Optimizer

01_Pytorch_Basic.ipynb - Colab | NN SVG

https://colab.research.google.com/drive/1caHYei7pcxG4MYoP1FoFJP25eBsDZo9C#scrollTo=kfYXf7m5p7lx

01_Pytorch_Basic.ipynb

파일 수정 보기 삽입 런타임 도구 도움말 모든 변경사항이 저장됨

+ 코드 + 텍스트

```python
dir(torch.optim)
```

```
['ASGD',
 'Adadelta',
 'Adagrad',
 'Adam',
 'AdamW',
 'Adamax',
 'LBFGS',
 'Optimizer',
 'RMSprop',
 'Rprop',
 'SGD',
 'SparseAdam',
 '__builtins__',
 '__cached__',
 '__doc__',
 '__file__',
 '__loader__',
 '__name__',
 '__package__',
 '__path__',
 '__spec__',
 '_functional',
 '_multi_tensor',
 'lr_scheduler',
 'swa_utils']
```

```python
loss_func = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=1e-2)
optimizer.zero_grad()
pred = model(images[0])
loss = loss_func(pred, labels[0].view(1))
```

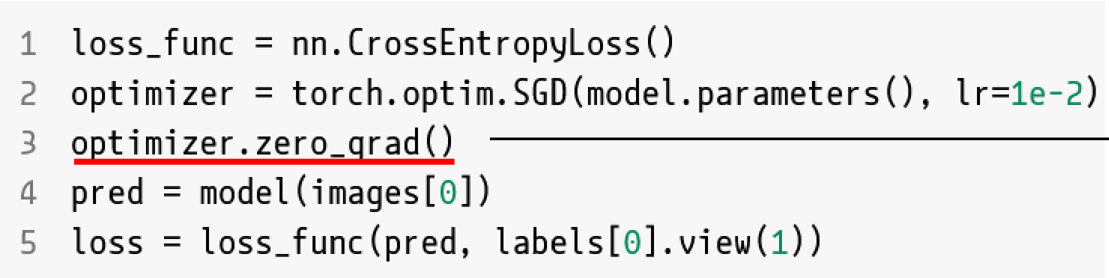
[278]

```python
model.fc3.weight.grad
```

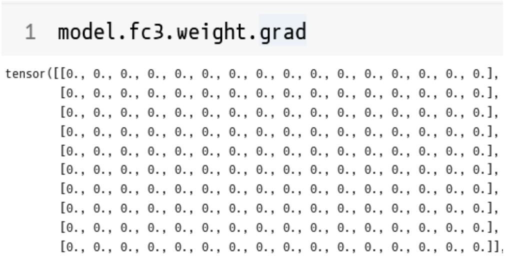
0초 오전 10:02에 완료됨

Optimizer

❑ Parameter Update Method



```
loss_func = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=1e-2)
optimizer.zero_grad()
pred = model(images[0])
loss = loss_func(pred, labels[0].view(1))
```



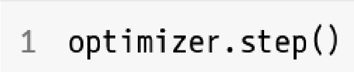
```
model.fc3.weight.grad
```

```
tensor([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]],
```

- Calculate gradient

```
loss.backward() # calculate gradient
```

- Update parameters



```
optimizer.step()
```

$$\theta = \theta - \eta \nabla_{\theta} J(\theta)$$

θ : Parameter η : Learning Rate $J(\theta)$: Loss Function

$\nabla_{\theta} J(\theta)$: Gradient of Loss

Training & Test (Inference)

01_Pytorch_Basic.ipynb - Colab | NN SVG

https://colab.research.google.com/drive/1caHYei7pcxG4MYoP1FoFJP25eBsDZo9C#scrollTo=iP2PIpJr_x7E

01_Pytorch_Basic.ipynb

파일 수정 보기 삽입 런타임 도구 도움말 모든 변경사항이 저장됨

+ 코드 + 텍스트

RAM 디스크 수정 가능

Train & Test (Inference)

```python
def train(dataloader, model, loss_func, optimizer):
    size = len(dataloader.dataset) # 60000
    for batch, (X, y) in enumerate(dataloader):
        X, y = X.to(device), y.to(device) # to(device): model, images, labels

        pred = model(X)
        loss = loss_func(pred, y)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if batch % 100 == 0:
            loss, current = loss.item(), batch * len(X)
            print(f"loss: {loss:>7f}  [{current:>5d}/{size:>5d}]")
```

