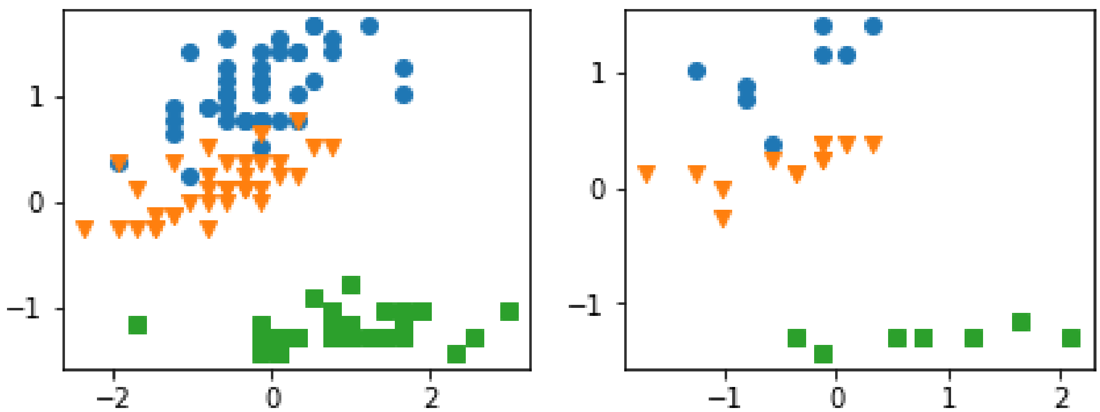
lab-6-softmax-regression

March 4, 2023

[1]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import pandas as pd
import torch
import torch.nn.functional as F
```

[14]:
```
df = pd.read_csv('./data/iris.data', index_col=None, header=None)
df.columns = ['x1', 'x2', 'x3', 'x4', 'y']

d = {'Iris-versicolor': 1,
     'Iris-virginica': 2,
     'Iris-setosa': 0,
}

df['y'] = df['y'].map(d)

# Assign features and target

X = torch.tensor(df[['x2', 'x4']].values, dtype=torch.float)
y = torch.tensor(df['y'].values, dtype=torch.int)

# Shuffling & train/test split

torch.manual_seed(123)
shuffle_idx = torch.randperm(y.size(0), dtype=torch.long)

X, y = X[shuffle_idx], y[shuffle_idx]

percent80 = int(shuffle_idx.size(0)*0.8)

X_train, X_test = X[shuffle_idx[:percent80]], X[shuffle_idx[percent80:]]
y_train, y_test = y[shuffle_idx[:percent80]], y[shuffle_idx[percent80:]]

# Normalize (mean zero, unit variance)

mu, sigma = X_train.mean(dim=0), X_train.std(dim=0)
X_train = (X_train - mu) / sigma
```

```
X_test = (X_test - mu) / sigma


fig, ax = plt.subplots(1, 2, figsize=(7, 2.5))
ax[0].scatter(X_train[y_train == 2, 0], X_train[y_train == 2, 1])
ax[0].scatter(X_train[y_train == 1, 0], X_train[y_train == 1, 1], marker='v')
ax[0].scatter(X_train[y_train == 0, 0], X_train[y_train == 0, 1], marker='s')
ax[1].scatter(X_test[y_test == 2, 0], X_test[y_test == 2, 1])
ax[1].scatter(X_test[y_test == 1, 0], X_test[y_test == 1, 1], marker='v')
ax[1].scatter(X_test[y_test == 0, 0], X_test[y_test == 0, 1], marker='s')
plt.show()
```



High Level Implementation

[6]:
```
DEVICE = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

