Mock Coding Questions - 2024/25

January 21, 2025

1 Small coding questions related to exercises

1. *(CNN)* In the following code for defining a convolutional neural network that does a classification between 10 classes

```
model = nn.Sequential(
    nn.Conv2d(3, 6, 5),
    nn.ReLU(),
    nn.MaxPool2d(2, 2),
    nn.Conv2d(6, 16, 5),
    nn.ReLU(),
    nn.MaxPool2d(2, 2),
    nn.Flatten(),
    nn.Linear(16 * 5 * 5, 120),
    nn.ReLU(),
    nn.Linear(120, 84),
    nn.ReLU(),
    nn.Linear(84, 10)
)
```

   (a) What does the `nn.Flatten()` operation do, and why do you need it there?

   (b) Write a line of code for an input X, that is a batch of samples in `(17,28,28)` dimensions, returns the class predicted by the model, i.e. a tensor with the shape `(17,)`. Assume that you are using the negative cross entropy as a loss.

2 Two-Hidden-Layer Neural Networks

Below you find some code snippets related to a two-hidden-layer neural network using PyTorch for classifying MNIST images. The architecture and training loop are partly shown below:

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms

# 1) Define transformations for training and validation sets
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

# 2) Download and load the training and validation data
trainset = torchvision.datasets.MNIST(
    root='./data', train=True, download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(
    trainset, batch_size=64, shuffle=True)

validationset = torchvision.datasets.MNIST(
    root='./data', train=False, download=True, transform=transform)
validationloader = torch.utils.data.DataLoader(
    validationset, batch_size=64, shuffle=False)

# 3) Define a two-hidden-layer neural network
class TwoHiddenLayerNN(nn.Module):
    def __init__(self):
        super(TwoHiddenLayerNN, self).__init__()
        self.fc1 = nn.Linear(28 * 28, 256)
        self.fc2 = nn.Linear(256, 128)
        self.fc3 = nn.Linear(128, 10)
        self.relu = nn.ReLU()

    def forward(self, x):
        # Flatten image to vector of size 784
        x = x.view(-1, 28 * 28)
        x = self.relu(self.fc1(x))
        x = self.relu(self.fc2(x))
        x = self.fc3(x)
        return x

net = TwoHiddenLayerNN()

# 4) Set up loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(net.parameters(), lr=0.001)

# 5) Training loop
epochs = 5
for epoch in range(epochs):
    net.train()
    running_loss = 0.0
    for images, labels in trainloader:
        # (A)
        outputs = net(images)
        # (B)
        loss = criterion(outputs, labels)
        # (C)
        loss.backward()
        # (D)
        optimizer.step()

        running_loss += loss.item()

    # Validation
    net.eval()
    val_loss = 0.0
```

```
    correct_val = 0
    total_val = 0
    with torch.no_grad():
        for images_val, labels_val in validationloader:
            outputs_val = net(images_val)
            loss_val = criterion(outputs_val, labels_val)
            val_loss += loss_val.item()

            # Compute validation accuracy
            _, predicted = torch.max(outputs_val.data, 1)
            total_val += labels_val.size(0)
            correct_val += (predicted == labels_val).sum().item()

    print(f"Epoch [{epoch+1}/{epochs}], "
          f"Train Loss: {running_loss/len(trainloader):.4f}, "
          f"Val Loss: {val_loss/len(validationloader):.4f}, "
          f"Val Accuracy: {(100*correct_val/total_val):.2f}%")
```

Answer the following questions:

1. Which single line must be added (or re-inserted) in the training loop (from line 47 to 62) to ensure that gradients do *not* accumulate from one batch to the next? Where in (A), (B), (C), (D)? Note that more than one answer is possible among the 4.

2. Suppose the shape of `images` entering the `net` is `[64, 1, 28, 28]`. After the line 35:

   ```
   x = x.view(-1, 28 * 28)
   ```

   What is the resulting shape of `x` that is fed into `self.fc1`?

3. If we mistakenly remove `shuffle=True` when building `trainloader` in line 17 of the code above, how might this impact the network's training performance? Provide a brief reasoning.

4. In the validation loop, a student tries to compute the **average** validation loss by writing:

   ```
   val_loss = float(val_loss) / len(validationloader.dataset)
   ```

   Instead of dividing by `len(validationloader)`, they divide by `len(validationloader.dataset)`. Explain why this is likely *incorrect* and provide a corrected line.

3 Polynomial Regression

Consider the following code performing a polynomial regression on random generated data.

