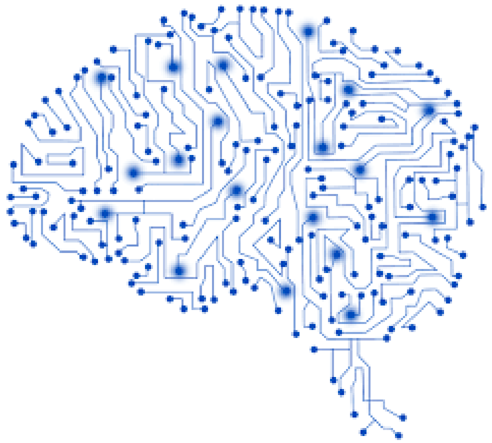
University of Minho
School of Engineering



Aprendizagem Profunda

Deep Neural Network – MLP Multiclass

AP @ MEI/1º ano – 2º Semestre

Victor Alves

Part II

Hands On

MLP for multiclass classification of images

MNIST (Modified National Institute of Standards and Technology) Dataset

It is considered the "hello world" dataset of *computer vision*.

- Dataset of manually written digit images
- Inputs: 28x28 pixels images
- Output: class representing the digit (10 classes, digits 0-9)
- 70k images of which 60k are for training and 10k for testing
- 2 attributes: the image id and its label

We will use a neural network to classify the digit in each 28x28 image.

0. Prepare the setup

Install pytorch (if needed)

Imports

Constants

```
PATH = './'
PATH_TRAIN = './mnist_train.csv'
PATH_TEST = './mnist_test.csv'

device = torch.device("cpu")

BATCH_SIZE = 32
```

1. Prepare the data

```
class CSVDataset(Dataset):
  def __init__(self, path_train, path_test):
        df_train = pd.read_csv(path_train, header=0)
        df_test = pd.read_csv(path_test, header=0)

        self.x_train = df_train.values[:, 1:]
        self.y_train = df_train.values[:, 0]
        self.x_test = df_test.values[:, 1:]
        self.y_test = df_test.values[:, 0]

        self.x_train = self.x_train.astype('float32')
        self.x_test = self.x_test.astype('float32')
        self.y_train = self.y_train.astype('long')
        self.y_test = self.y_test.astype('long')
```

1. Prepare the data

```
def __len_train__(self):
    return len(self.x_train)

def __len_test__(self):
    return len(self.x_test)

def __getitem_train__(self, idx):
    return [self.x_train[idx], self.y_train[idx]]

def __getitem_test__(self, idx):
    return [self.x_test[idx], self.y_test[idx]]
```

```
def get_splits_flat(self):
        x_train  = torch.from_numpy(np.array(self.x_train))
        y_train  = torch.from_numpy(np.array(self.y_train))
        x_test  = torch.from_numpy(np.array(self.x_test))
        y_test  = torch.from_numpy(np.array(self.y_test))
        train = torch.utils.data.TensorDataset(x_train,y_train)
        test = torch.utils.data.TensorDataset(x_test,y_test)
        return train, test
```

1. Prepare the data

```
def prepare_data_flat(path_train, path_test):
    dataset = CSVDataset(path_train, path_test)
    train, test = dataset.get_splits_flat()

    train_dl = DataLoader(train, batch_size=BATCH_SIZE, shuffle=True)
    test_dl = DataLoader(test, batch_size=BATCH_SIZE, shuffle=True)
    train_dl_all = DataLoader(train, batch_size=len(train), shuffle=False)
    test_dl_all = DataLoader(test, batch_size=len(test), shuffle=False)
    return train_dl, test_dl, train_dl_all, test_dl_all

train_dl, test_dl,  train_dl_all, test_dl_all = prepare_data_flat(PATH_TRAIN, PATH_TEST)
```

1.1 Visualize the data

```
from IPython.display import display
def visualize_data(path):
    df = pd.read_csv(path, header=0)
    display(df)

def visualize_dataset(train_dl, test_dl):
    print(f"Quantidade de casos de Treino:{len(train_dl.dataset)}")
    print(f"Quantidade de casos de Teste:{len(test_dl.dataset)}")
    x, y = next(iter(train_dl))
    print(f"Shape tensor batch casos treino, input: {x.shape}, output: {y.shape}")
    x, y = next(iter(test_dl))
    print(f"Shape tensor batch casos test, input: {x.shape}, output: {y.shape}")
    print(y)

visualize_data(PATH_TRAIN)
visualize_dataset(train_dl, test_dl)
```