[84]

```python
size = len(test_dataloader.dataset)
num_batches = len(test_dataloader)
size, num_batches
```

```
(10000, 157)
```

[72]

```python
def test(dataloader, model, loss_func):
    size = len(dataloader.dataset)
```

0초 오전 11:25에 완료됨

Training & Test (Inference)

❑ Train & Test Function

```python
def train(dataloader, model, loss_func, optimizer):
    size = len(dataloader.dataset) # 60000
    for batch, (X, y) in enumerate(dataloader):
        X, y = X.to(device), y.to(device) # to(device): model, images, labels from dataloader

        pred = model(X)
        loss = loss_func(pred, y)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if batch % 100 == 0:
            loss, current = loss.item(), batch * len(X)
            print(f"loss: {loss:>7f}  [{current:>5d}/{size:>5d}]")
```

```python
def test(dataloader, model, loss_func):
    size = len(dataloader.dataset)
    num_batches = len(dataloader)
    model.eval()
    test_loss, correct = 0, 0
    with torch.no_grad():
        for X, y in dataloader:
            X, y = X.to(device), y.to(device)
            pred = model(X)
            test_loss += loss_func(pred, y).item()
            correct += (pred.argmax(1) == y).type(torch.float).sum().item()
    test_loss /= num_batches
    correct /= size
    print(f"Test Error: \n Accuracy: {(100*correct):>0.1f}%, Avg loss: {test_loss:>8f} \n")
    return correct
```

```python
size = len(test_dataloader.dataset)
num_batches = len(test_dataloader)
size, num_batches
```

```
(10000, 157)
```

Training & Test (Inference)

❑ Epochs

```python
epochs = 5              # 60000 x 5
accuracy_list = []
for t in range(epochs):
    print(f"Epoch {t+1}\n-------------------------------")
    train(train_dataloader, model, loss_func, optimizer)
    accuracy = test(test_dataloader, model, loss_func)
    accuracy_list.append(accuracy)
print("Done!")
```

❑ 주의할 점

```python
size = len(test_dataloader.dataset)
num_batches = len(test_dataloader)
size, num_batches
```

```
(10000, 157)
```

```python
model = MLP().to(device)
optimizer = torch.optim.Adam(model.parameters(), lr=1e-2)
```

SGD

Model Save & Load

01_Pytorch_Basic.ipynb - Colab | NN SVG

https://colab.research.google.com/drive/1caHYei7pcxG4MYoP1FoFJP25eBsDZo9C#scrollTo=IREuVqcbbKL5

01_Pytorch_Basic.ipynb

파일 수정 보기 삽입 런타임 도구 도움말 모든 변경사항이 저장됨

+ 코드 + 텍스트

RAM 디스크 수정 가능

Model Save & Load

```
torch.save(model.state_dict(), "model.pth")
print("Saved PyTorch Model State to model.pth")
```

```
Saved PyTorch Model State to model.pth
```

[88]

```
model_loaded = MLP().to(device)
model_loaded.load_state_dict(torch.load("model.pth"))
```

```
<All keys matched successfully>
```

[89]

```
torch.equal(model.fc1.weight, model_loaded.fc1.weight)
```

```
True
```

