令和７年度 深層学習

活性化関数・学習率・バッチサイズの違いによる CNN 分類モデルの学習比較実験レポート

令和 7 年 5 月 22 日

大阪国際工科専門職大学
工科学部
情報工学科
AI 開発コース

OK240100 藤村勇仁

目次

1　はじめに

本実験では、畳み込みニューラルネットワーク（CNN）において、活性化関数（ReLU, sigmoid）、学習率、バッチサイズの違いが学習速度や検証精度に与える影響を比較・検討した。最適なハイパーパラメータを探索することは、モデルの性能向上に不可欠であるため、その基礎的知見を得ることを目的とした。

2　実験方法

モデルには PyTorch を用いた SimpleCNN を採用した。活性化関数としては ReLU および sigmoid を中間層に適用し、出力層には活性化関数を用いずクロスエントロピー誤差で損失を算出した。学習率は 0.01、0.005、0.0001、0.00005 を、バッチサイズは 32 および 64 を選択した。

データセットには Kaggle で公開されている「Dogs vs. Cats」データセット[*1]を利用し、犬と猫の画像を 2 クラスに分類する課題設定とした。Kaggle 上の公式データセットをダウンロードし、PyTorch の DataLoader を用いて学習・検証データとして読み込んだ。エポック数は 5 または 10 とした。

実験に使用したコードは、7 部に示す。

3　実験結果

図 1 に、総学習時間と精度の関係を示す。

[*1] https://www.kaggle.com/c/dogs-vs-cats

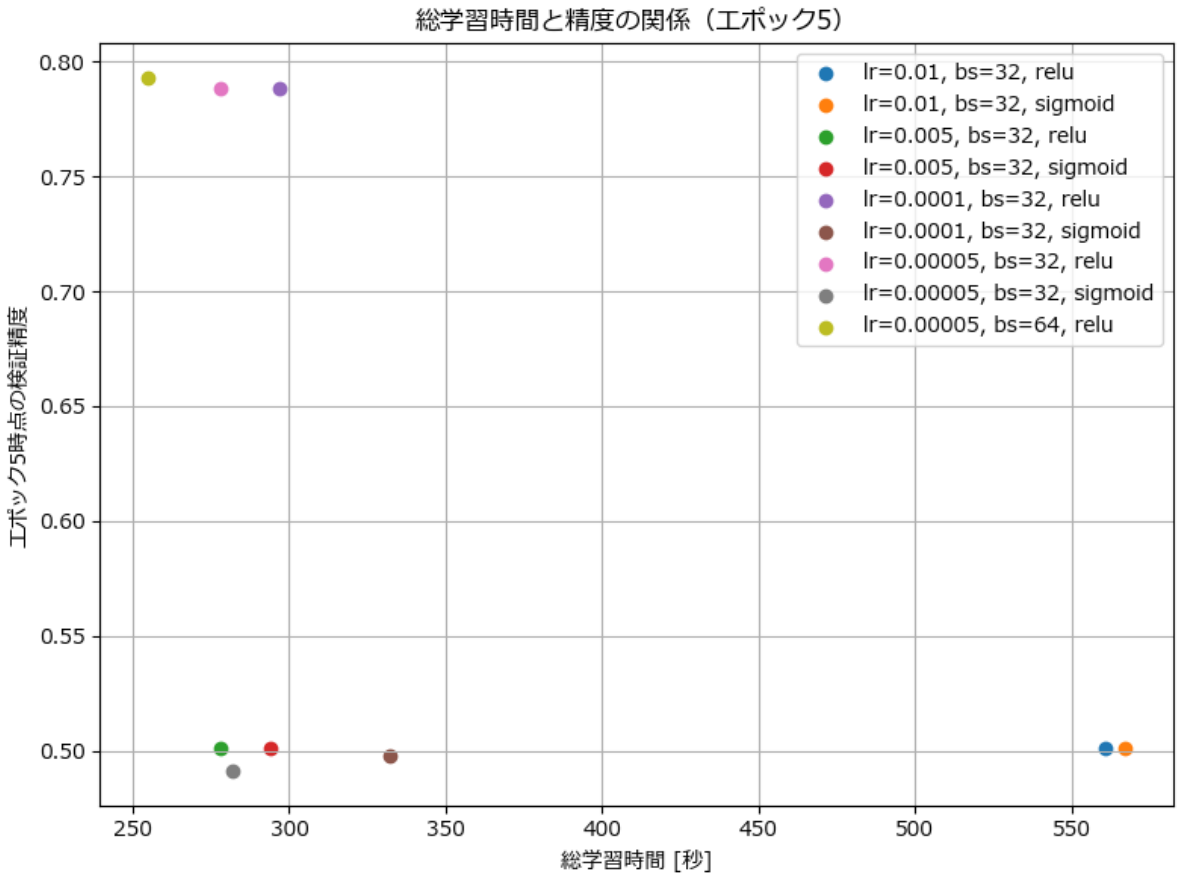

図 1　実験結果の散布図

学習率 0.01 および 0.005 では、ReLU・sigmoid のどちらを用いても学習が進まず、エポック 5 時点の検証精度は 0.5 前後に留まった。例えば学習率 0.01, バッチサイズ 32, ReLU の場合、エポック 5 までの総学習時間は約 279.96 秒、検証精度は 0.5014 であった。sigmoid の場合も同様に、総学習時間 283.59 秒、検証精度 0.4986 と変わらなかった。学習率 0.005 でも、ReLU は 279.50 秒、sigmoid は 300.09 秒で、いずれも検証精度は 0.5014 であった。

一方、学習率を 0.0001 や 0.00005 まで下げると、ReLU の場合に限り学習が進み、検証精度が向上した。学習率 0.0001, ReLU, バッチサイズ 32 の条件では、5 エポックでの総学習時間は 296.99 秒、検証精度は 0.7880 となった。sigmoid ではこの条件でも精度が 0.4976 に留まり、依然として学習が進まない傾向が見られた。バッチサイズを 64 に増やした場合、エポックごとの学習時間は短縮しつつも、検証精度は大きく変化しなかった。

4　考察

今回の実験から、学習率が高すぎる場合には損失が減少せず、検証精度も向上しないことが明らかとなった。特に sigmoid を活性化関数に用いた場合、勾配消失などの影響もあり、学習率を下げても十分な精度向上が得られなかった。ReLU では学習率を適切に下げることで損失が減少し、検証精度も 0.78 程度まで上昇した。バッチサイズの増加による学習時間短縮効果は確認できたが、今回は精度への顕著な影響は見られなかった。

5　まとめ

活性化関数として ReLU を用い、学習率を 0.0001 程度まで下げることで、CNN モデルは安定して高い精度を発揮できることが確認できた。一方、sigmoid では学習が進まず、検証精度も上がらない点から、深層学習においては ReLU の有効性が再確認された。今後は層構造や正則化手法、学習率スケジューラの導入なども視野にさらなる性能向上を目指す。

6　参考データ

表 1 に各条件でのエポック 5 時点の総学習時間と検証精度をまとめる。

活性化関数	学習率	バッチサイズ	総学習時間 (秒)	検証精度
ReLU	0.01	32	279.96	0.5014
sigmoid	0.01	32	283.59	0.4986
ReLU	0.005	32	279.50	0.5014
sigmoid	0.005	32	300.09	0.5014
ReLU	0.0001	32	296.99	0.7880
sigmoid	0.0001	32	331.55	0.4976

表 1　各条件でのエポック 5 時点の総学習時間と検証精度

7　使用コード

コード 1　データセット整形

```
import os
import shutil

os.makedirs("./train/cat", exist_ok=True)
os.makedirs("./train/dog", exist_ok=True)

for fname in os.listdir("./train"):
    path = os.path.join("./train", fname)
    if os.path.isfile(path): # ファイルのみ処理
        if fname.startswith("cat"):
            shutil.move(path, f"./train/cat/{fname}")
        elif fname.startswith("dog"):
            shutil.move(path, f"./train/dog/{fname}")
```

コード 2　CNN モデル定義（PyTorch）

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


class SimpleCNN(nn.Module):
    def __init__(self, num_classes=2, activation="relu"):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 16, kernel_size=3, padding=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(16, 32, kernel_size=3, padding=1)
```

```
        self.fc1 = nn.Linear(32 * 32 * 32, 128)
        self.fc2 = nn.Linear(128, num_classes)
        self.activation = activation

    def forward(self, x):
        x = self.pool(self._act(self.conv1(x)))
        x = self.pool(self._act(self.conv2(x)))
        x = x.view(-1, 32 * 32 * 32)
        x = self._act(self.fc1(x))
        x = self.fc2(x)
        return x

    def _act(self, x):
        if self.activation == "relu":
            return F.relu(x)
        elif self.activation == "sigmoid":
            return torch.sigmoid(x)
        else:
            raise ValueError("Unknown activation function")
```

コード 3　学習

```
import torch
import torch.nn as nn
import torch.optim as optim
from tqdm import tqdm
import time


def train_model(
    model, train_loader, val_loader, device, num_epochs=3, learning_rate=0.001
):
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=learning_rate)

    train_losses = []
    val_accuracies = []
    epoch_times = []

    model.to(device)

    for epoch in range(num_epochs):
        start_time = time.time()
        model.train()
        running_loss = 0.0

        for images, labels in tqdm(
            train_loader, desc=f"Epoch {epoch + 1}/{num_epochs}"
        ):
            images, labels = images.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = model(images)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            running_loss += loss.item() * images.size(0)

        end_time = time.time()
        epoch_time = end_time - start_time
        epoch_times.append(epoch_time)

        avg_loss = running_loss / len(train_loader.dataset)
```

```
        train_losses.append(avg_loss)

        model.eval()
        correct = 0
        total = 0
        with torch.no_grad():
            for images, labels in val_loader:
                images, labels = images.to(device), labels.to(device)
                outputs = model(images)
                _, predicted = torch.max(outputs, 1)
                total += labels.size(0)
                correct += (predicted == labels).sum().item()
        val_accuracy = correct / total
        val_accuracies.append(val_accuracy)

        print(
            f"Epoch {epoch + 1}: Loss={avg_loss:.4f}, Val Acc={val_accuracy:.4f}, Time
                ={epoch_time:.2f}s"
        )

    return train_losses, val_accuracies, epoch_times
```

コード 4　実験の実行

```
import torch
from model import SimpleCNN
from train_with_time import train_model
from torchvision import transforms, datasets
from torch.utils.data import DataLoader, random_split


def run_experiments():
    # 前処理
    transform = transforms.Compose(
        [
            transforms.Resize((128, 128)),
            transforms.ToTensor(),
            transforms.Normalize([0.5] * 3, [0.5] * 3),
        ]
    )
    dataset = datasets.ImageFolder("train", transform=transform)
    train_size = int(0.8 * len(dataset))
    val_size = len(dataset) - train_size
    train_dataset, val_dataset = random_split(dataset, [train_size, val_size])
    train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
    val_loader = DataLoader(val_dataset, batch_size=64)

    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

    # ハイパーパラメータ設定
    num_epochs = 5
    activation_list = ["relu"]

    results = {} # 結果を格納

    for act in activation_list:
        print(f"==== 実験: 活性化関数={act} ====")
        model = SimpleCNN(num_classes=2, activation=act)
        model = SimpleCNN(num_classes=2, activation=act)
        train_losses, val_accuracies, epoch_times = train_model(
            model,
            train_loader,
```

```
            val_loader,
            device,
            num_epochs=num_epochs,
            learning_rate=0.0005,
        )
        results[act] = {
            "train_losses": train_losses,
            "val_accuracies": val_accuracies,
            "epoch_times": epoch_times,
        }
    return results
```

コード 5　総学習時間と精度の関係グラフ描画

```
import matplotlib.pyplot as plt

plt.rcParams["font.family"] = "Meiryo"

# データ: [総学習時間, 精度, ラベル]
data = [
    [280, 0.5014, "lr=0.01, bs=32, relu"],
    [284, 0.4986, "lr=0.01, bs=32, sigmoid"],
    [279, 0.5014, "lr=0.005, bs=32, relu"],
    [300, 0.5014, "lr=0.005, bs=32, sigmoid"],
    [297, 0.7880, "lr=0.0001, bs=32, relu"],
    [332, 0.4976, "lr=0.0001, bs=32, sigmoid"],
    [278, 0.7884, "lr=0.00005, bs=32, relu"],
    [282, 0.4914, "lr=0.00005, bs=32, sigmoid"],
    [255, 0.7928, "lr=0.00005, bs=64, relu"],
]

fig, ax = plt.subplots(figsize=(8, 6))
for t, acc, label in data:
    plt.scatter(t, acc, label=label)

plt.xlabel("総学習時間 [秒]")
plt.ylabel("エポック5時点の検証精度")
plt.title("総学習時間と精度の関係 (エポック5) ")
plt.legend(loc="upper right")
plt.grid(True)
plt.tight_layout()
plt.show()
```