[23]:
```
class SoftmaxRegression(torch.nn.Module):
    def __init__(self, num_features, num_class):
        super(SoftmaxRegression, self).__init__()
        self.linear = torch.nn.Linear(num_features, num_class, dtype=torch.
 ↪float32, device=DEVICE)

        self.linear.weight.detach().zero_()
        self.linear.weight.detach().zero_()
    def forward(self, x):
        logits = self.linear(x);
        probas = F.softmax(logits, dim=1)
        return logits, probas

    def predict_labels(self, x):
        logits, probas = self.forward(x)
        labels = torch.argmax(probas, dim=1)
```

```
        return labels

    def evaluate(self, x, y):
        labels = self.predict_labels(x).float()
        accuracy = torch.sum(labels.view(-1) == y.float()).item() / y.size(0)
        return accuracy

model = SoftmaxRegression(num_features=2, num_class=3).to(DEVICE)
optimizer = torch.optim.SGD(model.parameters(), lr=0.1)
```

[24]:
```
def comp_accuracy(true_labels, pred_labels):
    accuracy = torch.sum(true_labels.view(-1).float() ==
                         pred_labels.float()).item() / true_labels.size(0)
    return accuracy

X_train = X_train.to(DEVICE)
y_train = y_train.to(DEVICE)
X_test = X_test.to(DEVICE)
y_test = y_test.to(DEVICE)

num_epochs = 50
for epoch in range(num_epochs):


    logits, probas = model(X_train)

    # Compute gradients
    cost = F.cross_entropy(probas,y_train.long())
    optimser.zero_grad()
    cost.backward();

    # Update weights
    optimizer.step()

    logits, probas = model(X_train)
    acc = comp_accuracy(y_train, torch.argmax(probas, dim=1))
    print('Epoch: %03d' % (epoch + 1), end="")
    print(' | Train ACC: %.3f' % acc, end="")
    print(' | Cost: %.3f' % F.cross_entropy(logits, y_train.long()))

print('\nModel parameters:')
print('  Weights: %s' % model.linear.weight)
print('  Bias: %s' % model.linear.bias)
```

```
Epoch: 001 | Train ACC: 0.342 | Cost: 1.199
```

```
Epoch: 002 | Train ACC: 0.342 | Cost: 1.167
Epoch: 003 | Train ACC: 0.342 | Cost: 1.122
Epoch: 004 | Train ACC: 0.342 | Cost: 1.065
Epoch: 005 | Train ACC: 0.342 | Cost: 1.000
Epoch: 006 | Train ACC: 0.358 | Cost: 0.929
Epoch: 007 | Train ACC: 0.542 | Cost: 0.856
Epoch: 008 | Train ACC: 0.658 | Cost: 0.784
Epoch: 009 | Train ACC: 0.708 | Cost: 0.716
Epoch: 010 | Train ACC: 0.767 | Cost: 0.653
Epoch: 011 | Train ACC: 0.767 | Cost: 0.597
Epoch: 012 | Train ACC: 0.783 | Cost: 0.549
Epoch: 013 | Train ACC: 0.783 | Cost: 0.507
Epoch: 014 | Train ACC: 0.792 | Cost: 0.473
Epoch: 015 | Train ACC: 0.783 | Cost: 0.444
Epoch: 016 | Train ACC: 0.783 | Cost: 0.421
Epoch: 017 | Train ACC: 0.783 | Cost: 0.403
Epoch: 018 | Train ACC: 0.792 | Cost: 0.388
Epoch: 019 | Train ACC: 0.800 | Cost: 0.376
Epoch: 020 | Train ACC: 0.792 | Cost: 0.367
Epoch: 021 | Train ACC: 0.800 | Cost: 0.360
Epoch: 022 | Train ACC: 0.800 | Cost: 0.355
Epoch: 023 | Train ACC: 0.800 | Cost: 0.351
Epoch: 024 | Train ACC: 0.825 | Cost: 0.348
Epoch: 025 | Train ACC: 0.833 | Cost: 0.346
Epoch: 026 | Train ACC: 0.833 | Cost: 0.344
Epoch: 027 | Train ACC: 0.833 | Cost: 0.344
Epoch: 028 | Train ACC: 0.833 | Cost: 0.343
Epoch: 029 | Train ACC: 0.842 | Cost: 0.344
Epoch: 030 | Train ACC: 0.867 | Cost: 0.345
Epoch: 031 | Train ACC: 0.858 | Cost: 0.346
Epoch: 032 | Train ACC: 0.858 | Cost: 0.347
Epoch: 033 | Train ACC: 0.858 | Cost: 0.349
Epoch: 034 | Train ACC: 0.858 | Cost: 0.352
Epoch: 035 | Train ACC: 0.858 | Cost: 0.355
Epoch: 036 | Train ACC: 0.858 | Cost: 0.358
Epoch: 037 | Train ACC: 0.883 | Cost: 0.361
Epoch: 038 | Train ACC: 0.883 | Cost: 0.365
Epoch: 039 | Train ACC: 0.883 | Cost: 0.369
Epoch: 040 | Train ACC: 0.883 | Cost: 0.373
Epoch: 041 | Train ACC: 0.883 | Cost: 0.378
Epoch: 042 | Train ACC: 0.883 | Cost: 0.383
Epoch: 043 | Train ACC: 0.883 | Cost: 0.388
Epoch: 044 | Train ACC: 0.883 | Cost: 0.394
Epoch: 045 | Train ACC: 0.875 | Cost: 0.400
Epoch: 046 | Train ACC: 0.883 | Cost: 0.406
Epoch: 047 | Train ACC: 0.883 | Cost: 0.412
Epoch: 048 | Train ACC: 0.883 | Cost: 0.419
Epoch: 049 | Train ACC: 0.883 | Cost: 0.425
```

```
Epoch: 050 | Train ACC: 0.883 | Cost: 0.432

Model parameters:
  Weights: Parameter containing:
tensor([[ 4.0129, -7.3231],
        [-4.1433,  0.0976],
        [ 0.1304,  7.2254]], requires_grad=True)
  Bias: Parameter containing:
tensor([-2.0453,  1.4199, -0.1037], requires_grad=True)
```

