LAB 5: NHẬN DẠNG QUẦN ÁO GIÀY DÉP THỜI TRANG VỚI BỘ DỮ LIỆU FASHION-MNIST, DÙNG THƯ VIỆN PYTORCH HUẤN LUYỆN MÔ HÌNH TRÊN GPU CỦA FIT-LAB

III. NHẬN DẠNG QUẦN ÁO GIÀY DÉP THỜI TRANG VỚI BỘ DỮ LIỆU FASHION-MNIST:

1. Tạo project Lab5 và tạo môi trường ảo để cài đặt các thư viện cần thiết cho Lab5

```
(venv) ubuntu@ubuntu-2274802010449:~/Lab5$ nano prepare_data.py
(venv) ubuntu@ubuntu-2274802010449:~/Lab5$ python prepare_data.py
100.0%
100.0%
100.0%
100.0%
Dataset Size: torch.Size([60000, 28, 28])
Unique Classes: ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
(venv) ubuntu@ubuntu-2274802010449:~/Lab5$
```

2. Chuẩn bị dữ liệu cho bài toán phân loại:

Bắt đầu bằng cách tải xuống tập dữ liệu và import các gói có liên quan.

Gói torchvision chứa nhiều bộ dữ liệu khác nhau – một trong số đó là bộ dữ liệu FashionMNIST

```
from torchvision import datasets import torch data_folder =
'~/data/FMNIST' # This can be any directory you want
# to download FMNIST to fmnist = datasets.FashionMNIST(data_folder,
download=True, train=True)
```

Tiếp theo, chúng ta phải lưu trữ những hình ảnh có sẵn trong fmnist.data với biến tr_images và các nhãn (targets) có sẵn trong fmnist.targets với tên biến tr_targets:

```
tr_images = fmnist.data
tr_targets = fmnist.targets
```

Kiểm tra các tensor mà chúng ta đang xử lý:

```
unique_values = tr_targets.unique() print(f'tr_images &
tr_targets:\n\tX - {tr_images.shape}\n\tY -
{tr_targets.shape}\n\tY - Unique Values : {unique_values}')
print(f'TASK:\n\t{len(unique_values)} class Classification')
print(f'UNIQUE CLASSES:\n\t{fmnist.classes}')
```

Đầu ra của đoạn mã trước như sau:

tr_images & tr_targets:

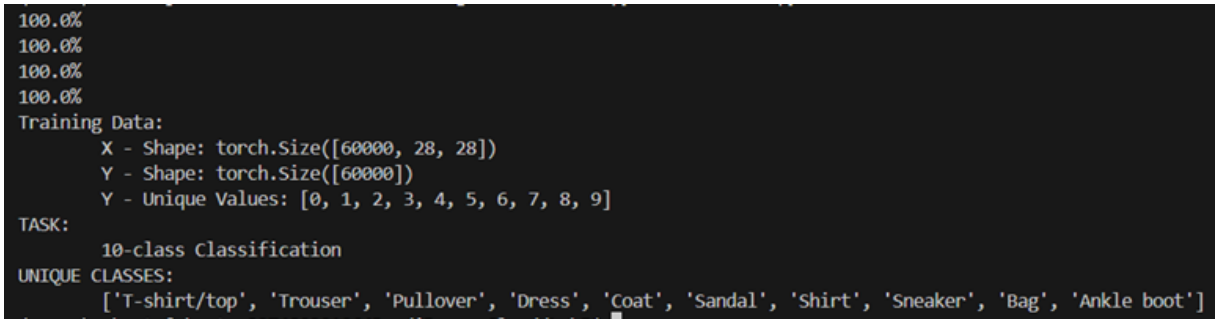
X - torch.Size([60000, 28, 28])

Y - torch.Size([60000])

Y - Unique Values : tensor([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]) TASK: 10 class Classification UNIQUE

CLASSES:

['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']



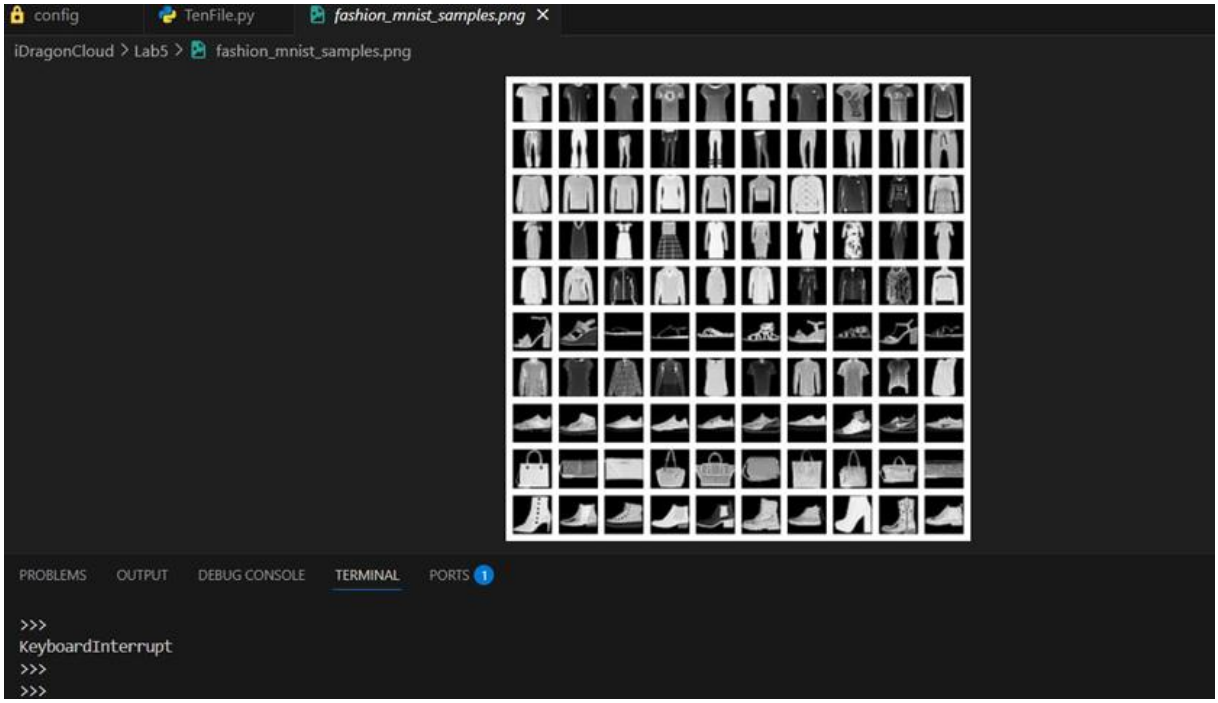
Ở đây, chúng ta có thể thấy rằng có 60.000 hình ảnh có kích thước 28 x 28 và có 10 lớp có thể có trên tất cả các hình ảnh. Lưu ý rằng tr_targets chứa các giá trị số cho mỗi lớp, trong khi fmnist.classes cung cấp cho chúng ta các tên tương ứng với từng giá trị số trong tr_targets. Vẽ một mẫu ngẫu nhiên gồm 10 hình ảnh cho tất cả 10 lớp có thể: Import các gói có liên quan để vẽ đồ thị các hình ảnh và để bạn cũng có thể làm việc trên các mảng:

```
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
```

Tạo một biểu đồ trong đó chúng ta có thể hiển thị một lưới 10 x 10, trong đó mỗi hàng của lưới tương ứng với một lớp và mỗi cột trình bày một hình ảnh mẫu thuộc về lớp của hàng. Lặp lại các số lớp duy nhất (label_class) và tìm nạp chỉ mục của các hàng (label_x_rows) tương ứng với số lớp đã cho:

```
R, C = len(tr_targets.unique()), 10 fig, ax
= plt.subplots(R, C, figsize=(10,10)) for
label_class, plot_row in enumerate(ax):
    label_x_rows = np.where(tr_targets == label_class)[0]
for plot_cell in plot_row:
        plot_cell.grid(False);
plot_cell.axis('off')          ix =
np.random.choice(label_x_rows)          x, y =
tr_images[ix], tr_targets[ix]
plot_cell.imshow(x, cmap='gray')
plt.tight_layout()
```

Kết quả:




```
Training Data:
        X - Shape: torch.Size([60000, 28, 28])
        Y - Shape: torch.Size([60000])
        Y - Unique Values: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
TASK:
        10-class Classification
UNIQUE CLASSES:
        ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
Đã lưu ảnh vào 'fashion_mnist_samples.png'
```

3. Huấn luyện mạng nơ-ron

Để huấn luyện mạng nơ-ron, chúng ta phải thực hiện các bước sau:

1. Import các gói có liên quan.

2. Xây dựng tập dữ liệu có thể tìm nạp dữ liệu từng ảnh một lần.

3. Gói DataLoader từ tập dữ liệu.

4. Xây dựng mô hình, sau đó xác định hàm sai số và trình tối ưu hóa.

5. Xác định hai hàm để huấn luyện và xác thực một bó dữ liệu tương ứng.

6. Xác định hàm sẽ tính toán độ chính xác của dữ liệu.

7. Thực hiện cập nhật trọng số theo từng bó dữ liệu tăng dần qua từng epoch.

- Các bước thực hiện cụ thể như sau:

1. Import các gói có liên quan và bộ dữ liệu FMNIST:

```
from torch.utils.data import Dataset,
DataLoader import torch import torch.nn as nn
import numpy as np
```

```
import matplotlib.pyplot as plt
%matplotlib inline device = "cuda" if torch.cuda.is_available()
else "cpu" from torchvision import datasets data_folder =
'~/data/FMNIST' # This can be any directory you want to
# download FMNIST to fmnist = datasets.FashionMNIST(data_folder,
download=True, train=True) tr_images = fmnist.data tr_targets =
fmnist.targets
```

2. Xây dựng một lớp tìm nạp tập dữ liệu. Chúng ta cần ba hàm __init__, __getitem__ và __len__:

```
class
FMNISTDataset(Dataset):
def __init__(self, x, y):
x = x.float()          x =
x.view(-1,28*28)
self.x, self.y = x, y
def __getitem__(self, ix):
        x, y = self.x[ix], self.y[ix]
return x.to(device), y.to(device)
def __len__(self):           return
len(self.x)
```

3. Tạo một hàm tạo DataLoader – trn_dl từ tập dữ liệu – được gọi là FMNISTDataset. Điều này sẽ lấy mẫu 32 ảnh ngẫu nhiên cho kích thước bó:

```
def get_data():
    train = FMNISTDataset(tr_images, tr_targets)
trn_dl = DataLoader(train, batch_size=32, shuffle=True)
return trn_dl
```

4. Xác định một mô hình, cũng như hàm sai số và trình tối ưu hóa:

```
from torch.optim import SGD
def get_model():
    model = nn.Sequential(
nn.Linear(28 * 28, 1000),
nn.ReLU(),
        nn.Linear(1000, 10)
    ).to(device)     loss_fn =
nn.CrossEntropyLoss()     optimizer =
SGD(model.parameters(), lr=1e-2)     return
model, loss_fn, optimizer tr_images =
fmnist.data
```

```
tr_targets = fmnist.targets
```

5. Xác định hàm sẽ huấn luyện tập dữ liệu trên một loạt hình ảnh:

```
def train_batch(x, y, model, opt, loss_fn):
    model.train() # <- let's hold on to this until we reach dropout
section     # call your model like any python function on your batch of
inputs     prediction = model(x)     # compute loss     batch_loss =
loss_fn(prediction, y)
    # based on the forward pass in `model(x)` compute all the gradients of
    # 'model.parameters()'
batch_loss.backward()
    # apply new-weights = f(old-weights, old-weight-gradients) where
    # "f" is the optimizer
optimizer.step()
    # Flush gradients memory for next batch of
calculations     optimizer.zero_grad()     return
batch_loss.item()
```

