E11

Connor White & David Chalifoux

We decided to use the Pytorch framework for our back-propagating neural network. We based our work off of a straight-forward example by Nikolai Janakiev. We used the iris data set from sklearn's dataset package. The input data is then scaled to have a mean of 0 and variance of 1 using sklearn's `StandardScaler`.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

From there, the data is split into three groups for training, testing, and validating using sklearn's `train_test_split` function. In this case, 60% of the data is used for training, while 40% is split evenly for training and validating.

```
# Split the data set into training, testing, and validation
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.4
)  # Split for training
X_test, X_validate, y_test, y_validate = train_test_split(
    X_test, y_test, test_size=0.5
)  # Split for validation
```

Using pytorch, we define a neural network with three layers. Layer 1 is the input layer with 4 input features and 50 output features, layer 2 is the hidden layer with 50 input and output features, and layer 3 is the output layer with 50 input features and 3 output features. We use Pytorch's "Adam" optimizer with a learning rate of `0.001` and Pytorch's `CrossEntropyLoss` as our loss function.

```
class Model(nn.Module):
    def __init__(self, input_dim):
```

```
        super(Model, self).__init__()
        self.layer1 = nn.Linear(input_dim, 50)
        self.layer2 = nn.Linear(50, 50)
        self.layer3 = nn.Linear(50, 3)

    def forward(self, x):
        x = F.relu(self.layer1(x))
        x = F.relu(self.layer2(x))
        x = F.softmax(self.layer3(x), dim=1)
        return x

model = Model(X_train.shape[1])
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
loss_fn = nn.CrossEntropyLoss()
```

For each epoch, the model is trained using Pytorch's built-in back-propagation method and tested again the training and validation data.

```
for epoch in range(EPOCHS):
    y_pred = model(X_train)
    loss = loss_fn(y_pred, y_train)

    # Zero gradients
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    with torch.no_grad():
        # Test
        y_pred_test = model(X_test)
        correct_test = (torch.argmax(y_pred_test, dim=1) == y_test).type(
            torch.FloatTensor
        )

        # Validate
        y_pred_validate = model(X_validate)
        correct_validate = (torch.argmax(y_pred_validate, dim=1) == y_validate).type(
            torch.FloatTensor
        )
        print(
            "Training epoch",
            epoch + 1,
            "loss %.4f" % loss.item(),
            "test accuracy %.4f" % correct_test.mean().item(),
            "validation accuracy %.4f" % correct_validate.mean().item(),
        )
```

Result

This model has been able to reach a high level of accuracy within 100 epochs. Test and validation accuracy is typical around 90-100%.

```
Data length: 150
Training data length: 90
Testing data length: 30
Validation data length: 30
Training epoch 1 loss 1.0862 test accuracy 0.5000 validation accuracy 0.5667
Training epoch 2 loss 1.0807 test accuracy 0.5000 validation accuracy 0.6000
Training epoch 3 loss 1.0752 test accuracy 0.7333 validation accuracy 0.7667
Training epoch 4 loss 1.0698 test accuracy 0.8000 validation accuracy 0.8000
Training epoch 5 loss 1.0643 test accuracy 0.8000 validation accuracy 0.8000
Training epoch 6 loss 1.0588 test accuracy 0.8333 validation accuracy 0.8333
Training epoch 7 loss 1.0532 test accuracy 0.8333 validation accuracy 0.8333
Training epoch 8 loss 1.0476 test accuracy 0.8333 validation accuracy 0.8667
Training epoch 9 loss 1.0420 test accuracy 0.8667 validation accuracy 0.8667
Training epoch 10 loss 1.0362 test accuracy 0.8667 validation accuracy 0.9000
Training epoch 11 loss 1.0304 test accuracy 0.8667 validation accuracy 0.8667
Training epoch 12 loss 1.0246 test accuracy 0.8667 validation accuracy 0.8667
Training epoch 13 loss 1.0186 test accuracy 0.8667 validation accuracy 0.8667
Training epoch 14 loss 1.0126 test accuracy 0.8333 validation accuracy 0.8667
Training epoch 15 loss 1.0065 test accuracy 0.8333 validation accuracy 0.8667
Training epoch 16 loss 1.0004 test accuracy 0.7667 validation accuracy 0.8667
Training epoch 17 loss 0.9942 test accuracy 0.7667 validation accuracy 0.8333
Training epoch 18 loss 0.9879 test accuracy 0.7667 validation accuracy 0.8333
Training epoch 19 loss 0.9814 test accuracy 0.7667 validation accuracy 0.8333
Training epoch 20 loss 0.9749 test accuracy 0.7667 validation accuracy 0.8333
Training epoch 21 loss 0.9683 test accuracy 0.7667 validation accuracy 0.8333
Training epoch 22 loss 0.9616 test accuracy 0.7667 validation accuracy 0.8333
Training epoch 23 loss 0.9549 test accuracy 0.7667 validation accuracy 0.8333
Training epoch 24 loss 0.9481 test accuracy 0.7667 validation accuracy 0.8333
Training epoch 25 loss 0.9413 test accuracy 0.7667 validation accuracy 0.8333
...
Training epoch 500 loss 0.5657 test accuracy 0.9667 validation accuracy 1.0000
```