```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision
import torchvision.transforms as transforms
import numpy as np
import matplotlib.pyplot as plt
import torch.optim as optim
torch.set_grad_enabled(True)

from collections import OrderedDict
from collections import namedtuple
from itertools import product

from IPython.display import display, clear_output
import pandas as pd
import time
import json
```

```
def get_num_correct(preds, labels):
  return preds.argmax(dim=1).eq(labels).sum().item()
```

```
class Network(nn.Module):
  def __init__(self):
    super(Network, self).__init__()
    self.conv1 = nn.Conv2d(in_channels=1, out_channels=6, kernel_size=5)
    self.conv2 = nn.Conv2d(in_channels=6, out_channels=12, kernel_size=5)

    self.fc1 = nn.Linear(in_features=12*4*4, out_features=120)
    self.fc2 = nn.Linear(in_features=120, out_features=60)
    self.out = nn.Linear(in_features=60, out_features=10)

  def forward(self, t):
    # (1) Input layer
    t=t

    # (2) Hidden conv layer
    t=self.conv1(t)
    t=F.relu(t)
    t=F.max_pool2d(t, kernel_size=2, stride=2)

    # (3) Hidden conv layer
    t=self.conv2(t)
    t=F.relu(t)
    t=F.max_pool2d(t, kernel_size=2, stride=2)

    # (4) Hidden Linear layer
    t=t.reshape(-1, 12*4*4)
    t=self.fc1(t)
    t=F.relu(t)

    # (5) Hidden Linear layer
    t=self.fc2(t)
```

```
    t=F.relu(t)

    # (6) Output layer
    t=self.out(t)

    return t
```

```
train_set = torchvision.datasets.FashionMNIST(root='./data/FashionMNIST',
                                              train=True,
                                              download=True,
                                              transform=transforms.Compose([transforms.
```

```
# loader = torch.utils.data.DataLoader(train_set, batch_size=100, shuffle=True)
```

```
!pip install tensorflow==2.0.0
```

```
!pip uninstall -q tensorboard tb-nightly
```

```
  Proceed (y/n)? y
  WARNING: Skipping tb-nightly as it is not installed.
```

```
!pip install tensorboard==2.0.0
```

```
%load_ext tensorboard
```

```
from torch.utils.tensorboard import SummaryWriter
```

```
class RunManager():
  def __init__(self):

    self.epoch_count = 0
    self.epoch_loss = 0
    self.epoch_num_correct = 0
    self.epoch_start_time = None

    self.run_params = None
    self.run_count = 0
    self.run_data = []
    self.run_start_time = None

    self.network = None
    self.loader = None
    self.tb = None

  def begin_run(self, run, network, loader):
      self.run_start_time = time.time()
      self.run_params = run
      self.run_count += 1
      self.network = network
      self.loader = loader
      self.tb = SummaryWriter(comment=f'-{run}')
```

```
25
26      images, labels = next(iter(self.loader))
27      grid = torchvision.utils.make_grid(images)
28
29      self.tb.add_image('images', grid)
30      self.tb.add_graph(self.network, images)
31
32  def end_run(self):
33      self.tb.close()
34      self.epoch_count = 0
35
36  def begin_epoch(self):
37      self.epoch_start_time = time.time()
38      self.epoch_count += 1
39      self.epoch_loss = 0
40      self.epoch_num_correct = 0
41
42  def end_epoch(self):
43      epoch_duration = time.time() - self.epoch_start_time
44      run_duration = time.time() - self.run_start_time
45      loss = self.epoch_loss / len(self.loader.dataset)
46      accuracy = self.epoch_num_correct / len(self.loader.dataset)
47      self.tb.add_scalar('Loss', loss, self.epoch_count)
48      self.tb.add_scalar('Accuracy', accuracy, self.epoch_count)
49
50      for name, param in self.network.named_parameters():
51        self.tb.add_histogram(name, param, self.epoch_count)
52        self.tb.add_histogram(f'{name}.grad', param.grad, self.epoch_count)
53
54      results = OrderedDict()
55      results["run"] = self.run_count
56      results["epoch"] = self.epoch_count
57      results['loss'] = loss
58      results["accuracy"] = accuracy
59      results['epoch duration'] = epoch_duration
60      results['run duration'] = run_duration
61      for k,v in self.run_params._asdict().items(): results[k] = v
62      self.run_data.append(results)
63
64      df = pd.DataFrame.from_dict(self.run_data, orient='columns')
65      clear_output(wait=True)
66      display(df)
67
68  def track_loss(self, loss):
69      self.epoch_loss += loss.item() * self.loader.batch_size
70
71  def track_num_correct(self, preds, labels):
72      self.epoch_num_correct += self._get_num_correct(preds, labels)
73
74  @torch.no_grad()
75  def _get_num_correct(self, preds, labels):
76      return preds.argmax(dim=1).eq(labels).sum().item()
77
78  def save(self, fileName):
79      pd.DataFrame.from_dict(
```

```
            self.run_data, orient='columns'
        ).to_csv(f'{fileName}.csv')

        with open(f'{fileName}.json', 'w', encoding='utf-8') as f:
            json.dump(self.run_data, f, ensure_ascii=False, indent=4)
```

```
class Epoch():
    def __init__(self):
        self.count = 0
        self.loss = 0
        self.num_correct = 0
        self.start_time = None
```

```
class RunBuilder():
    @staticmethod
    def get_runs(params):

        Run = namedtuple('Run', params.keys())

        runs = []
        for v in product(*params.values()):
            runs.append(Run(*v))

        return runs
```

```
params = OrderedDict(
    lr = [.01]
    ,batch_size = [1000, 10000]
    # ,num_workers = [0,1,2,4,8,16]
    # ,device = ["cuda", "cpu"]
    # ,shuffle = [True, False]
)
m = RunManager()

for run in RunBuilder.get_runs(params):
  network = Network()
  loader = torch.utils.data.DataLoader(train_set, batch_size=run.batch_size)
  optimizer = optim.Adam(network.parameters(), lr=run.lr)

  m.begin_run(run, network, loader)

  for epoch in range(5):
    m.begin_epoch()
    for batch in loader:

      images, labels = batch
      preds = network(images) # pass the batch to the network
      loss = F.cross_entropy(preds, labels) # Calculate loss
      optimizer.zero_grad() #make gradients zero
      loss.backward() #Calculate gradients
      optimizer.step() #Update weights

      m.track_loss(loss)
      m.track_num_correct(preds, labels)
```

```
30
31     m.end_epoch()
32   m.end_run()
33 m.save('results')
```

