```
import numpy as np
import pandas as pd
```

```
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)
```

```
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-images-idx3-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-images-idx3-ubyte.gz to /root/data/FMNIST/FashionMNIST/raw/
100%|██████████| 26421880/26421880 [00:01<00:00, 14414812.68it/s]
Extracting /root/data/FMNIST/FashionMNIST/raw/train-images-idx3-ubyte.gz to /root/data/FMNIST/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-labels-idx1-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-labels-idx1-ubyte.gz to /root/data/FMNIST/FashionMNIST/raw/
100%|██████████| 29515/29515 [00:00<00:00, 302003.56it/s]
Extracting /root/data/FMNIST/FashionMNIST/raw/train-labels-idx1-ubyte.gz to /root/data/FMNIST/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-images-idx3-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-images-idx3-ubyte.gz to /root/data/FMNIST/FashionMNIST/raw/t
100%|██████████| 4422102/4422102 [00:00<00:00, 5628820.83it/s]
Extracting /root/data/FMNIST/FashionMNIST/raw/t10k-images-idx3-ubyte.gz to /root/data/FMNIST/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-labels-idx1-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-labels-idx1-ubyte.gz to /root/data/FMNIST/FashionMNIST/raw/t
100%|██████████| 5148/5148 [00:00<00:00, 14462342.26it/s]Extracting /root/data/FMNIST/FashionMNIST/raw/t10k-labels-idx1-ubyte.gz to /roo
```

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

```
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}')
```

```
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']
```

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

```
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()
```

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:

```
# since there's no need for updating weights,
# we might as well not compute the gradients.
# Using this '@' decorator on top of functions
# will disable gradient computation in the entire function
@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()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-28-58042d122d20> in <cell line: 1>()
----> 1 trn_dl = get_data()
      2 model, loss_fn, optimizer = get_model()
      3 losses, accuracies = [], []
      4 for epoch in range(5):
      5   print(epoch)

                          3 frames

/usr/local/lib/python3.10/dist-packages/torch/utils/data/sampler.py in num_samples(self)
    147         # dataset size might change at runtime
    148         if self._num_samples is None:
--> 149             return len(self.data_source)
    150         return self._num_samples
    151

TypeError: object of type 'FMNISTDataset' has no len()
```

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
```

2. 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)

```
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)
```

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

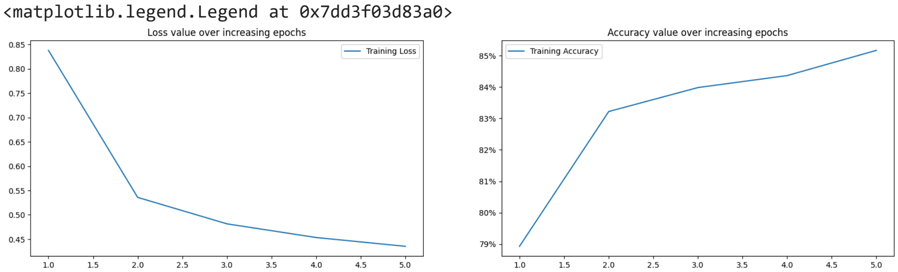
```
def get_data():
  train = FMNISTDataset(tr_images, tr_targets)
  trn_dl = DataLoader(train, batch_size=32, shuffle=True)
  return trn_dl
from torch.optim import SGD
def get_model():
  model = nn.Sequential(nn.Linear(28 * 28, 1000),nn.ReLU(),nn.Linear(1000, 10)).to(device)
  loss_fn = nn.CrossEntropyLoss()
  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)
```

```
0
1
2
3
4
```

4. 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()
```

```
<ipython-input-32-48201efd1fe1>:12: UserWarning: FixedFormatter should only be used toge
  plt.gca().set_yticklabels(['{:.0f}%'.format(x*100) for x in
<matplotlib.legend.Legend at 0x7dd3f03d83a0>
```




**4. Hiểu được tác động của việc thay đổi kích thước bó** Kích thước bó 32

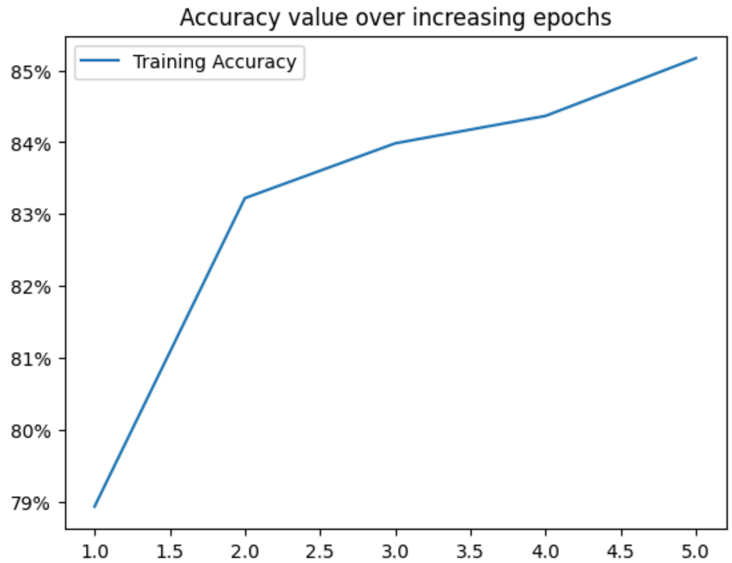
1. Tải xuống và import các hình ảnh huấn luyện và gán nhãn vào biến tr_targets:

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

```
<ipython-input-35-805124f35414>:6: UserWarning: FixedFormatter should only be used together with FixedLocator
  plt.gca().set_yticklabels(['{:.0f}%'.format(x*100) for x in
<matplotlib.legend.Legend at 0x7dd2fd1bace0>
```




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()
```

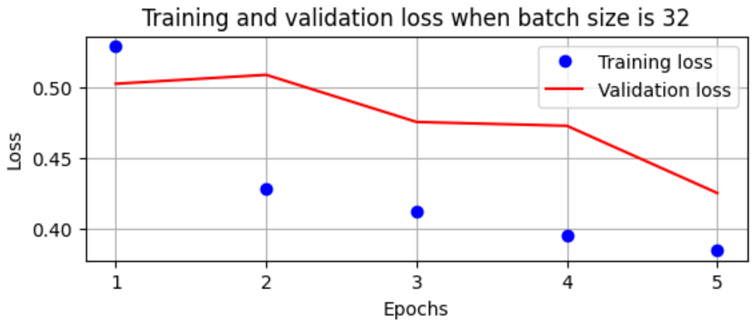
8. Huấn luyện mô hình như sau:

```
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)
```

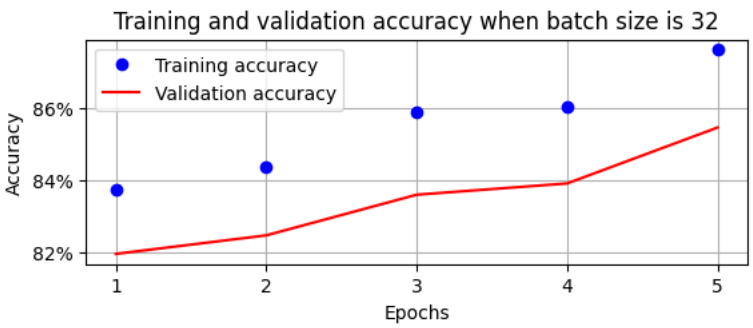
```
0
1
2
3
4
```

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
%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 32')
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 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()
```



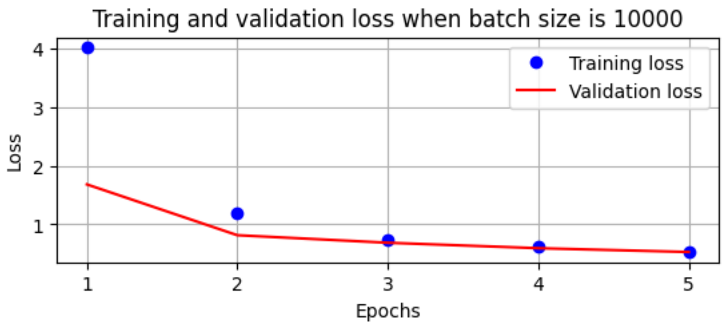

```
<ipython-input-42-f55b9a42e238>:23: UserWarning: FixedFormatter should only be used together with FixedLocator
  plt.gca().set_yticklabels(['{:.0f}%'.format(x*100) for x in
```



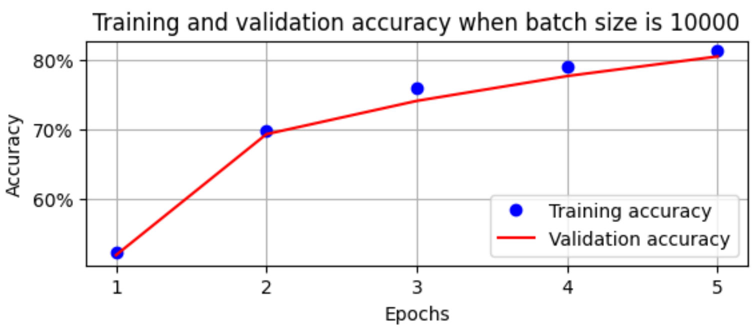

Kích thước bó 10.000:

```
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()
```

```
0
1
2
3
4
```




```
<ipython-input-43-9401dbfe7c55>:57: UserWarning: FixedFormatter should only be used together with FixedLocator
  plt.gca().set_yticklabels(['{:.0f}%'.format(x*100) for x in
```

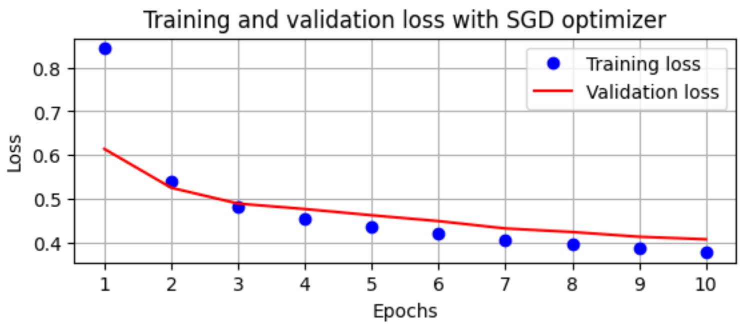



Hiểu được tác động của việc thay đổi trình tối ưu hóa sai số.

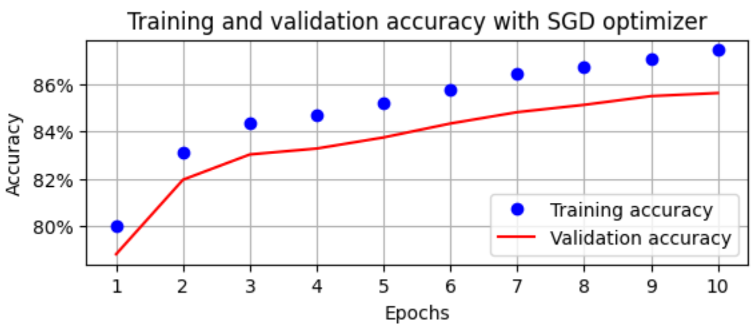
```
from torchvision import datasets
import torch
data_folder = '~/data/FMNIST' # This can be any directory you want to
# download FMNIST to
fmnist = datasets.FashionMNIST(data_folder, download=True, train=True)
tr_images = fmnist.data
tr_targets = fmnist.targets
val_fmnist = datasets.FashionMNIST(data_folder,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

```
0
1
2
3
4
5
6
7
8
9
```



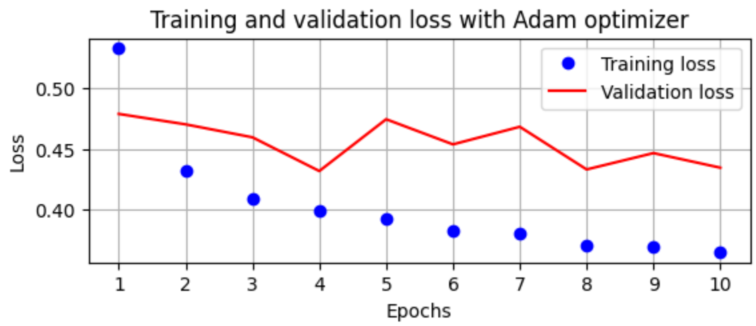

```
<ipython-input-44-d00fd1273de7>:118: UserWarning: FixedFormatter should only be used together with FixedLocator
  plt.gca().set_yticklabels(['{:.0f}%'.format(x*100) for x in
```



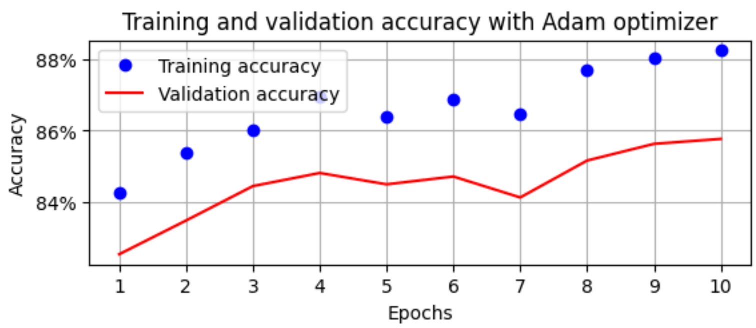

Adam optimizer

```
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()
```

```
0
1
2
3
4
5
6
7
8
9
```




```
<ipython-input-46-ca83dc594806>:57: UserWarning: FixedFormatter should only be used together with FixedLocator
  plt.gca().set_yticklabels(['{:.0f}%'.format(x*100) for x in
```




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

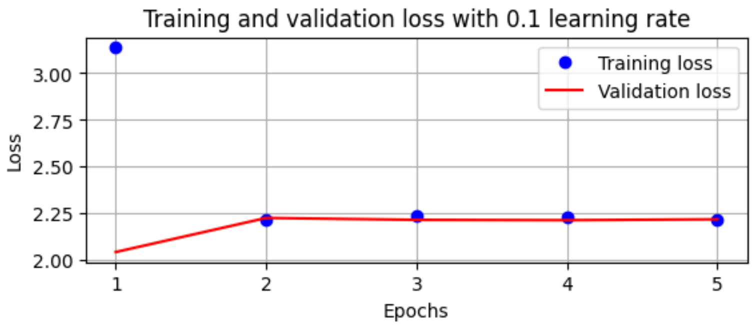
- Mục danh sách
- Mục danh sách

• Tốc độ học trung bình • Tốc độ học thấp

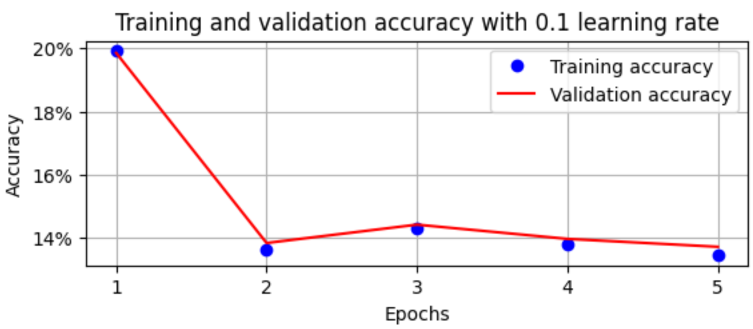
```
from torchvision import datasets
import torch
data_folder = '~/data/FMNIST' # This can be any directory you want to
# download FMNIST to
fmnist = datasets.FashionMNIST(data_folder, download=True, train=True)
tr_images = fmnist.data
tr_targets = fmnist.targets

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()
```

```
0
1
2
3
4
```




```
<ipython-input-47-aba86d40db82>:121: UserWarning: FixedFormatter should only be used together with FixedLocator
  plt.gca().set_yticklabels(['{:.0f}%'.format(x*100) for x in
```




**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()
```

**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()
```