assignment0

January 14, 2026

1 CS260R Reinforcement Learning Assignment 0: Jupyter Notebook usage and assignment submission workflow

---

CS260R: Reinforcement Learning. Department of Computer Science at University of California, Los Angeles. Course Instructor: Professor Bolei ZHOU. Assignment author: Zhenghao PENG.

You are asked to finish four tasks:

1. Fill in your name and University ID in the next cell.
2. Install pytorch and finish the Kindergarten Pytorch section.
3. Run all cells and export this notebook **as a PDF file**.
4. Compress this folder `assignment0` **as a ZIP file** and submit **the PDF file and the ZIP file separately as two files** in BruinLearn.

[1]:
```
# TODO: Fill your name and UID here
my_name = "Anthony Zhao"
my_student_id = "906173995"
```

[2]:
```
# Run this cell without modification

text = "Oh, I finished this assignment! I am {} ({})".format(my_name,
 ↪my_student_id)
print(text)
with open("{}.txt".format(text), "w") as f:
    f.write(text)
```

```
Oh, I finished this assignment! I am Anthony Zhao (906173995)
```

1.1 Kindergarten Pytorch

1. Please install pytorch in your virtual environment following the instruction: https://pytorch.org/get-started/locally/.

```
pip install torch torchvision
```

2. If you are not familiar with Pytorch, please go through the tutorial in official website until you can understand the quick start tutorial.
3. The following code is copied from the quick start tutorial, please solve all `TODO`s and print the result in the cells before generating the PDF file.

[3]:
```
# You can also run this cell in notebook:

# !pip install torch torchvision
```

1.1.1 Prepare data

[4]:
```
import torch
from torch import nn
from torch.utils.data import DataLoader
from torchvision import datasets
from torchvision.transforms import ToTensor

# Download training data from open datasets.
training_data = datasets.FashionMNIST(
    root="data",
    train=True,
    download=True,
    transform=ToTensor(),
)

# Download test data from open datasets.
test_data = datasets.FashionMNIST(
    root="data",
    train=False,
    download=True,
    transform=ToTensor(),
)

batch_size = 64

# Create data loaders.
train_dataloader = DataLoader(training_data, batch_size=batch_size)
test_dataloader = DataLoader(test_data, batch_size=batch_size)
```

1.1.2 Define model

[5]:
```
# Get cpu, gpu or mps device for training.
device = (
    "cuda"
    if torch.cuda.is_available()
    else "mps"
    if torch.backends.mps.is_available()
    else "cpu"
)
print(f"Using {device} device")

# Define model
```

```
class NeuralNetwork(nn.Module):
    def __init__(self):
        super().__init__()
        self.flatten = nn.Flatten()

        # TODO: Define the self.linear_relu_stack by uncommenting next few lines
        # and understand what they mean
        # Neural net with 2 hidden layers (512 neurons each) and 1 output layer␣
 ↪(10 neurons) and 1 input layer (28*28)
        # there are 10 classes of articles of clothing and each image is␣
 ↪transformed into a 28x28 pixel image
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(28 * 28, 512),
            nn.ReLU(),
            nn.Linear(512, 512),
            nn.ReLU(),
            nn.Linear(512, 10)
        )


    def forward(self, x):
        x = self.flatten(x)
        logits = self.linear_relu_stack(x)
        return logits

model = NeuralNetwork().to(device)
print(model)
```

```
Using mps device
NeuralNetwork(
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (linear_relu_stack): Sequential(
    (0): Linear(in_features=784, out_features=512, bias=True)
    (1): ReLU()
    (2): Linear(in_features=512, out_features=512, bias=True)
    (3): ReLU()
    (4): Linear(in_features=512, out_features=10, bias=True)
  )
)
```

1.1.3 Define training and test pipelines

[6]:
```
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=1e-3)

def train(dataloader, model, loss_fn, optimizer):
    size = len(dataloader.dataset)
```

```
    model.train()
    for batch, (X, y) in enumerate(dataloader):
        X, y = X.to(device), y.to(device)

        # Compute prediction error
        pred = model(X)
        loss = loss_fn(pred, y)

        # Backpropagation

        # TODO: Uncomment next three lines and understand what they mean
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()

        if batch % 100 == 0:
            loss, current = loss.item(), (batch + 1) * len(X)
            print(f"loss: {loss:>7f}  [{current:>5d}/{size:>5d}]")

def test(dataloader, model, loss_fn):
    size = len(dataloader.dataset)
    num_batches = len(dataloader)
    model.eval()
    test_loss, correct = 0, 0
    with torch.no_grad():
        for X, y in dataloader:
            X, y = X.to(device), y.to(device)
            pred = model(X)
            test_loss += loss_fn(pred, y).item()

            # TODO: Uncomment next line and understand what it means
            # "how many predictions are correct in the batch"
            # pred.argmax(1) is the predicted class of the batch
            # y is the actual class of the batch
            # so (pred.argmax(1) == y) is a boolean array of the correct predictions of size batch_size
            # .type(torch.float) converts the boolean array to a float array of 0s and 1s
            # .sum() sums the float array to get the number of correct predictions
            # .item() gets the number of correct predictions as a scalar
            correct += (pred.argmax(1) == y).type(torch.float).sum().item()

    test_loss /= num_batches
    correct /= size
    print(f"Test Error: \n Accuracy: {(100*correct):>0.1f}%, Avg loss: {test_loss:>8f} \n")
```

1.1.4 Run the training and test pipelines

[7]:
```python
epochs = 5
for t in range(epochs):
    print(f"Epoch {t+1}\n-------------------------------")
    train(train_dataloader, model, loss_fn, optimizer)
    test(test_dataloader, model, loss_fn)
print("Done!")
```

```
Epoch 1
-------------------------------
loss: 2.298511  [   64/60000]
loss: 2.290683  [ 6464/60000]
loss: 2.267752  [12864/60000]
loss: 2.267349  [19264/60000]
loss: 2.248700  [25664/60000]
loss: 2.217807  [32064/60000]
loss: 2.228979  [38464/60000]
loss: 2.195093  [44864/60000]
loss: 2.187416  [51264/60000]
loss: 2.164214  [57664/60000]
Test Error:
 Accuracy: 45.4%, Avg loss: 2.150111

Epoch 2
-------------------------------
loss: 2.157988  [   64/60000]
loss: 2.147011  [ 6464/60000]
loss: 2.080500  [12864/60000]
loss: 2.104212  [19264/60000]
loss: 2.049226  [25664/60000]
loss: 1.986271  [32064/60000]
loss: 2.019812  [38464/60000]
loss: 1.935037  [44864/60000]
loss: 1.937029  [51264/60000]
loss: 1.878115  [57664/60000]
Test Error:
 Accuracy: 55.6%, Avg loss: 1.861714

Epoch 3
-------------------------------
loss: 1.893943  [   64/60000]
loss: 1.862058  [ 6464/60000]
loss: 1.733222  [12864/60000]
loss: 1.782529  [19264/60000]
loss: 1.678723  [25664/60000]
loss: 1.628315  [32064/60000]
loss: 1.655413  [38464/60000]
```

```
loss: 1.559976  [44864/60000]
loss: 1.583152  [51264/60000]
loss: 1.491168  [57664/60000]
Test Error:
 Accuracy: 62.5%, Avg loss: 1.499361

Epoch 4
-------------------------------
loss: 1.563573  [   64/60000]
loss: 1.535503  [ 6464/60000]
loss: 1.379463  [12864/60000]
loss: 1.451912  [19264/60000]
loss: 1.345607  [25664/60000]
loss: 1.336782  [32064/60000]
loss: 1.348679  [38464/60000]
loss: 1.283909  [44864/60000]
loss: 1.315861  [51264/60000]
loss: 1.225344  [57664/60000]
Test Error:
 Accuracy: 64.3%, Avg loss: 1.244303

Epoch 5
-------------------------------
loss: 1.317363  [   64/60000]
loss: 1.307872  [ 6464/60000]
loss: 1.137531  [12864/60000]
loss: 1.237312  [19264/60000]
loss: 1.123834  [25664/60000]
loss: 1.142267  [32064/60000]
loss: 1.157967  [38464/60000]
loss: 1.108679  [44864/60000]
loss: 1.147247  [51264/60000]
loss: 1.066365  [57664/60000]
Test Error:
 Accuracy: 65.3%, Avg loss: 1.081602

Done!
```

1.1.5 Save model

[8]:
```
torch.save(model.state_dict(), "model.pth")
print("Saved PyTorch Model State to model.pth")
```

```
Saved PyTorch Model State to model.pth
```

1.1.6 Load model and run the inference

[9]:
```
model = NeuralNetwork().to(device)
model.load_state_dict(torch.load("model.pth"))
```

[9]: <All keys matched successfully>

[10]:
```
classes = [
    "T-shirt/top",
    "Trouser",
    "Pullover",
    "Dress",
    "Coat",
    "Sandal",
    "Shirt",
    "Sneaker",
    "Bag",
    "Ankle boot",
]

model.eval()
x, y = test_data[0][0], test_data[0][1]
with torch.no_grad():
    x = x.to(device)
    pred = model(x)
    predicted, actual = classes[pred[0].argmax(0)], classes[y]
    print(f'Predicted: "{predicted}", Actual: "{actual}"')
```

```
Predicted: "Ankle boot", Actual: "Ankle boot"
```