[25]:
```
X_test = X_test.to(DEVICE)
y_test = y_test.to(DEVICE)

test_acc = model.evaluate(X_test, y_test)
print('Test set accuracy: %.2f%%' % (test_acc*100))
```

```
Test set accuracy: 83.33%
```

0.1 Softmax Regression on MNIST

[61]:
```
from torchvision import datasets
from torchvision import transforms
from torch.utils.data import DataLoader
import time
```

[62]:
```
# Device
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

# Hyperparameters
random_seed = 123
learning_rate = 0.1
num_epochs = 25
batch_size = 256

# Architecture
num_features = 784
num_classes = 10


train_dataset = datasets.MNIST(root='data',
                               train=True,
                               transform=transforms.ToTensor(),
                               download=True)

test_dataset = datasets.MNIST(root='data',
                              train=False,
                              transform=transforms.ToTensor())
```

```python
train_loader = DataLoader(dataset=train_dataset,
                          batch_size=batch_size,
                          shuffle=True)

test_loader = DataLoader(dataset=test_dataset,
                         batch_size=batch_size,
                         shuffle=False)


# Checking the dataset
for images, labels in train_loader:
    print('Image batch dimensions:', images.shape) #NCHW
    print('Image label dimensions:', labels.shape)
    break
```

```
Image batch dimensions: torch.Size([256, 1, 28, 28])
Image label dimensions: torch.Size([256])
```

[63]:
```python
labels[:10]
```

[63]:
```
tensor([0, 8, 8, 4, 3, 7, 6, 3, 2, 5])
```

[64]:
```python
class SoftmaxRegression(torch.nn.Module):

    def __init__(self, num_features, num_classes):
        super(SoftmaxRegression, self).__init__()
        self.linear = torch.nn.Linear(num_features, num_classes)

        self.linear.weight.detach().zero_()
        self.linear.bias.detach().zero_()

    def forward(self, x):
        logits = self.linear(x)
        probas = F.softmax(logits, dim=1)
        return logits, probas

model = SoftmaxRegression(num_features=num_features,
                          num_classes=num_classes)

model.to(device)
```

[64]:
```
SoftmaxRegression(
  (linear): Linear(in_features=784, out_features=10, bias=True)
)
```

