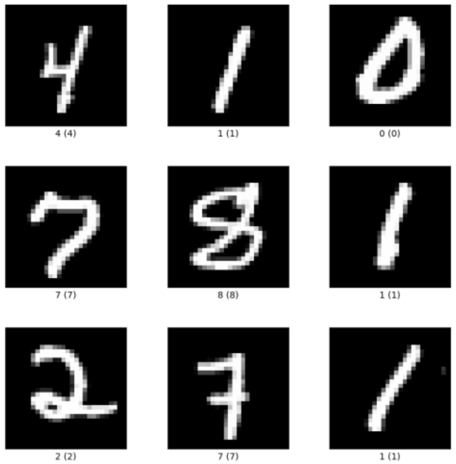
The MNIST dataset

The MNIST dataset

- **MNIST Dataset**:
- A benchmark in machine learning and computer vision,
- it consists of 70,000 grayscale images of handwritten digits from 0 to 9.
- Each image is 28×28 pixels in size.



Lab Session Objectives

- In this lab session we will train a Multilayer Perceptron on the MNIST dataset and we will evaluate how changing:
- NN structure
- The activation function
- Loss Functions
- Optimizers
- Learning Rate
- Batch size

Install Pytorch

- In order to do this lab session in roboenv2 you need to install pytorch

- **If you have cuda (nvidia GPU) you can do**
- mamba install pytorch torchvision torchaudio pytorch-cuda=11.8 -c pytorch -c nvidia

- For pytorch-cuda=11.8 here you need to specify your cuda version

Install Pytorch

- In order to do this lab session in roboenv2 you need to install pytorch

- **If you DO NOT have cuda (nvidia GPU)**
- mamba install pytorch torchvision torchaudio cpuonly -c pytorch

NN Structure

```python
# 2. Model Construction
class MLP(nn.Module):
    def __init__(self):
        super(MLP, self).__init__()
        self.flatten = nn.Flatten()
        self.hidden = nn.Linear(28*28, 128)  # Input layer to hidden layer
        self.relu = nn.ReLU()
        self.output = nn.Linear(128, 10)     # Hidden layer to output layer
        self.softmax = nn.LogSoftmax(dim=1)  # Use LogSoftmax for numerical stability

    def forward(self, x):
        x = self.flatten(x)
        x = self.hidden(x)
        x = self.relu(x)
        x = self.output(x)
        x = self.softmax(x)
        return x

model = MLP()
```

NN Structure

- For the hidden layer:
- Increase the number of nodes to 256
- Reduce the number of nodes 64
- Add another hidden layer hidden2

```
self.hidden2 = nn.Linear(128, 64)
```

```
x = self.hidden2(x)
```

NN structure

- For each test when possible, restore back the value to 128
- Compare the different loss on training and prediction accuracy for each change

Loss Function

- Use Mean Squared Error (MSE) instead of Negative Log Likelihood Loss

```
# Modify the training loop
criterion = nn.MSELoss()

for epoch in range(epochs):
    model.train()
    epoch_loss = 0
    correct = 0

    for data, target in train_loader:
        optimizer.zero_grad()
        output = model(data)
        target_one_hot = one_hot_encode(target)  # Convert target to one-hot
        loss = criterion(torch.exp(output), target_one_hot)  # Use exp(output) to invert LogSoftmax
        loss.backward()
        optimizer.step()

        epoch_loss += loss.item()
        pred = output.argmax(dim=1
        correct += pred.eq(target.view_as(pred)).sum().item()

    # Continue as before
```

(function) view_as: Any

Activation Function

- Put the neural network back to the original shape and replace all the RELU

```
self.relu = nn.ReLU()
```

- With a sigmoid activation functions and see how the loss change

Loss Function

- Test which loss function increase the prediction capability on the test set and check the loss on the training set

Optimizers

- Replace sdg with adam

```
optimizer = optim.Adam(model.parameters(), lr=0.001)  # Adam uses a different default learning rate
```

- Replace sdg with RMSprop

```
optimizer = optim.RMSprop(model.parameters(), lr=0.001)
```

- See which of the 3 produces the best prediction results on the test and check the loss on the training set

Learning rate

- Bring back stochastic gradient

- Set learning rate to 1.0
- Set learning rate to 0.0001

- Test the different Learning rate and see the impact on the loss on the training and the prediction accuracy on the test

Batch Size

- Train with batch size
- 64
- 128
- 256

- See what is the impact on the training loss function

```
train_loader = DataLoader(dataset=train_dataset, batch_size=32, shuffle=True)
```

- The code is available here:

- https://github.com/VModugno/lab_sessions_COMP0245_PUBLIC/tree/main/week_4