6. Xây dựng hàm tính toán độ chính xác của tập dữ liệu nhất định:

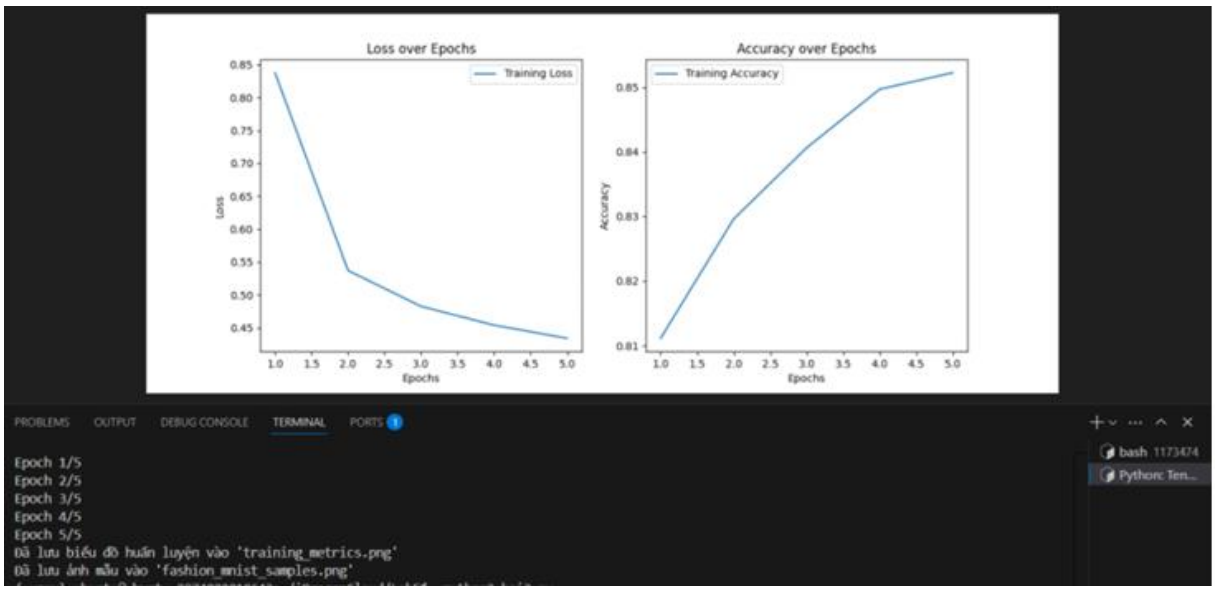
```
@torch.no_grad() def
accuracy(x, y, model):
    model.eval() # <- let's wait till we get to dropout section
# get the prediction matrix for a tensor of `x` images
prediction = model(x)
    # compute if the location of maximum in each row coincides
    # with ground truth
    max_values, argmaxes = prediction.max(-
1)     is_correct = argmaxes == y
return is_correct.cpu().numpy().tolist()
```

7. Huấn luyện mạng nơ-ron bằng cách sử dụng các đoạn code sau:

```
trn_dl = get_data()
model, loss_fn, optimizer =
get_model() losses, accuracies = [],
[] for epoch in range(5):
    print(epoch)     epoch_losses,
epoch_accuracies = [], []
```

```
    for ix, batch in enumerate(iter(trn_dl)):
        x, y = batch         batch_loss = train_batch(x, y,
model, optimizer, loss_fn)
epoch_losses.append(batch_loss)     epoch_loss =
np.array(epoch_losses).mean()     for ix, batch in
enumerate(iter(trn_dl)):
        x, y = batch         is_correct = accuracy(x, y,
model)         epoch_accuracies.extend(is_correct)
epoch_accuracy = np.mean(epoch_accuracies)
losses.append(epoch_loss)
accuracies.append(epoch_accuracy) epochs = np.arange(5)+1
plt.figure(figsize=(20,5)) plt.subplot(121)
plt.title('Loss value over increasing epochs')
plt.plot(epochs, losses, label='Training Loss')
plt.legend() plt.subplot(122) plt.title('Accuracy value
over increasing epochs') plt.plot(epochs, accuracies,
label='Training Accuracy')
plt.gca().set_yticklabels(['{:.0f}%'.format(x*100) for x
in plt.gca().get_yticks()])  plt.legend()
```

Kết quả như sau:



1. Tìm nạp tập dữ liệu cũng như các hình ảnh và gán nhãn vào biến tr_targets như chúng ta đã làm trong phần trước:

```
from torchvision import datasets from torch.utils.data import
Dataset, DataLoader import torch import torch.nn as nn device =
"cuda" if torch.cuda.is_available() else "cpu" import numpy as
np data_folder = '~/data/FMNIST' # This can be any directory you
want
# to download FMNIST to fmnist = datasets.FashionMNIST(data_folder,
download=True, train=True) tr_images = fmnist.data tr_targets =
fmnist.targets
```

1. Sửa đổi FMNISTDataset để tìm nạp dữ liệu sao cho hình ảnh đầu vào được chia cho 255 (cường độ/giá trị tối đa của pixel):

```
class
FMNISTDataset(Dataset):
def __init__(self, x, y):
x = x.float()/255          x
= x.view(-1,28*28)
self.x, self.y = x, y
def __getitem__(self, ix):
        x, y = self.x[ix], self.y[ix]
return x.to(device), y.to(device)
def __len__(self):           return
len(self.x)
```

2. Huấn luyện một mô hình, giống như chúng ta đã làm ở các bước 4, 5, 6 và 7 của phần trước:

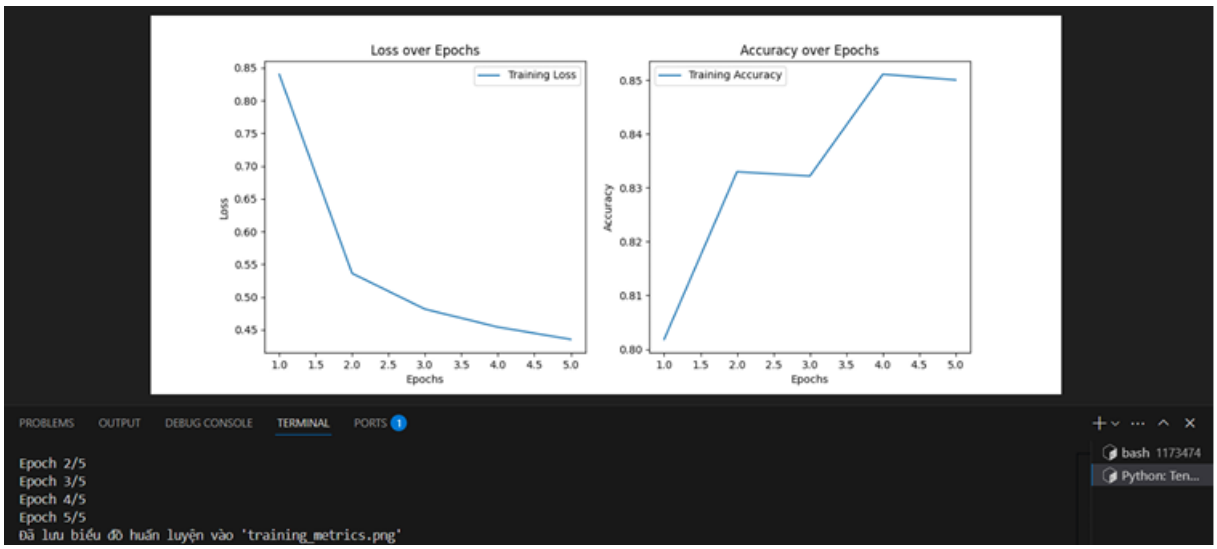
```
def get_data():
    train = FMNISTDataset(tr_images, tr_targets)
trn_dl = DataLoader(train, batch_size=32, shuffle=True)
return trn_dl from torch.optim import SGD def
get_model():
    model = nn.Sequential(
nn.Linear(28 * 28, 1000),
nn.ReLU(),
        nn.Linear(1000, 10)
    ).to(device)     loss_fn =
nn.CrossEntropyLoss()     optimizer =
SGD(model.parameters(), lr=1e-2)     return
model, loss_fn, optimizer def train_batch(x,
y, model, opt, loss_fn):
    model.train()
    # call your model like any python function on your batch of
inputs     prediction = model(x)     # compute loss     batch_loss =
loss_fn(prediction, y)
    # based on the forward pass in `model(x)` compute all the gradients of
    # 'model.parameters()'
batch_loss.backward()
    # apply new-weights = f(old-weights, old-weight-gradients)
    # where "f" is the optimizer
optimizer.step()
    # Flush memory for next batch of
calculations     optimizer.zero_grad()
return batch_loss.item() def accuracy(x, y,
model):
    model.eval()
    # since there's no need for updating weights, we might
    # as well not compute the gradients
with torch.no_grad():
        # get the prediction matrix for a tensor of `x` images
prediction = model(x)
    # compute if the location of maximum in each row coincides
    # with ground truth     max_values,
argmaxes = prediction.max(-1)
is_correct = argmaxes == y     return
is_correct.cpu().numpy().tolist() trn_dl =
get_data() model, loss_fn, optimizer =
get_model()
```

```
losses, accuracies = [], []
for epoch in range(5):
    print(epoch)     epoch_losses,
epoch_accuracies = [], []     for ix, batch
in enumerate(iter(trn_dl)):
        x, y = batch         batch_loss = train_batch(x, y,
model, optimizer, loss_fn)
epoch_losses.append(batch_loss)     epoch_loss =
np.array(epoch_losses).mean()     for ix, batch in
enumerate(iter(trn_dl)):
        x, y = batch         is_correct =
accuracy(x, y, model)
epoch_accuracies.extend(is_correct)
epoch_accuracy = np.mean(epoch_accuracies)
losses.append(epoch_loss)
    accuracies.append(epoch_accuracy)
```

3. Vẽ biểu đồ độ chính xác và sai số:

```
epochs = np.arange(5)+1
import matplotlib.pyplot as plt
%matplotlib inline plt.figure(figsize=(20,5))
plt.subplot(121) plt.title('Loss value over increasing
epochs') plt.plot(epochs, losses, label='Training Loss')
plt.legend() plt.subplot(122) plt.title('Accuracy value
over increasing epochs') plt.plot(epochs, accuracies,
label='Training Accuracy')
plt.gca().set_yticklabels(['{:.0f}%'.format(x*100) for x
in plt.gca().get_yticks()])  plt.legend()
```

Kết quả như sau:



4. Hiểu được tác động của việc thay đổi kích thước bó

Trong phần trước, 32 hình ảnh đã được xem xét mỗi đợt trong tập dữ liệu huấn luyện. Điều này dẫn đến số lượng cập nhật trọng số trên mỗi epoch lớn hơn vì có 1.875 cập nhật trọng số trên mỗi epoch (60.000/32 gần bằng 1.875, trong đó 60.000 là số lượng hình ảnh huấn luyện). Hơn nữa, chúng ta không xem xét hiệu suất của mô hình trên tập

dữ liệu không nhìn thấy (tập dữ liệu xác thực). Chúng ta sẽ khám phá điều này trong phần này. Trong phần này chúng ta sẽ so sánh như sau:

• Giá trị sai số và độ chính xác của dữ liệu huấn luyện và xác thực khi kích thước bó huấn luyện là 32.

• Giá trị sai số và độ chính xác của dữ liệu huấn luyện và xác thực khi kích thước bó huấn luyện là 10.000.