[65]:
```
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
```

[66]:
```
# Manual seed for deterministic data loader
torch.manual_seed(random_seed)


def compute_accuracy(model, data_loader):
    correct_pred, num_examples = 0, 0

    for features, targets in data_loader:
        features = features.view(-1, 28*28).to(device)
        targets = targets.to(device)
        logits, probas = model(features)
        _, predicted_labels = torch.max(probas, 1)
        num_examples += targets.size(0)
        correct_pred += (predicted_labels == targets).sum()

    return correct_pred.float() / num_examples * 100
```

[67]:
```
for f in train_loader:
    print(f[0].shape)
    print(f[1].shape)
    break;
```

```
torch.Size([256, 1, 28, 28])
torch.Size([256])
```

[68]:
```
start_time = time.time()
epoch_costs = []
for epoch in range(num_epochs):
    avg_cost = 0.
    for batch_idx, (features, targets) in enumerate(train_loader):
        features = features.view(-1, 28*28).to(device)
        targets = targets.to(device)

        # forward propagation
        logits, probas = model(features)

        # calculate loss
        cost = F.cross_entropy(logits, targets)
        # calculate gradient
        optimizer.zero_grad()
        cost.backward()
        avg_cost += cost

        # update model params
        optimizer.step()
```

```
        # log
        if not batch_idx % 50:
            print ('Epoch: %03d/%03d | Batch %03d/%03d | Cost: %.4f'
                   %(epoch+1, num_epochs, batch_idx,
                     len(train_dataset)//batch_size, cost))

    with torch.set_grad_enabled(False):
        avg_cost = avg_cost/len(train_dataset)
        epoch_costs.append(avg_cost)
        print('Epoch: %03d/%03d training accuracy: %.2f%%' % (epoch+1,␣
↪num_epochs, compute_accuracy(model, train_loader)))
        print('Time elapsed: %.2f min' % ((time.time() - start_time)/60))
```

