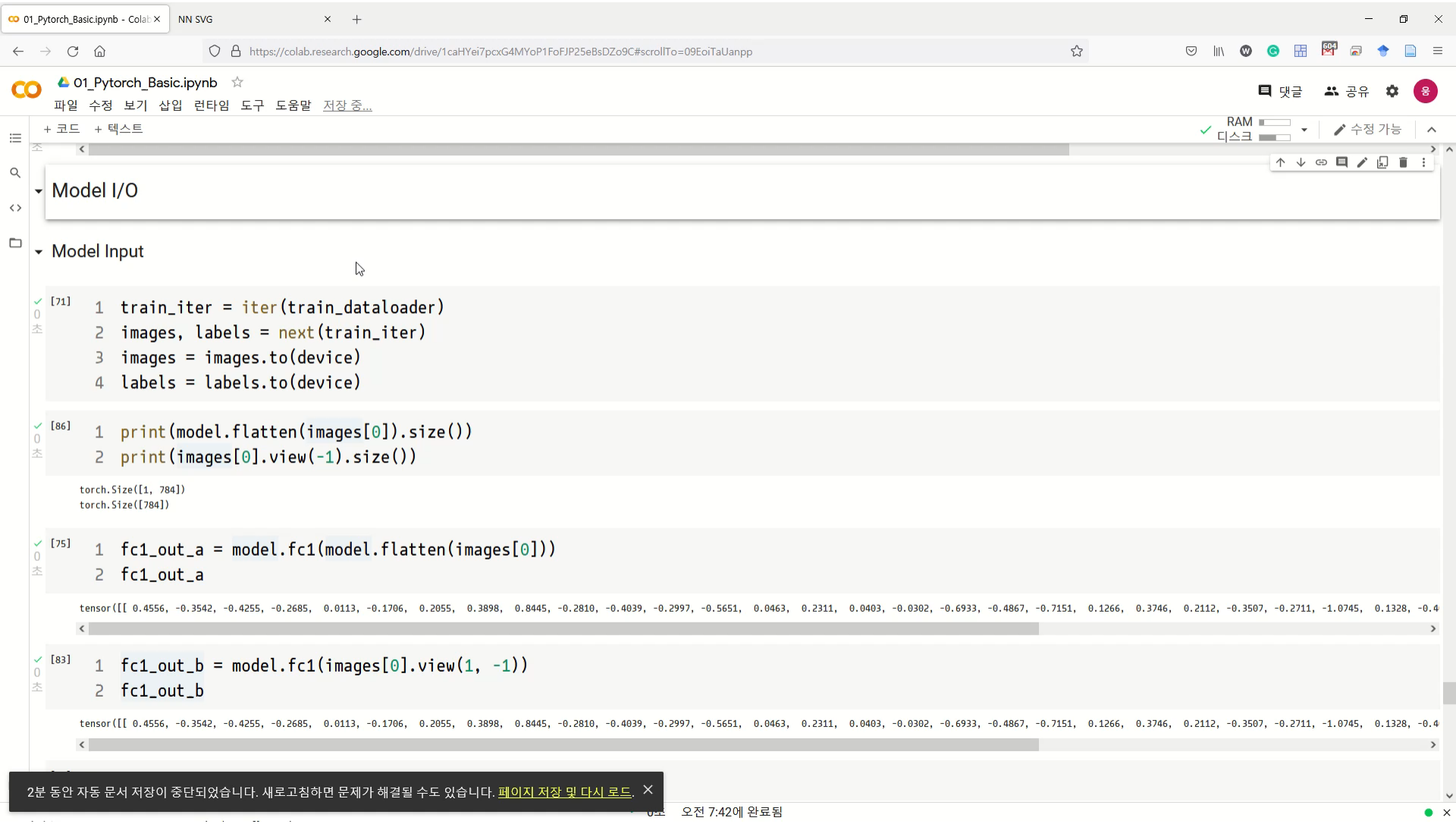
```
1 fig, axs = plt.subplots(1, 3, figsize=(20, 10))
2 axs[0].hist(model.fc1.weight.data.view(-1).to('cpu').numpy())
3 axs[1].hist(model.fc2.weight.data.view(-1).to('cpu').numpy())
4 axs[2].hist(model.fc3.weight.data.view(-1).to('cpu').numpy())
```



## ❑ torch.nn.Module 기반 모델 생성

```
1 fc1 = torch.nn.Linear(28*28, 32)
2 fc2 = torch.nn.Linear(32, 16)
3 fc3 = torch.nn.Linear(16, 10)
4 relu = torch.nn.ReLU()
5 model_with_seq = torch.nn.Sequential(fc1, relu, fc2, relu, fc3).to(device)
```