	run	epoch	loss	accuracy	epoch duration	run duration	lr	batch_size
0	1	1	0.964658	0.633033	12.759357	13.518127	0.01	1000
1	1	2	0.530516	0.793583	12.695007	26.330733	0.01	1000
2	1	3	0.434502	0.840700	12.844998	39.285787	0.01	1000
3	1	4	0.373670	0.863283	12.654268	52.040040	0.01	1000
4	1	5	0.341758	0.874450	12.797414	64.948318	0.01	1000
5	2	1	2.081127	0.224250	12.669566	19.573488	0.01	10000
6	2	2	1.354049	0.471917	12.897310	32.576568	0.01	10000
7	2	3	1.040429	0.593433	12.696321	45.389613	0.01	10000
8	2	4	0.889843	0.655400	12.755024	58.249456	0.01	10000
9	2	5	0.801569	0.694717	12.656359	71.013590	0.01	10000

```
1 tensorboard --logdir=runs
```

```
Reusing TensorBoard on port 6006 (pid 330), started 0:00:11 ago. (Use '!kill 330' to k
```

TensorBoard SCALARS IMAGES GRAPHS INACTIVE

Show data download links

Ignore outliers in chart scalin

Tooltip sorting method: default

Smoothing 0.6

Horizontal Axis

STEP RELATIVE

WALL

Runs

Write a regex to filter runs

Jan03_21-56-11_2f5062ff8145-Run(lr=0.01, batch_size=1000)

Jan03_21-58-56_2f5062ff8145-Run(lr=0.01, batch_size=1000)

Jan03_22-00-01_2f5062ff8145-Run(lr=0.01, batch_size=10000)

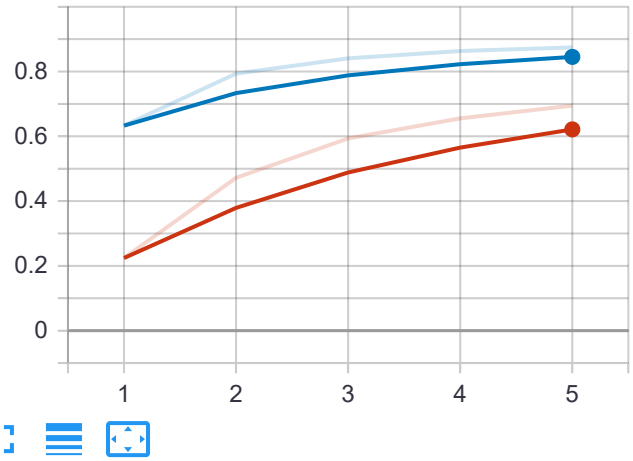
TOGGLE ALL RUNS

runs

Filter tags (regular expressions supported)

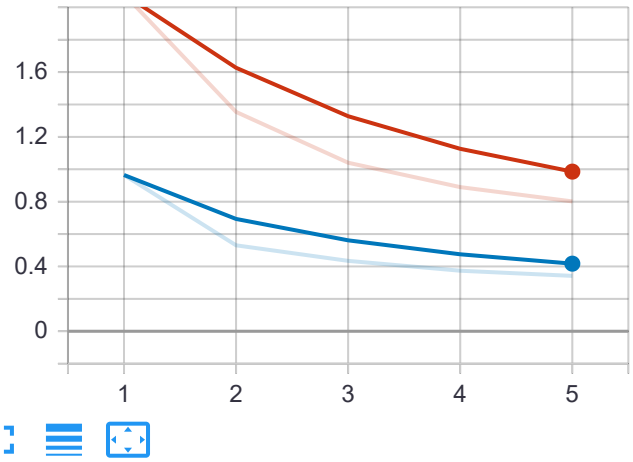
Accuracy

Accuracy



Loss

Loss



1