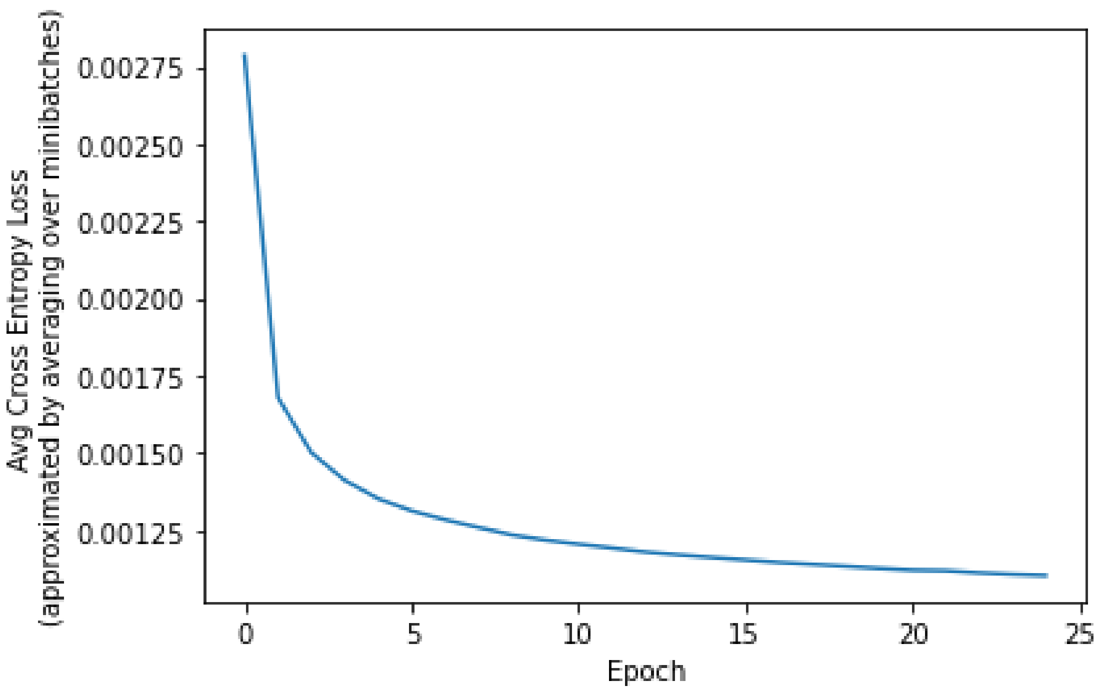
```
Epoch: 001/025 | Batch 000/234 | Cost: 2.3026
Epoch: 001/025 | Batch 050/234 | Cost: 0.7769
Epoch: 001/025 | Batch 100/234 | Cost: 0.6825
Epoch: 001/025 | Batch 150/234 | Cost: 0.5251
Epoch: 001/025 | Batch 200/234 | Cost: 0.5382
Epoch: 001/025 training accuracy: 87.92%
Time elapsed: 0.12 min
Epoch: 002/025 | Batch 000/234 | Cost: 0.5015
Epoch: 002/025 | Batch 050/234 | Cost: 0.4031
Epoch: 002/025 | Batch 100/234 | Cost: 0.4447
Epoch: 002/025 | Batch 150/234 | Cost: 0.4757
Epoch: 002/025 | Batch 200/234 | Cost: 0.4474
Epoch: 002/025 training accuracy: 89.28%
Time elapsed: 0.23 min
Epoch: 003/025 | Batch 000/234 | Cost: 0.4091
Epoch: 003/025 | Batch 050/234 | Cost: 0.4192
Epoch: 003/025 | Batch 100/234 | Cost: 0.2462
Epoch: 003/025 | Batch 150/234 | Cost: 0.4043
Epoch: 003/025 | Batch 200/234 | Cost: 0.3774
Epoch: 003/025 training accuracy: 89.82%
Time elapsed: 0.35 min
Epoch: 004/025 | Batch 000/234 | Cost: 0.3269
Epoch: 004/025 | Batch 050/234 | Cost: 0.3283
Epoch: 004/025 | Batch 100/234 | Cost: 0.3350
Epoch: 004/025 | Batch 150/234 | Cost: 0.3011
Epoch: 004/025 | Batch 200/234 | Cost: 0.4058
Epoch: 004/025 training accuracy: 90.35%
Time elapsed: 0.46 min
Epoch: 005/025 | Batch 000/234 | Cost: 0.3713
Epoch: 005/025 | Batch 050/234 | Cost: 0.3954
Epoch: 005/025 | Batch 100/234 | Cost: 0.3983
```

```
Epoch: 005/025 | Batch 150/234 | Cost: 0.2824
Epoch: 005/025 | Batch 200/234 | Cost: 0.3520
Epoch: 005/025 training accuracy: 90.62%
Time elapsed: 0.59 min
Epoch: 006/025 | Batch 000/234 | Cost: 0.2830
Epoch: 006/025 | Batch 050/234 | Cost: 0.3104
Epoch: 006/025 | Batch 100/234 | Cost: 0.3334
Epoch: 006/025 | Batch 150/234 | Cost: 0.3317
Epoch: 006/025 | Batch 200/234 | Cost: 0.3608
Epoch: 006/025 training accuracy: 90.82%
Time elapsed: 0.71 min
Epoch: 007/025 | Batch 000/234 | Cost: 0.3388
Epoch: 007/025 | Batch 050/234 | Cost: 0.3077
Epoch: 007/025 | Batch 100/234 | Cost: 0.3147
Epoch: 007/025 | Batch 150/234 | Cost: 0.2924
Epoch: 007/025 | Batch 200/234 | Cost: 0.3148
Epoch: 007/025 training accuracy: 91.03%
Time elapsed: 0.84 min
Epoch: 008/025 | Batch 000/234 | Cost: 0.3366
Epoch: 008/025 | Batch 050/234 | Cost: 0.3519
Epoch: 008/025 | Batch 100/234 | Cost: 0.3838
Epoch: 008/025 | Batch 150/234 | Cost: 0.2811
Epoch: 008/025 | Batch 200/234 | Cost: 0.3233
Epoch: 008/025 training accuracy: 91.17%
Time elapsed: 0.97 min
Epoch: 009/025 | Batch 000/234 | Cost: 0.3393
Epoch: 009/025 | Batch 050/234 | Cost: 0.3493
Epoch: 009/025 | Batch 100/234 | Cost: 0.3406
Epoch: 009/025 | Batch 150/234 | Cost: 0.3271
Epoch: 009/025 | Batch 200/234 | Cost: 0.2886
Epoch: 009/025 training accuracy: 91.29%
Time elapsed: 1.10 min
Epoch: 010/025 | Batch 000/234 | Cost: 0.2683
Epoch: 010/025 | Batch 050/234 | Cost: 0.2920
Epoch: 010/025 | Batch 100/234 | Cost: 0.2856
Epoch: 010/025 | Batch 150/234 | Cost: 0.3030
Epoch: 010/025 | Batch 200/234 | Cost: 0.3434
Epoch: 010/025 training accuracy: 91.42%
Time elapsed: 1.23 min
Epoch: 011/025 | Batch 000/234 | Cost: 0.3203
Epoch: 011/025 | Batch 050/234 | Cost: 0.2722
Epoch: 011/025 | Batch 100/234 | Cost: 0.3426
Epoch: 011/025 | Batch 150/234 | Cost: 0.3474
Epoch: 011/025 | Batch 200/234 | Cost: 0.2716
Epoch: 011/025 training accuracy: 91.56%
Time elapsed: 1.37 min
Epoch: 012/025 | Batch 000/234 | Cost: 0.3234
Epoch: 012/025 | Batch 050/234 | Cost: 0.3058
```

```
Epoch: 012/025 | Batch 100/234 | Cost: 0.3143
Epoch: 012/025 | Batch 150/234 | Cost: 0.2685
Epoch: 012/025 | Batch 200/234 | Cost: 0.2487
Epoch: 012/025 training accuracy: 91.64%
Time elapsed: 1.50 min
Epoch: 013/025 | Batch 000/234 | Cost: 0.3649
Epoch: 013/025 | Batch 050/234 | Cost: 0.2811
Epoch: 013/025 | Batch 100/234 | Cost: 0.2619
Epoch: 013/025 | Batch 150/234 | Cost: 0.3158
Epoch: 013/025 | Batch 200/234 | Cost: 0.2522
Epoch: 013/025 training accuracy: 91.68%
Time elapsed: 1.64 min
Epoch: 014/025 | Batch 000/234 | Cost: 0.3492
Epoch: 014/025 | Batch 050/234 | Cost: 0.2866
Epoch: 014/025 | Batch 100/234 | Cost: 0.2746
Epoch: 014/025 | Batch 150/234 | Cost: 0.3830
Epoch: 014/025 | Batch 200/234 | Cost: 0.3988
Epoch: 014/025 training accuracy: 91.73%
Time elapsed: 1.77 min
Epoch: 015/025 | Batch 000/234 | Cost: 0.1983
Epoch: 015/025 | Batch 050/234 | Cost: 0.3476
Epoch: 015/025 | Batch 100/234 | Cost: 0.2963
Epoch: 015/025 | Batch 150/234 | Cost: 0.2509
Epoch: 015/025 | Batch 200/234 | Cost: 0.3132
Epoch: 015/025 training accuracy: 91.83%
Time elapsed: 1.91 min
Epoch: 016/025 | Batch 000/234 | Cost: 0.2453
Epoch: 016/025 | Batch 050/234 | Cost: 0.3091
Epoch: 016/025 | Batch 100/234 | Cost: 0.2818
Epoch: 016/025 | Batch 150/234 | Cost: 0.2680
Epoch: 016/025 | Batch 200/234 | Cost: 0.2571
Epoch: 016/025 training accuracy: 91.88%
Time elapsed: 10.15 min
Epoch: 017/025 | Batch 000/234 | Cost: 0.2611
Epoch: 017/025 | Batch 050/234 | Cost: 0.3563
Epoch: 017/025 | Batch 100/234 | Cost: 0.2167
Epoch: 017/025 | Batch 150/234 | Cost: 0.3099
Epoch: 017/025 | Batch 200/234 | Cost: 0.3305
Epoch: 017/025 training accuracy: 91.86%
Time elapsed: 10.27 min
Epoch: 018/025 | Batch 000/234 | Cost: 0.2718
Epoch: 018/025 | Batch 050/234 | Cost: 0.2744
Epoch: 018/025 | Batch 100/234 | Cost: 0.3043
