module_5_assignment

April 12, 2020

1 Graded Assessment

In this assessment you will write a full end-to-end training process using gluon and MXNet. We will train the LeNet-5 classifier network on the MNIST dataset. The network will be defined for you but you have to fill in code to prepare the dataset, train the network, and evaluate it's performance on a held out dataset.

[1]:
```
from pathlib import Path
from mxnet import gluon, metric, autograd, init, nd
import os
```

[2]:
```
M5_DATA = Path(os.getenv('DATA_DIR', '../../data'), 'module_5')
M5_IMAGES = Path(M5_DATA, 'images')
```

1.1 Question 1

1.1.1 Prepare and the data and construct the dataloader

- First, get the MNIST dataset from `gluon.data.vision.datasets`. Use
- Don't forget the ToTensor and normalize Transformations. Use `0.13` and `0.31` as the mean and standard deviation respectively
- Construct the dataloader with the batch size provide. Ensure that the train_dataloader is shuffled.

CAUTION!: Although the notebook interface has internet connectivity, the **autograders are not permitted to access the internet**. We have already downloaded the correct models and data for you to use so you don't need access to the internet. Set the `root` parameter to `M5_IMAGES` when using a preset dataset. Usually, in the real world, you have internet access, so setting the `root` parameter isn't required (and it's set to `~/.mxnet` by default).

[9]:
```
import os
from pathlib import Path
from mxnet.gluon.data.vision import transforms

def get_mnist_data(batch=128):
    """
```

```
    Should construct a dataloader with the MNIST Dataset with the necessary␣
↪transforms applied.

    :param batch: batch size for the DataLoader.
    :type batch: int

    :return: a tuple of the training and validation DataLoaders
    :rtype: (gluon.data.DataLoader, gluon.data.DataLoader)
    """

    train_data = gluon.data.vision.datasets.MNIST(root = M5_IMAGES, train=True)
    val_data   = gluon.data.vision.datasets.MNIST(root = M5_IMAGES, train=False)

    data_transform = transforms.Compose([
        gluon.data.vision.transforms.ToTensor(),
        gluon.data.vision.transforms.Normalize(mean=0.13, std=0.31)
    ])

    train_dataloader = gluon.data.DataLoader(train_data.
↪transform_first(data_transform), batch, shuffle=True)
    validation_dataloader = gluon.data.DataLoader(val_data.
↪transform_first(data_transform), batch, shuffle=False)

    return train_dataloader, validation_dataloader
```

[10]:
```
t, v = get_mnist_data()
assert isinstance(t, gluon.data.DataLoader)
assert isinstance(v, gluon.data.DataLoader)

d, l = next(iter(t))
assert d.shape == (128, 1, 28, 28) #check Channel First and Batch Size
assert l.shape == (128,)

assert nd.max(d).asscalar() <= 2.9 # check for normalization
assert nd.min(d).asscalar() >= -0.5 # check for normalization
```

1.2 Question 2

1.2.1 Write the training loop

- Create the loss function. This should be a loss function suitable for multi-class classification.
- Create the metric accumulator. This should the compute and store the accuracy of the model during training
- Create the trainer with the `adam` optimizer and learning rate of `0.002`

- Write the training loop

[16]:
```
def train(network, training_dataloader, batch_size, epochs):
    """
    Should take an initialized network and train that network using data from␣
↪the data loader.

    :param network: initialized gluon network to be trained
    :type network: gluon.Block

    :param training_dataloader: the training DataLoader provides batches for␣
↪data for every iteration
    :type training_dataloader: gluon.data.DataLoader

    :param batch_size: batch size for the DataLoader.
    :type batch_size: int

    :param epochs: number of epochs to train the DataLoader
    :type epochs: int

    :return: tuple of trained network and the final training accuracy
    :rtype: (gluon.Block, float)
    """

    loss_fn = gluon.loss.SoftmaxCrossEntropyLoss()
    train_acc = metric.Accuracy()
    trainer = gluon.Trainer(net.collect_params(), 'adam', {'learning_rate':0.
↪002})

    for epoch in range(epochs):
        train_loss = 0.

        for data, label in training_dataloader:
            with autograd.record():
                output = network(data)
                loss = loss_fn(output, label)
            loss.backward()

            trainer.step(batch_size)

            train_loss += loss.mean().asscalar()
            train_acc.update(label, output)

            training_accuracy = train_acc.get()[1]

        print("Epoch [%d], Loss: %.3f, Acc: %3f" %(
            epoch, train_loss/len(training_dataloader), training_accuracy))
```

```
    network.save_parameters("trained_net.params")

    return network, training_accuracy
```

Let's define and initialize a network to test the train function.

[17]:
```
net = gluon.nn.Sequential()
net.add(gluon.nn.Conv2D(channels=6, kernel_size=5, activation='relu'),
        gluon.nn.MaxPool2D(pool_size=2, strides=2),
        gluon.nn.Conv2D(channels=16, kernel_size=3, activation='relu'),
        gluon.nn.MaxPool2D(pool_size=2, strides=2),
        gluon.nn.Flatten(),
        gluon.nn.Dense(120, activation="relu"),
        gluon.nn.Dense(84, activation="relu"),
        gluon.nn.Dense(10))
net.initialize(init=init.Xavier())
```

[18]:
```
n, ta = train(net, t, 128, 5)
assert ta >= .95

d, l = next(iter(v))
p = (n(d).argmax(axis=1))
assert (p.asnumpy() == l.asnumpy()).sum()/128.0 > .95
```

```
Epoch [0], Loss: 0.205, Acc: 0.937017
Epoch [1], Loss: 0.061, Acc: 0.958800
Epoch [2], Loss: 0.045, Acc: 0.967828
Epoch [3], Loss: 0.035, Acc: 0.973125
Epoch [4], Loss: 0.031, Acc: 0.976560
```

1.3 Question 3

1.3.1 Write the validation loop

- Create the metric accumulator. This should the compute and store the accuracy of the model on the validation set
- Write the validation loop

[19]:
```
def validate(network, validation_dataloader):
    """
    Should compute the accuracy of the network on the validation set.

    :param network: initialized gluon network to be trained
    :type network: gluon.Block
```

```
    :param validation_dataloader: the training DataLoader provides batches for␣
↪data for every iteration
    :type validation_dataloader: gluon.data.DataLoader

    :return: validation accuracy
    :rtype: float
    """
    preds = []
    valid_acc = metric.Accuracy()
    network.load_parameters("trained_net.params")

    for data, label in validation_dataloader:
        output = network(data)
        valid_acc.update(label, output)

    validation_accuracy = valid_acc.get()[1]

    return validation_accuracy
```

[20]:
```
assert validate(n, v) > .95
```

[ ]:
```

```