

# 学習状況の可視化

---

昨日、適切なデータ整理を自動的に行えるようにしたことで、NestedUNet を訓練することが可能になった。そこで、今日は UNet の訓練状況を TensorBoard を使用して可視化することに取り組む。

## 1. TensorBoard とは

Google の python 用機械学習パッケージである **TensorFlow** に付属しているツールであり、データフローグラフの可視化や学習の履歴（損失関数の変化など）の可視化、あるいは途中過程で生成される画像や音声の表示を行うことができる『[TensorBoard とは？ スカラー値やデータフローグラフの可視化](#)』。

そのため、**TensorFlow** で作成されたモデルに対して使用することを前提としているが、サードパーティーが開発したパッケージを使用することで **PyTorch** などで作成したモデルに対しても使用することができる。

応用方法に関しては、[学習済み分散表現を TensorBoard で可視化する](#)などを参照。

---

## 2. JupyterNotebook 上での実装

**TensorBoard** は初めて使用するため、デバッグが簡単な対話型開発環境である **JupyterNotebook** 上にプログラムを構築していく『[Jupyter](#)』。

[PyTorch 1.8 : PyTorch の学習 : TensorBoard でモデル、データと訓練を可視化する](#)を参考に、**TensorBoard** で表示するグラフの生データを生成していく。

---

### 2.1. TensorBoard で使用する生データをつくるための CNN モデルの設計および訓練

生データを生成するための機械学習のモデルのプログラムを、次のページに示す。

訓練で使用する **FashionMNIST** のダウンロードと、入力配列への変換を行うプログラム。

```
# imports
import matplotlib.pyplot as plt
import numpy as np

import torch
import torchvision
import torchvision.transforms as transforms

import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

# transforms
transform = transforms.Compose([transforms.ToTensor(),
                                transforms.Normalize((0.5,), (0.5,))])

# datasets
trainset = torchvision.datasets.FashionMNIST('./data', download=True, train=True,
```

```

transform=transform)
testset = torchvision.datasets.FashionMNIST('./data', download=True, train=False,
transform=transform)

# dataloaders
trainloader = torch.utils.data.DataLoader(trainset, batch_size=4, shuffle=True,
num_workers=2)
testloader = torch.utils.data.DataLoader(testset, batch_size=4, shuffle=False,
num_workers=2)

# constant for classes
classes = ('T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal',
'Shirt', 'Sneaker', 'Bag', 'Ankle Boot')

# helper function to show an image
# (used in the `plot_classes_preds` function below)
def matplotlib_imshow(img, one_channel=False):
    if one_channel:
        img = img.mean(dim=0)
    img = img / 2 + 0.5 # unnormalize
    npimg = img.numpy()
    if one_channel:
        plt.imshow(npimg, cmap="Greys")
    else:
        plt.imshow(np.transpose(npimg, (1, 2, 0)))

```

---

## 単純な構造を持つ CNN モデル

```

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 4 * 4, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 4 * 4)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

net = Net()

```

CNN を訓練する際に使用する損失関数と最適化関数の定義

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

---

## 2.2. TensorBoard セットアップ

まず, `torch.utils` から `tensorboard` をインポートして `SummaryWriter` を定義する. これは, 情報を `TensorBoard` を書くためのオブジェクトである.

```
from torch.utils.tensorboard import SummaryWriter

# default `log_dir` is "runs" - we'll be more specific here
writer = SummaryWriter(log_dir=log_dir)
```

ここで, `log_dir` は `TensorBoard` に書く情報を保存しておくディレクトリへのパスであり, 何も指定しなかった場合には, カレントディレクトリ上に `log` ディレクトリが新規で作成され, その中に情報が保存される.

また, `log_dir` 内には `events.out.tfevents.1617710098.DESKTOP-Q123T6P.15916.0` のようなファイルが生成される.

---

## 2.3. TensorBoard に画像を表示する.

以下のプログラムを `FashionMNIST` のダウンロードプログラムの後に実行することで, 2 種類の画像をグリッド上に表示するデータを, `log_dir` 内に生成することができる.

```
# get some random training images
dataiter = iter(trainloader)
images, labels = dataiter.next()

# create grid of images
img_grid = torchvision.utils.make_grid(images)

# show images
matplotlib.imshow(img_grid, one_channel=True)

# write to tensorboard
writer.add_image('four_fashion_mnist_images', img_grid)
```

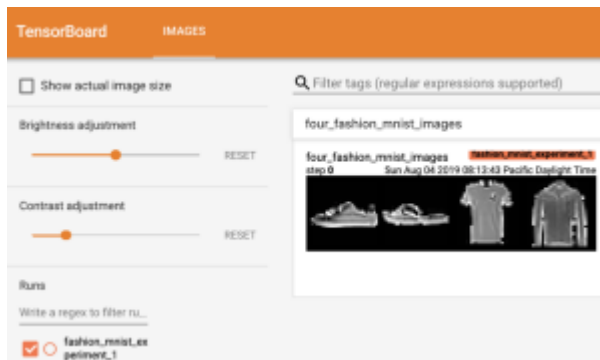
上記のプログラミングを実行した後に, コマンドライン上で以下のコマンドを実行する

```
tensorboard --logdir=log_dir
```

すると,

TensorBoard 2.4.0 at http://localhost:6006/ (Press CTRL+C to quit)

という URL が表示されるので, Web ブラウザからその URL にアクセスすると, 次のような画像が表示される.



JupyterNotebook で開発している場合, ノートブックで TensorBoard を使用する に記載されているように, magic を使用することで Notebook 上に直接 TensorBoard を表示することができる.

具体的には, `tensorboard --logdir=log_dir` をコマンドライン上で実行するのではなく, JupyterNotebook 上に次のように記述する.

```
%tensorboard --logdir=log_dir
```

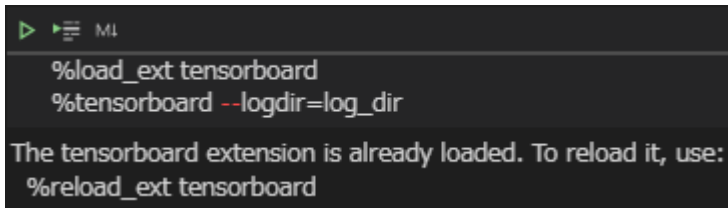
これにより, 直接 TensorBoard を確認できるようになるはずであったが, 自分の環境では上手く表示することができなかった.

そこで, 次のようなコマンドを JupyterNotebook 上で事前に読み込んでおかなければならない可能性が浮上した『学習中モデルを Jupyter Notebook 上の TensorBord でモニタリングする』.

```
!pip install cloud-tpu-client
%load_ext tensorboard
```

先ほどのコマンドを事前に実行すると, TensorBoard を Notebook 上に表示することに成功した. しかし, 画像情報など log\_dir に格納されているはずの情報を表示することができなかった.

また, 先ほどのコマンドは 2 回目を実行すると以下のようなエラーを吐き, 削除しても TensorBoard が実行出来なくなることにはなかったので削除した.

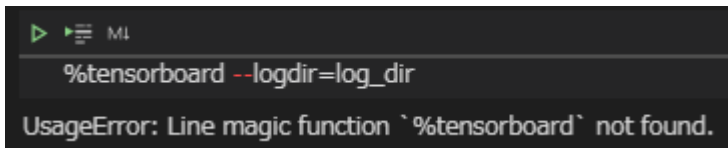


```
▶ ML
%load_ext tensorboard
%tensorboard --logdir=log_dir

The tensorboard extension is already loaded. To reload it, use:
%reload_ext tensorboard
```

結果として、今回は直接表示することをあきらめ Web からアクセスする方法を取ることにする。

【4月7日追記】：後日 **Notebook** を再起動したところ、以下のようなエラーが発生した。



```
▶ ML
%tensorboard --logdir=log_dir

UsageError: Line magic function `%tensorboard` not found.
```

このエラーが発生する場所以前で先ほどのコマンドを実行すると、再度 **TensorBoard** が起動したため、新規および再起動時には必要なコマンドであると考えられる。

さらに、**log\_dir** 内にログが生成されているにも関わらず

---

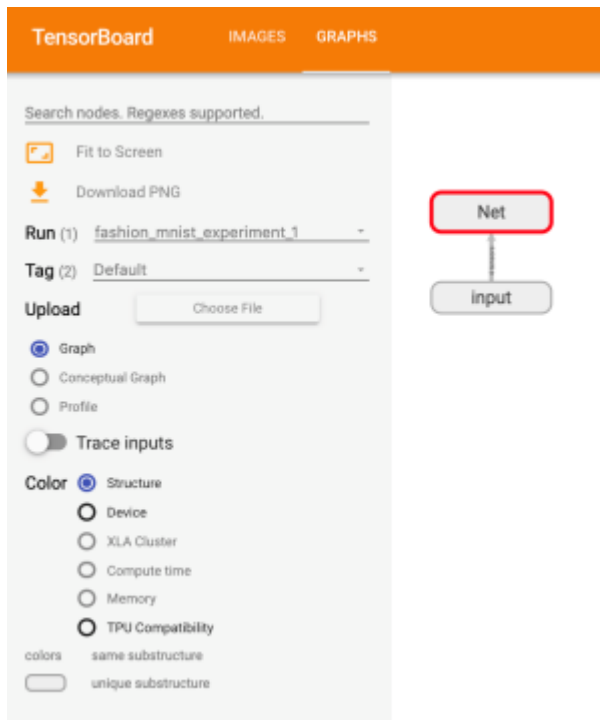
## 2.4. CNN モデルを TensorBoard 上に表示する

先ほどまでのプログラムの後ろに、以下プログラムを記述する。

```
writer.add_graph(net, images)
writer.close()
```

ここで、**write.close()** は **SummaryWriter** がなくなっただけの場合に呼ぶ必要があるコードです。

このプログラムを実行すると、**TensorBoard** の上部分に **model** ボタンが表示され、クリックすると右図のようなモデルが表示される。



## 2.5. 訓練を TensorBoard で追跡する

```

running_loss = 0.0
for epoch in range(1): # loop over the dataset multiple times

    for i, data in enumerate(trainloader, 0):

        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        if i % 1000 == 999: # every 1000 mini-batches...

            # ...log the running loss
            writer.add_scalar('training loss',
                             running_loss / 1000,
                             epoch * len(trainloader) + i)

            # ...log a Matplotlib Figure showing the model's predictions on a
            # random mini-batch
            writer.add_figure('predictions vs. actuals',
                              plot_classes_preds(net, inputs, labels),

```

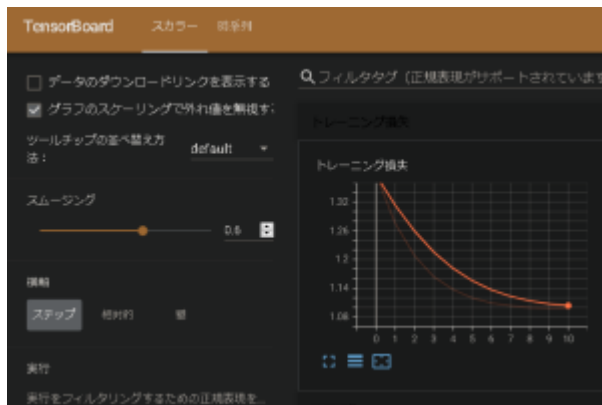
```

        global_step=epoch * len(trainloader) + i)
    running_loss = 0.0
    print('Finished Training')

```

これまでのプログラムに続いて上記のコードを実行することで、先ほど設計した単純な構造を持つ CNN モデルを FashionMNIST で訓練する。そして、訓練中の epoch ごとの損失は `writer.add_scalar` によって `log_dir` 内に保存される。

このログを **TensorBoard** 上で表示すると右図のようになる。



### 3. UNet への実装

ここまでの経験をもとに、UNet の訓練を **TensorBoard** 上で表示できるよう改良する。方法は非常に簡単で、`main` の最初に

```

# Tensorboard 用のログを記録するディレクトリパス
log_dir = mkdir(cfg.NESTED_UNET_DIR, 'log')
writer = SummaryWriter(log_dir=log_dir)

```

を追加し、訓練を回す `for` 文の途中に

```

# Tensorboard用のデータ
writer.add_scalar('training loss', train_log['loss'], epoch)
writer.add_scalar('validation loss', val_log['loss'], epoch)

```

を追加する。その結果、右図のように訓練と検証の誤差を **TensorBoard** 上に表示できるようになった。