Epoch: 018/025 | Batch 150/234 | Cost: 0.2605
Epoch: 018/025 | Batch 200/234 | Cost: 0.2321
Epoch: 018/025 training accuracy: 91.97%
Time elapsed: 10.39 min
Epoch: 019/025 | Batch 000/234 | Cost: 0.3145
```

```
Epoch: 019/025 | Batch 050/234 | Cost: 0.3303
Epoch: 019/025 | Batch 100/234 | Cost: 0.2645
Epoch: 019/025 | Batch 150/234 | Cost: 0.3165
Epoch: 019/025 | Batch 200/234 | Cost: 0.2818
Epoch: 019/025 training accuracy: 91.98%
Time elapsed: 10.50 min
Epoch: 020/025 | Batch 000/234 | Cost: 0.3309
Epoch: 020/025 | Batch 050/234 | Cost: 0.2844
Epoch: 020/025 | Batch 100/234 | Cost: 0.1714
Epoch: 020/025 | Batch 150/234 | Cost: 0.3025
Epoch: 020/025 | Batch 200/234 | Cost: 0.2996
Epoch: 020/025 training accuracy: 91.97%
Time elapsed: 10.62 min
Epoch: 021/025 | Batch 000/234 | Cost: 0.3753
Epoch: 021/025 | Batch 050/234 | Cost: 0.2494
Epoch: 021/025 | Batch 100/234 | Cost: 0.2973
Epoch: 021/025 | Batch 150/234 | Cost: 0.2705
Epoch: 021/025 | Batch 200/234 | Cost: 0.2683
Epoch: 021/025 training accuracy: 92.03%
Time elapsed: 10.74 min
Epoch: 022/025 | Batch 000/234 | Cost: 0.2227
Epoch: 022/025 | Batch 050/234 | Cost: 0.2263
Epoch: 022/025 | Batch 100/234 | Cost: 0.2010
Epoch: 022/025 | Batch 150/234 | Cost: 0.3033
Epoch: 022/025 | Batch 200/234 | Cost: 0.3440
Epoch: 022/025 training accuracy: 92.13%
Time elapsed: 10.85 min
Epoch: 023/025 | Batch 000/234 | Cost: 0.2959
Epoch: 023/025 | Batch 050/234 | Cost: 0.2255
Epoch: 023/025 | Batch 100/234 | Cost: 0.3101
Epoch: 023/025 | Batch 150/234 | Cost: 0.2969
Epoch: 023/025 | Batch 200/234 | Cost: 0.3026
Epoch: 023/025 training accuracy: 92.17%
Time elapsed: 10.97 min
Epoch: 024/025 | Batch 000/234 | Cost: 0.3101
Epoch: 024/025 | Batch 050/234 | Cost: 0.2736
Epoch: 024/025 | Batch 100/234 | Cost: 0.2523
Epoch: 024/025 | Batch 150/234 | Cost: 0.3125
Epoch: 024/025 | Batch 200/234 | Cost: 0.2340
Epoch: 024/025 training accuracy: 92.12%
Time elapsed: 11.08 min
Epoch: 025/025 | Batch 000/234 | Cost: 0.3074
Epoch: 025/025 | Batch 050/234 | Cost: 0.2794
Epoch: 025/025 | Batch 100/234 | Cost: 0.2232
Epoch: 025/025 | Batch 150/234 | Cost: 0.3031
Epoch: 025/025 | Batch 200/234 | Cost: 0.2244
Epoch: 025/025 training accuracy: 92.26%
Time elapsed: 11.20 min
```

[69]:
```
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt


plt.plot(epoch_costs)
plt.ylabel('Avg Cross Entropy Loss\n(approximated by averaging over␣
 ↪minibatches)')
plt.xlabel('Epoch')
plt.show()
```



[70]:
```
print('Test accuracy: %.2f%%' % (compute_accuracy(model, test_loader)))
```

```
Test accuracy: 92.23%
```

[ ]: