In [1]:
```
"""
import things required
"""
import numpy as np
import matplotlib.pyplot as plt
import cv2
import os
```

In [2]:
```
"""
Helper functions to load data
"""
def load_mnist_dataset(dataset, path):
    """
    returns samples and lables specified by path and dataset params

    loop through each label and append image to X and label to y
    """
    labels = os.listdir(os.path.join(path, dataset))

    X = []
    y = []

    for label in labels:
        image_counter = 0
        for file in os.listdir(os.path.join(path, dataset, label)):
            image = cv2.imread(os.path.join(path, dataset, label, file), cv2.IMREAD_UN
            X.append(image)
            y.append(label)

    return np.array(X), np.array(y).astype('uint8')

def create_data_mnist(path):
    """
    returns train X, y and test X and y
    """
    X, y = load_mnist_dataset('train', path)
    X_test, y_test = load_mnist_dataset('test', path)

    return X, y, X_test, y_test
```

In [3]:
```
"""
get train and test data
scale data to be in range of -1 to 1 i.e. centered around 0
"""
X, y, X_test, y_test = create_data_mnist('fashion_mnist_images')

# Scale features
X = (X.astype(np.float32) - 127.5) / 127.5
X_test = (X_test.astype(np.float32) - 127.5) / 127.5

# print(X.min(), X.max())
# print(X.shape)
```

In [5]:
```
print(X.shape)
print(y.shape)
```

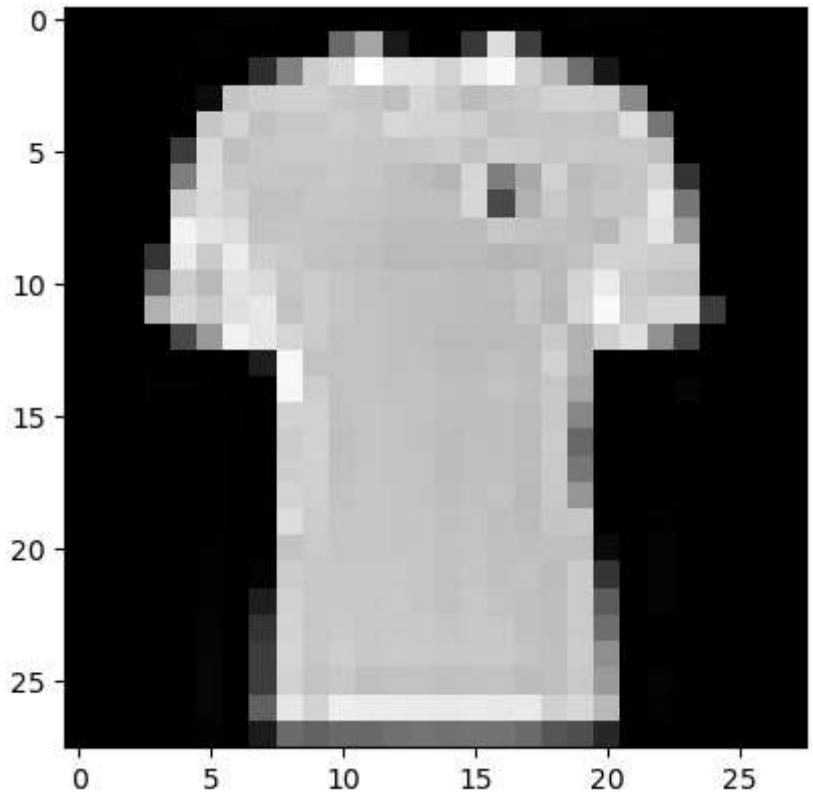
```
(60000, 28, 28)
(60000,)
```

In [6]:
```
"""
pixel size - 28 x 28 - 784 features
images are reshaped to be next to each other
"""
# Reshape to vectors
X = X.reshape(X.shape[0], -1)
X_test = X_test.reshape(X_test.shape[0], -1)
X.shape[1]
```

Out[6]: 784

In [7]:
```
"""
shuffle the keys to remove biase that might be caused towards one label
"""
keys = np.array(range(X.shape[0]))
print(keys[:10])
np.random.shuffle(keys)
X = X[keys]
y = y[keys]
plt.imshow((X[4].reshape(28, 28)), cmap='gray')
plt.show()
```

```
[0 1 2 3 4 5 6 7 8 9]
```



In [8]:
```
class Layer_Dense:
    def __init__(self, n_inputs, n_neurons,
                 weight_regularizer_l1=0, weight_regularizer_l2=0,
                 bias_regularizer_l1=0, bias_regularizer_l2=0):
        """
        n_inputs represents the no of features
        n_neurons represents the no of neurons we want in this particular layer
```

```
        weights are inputs x no of neurons - helps avoid transpose operation in dot pr
        weights range between -1 to +1 so tht they are close to each other and NN does

        biases is set to zero initially and if we encounter error where the entore out
        NN is zero we cqan intialize it to some value to avoid dead ANN

        rest four parameters referes to lambda that will be used for
        L1 and L2 regularization
        """
        self.weights = 0.01 * np.random.randn(n_inputs, n_neurons)
        self.biases = np.zeros((1, n_neurons))

        # Set regularization strength
        self.weight_regularizer_l1 = weight_regularizer_l1
        self.weight_regularizer_l2 = weight_regularizer_l2
        self.bias_regularizer_l1 = bias_regularizer_l1
        self.bias_regularizer_l2 = bias_regularizer_l2

    def forward(self, inputs):
        self.inputs = inputs
        self.output = np.dot(inputs, self.weights) + self.biases

    def backward(self, dvalues):
        # Gradients on parameters
        self.dweights = np.dot(self.inputs.T, dvalues)
        self.dbiases = np.sum(dvalues, axis=0, keepdims=True)

        # Gradients on regularization
        # L1 on weights
        if self.weight_regularizer_l1 > 0:
            dL1 = np.ones_like(self.weights)
            dL1[self.weights < 0] = -1
            self.dweights += self.weight_regularizer_l1 * dL1

        # L2 on weights
        if self.weight_regularizer_l2 > 0:
            self.dweights += 2 * self.weight_regularizer_l2 * \
            self.weights

        # L1 on biases
        if self.bias_regularizer_l1 > 0:
            dL1 = np.ones_like(self.biases)
            dL1[self.biases < 0] = -1
            self.dbiases += self.bias_regularizer_l1 * dL1

        # L2 on biases
        if self.bias_regularizer_l2 > 0:
            self.dbiases += 2 * self.bias_regularizer_l2 * \
            self.biases

        # Gradient on values
        self.dinputs = np.dot(dvalues, self.weights.T)
```

In [9]:

```
class Activation_ReLU:
    def forward(self, inputs):
        self.inputs = inputs
        self.output = np.maximum(0, inputs)
```

```
    def backward(self, dvalues):
        self.dinputs = dvalues.copy()
        # input negative - gradient zero
        self.dinputs[self.inputs <= 0] = 0

class Activation_Softmax:
    def forward(self, inputs):
        self.inputs = inputs

        exp_values = np.exp(inputs - np.max(inputs, axis=1, keepdims=True))
        probabilities = exp_values / np.sum(exp_values, axis=1, keepdims=True)
        self.output = probabilities

    def backward(self, dvalues):
        # Create uninitialized array
        self.dinputs = np.empty_like(dvalues)

        # Enumerate outputs and gradients
        for index, (single_output, single_dvalues) in enumerate(zip(self.output, dvalu
            # Flatten output array
            single_output = single_output.reshape(-1, 1)
            # Calculate Jacobian matrix of the output and
            jacobian_matrix = np.diagflat(single_output) - np.dot(single_output, singl
            # Calculate sample-wise gradient
            # and add it to the array of sample gradients
            self.dinputs[index] = np.dot(jacobian_matrix, single_dvalues)
```

In [10]:

```
class Loss:
    def calculate(self, output, y):
        sample_losses = self.forward(output, y)
        mean_loss = np.mean(sample_losses)
        return mean_loss

    def regularization_loss(self, layer):

        regularization_loss = 0

        # L1 regularization - weights
        # calculate only when factor greater than 0
        if layer.weight_regularizer_l1 > 0:
            regularization_loss += layer.weight_regularizer_l1 * np.sum(np.abs(layer.w

        # L2 regularization - weights
        if layer.weight_regularizer_l2 > 0:
            regularization_loss += layer.weight_regularizer_l2 * np.sum(layer.weights

        # L1 regularization - biases
        # calculate only when factor greater than 0
        if layer.bias_regularizer_l1 > 0:
            regularization_loss += layer.bias_regularizer_l1 * np.sum(np.abs(layer.bia

        # L2 regularization - biases
        if layer.bias_regularizer_l2 > 0:
            regularization_loss += layer.bias_regularizer_l2 * np.sum(layer.biases * l

        return regularization_loss

class Loss_CategoricalCrossentropy(Loss):
    def forward(self, y_pred, y_true):
```

```
        samples = len(y_pred)

        # account for zero values -log(0) = inf and then remove biasis caused by its i
        y_pred_clipped = np.clip(y_pred, 1e-7, 1-1e-7)

        if len(y_true.shape) == 1:
            # y_true is in the form of [1, 0, 1, 1 ....]
            correct_confidences = y_pred_clipped[range(samples), y_true]

        elif len(y_true.shape) == 2:
            # y_true is in the form of matric [[0, 1, 0], [1, 0, 0], [0, 1, 0], [0, 1,
            correct_confidences = np.sum(y_pred_clipped * y_true, axis=1)

        negative_log_likelihood = - np.log(correct_confidences)

        return negative_log_likelihood

    def backward(self, dvalues, y_true):

        samples = len(dvalues)
        labels = len(dvalues[0])
        # convert to hot vector if not
        if len(y_true.shape) == 1:
            y_true = np.eye(labels)[y_true]
        # Calculate gradient
        self.dinputs = -y_true / dvalues
        # Normalize gradient
        self.dinputs = self.dinputs / samples

class Activation_Softmax_Loss_CategoricalCrossentropy():
    def __init__(self):
        self.activation = Activation_Softmax()
        self.loss = Loss_CategoricalCrossentropy()

    def forward(self, inputs, y_true):
        # Output layer's activation function
        self.activation.forward(inputs)
        # Set the output
        self.output = self.activation.output
        # Calculate and return loss value
        return self.loss.calculate(self.output, y_true)

    def backward(self, dvalues, y_true):
        # Number of samples
        samples = len(dvalues)
        # If labels are one-hot encoded,
        # turn them into discrete values
        if len(y_true.shape) == 2:
            y_true = np.argmax(y_true, axis=1)
        # Copy so we can safely modify
        self.dinputs = dvalues.copy()
        # Calculate gradient
        self.dinputs[range(samples), y_true] -= 1
        # Normalize gradient
        self.dinputs = self.dinputs / samples
```

In [ ]:

```
class Optimizer_SGD:
    """
    Vanilla option - SGD
```

```
    Momentum option - if specified else default 0
    """
    def __init__(self, learning_rate=1.0, decay=0, momentum=0):
        self.learning_rate = learning_rate
        self.current_learning_rate = learning_rate
        self.decay = decay
        self.iterations = 0
        self.momentum = momentum

    def pre_update_params(self):
        if self.decay:
            self.current_learning_rate = self.learning_rate * (1. / (1. + self.decay *

    # Update parameters
    def update_params(self, layer):
        # If we use momentum
        if self.momentum:

            # create momentum array if not present along with biases
            if not hasattr(layer, 'weight_momentums'):
                layer.weight_momentums = np.zeros_like(layer.weights)
                layer.bias_momentums = np.zeros_like(layer.biases)

            weight_updates = self.momentum * layer.weight_momentums - self.current_lea
            layer.weight_momentums = weight_updates

            bias_updates = self.momentum * layer.bias_momentums - self.current_learnir
            layer.bias_momentums = bias_updates

        # Vanilla SGD updates (as before momentum update)
        else:
            weight_updates = -self.current_learning_rate * layer.dweights
            bias_updates = -self.current_learning_rate * layer.dbiases

        # Update weights and biases using either vanilla or momentum update
        layer.weights += weight_updates
        layer.biases += bias_updates

    def post_update_params(self):
        self.iterations += 1
```

In [12]:

```
"""
Features = 784
Samples = 60,000
class labels = 10 (0 - 9)

2 hidden layes - ReLU    - 128 neurons each
1 output layer - Softmax - 10 output neurons

Optimizer - SGD
learning rate decay - 0.01
l2 regularization - lambda - 5e-4 (0.0005)
"""

dense1 = Layer_Dense(X.shape[1], 128, weight_regularizer_l2=5e-4, bias_regularizer_l2=
activation1 = Activation_ReLU()
dense2 = Layer_Dense(128, 128)
activation2 = Activation_ReLU()
dense3 = Layer_Dense(128, 10)
```

```
loss_activation = Activation_Softmax_Loss_CategoricalCrossentropy()

optimizer = Optimizer_SGD(decay=0.01)

accuracies_sgd = []
losses_sgd = []
learning_rate = []
```

In [13]:

```
for epoch in range(1100):
    dense1.forward(X)
    activation1.forward(dense1.output)
    dense2.forward(activation1.output)
    activation2.forward(dense2.output)
    dense3.forward(activation2.output)
    data_loss = loss_activation.forward(dense3.output, y)

    regularization_loss = loss_activation.loss.regularization_loss(dense1) + loss_acti
        loss_activation.loss.regularization_loss(dense3)

    loss = data_loss + regularization_loss

    predictions = np.argmax(loss_activation.output, axis=1)
    if len(y.shape) == 2:
        y = np.argmax(y, axis=1)
    accuracy = np.mean(predictions==y)


    if not epoch % 10:
        accuracies_sgd.append(accuracy)
        losses_sgd.append(loss)
        learning_rate.append(optimizer.current_learning_rate)
        print(f'epoch: {epoch}, ' + f'acc: {accuracy:.3f}, ' + f'loss: {loss:.3f}, ' +
              + f'Rr: {regularization_loss}')

    loss_activation.backward(loss_activation.output, y)
    dense3.backward(loss_activation.dinputs)
    activation2.backward(dense3.dinputs)
    dense2.backward(activation2.dinputs)
    activation1.backward(dense2.dinputs)
    dense1.backward(activation1.dinputs)

    optimizer.pre_update_params()
    optimizer.update_params(dense1)
    optimizer.update_params(dense2)
    optimizer.update_params(dense3)
    optimizer.post_update_params()
```

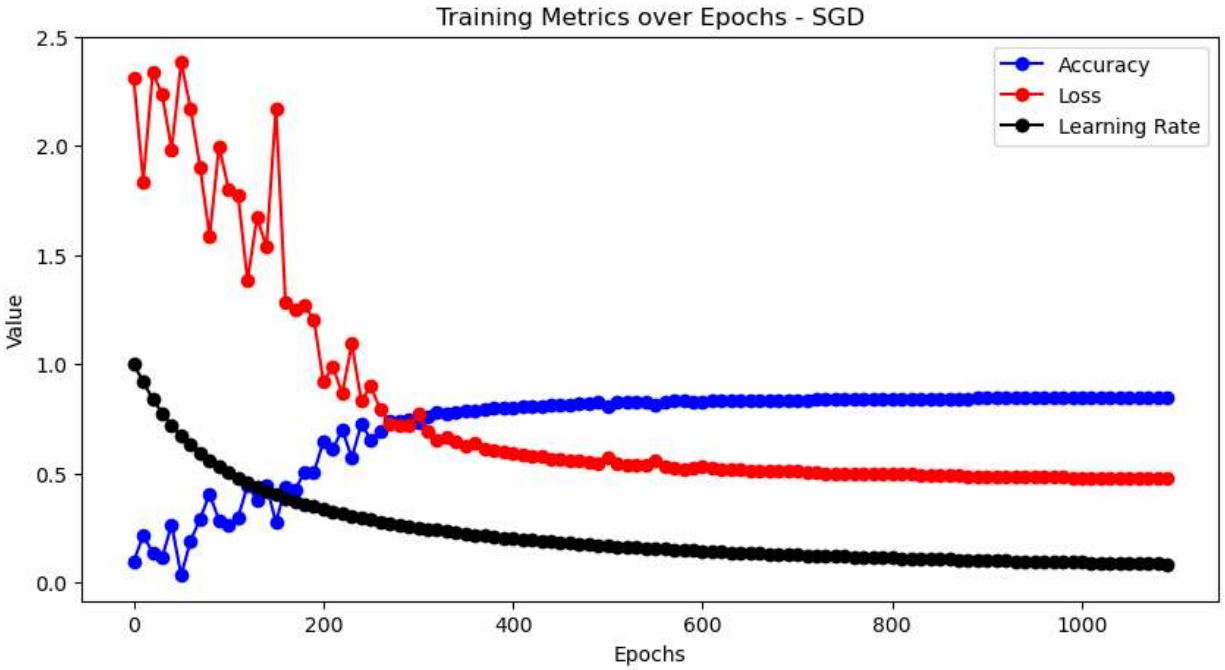
```
epoch: 0, acc: 0.096, loss: 2.308, lr: 1.0Rr: 0.005013977771864498
epoch: 10, acc: 0.215, loss: 1.837, lr: 0.9174311926605504Rr: 0.005293221293737015
epoch: 20, acc: 0.136, loss: 2.335, lr: 0.8403361344537815Rr: 0.04673865016803641
epoch: 30, acc: 0.115, loss: 2.235, lr: 0.7751937984496123Rr: 0.0474798288857692
epoch: 40, acc: 0.264, loss: 1.982, lr: 0.7194244604316546Rr: 0.04845644284228443
epoch: 50, acc: 0.032, loss: 2.383, lr: 0.6711409395973155Rr: 0.054531087131002756
epoch: 60, acc: 0.191, loss: 2.169, lr: 0.628930817610063Rr: 0.058298930333085
epoch: 70, acc: 0.290, loss: 1.902, lr: 0.591715976331361Rr: 0.06091108234272053
epoch: 80, acc: 0.404, loss: 1.584, lr: 0.5586592178770949Rr: 0.06103500102755085
epoch: 90, acc: 0.284, loss: 1.997, lr: 0.5291005291005291Rr: 0.06156542051235255
epoch: 100, acc: 0.260, loss: 1.802, lr: 0.5025125628140703Rr: 0.06244491644395994
epoch: 110, acc: 0.296, loss: 1.776, lr: 0.4784688995215311Rr: 0.062347514228712556
epoch: 120, acc: 0.443, loss: 1.382, lr: 0.4566210045662101Rr: 0.06184696058734526
epoch: 130, acc: 0.377, loss: 1.670, lr: 0.4366812227074236Rr: 0.06137280013470734
epoch: 140, acc: 0.444, loss: 1.536, lr: 0.41841004184100417Rr: 0.061026834306485166
epoch: 150, acc: 0.274, loss: 2.170, lr: 0.4016064257028112Rr: 0.06323172384107752
epoch: 160, acc: 0.437, loss: 1.282, lr: 0.3861003861003861Rr: 0.06330501393918851
epoch: 170, acc: 0.422, loss: 1.249, lr: 0.3717472118959108Rr: 0.06292377386389489
epoch: 180, acc: 0.504, loss: 1.267, lr: 0.35842293906810035Rr: 0.0628429955397116
epoch: 190, acc: 0.504, loss: 1.204, lr: 0.3460207612456747Rr: 0.06270819805043129
epoch: 200, acc: 0.645, loss: 0.920, lr: 0.33444816053511706Rr: 0.06268877794827858
epoch: 210, acc: 0.613, loss: 0.986, lr: 0.3236245954692557Rr: 0.0627473869913912
epoch: 220, acc: 0.698, loss: 0.864, lr: 0.31347962382445144Rr: 0.06274600823106806
epoch: 230, acc: 0.570, loss: 1.094, lr: 0.303951367781155Rr: 0.06263656733423358
epoch: 240, acc: 0.724, loss: 0.830, lr: 0.2949852507374631Rr: 0.06330399699887226
epoch: 250, acc: 0.655, loss: 0.898, lr: 0.2865329512893982Rr: 0.06322070313499537
epoch: 260, acc: 0.692, loss: 0.792, lr: 0.2785515320334262Rr: 0.06318213180790545
epoch: 270, acc: 0.738, loss: 0.727, lr: 0.2710027100271003Rr: 0.06303626100208753
epoch: 280, acc: 0.737, loss: 0.719, lr: 0.2638522427440633Rr: 0.06286165327747592
epoch: 290, acc: 0.745, loss: 0.715, lr: 0.2570694087403599Rr: 0.06268488233834218
epoch: 300, acc: 0.730, loss: 0.771, lr: 0.2506265664160401Rr: 0.06256858738732937
epoch: 310, acc: 0.757, loss: 0.691, lr: 0.24449877750611249Rr: 0.06244367626578982
epoch: 320, acc: 0.777, loss: 0.655, lr: 0.23866348448687355Rr: 0.06230328555821758
epoch: 330, acc: 0.772, loss: 0.662, lr: 0.2331002331002331Rr: 0.06216682391738058
epoch: 340, acc: 0.782, loss: 0.643, lr: 0.2277904328018223Rr: 0.062016140057243097
epoch: 350, acc: 0.789, loss: 0.628, lr: 0.22271714922048996Rr: 0.061861290971575264
epoch: 360, acc: 0.784, loss: 0.635, lr: 0.2178649237472767Rr: 0.061721987516224784
epoch: 370, acc: 0.795, loss: 0.610, lr: 0.21321961620469085Rr: 0.06158392645501245
epoch: 380, acc: 0.798, loss: 0.604, lr: 0.20876826722338204Rr: 0.06143068306088759
epoch: 390, acc: 0.799, loss: 0.600, lr: 0.20449897750511245Rr: 0.061281684545059195
epoch: 400, acc: 0.802, loss: 0.593, lr: 0.2004008016032064Rr: 0.061136964685608586
epoch: 410, acc: 0.809, loss: 0.581, lr: 0.1964636542396858Rr: 0.06099000713993547
epoch: 420, acc: 0.807, loss: 0.580, lr: 0.19267822736030826Rr: 0.06084373711793871
epoch: 430, acc: 0.808, loss: 0.577, lr: 0.1890359168241966Rr: 0.060704230639868406
epoch: 440, acc: 0.813, loss: 0.566, lr: 0.1855287569573284Rr: 0.060561759363820064
epoch: 450, acc: 0.814, loss: 0.563, lr: 0.18214936247723132Rr: 0.06041537854882529
epoch: 460, acc: 0.816, loss: 0.560, lr: 0.17889087656529518Rr: 0.06027082281540782
epoch: 470, acc: 0.817, loss: 0.559, lr: 0.17574692442882248Rr: 0.06013205349882125
epoch: 480, acc: 0.820, loss: 0.553, lr: 0.17271157167530224Rr: 0.05999921052579664
epoch: 490, acc: 0.824, loss: 0.546, lr: 0.16977928692699493Rr: 0.05986440916753919
epoch: 500, acc: 0.806, loss: 0.573, lr: 0.1669449081803005Rr: 0.05973055005807647
epoch: 510, acc: 0.823, loss: 0.545, lr: 0.16420361247947454Rr: 0.059602475530373095
epoch: 520, acc: 0.826, loss: 0.538, lr: 0.16155088852988692Rr: 0.059469428513397544
epoch: 530, acc: 0.826, loss: 0.536, lr: 0.1589825119236884Rr: 0.0593332466644326
epoch: 540, acc: 0.826, loss: 0.536, lr: 0.1564945226917058Rr: 0.05919807306480461
epoch: 550, acc: 0.815, loss: 0.556, lr: 0.15408320493066255Rr: 0.05907423578497166
epoch: 560, acc: 0.826, loss: 0.533, lr: 0.15174506828528073Rr: 0.058977261611299805
epoch: 570, acc: 0.831, loss: 0.522, lr: 0.14947683109118085Rr: 0.058849791777276866
epoch: 580, acc: 0.832, loss: 0.520, lr: 0.1472754050073637Rr: 0.058717920825117446
epoch: 590, acc: 0.829, loss: 0.527, lr: 0.14513788098693758Rr: 0.058588254714291216
```

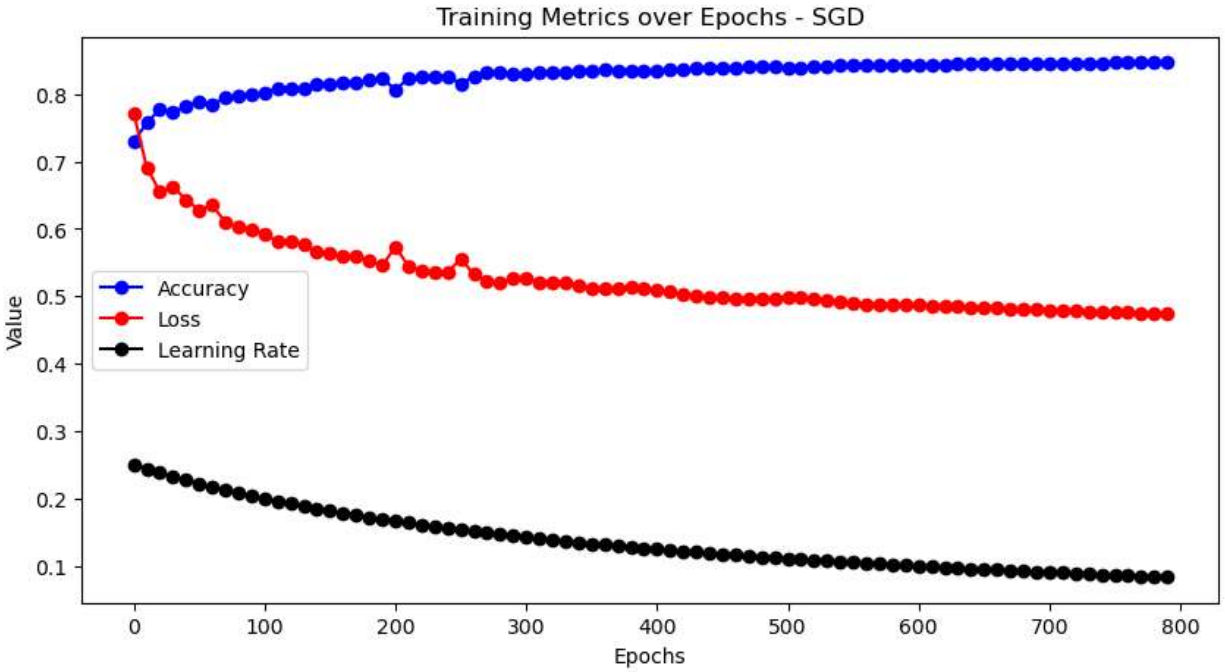
```
epoch: 600, acc: 0.829, loss: 0.527, lr: 0.14306151645207438Rr: 0.058465667280048296
epoch: 610, acc: 0.832, loss: 0.521, lr: 0.14104372355430184Rr: 0.05834607563849975
epoch: 620, acc: 0.832, loss: 0.520, lr: 0.13908205841446453Rr: 0.05822787940905592
epoch: 630, acc: 0.832, loss: 0.520, lr: 0.13717421124828533Rr: 0.0581131494385723
epoch: 640, acc: 0.833, loss: 0.517, lr: 0.13531799729364005Rr: 0.0580000722328097
epoch: 650, acc: 0.835, loss: 0.512, lr: 0.13351134846461948Rr: 0.05788509508759707
epoch: 660, acc: 0.835, loss: 0.511, lr: 0.13175230566534915Rr: 0.057769045249777204
epoch: 670, acc: 0.835, loss: 0.512, lr: 0.13003901170351104Rr: 0.0576542116791727
epoch: 680, acc: 0.834, loss: 0.513, lr: 0.12836970474967907Rr: 0.0575428152855702
epoch: 690, acc: 0.834, loss: 0.512, lr: 0.12674271229404308Rr: 0.05743526960289972
epoch: 700, acc: 0.834, loss: 0.510, lr: 0.1251564455569462Rr: 0.0573304431793102
epoch: 710, acc: 0.835, loss: 0.506, lr: 0.12360939431396786Rr: 0.05722570104362908
epoch: 720, acc: 0.837, loss: 0.502, lr: 0.12210012210012208Rr: 0.057119473098698595
epoch: 730, acc: 0.838, loss: 0.500, lr: 0.12062726176115804Rr: 0.05701282029208169
epoch: 740, acc: 0.838, loss: 0.499, lr: 0.11918951132300357Rr: 0.056906645886608556
epoch: 750, acc: 0.839, loss: 0.497, lr: 0.11778563015312131Rr: 0.05680143324128258
epoch: 760, acc: 0.839, loss: 0.496, lr: 0.11641443538998836Rr: 0.056697394746520476
epoch: 770, acc: 0.840, loss: 0.496, lr: 0.1150747986191024Rr: 0.056594706953099226
epoch: 780, acc: 0.840, loss: 0.496, lr: 0.11376564277588169Rr: 0.05649359332149263
epoch: 790, acc: 0.840, loss: 0.497, lr: 0.11248593925759279Rr: 0.05639465694136939
epoch: 800, acc: 0.839, loss: 0.499, lr: 0.11123470522803114Rr: 0.056298882215286665
epoch: 810, acc: 0.839, loss: 0.499, lr: 0.11001100110011001Rr: 0.05620629109305979
epoch: 820, acc: 0.840, loss: 0.496, lr: 0.1088139281828074Rr: 0.056115367192420794
epoch: 830, acc: 0.841, loss: 0.493, lr: 0.1076426264800861Rr: 0.05602445699321481
epoch: 840, acc: 0.842, loss: 0.491, lr: 0.10649627263045792Rr: 0.05593307717217562
epoch: 850, acc: 0.842, loss: 0.489, lr: 0.1053740779768177Rr: 0.05584153908931848
epoch: 860, acc: 0.842, loss: 0.488, lr: 0.10427528675703858Rr: 0.05575044785597827
epoch: 870, acc: 0.843, loss: 0.487, lr: 0.10319917440660475Rr: 0.055660126060138856
epoch: 880, acc: 0.843, loss: 0.487, lr: 0.10214504596527067Rr: 0.05557079576235481
epoch: 890, acc: 0.843, loss: 0.487, lr: 0.10111223458038422Rr: 0.05548270680771946
epoch: 900, acc: 0.843, loss: 0.487, lr: 0.10010010010010009Rr: 0.05539601900605836
epoch: 910, acc: 0.843, loss: 0.486, lr: 0.09910802775024777Rr: 0.05531065275151828
epoch: 920, acc: 0.844, loss: 0.485, lr: 0.09813542688910697Rr: 0.05522636415299043
epoch: 930, acc: 0.844, loss: 0.485, lr: 0.09718172983479105Rr: 0.05514299624985183
epoch: 940, acc: 0.844, loss: 0.484, lr: 0.09624639076034648Rr: 0.05506043831713899
epoch: 950, acc: 0.844, loss: 0.483, lr: 0.09532888465204957Rr: 0.054978713251017934
epoch: 960, acc: 0.845, loss: 0.482, lr: 0.09442870632672333Rr: 0.054897777460828516
epoch: 970, acc: 0.845, loss: 0.481, lr: 0.09354536950420955Rr: 0.05481759759920185
epoch: 980, acc: 0.845, loss: 0.481, lr: 0.09267840593141798Rr: 0.054738194278115
epoch: 990, acc: 0.845, loss: 0.480, lr: 0.09182736455463728Rr: 0.054659533139755974
epoch: 1000, acc: 0.845, loss: 0.479, lr: 0.09099181073703366Rr: 0.05458161863224329
epoch: 1010, acc: 0.845, loss: 0.479, lr: 0.09017132551848513Rr: 0.054504494436765694
epoch: 1020, acc: 0.845, loss: 0.478, lr: 0.08936550491510277Rr: 0.054428168792544265
epoch: 1030, acc: 0.846, loss: 0.478, lr: 0.08857395925597873Rr: 0.054352662348855975
epoch: 1040, acc: 0.846, loss: 0.477, lr: 0.08779631255487269Rr: 0.05427800622064430
epoch: 1050, acc: 0.846, loss: 0.477, lr: 0.08703220191470844Rr: 0.05420418669212498
epoch: 1060, acc: 0.846, loss: 0.476, lr: 0.08628127696289906Rr: 0.05413109160948577
epoch: 1070, acc: 0.846, loss: 0.476, lr: 0.0855431993156544Rr: 0.05405867467525899
epoch: 1080, acc: 0.847, loss: 0.475, lr: 0.08481764206955046Rr: 0.05398690000759705
epoch: 1090, acc: 0.847, loss: 0.474, lr: 0.08410428931875526Rr: 0.05391577071235945
```

In [14]:

```
epochs = range(0, 1100, 10)
plt.figure(figsize=(10, 5))
plt.plot(epochs, accuracies_sgd, label='Accuracy', marker='o', color="blue")
plt.plot(epochs, losses_sgd, label='Loss', marker='o', color='red')
plt.plot(epochs, learning_rate, label='Learning Rate', marker='o', color='black')
plt.title('Training Metrics over Epochs - SGD')
plt.xlabel('Epochs')
plt.ylabel('Value')
plt.legend()
```

```
plt.savefig('training_metrics_plot.png')
plt.show()

epochs = range(0, 800, 10)
plt.figure(figsize=(10, 5))
plt.plot(epochs, accuracies_sgd[30:], label='Accuracy', marker='o', color="blue")
plt.plot(epochs, losses_sgd[30:], label='Loss', marker='o', color='red')
plt.plot(epochs, learning_rate[30:], label='Learning Rate', marker='o', color='black')
plt.title('Training Metrics over Epochs - SGD')
plt.xlabel('Epochs')
plt.ylabel('Value')
plt.legend()
plt.savefig('training_metrics_plot.png')
plt.show()
```





In [15]:
```
# Validate the model
```

```
dense1.forward(X_test)
activation1.forward(dense1.output)
dense2.forward(activation1.output)
activation2.forward(dense2.output)
dense3.forward(activation2.output)
data_loss = loss_activation.forward(dense3.output, y_test)

predictions = np.argmax(loss_activation.output, axis=1)
if len(y_test.shape) == 2:
    y_test = np.argmax(y, axis=1)
accuracy = np.mean(predictions==y_test)
print(f'validation - SGD, acc: {accuracy:.3f}, loss: {loss:.3f}')
```

```
validation - SGD, acc: 0.833, loss: 0.474
```

In [20]:

```
"""
Features = 784
Samples = 60,000
class labels = 10 (0 - 9)

2 hidden layes - ReLU    - 128 neurons each
1 output layer - Softmax - 10 output neurons

Optimizer - SGD with momentum (0.5)
learning rate decay - 0.01
l2 regularization - lambda - 5e-4 (0.0005)
"""

dense1 = Layer_Dense(X.shape[1], 128, weight_regularizer_l2=5e-4, bias_regularizer_l2=
activation1 = Activation_ReLU()
dense2 = Layer_Dense(128, 128)
activation2 = Activation_ReLU()
dense3 = Layer_Dense(128, 10)
loss_activation = Activation_Softmax_Loss_CategoricalCrossentropy()

optimizer = Optimizer_SGD(decay=0.01, momentum=0.5)

accuracies_mom = []
losses_mom = []
learning_rate = []
```

In [21]:

```
for epoch in range(1100):
    dense1.forward(X)
    activation1.forward(dense1.output)
    dense2.forward(activation1.output)
    activation2.forward(dense2.output)
    dense3.forward(activation2.output)
    data_loss = loss_activation.forward(dense3.output, y)

    regularization_loss = loss_activation.loss.regularization_loss(dense1) + loss_acti
        loss_activation.loss.regularization_loss(dense3)

    loss = data_loss + regularization_loss

    predictions = np.argmax(loss_activation.output, axis=1)
    if len(y.shape) == 2:
        y = np.argmax(y, axis=1)
    accuracy = np.mean(predictions==y)
```

```
    if not epoch % 10:
        accuracies_mom.append(accuracy)
        losses_mom.append(loss)
        learning_rate.append(optimizer.current_learning_rate)
        print(f'epoch: {epoch}, ' + f'acc: {accuracy:.3f}, ' + f'loss: {loss:.3f}, ' +
              + f'Rr: {regularization_loss}')

    loss_activation.backward(loss_activation.output, y)
    dense3.backward(loss_activation.dinputs)
    activation2.backward(dense3.dinputs)
    dense2.backward(activation2.dinputs)
    activation1.backward(dense2.dinputs)
    dense1.backward(activation1.dinputs)

    optimizer.pre_update_params()
    optimizer.update_params(dense1)
    optimizer.update_params(dense2)
    optimizer.update_params(dense3)
    optimizer.post_update_params()
```

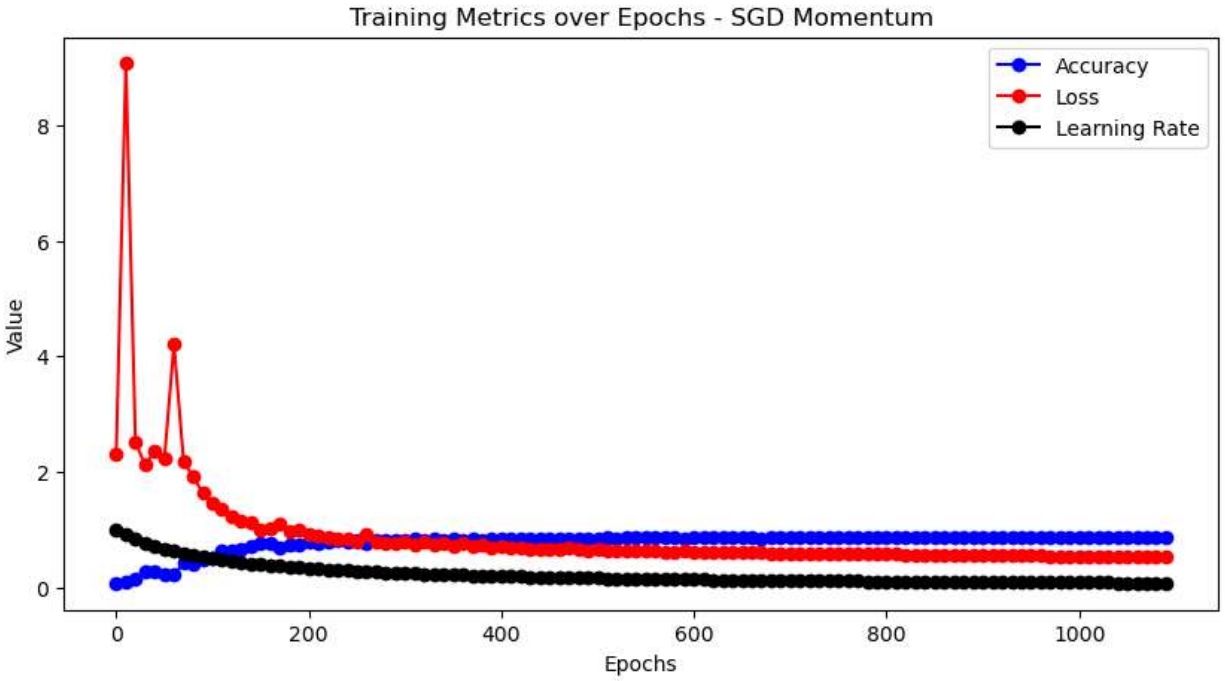
```
epoch: 0, acc: 0.070, loss: 2.308, lr: 1.0Rr: 0.00503343727903207
epoch: 10, acc: 0.102, loss: 9.068, lr: 0.9174311926605504Rr: 0.007474826802391185
epoch: 20, acc: 0.158, loss: 2.516, lr: 0.8403361344537815Rr: 0.27192697559379725
epoch: 30, acc: 0.279, loss: 2.122, lr: 0.7751937984496123Rr: 0.26757800612310556
epoch: 40, acc: 0.272, loss: 2.356, lr: 0.7194244604316546Rr: 0.26561551479816753
epoch: 50, acc: 0.240, loss: 2.225, lr: 0.6711409395973155Rr: 0.2616486037013331
epoch: 60, acc: 0.230, loss: 4.213, lr: 0.628930817610063Rr: 0.2595395807597197
epoch: 70, acc: 0.423, loss: 2.184, lr: 0.591715976331361Rr: 0.40876334154444693
epoch: 80, acc: 0.402, loss: 1.914, lr: 0.5586592178770949Rr: 0.4016385205904229
epoch: 90, acc: 0.488, loss: 1.638, lr: 0.529100529100529Rr: 0.39527351468614047
epoch: 100, acc: 0.502, loss: 1.463, lr: 0.5025125628140703Rr: 0.3878204130123266
epoch: 110, acc: 0.631, loss: 1.361, lr: 0.4784688995215311Rr: 0.3814228475238515
epoch: 120, acc: 0.647, loss: 1.227, lr: 0.4566210045662101Rr: 0.37504088735565233
epoch: 130, acc: 0.676, loss: 1.148, lr: 0.4366812227074236Rr: 0.36889374531970465
epoch: 140, acc: 0.710, loss: 1.124, lr: 0.4184100418410417Rr: 0.36337168187175656
epoch: 150, acc: 0.776, loss: 1.004, lr: 0.4016064257028112Rr: 0.35808043777823434
epoch: 160, acc: 0.761, loss: 1.036, lr: 0.3861003861003861Rr: 0.35305387279779993
epoch: 170, acc: 0.697, loss: 1.115, lr: 0.3717472118959108Rr: 0.3480981791843632
epoch: 180, acc: 0.742, loss: 0.979, lr: 0.35842293906810035Rr: 0.34340422955457034
epoch: 190, acc: 0.750, loss: 1.011, lr: 0.3460207612456747Rr: 0.3392647592514117
epoch: 200, acc: 0.784, loss: 0.916, lr: 0.33444816053511706Rr: 0.3351876274681324
epoch: 210, acc: 0.780, loss: 0.908, lr: 0.3236245954692557Rr: 0.3312183802860451
epoch: 220, acc: 0.799, loss: 0.863, lr: 0.31347962382445144Rr: 0.327274153870548
epoch: 230, acc: 0.810, loss: 0.843, lr: 0.303951367781155Rr: 0.32359019167753106
epoch: 240, acc: 0.806, loss: 0.848, lr: 0.2949852507374631Rr: 0.31997933097325565
epoch: 250, acc: 0.824, loss: 0.800, lr: 0.2865329512893982Rr: 0.31655447801666003
epoch: 260, acc: 0.772, loss: 0.921, lr: 0.2785515320334262Rr: 0.31318410556074927
epoch: 270, acc: 0.830, loss: 0.783, lr: 0.2710027100271003Rr: 0.310150728567035
epoch: 280, acc: 0.831, loss: 0.773, lr: 0.2638522427440633Rr: 0.30699763738602487
epoch: 290, acc: 0.832, loss: 0.770, lr: 0.2570694087403599Rr: 0.30399050153228235
epoch: 300, acc: 0.816, loss: 0.799, lr: 0.2506265664160401Rr: 0.30108325309492445
epoch: 310, acc: 0.840, loss: 0.745, lr: 0.2444987775061124Rr: 0.29833816033192995
epoch: 320, acc: 0.813, loss: 0.796, lr: 0.2386634844868735Rr: 0.29560779799951425
epoch: 330, acc: 0.842, loss: 0.731, lr: 0.233100233100233Rr: 0.29302105604733414
epoch: 340, acc: 0.822, loss: 0.771, lr: 0.227790432801822Rr: 0.2904695733488789
epoch: 350, acc: 0.846, loss: 0.717, lr: 0.2227171492204899Rr: 0.2880391724686652
epoch: 360, acc: 0.826, loss: 0.759, lr: 0.217864923747276Rr: 0.28565436829422114
epoch: 370, acc: 0.848, loss: 0.706, lr: 0.2132196162046908Rr: 0.2833683516787223
epoch: 380, acc: 0.828, loss: 0.747, lr: 0.2087682672233821Rr: 0.28110098905397324
epoch: 390, acc: 0.850, loss: 0.696, lr: 0.2044989775051124Rr: 0.27894877703455734
epoch: 400, acc: 0.839, loss: 0.714, lr: 0.200400801603206Rr: 0.2768103975354487
epoch: 410, acc: 0.853, loss: 0.685, lr: 0.1964636542239685Rr: 0.27476520377417696
epoch: 420, acc: 0.846, loss: 0.699, lr: 0.1926782273603082Rr: 0.27273632896736283
epoch: 430, acc: 0.854, loss: 0.677, lr: 0.189035916824196Rr: 0.27080648261257756
epoch: 440, acc: 0.854, loss: 0.675, lr: 0.1855287569573284Rr: 0.2688844029505018
epoch: 450, acc: 0.854, loss: 0.673, lr: 0.1821493624772313Rr: 0.2670400261857702
epoch: 460, acc: 0.857, loss: 0.663, lr: 0.1788908765652951Rr: 0.2652248471908936
epoch: 470, acc: 0.844, loss: 0.691, lr: 0.1757469244288224Rr: 0.26344370501814346
epoch: 480, acc: 0.858, loss: 0.656, lr: 0.1727115716753022Rr: 0.2617415971846138
epoch: 490, acc: 0.858, loss: 0.654, lr: 0.1697792869269949Rr: 0.2600426286390037
epoch: 500, acc: 0.851, loss: 0.668, lr: 0.166944908180301Rr: 0.25840563360391094
epoch: 510, acc: 0.860, loss: 0.645, lr: 0.1642036124794745Rr: 0.25680063631500005
epoch: 520, acc: 0.859, loss: 0.647, lr: 0.161550888529887Rr: 0.255214599269414
epoch: 530, acc: 0.860, loss: 0.642, lr: 0.158982511923688Rr: 0.2536944348514887
epoch: 540, acc: 0.862, loss: 0.635, lr: 0.156494522691706Rr: 0.2521860111355852
epoch: 550, acc: 0.860, loss: 0.636, lr: 0.1540832049306626Rr: 0.2507017376340393
epoch: 560, acc: 0.860, loss: 0.635, lr: 0.1517450682852807Rr: 0.2492765087324014
epoch: 570, acc: 0.863, loss: 0.626, lr: 0.1494768310911809Rr: 0.2478632429819722
epoch: 580, acc: 0.864, loss: 0.624, lr: 0.1472754050073638Rr: 0.24646963727647586
epoch: 590, acc: 0.853, loss: 0.649, lr: 0.1451378809869376Rr: 0.245116360690906
```

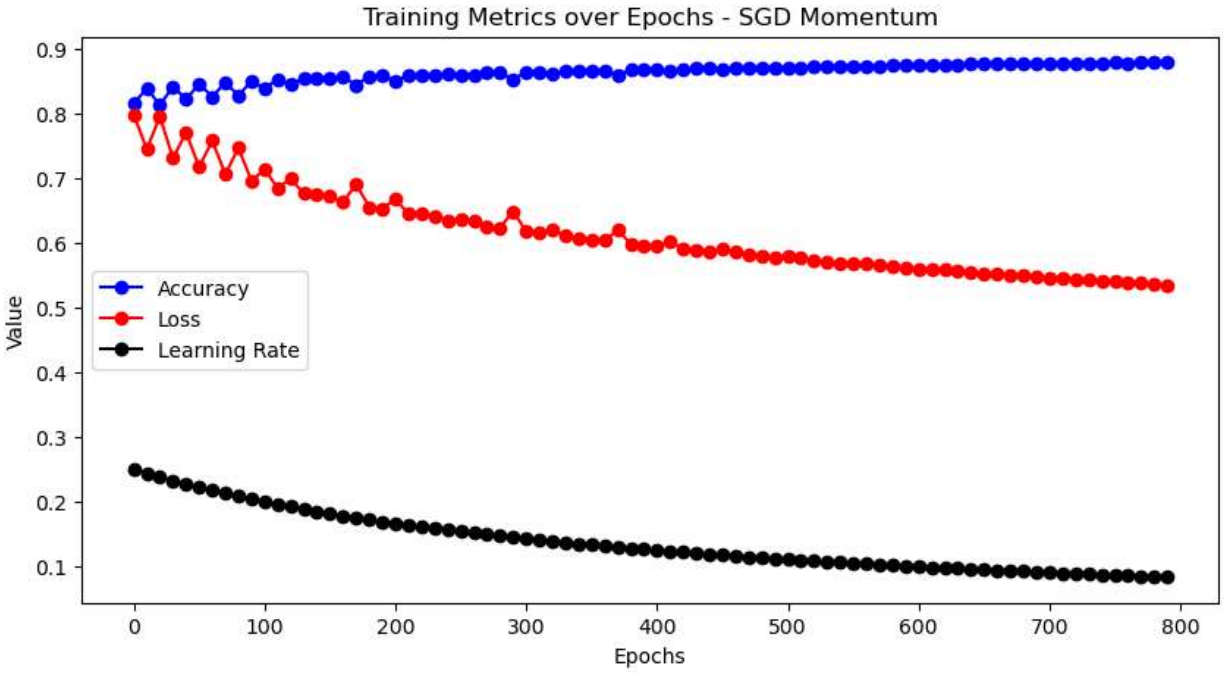
```
epoch: 600, acc: 0.865, loss: 0.618, lr: 0.14306151645207438Rr: 0.24380180424591147
epoch: 610, acc: 0.865, loss: 0.616, lr: 0.14104372355430184Rr: 0.24249199886800604
epoch: 620, acc: 0.862, loss: 0.620, lr: 0.13908205841446453Rr: 0.24120895968213402
epoch: 630, acc: 0.866, loss: 0.612, lr: 0.13717421124828533Rr: 0.23997083065036243
epoch: 640, acc: 0.867, loss: 0.608, lr: 0.13531799729364005Rr: 0.23873832464998349
epoch: 650, acc: 0.867, loss: 0.606, lr: 0.13351134846461948Rr: 0.23752516642494442
epoch: 660, acc: 0.867, loss: 0.605, lr: 0.13175230566534915Rr: 0.23633423717303495
epoch: 670, acc: 0.859, loss: 0.620, lr: 0.13003901170351104Rr: 0.2351814801993159
epoch: 680, acc: 0.868, loss: 0.599, lr: 0.12836970474967907Rr: 0.2340455106630411
epoch: 690, acc: 0.869, loss: 0.597, lr: 0.12674271229404308Rr: 0.23291826340858027
epoch: 700, acc: 0.869, loss: 0.595, lr: 0.1251564455569462Rr: 0.23180973508240504
epoch: 710, acc: 0.866, loss: 0.602, lr: 0.12360939431396786Rr: 0.23072340381855272
epoch: 720, acc: 0.869, loss: 0.592, lr: 0.12210012210012208Rr: 0.2296715342332999
epoch: 730, acc: 0.870, loss: 0.588, lr: 0.12062726176115804Rr: 0.22862277852564103
epoch: 740, acc: 0.870, loss: 0.587, lr: 0.11918951132300357Rr: 0.22758916323147182
epoch: 750, acc: 0.869, loss: 0.590, lr: 0.11778563015312131Rr: 0.2265751556642837
epoch: 760, acc: 0.870, loss: 0.586, lr: 0.11641443538998836Rr: 0.22558565524954838
epoch: 770, acc: 0.871, loss: 0.581, lr: 0.1150747986191024Rr: 0.22460566290643622
epoch: 780, acc: 0.871, loss: 0.579, lr: 0.11376564277588169Rr: 0.223638357836161
epoch: 790, acc: 0.872, loss: 0.578, lr: 0.11248593925759279Rr: 0.2226859364295567
epoch: 800, acc: 0.871, loss: 0.579, lr: 0.11123470522803114Rr: 0.22175009651495672
epoch: 810, acc: 0.871, loss: 0.578, lr: 0.11001100110011001Rr: 0.2208344680147308
epoch: 820, acc: 0.873, loss: 0.572, lr: 0.1088139281828074Rr: 0.21992957646301428
epoch: 830, acc: 0.873, loss: 0.570, lr: 0.1076426264800861Rr: 0.2190344781467941
epoch: 840, acc: 0.874, loss: 0.569, lr: 0.10649627263045792Rr: 0.21815191630845612
epoch: 850, acc: 0.874, loss: 0.568, lr: 0.1053740779768177Rr: 0.2172832579273551
epoch: 860, acc: 0.873, loss: 0.568, lr: 0.10427528675703858Rr: 0.21642896782519452
epoch: 870, acc: 0.874, loss: 0.566, lr: 0.10319917440660475Rr: 0.21558823470057467
epoch: 880, acc: 0.875, loss: 0.563, lr: 0.10214504596527067Rr: 0.21475829586756745
epoch: 890, acc: 0.875, loss: 0.561, lr: 0.10111223458038422Rr: 0.2139387607900529
epoch: 900, acc: 0.875, loss: 0.560, lr: 0.10010010010010009Rr: 0.21313044512539342
epoch: 910, acc: 0.875, loss: 0.559, lr: 0.09910802775024777Rr: 0.2123339680366307
epoch: 920, acc: 0.875, loss: 0.558, lr: 0.09813542688910697Rr: 0.21154960267445724
epoch: 930, acc: 0.876, loss: 0.556, lr: 0.09718172983479105Rr: 0.21077618636725431
epoch: 940, acc: 0.877, loss: 0.554, lr: 0.09624639076034648Rr: 0.21001208928853926
epoch: 950, acc: 0.877, loss: 0.553, lr: 0.09532888465204957Rr: 0.20925735365297157
epoch: 960, acc: 0.877, loss: 0.552, lr: 0.09442870632672333Rr: 0.20851258711738974
epoch: 970, acc: 0.877, loss: 0.551, lr: 0.09354536950420955Rr: 0.20777785779444702
epoch: 980, acc: 0.877, loss: 0.549, lr: 0.09267840593141798Rr: 0.20705289202658866
epoch: 990, acc: 0.877, loss: 0.548, lr: 0.09182736455463728Rr: 0.2063370311657284
epoch: 1000, acc: 0.878, loss: 0.546, lr: 0.09099181073703366Rr: 0.20562971923802648
epoch: 1010, acc: 0.878, loss: 0.545, lr: 0.09017132551848513Rr: 0.20493095540384038
epoch: 1020, acc: 0.878, loss: 0.544, lr: 0.08936550491510277Rr: 0.2042408285638163
epoch: 1030, acc: 0.878, loss: 0.543, lr: 0.08857395925597873Rr: 0.20355937531311674
epoch: 1040, acc: 0.879, loss: 0.541, lr: 0.08779631255487269Rr: 0.2028862578301819
epoch: 1050, acc: 0.879, loss: 0.540, lr: 0.08703220191470844Rr: 0.20222128891233956
epoch: 1060, acc: 0.879, loss: 0.539, lr: 0.08628127696289906Rr: 0.2015643424047353
epoch: 1070, acc: 0.879, loss: 0.538, lr: 0.0855431993156544Rr: 0.20091534385054344
epoch: 1080, acc: 0.879, loss: 0.537, lr: 0.08481764206955046Rr: 0.20027405678769697
epoch: 1090, acc: 0.879, loss: 0.536, lr: 0.08410428931875526Rr: 0.1996403552875046
```

In [22]:

```
epochs = range(0, 1100, 10)
plt.figure(figsize=(10, 5))
plt.plot(epochs, accuracies_mom, label='Accuracy', marker='o', color="blue")
plt.plot(epochs, losses_mom, label='Loss', marker='o', color='red')
plt.plot(epochs, learning_rate, label='Learning Rate', marker='o', color='black')
plt.title('Training Metrics over Epochs - SGD Momentum')
plt.xlabel('Epochs')
plt.ylabel('Value')
plt.legend()
```

```
plt.savefig('training_metrics_plot.png')
plt.show()

epochs = range(0, 800, 10)
plt.figure(figsize=(10, 5))
plt.plot(epochs, accuracies_mom[30:], label='Accuracy', marker='o', color="blue")
plt.plot(epochs, losses_mom[30:], label='Loss', marker='o', color='red')
plt.plot(epochs, learning_rate[30:], label='Learning Rate', marker='o', color='black')
plt.title('Training Metrics over Epochs - SGD Momentum')
plt.xlabel('Epochs')
plt.ylabel('Value')
plt.legend()
plt.savefig('training_metrics_plot.png')
plt.show()
```

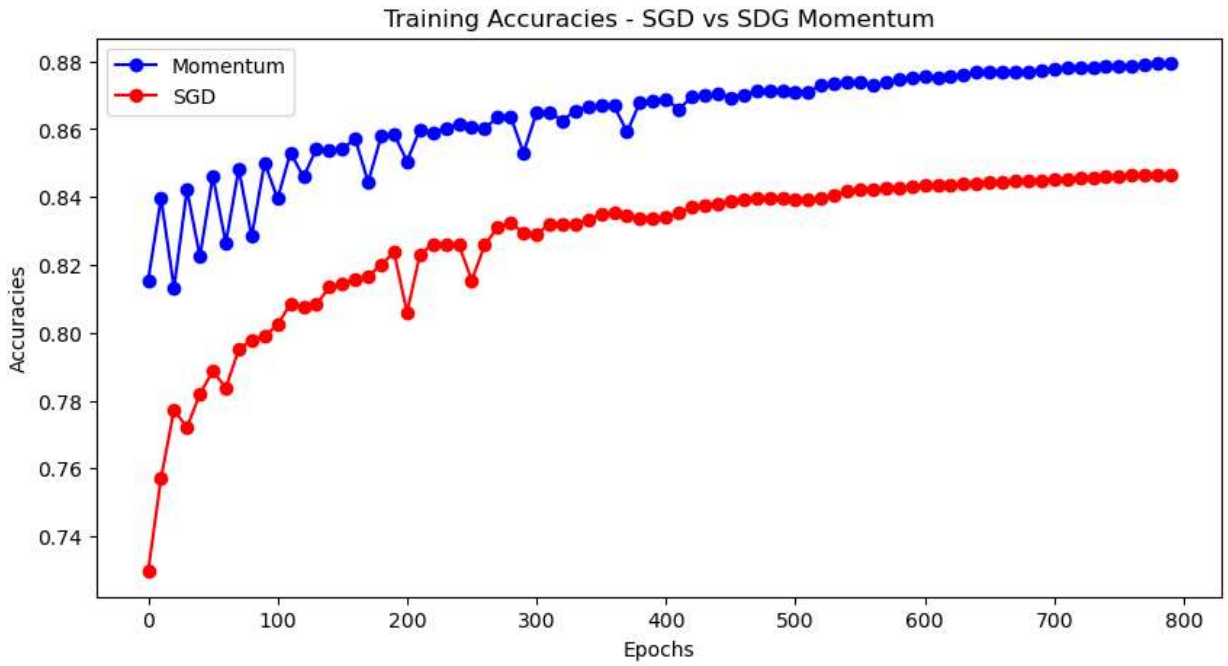




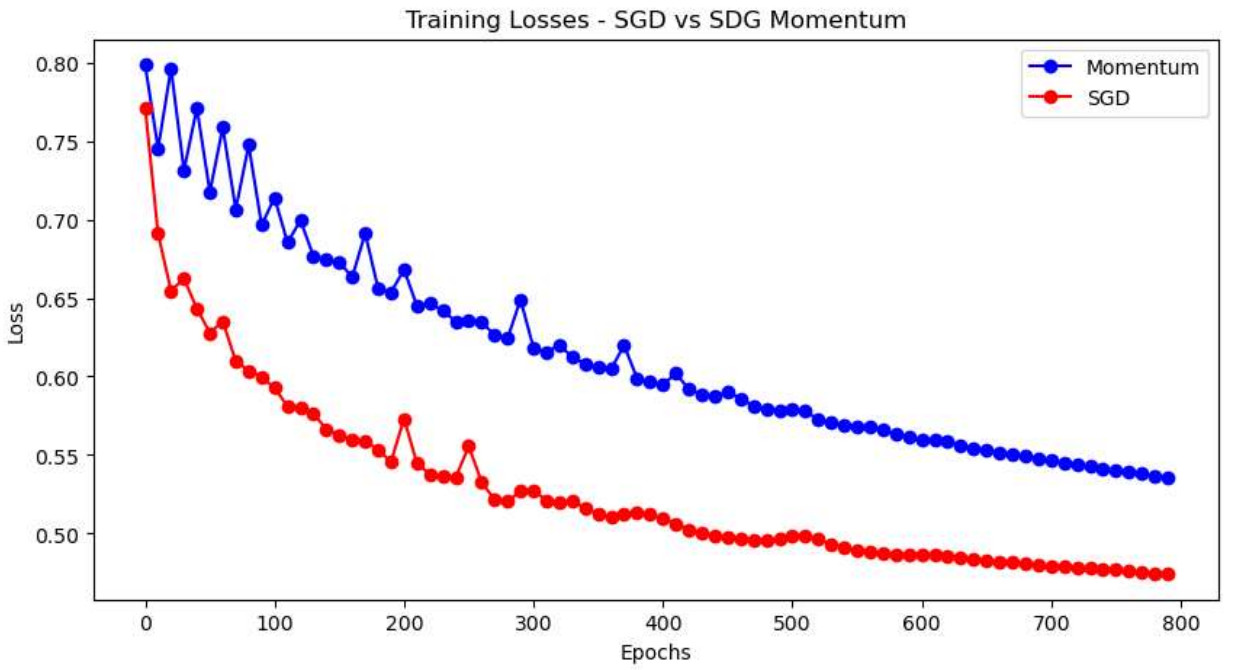
In [23]:
```
# Validate the model

dense1.forward(X_test)
activation1.forward(dense1.output)
dense2.forward(activation1.output)
activation2.forward(dense2.output)
dense3.forward(activation2.output)
data_loss = loss_activation.forward(dense3.output, y_test)

predictions = np.argmax(loss_activation.output, axis=1)
if len(y_test.shape) == 2:
    y_test = np.argmax(y, axis=1)
accuracy = np.mean(predictions==y_test)
print(f'validation - SGD Momentum, acc: {accuracy:.3f}, loss: {loss:.3f}')
```

```
validation - SGD Momentum, acc: 0.859, loss: 0.535
```

In [24]:
```
epochs = range(0, 800, 10)
plt.figure(figsize=(10, 5))
plt.plot(epochs, accuracies_mom[30:], label='Momentum', marker='o', color="blue")
plt.plot(epochs, accuracies_sgd[30:], label='SGD', marker='o', color='red')
plt.title('Training Accuracies - SGD vs SGD Momentum')
plt.xlabel('Epochs')
plt.ylabel('Accuracies')
plt.legend()
plt.savefig('training_metrics_plot.png')
plt.show()
```



In [25]:
```
epochs = range(0, 800, 10)
plt.figure(figsize=(10, 5))
plt.plot(epochs, losses_mom[30:], label='Momentum', marker='o', color="blue")
plt.plot(epochs, losses_sgd[30:], label='SGD', marker='o', color='red')
plt.title('Training Losses - SGD vs SGD Momentum')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.savefig('training_metrics_plot.png')
plt.show()
```



In [28]:
```
class Optimizer_Adam:
    # Initialize optimizer - set settings
    def __init__(self, learning_rate=0.001, decay=0., epsilon=1e-7, beta_1=0.9, beta_2
        self.learning_rate = learning_rate
        self.current_learning_rate = learning_rate
        self.decay = decay
        self.iterations = 0
        self.epsilon = epsilon
        self.beta_1 = beta_1
        self.beta_2 = beta_2

    def pre_update_params(self):
        if self.decay:
            self.current_learning_rate = self.learning_rate * (1. / (1. + self.decay *

    def post_update_params(self):
        self.iterations += 1

    def update_params(self, layer):
        if not hasattr(layer, 'weight_cache'):
            layer.weight_momentums = np.zeros_like(layer.weights)
            layer.weight_cache = np.zeros_like(layer.weights)
            layer.bias_momentums = np.zeros_like(layer.biases)
            layer.bias_cache = np.zeros_like(layer.biases)

        layer.weight_momentums = self.beta_1 * layer.weight_momentums + (1 - self.beta
        layer.bias_momentums = self.beta_1 * layer.bias_momentums + (1 - self.beta_1)

        weight_momentums_corrected = layer.weight_momentums / (1 - self.beta_1 ** (sel
        bias_momentums_corrected = layer.bias_momentums / (1 - self.beta_1 ** (self.it

        layer.weight_cache = self.beta_2 * layer.weight_cache + (1 - self.beta_2) * la
        layer.bias_cache = self.beta_2 * layer.bias_cache + (1 - self.beta_2) * layer.

        weight_cache_corrected = layer.weight_cache / (1 - self.beta_2 ** (self.iterat
        bias_cache_corrected = layer.bias_cache / (1 - self.beta_2 ** (self.iterations
```

```
        layer.weights += -self.current_learning_rate * weight_momentums_corrected / (r
        layer.biases += -self.current_learning_rate * bias_momentums_corrected / (np.s
```

In [29]:
```
"""
Features = 784
Samples = 60,000
class labels = 10 (0 - 9)

2 hidden layes - ReLU    - 128 neurons each
1 output layer - Softmax - 10 output neurons

Optimizer - Adam
learning rate decay - 0.00001
l2 regularization - lambda - 5e-4 (0.0005)
"""

dense1 = Layer_Dense(X.shape[1], 128, weight_regularizer_l2=5e-4, bias_regularizer_l2=
activation1 = Activation_ReLU()
dense2 = Layer_Dense(128, 128)
activation2 = Activation_ReLU()
dense3 = Layer_Dense(128, 10)
loss_activation = Activation_Softmax_Loss_CategoricalCrossentropy()

optimizer = Optimizer_Adam(learning_rate=0.02, decay=1e-5)

accuracies_adam = []
losses_adam = []
learning_rate = []
```

In [30]:
```
for epoch in range(1100):
    dense1.forward(X)
    activation1.forward(dense1.output)
    dense2.forward(activation1.output)
    activation2.forward(dense2.output)
    dense3.forward(activation2.output)
    data_loss = loss_activation.forward(dense3.output, y)

    regularization_loss = loss_activation.loss.regularization_loss(dense1) + loss_acti
        loss_activation.loss.regularization_loss(dense3)

    loss = data_loss + regularization_loss

    predictions = np.argmax(loss_activation.output, axis=1)
    if len(y.shape) == 2:
        y = np.argmax(y, axis=1)
    accuracy = np.mean(predictions==y)


    if not epoch % 10:
        accuracies_adam.append(accuracy)
        losses_adam.append(loss)
        learning_rate.append(optimizer.current_learning_rate)
        print(f'epoch: {epoch}, ' + f'acc: {accuracy:.3f}, ' + f'loss: {loss:.3f}, ' +
              + f'Rr: {regularization_loss}')

    loss_activation.backward(loss_activation.output, y)
    dense3.backward(loss_activation.dinputs)
    activation2.backward(dense3.dinputs)
    dense2.backward(activation2.dinputs)
```

```
    activation1.backward(dense2.dinputs)
    dense1.backward(activation1.dinputs)

    optimizer.pre_update_params()
    optimizer.update_params(dense1)
    optimizer.update_params(dense2)
    optimizer.update_params(dense3)
    optimizer.post_update_params()
```

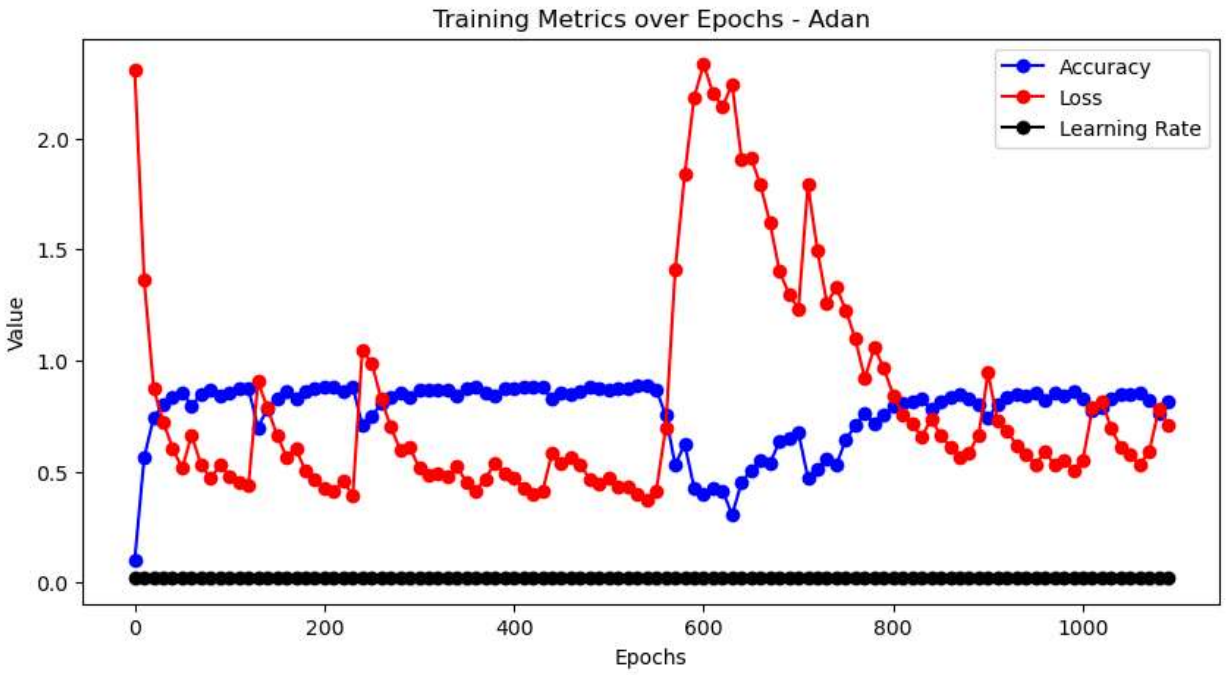
```
epoch: 0, acc: 0.102, loss: 2.308, lr: 0.02Rr: 0.004996665831396659
epoch: 10, acc: 0.565, loss: 1.361, lr: 0.019998200161985422Rr: 0.14170242306860456
epoch: 20, acc: 0.743, loss: 0.872, lr: 0.019996200721862846Rr: 0.21005682255472272
epoch: 30, acc: 0.801, loss: 0.724, lr: 0.019994201681512364Rr: 0.19406223297949274
epoch: 40, acc: 0.835, loss: 0.604, lr: 0.019992203040814085Rr: 0.15948530013349232
epoch: 50, acc: 0.857, loss: 0.514, lr: 0.01999020479964817Rr: 0.1228218672809067
epoch: 60, acc: 0.795, loss: 0.663, lr: 0.01998820695789484Rr: 0.11040424993821865
epoch: 70, acc: 0.849, loss: 0.529, lr: 0.01998620951543435Rr: 0.11473025966845592
epoch: 80, acc: 0.865, loss: 0.470, lr: 0.019984212472147003Rr: 0.1033563135580361
epoch: 90, acc: 0.844, loss: 0.528, lr: 0.019982215827913156Rr: 0.09009759562034428
epoch: 100, acc: 0.856, loss: 0.478, lr: 0.01998021958261321Rr: 0.09258836524122703
epoch: 110, acc: 0.872, loss: 0.449, lr: 0.01997822373612762Rr: 0.10291756329531905
epoch: 120, acc: 0.871, loss: 0.440, lr: 0.01997622828833688Rr: 0.08970743076139927
epoch: 130, acc: 0.699, loss: 0.904, lr: 0.019974233239121533Rr: 0.11600543371767164
epoch: 140, acc: 0.783, loss: 0.785, lr: 0.01997223858836217 8Rr: 0.1789446646122182
epoch: 150, acc: 0.829, loss: 0.665, lr: 0.01997024433593945Rr: 0.19807191692016152
epoch: 160, acc: 0.858, loss: 0.566, lr: 0.01996825048173404Rr: 0.17014569814951483
epoch: 170, acc: 0.831, loss: 0.601, lr: 0.019966257025626693Rr: 0.1372904928599926
epoch: 180, acc: 0.860, loss: 0.503, lr: 0.019964263967498178Rr: 0.12555371631636456
epoch: 190, acc: 0.871, loss: 0.461, lr: 0.01996227130722934Rr: 0.10935366314268374
epoch: 200, acc: 0.882, loss: 0.422, lr: 0.019960279044701046Rr: 0.09355605527815435
epoch: 210, acc: 0.880, loss: 0.412, lr: 0.019958287179794233Rr: 0.08124032410321512
epoch: 220, acc: 0.858, loss: 0.454, lr: 0.01995629571238986 8Rr: 0.07695414690903532
epoch: 230, acc: 0.883, loss: 0.392, lr: 0.019954304642368977Rr: 0.06902294203970671
epoch: 240, acc: 0.706, loss: 1.045, lr: 0.01995231396961263Rr: 0.13492154589607722
epoch: 250, acc: 0.747, loss: 0.988, lr: 0.0199503236940019 3Rr: 0.27971085241639354
epoch: 260, acc: 0.807, loss: 0.825, lr: 0.019948333815418065Rr: 0.2999324314262388
epoch: 270, acc: 0.833, loss: 0.701, lr: 0.01994634433374223 3Rr: 0.25459211208844057
epoch: 280, acc: 0.854, loss: 0.598, lr: 0.01994435524885569 3Rr: 0.2028971818609663
epoch: 290, acc: 0.835, loss: 0.611, lr: 0.0199423665606397 5Rr: 0.1643061436219584
epoch: 300, acc: 0.865, loss: 0.515, lr: 0.01994037826897576 3Rr: 0.14009671687117065
epoch: 310, acc: 0.867, loss: 0.484, lr: 0.0199383903737451 3Rr: 0.12319717678511542
epoch: 320, acc: 0.865, loss: 0.491, lr: 0.019936402874829295Rr: 0.11737223056720762
epoch: 330, acc: 0.867, loss: 0.477, lr: 0.01993441577210976Rr: 0.11535715092223137
epoch: 340, acc: 0.843, loss: 0.521, lr: 0.019932429065468063Rr: 0.10424979308063556
epoch: 350, acc: 0.873, loss: 0.451, lr: 0.01993044427547858Rr: 0.10044562279266711
epoch: 360, acc: 0.882, loss: 0.415, lr: 0.0199284568399446Rr: 0.09157738194393035
epoch: 370, acc: 0.854, loss: 0.467, lr: 0.01992647132082615Rr: 0.08033682671234241
epoch: 380, acc: 0.842, loss: 0.538, lr: 0.01992448619731219Rr: 0.11594202259649856
epoch: 390, acc: 0.875, loss: 0.492, lr: 0.01992250146928448 5Rr: 0.14637807137378894
epoch: 400, acc: 0.874, loss: 0.473, lr: 0.0199205171366248 7Rr: 0.12751576633193287
epoch: 410, acc: 0.883, loss: 0.425, lr: 0.01991853319921521 2Rr: 0.10528802546096354
epoch: 420, acc: 0.884, loss: 0.399, lr: 0.0199165496569374 34Rr: 0.08305689342070251
epoch: 430, acc: 0.880, loss: 0.409, lr: 0.0199145665096735 03Rr: 0.0837260471239533
epoch: 440, acc: 0.825, loss: 0.582, lr: 0.01991258375730543Rr: 0.09361558409383294
epoch: 450, acc: 0.852, loss: 0.540, lr: 0.01991060139971527 5Rr: 0.13021416317917678
epoch: 460, acc: 0.846, loss: 0.564, lr: 0.01990861943678515 2Rr: 0.14860244106561502
epoch: 470, acc: 0.864, loss: 0.528, lr: 0.0199066377686839 7217Rr: 0.14873412727951443
epoch: 480, acc: 0.878, loss: 0.467, lr: 0.0199046565694433 663Rr: 0.13168427771472302
epoch: 490, acc: 0.875, loss: 0.447, lr: 0.019902675914776 74Rr: 0.10780081058876319
epoch: 500, acc: 0.865, loss: 0.468, lr: 0.01990069552930875Rr: 0.1003307641872221
epoch: 510, acc: 0.873, loss: 0.434, lr: 0.01989871553791203Rr: 0.09299588804233423
epoch: 520, acc: 0.874, loss: 0.432, lr: 0.0198967359404689 65Rr: 0.0913840921581966
epoch: 530, acc: 0.885, loss: 0.399, lr: 0.019894756736862Rr: 0.08452024699474316
epoch: 540, acc: 0.889, loss: 0.370, lr: 0.0198927779269736 13Rr: 0.06773907658272386
epoch: 550, acc: 0.870, loss: 0.413, lr: 0.0198907995106863 34Rr: 0.05615687918664161
epoch: 560, acc: 0.753, loss: 0.693, lr: 0.0198888214878827 35Rr: 0.09829468296658146
epoch: 570, acc: 0.529, loss: 1.407, lr: 0.0198868438584454 48Rr: 0.3064670419700132
epoch: 580, acc: 0.624, loss: 1.836, lr: 0.0198848666222571 32Rr: 0.5685042349173975
epoch: 590, acc: 0.426, loss: 2.185, lr: 0.0198828897792005 1Rr: 0.6664401603781481
```

```
epoch: 600, acc: 0.396, loss: 2.333, lr: 0.019880913329158343Rr: 0.691344394495626
epoch: 610, acc: 0.427, loss: 2.202, lr: 0.019878937272013 44Rr: 0.6947282334859429
epoch: 620, acc: 0.414, loss: 2.142, lr: 0.019876961607648656Rr: 0.630958707575741
epoch: 630, acc: 0.305, loss: 2.245, lr: 0.019874986335946896Rr: 0.5917230259860978
epoch: 640, acc: 0.450, loss: 1.907, lr: 0.019873011456791108Rr: 0.5518324492819404
epoch: 650, acc: 0.506, loss: 1.912, lr: 0.01987103697006428Rr: 0.5018659713544907
epoch: 660, acc: 0.549, loss: 1.790, lr: 0.0198690628756 4947Rr: 0.5343277687065641
epoch: 670, acc: 0.537, loss: 1.620, lr: 0.019867089173429754Rr: 0.4900358921335707
epoch: 680, acc: 0.636, loss: 1.401, lr: 0.01986511586328 8273Rr: 0.42764551229220865
epoch: 690, acc: 0.649, loss: 1.300, lr: 0.019863142945108204Rr: 0.36748845574328637
epoch: 700, acc: 0.675, loss: 1.231, lr: 0.019861170418772778Rr: 0.3286655553407996
epoch: 710, acc: 0.472, loss: 1.792, lr: 0.01985919828416 5266Rr: 0.3226212382388002
epoch: 720, acc: 0.511, loss: 1.497, lr: 0.019857226541168997Rr: 0.31575968927724296
epoch: 730, acc: 0.558, loss: 1.258, lr: 0.01985525518966 7326Rr: 0.2729615312999502
epoch: 740, acc: 0.530, loss: 1.327, lr: 0.019853284229543675Rr: 0.25300599925525835
epoch: 750, acc: 0.644, loss: 1.227, lr: 0.01985131366068 1495Rr: 0.28713433511092706
epoch: 760, acc: 0.706, loss: 1.096, lr: 0.019849343482964302Rr: 0.2709344937510289
epoch: 770, acc: 0.764, loss: 0.918, lr: 0.019847373696275643Rr: 0.23280955861547573
epoch: 780, acc: 0.714, loss: 1.059, lr: 0.019845404300499112Rr: 0.20486316561739437
epoch: 790, acc: 0.754, loss: 0.965, lr: 0.019843435295518363Rr: 0.2429355256999479
epoch: 800, acc: 0.797, loss: 0.843, lr: 0.019841466681217078Rr: 0.22539154914604564
epoch: 810, acc: 0.806, loss: 0.758, lr: 0.019839498457478996Rr: 0.18355716913267786
epoch: 820, acc: 0.815, loss: 0.715, lr: 0.019837530624187902Rr: 0.16145457156685958
epoch: 830, acc: 0.826, loss: 0.657, lr: 0.019835563181227624Rr: 0.14215841912869276
epoch: 840, acc: 0.778, loss: 0.735, lr: 0.01983359612848204Rr: 0.12514491586780488
epoch: 850, acc: 0.818, loss: 0.665, lr: 0.019831629465835058Rr: 0.12812846350534512
epoch: 860, acc: 0.833, loss: 0.609, lr: 0.01982966319317066Rr: 0.11766045970075631
epoch: 870, acc: 0.845, loss: 0.567, lr: 0.019827697310372858Rr: 0.10271255519523197
epoch: 880, acc: 0.830, loss: 0.585, lr: 0.01982573181732 5706Rr: 0.09492582657474255
epoch: 890, acc: 0.804, loss: 0.663, lr: 0.0198237667139 1331Rr: 0.09459173070625977
epoch: 900, acc: 0.741, loss: 0.944, lr: 0.01982180200001982Rr: 0.10230811953680947
epoch: 910, acc: 0.803, loss: 0.726, lr: 0.019819837675529438Rr: 0.16794456167205016
epoch: 920, acc: 0.832, loss: 0.685, lr: 0.01981787374032 64Rr: 0.186024549536082
epoch: 930, acc: 0.846, loss: 0.617, lr: 0.019815910194295Rr: 0.15399518032370577
epoch: 940, acc: 0.844, loss: 0.579, lr: 0.01981394703731957Rr: 0.12059156809302897
epoch: 950, acc: 0.853, loss: 0.531, lr: 0.019811984269284492Rr: 0.09815467965363092
epoch: 960, acc: 0.820, loss: 0.588, lr: 0.01981002189007419Rr: 0.08489357075056499
epoch: 970, acc: 0.851, loss: 0.531, lr: 0.019808059899573138Rr: 0.09163015171806459
epoch: 980, acc: 0.844, loss: 0.549, lr: 0.019806098297665855Rr: 0.08938889699288345
epoch: 990, acc: 0.858, loss: 0.502, lr: 0.01980413708423 6898Rr: 0.08562149507985403
epoch: 1000, acc: 0.831, loss: 0.552, lr: 0.019802176259170884Rr: 0.07734252300870392
epoch: 1010, acc: 0.773, loss: 0.785, lr: 0.019800215822352467Rr: 0.12127206453256004
epoch: 1020, acc: 0.787, loss: 0.814, lr: 0.019798255773666344Rr: 0.2025755404485572
epoch: 1030, acc: 0.830, loss: 0.693, lr: 0.019796296112997262Rr: 0.19984875656040482
epoch: 1040, acc: 0.846, loss: 0.608, lr: 0.019794336840230013Rr: 0.16218725068749634
epoch: 1050, acc: 0.844, loss: 0.573, lr: 0.01979237795524943Rr: 0.1268952856549617
epoch: 1060, acc: 0.855, loss: 0.531, lr: 0.019790419457940408Rr: 0.10643901336286486
epoch: 1070, acc: 0.824, loss: 0.591, lr: 0.01978846134818787Rr: 0.09204123745057095
epoch: 1080, acc: 0.762, loss: 0.780, lr: 0.0197865036258 7679Rr: 0.13911956141786772
epoch: 1090, acc: 0.813, loss: 0.712, lr: 0.019784546290892182Rr: 0.1881587873272313
```

In [31]:

```
epochs = range(0, 1100, 10)
plt.figure(figsize=(10, 5))
plt.plot(epochs, accuracies_adam, label='Accuracy', marker='o', color="blue")
plt.plot(epochs, losses_adam, label='Loss', marker='o', color='red')
plt.plot(epochs, learning_rate, label='Learning Rate', marker='o', color='black')
plt.title('Training Metrics over Epochs - Adan')
plt.xlabel('Epochs')
plt.ylabel('Value')
plt.legend()
```

```
plt.savefig('training_metrics_plot.png')
plt.show()
```



In [32]:
```
# Validate the model

dense1.forward(X_test)
activation1.forward(dense1.output)
dense2.forward(activation1.output)
activation2.forward(dense2.output)
dense3.forward(activation2.output)
data_loss = loss_activation.forward(dense3.output, y_test)

predictions = np.argmax(loss_activation.output, axis=1)
if len(y_test.shape) == 2:
    y_test = np.argmax(y, axis=1)
accuracy = np.mean(predictions==y_test)
print(f'validation - Adam, acc: {accuracy:.3f}, loss: {loss:.3f}')
```

```
validation - Adam, acc: 0.817, loss: 0.657
```

In [ ]: