hw5

October 14, 2021

1 Mathematical Foundations of Deep Neural Network

1.1 Homework 5

1.1.1 2017-11362

1.1.2 Problem 1: *Implementing backprop for MLP.*

[1]:
```
import torch
from torch import nn

def sigma(x):
    return torch.sigmoid(x)
def sigma_prime(x):
    return sigma(x)*(1-sigma(x))


torch.manual_seed(0)
L = 6
X_data = torch.rand(4, 1)
Y_data = torch.rand(1, 1)

A_list,b_list = [],[]
for _ in range(L-1):
    A_list.append(torch.rand(4, 4))
    b_list.append(torch.rand(4, 1))
A_list.append(torch.rand(1, 4))
b_list.append(torch.rand(1, 1))

# Option 1: directly use PyTorch's autograd feature
for A in A_list:
    A.requires_grad = True
for b in b_list:
    b.requires_grad = True

y = X_data
for ell in range(L):
    S = sigma if ell<L-1 else lambda x: x
    y = S(A_list[ell]@y+b_list[ell])
```

```python
# backward pass in pytorch
loss=torch.square(y-Y_data)/2
loss.backward()
print("autograd")
print(A_list[0].grad)
```

```
autograd
tensor([[2.3943e-05, 3.7064e-05, 4.2687e-06, 6.3700e-06],
        [3.4104e-05, 5.2794e-05, 6.0804e-06, 9.0735e-06],
        [2.4438e-05, 3.7831e-05, 4.3571e-06, 6.5019e-06],
        [2.0187e-05, 3.1250e-05, 3.5991e-06, 5.3707e-06]])
```

[2]:
```python
torch.manual_seed(0)
L = 6
X_data = torch.rand(4, 1)
Y_data = torch.rand(1, 1)

A_list,b_list = [],[]
for _ in range(L-1):
    A_list.append(torch.rand(4, 4))
    b_list.append(torch.rand(4, 1))
A_list.append(torch.rand(1, 4))
b_list.append(torch.rand(1, 1))

# Option 3: implement backprop yourself
y_list = [X_data]
y = X_data
for ell in range(L):
    S = sigma if ell<L-1 else lambda x: x
    y = S(A_list[ell]@y+b_list[ell])
    y_list.append(y)

dA_list = []
db_list = []
dy = y-Y_data  # dloss/dy_L
for ell in reversed(range(L)):
    S = sigma_prime if ell<L-1 else lambda x: torch.ones(x.shape)
    A, b, y = A_list[ell], b_list[ell], y_list[ell]
    dd = torch.diag(S(A @ y + b).view(-1))
    db = dy @ dd      # dloss/db_ell
    dA = (y @ db).T   # dloss/dA_ell
    dy = db @ A       # dloss/dy_{ell-1}

    dA_list.insert(0, dA)
    db_list.insert(0, db)
print("backprop")
```

```python
print(dA_list[0])
```

```
backprop
tensor([[2.3943e-05, 3.7064e-05, 4.2687e-06, 6.3700e-06],
        [3.4104e-05, 5.2794e-05, 6.0804e-06, 9.0735e-06],
        [2.4438e-05, 3.7831e-05, 4.3571e-06, 6.5019e-06],
        [2.0187e-05, 3.1250e-05, 3.5991e-06, 5.3707e-06]])
```

1.1.3 Problem 2: *Vanishing gradients.*

① $A_1, \cdots, A_L$: not too large. $\exists j \in \{\ell+1, \cdots, L\}$ s.t A_j is small.

Let $i \in \{1, \cdots, \ell\}$.

$$\frac{\partial y_L}{\partial b_i} = \frac{\partial y_L}{\partial y_{L-1}} \frac{\partial y_{L-1}}{\partial y_{L-2}} \cdots \frac{\partial y_{i+1}}{\partial y_i} \frac{\partial y_i}{\partial b_i} \quad : \text{not too large}$$

$$\frac{\partial y_i}{\partial b_i} = \frac{\partial}{\partial b_i} \sigma(A_i y_{i-1} + b_i) = \mathrm{diag}(\sigma'(A_i y_{i-1} + b_i)) = \mathrm{diag}(\sigma'(\tilde{y}_i)) \quad : \text{not too large}$$

$$\frac{\partial y_j}{\partial y_{j-1}} = \frac{\partial}{\partial y_{j-1}} \sigma(A_j y_{j-1} + b_j) = \mathrm{diag}(\sigma'(\tilde{y}_j)) \cdot A_j \rightarrow \text{small}$$

$$\frac{\partial y_k}{\partial y_{k-1}} = \mathrm{diag}(\sigma'(\tilde{y}_k)) A_k \quad : \text{not too large} \quad (k \neq j)$$

$$\frac{\partial y_L}{\partial A_i} = \mathrm{diag}(\sigma'(\tilde{y}_i)) \left(\frac{\partial y_L}{\partial y_i}\right)^T y_{i-1}^T \quad : \text{small}$$

$$\frac{\partial y_L}{\partial y_i} = \frac{\partial y_L}{\partial y_{L-1}} \cdots \underbrace{\frac{\partial y_j}{\partial y_{j-1}}}_{\text{small}} \cdots \frac{\partial y_{i+1}}{\partial y_i} \quad : \text{small}$$

② $\exists j \in \{\ell+1, \cdots, L\}$ s.t $|\tilde{y}_j| \rightarrow \infty$ (i.e. $\sigma'(\tilde{y}_j) \rightarrow 0$, "small")

$$\frac{\partial y_L}{\partial b_i} = \underbrace{\frac{\partial y_L}{\partial y_i}}_{\text{not too large}} \cdot \underbrace{\mathrm{diag}(\sigma'(\tilde{y}_i))}_{\text{small}} \quad : \text{small}$$

$$\frac{\partial y_L}{\partial A_i} = \underbrace{\mathrm{diag}(\sigma'(\tilde{y}_i))}_{\text{small}} \underbrace{\left(\frac{\partial y_L}{\partial y_i}\right)^T}_{\text{not too large}} y_{i-1}^T \quad : \text{small}$$

1.1.4 Problem 3. *Two forms of momentum SGD*

① Form I. $\theta^{k+1} = \theta^k - \alpha g^k + \beta(\theta^k - \theta^{k-1})$

② Form II. $\begin{cases} v^{k+1} = g^k + \beta v^k \\ \tilde{\theta}^{k+1} = \tilde{\theta}^k - \alpha v^{k+1} \end{cases}$

Let's show that $\theta^k = \tilde{\theta}^k$ for $k=1,2,\cdots$ where $\theta^0, g^0, g^1, \cdots \in \mathbb{R}^n$ are given.

$\theta^1 = \theta^0 - \alpha g^0 + \beta(\theta^0 - \theta^{-1}) = \theta^0 - \alpha g^0$

$\tilde{\theta}^1 = \tilde{\theta}^0 - \alpha v^1 = \tilde{\theta}^0 - \alpha(g^0 + \beta v^0) = \tilde{\theta}^0 - \alpha g^0$.

Assume that $\theta^i = \tilde{\theta}^i$ for $i=1,\cdots,k$.

i.e. $\theta^k = \theta^{k-1} - \alpha g^{k-1} + \beta(\theta^{k-1} - \theta^{k-2}) \Rightarrow \theta^k - \theta^{k-1} = -\alpha g^{k-1} + \beta(\theta^{k-1} - \theta^{k-2})$

$\tilde{\theta}^k = \tilde{\theta}^{k-1} - \alpha v^k \Rightarrow \alpha v^k = \alpha g^{k-1} - \beta(\theta^{k-1} - \theta^{k-2})$

Then, $\tilde{\theta}^{k+1} = \tilde{\theta}^k - \alpha v^{k+1} = \tilde{\theta}^k - \alpha(g^k + \beta v^k)$

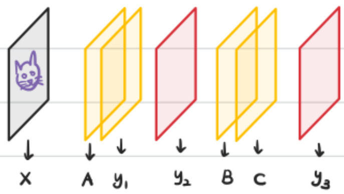
$= \tilde{\theta}^k - \alpha g^k - \beta(\alpha v^k)$

$= \theta^k - \alpha g^k - \beta(\alpha g^{k-1} - \beta(\theta^{k-1} - \theta^{k-2}))$

$= \theta^k - \alpha g^k + \beta(\theta^k - \theta^{k-1}) = \theta^{k+1}$.

$\therefore$ By induction, $\tilde{\theta}^k = \theta^k$ for $k=1,2,\cdots$.

1.1.5 Problem 4: *Receptive field*



W^1 : 3x3 Conv $(3 \to 64)$, W^2: 3x3 Conv $(64 \to 64)$

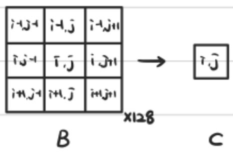
W^3 : maxPool $(2\times2 \to 1)$

W^4 : 3x3 conv $(64 \to 128)$, W^5: 3x3 conv $(128 \to 128)$

$$X \xrightarrow{W^1} A \xrightarrow{W^2} y_1 \xrightarrow{W^3} y_2 \xrightarrow{W^4} B \xrightarrow{W^5} C \xrightarrow{W^3} y_3$$

① $y_3[i,j,k] = \max C[2i:2i+1,\ 2j:2j+1,\ k] \Rightarrow C[2i:2i+1,\ 2j:2j+1,\ k] \mapsto y_3[i,j,k]$ (a:b means a, a+1, ⋯, b)

②



$\Rightarrow B[i-1:i+1,\ j-1:j+1,\ :] \mapsto C[i,j,k]$

$\therefore B[2i-1:2i+2,\ 2j-1:2j+2,\ :] \mapsto y_3[i,j,k]$.

③ $y_2 \to B$: same kernel with ② $\Rightarrow y_2[i-1:i+1,\ j-1:j+1,\ :] \mapsto B[i,j,k]$

$\therefore y_2[2i-2:2i+3,\ 2j-2:2j+3,\ :] \mapsto y_3[i,j,k]$

④ $y_1 \to y_2$: same layer with ① $\Rightarrow y_1[2i:2i+1,\ 2j:2j+1,\ k] \mapsto y_2[i,j,k]$

$\therefore y_1[4i-4:4i+7,\ 4j-4:4j+7,\ :] \mapsto y_3[i,j,k]$

⑤ $A \to y_1$: Same kernel with ② $\Rightarrow A[i-1:i+1,\ j-1:j+1,\ :] \mapsto y_1[i,j,k]$

$\therefore A[2i-1:2i+2,\ 2j-1:2j+2,\ :] \mapsto y_2[i,j,k]$

$A[4i-5:4i+8,\ 4j-5:4j+8,\ :] \mapsto y_3[i,j,k]$

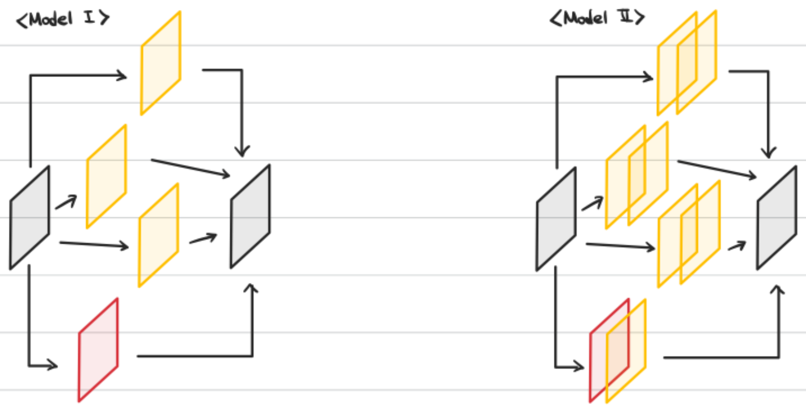
⑥ $X \to A$: Same kernel with ② $\Rightarrow X[i-1:i+1,\ j-1:j+1,\ :] \mapsto A[i,j,k]$

$\therefore X[i-2:i+2,\ j-2:j+2,\ :] \mapsto y_1[i,j,k]$

$X[2i-2:2i+3,\ 2j-2:2j+3,\ :] \mapsto y_2[i,j,k]$

$X[4i-6:4i+9,\ 4j-6:4j+9,\ :] \mapsto y_3[i,j,k]$.

1.1.6 Problem 5: *bottleneck convolution*



① trainable parameters

For $k \times k$ conv ($C_{in} \to C_{out}$), # of trainable params $= \underbrace{k^2 \cdot C_{in} \cdot C_{out}}_{\text{filters}} + \underbrace{C_{out}}_{\text{bias}}$

<Model I> : 1x1 Conv (256 → 124) : 31868
3x3 Conv (256 → 192) : 442560
5x5 conv (256 → 96) : 614496
$\}$ 1088924

<Model II> : 1x1 Conv (256 → 124) : 31868
1x1 Conv (256 → 64) x 3 : 49344
3x3 Conv (64 → 192) : 110784
5x5 Conv (64 → 96) : 153696
$\}$ 345692

② # of addition, multiplication, activation (+, ×, σ)

For $k \times k$ conv ($C_{in} \to C_{out}$), addition : $32^2 \cdot k^2 \cdot C_{in} \cdot C_{out}$, multiplication : $\underbrace{32^2}_{\text{total pixel of image}} \cdot k^2 \cdot C_{in} \cdot C_{out}$, activation : $32^2 \cdot C_{out}$

<Model I> : 1x1 Conv (256 → 124) : (32505856, 32505856, 126976)
3x3 Conv (256 → 192) : (452984832, 452984832, 196608)
5x5 conv (256 → 96) : (629145600, 629145600, 98304)
$\}$ + : 1,114,636,288
× : 〃
σ : 421,888

<Model II> : 1x1 Conv (256 → 124) : (32505856, 32505856, 126976)
1x1 Conv (256 → 64) x 3 : (50331648, 50331648, 196608)
3x3 Conv (64 → 192) : (113246208, 113246208, 196608)
5x5 Conv (64 → 96) : (157286400, 157286400, 98304)
$\}$ + : 353,370,112
× : 〃
σ : 618,496.

1.1.7 Problem 6: *label - memorization*

[3]:
```
import torch
import torch.nn as nn
import time

# Make sure to use only 10% of the available MNIST data.
# Otherwise, experiment will take quite long (around 90 minutes).

from torchvision import datasets
from torchvision.transforms import transforms
from torch.utils.data import DataLoader

train_set = datasets.MNIST('./mnist_data', train=True, transform = transforms.
 ↪ToTensor(), download=True)
# 6,000 train set
idx = list(range(6000))
train_set.data = train_set.data[idx]
# randomized label
train_set.targets = torch.randint(0,9,(6000,))

# (Modified version of AlexNet)
class AlexNet(nn.Module):
    def __init__(self, num_class=10):
        super(AlexNet, self).__init__()

        self.conv_layer1 = nn.Sequential(
            nn.Conv2d(1, 96, kernel_size=4),
            nn.ReLU(inplace=True),
            nn.Conv2d(96, 96, kernel_size=3),
            nn.ReLU(inplace=True)
        )
        self.conv_layer2 = nn.Sequential(
            nn.Conv2d(96, 256, kernel_size=5, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2)
        )
        self.conv_layer3 = nn.Sequential(
            nn.Conv2d(256, 384, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(384, 384, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(384, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2)
        )
```

```
        self.fc_layer1 = nn.Sequential(
            nn.Dropout(),
            nn.Linear(6400, 800),
            nn.ReLU(inplace=True),
            nn.Linear(800, 10)
        )

    def forward(self, x):
        output = self.conv_layer1(x)
        output = self.conv_layer2(output)
        output = self.conv_layer3(output)
        output = torch.flatten(output, 1)
        output = self.fc_layer1(output)
        return output


learning_rate = 0.1
batch_size = 64
epochs = 150

train_loader = DataLoader(dataset=train_set, batch_size=batch_size,␣
 ↪shuffle=True)
test_loader = DataLoader(dataset=train_set, batch_size=batch_size)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = AlexNet().to(device)
loss_function = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)

Train_Accuracy = []
Train_Loss = []

tick = time.time()
for epoch in range(150):
    print(f"Epoch {epoch + 1} / {epochs}")
    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)

        optimizer.zero_grad()
        loss = loss_function(model(images), labels)
        loss.backward()

        optimizer.step()

    # Accuracy & Loss
    with torch.no_grad():
        correct = 0
```

```
        for images, labels in test_loader:
            output = model(images.to(device))
            pred = output.max(1, keepdim=True)[1].cpu().view(-1)
            correct += torch.sum(labels == pred)
        Train_Accuracy.append(correct/6000)
        Train_Loss.append(loss.item())

tock = time.time()
print(f"Total training time: {tock - tick}")
```

```
/home/zendo/anaconda3/lib/python3.8/site-
packages/torchvision/datasets/mnist.py:498: UserWarning: The given NumPy array
is not writeable, and PyTorch does not support non-writeable tensors. This means
you can write to the underlying (supposedly non-writeable) NumPy array using the
tensor. You may want to copy the array to protect its data or make it writeable
before converting it to a tensor. This type of warning will be suppressed for
the rest of this program. (Triggered internally at
../torch/csrc/utils/tensor_numpy.cpp:180.)
  return torch.from_numpy(parsed.astype(m[2], copy=False)).view(*s)
```

```
Epoch 1 / 150
Epoch 2 / 150
Epoch 3 / 150
Epoch 4 / 150
Epoch 5 / 150
Epoch 6 / 150
Epoch 7 / 150
Epoch 8 / 150
Epoch 9 / 150
Epoch 10 / 150
Epoch 11 / 150
Epoch 12 / 150
Epoch 13 / 150
Epoch 14 / 150
Epoch 15 / 150
Epoch 16 / 150
Epoch 17 / 150
Epoch 18 / 150
Epoch 19 / 150
Epoch 20 / 150
Epoch 21 / 150
Epoch 22 / 150
Epoch 23 / 150
Epoch 24 / 150
Epoch 25 / 150
Epoch 26 / 150
Epoch 27 / 150
Epoch 28 / 150
```

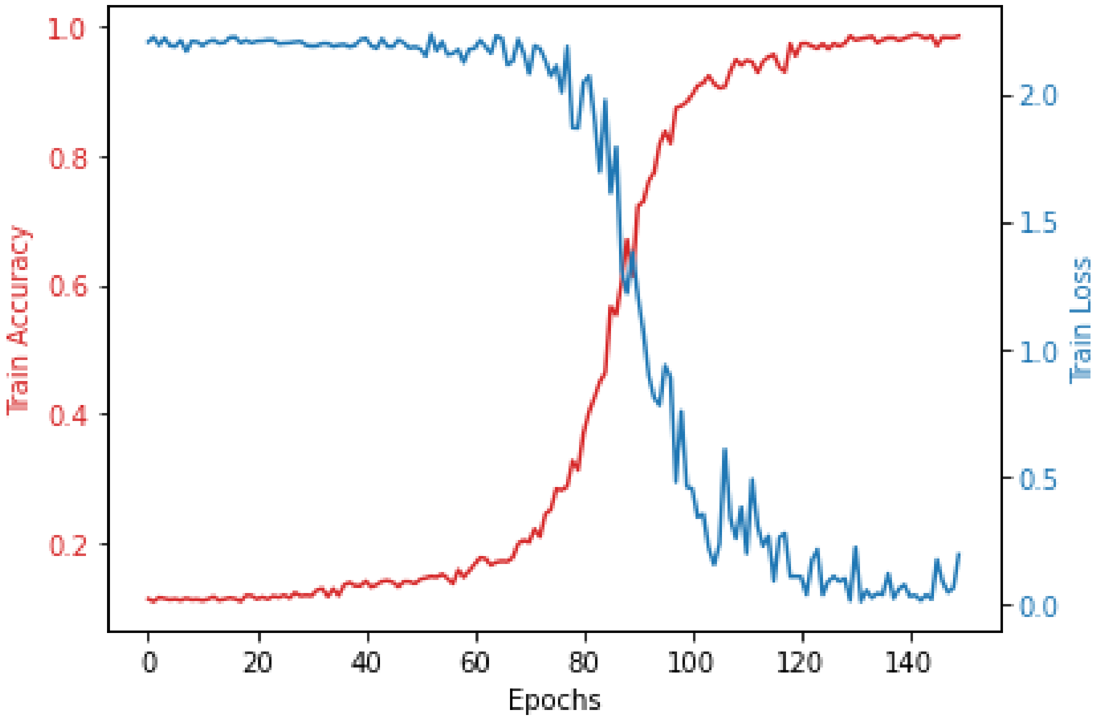
```
Epoch 29 / 150
Epoch 30 / 150
Epoch 31 / 150
Epoch 32 / 150
Epoch 33 / 150
Epoch 34 / 150
Epoch 35 / 150
Epoch 36 / 150
Epoch 37 / 150
Epoch 38 / 150
Epoch 39 / 150
Epoch 40 / 150
Epoch 41 / 150
Epoch 42 / 150
Epoch 43 / 150
Epoch 44 / 150
Epoch 45 / 150
Epoch 46 / 150
Epoch 47 / 150
Epoch 48 / 150
Epoch 49 / 150
Epoch 50 / 150
Epoch 51 / 150
Epoch 52 / 150
Epoch 53 / 150
Epoch 54 / 150
Epoch 55 / 150
Epoch 56 / 150
Epoch 57 / 150
Epoch 58 / 150
Epoch 59 / 150
Epoch 60 / 150
Epoch 61 / 150
Epoch 62 / 150
Epoch 63 / 150
Epoch 64 / 150
Epoch 65 / 150
Epoch 66 / 150
Epoch 67 / 150
Epoch 68 / 150
Epoch 69 / 150
Epoch 70 / 150
Epoch 71 / 150
Epoch 72 / 150
Epoch 73 / 150
Epoch 74 / 150
Epoch 75 / 150
Epoch 76 / 150
```

```
Epoch 77 / 150
Epoch 78 / 150
Epoch 79 / 150
Epoch 80 / 150
Epoch 81 / 150
Epoch 82 / 150
Epoch 83 / 150
Epoch 84 / 150
Epoch 85 / 150
Epoch 86 / 150
Epoch 87 / 150
Epoch 88 / 150
Epoch 89 / 150
Epoch 90 / 150
Epoch 91 / 150
Epoch 92 / 150
Epoch 93 / 150
Epoch 94 / 150
Epoch 95 / 150
Epoch 96 / 150
Epoch 97 / 150
Epoch 98 / 150
Epoch 99 / 150
Epoch 100 / 150
Epoch 101 / 150
Epoch 102 / 150
Epoch 103 / 150
Epoch 104 / 150
Epoch 105 / 150
Epoch 106 / 150
Epoch 107 / 150
Epoch 108 / 150
Epoch 109 / 150
Epoch 110 / 150
Epoch 111 / 150
Epoch 112 / 150
Epoch 113 / 150
Epoch 114 / 150
Epoch 115 / 150
Epoch 116 / 150
Epoch 117 / 150
Epoch 118 / 150
Epoch 119 / 150
Epoch 120 / 150
Epoch 121 / 150
Epoch 122 / 150
Epoch 123 / 150
Epoch 124 / 150
```

```
Epoch 125 / 150
Epoch 126 / 150
Epoch 127 / 150
Epoch 128 / 150
Epoch 129 / 150
Epoch 130 / 150
Epoch 131 / 150
Epoch 132 / 150
Epoch 133 / 150
Epoch 134 / 150
Epoch 135 / 150
Epoch 136 / 150
Epoch 137 / 150
Epoch 138 / 150
Epoch 139 / 150
Epoch 140 / 150
Epoch 141 / 150
Epoch 142 / 150
Epoch 143 / 150
Epoch 144 / 150
Epoch 145 / 150
Epoch 146 / 150
Epoch 147 / 150
Epoch 148 / 150
Epoch 149 / 150
Epoch 150 / 150
Total training time: 671.0086727142334
```

[4]:
```
import matplotlib.pyplot as plt

fig, ax1 = plt.subplots()

color = 'tab:red'
ax1.set_xlabel('Epochs')
ax1.set_ylabel('Train Accuracy', color=color)
ax1.plot(range(150), Train_Accuracy, color=color)
ax1.tick_params(axis='y', labelcolor=color)

ax2 = ax1.twinx()  # instantiate a second axes that shares the same x-axis

color = 'tab:blue'
ax2.set_ylabel('Train Loss', color=color)  # we already handled the x-label␣
 ↪with ax1
ax2.plot(range(150), Train_Loss, color=color)
ax2.tick_params(axis='y', labelcolor=color)

fig.tight_layout()  # otherwise the right y-label is slightly clipped
```

```
plt.show()
```



Train Accuracy
Train Loss
Epochs
1.0
0.8
0.6
0.4
0.2
2.0
1.5
1.0
0.5
0.0
0
20
40
60
80
100
120
140