```
import numpy as np
import sklearn.linear_model as linear_model
import sklearn.preprocessing as preprocessing

# This function generates data from a polynome, possibly with noise
def generate_polynome_data(polynome, noise_std, n, bound):
    rs = np.random.RandomState(57)
    degree = len(polynome) - 1
    samples = rs.uniform(-bound, bound, size=n)
    xs = np.copy(samples)
    samples = np.ones((n, degree + 1)).T * samples
    powers = np.power(samples.T, np.arange(0, degree + 1, 1))
    return xs.reshape(-1, 1), np.sum(powers * polynome, axis=1) + rs.normal(
        0, noise_std, size=n
    )

# Create some train and test data from a given ground truth polynome
polynom = np.array([0, 0, 1, 0, 0.3])  # function y = x^2 + 0.3 * x^4

n_train = 8
train_bound = 2
noise_std = 0.1
X_train, y_train = generate_polynome_data(polynom, noise_std, n_train, train_bound)

n_test = 25
test_bound = 2
X_test, y_test = generate_polynome_data(polynom, 0.0, n_test, test_bound)

# fit a model for every even polynomial degree between 0 and 6
degrees = np.arange(0, 8, 2)

def PolyFit(degrees, X_train, y_train, X_test, y_test, alpha):
    for d in degrees:
        # add more features to the data input, up until degree d
        poly = preprocessing.PolynomialFeatures(d)
        X_train_features = poly.fit_transform(X_train)
        X_test_features = poly.fit_transform(X_test)

        # fit with a polynomial regression
        lr = linear_model.Ridge(alpha=alpha, fit_intercept=False)
        lr.fit(X_train_features, y_train)

        # Predict the targets for train and test data
        y_train_pred = lr.predict(X_train_features)
        y_test_pred = lr.predict(X_test_features)

        # Show the MSE
        print(
            f"Degree {d} : training error is {np.linalg.norm(y_train - y_train_pred)}, test error is
        )

# Fit
PolyFit(degrees, X_train, y_train, X_test, y_test, 0.0)

# Output
# Degree 0 : training error is 5.501160576266574, test error is 13.026269448141656
# Degree 2 : training error is 0.704069109440556, test error is 2.0997676486970076
# Degree 4 : training error is 0.2860769571840801, test error is 0.6498095945008711
# Degree 6 : training error is 0.17994948328710797, test error is 5.825662882158658
```

Answer the following questions.

1. What is the shape of `X_train_features`?

2. What is the role of the regularization and why is it set to zero in this case?

3. Which is the optimal model and why?

Now consider this variation of the code which differentiates from the previous code from the line `degrees = np.arange(0, 8, 2)` in the following way:

```
# fit a model for every even polynomial degree between 10 and 14
degrees = np.arange(10, 16, 2)

def PolyFit(degrees, X_train, y_train, X_test, y_test, alpha):
    for d in degrees:
        # add more features to the data input, up until degree d
        poly = preprocessing.PolynomialFeatures(d)
        X_train_features = poly.fit_transform(X_train)
        X_test_features = poly.fit_transform(X_test)

        # do not use a regularizer for now, but only fit a linear regression
        lr = linear_model.Ridge(alpha=alpha, fit_intercept=False)
        lr.fit(X_train_features, y_train)

        # Predict the targets for train, test and plot data
        y_train_pred = lr.predict(X_train_features)
        y_test_pred = lr.predict(X_test_features)

        # Show the MSE
        print(
            f"Degree {d} : training error is {np.linalg.norm(y_train - y_train_pred)}, test error is
        )

# Fit
PolyFit(degrees, X_train, y_train, X_test, y_test, 0.5)

# Output
# Degree 10 : training error is 0.22004580593405246, test error is 26.421400718553116
# Degree 12 : training error is 0.20674169975405737, test error is 88.50310100797962
# Degree 14 : training error is 0.20433046276146066, test error is 262.1837993940009
```

Answer the following questions.

1. Why is the regularization parameter chosen to be different from zero now?

2. Which is the optimal model? Is it independent from the value of the regularization strength.

3. What is a systematic way to select the right value of the regularization strength?