PyTorch 講座 7

肥田　歩華

前回のおさらい

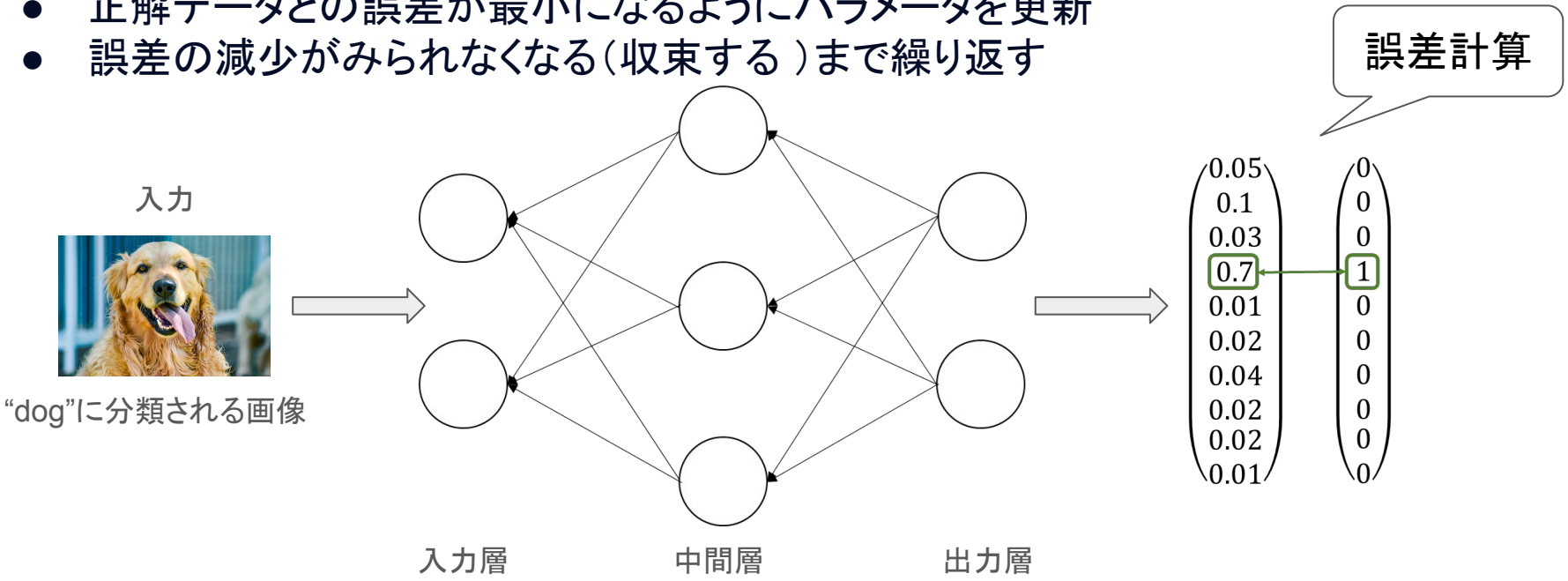
- 画像の前処理
  - データの正規化
  - データ拡張

- データローダー
  - バッチ化

PyTorch - トレーニング -

どうやって学習するか　-誤差逆伝播法-

- モデルに訓練データを入力し、予測結果を受け取る
- 予測結果と正解ラベルを比較・誤差を計算
- 正解データとの誤差が最小になるようにパラメータを更新
- 誤差の減少がみられなくなる（収束する ）まで繰り返す