[90]

```
model.fc3.weight
```

```
Parameter containing:
tensor([[-4.6450e-01, -3.4110e-01, -1.9350e-01,  1.9199e-01,  3.1358e-01,  1.8389e-01, -9.4993e-02,  8.6759e-02, -1.6160e-02,  1.8045e-01, -2.8536e-01,  2.4840e-01, -3.9225e-01, -4.9594e-02, -2.1789e-01, -1.2075e-01],
        [ 5.9857e-01, -1.5871e-01,  2.9557e-01, -2.0371e-01, -1.7768e-01,  8.6473e-02, -4.1335e-01, -4.6811e-01, -4.2830e-01,  4.4499e-01, -1.6437e-01,  1.1743e-03,  2.8366e-01, -7.7344e-02,  3.2878e-01,  6.6524e-02],
        [-5.6858e-02, -1.1927e-01,  3.8771e-01, -4.2081e-01, -2.7064e-01,  6.1097e-02,  1.1628e-01, -4.3551e-01,  7.5369e-02, -2.2377e-01, -1.0241e-01,  4.5000e-01,  1.9044e-01, -2.0555e-01, -1.4331e-01,  2.4256e-02],
        [ 3.4073e-02,  3.6927e-01,  1.7567e-01, -7.1020e-02, -2.9981e-01, -1.3702e-01, -1.7296e-01, -1.0990e-01, -2.2422e-01, -6.3069e-01, -1.8725e-01,  1.1592e-02,  3.2966e-01,  1.7707e-01, -2.7907e-01,  3.7975e-01],
        [-2.1025e-01, -8.2543e-02, -1.7876e-01, -4.9669e-01, -5.2815e-01, -3.9244e-02, -2.6792e-02,  1.0922e-01,  3.2637e-01,  4.0488e-01,  1.4062e-01,  1.1666e-01, -3.2505e-01,  8.1928e-02,  6.0206e-01, -5.2172e-01],
        [ 2.4592e-01,  2.3513e-01, -3.8245e-01,  1.6309e-04, -1.1499e-02,  6.6467e-02, -1.4550e-01,  3.9225e-01, -1.7764e-01, -6.2068e-01,  2.3243e-01, -1.4810e-01, -4.1585e-02,  1.1214e-01, -1.1802e-01,  2.9939e-01],
        [ 2.0766e-01, -2.6288e-01, -9.7598e-02, -1.2235e-01, -5.4985e-02,  2.0135e-01,  4.4118e-01,  3.3442e-01, -1.6643e-01, -3.9087e-02, -5.8649e-01, -1.4631e-01, -4.1724e-01, -1.1473e-01, -1.8625e-01, -3.1708e-01],
        [-3.4763e-01, -2.3208e-01,  1.1883e-01, -7.4893e-01, -3.1651e-02, -3.4006e-01, -2.8288e-01, -2.9991e-02, -3.1135e-01,  2.7602e-01,  4.1539e-01,  1.9563e-01,  3.2453e-01,  3.7029e-01, -3.9741e-02, -2.0471e-01],
        [ 9.2833e-02,  7.6047e-02,  2.5399e-01, -2.1284e-01, -2.7505e-01,  5.1622e-01, -2.4361e-01, -3.2365e-02, -2.5078e-01, -2.6291e-03,  9.5320e-02, -2.2341e-01, -2.3715e-01,  2.9223e-01, -8.1676e-03,  1.4268e-01],
        [-8.2773e-02, -2.5413e-01, -7.5921e-02, -4.4573e-01,  5.8676e-02, -1.6114e-01, -3.0765e-01, -4.3192e-02,  3.2832e-01, -3.8689e-02,  9.9853e-02, -3.1160e-01,  1.9427e-01,  3.1543e-01,  1.9952e-01,  2.5807e-01]], device='cuda:0', requires_grad=True)
```

```
model_loaded.fc3.weight
```

```
Parameter containing:
tensor([[-4.6450e-01, -3.4110e-01, -1.9350e-01,  1.9199e-01,  3.1358e-01,  1.8389e-01, -9.4993e-02,  8.6759e-02, -1.6160e-02,  1.8045e-01, -2.8536e-01,  2.4840e-01, -3.9225e-01, -4.9594e-02, -2.1789e-01, -1.2075e-01],
        [ 5.9857e-01, -1.5871e-01,  2.9557e-01, -2.0371e-01, -1.7768e-01,  8.6473e-02, -4.1335e-01, -4.6811e-01, -4.2830e-01,  4.4499e-01, -1.6437e-01,  1.1743e-03,  2.8366e-01, -7.7344e-02,  3.2878e-01,  6.6524e-02],
```

0초 오후 12:05에 완료됨

Model Save & Load

❑ Save

```
torch.save(model.state_dict(), "model.pth")
print("Saved PyTorch Model State to model.pth")
```

❑ Load

```
model_loaded = MLP().to(device)
model_loaded.load_state_dict(torch.load("model.pth"))
```

❑ Equivalent Check

```
torch.equal(model.fc2.bias, model_loaded.fc2.bias)
```

```
True
```