H5P1

November 1, 2024

[ ]:
```
from torchvision import datasets
from torchvision.transforms import ToTensor
from torch.utils.data import DataLoader
import torch
from torch import nn
import numpy as np
import matplotlib.pyplot as plt
```

[ ]:
```
# Load the USPS dataset
train_data = datasets.USPS(root='usps', download=True, transform=ToTensor(),␣
 ↪train=True)
test_data = datasets.USPS(root='usps', download=True, transform=ToTensor(),␣
 ↪train=False)

# Create DataLoaders for training and testing
train_loader = DataLoader(train_data, batch_size=1024, shuffle=True)
test_loader = DataLoader(test_data, batch_size=len(test_data), shuffle=False)
```

[ ]:
```
# Define the MLP model with 2 hidden layers, both with 128 units
class MLP(nn.Module):
    def __init__(self):
        super().__init__()
        self.flatten = nn.Flatten()
        self.mlp = nn.Sequential(
            nn.Linear(16 * 16, 128),  # Input layer to first hidden layer
            nn.ReLU(),
            nn.Linear(128, 128),      # First hidden layer to second hidden␣
 ↪layer
            nn.ReLU(),
            nn.Linear(128, 128),      # Second hidden layer
            nn.ReLU(),
            nn.Linear(128, 10)        # Output layer for 10 classes
        )

    def forward(self, X):
        return self.mlp(self.flatten(X))
```

```
model = MLP().to('cuda')
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.5)
```

[ ]:
```
%%time
epochs = 1000
CE = torch.zeros((epochs))
Training = torch.zeros((epochs))
Test = torch.zeros((epochs))
for epoch in range(epochs):
    cumulative_accuracy = 0
    cumulative_loss = 0
    for X, Y in train_loader:
        X, Y = X.to('cuda'), Y.to('cuda')
        out = model(X)
        loss = loss_fn(out, Y)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        cumulative_loss += loss.item()
        cumulative_accuracy += (out.argmax(axis=1) == Y).sum().item()
    CE[epoch] = cumulative_loss / len(train_loader)
    Training[epoch] = cumulative_accuracy / len(train_data)
    with torch.no_grad():
        for Xt, Yt in test_loader:
            Xt, Yt = Xt.to('cuda'), Yt.to('cuda')
            test_out = model(Xt)
            test_accuracy_epoch = (test_out.argmax(axis=1) == Yt).sum().item() /
 ↪ len(test_data)
        Test[epoch] = test_accuracy_epoch
    if (epoch + 1) % 100 == 0:
        print(f"Epoch {epoch + 1}/{epochs} | Loss: {CE[epoch]:.4f} | Training␣
↪Accuracy: {Training[epoch]:.4f} | Test Accuracy: {Test[epoch]:.4f}")
```

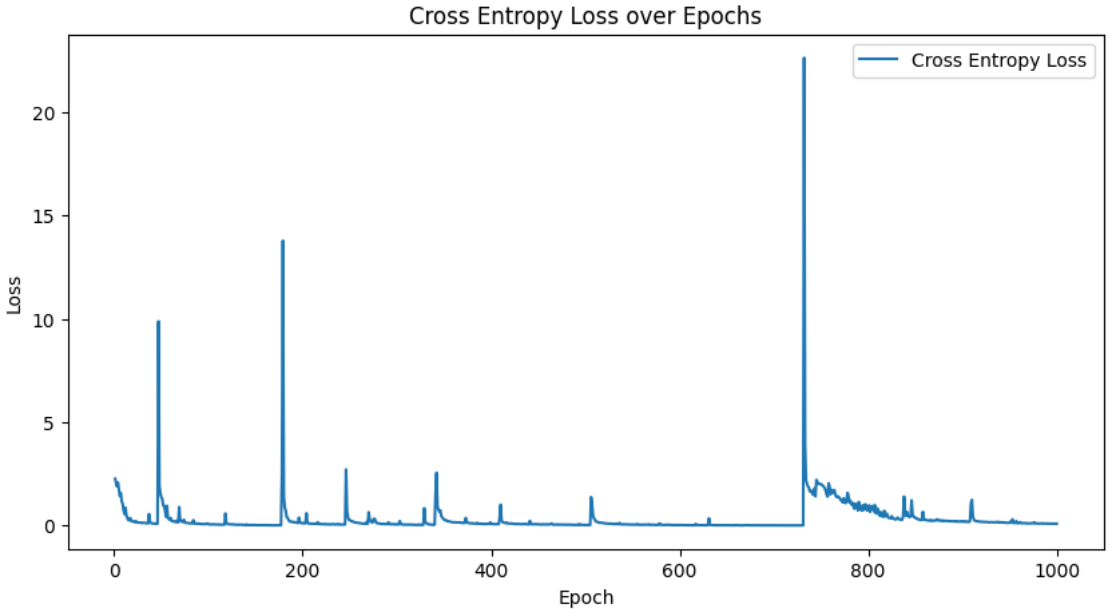
```
Epoch 100/1000 | Loss: 0.0631 | Training Accuracy: 0.9815 | Test Accuracy:
0.9268
Epoch 200/1000 | Loss: 0.1229 | Training Accuracy: 0.9624 | Test Accuracy:
0.9033
Epoch 300/1000 | Loss: 0.0545 | Training Accuracy: 0.9824 | Test Accuracy:
0.9178
Epoch 400/1000 | Loss: 0.0751 | Training Accuracy: 0.9772 | Test Accuracy:
0.9203
Epoch 500/1000 | Loss: 0.0304 | Training Accuracy: 0.9938 | Test Accuracy:
0.9178
Epoch 600/1000 | Loss: 0.0187 | Training Accuracy: 0.9945 | Test Accuracy:
0.9297
Epoch 700/1000 | Loss: 0.0057 | Training Accuracy: 0.9989 | Test Accuracy:
0.9273
```

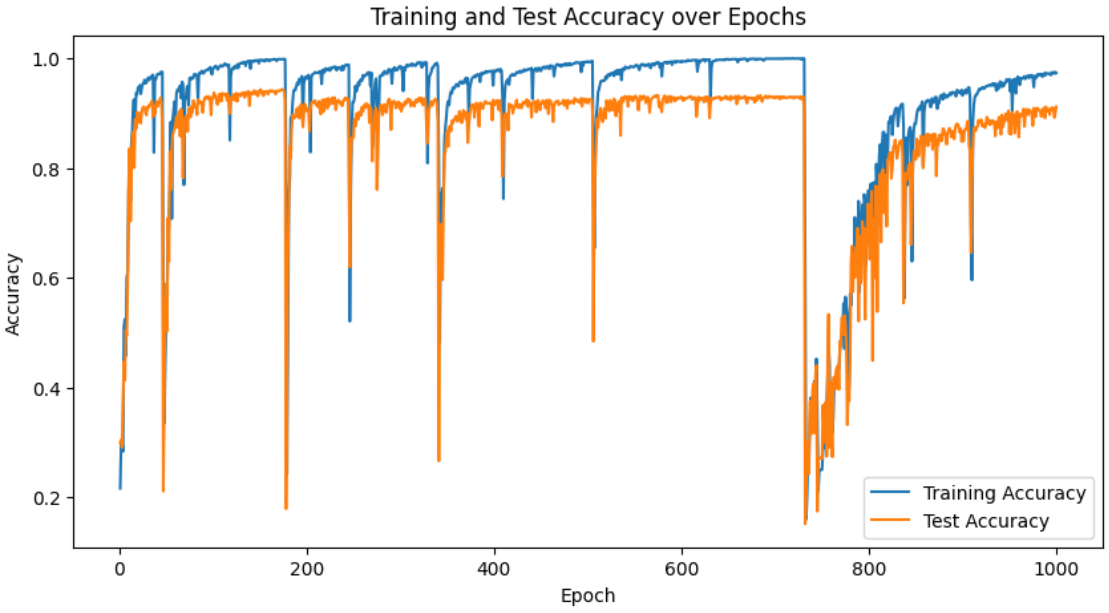
```
Epoch 800/1000 | Loss: 0.7398 | Training Accuracy: 0.7594 | Test Accuracy:
0.7260
Epoch 900/1000 | Loss: 0.2131 | Training Accuracy: 0.9317 | Test Accuracy:
0.8660
Epoch 1000/1000 | Loss: 0.0930 | Training Accuracy: 0.9728 | Test Accuracy:
0.9113
CPU times: user 12min 38s, sys: 1.7 s, total: 12min 39s
Wall time: 12min 44s
```

[ ]:
```python
epochs_range = np.arange(1, epochs + 1)
CE_np = CE.cpu().numpy()
Training_np = Training.cpu().numpy()
Test_np = Test.cpu().numpy()

plt.figure(figsize=(10, 5))
plt.plot(epochs_range, CE_np, label="Cross Entropy Loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.title("Cross Entropy Loss over Epochs")
plt.legend()
plt.show()

plt.figure(figsize=(10, 5))
plt.plot(epochs_range, Training_np, label="Training Accuracy")
plt.plot(epochs_range, Test_np, label="Test Accuracy")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.title("Training and Test Accuracy over Epochs")
plt.legend()
plt.show()
```


Cross Entropy Loss over Epochs
Cross Entropy Loss
Loss
Epoch
0
5
10
15
20
0
200
400
600
800
1000


Training and Test Accuracy over Epochs
Training Accuracy
Test Accuracy
Accuracy
Epoch
0.2
0.4
0.6
0.8
1.0
0
200
400
600
800
1000

H5P2

November 1, 2024

[1]:
```
from torchvision import datasets
from torchvision.transforms import ToTensor
from torch.utils.data import DataLoader
import torch
from torch import nn
import numpy as np
import matplotlib.pyplot as plt
```

[2]:
```
# Load the USPS dataset
train_data = datasets.USPS(root='usps', download=True, transform=ToTensor(),␣
 ↪train=True)
test_data = datasets.USPS(root='usps', download=True, transform=ToTensor(),␣
 ↪train=False)

# Create DataLoaders for training and testing
train_loader = DataLoader(train_data, batch_size=1024, shuffle=True)
test_loader = DataLoader(test_data, batch_size=len(test_data), shuffle=False)
```

```
Downloading
https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multiclass/usps.bz2 to
usps/usps.bz2

100%|      | 6.58M/6.58M [00:01<00:00, 5.99MB/s]

Downloading
https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multiclass/usps.t.bz2 to
usps/usps.t.bz2

100%|      | 1.83M/1.83M [00:00<00:00, 2.16MB/s]
```

[3]:
```
# Define the MLP model with 1 hidden layer of width 64, following the specified␣
 ↪structure
class MLP(nn.Module):
    def __init__(self):
        super().__init__()
        self.flatten = nn.Flatten()
        self.mlp = nn.Sequential(
            nn.Linear(16 * 16, 64),  # Input layer to hidden layer
            nn.ReLU(),
```

```
            nn.Linear(64, 64),        # Hidden layer
            nn.ReLU(),
            nn.Linear(64, 10)         # Output layer
        )

    def forward(self, X):
        return self.mlp(self.flatten(X))


model = MLP().to('cuda')
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.5)
```

[4]:
```
%%time
epochs = 1000
CE = torch.zeros((epochs))
Training = torch.zeros((epochs))
Test = torch.zeros((epochs))
for epoch in range(epochs):
    cumulative_accuracy = 0
    cumulative_loss = 0
    for X, Y in train_loader:
        X, Y = X.to('cuda'), Y.to('cuda')
        out = model(X)
        loss = loss_fn(out, Y)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        cumulative_loss += loss.item()
        cumulative_accuracy += (out.argmax(axis=1) == Y).sum().item()
    CE[epoch] = cumulative_loss / len(train_loader)
    Training[epoch] = cumulative_accuracy / len(train_data)
    with torch.no_grad():
        for Xt, Yt in test_loader:
            Xt, Yt = Xt.to('cuda'), Yt.to('cuda')
            test_out = model(Xt)
            test_accuracy_epoch = (test_out.argmax(axis=1) == Yt).sum().item() /
 ↪ len(test_data)
        Test[epoch] = test_accuracy_epoch
    if (epoch + 1) % 100 == 0:
        print(f"Epoch {epoch + 1}/{epochs} | Loss: {CE[epoch]:.4f} | Training␣
 ↪Accuracy: {Training[epoch]:.4f} | Test Accuracy: {Test[epoch]:.4f}")
```

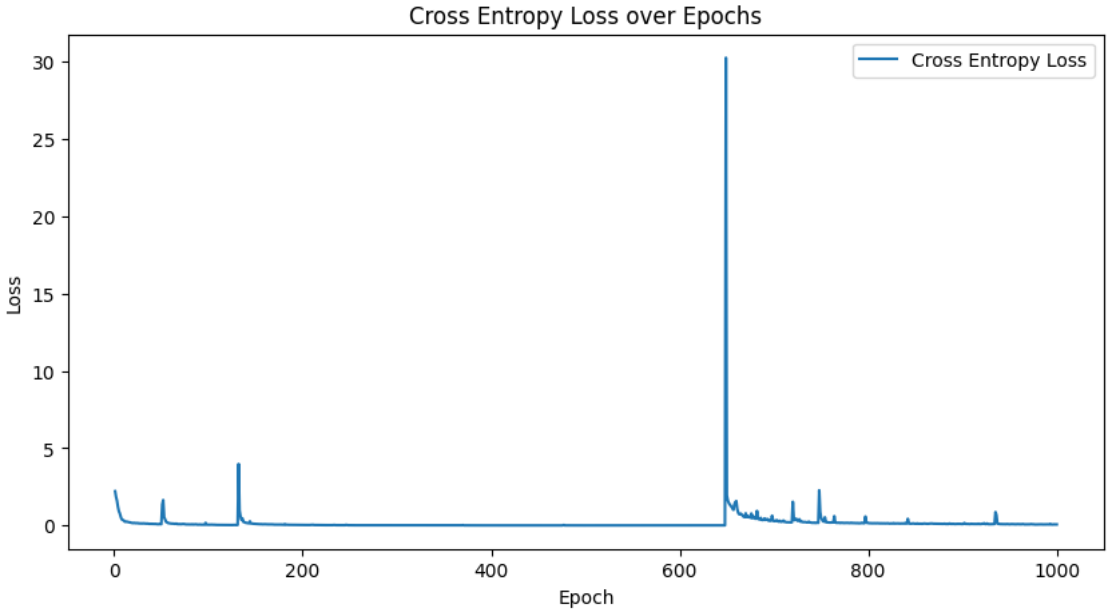
```
Epoch 100/1000 | Loss: 0.0483 | Training Accuracy: 0.9868 | Test Accuracy:
0.9387
Epoch 200/1000 | Loss: 0.0355 | Training Accuracy: 0.9894 | Test Accuracy:
0.9287
Epoch 300/1000 | Loss: 0.0086 | Training Accuracy: 0.9992 | Test Accuracy:
```

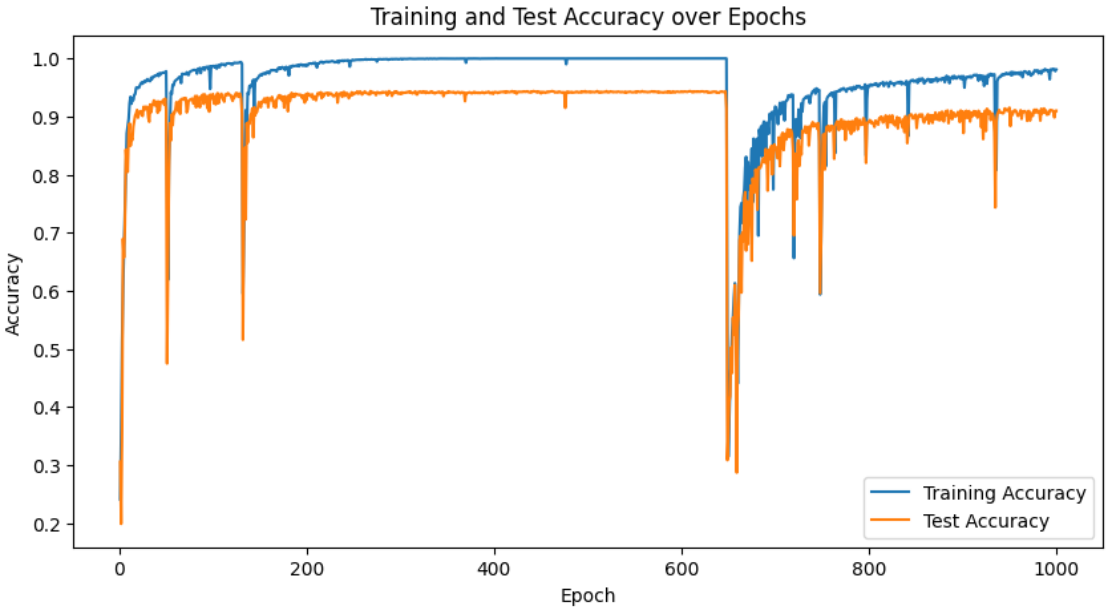
```
0.9412
Epoch 400/1000 | Loss: 0.0047 | Training Accuracy: 0.9996 | Test Accuracy:
0.9417
Epoch 500/1000 | Loss: 0.0031 | Training Accuracy: 0.9999 | Test Accuracy:
0.9422
Epoch 600/1000 | Loss: 0.0023 | Training Accuracy: 0.9999 | Test Accuracy:
0.9432
Epoch 700/1000 | Loss: 0.2746 | Training Accuracy: 0.9172 | Test Accuracy:
0.8475
Epoch 800/1000 | Loss: 0.1600 | Training Accuracy: 0.9510 | Test Accuracy:
0.8909
Epoch 900/1000 | Loss: 0.0972 | Training Accuracy: 0.9639 | Test Accuracy:
0.9088
Epoch 1000/1000 | Loss: 0.0675 | Training Accuracy: 0.9809 | Test Accuracy:
0.9098
CPU times: user 12min 14s, sys: 2.01 s, total: 12min 16s
Wall time: 12min 24s
```

[5]:
```
epochs_range = np.arange(1, epochs + 1)
CE_np = CE.cpu().numpy()
Training_np = Training.cpu().numpy()
Test_np = Test.cpu().numpy()

plt.figure(figsize=(10, 5))
plt.plot(epochs_range, CE_np, label="Cross Entropy Loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.title("Cross Entropy Loss over Epochs")
plt.legend()
plt.show()

plt.figure(figsize=(10, 5))
plt.plot(epochs_range, Training_np, label="Training Accuracy")
plt.plot(epochs_range, Test_np, label="Test Accuracy")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.title("Training and Test Accuracy over Epochs")
plt.legend()
plt.show()
```


Cross Entropy Loss over Epochs
Cross Entropy Loss
Loss
Epoch
0
5
10
15
20
25
30
0
200
400
600
800
1000


Training and Test Accuracy over Epochs
Training Accuracy
Test Accuracy
Accuracy
Epoch
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
0
200
400
600
800
1000

H5P3

November 2, 2024

[12]:
```
from torchvision import datasets
from torchvision.transforms import ToTensor
from torch.utils.data import DataLoader
import torch
from torch import nn
import numpy as np
import matplotlib.pyplot as plt
```

[13]:
```
train_data = datasets.USPS(root='usps', download=True, transform=ToTensor(),␣
 ↪train=True)
test_data = datasets.USPS(root='usps', download=True, transform=ToTensor(),␣
 ↪train=False)

# Create DataLoaders for training and testing
train_loader = DataLoader(train_data, batch_size=128, shuffle=True)
test_loader = DataLoader(test_data, batch_size=len(test_data), shuffle=False)
```

[14]:
```
# Define the MLP model with 2 hidden layers, both with 128 units
class MLP(nn.Module):
    def __init__(self):
        super().__init__()
        self.flatten = nn.Flatten()
        self.mlp = nn.Sequential(
            nn.Linear(16 * 16, 128),  # Input layer to first hidden layer
            nn.ReLU(),
            nn.Linear(128, 128),      # First hidden layer to second hidden␣
 ↪layer
            nn.ReLU(),
            nn.Linear(128, 128),      # Second hidden layer
            nn.ReLU(),
            nn.Linear(128, 10)        # Output layer for 10 classes
        )

    def forward(self, X):
        return self.mlp(self.flatten(X))
```

```
model = MLP().to('cuda')
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.5)
```

[15]:
```
%%time
epochs = 1000
CE = torch.zeros((epochs))
Training = torch.zeros((epochs))
Test = torch.zeros((epochs))
for epoch in range(epochs):
    cumulative_accuracy = 0
    cumulative_loss = 0
    for X, Y in train_loader:
        X, Y = X.to('cuda'), Y.to('cuda')
        out = model(X)
        loss = loss_fn(out, Y)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        cumulative_loss += loss.item()
        cumulative_accuracy += (out.argmax(axis=1) == Y).sum().item()
    CE[epoch] = cumulative_loss / len(train_loader)
    Training[epoch] = cumulative_accuracy / len(train_data)
    with torch.no_grad():
        for Xt, Yt in test_loader:
            Xt, Yt = Xt.to('cuda'), Yt.to('cuda')
            test_out = model(Xt)
            test_accuracy_epoch = (test_out.argmax(axis=1) == Yt).sum().item() /
 ↪ len(test_data)
        Test[epoch] = test_accuracy_epoch
    if (epoch + 1) % 100 == 0:
        print(f"Epoch {epoch + 1}/{epochs} | Loss: {CE[epoch]:.4f} | Training␣
↪Accuracy: {Training[epoch]:.4f} | Test Accuracy: {Test[epoch]:.4f}")
```

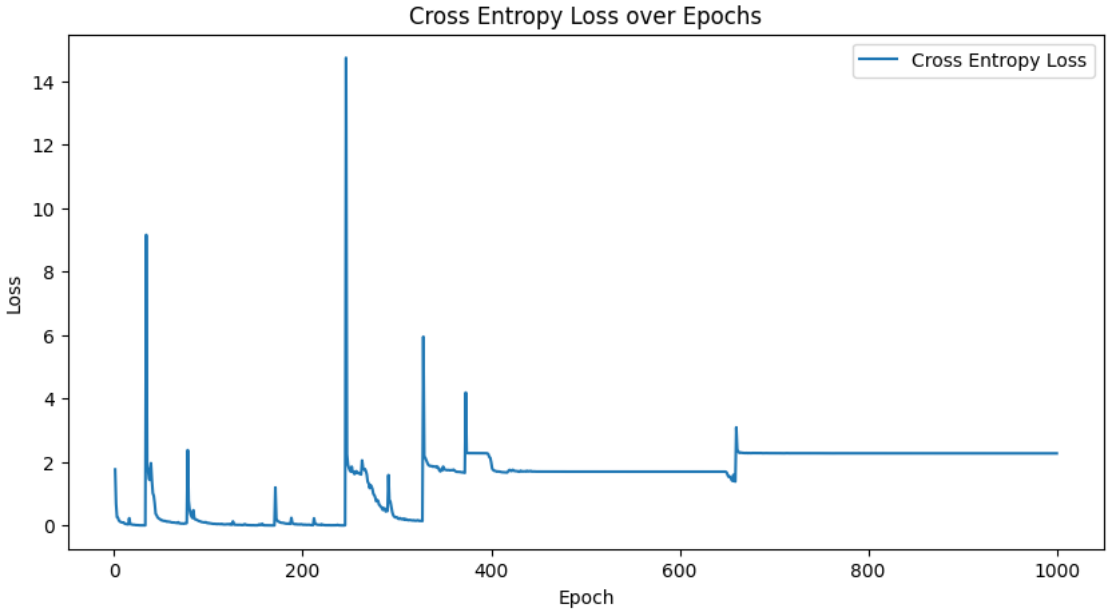
```
Epoch 100/1000 | Loss: 0.0847 | Training Accuracy: 0.9717 | Test Accuracy:
0.9223
Epoch 200/1000 | Loss: 0.0234 | Training Accuracy: 0.9926 | Test Accuracy:
0.9253
Epoch 300/1000 | Loss: 0.2611 | Training Accuracy: 0.9281 | Test Accuracy:
0.8749
Epoch 400/1000 | Loss: 1.9852 | Training Accuracy: 0.2347 | Test Accuracy:
0.3039
Epoch 500/1000 | Loss: 1.6971 | Training Accuracy: 0.3000 | Test Accuracy:
0.3044
Epoch 600/1000 | Loss: 1.6961 | Training Accuracy: 0.2980 | Test Accuracy:
0.3044
Epoch 700/1000 | Loss: 2.2770 | Training Accuracy: 0.1625 | Test Accuracy:
0.1315
```

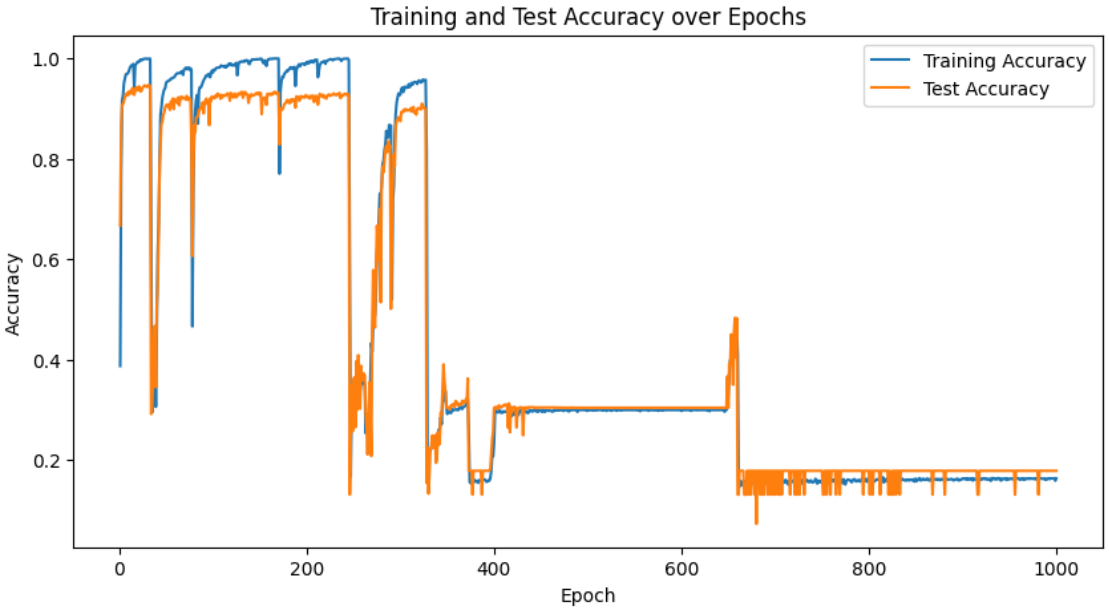
```
Epoch 800/1000 | Loss: 2.2732 | Training Accuracy: 0.1643 | Test Accuracy:
0.1789
Epoch 900/1000 | Loss: 2.2735 | Training Accuracy: 0.1638 | Test Accuracy:
0.1789
Epoch 1000/1000 | Loss: 2.2716 | Training Accuracy: 0.1638 | Test Accuracy:
0.1789
CPU times: user 13min 34s, sys: 2.98 s, total: 13min 37s
Wall time: 13min 42s
```

[16]:
```python
epochs_range = np.arange(1, epochs + 1)
CE_np = CE.cpu().numpy()
Training_np = Training.cpu().numpy()
Test_np = Test.cpu().numpy()

plt.figure(figsize=(10, 5))
plt.plot(epochs_range, CE_np, label="Cross Entropy Loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.title("Cross Entropy Loss over Epochs")
plt.legend()
plt.show()

plt.figure(figsize=(10, 5))
plt.plot(epochs_range, Training_np, label="Training Accuracy")
plt.plot(epochs_range, Test_np, label="Test Accuracy")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.title("Training and Test Accuracy over Epochs")
plt.legend()
plt.show()
```


Cross Entropy Loss over Epochs
Cross Entropy Loss
Loss
Epoch
0
2
4
6
8
10
12
14
0
200
400
600
800
1000


Training and Test Accuracy over Epochs
Training Accuracy
Test Accuracy
Accuracy
Epoch
0.2
0.4
0.6
0.8
1.0
0
200
400
600
800
1000