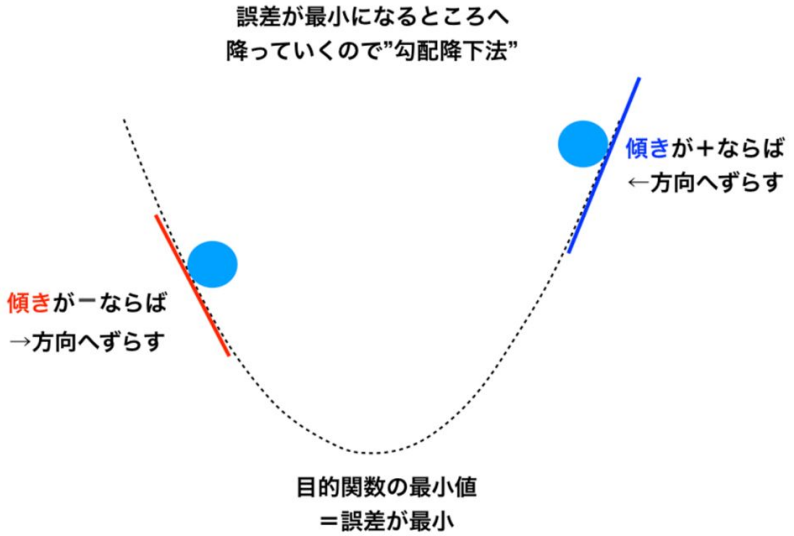


最適化アルゴリズム (Optimizer)

損失関数の値を最小化するようなパラメータを見つけ
最適化(Optimization)を行う機構

代表例

- SGD（確率的勾配降下法）
- Momentum
- Adagrad
- RMSProp
- Adam



PyTorch での実装

PyTorch では torch.optim で実装されている

SGDを使う場合

```
CLASS  torch.optim.SGD(params, lr=<required parameter>, momentum=0,
       dampening=0, weight_decay=0, nesterov=False, *, maximize=False,
       foreach=None, differentiable=False)  [SOURCE]
```

主に使われる引数は
- params（必須）：モデルのパラメータ
- lr（必須）　　：学習率

学習率：
1回の学習でどれだけ重みを更新するかを表すパラメータ

利用例

optimizer.py U

codes > 07 > example > optimizer.py > ...

```python
import torch
import torch.nn as nn
import torch.optim as optim

# モデルを定義
class MyModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.linear = nn.Linear(1, 1)

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

学習を重ねるごとに
誤差が小さくなっていく

```
Epoch: 10 , Loss: 6.2801
Epoch: 20 , Loss: 4.2162
Epoch: 30 , Loss: 2.8549
Epoch: 40 , Loss: 1.9566
Epoch: 50 , Loss: 1.3637
Epoch: 60 , Loss: 0.9722
Epoch: 70 , Loss: 0.7136
Epoch: 80 , Loss: 0.5428
Epoch: 90 , Loss: 0.4298
Epoch: 100, Loss: 0.3552
```

optimizer.py U

codes > 07 > example > optimizer.py > ...

```python
if __name__=="__main__":
    torch.manual_seed(0)
    x = torch.randn(100, 1)
    y = 3 * x + 0.5 * torch.randn(100, 1)

    model = MyModel()

    # 損失関数（平均二乗誤差）
    criterion = nn.MSELoss()

    # 最適化関数（SGD）
    optimizer = optim.SGD(model.parameters(), lr=0.01)

    # 学習
    epochs = 100
    losses = []
    for epoch in range(epochs):
        model.train()

        # パラメータの勾配を初期化
        optimizer.zero_grad()

        # 順伝搬・逆伝搬・パラメータ更新
        outputs = model(x)
        loss = criterion(outputs, y)
        loss.backward()
        optimizer.step()

        losses.append(loss.item())

        if (epoch + 1) % 10 == 0:
            print(f'Epoch: {epoch + 1: <3}, Loss: {loss.item():.4f}')
```

実践編

実際に画像分類をやってみよう

学習の流れ

- データセットの読み込み
- モデルの作成
- データローダーからデータを受け取る
- 受け取ったデータをCNNモデルに入力
- モデルからの出力を受け取る
- 出力を使って誤差を算出
- 誤差を使って重みを更新

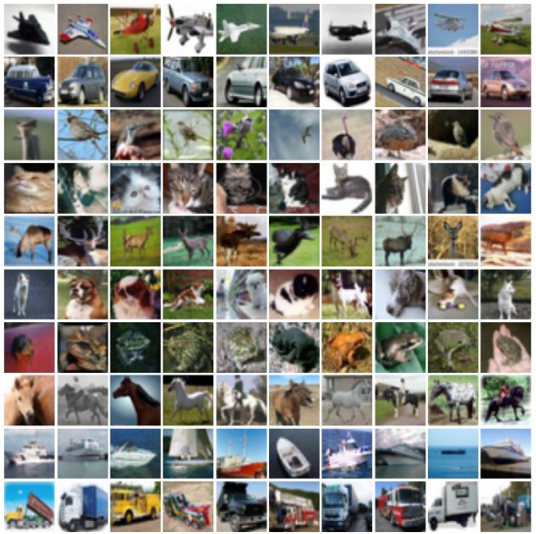
データセット　CIFAR-10

- 10クラスのカラー画像からなるデータセット

- 5万枚の訓練データ
  1万枚のテストデータ

- 1つのデータサイズは3x32x32

airplane : 0
automobile : 1
bird : 2
cat : 3
deer : 4
dog : 5
frog : 6
horse : 7
ship : 8
truck : 9



データセットの読み込み

dataset.py

pytorch-training > codes > 07 > example > dataset.py > ...

```python
from torchvision import transforms, datasets

def cifar_datasets():
    # データセットの読み込み
    train_data = datasets.CIFAR10(root="../",
                                  train=True,
                                  transform=transforms.ToTensor(),
                                  download=True)

    test_data = datasets.CIFAR10(root="../",
                                 train=False,
                                 transform=transforms.ToTensor(),
                                 download=True)

    return train_data, test_data

if __name__=="__main__":
    train_data, test_data = cifar_datasets()

    # 訓練データの1枚目を取得
    image, label = train_data[0]
    print("image size: ", image.size())
    print("image label: ", label)
```

訓練データの1枚目は
データサイズ：3x32x32
ラベル　　：6

```
image size:  torch.Size([3, 32, 32])
image label:  6
```

モデルの作成

model.py

codes > 07 > example > model.py > CNN > __init__

```python
import torch.nn as nn

class CNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 8, kernel_size=3, stride=1, padding=1)
        self.conv2 = nn.Conv2d(8, 16, kernel_size=3, stride=1, padding=1)
        self.l1 = nn.Linear(8 * 8 * 16, 128)
        self.l2 = nn.Linear(128, 64)
        self.l3 = nn.Linear(64, 10)
        self.act = nn.ReLU()
        self.pool = nn.MaxPool2d(2, 2)

    def forward(self, x):
        x = self.pool(self.act(self.conv1(x)))
        x = self.pool(self.act(self.conv2(x)))
        x = x.view(x.size()[0], -1)
        x = self.act(self.l1(x))
        x = self.act(self.l2(x))
        x = self.l3(x)
        return x

if __name__=="__main__":
    model = CNN()
    print(model)
```

今回はCNNを使用

```
CNN(
  (conv1): Conv2d(3, 8, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2): Conv2d(8, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (l1): Linear(in_features=1024, out_features=128, bias=True)
  (l2): Linear(in_features=128, out_features=64, bias=True)
  (l3): Linear(in_features=64, out_features=10, bias=True)
  (act): ReLU()
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
)
```

データローダーからデータを受け取る

dataloader.py

pytorch-training > codes > 07 > example > dataloader.py > ...

```python
from torch.utils.data import DataLoader
from dataset import cifar_datasets

# データローダーからデータを受け取る
train_data, test_data = cifar_datasets()
train_loader = DataLoader(train_data, batch_size=64, shuffle=True)
test_loader = DataLoader(test_data, batch_size=64, shuffle=False)

if __name__=="__main__":
    # イテレータを作成
    train_iter = iter(train_loader)

    # 次のバッチを取得
    image, labels = next(train_iter)

    print("Image shape: ", image.shape) # [batch_size, channels, height, width]
    print("Labels: ", labels)
```

```
Image shape:  torch.Size([64, 3, 32, 32])
Labels:  tensor([8, 9, 8, 4, 5, 4, 9, 6, 3, 2, 3, 6, 1, 8, 6, 0, 5, 9, 7, 8, 6, 5, 2, 7,
        2, 7, 1, 0, 2, 3, 5, 0, 2, 7, 9, 2, 1, 5, 7, 9, 9, 4, 0, 2, 4, 5, 6, 6,
        1, 3, 2, 1, 0, 4, 5, 5, 0, 0, 9, 3, 3, 2, 6, 1])
```

データをモデルに入力

モデルにデータを入力し予測結果を受け取る

最初のバッチを入力したときの
モデルの出力サイズ

```
Train Output Size: torch.Size([64, 10])
Test Output Size: torch.Size([64, 10])
```

エポック数：訓練データの学習回数

つまり・・・
訓練データを一通り使用した状態が 1エポック

predict.py

pytorch-training > codes > 07 > example > predict.py > ...

```python
import torch
from torch.utils.data import DataLoader

from dataset import cifar_datasets
from model import CNN

# データローダーからデータを受け取る（参考：dataloader.py）
train_data, test_data = cifar_datasets()
train_loader = DataLoader(train_data, batch_size=64, shuffle=True)
test_loader = DataLoader(test_data, batch_size=64, shuffle=False)

# モデルの定義
model = CNN()

# 各バッチにおけるモデルの出力サイズのリスト
train_outputs_list = []
test_outputs_list = []

for epoch in range(1):
    # 訓練データでモデルの出力
    model.train()
    for images, labels in train_loader:
        train_outputs = model(images) # [batch_size, label]
        train_outputs_list.append(train_outputs)

    # テストデータでモデルの出力
    model.eval()
    with torch.no_grad():
        for images, labels in test_loader:
            test_outputs = model(images) # [batch_size, label]
            test_outputs_list.append(test_outputs)

if __name__=="__main__":
    # 1番目のバッチにおけるモデルの出力サイズ
    print("Train Output Size:", train_outputs_list[0].shape)
    print("Test Output Size:", test_outputs_list[0].shape)
```

出力を使って誤差を算出

loss.py

pytorch-training > codes > 07 > example > loss.py > ...

```python
import torch
import torch.nn as nn
from torch.utils.data import DataLoader

from dataset import cifar_datasets
from model import CNN

# データローダーからデータを受け取る（参考：dataloader.py）
train_data, test_data = cifar_datasets()
train_loader = DataLoader(train_data, batch_size=64, shuffle=True)
test_loader = DataLoader(test_data, batch_size=64, shuffle=False)

# モデルの定義
model = CNN()

# 損失関数
criterion = nn.CrossEntropyLoss()

for epoch in range(1):
    train_loss = 0
    val_loss = 0

    model.train()
    for images, labels in train_loader:
        train_outputs = model(images) # [batch_size, label]

        # 出力を使って誤差を算出
        loss = criterion(train_outputs, labels)

        # 各バッチで計算されたlossを累積
        train_loss += loss.item()

    # 1エポック内のlossの平均
    avg_train_loss = train_loss / len(train_loader)
```

```python
    model.eval()
    with torch.no_grad():
        for images, labels in test_loader:
            test_outputs = model(images) # [batch_size, label]
            loss = criterion(test_outputs, labels)
            val_loss += loss.item()
        avg_val_loss = val_loss / len(test_loader)

if __name__=="__main__":
    print("Train Loss: ", avg_train_loss)
    print("Validation Loss: ", avg_val_loss)
```

```
Train Loss:  2.3043234982453953
Validation Loss:  2.3043630593901225
```

Optimizerを使っていないため、
モデルの重みは更新されず学習は行われない

モデルはランダムな重みで初期化され、
予測結果に対する損失を計算しているだけ

誤差を使って重みを更新

backpropagation.py

pytorch-training > codes > 07 > example > backpropagation.py > ...

```python
import torch
import torch.nn as nn
from torch.utils.data import DataLoader
import torch.optim as optim

from dataset import cifar_datasets
from model import CNN

# データローダーからデータを受け取る（参考：dataloader.py）
train_data, test_data = cifar_datasets()
train_loader = DataLoader(train_data, batch_size=64, shuffle=True)
test_loader = DataLoader(test_data, batch_size=64, shuffle=False)

# モデルの定義
model = CNN()

# 損失関数
criterion = nn.CrossEntropyLoss()

# 最適化関数
learning_rate = 0.01
optimizer = optim.SGD(model.parameters(), lr=learning_rate)

```

```
Train Loss:  2.3016982404777155
Validation Loss:  2.2996954447144917
```

backpropagation.py

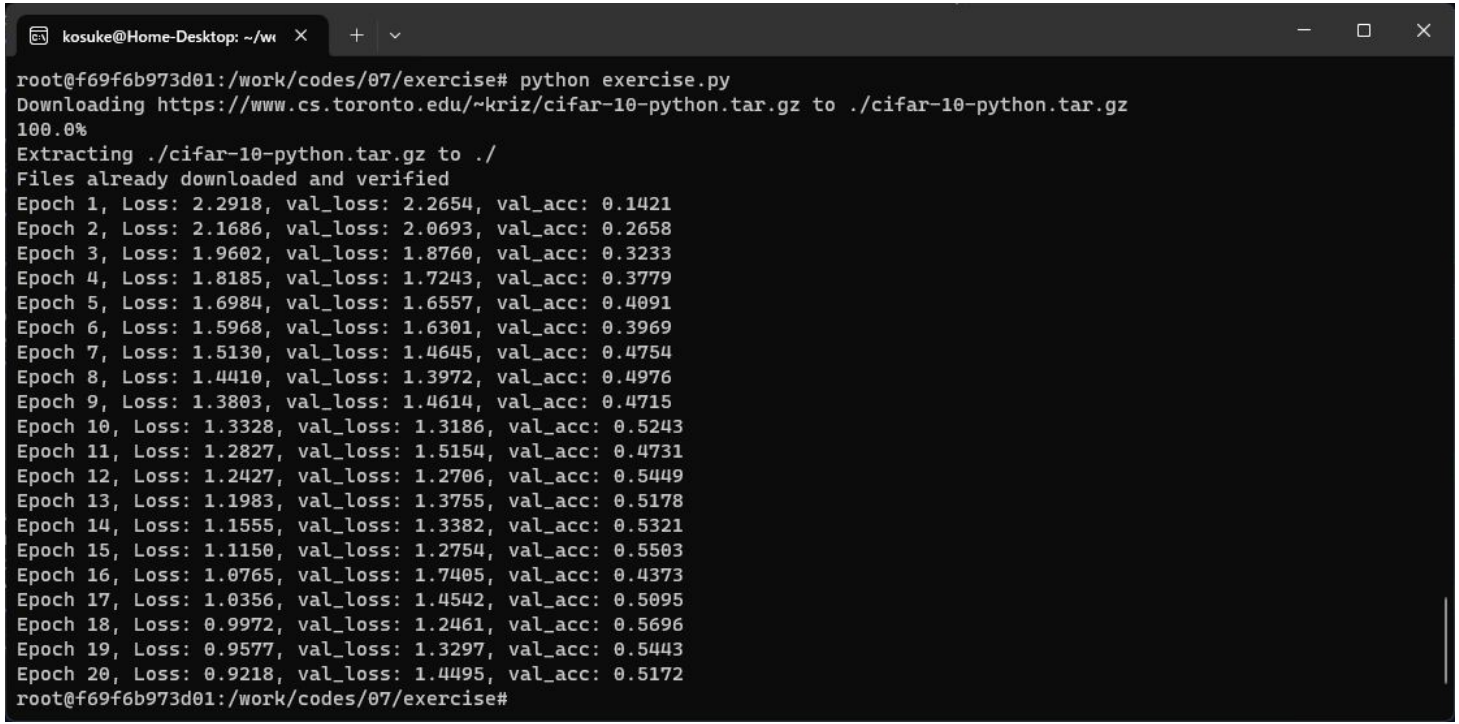
pytorch-training > codes > 07 > example > backpropagation.py > ...

```python
for epoch in range(1):
    train_loss = 0
    val_loss = 0

    model.train()
    for images, labels in train_loader:
        # 勾配を初期化
        optimizer.zero_grad()

        train_outputs = model(images) # [batch_size, label]
        loss = criterion(train_outputs, labels)
        train_loss += loss.item()

        # 誤差逆伝播
        loss.backward()

        # 重みを更新
        optimizer.step()

    avg_train_loss = train_loss / len(train_loader)

    model.eval()
    with torch.no_grad():
        for images, labels in test_loader:
            test_outputs = model(images) # [batch_size, label]
            loss = criterion(test_outputs, labels)
            val_loss += loss.item()
        avg_val_loss = val_loss / len(test_loader)

if __name__=="__main__":
    print("Train Loss: ", avg_train_loss)
    print("Validation Loss: ", avg_val_loss)
```

演習

- モデルを学習させ誤差が減少するのに伴い正答率が向上するのを出力し確認してください

解答例


kosuke@Home-Desktop: ~/w
root@f69f6b973d01:/work/codes/07/exercise# python exercise.py
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./cifar-10-python.tar.gz
100.0%
Extracting ./cifar-10-python.tar.gz to ./
Files already downloaded and verified
Epoch 1, Loss: 2.2918, val_loss: 2.2654, val_acc: 0.1421
Epoch 2, Loss: 2.1686, val_loss: 2.0693, val_acc: 0.2658
Epoch 3, Loss: 1.9602, val_loss: 1.8760, val_acc: 0.3233
Epoch 4, Loss: 1.8185, val_loss: 1.7243, val_acc: 0.3779
Epoch 5, Loss: 1.6984, val_loss: 1.6557, val_acc: 0.4091
Epoch 6, Loss: 1.5968, val_loss: 1.6301, val_acc: 0.3969
Epoch 7, Loss: 1.5130, val_loss: 1.4645, val_acc: 0.4754
Epoch 8, Loss: 1.4410, val_loss: 1.3972, val_acc: 0.4976
Epoch 9, Loss: 1.3803, val_loss: 1.4614, val_acc: 0.4715
Epoch 10, Loss: 1.3328, val_loss: 1.3186, val_acc: 0.5243
Epoch 11, Loss: 1.2827, val_loss: 1.5154, val_acc: 0.4731
Epoch 12, Loss: 1.2427, val_loss: 1.2706, val_acc: 0.5449
Epoch 13, Loss: 1.1983, val_loss: 1.3755, val_acc: 0.5178
Epoch 14, Loss: 1.1555, val_loss: 1.3382, val_acc: 0.5321
Epoch 15, Loss: 1.1150, val_loss: 1.2754, val_acc: 0.5503
Epoch 16, Loss: 1.0765, val_loss: 1.7405, val_acc: 0.4373
Epoch 17, Loss: 1.0356, val_loss: 1.4542, val_acc: 0.5095
Epoch 18, Loss: 0.9972, val_loss: 1.2461, val_acc: 0.5696
Epoch 19, Loss: 0.9577, val_loss: 1.3297, val_acc: 0.5443
Epoch 20, Loss: 0.9218, val_loss: 1.4495, val_acc: 0.5172
root@f69f6b973d01:/work/codes/07/exercise#