# Model I/O


01_Pytorch_Basic.ipynb - Colab
NN SVG
https://colab.research.google.com/drive/1caHYei7pcxG4MYoP1FoFJP25eBsDZo9C#scrollTo=09EoiTaUanpp
01_Pytorch_Basic.ipynb
파일 수정 보기 삽입 런타임 도구 도움말 저장 중...
댓글 공유
+ 코드 + 텍스트
RAM
디스크
수정 가능
Model I/O
Model Input
[71]
train_iter = iter(train_dataloader)
images, labels = next(train_iter)
images = images.to(device)
labels = labels.to(device)
[86]
print(model.flatten(images[0]).size())
print(images[0].view(-1).size())
torch.Size([1, 784])
torch.Size([784])
[75]
fc1_out_a = model.fc1(model.flatten(images[0]))
fc1_out_a
tensor([[ 0.4556, -0.3542, -0.4255, -0.2685, 0.0113, -0.1706, 0.2055, 0.3898, 0.8445, -0.2810, -0.4039, -0.2997, -0.5651, 0.0463, 0.2311, 0.0403, -0.0302, -0.6933, -0.4867, -0.7151, 0.1266, 0.3746, 0.2112, -0.3507, -0.2711, -1.0745, 0.1328, -0.4
[83]
fc1_out_b = model.fc1(images[0].view(1, -1))
fc1_out_b
tensor([[ 0.4556, -0.3542, -0.4255, -0.2685, 0.0113, -0.1706, 0.2055, 0.3898, 0.8445, -0.2810, -0.4039, -0.2997, -0.5651, 0.0463, 0.2311, 0.0403, -0.0302, -0.6933, -0.4867, -0.7151, 0.1266, 0.3746, 0.2112, -0.3507, -0.2711, -1.0745, 0.1328, -0.4
2분 동안 자동 문서 저장이 중단되었습니다. 새로고침하면 문제가 해결될 수도 있습니다. 페이지 저장 및 다시 로드.
오전 7:42에 완료됨

# Model I/O

## ❑ Model Input

```
train_iter = iter(train_dataloader)
images, labels = next(train_iter)
images = images.to(device)
labels = labels.to(device)
```

```
print(model.flatten(images[0]).size())
print(images[0].view(1,-1).size())
```

```
fc1_out_a = model.fc1(model.flatten(images[0]))
fc1_out_a
```

```
tensor([[ 0.4556, -0.3542, -0.4255, -0.2685,  0.0113, -0.1706,  0.2055,  0.3898,  0.8445,
```

```
fc1_out_b = model.fc1(images[0].view(1, -1))
fc1_out_b
```

```
tensor([[ 0.4556, -0.3542, -0.4255, -0.2685,  0.0113, -0.1706,  0.2055,  0.3898,  0.8445,
```

```
torch.equal(fc1_out_a, fc1_out_b)
```

```
True
```

## ❑ Model Output

```
y_direct = model(images[0])
y_direct
```

```
tensor([[ 0.0093,  0.1454,  0.1872,  0.0068, -0.1608,  0.1232,  0.0666,  0.1991, -0.0420,  0.1664]],
```

```
x = images[0].view(1, -1) # model.flatten(images[0])
y1 = model.relu1(model.fc1(x))
y2 = model.relu2(model.fc2(y1))
y = model.fc3(y2)
y
```

```
tensor([[ 0.0093,  0.1454,  0.1872,  0.0068, -0.1608,  0.1232,  0.0666,  0.1991, -0.0420,  0.1664]],
```

```
torch.equal(y_direct, y)
```

```
True
```

```
x = images[0].view(1, -1)
act1_direct = model.fc1(x)
act1_direct
```

```
tensor([[ 0.4556, -0.3542, -0.4255, -0.2685,  0.0113, -0.1706,  0.2055,  0.3898,  0.8445, -0.2810, -0.4039, -0.2997,
```

```
x = images[0].view(1, -1)
act1 = torch.matmul(x, model.fc1.weight.t()) + model.fc1.bias
act1
```

```
tensor([[ 0.4556, -0.3542, -0.4255, -0.2685,  0.0113, -0.1706,  0.2055,  0.3898,  0.8445, -0.2810, -0.4039, -0.2997,
```

```
torch.equal(act1_direct, act1)
```

```
True
```

# Loss Function

01_Pytorch_Basic.ipynb

## Define Loss Function

$$H(p, q) = -\sum_{x \in \mathcal{X}} p(x) \log q(x)$$

```
# y  = [1      0     0  0 ... 0]
# y_ = [0.9    0.05  0  0 ... 0]
loss_func = nn.CrossEntropyLoss()
```

[199]

```
print(model(images[0]))
print(model(images[0]).size())
print(labels[0])
print(labels[0].view(1))
```

```
tensor([[ 0.0505,  0.1701,  0.1653, -0.0931, -0.2277,  0.2645,  0.1575,  0.1905,  0.0016,  0.0455]], device='cuda:0', grad_fn=<AddmmBackward>)
torch.Size([1, 10])
tensor(5, device='cuda:0')
tensor([5], device='cuda:0')
```

[206]

```
def myCrossEntropyLoss(logits, labels):
    cee = 0
    for idx, logit in enumerate(logits):
        #print(f'IDX:{idx}, Logit:{logit}')
        y  = labels[idx]
        y_ = nn.functional.softmax(logit)[y]
        cee = cee - torch.log(y_)/len(logits)
    return cee
```

0초 오전 8:52에 완료됨

# Loss Function

## ❑ Classification → CrossEntropyLoss

- Equation $$H(p, q) = -\sum_{x \in \mathcal{X}} p(x) \log q(x)$$

- 'Built-in' vs 'Custom'

```
# y  = [0      1     0  0 ... 0]
# y_ = [0.9    0.05  0  0 ... 0] # model output > softmax
loss_func = nn.CrossEntropyLoss()
```

```
def myCrossEntropyLoss(logits, labels):
    cee = 0
    for idx, logit in enumerate(logits): # logits = [[~~~~~],..., [~~~~~]] logit = [~~~~~: 10 elements]
        #print(f'IDX:{idx}, Logit:{logit}')
        y  = labels[idx]
        y_ = nn.functional.softmax(logit)[y]
        cee = cee - torch.log(y_)/len(logits)
    return cee
```

```
loss_builtin = loss_func(model(images), labels)
loss_builtin
```

```
tensor(2.3562, device='cuda:0', grad_fn=<NllLossBackward>)
```

```
loss_custom = myCrossEntropyLoss(model(images), labels)
loss_custom
```

```
tensor(2.3562, device='cuda:0', grad_fn=<SubBackward0>)
```