Kích thước bó 32

1. Tải xuống và import các hình ảnh huấn luyện và gán nhãn vào biến tr_targets:

```
from torchvision import datasets import torch data_folder =
'~/data/FMNIST' # This can be any directory you want to
# download FMNIST to fmnist = datasets.FashionMNIST(data_folder,
download=True, train=True) tr_images = fmnist.data tr_targets =
fmnist.targets
```

2. Theo cách tương tự với hình ảnh huấn luyện, chúng ta phải tải xuống và nhập tập dữ liệu xác thực bằng cách chỉ định train = False trong khi gọi phương thức FashionMNIST trong bộ dữ liệu của chúng ta:

```
val_fmnist = datasets.FashionMNIST(data_folder, download=True,
train=False) val_images = val_fmnist.data val_targets =
val_fmnist.targets plt.title('Accuracy value over increasing epochs')
plt.plot(epochs, accuracies, label='Training Accuracy')
plt.gca().set_yticklabels(['{:.0f}%'.format(x*100) for x in
plt.gca().get_yticks()])  plt.legend()
```

3. Import các gói có liên quan và xác định thiết bị:

```
import matplotlib.pyplot as plt
%matplotlib inline import numpy as np from
torch.utils.data import Dataset, DataLoader import
torch import torch.nn as nn device = 'cuda' if
torch.cuda.is_available() else 'cpu'
```

4. Xác định lớp tập dữ liệu (FashionMNIST), các hàm sẽ được sử dụng để huấn luyện trên một bó dữ liệu (train_batch), tính toán độ chính xác (accuracy), sau đó xác định kiến trúc mô hình, hàm sai số và trình tối ưu hóa (get_model). Lưu ý rằng hàm lấy dữ liệu sẽ là hàm duy nhất có sai lệch so với những gì chúng ta đã thấy trong các phần trước (vì chúng ta hiện đang nghiên cứu các tập dữ liệu huấn luyện và xác thực), vì vậy chúng ta sẽ xây dựng nó trong bước tiếp theo:

```
class
FMNISTDataset(Dataset):
def __init__(self, x, y):
x = x.float()/255          x
= x.view(-1,28*28)
self.x, self.y = x, y
def __getitem__(self, ix):
        x, y = self.x[ix], self.y[ix]
```

```
        return x.to(device),
y.to(device)     def __len__(self):
return len(self.x)
 from torch.optim import SGD,
Adam def get_model():
    model = nn.Sequential(
nn.Linear(28 * 28, 1000),
nn.ReLU(),
        nn.Linear(1000, 10)
    ).to(device)
     loss_fn = nn.CrossEntropyLoss()
optimizer = Adam(model.parameters(), lr=1e-2)
return model, loss_fn, optimizer
 def train_batch(x, y, model, opt,
loss_fn):
    model.train()     prediction =
model(x)     batch_loss =
loss_fn(prediction, y)
batch_loss.backward()
optimizer.step()
optimizer.zero_grad()     return
batch_loss.item()
 def accuracy(x, y,
model):
    model.eval()
    # this is the same as @torch.no_grad
    # at the top of function, only difference
# being, grad is not computed in the with scope
with torch.no_grad():
        prediction = model(x)
max_values, argmaxes = prediction.max(-1)
is_correct = argmaxes == y
    return is_correct.cpu().numpy().tolist()
```

5. Xác định hàm sẽ lấy dữ liệu; hàm get_data. Hàm này sẽ trả về dữ liệu huấn luyện với kích thước bó là 32 và tập dữ liệu xác thực có kích thước bó bằng độ dài của dữ liệu xác thực (chúng ta sẽ không sử dụng dữ liệu xác thực để huấn luyện mô hình; chúng ta sẽ chỉ sử dụng dữ liệu đó để hiểu độ chính xác của mô hình khi không nhìn thấy được dữ liệu):

```
def get_data():
    train = FMNISTDataset(tr_images, tr_targets)
trn_dl = DataLoader(train, batch_size=32, shuffle=True)
val = FMNISTDataset(val_images, val_targets)
```

```
   val_dl = DataLoader(val, batch_size=len(val_images),
shuffle=False)     return trn_dl, val_dl
```

6. Xác định hàm tính toán việc mất dữ liệu xác thực; cái đó là, val_loss. Lưu ý rằng chúng ta đang tính toán giá trị này một cách riêng biệt vì mất dữ liệu huấn luyện đang được tính toán trong khi huấn luyện mô hình:

```
@torch.no_grad() def val_loss(x, y,
model):     prediction = model(x)
val_loss = loss_fn(prediction, y)
return val_loss.item()
```

7. Tìm nạp DataLoaders huấn luyện và xác thực. Ngoài ra, hãy khởi tạo mô hình, hàm sai số và trình tối ưu hóa:

```
trn_dl, val_dl = get_data() model,
loss_fn, optimizer = get_model()
```

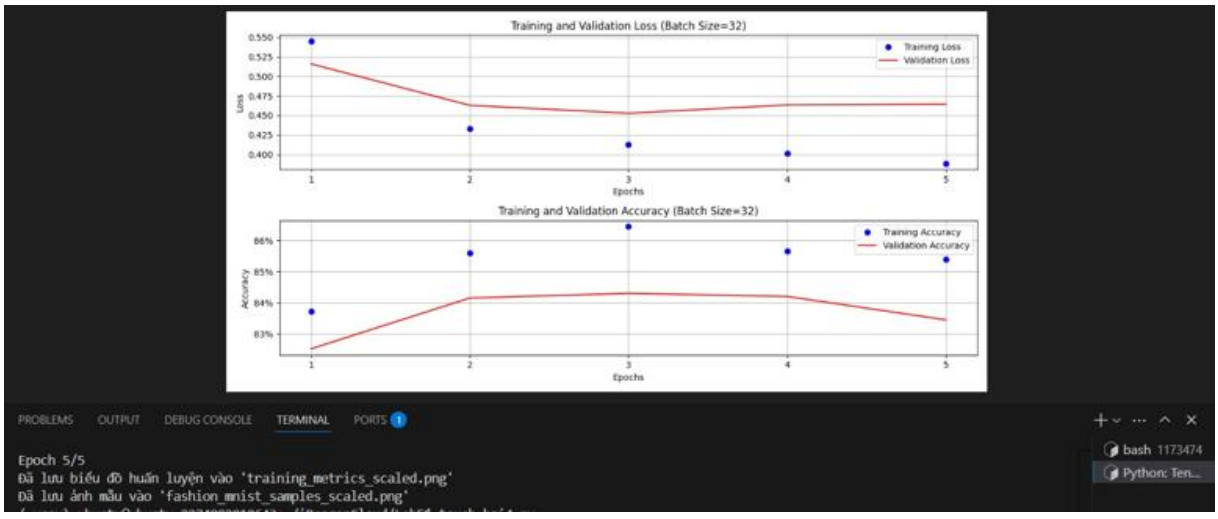
8. Huấn luyện mô hình như sau:

```
train_losses, train_accuracies = [],
[] val_losses, val_accuracies = [],
[] for epoch in range(5):
    print(epoch)     train_epoch_losses,
train_epoch_accuracies = [], []     for ix, batch in
enumerate(iter(trn_dl)):
        x, y = batch         batch_loss = train_batch(x, y,
model, optimizer, loss_fn)
train_epoch_losses.append(batch_loss)      train_epoch_loss =
np.array(train_epoch_losses).mean()
     for ix, batch in
enumerate(iter(trn_dl)):
        x, y = batch         is_correct = accuracy(x, y,
model)         train_epoch_accuracies.extend(is_correct)
train_epoch_accuracy = np.mean(train_epoch_accuracies)
for ix, batch in enumerate(iter(val_dl)):         x, y =
batch         val_is_correct = accuracy(x, y, model)
validation_loss = val_loss(x, y, model)
val_epoch_accuracy = np.mean(val_is_correct)
train_losses.append(train_epoch_loss)
train_accuracies.append(train_epoch_accuracy)
val_losses.append(validation_loss)
val_accuracies.append(val_epoch_accuracy)
```

9. Trực quan hóa sự cải thiện về độ chính xác và giá trị sai số trong bộ dữ liệu huấn luyện và xác thực qua các epoch ngày càng tăng:

```
epochs = np.arange(5)+1
import matplotlib.ticker as mtick
import matplotlib.pyplot as plt
import matplotlib.ticker as mticker
plt.subplot(211)
plt.plot(epochs, train_losses, 'bo', label='Training loss')
plt.plot(epochs, val_losses, 'r', label='Validation loss')
plt.gca().xaxis.set_major_locator(mticker.MultipleLocator(1))
plt.title('Training and validation loss when batch size is 32')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid('off')
plt.show()
plt.subplot(212)
plt.plot(epochs, train_accuracies, 'bo', label='Trainingaccuracy')
plt.plot(epochs, val_accuracies, 'r', label='Validationaccuracy')
plt.gca().xaxis.set_major_locator(mticker.MultipleLocator(1))
plt.title('Training and validation accuracy when batch size is 32')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.gca().set_yticklabels(['{:.0f}%'.format(x*100) for x in
                           plt.gca().get_yticks()])
plt.legend()
plt.grid('off')
plt.show()
```

Kết quả như sau:



Như bạn có thể thấy, độ chính xác trong quá trình huấn luyện và xác thực là ~85% vào cuối 5 epoch khi kích thước bó là 32. Tiếp theo, chúng ta sẽ thay đổi

tham số batch_size khi huấn luyện DataLoader trong hàm get_data để xem tác động của nó đến độ chính xác kết thúc tại cuối 5 epoch. **Kích thước bó 10.000:**

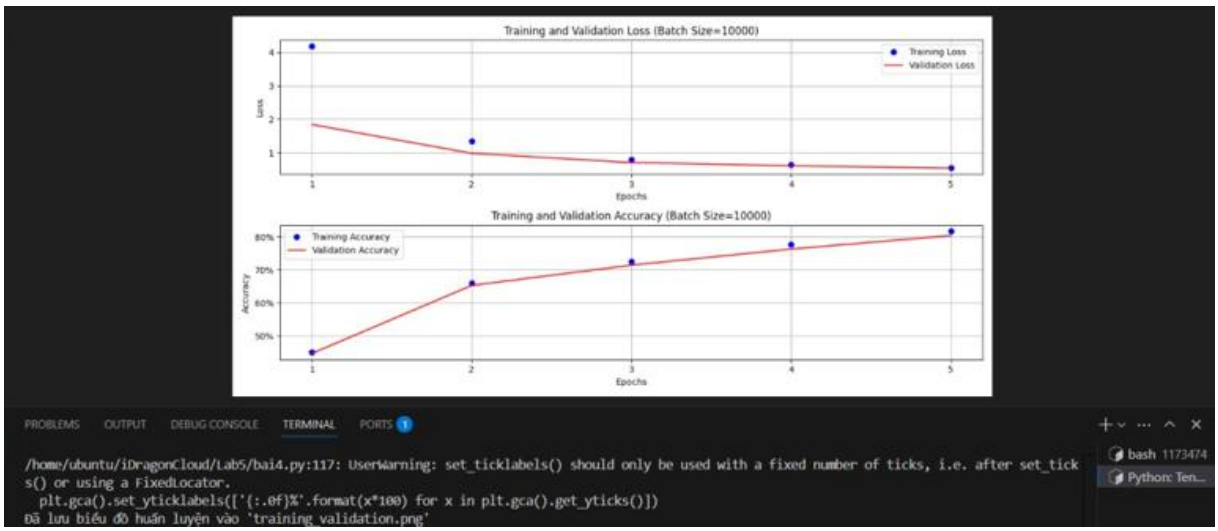
Trong phần này, chúng ta sẽ sử dụng 10.000 ảnh mỗi đợt để có thể hiểu tác động của việc thay đổi quy mô bó.

Chúng ta sẽ sửa đổi get_data để nó có kích thước bó là 10.000 trong khi tìm nạp DataLoader huấn luyện từ tập dữ liệu huấn luyện, như sau:

```
def get_data():
    train = FMNISTDataset(tr_images, tr_targets)      trn_dl =
DataLoader(train, batch_size=10000, shuffle=True)     val =
FMNISTDataset(val_images, val_targets)      val_dl =
DataLoader(val, batch_size=len(val_images), shuffle=False)
return trn_dl, val_dl trn_dl, val_dl = get_data() model, loss_fn,
optimizer = get_model() train_losses, train_accuracies = [], []
val_losses, val_accuracies = [], [] for epoch in range(5):
print(epoch)     train_epoch_losses, train_epoch_accuracies = [],
[]     for ix, batch in enumerate(iter(trn_dl)):
        x, y = batch         batch_loss = train_batch(x, y,
model, optimizer, loss_fn)
train_epoch_losses.append(batch_loss)      train_epoch_loss
= np.array(train_epoch_losses).mean()      for ix, batch in
enumerate(iter(trn_dl)):
        x, y = batch         is_correct = accuracy(x, y,
model)         train_epoch_accuracies.extend(is_correct)
train_epoch_accuracy  =  np.mean(train_epoch_accuracies)
for ix, batch in enumerate(iter(val_dl)):         x, y =
batch            val_is_correct = accuracy(x, y, model)
validation_loss     =     val_loss(x,     y,     model)
val_epoch_accuracy       =       np.mean(val_is_correct)
train_losses.append(train_epoch_loss)
train_accuracies.append(train_epoch_accuracy)
val_losses.append(validation_loss)
val_accuracies.append(val_epoch_accuracy)    epochs    =
np.arange(5)+1 import matplotlib.ticker as mtick import
matplotlib.pyplot  as  plt  import  matplotlib.ticker  as
mticker
%matplotlib inline
```

```
plt.subplot(211) plt.plot(epochs, train_losses, 'bo',
label='Training loss') plt.plot(epochs, val_losses, 'r',
label='Validation loss')
plt.gca().xaxis.set_major_locator(mticker.MultipleLocator(1))
plt.title('Training and validation loss when batch size is 10000')
plt.xlabel('Epochs') plt.ylabel('Loss') plt.legend() plt.grid('off')
plt.show() plt.subplot(212) plt.plot(epochs, train_accuracies, 'bo',
label='Training accuracy') plt.plot(epochs, val_accuracies, 'r',
label='Validation accuracy')
plt.gca().xaxis.set_major_locator(mticker.MultipleLocator(1))
plt.title('Training and validation accuracy when batch size is
10000') plt.xlabel('Epochs') plt.ylabel('Accuracy')
plt.gca().set_yticklabels(['{:.0f}%'.format(x*100) for x in
plt.gca().get_yticks()])  plt.legend() plt.grid('off') plt.show()
```

Kết quả như sau:



Hiểu được tác động của việc thay đổi trình tối ưu hóa sai số.

Chúng ta đã tối ưu hóa sai số dựa trên trình tối ưu hóa Adam. Trong phần này, chúng ta sẽ làm như sau:

- Sửa đổi trình tối ưu hóa để nó trở thành trình tối ưu hóa giảm dần độ dốc ngẫu nhiên (SGD)
- Kích thước bó là 32 khi tìm nạp dữ liệu trong DataLoader
- Tăng số epoch lên 10 (để chúng ta có thể so sánh hiệu suất của SGD và Adam trong một số epoch lớn hơn)

Thực hiện những thay đổi này có nghĩa là chỉ một bước trong kích thước bó là 32 sẽ thay đổi; nghĩa là chúng ta sẽ sửa đổi trình tối ưu hóa thành trình tối ưu hóa SGD.

Sửa đổi trình tối ưu hóa mà ta đang sử dụng thành trình tối ưu hóa SGD trong hàm get_model trong khi vẫn đảm bảo rằng mọi thứ khác vẫn giữ nguyên:

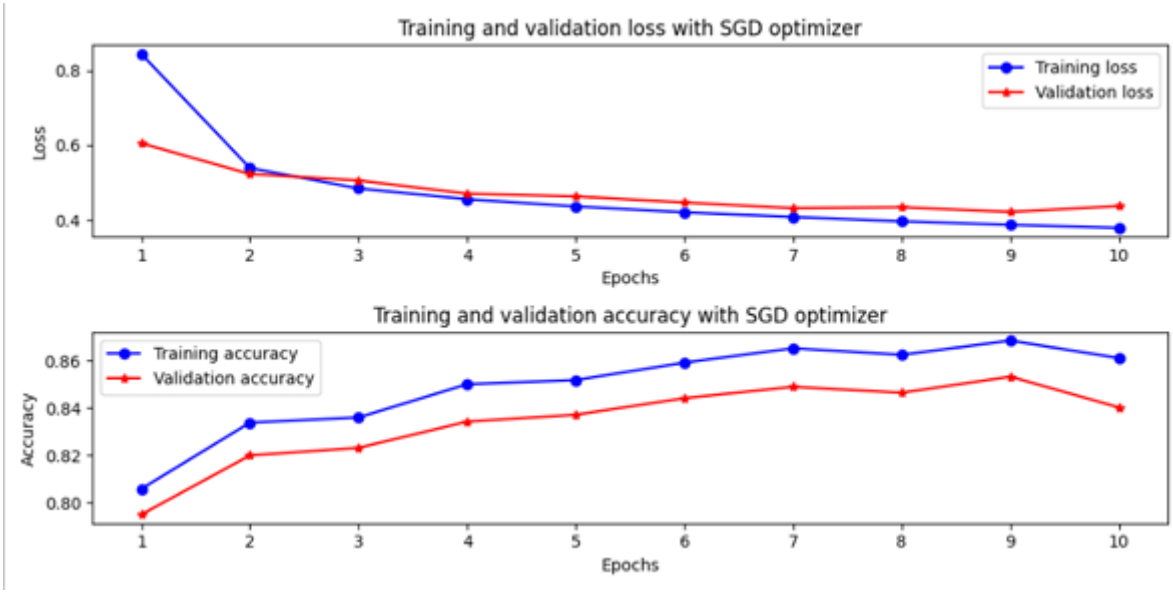
```
from torchvision import datasets import torch data_folder =
'~/data/FMNIST' # This can be any directory you want to
# download FMNIST to fmnist = datasets.FashionMNIST(data_folder,
download=True, train=True) tr_images = fmnist.data tr_targets =
fmnist.targets
 val_fmnist = datasets.FashionMNIST(data_folder, download=True,
train=False) val_images = val_fmnist.data val_targets =
val_fmnist.targets
 import matplotlib.pyplot as
plt
%matplotlib inline import numpy as np from
torch.utils.data import Dataset, DataLoader import
torch import torch.nn as nn device = 'cuda' if
torch.cuda.is_available() else 'cpu'
# SGD optimizer class
FMNISTDataset(Dataset):
def __init__(self, x, y):
x = x.float()/255          x
= x.view(-1,28*28)
self.x, self.y = x, y
def __getitem__(self, ix):
        x, y = self.x[ix], self.y[ix]
return x.to(device), y.to(device)
```

```
    def __len__(self):
return len(self.x)
 from torch.optim import SGD,
Adam def get_model():
    model = nn.Sequential(
nn.Linear(28 * 28, 1000),
nn.ReLU(),
        nn.Linear(1000, 10)
    ).to(device)
     loss_fn = nn.CrossEntropyLoss()
optimizer = SGD(model.parameters(), lr=1e-2)
return model, loss_fn, optimizer
 def train_batch(x, y, model, opt,
loss_fn):
    model.train()     prediction =
model(x)     batch_loss =
loss_fn(prediction, y)
batch_loss.backward()
optimizer.step()
optimizer.zero_grad()     return
batch_loss.item()
 def accuracy(x, y,
model):
    model.eval()
    # this is the same as @torch.no_grad
    # at the top of function, only difference
# being, grad is not computed in the with scope
with torch.no_grad():
        prediction = model(x)
    max_values, argmaxes = prediction.max(-1)
is_correct = argmaxes == y
    return is_correct.cpu().numpy().tolist()
 def
get_data():
    train = FMNISTDataset(tr_images, tr_targets)      trn_dl =
DataLoader(train, batch_size=32, shuffle=True)     val =
FMNISTDataset(val_images, val_targets)      val_dl = DataLoader(val,
batch_size=len(val_images), shuffle=False)     return trn_dl, val_dl
@torch.no_grad() def
val_loss(x, y, model):
prediction = model(x)
```

```
    val_loss = loss_fn(prediction, y)
return val_loss.item() trn_dl, val_dl =
get_data() model, loss_fn, optimizer =
get_model()
 train_losses, train_accuracies = [],
[] val_losses, val_accuracies = [], []
for epoch in range(10):
    print(epoch)     train_epoch_losses,
train_epoch_accuracies = [], []     for ix, batch in
enumerate(iter(trn_dl)):
        x, y = batch         batch_loss = train_batch(x, y,
model, optimizer, loss_fn)
train_epoch_losses.append(batch_loss)      train_epoch_loss =
np.array(train_epoch_losses).mean()
     for ix, batch in
enumerate(iter(trn_dl)):
        x, y = batch         is_correct = accuracy(x, y,
model)         train_epoch_accuracies.extend(is_correct)
train_epoch_accuracy = np.mean(train_epoch_accuracies)
for ix, batch in enumerate(iter(val_dl)):         x, y =
batch         val_is_correct = accuracy(x, y, model)
validation_loss = val_loss(x, y, model)
val_epoch_accuracy = np.mean(val_is_correct)
train_losses.append(train_epoch_loss)
train_accuracies.append(train_epoch_accuracy)
val_losses.append(validation_loss)
val_accuracies.append(val_epoch_accuracy)

epochs = np.arange(10)+1 import
matplotlib.ticker as mtick import
matplotlib.pyplot as plt import
matplotlib.ticker as mticker
%matplotlib inline plt.subplot(211)
plt.plot(epochs, train_losses, 'bo', label='Training loss')
plt.plot(epochs, val_losses, 'r', label='Validation loss')
plt.gca().xaxis.set_major_locator(mticker.MultipleLocator(1))
plt.title('Training and validation loss with SGD optimizer')
plt.xlabel('Epochs') plt.ylabel('Loss') plt.legend()
```

```
plt.grid('off') plt.show() plt.subplot(212) plt.plot(epochs,
train_accuracies, 'bo', label='Training accuracy') plt.plot(epochs,
val_accuracies, 'r', label='Validation accuracy')
plt.gca().xaxis.set_major_locator(mticker.MultipleLocator(1))
plt.title('Training and validation accuracy with SGD optimizer')
plt.xlabel('Epochs') plt.ylabel('Accuracy')
plt.gca().set_yticklabels(['{:.0f}%'.format(x*100) for x in
plt.gca().get_yticks()])  plt.legend() plt.grid('off') plt.show()
```

Kết quả như sau:



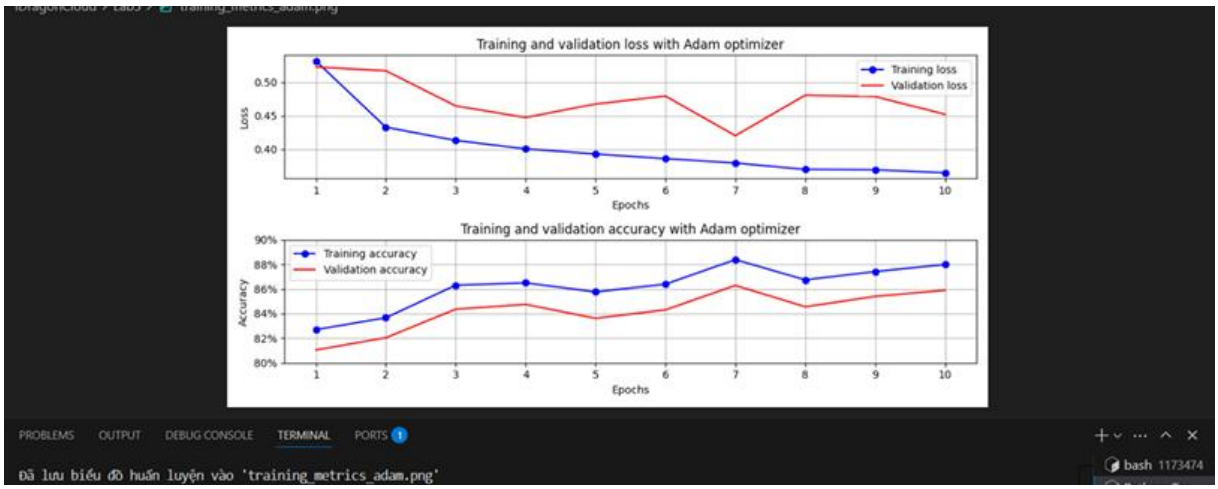
Adam optimizer

```
from torch.optim import SGD, Adam
def get_model():
    model = nn.Sequential(
nn.Linear(28 * 28, 1000),
nn.ReLU(),
        nn.Linear(1000, 10)
    ).to(device)

    loss_fn = nn.CrossEntropyLoss()
    optimizer = Adam(model.parameters(), lr=1e-
2)     return model, loss_fn, optimizer trn_dl,
val_dl = get_data() model, loss_fn, optimizer =
get_model()
```

```
train_losses, train_accuracies = [], []
val_losses, val_accuracies = [], [] for
epoch in range(10):
    print(epoch)     train_epoch_losses,
train_epoch_accuracies = [], []     for ix, batch in
enumerate(iter(trn_dl)):
        x, y = batch         batch_loss = train_batch(x, y,
model, optimizer, loss_fn)
train_epoch_losses.append(batch_loss)      train_epoch_loss =
np.array(train_epoch_losses).mean()
     for ix, batch in
enumerate(iter(trn_dl)):
        x, y = batch         is_correct = accuracy(x, y,
model)         train_epoch_accuracies.extend(is_correct)
train_epoch_accuracy = np.mean(train_epoch_accuracies)
for ix, batch in enumerate(iter(val_dl)):         x, y =
batch         val_is_correct = accuracy(x, y, model)
validation_loss = val_loss(x, y, model)
val_epoch_accuracy = np.mean(val_is_correct)
train_losses.append(train_epoch_loss)
train_accuracies.append(train_epoch_accuracy)
val_losses.append(validation_loss)
val_accuracies.append(val_epoch_accuracy) epochs =
np.arange(10)+1 import matplotlib.ticker as mtick import
matplotlib.pyplot as plt import matplotlib.ticker as
mticker
%matplotlib inline plt.subplot(211)
plt.plot(epochs, train_losses, 'bo', label='Training loss')
plt.plot(epochs, val_losses, 'r', label='Validation loss')
plt.gca().xaxis.set_major_locator(mticker.MultipleLocator(1))
plt.title('Training and validation loss with Adam optimizer')
plt.xlabel('Epochs') plt.ylabel('Loss') plt.legend()
plt.grid('off') plt.show() plt.subplot(212)
plt.plot(epochs, train_accuracies, 'bo', label='Training accuracy')
plt.plot(epochs, val_accuracies, 'r', label='Validation accuracy')
plt.gca().xaxis.set_major_locator(mticker.MultipleLocator(1))
```

```
plt.title('Training and validation accuracy with Adam optimizer')
plt.xlabel('Epochs') plt.ylabel('Accuracy')
plt.gca().set_yticklabels(['{:.0f}%'.format(x*100) for x in
plt.gca().get_yticks()])  plt.legend() plt.grid('off') plt.show()
```

Kết quả như sau:



Như chúng ta có thể thấy, khi chúng ta sử dụng trình tối ưu hóa Adam, độ chính xác vẫn rất gần với 85%. Tuy nhiên, lưu ý rằng, tốc độ học là 0,01. Trong phần tiếp theo, chúng ta sẽ tìm hiểu về tác động của tốc độ học đối với độ chính xác của tập dữ liệu xác thực.

Hiểu tác động của việc thay đổi tốc độ học.

Để hiểu tác động của tốc độ học khác nhau, chúng ta sẽ xem xét tình huống sau:

- Tốc độ học cao hơn (0,1) trên tập dữ liệu được chia tỷ lệ
- Tốc độ học thấp hơn (0,00001) trên tập dữ liệu được chia tỷ lệ
- Tốc độ học thấp hơn (0,001) trên tập dữ liệu không được chia tỷ lệ
- Tốc độ học cao hơn (0,1) trên tập dữ liệu không được chia tỷ lệ

Nhìn chung, trong phần này, chúng ta sẽ tìm hiểu về tác động của các giá trị tốc độ học khác nhau đối với các tập dữ liệu được chia tỷ lệ và không chia tỷ lệ.

Tác động của tốc độ học lên tập dữ liệu được chia tỷ lệ

Trong phần này, chúng ta sẽ so sánh độ chính xác của tập dữ liệu huấn luyện và xác thực với các tốc độ học sau:

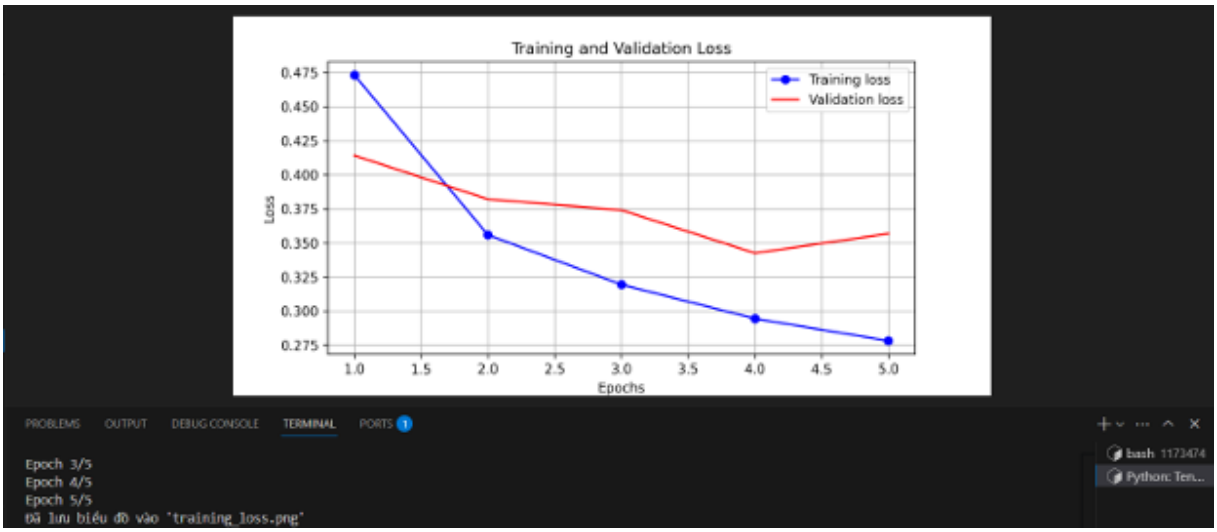
- Tốc độ học cao
- Tốc độ học trung bình
- Tốc độ học thấp

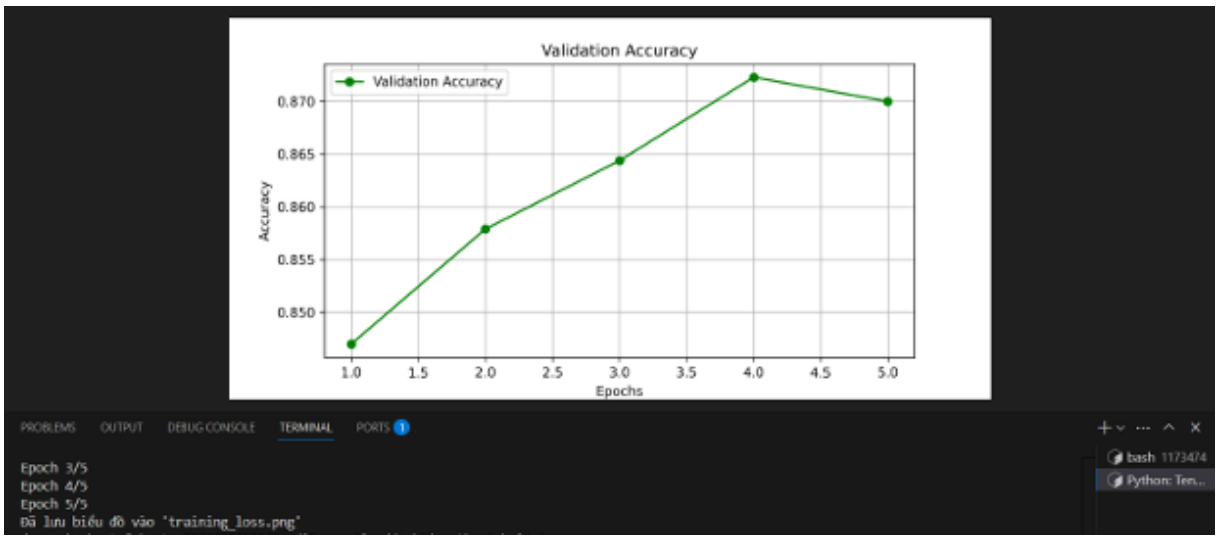
```
from torchvision import datasets import torch data_folder =
'~/data/FMNIST' # This can be any directory you want to
# download FMNIST to fmnist = datasets.FashionMNIST(data_folder,
download=True, train=True) tr_images = fmnist.data tr_targets =
fmnist.targets
 val_fmnist = datasets.FashionMNIST(data_folder, download=True,
train=False) val_images = val_fmnist.data val_targets =
val_fmnist.targets
 import matplotlib.pyplot as
plt
%matplotlib inline import numpy as np from
torch.utils.data import Dataset, DataLoader import
torch import torch.nn as nn device = 'cuda' if
torch.cuda.is_available() else 'cpu'
# High Learning Rate class
FMNISTDataset(Dataset):
def __init__(self, x, y):
x = x.float()/255         x
= x.view(-1,28*28)
self.x, self.y = x, y
def __getitem__(self, ix):
        x, y = self.x[ix], self.y[ix]
return x.to(device), y.to(device)
def __len__(self):          return
len(self.x)

from torch.optim import SGD, Adam
def get_model():
    model = nn.Sequential(
nn.Linear(28 * 28, 1000),
nn.ReLU(),
        nn.Linear(1000, 10)
    ).to(device)
     loss_fn = nn.CrossEntropyLoss()
optimizer = Adam(model.parameters(), lr=1e-1)
return model, loss_fn, optimizer
```

```
 def train_batch(x, y, model, opt,
loss_fn):
    model.train()     prediction =
model(x)     batch_loss =
loss_fn(prediction, y)
batch_loss.backward()
optimizer.step()
optimizer.zero_grad()     return
batch_loss.item()
 def accuracy(x, y,
model):
    model.eval()
    # this is the same as @torch.no_grad
    # at the top of function, only difference
# being, grad is not computed in the with scope
with torch.no_grad():
        prediction = model(x)
max_values, argmaxes = prediction.max(-1)
is_correct = argmaxes == y     return
is_correct.cpu().numpy().tolist() def
get_data():
    train = FMNISTDataset(tr_images, tr_targets)      trn_dl =
DataLoader(train, batch_size=32, shuffle=True)     val =
FMNISTDataset(val_images, val_targets)      val_dl = DataLoader(val,
batch_size=len(val_images), shuffle=False)     return trn_dl, val_dl

@torch.no_grad() def val_loss(x, y,
model):     prediction = model(x)
val_loss = loss_fn(prediction, y)
return val_loss.item()

trn_dl, val_dl = get_data()
model, loss_fn, optimizer = get_model()
 train_losses, train_accuracies = [],
[] val_losses, val_accuracies = [],
[] for epoch in range(5):
    print(epoch)     train_epoch_losses,
train_epoch_accuracies = [], []     for ix, batch in
enumerate(iter(trn_dl)):
        x, y = batch         batch_loss = train_batch(x, y,
model, optimizer, loss_fn)
```

```
        train_epoch_losses.append(batch_loss)
train_epoch_loss = np.array(train_epoch_losses).mean()
     for ix, batch in
enumerate(iter(trn_dl)):
        x, y = batch         is_correct = accuracy(x, y,
model)         train_epoch_accuracies.extend(is_correct)
train_epoch_accuracy = np.mean(train_epoch_accuracies)
for ix, batch in enumerate(iter(val_dl)):         x, y =
batch         val_is_correct = accuracy(x, y, model)
validation_loss = val_loss(x, y, model)
val_epoch_accuracy = np.mean(val_is_correct)
train_losses.append(train_epoch_loss)
train_accuracies.append(train_epoch_accuracy)
val_losses.append(validation_loss)
val_accuracies.append(val_epoch_accuracy) epochs =
np.arange(5)+1 import matplotlib.ticker as mtick import
matplotlib.pyplot as plt import matplotlib.ticker as
mticker
%matplotlib inline plt.subplot(211) plt.plot(epochs,
train_losses, 'bo', label='Training loss') plt.plot(epochs,
val_losses, 'r', label='Validation loss')
plt.gca().xaxis.set_major_locator(mticker.MultipleLocator(1))
plt.title('Training and validation loss with 0.1 learning
rate') plt.xlabel('Epochs') plt.ylabel('Loss') plt.legend()
plt.grid('off') plt.show() plt.subplot(212)
plt.plot(epochs, train_accuracies, 'bo', label='Training
accuracy') plt.plot(epochs, val_accuracies, 'r', label='Validation
accuracy')
plt.gca().xaxis.set_major_locator(mticker.MultipleLocator(1))
plt.title('Training and validation accuracy with 0.1 learning
rate') plt.xlabel('Epochs') plt.ylabel('Accuracy')
plt.gca().set_yticklabels(['{:.0f}%'.format(x*100) for x in
plt.gca().get_yticks()])  plt.legend() plt.grid('off') plt.show()
```

Kết quả như sau:





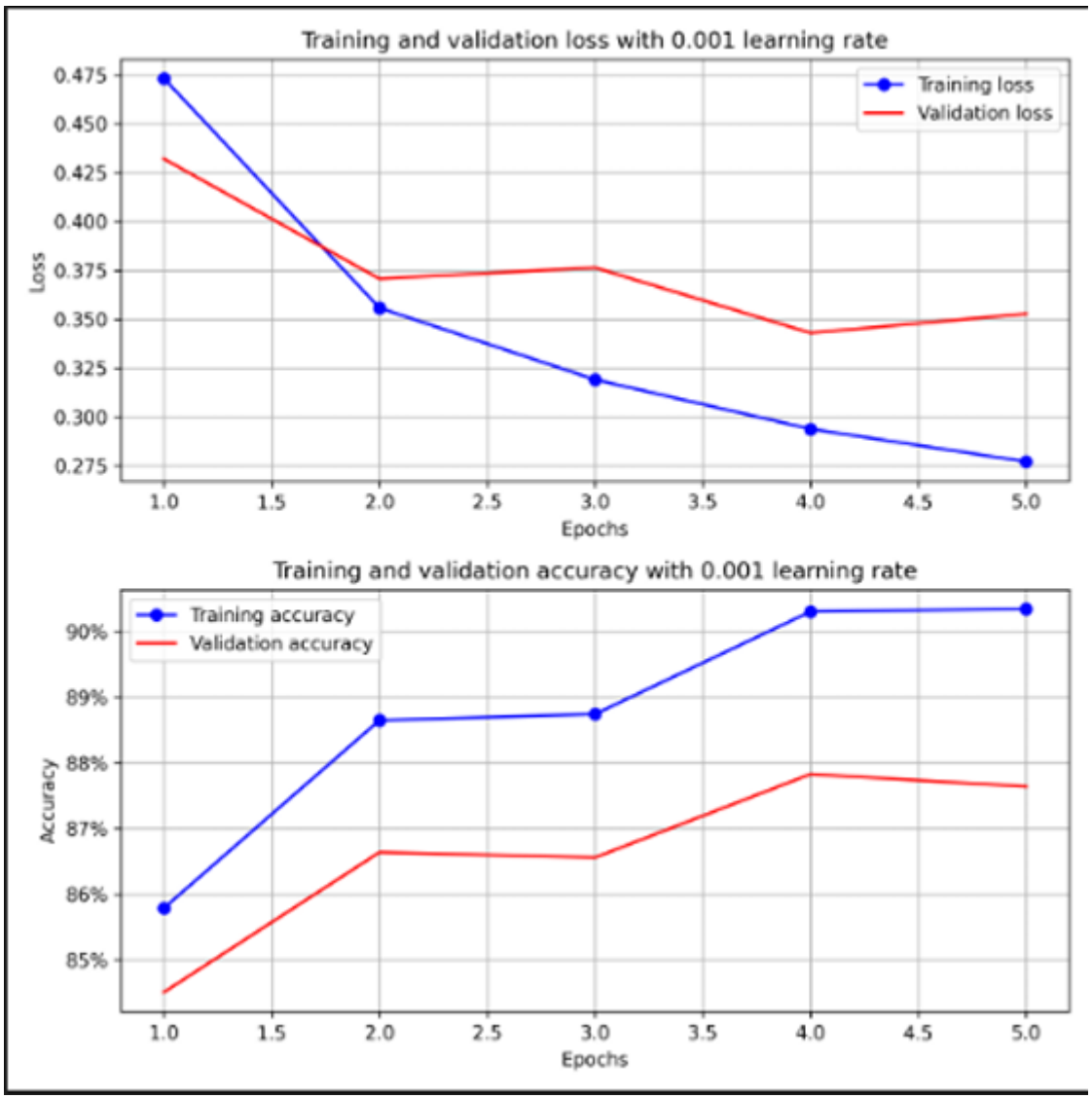
Tốc độ học trung bình

```
def get_model():
    model = nn.Sequential(
nn.Linear(28 * 28, 1000),
nn.ReLU(),
        nn.Linear(1000, 10)
    ).to(device)
     loss_fn = nn.CrossEntropyLoss()
optimizer = Adam(model.parameters(), lr=1e-3)
return model, loss_fn, optimizer trn_dl, val_dl
= get_data()
model, loss_fn, optimizer = get_model()
 train_losses, train_accuracies = [],
[] val_losses, val_accuracies = [],
[] for epoch in range(5):
    print(epoch)     train_epoch_losses,
train_epoch_accuracies = [], []     for ix, batch in
enumerate(iter(trn_dl)):
```

```
        x, y = batch         batch_loss = train_batch(x, y,
model, optimizer, loss_fn)
train_epoch_losses.append(batch_loss)      train_epoch_loss =
np.array(train_epoch_losses).mean()
     for ix, batch in
enumerate(iter(trn_dl)):
        x, y = batch         is_correct = accuracy(x, y,
model)         train_epoch_accuracies.extend(is_correct)
train_epoch_accuracy = np.mean(train_epoch_accuracies)
for ix, batch in enumerate(iter(val_dl)):         x, y =
batch         val_is_correct = accuracy(x, y, model)
validation_loss = val_loss(x, y, model)
val_epoch_accuracy = np.mean(val_is_correct)
train_losses.append(train_epoch_loss)
train_accuracies.append(train_epoch_accuracy)
val_losses.append(validation_loss)
val_accuracies.append(val_epoch_accuracy)
 epochs = np.arange(5)+1 import
matplotlib.ticker as mtick import
matplotlib.pyplot as plt import
matplotlib.ticker as mticker
%matplotlib inline plt.subplot(211) plt.plot(epochs, train_losses,
'bo', label='Training loss') plt.plot(epochs, val_losses, 'r',
label='Validation loss')
plt.gca().xaxis.set_major_locator(mticker.MultipleLocator(1))
plt.title('Training and validation loss with 0.001 learning rate')
plt.xlabel('Epochs') plt.ylabel('Loss') plt.legend() plt.grid('off')
plt.show() plt.subplot(212) plt.plot(epochs, train_accuracies, 'bo',
label='Training accuracy') plt.plot(epochs, val_accuracies, 'r',
label='Validation accuracy')
plt.gca().xaxis.set_major_locator(mticker.MultipleLocator(1))
plt.title('Training and validation accuracy with 0.001 learning
rate') plt.xlabel('Epochs') plt.ylabel('Accuracy')
plt.gca().set_yticklabels(['{:.0f}%'.format(x*100) for x in
plt.gca().get_yticks()])
```

```
plt.legend()
plt.grid('off')
plt.show()
```

Kết quả như sau:

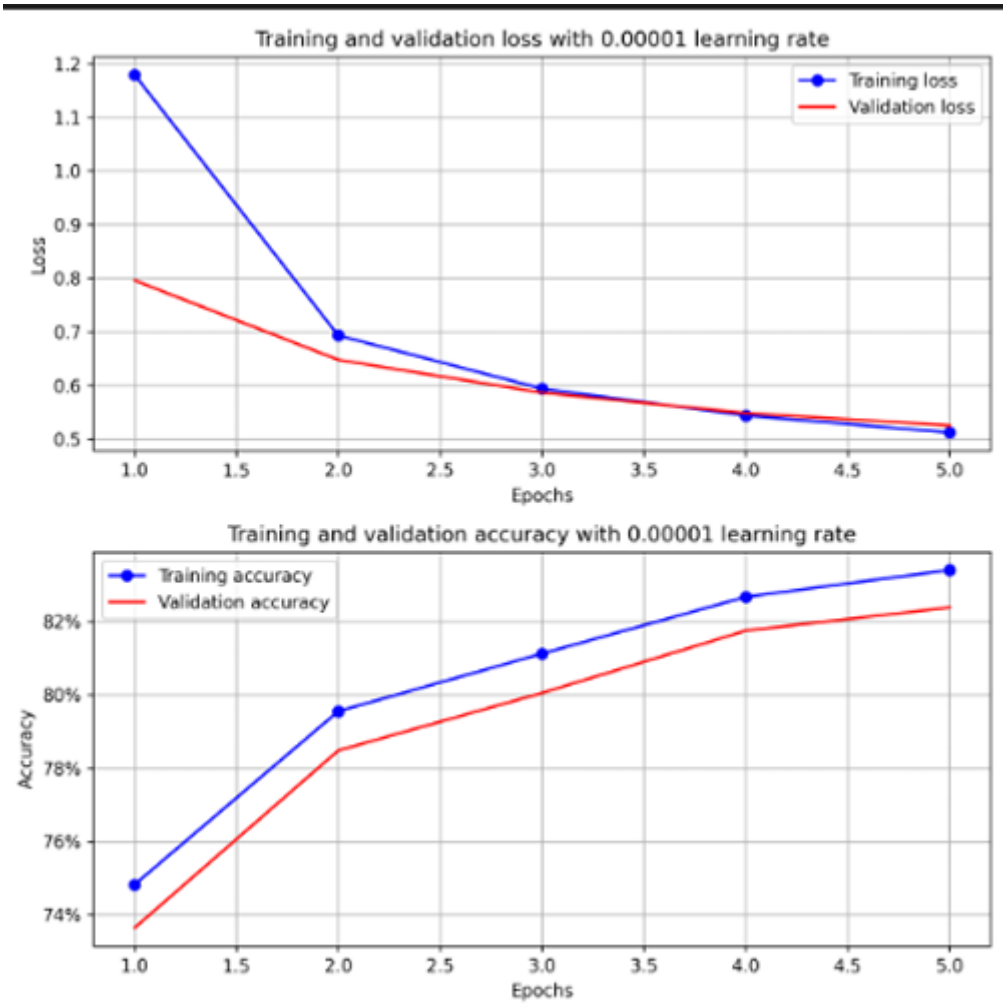


Tốc độ học thấp

```
def get_model():
    model = nn.Sequential(
nn.Linear(28 * 28, 1000),
nn.ReLU(),
        nn.Linear(1000, 10)
    ).to(device)
     loss_fn =
nn.CrossEntropyLoss()
    optimizer = Adam(model.parameters(), lr=1e-5)
return model, loss_fn, optimizer

trn_dl, val_dl = get_data()
model, loss_fn, optimizer = get_model()
 train_losses, train_accuracies = [],
[] val_losses, val_accuracies = [],
[] for epoch in range(5):
    print(epoch)     train_epoch_losses,
train_epoch_accuracies = [], []     for ix, batch in
enumerate(iter(trn_dl)):
        x, y = batch
```

```
        batch_loss = train_batch(x, y, model, optimizer,
loss_fn)         train_epoch_losses.append(batch_loss)
train_epoch_loss = np.array(train_epoch_losses).mean()     for
ix, batch in enumerate(iter(trn_dl)):
        x, y = batch         is_correct = accuracy(x, y,
model)         train_epoch_accuracies.extend(is_correct)
train_epoch_accuracy = np.mean(train_epoch_accuracies)
for ix, batch in enumerate(iter(val_dl)):         x, y =
batch         val_is_correct = accuracy(x, y, model)
validation_loss = val_loss(x, y, model)
val_epoch_accuracy = np.mean(val_is_correct)
train_losses.append(train_epoch_loss)
train_accuracies.append(train_epoch_accuracy)
val_losses.append(validation_loss)
val_accuracies.append(val_epoch_accuracy) epochs =
np.arange(5)+1 import matplotlib.ticker as mtick import
matplotlib.pyplot as plt import matplotlib.ticker as
mticker
%matplotlib inline plt.subplot(211) plt.plot(epochs, train_losses,
'bo', label='Training loss') plt.plot(epochs, val_losses, 'r',
label='Validation loss')
plt.gca().xaxis.set_major_locator(mticker.MultipleLocator(1))
plt.title('Training and validation loss with 0.00001 learning
rate') plt.xlabel('Epochs') plt.ylabel('Loss') plt.legend()
plt.grid('off') plt.show() plt.subplot(212)
plt.plot(epochs, train_accuracies, 'bo', label='Training accuracy')
plt.plot(epochs, val_accuracies, 'r', label='Validation accuracy')
plt.gca().xaxis.set_major_locator(mticker.MultipleLocator(1))
plt.title('Training and validation accuracy with 0.00001 learning
rate') plt.xlabel('Epochs') plt.ylabel('Accuracy')
plt.gca().set_yticklabels(['{:.0f}%'.format(x*100) for x in
plt.gca().get_yticks()])  plt.legend() plt.grid('off') plt.show()
```

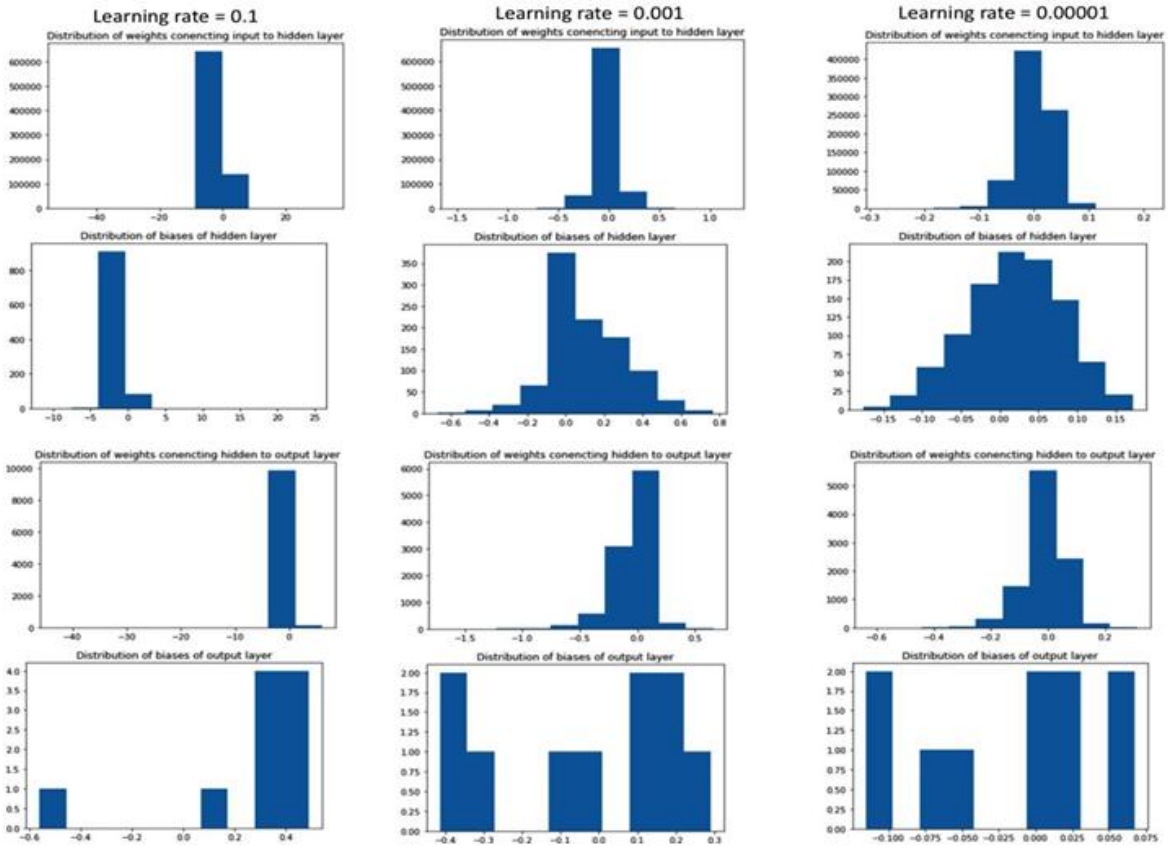
Kết quả như sau:



Từ các biểu đồ trước, chúng ta có thể thấy rằng mô hình học chậm hơn nhiều so với kịch bản trước đó (tốc độ học trung bình). Ở đây, phải mất ~ 100 epoch để đạt được độ chính xác ~ 89% so với 8 epoch khi tốc độ học là 0,001.

Ngoài ra, chúng ta cũng cần lưu ý rằng khoảng cách giữa sai số huấn luyện và xác thực thấp hơn nhiều khi tốc độ học thấp so với kịch bản trước đó (khi tồn tại khoảng cách tương tự vào cuối epoch 4). Lý do là do cập nhật trọng số thấp hơn nhiều khi tốc độ học thấp, điều đó có nghĩa là khoảng cách giữa quá trình huấn luyện và mất xác thực không mở rộng nhanh chóng. Chúng ta đã tìm hiểu về tác động của tốc độ học đối với độ chính xác của tập dữ liệu huấn luyện và xác thực. Trong phần tiếp theo, chúng ta sẽ tìm hiểu cách phân bổ các giá trị trọng số khác nhau giữa các lớp đối với các giá trị tốc độ học khác nhau.

Phân phối tham số giữa các lớp cho các tốc độ học khác nhau



• Khi tốc độ học cao, các tham số có phân bố lớn hơn nhiều so với tốc độ học trung bình và thấp.

• Khi các tham số có phân bố lớn hơn, tình trạng quá khớp (overfitting) sẽ xảy ra.

Chúng ta đã nghiên cứu tác động của việc thay đổi tốc độ học trên một mô hình đã được huấn luyện trên tập dữ liệu được chia tỷ lệ.

Trong phần tiếp theo, chúng ta sẽ tìm hiểu về tác động của việc thay đổi tốc độ học lên một mô hình đã được huấn luyện trên dữ liệu không chia tỷ lệ.

Tác động của việc thay đổi tốc độ học trên tập dữ liệu không được chia tỷ lệ

Trong phần này, chúng ta sẽ quay lại làm việc trên một tập dữ liệu bằng cách không thực hiện chia cho 255 trong lớp mà chúng ta xác định tập dữ liệu. Điều này có thể được thực hiện như sau:

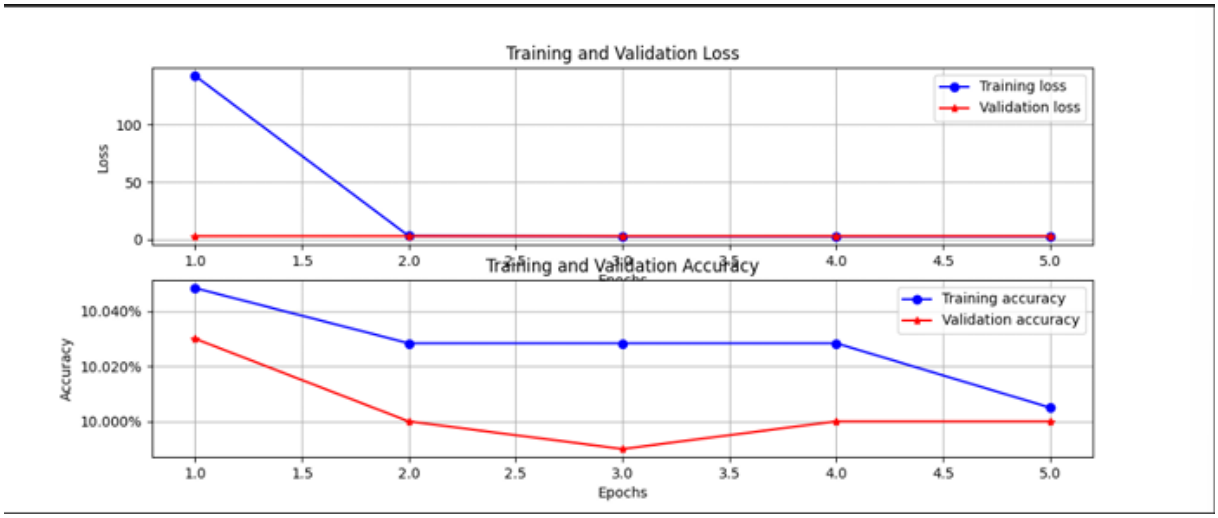
```
from torchvision import datasets import torch data_folder =
'~/data/FMNIST' # This can be any directory you want to
# download FMNIST to
fmnist = datasets.FashionMNIST(data_folder, download=True,
train=True) tr_images = fmnist.data tr_targets = fmnist.targets
```

```
 val_fmnist = datasets.FashionMNIST(data_folder, download=True,
train=False) val_images = val_fmnist.data val_targets =
val_fmnist.targets
 import matplotlib.pyplot as
plt
%matplotlib inline import numpy as np from
torch.utils.data import Dataset, DataLoader import
torch import torch.nn as nn device = 'cuda' if
torch.cuda.is_available() else 'cpu'
#Hingh Learning Rate class
FMNISTDataset(Dataset):
def __init__(self, x, y):
x = x.float()         x =
x.view(-1,28*28)
self.x, self.y = x, y
def __getitem__(self, ix):
        x, y = self.x[ix], self.y[ix]
return x.to(device), y.to(device)
def __len__(self):          return
len(self.x)
 from torch.optim import SGD,
Adam def get_model():
    model = nn.Sequential(
nn.Linear(28 * 28, 1000),
nn.ReLU(),
        nn.Linear(1000, 10)
    ).to(device)

    loss_fn = nn.CrossEntropyLoss()
    optimizer = Adam(model.parameters(), lr=1e-1)
return model, loss_fn, optimizer
 def train_batch(x, y, model, opt,
loss_fn):
    model.train()     prediction =
model(x)     batch_loss =
loss_fn(prediction, y)
batch_loss.backward()
optimizer.step()
optimizer.zero_grad()
```

```
    return batch_loss.item()
 def accuracy(x, y,
model):
    model.eval()
    # this is the same as @torch.no_grad
    # at the top of function, only difference
# being, grad is not computed in the with scope
with torch.no_grad():
        prediction = model(x)
max_values, argmaxes = prediction.max(-1)
is_correct = argmaxes == y     return
is_correct.cpu().numpy().tolist()
 def
get_data():
    train = FMNISTDataset(tr_images, tr_targets)      trn_dl =
DataLoader(train, batch_size=32, shuffle=True)     val =
FMNISTDataset(val_images, val_targets)      val_dl = DataLoader(val,
batch_size=len(val_images), shuffle=False)     return trn_dl, val_dl

@torch.no_grad() def val_loss(x, y,
model):     prediction = model(x)
val_loss = loss_fn(prediction, y)
return val_loss.item()
 trn_dl, val_dl = get_data() model,
loss_fn, optimizer = get_model()
 train_losses, train_accuracies = [],
[] val_losses, val_accuracies = [],
[] for epoch in range(5):
    print(epoch)
    train_epoch_losses, train_epoch_accuracies = [], []
for ix, batch in enumerate(iter(trn_dl)):
        x, y = batch         batch_loss = train_batch(x, y,
model, optimizer, loss_fn)
train_epoch_losses.append(batch_loss)
    train_epoch_loss = np.array(train_epoch_losses).mean()
     for ix, batch in
enumerate(iter(trn_dl)):
        x, y = batch         is_correct =
accuracy(x, y, model)
train_epoch_accuracies.extend(is_correct)
```

```
    train_epoch_accuracy =
np.mean(train_epoch_accuracies)     for ix, batch in
enumerate(iter(val_dl)):         x, y = batch
val_is_correct = accuracy(x, y, model)
validation_loss = val_loss(x, y, model)
val_epoch_accuracy = np.mean(val_is_correct)
train_losses.append(train_epoch_loss)
train_accuracies.append(train_epoch_accuracy)
val_losses.append(validation_loss)
val_accuracies.append(val_epoch_accuracy)
 epochs = np.arange(5)+1 import
matplotlib.ticker as mtick import
matplotlib.pyplot as plt import
matplotlib.ticker as mticker
%matplotlib inline plt.subplot(211) plt.plot(epochs, train_losses,
'bo', label='Training loss') plt.plot(epochs, val_losses, 'r',
label='Validation loss')
plt.gca().xaxis.set_major_locator(mticker.MultipleLocator(1))
plt.title('Training and validation loss with 0.1 learning rate')
plt.xlabel('Epochs') plt.ylabel('Loss') plt.legend()
plt.grid('off') plt.show() plt.subplot(212) plt.plot(epochs,
train_accuracies, 'bo', label='Training accuracy')
plt.plot(epochs, val_accuracies, 'r', label='Validation accuracy')
plt.gca().xaxis.set_major_locator(mticker.MultipleLocator(1))
plt.title('Training and validation accuracy with 0.1 learning
rate') plt.xlabel('Epochs') plt.ylabel('Accuracy')
plt.gca().set_yticklabels(['{:.0f}%'.format(x*100) for x
in plt.gca().get_yticks()])  plt.legend() plt.grid('off')
plt.show()
```

Kết quả như sau:



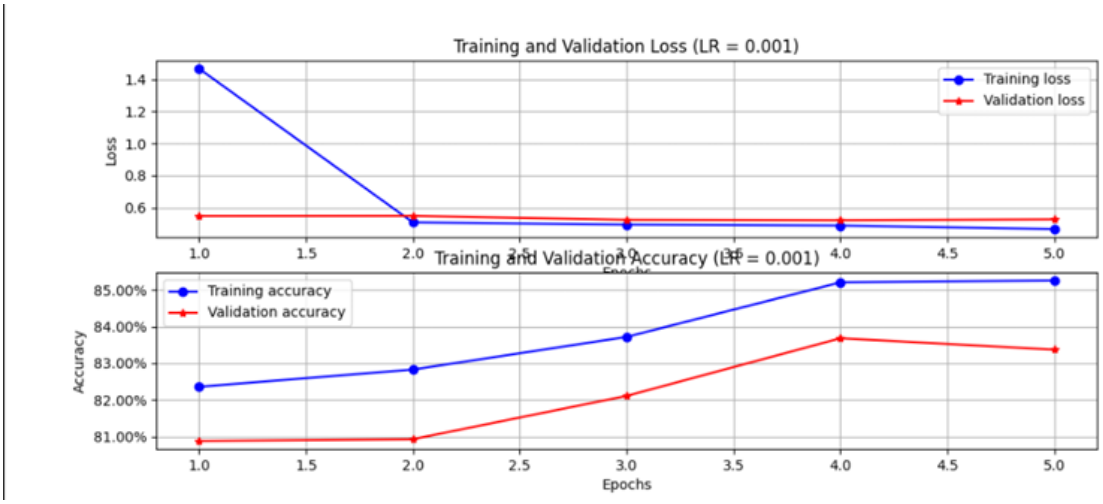
Tốc độ học trung bình:

```
def get_model():
    model = nn.Sequential(
nn.Linear(28 * 28, 1000),
nn.ReLU(),
        nn.Linear(1000, 10)
    ).to(device)

    loss_fn = nn.CrossEntropyLoss()
optimizer = Adam(model.parameters(), lr=1e-3)
return model, loss_fn, optimizer trn_dl, val_dl
= get_data() model, loss_fn, optimizer =
get_model()

train_losses, train_accuracies = [],
[] val_losses, val_accuracies = [],
[] for epoch in range(5):
    print(epoch)     train_epoch_losses,
train_epoch_accuracies = [], []     for ix, batch in
enumerate(iter(trn_dl)):
        x, y = batch
```

```
        batch_loss = train_batch(x, y, model, optimizer,
loss_fn)         train_epoch_losses.append(batch_loss)
train_epoch_loss = np.array(train_epoch_losses).mean()
     for ix, batch in
enumerate(iter(trn_dl)):
        x, y = batch         is_correct = accuracy(x, y,
model)         train_epoch_accuracies.extend(is_correct)
train_epoch_accuracy = np.mean(train_epoch_accuracies)
for ix, batch in enumerate(iter(val_dl)):         x, y =
batch         val_is_correct = accuracy(x, y, model)
validation_loss = val_loss(x, y, model)
val_epoch_accuracy = np.mean(val_is_correct)
train_losses.append(train_epoch_loss)
train_accuracies.append(train_epoch_accuracy)
val_losses.append(validation_loss)
val_accuracies.append(val_epoch_accuracy) epochs =
np.arange(5)+1 import matplotlib.ticker as mtick import
matplotlib.pyplot as plt import matplotlib.ticker as
mticker
%matplotlib inline plt.subplot(211) plt.plot(epochs,
train_losses, 'bo', label='Training loss') plt.plot(epochs,
val_losses, 'r', label='Validation loss')
plt.gca().xaxis.set_major_locator(mticker.MultipleLocator(1))
plt.title('Training and validation loss with 0.001 learning
rate') plt.xlabel('Epochs') plt.ylabel('Loss') plt.legend()
plt.grid('off') plt.show() plt.subplot(212)
plt.plot(epochs, train_accuracies, 'bo', label='Training accuracy')
plt.plot(epochs, val_accuracies, 'r', label='Validation accuracy')
plt.gca().xaxis.set_major_locator(mticker.MultipleLocator(1))
plt.title('Training and validation accuracy with 0.001 learning
rate') plt.xlabel('Epochs') plt.ylabel('Accuracy')
plt.gca().set_yticklabels(['{:.0f}%'.format(x*100) for x in
plt.gca().get_yticks()])  plt.legend() plt.grid('off')
```

```
plt.show()
```

Kết quả như sau:



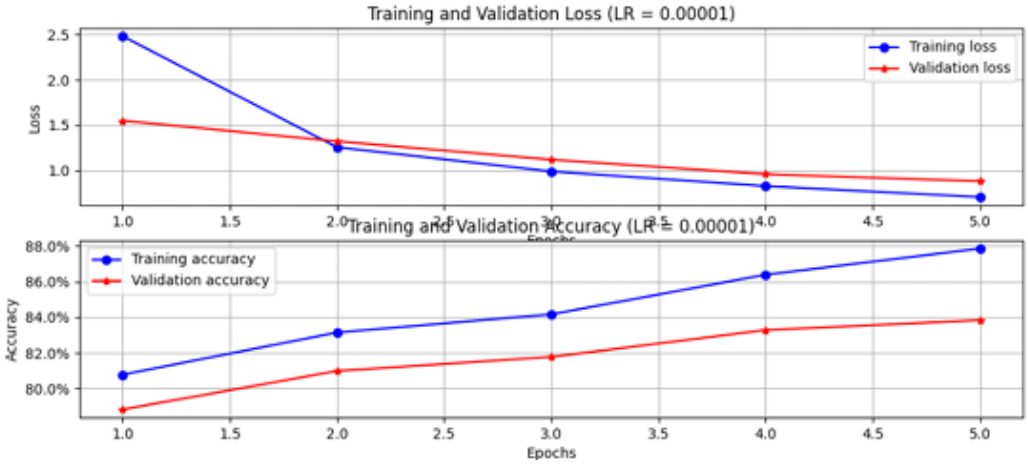
Tốc độ học thấp:

```
def get_model():
    model = nn.Sequential(
nn.Linear(28 * 28, 1000),
nn.ReLU(),
        nn.Linear(1000, 10)
    ).to(device)
     loss_fn = nn.CrossEntropyLoss()
optimizer = Adam(model.parameters(), lr=1e-5)
return model, loss_fn, optimizer  trn_dl,
val_dl = get_data() model, loss_fn, optimizer =
get_model()

train_losses, train_accuracies = [],
[] val_losses, val_accuracies = [],
[] for epoch in range(5):
    print(epoch)
```

```
    train_epoch_losses, train_epoch_accuracies = [], []
for ix, batch in enumerate(iter(trn_dl)):
        x, y = batch         batch_loss = train_batch(x, y,
model, optimizer, loss_fn)
train_epoch_losses.append(batch_loss)      train_epoch_loss =
np.array(train_epoch_losses).mean()
     for ix, batch in
enumerate(iter(trn_dl)):
        x, y = batch         is_correct = accuracy(x, y,
model)         train_epoch_accuracies.extend(is_correct)
train_epoch_accuracy = np.mean(train_epoch_accuracies)
for ix, batch in enumerate(iter(val_dl)):         x, y =
batch         val_is_correct = accuracy(x, y, model)
validation_loss = val_loss(x, y, model)
val_epoch_accuracy = np.mean(val_is_correct)
train_losses.append(train_epoch_loss)
train_accuracies.append(train_epoch_accuracy)
val_losses.append(validation_loss)
val_accuracies.append(val_epoch_accuracy) epochs =
np.arange(5)+1 import matplotlib.ticker as mtick import
matplotlib.pyplot as plt import matplotlib.ticker as
mticker
%matplotlib inline plt.subplot(211) plt.plot(epochs, train_losses,
'bo', label='Training loss') plt.plot(epochs, val_losses, 'r',
label='Validation loss')
plt.gca().xaxis.set_major_locator(mticker.MultipleLocator(1))
plt.title('Training and validation loss with 0.00001 learning
rate') plt.xlabel('Epochs') plt.ylabel('Loss') plt.legend()
plt.grid('off') plt.show() plt.subplot(212)
plt.plot(epochs, train_accuracies, 'bo', label='Training accuracy')
plt.plot(epochs, val_accuracies, 'r', label='Validation accuracy')
plt.gca().xaxis.set_major_locator(mticker.MultipleLocator(1))
plt.title('Training and validation accuracy with 0.00001 learning
rate') plt.xlabel('Epochs') plt.ylabel('Accuracy')
plt.gca().set_yticklabels(['{:.0f}%'.format(x*100) for x in
```

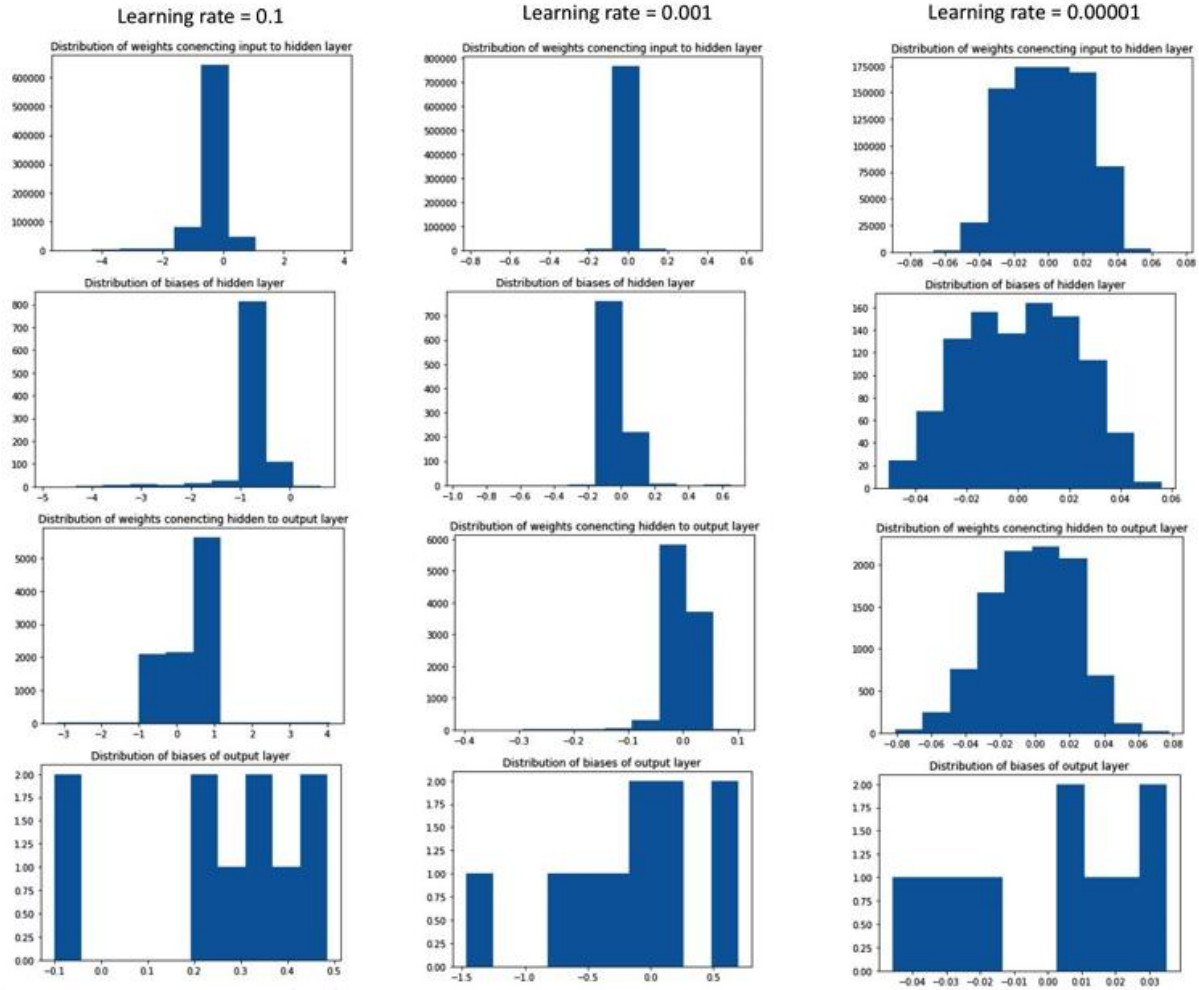
```
plt.gca().get_yticks()])
plt.legend()
plt.grid('off')
plt.show()
```

Kết quả như sau:



Như chúng ta có thể thấy, ngay cả khi tập dữ liệu không được chia tỷ lệ, chúng ta không thể huấn luyện một mô hình chính xác khi tốc độ học là 0,1. Hơn nữa, độ chính xác không cao như phần trước khi tỷ lệ học là 0,001.

Cuối cùng, khi tốc độ học rất nhỏ (0,00001), mô hình có thể học tốt như trong các phần trước, nhưng lần này bị tình trạng overfitting trên dữ liệu huấn luyện. Hãy tìm hiểu lý do tại sao điều này xảy ra bằng cách xem xét phân phối tham số giữa các lớp như sau:



Ở đây, chúng ta có thể thấy rằng khi độ chính xác của mô hình cao (tức là khi tốc độ học là 0,00001), các trọng số có phạm vi nhỏ hơn nhiều (thường nằm trong khoảng từ -0,05 đến 0,05 trong trường hợp này) so với khi tốc độ học cao .

Các trọng số có thể được điều chỉnh về một giá trị nhỏ vì tốc độ học nhỏ. Lưu ý rằng kịch bản trong đó tốc độ học là 0,00001 trên tập dữ liệu không được chia tỷ lệ tương đương với kịch bản tốc độ học là 0,001 trên tập dữ liệu được chia tỷ lệ. Điều này là do các trọng số bây giờ có thể di chuyển về một giá trị rất nhỏ (vì gradient * learning rate là một giá trị rất nhỏ, do tốc độ học là nhỏ).