import and args

```
from argparse import Namespace
import matplotlib.pyplot as plt
import numpy as np
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import pandas as pd

# Arguments
args = Namespace(
    seed=1234,
    dimensions=4,   # 4 features in iris dataset
    num_classes=3,  # 3 classes in iris dataset
    train_size=0.75,
    test_size=0.25,
    num_hidden_units=100,
    learning_rate=1e-2, # decrease from 1e-0 to 1e-2, 1e-3 is also ok
    num_epochs=200,
    data_file="exp4/iris.csv",
    name_dict = {"setosa":0,"versicolor":1,"virginica":2},
)

# Set seed for reproducability
np.random.seed(args.seed)
```

data prepocessing

```
####### data prepocessing #######
df = pd.read_csv(args.data_file, header=0)
def preprocess(df):
    # 删除掉含有空值的行
    df = df.dropna()
    df = df.drop(['Unnamed: 0'],axis=1)
    df["Species"] = df["Species"].map(args.name_dict)
    return df

df = preprocess(df)

# split train and test
mask = np.random.rand(len(df)) < args.train_size
train_df = df[mask]
test_df = df[~mask]
print ("Train size: {0}, test size: {1}".format(len(train_df),
len(test_df)))

X_train = train_df.drop(["Species"], axis=1)
```

```
y_train = train_df["Species"]
X_test = test_df.drop(["Species"], axis=1)
y_test = test_df["Species"]


# convert pd.DataFrame to torch.Tensor
'''
X_train is a torch.Tensor of shape (n_samples, 4)
y_train is a torch.Tensor of shape (n_samples, )
'''

X_train = torch.from_numpy(X_train.values).float()
X_test = torch.from_numpy(X_test.values).float()
y_train = torch.from_numpy(y_train.values).long()
y_test = torch.from_numpy(y_test.values).long()
```

model training

```
####### model training #######

# model definition
class MLP(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim):
        super(MLP, self).__init__()
        self.fc1 = nn.Linear(input_dim, hidden_dim)
        self.fc2 = nn.Linear(hidden_dim, output_dim)

    def forward(self, x_in):
        '''
        modified from the ipynob notebook
        1. implement softmax both in train and test to get the predictions
        2. return both prediction and logits back
        '''
        a_1 = F.relu(self.fc1(x_in))
        a_2 = self.fc2(a_1)
        a_softmaxed = F.softmax(a_2, dim=1)
        y_pred = torch.argmax(a_softmaxed, dim=1)
        y_pred = y_pred.long()
        return y_pred, a_2


model = MLP(input_dim=args.dimensions,
            hidden_dim=args.num_hidden_units,
            output_dim=args.num_classes)
print (model.named_modules)
loss_fn = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=args.learning_rate)


# Accuracy
def get_accuracy(y_pred, y_target):
```

```
    n_correct = torch.eq(y_pred, y_target).sum().item()
    accuracy = n_correct / len(y_pred) * 100
    return accuracy



# Training
for t in range(args.num_epochs):
    # Forward pass
    y_pred, a_2 = model(X_train)

    # Accuracy
    # _, predictions = y_pred.max(dim=1)
    accuracy = get_accuracy(y_pred=y_pred, y_target=y_train)

    # Loss
    loss = loss_fn(a_2, y_train)

    # Verbose
    if t%20==0:
        print ("epoch: {0:02d} | loss: {1:.4f} | acc: {2:.1f}%".format(
            t, loss, accuracy))

    # Zero all gradients
    optimizer.zero_grad()

    # Backward pass
    loss.backward()

    # Update weights
    optimizer.step()


pred_train,_ = model(X_train)
pred_test, _ = model(X_test)

train_acc = get_accuracy(y_pred=pred_train, y_target=y_train)
test_acc = get_accuracy(y_pred=pred_test, y_target=y_test)
print ("train acc: {0:.1f}%, test acc: {1:.1f}%".format(train_acc,
test_acc))
```

results of lr1e-2

```
epoch: 00 | loss: 1.1597 | acc: 0.9%
epoch: 20 | loss: 0.2877 | acc: 95.6%
epoch: 40 | loss: 0.1399 | acc: 97.3%
epoch: 60 | loss: 0.0981 | acc: 98.2%
epoch: 80 | loss: 0.0836 | acc: 98.2%
epoch: 100 | loss: 0.0765 | acc: 97.3%
```

```
epoch: 120 | loss: 0.0720 | acc: 97.3%
epoch: 140 | loss: 0.0688 | acc: 97.3%
epoch: 160 | loss: 0.0663 | acc: 97.3%
epoch: 180 | loss: 0.0643 | acc: 97.3%
train acc: 97.3%, test acc: 100.0%
```

results of lr1e-3

```
epoch: 00 | loss: 1.7032 | acc: 32.7%
epoch: 20 | loss: 0.9624 | acc: 34.5%
epoch: 40 | loss: 0.7098 | acc: 65.5%
epoch: 60 | loss: 0.5738 | acc: 71.7%
epoch: 80 | loss: 0.4967 | acc: 88.5%
epoch: 100 | loss: 0.4441 | acc: 92.9%
epoch: 120 | loss: 0.4025 | acc: 95.6%
epoch: 140 | loss: 0.3663 | acc: 95.6%
epoch: 160 | loss: 0.3334 | acc: 95.6%
epoch: 180 | loss: 0.3025 | acc: 96.5%
train acc: 96.5%, test acc: 100.0%
```