1.1 Visualize the data

	5	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	...	0.608	0.609	0.610	0.611	0.612	0.613	0.614	0.615	0.616	0.617
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	9	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	2	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
59994	8	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
59995	3	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
59996	5	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
59997	6	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
59998	8	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

59999 rows × 785 columns

```
Quantidade de casos de Treino:59999
Quantidade de casos de Teste:9999
Shape tensor batch casos treino, input: torch.Size([32, 784]), output: torch.Size([32])
Shape tensor batch casos test, input: torch.Size([32, 784]), output: torch.Size([32])
tensor([0, 4, 7, 6, 6, 8, 0, 8, 6, 6, 7, 3, 2, 3, 9, 8, 6, 2, 7, 1, 7, 0, 9, 1,
        5, 8, 1, 3, 0, 8, 6, 6])
```

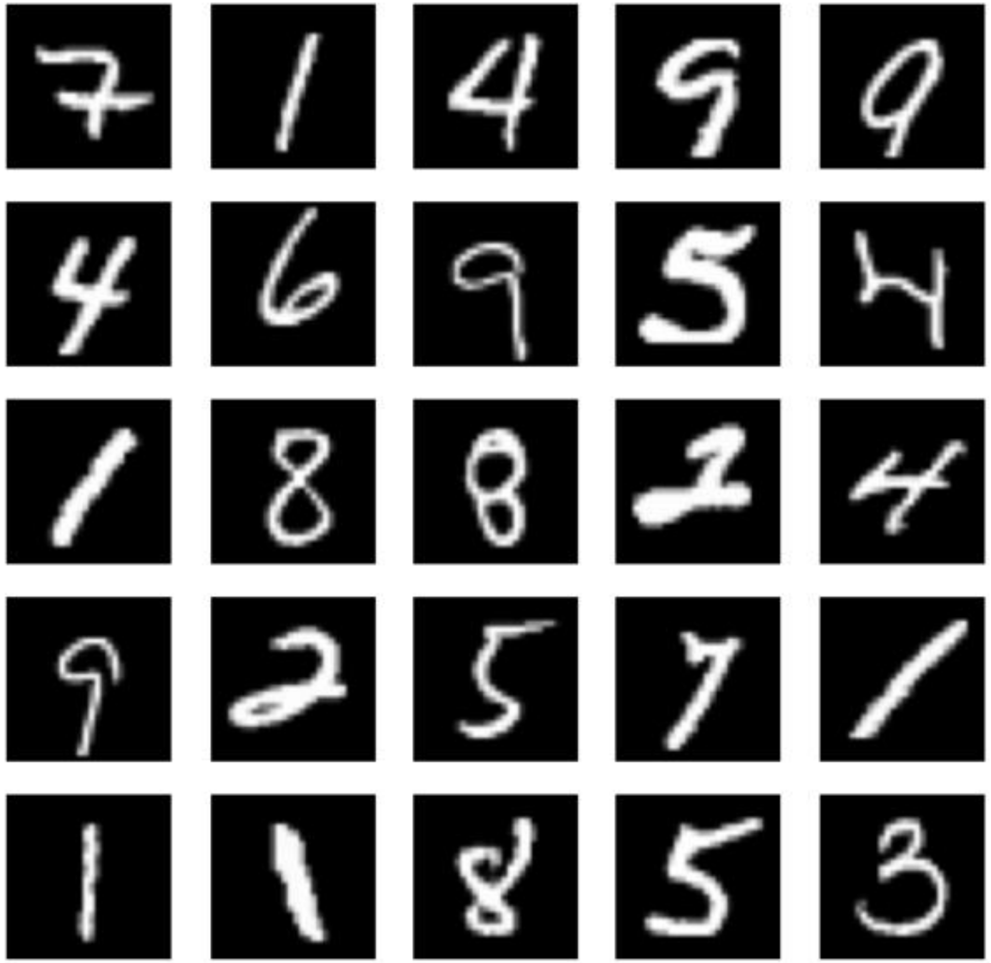
1.1 Visualize the data

```
def visualize_mnist_images_flat(dl):
    i, (inputs, targets) = next(enumerate(dl))
    print(inputs.shape)
    inputs = inputs.reshape(len(inputs), 1, 28, 28)
    print(inputs.shape)
    plt.figure(figsize=(8,8))
    for i in range(25):
        plt.subplot(5, 5, i+1)
        plt.axis('off')
        plt.grid(b=None)
        plt.imshow(inputs[i][0], cmap='gray')
    plt.show()

visualize_mnist_images_flat(train_dl)
```

1.1 Visualize the data

```
def visualize_mnist_images_flat(dl):
    i, (inputs, targets) = next(enumerate(dl))
    print(inputs.shape)
    inputs = inputs.reshape(len(inputs), 1, 28, 28)
    print(inputs.shape)
    plt.figure(figsize=(8,8))
    for i in range(25):
        plt.subplot(5, 5, i+1)
        plt.axis('off')
        plt.grid(b=None)
        plt.imshow(inputs[i][0], cmap='gray')
    plt.show()

visualize_mnist_images_flat(train_dl)
```

```
torch.Size([32, 784])
torch.Size([32, 1, 28, 28])
```



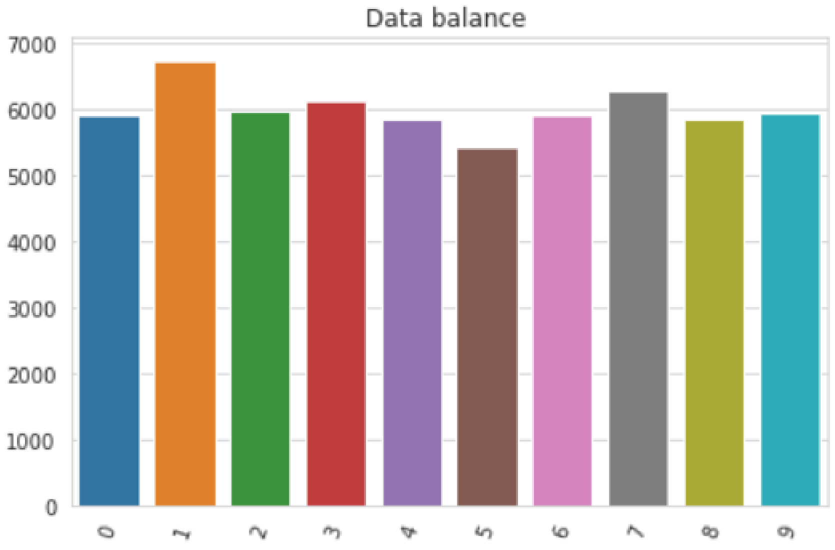
1.2 Verify dataset balancing

```
import seaborn as sns
import matplotlib.pyplot as plt
def visualize_holdout_balance(dl):
    _, labels = next(iter(dl))
    sns.set_style('whitegrid')
    print("casos:",len(labels))
    x, y = np.unique(labels, return_counts=True)
    x=[str(n) for n in x]
    print(x)
    print(y)
    print(np.sum(y))
    grafico=sns.barplot(x, y)
    grafico.set_title('Data balance ')
    plt.xticks(rotation=70)
    plt.show()
```

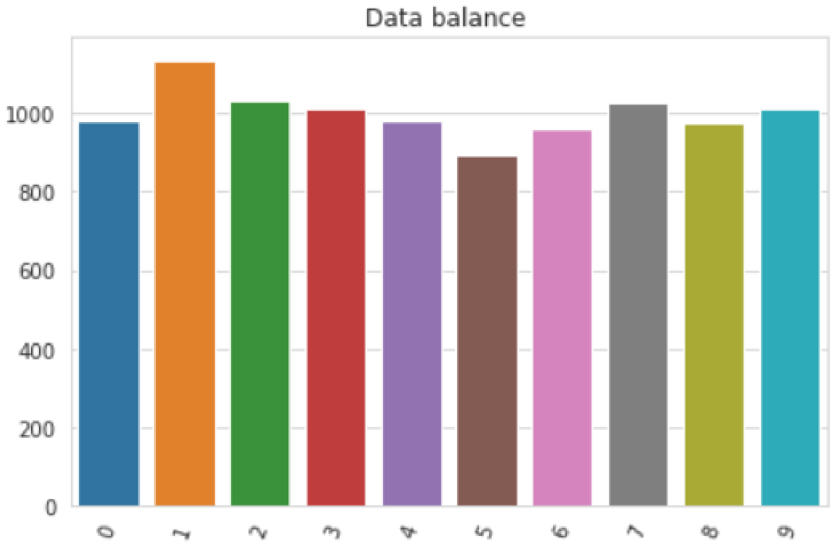
```
print("-----casos_treino------")
visualize_holdout_balance(train_dl_all)
print("-----casos_teste------")
visualize_holdout_balance(test_dl_all)
```

1.2 Verify dataset balancing

```
-----casos_treino------
casos: 59999
['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
[5923 6742 5958 6131 5842 5420 5918 6265 5851 5949]
59999
```




```
-----casos_teste------
casos: 9999
['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
[ 980 1135 1032 1010  982  892  958 1027  974 1009]
9999
```



2. Define the model

```
from torchinfo import summary

class MLP(Module):
    def __init__(self, n_inputs):
        ...
        self.hidden3 = Linear(20, 10)
        xavier_uniform_(self.hidden3.weight)
        self.act3 = Softmax()

    def forward(self, X):
        ...

model = MLP(784)
print(summary(model, input_size=(BATCH_SIZE, 784), verbose=0))
model.to(device)
```

2. Define the model

```
from torchinfo import summary

class MLP(Module):
    def __init__(self, n_inputs):
        ...
        self.hidden3 = Linear(20, 10)
        xavier_uniform_(self.hidden3.weight)
        self.act3 = Softmax()

    def forward(self, X):
        ...

model = MLP(784)
print(summary(model, input_size=(BATCH_SIZE, 784), verbose=0))
model.to(device)
```

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
├─Linear: 1-1                            [32, 20]                  15,700
├─ReLU: 1-2                              [32, 20]                  --
├─Linear: 1-3                            [32, 10]                  210
├─Softmax: 1-4                           [32, 10]                  --
==========================================================================================
Total params: 15,910
Trainable params: 15,910
Non-trainable params: 0
Total mult-adds (M): 0.51
==========================================================================================
Input size (MB): 0.10
Forward/backward pass size (MB): 0.01
Params size (MB): 0.06
Estimated Total Size (MB): 0.17
==========================================================================================
```

```
MLP(
  (hidden1): Linear(in_features=784, out_features=20, bias=True)
  (act1): ReLU()
  (hidden2): Linear(in_features=20, out_features=20, bias=True)
  (act2): ReLU()
  (hidden3): Linear(in_features=20, out_features=10, bias=True)
  (act3): Softmax(dim=1)
)
```

3. Train the model

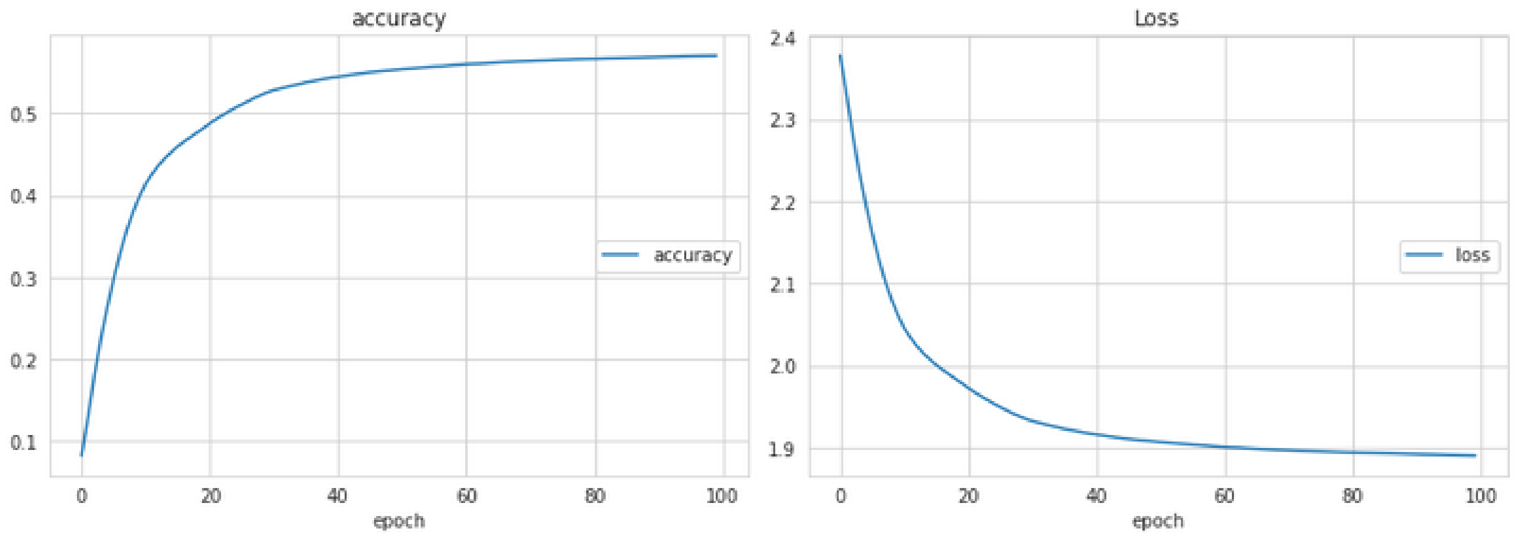

```
from livelossplot import PlotLosses

EPOCHS = 100
LEARNING_RATE = 0.001
```

3. Train the model


```
def train_model(train_dl, model):
    liveloss = PlotLosses()
    criterion = CrossEntropyLoss()
    optimizer = Adam(model.parameters(), lr=LEARNING_RATE)
    for epoch in range(EPOCHS):
        logs = {}
        epoch_loss  = 0
        epoch_acc  = 0
        for i, (inputs, labels) in enumerate(train_dl):
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            acc = accuracy_score(labels.numpy(), np.argmax(outputs.detach().numpy(), axis=1))
            loss.backward()
            optimizer.step()
            epoch_loss += loss.item()
            epoch_acc += acc.item()
        (...)
```

```
        (...)
        print(f'Epoch {epoch:03}:
        | Loss: {epoch_loss/len(train_dl):.5f}
        | Acc: {epoch_acc/len(train_dl):.3f}')
        logs['loss'] = epoch_loss
        logs['accuracy'] = epoch_acc/len(train_dl)
        liveloss.update(logs)
        liveloss.send()

train_model(train_dl_all, model)
```

3. Train the model




```
accuracy
        accuracy                 (min:    0.082, max:    0.571, cur:    0.571)
Loss
        loss                     (min:    1.890, max:    2.378, cur:    1.890)
```

4. Evaluate the model


```
def evaluate_model(test_dl, model):
    predictions = list()
    actual_values = list()
    for i, (inputs, labels) in enumerate(test_dl):
        yprev = model(inputs)
        yprev = yprev.detach().numpy()
        actual = labels.numpy()
        yprev = np.argmax(yprev, axis=1)
        actual = actual.reshape((len(actual), 1))
        yprev = yprev.reshape((len(yprev), 1))
        predictions.append(yprev)
        actual_values.append(actual)
    predictions, actual_values = np.vstack(predictions), np.vstack(actual_values)
    return actual_values, predictions
```

4. Evaluate the model

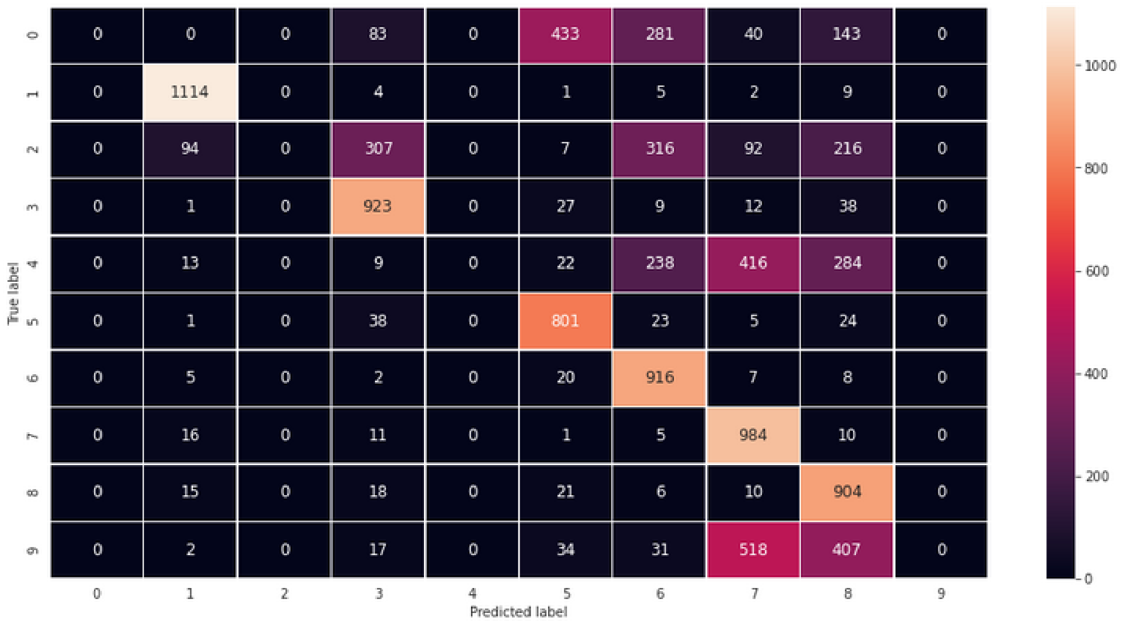

```
def display_confusion_matrix(cm,list_classes):
    plt.figure(figsize = (16,8))
    sns.heatmap(cm,annot=True,xticklabels=list_classes,yticklabels=list_classes, annot_kws={"size": 12},
                fmt='g', linewidths=.5)
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show()

actual_values, predictions = evaluate_model(test_dl, model)

acertou=0
falhou = 0
for r,p in zip(actual_values, predictions):
    if r==p: acertou+=1
    else: falhou+=1
```

4. Evaluate the model


```
acc = accuracy_score(actual_values, predictions)
print(f'Accuracy: {acc:0.3f}\n')
print(f'acertou:{acertou} falhou:{falhou}')

print(classification_report(actual_values, predictions))
cr =classification_report(actual_values, predictions, output_dict=True)
list_classes=list(cr.keys())[0:10]
cm = confusion_matrix(actual_values, predictions)
print (cm)
display_confusion_matrix(cm,list_classes)
```

4. Evaluate the model


```
Accuracy: 0.564

acertou:5642 falhou:4357
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       980
           1       0.88      0.98      0.93      1135
           2       0.00      0.00      0.00      1032
           3       0.65      0.91      0.76      1010
           4       0.00      0.00      0.00       982
           5       0.59      0.90      0.71       892
           6       0.50      0.96      0.66       958
           7       0.47      0.96      0.63      1027
           8       0.44      0.93      0.60       974
           9       0.00      0.00      0.00      1009

    accuracy                           0.56      9999
   macro avg       0.35      0.56      0.43      9999
weighted avg       0.36      0.56      0.43      9999

[[   0    0    0   83    0  433  281   40  143    0]
 [   0 1114    0    4    0    1    5    2    9    0]
 [   0   94    0  307    0    7  316   92  216    0]
 [   0    1    0  923    0   27    9   12   38    0]
 [   0   13    0    9    0   22  238  416  284    0]
 [   0    1    0   38    0  801   23    5   24    0]
 [   0    5    0    2    0   20  916    7    8    0]
 [   0   16    0   11    0    1    5  984   10    0]
 [   0   15    0   18    0   21    6   10  904    0]
 [   0    2    0   17    0   34   31  518  407    0]]
```



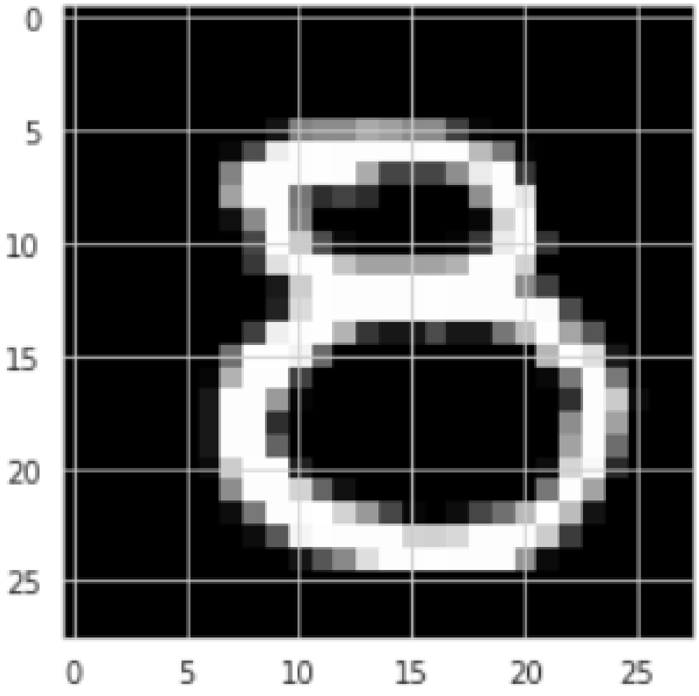
5. Use the model

```
def make_prediction(model, img_list, idx):
    print(img_list.shape)
    print(img_list.dtype)
    img_list = img_list.to(device)
    prediction = model(img_list).detach().numpy()[idx].argmax()
    print("predict:",prediction)
    img = img_list[idx].reshape(1,28, 28)
    plt.imshow(img[0], cmap=plt.get_cmap('gray'))
    plt.show()

_, (inputs, targets) = next(enumerate(test_dl))
make_prediction(model,inputs, 10)
```

5. Use the model

```
def make_prediction(model, img_list, idx):
    print(img_list.shape)
    print(img_list.dtype)
    img_list = img_list.to(device)
    prediction = model(img_list).detach().numpy()[idx].argmax()
    print("predict:",prediction)
    img = img_list[idx].reshape(1,28, 28)
    plt.imshow(img[0], cmap=plt.get_cmap('gray'))
    plt.show()

_, (inputs, targets) = next(enumerate(test_dl))
make_prediction(model,inputs, 10)
```

```
torch.Size([32, 784])
torch.float32
predict: 8
```



MLP for multiclass (binary) image classification

Fashion-MNIST Dataset

- Dataset of clothing images
- Each image has 28x28 pixels
- Contains 70 000 images of which 60 000 are for training and 10 000 for testing
- Contains 10 classes and 785 attributes (the 1st is the class to classify the type of clothing and the rest are values of the pixels of the images)
- The 10 classes are t-shirt/top, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag and ankle boot

Exercise 2

- Apply the same model to fashion-MNIST dataset, improve and present the best value

Exercise 3

epochs		
batch size		
lesrning rate		
size splits	test: train:	test: train:
layers + activation functions		
loss function		
optimization function		
accuracy		

1.2 Verify dataset balancing

```
import seaborn as sns
import matplotlib.pyplot as plt

def output_label(label):
    output_mapping = { 0: "T-shirt/Top", 1: "Trouser", 2: "Pullover",
                       3: "Dress",       4: "Coat",    5: "Sandal",
                       6: "Shirt",       7: "Sneaker", 8: "Bag",
                       9: "Ankle Boot" }
    output_mapping2 = { 0: "0", 1: "1", 2: "2", 3: "3", 4: "4",
                       5: "5", 6: "6", 7: "7", 8: "8", 9: "9"}
    input = (label.item() if type(label) == torch.Tensor else label)
    return output_mapping[input]
```

1.2 Verify dataset balancing

```
def visualize_holdout_balance(dl):
    _, labels = next(iter(dl))
    output_mapping = { 0: "T-shirt/Top", 1: "Trouser", 2: "Pullover",
                       3: "Dress",       4: "Coat",    5: "Sandal",
                       6: "Shirt",       7: "Sneaker", 8: "Bag",
                       9: "Ankle Boot" }
    sns.set_style('whitegrid')
    print("casos:",len(labels))
    x, y = np.unique(output_mapping[labels], return_counts=True)
    x=[str(n) for n in x]
    print(x)
    print(y)
    print(np.sum(y))
    grafico=sns.barplot(x, y)
    grafico.set_title('Data balance ')
    plt.xticks(rotation=70)
    plt.tight_layout()
    plt.show()
```

```
print("-----casos_treino------")
visualize_holdout_balance(train_dl_all)
print("-----casos_teste------")
visualize_holdout_balance(test_dl_all)
```