# Neural Network from Scratch Fashion MNIST - Assignment

## Submitted By:

**Ayush Dhanraj** - 2023DS06
**Ashutosh Kumar** - 2023DS26

### Imports

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

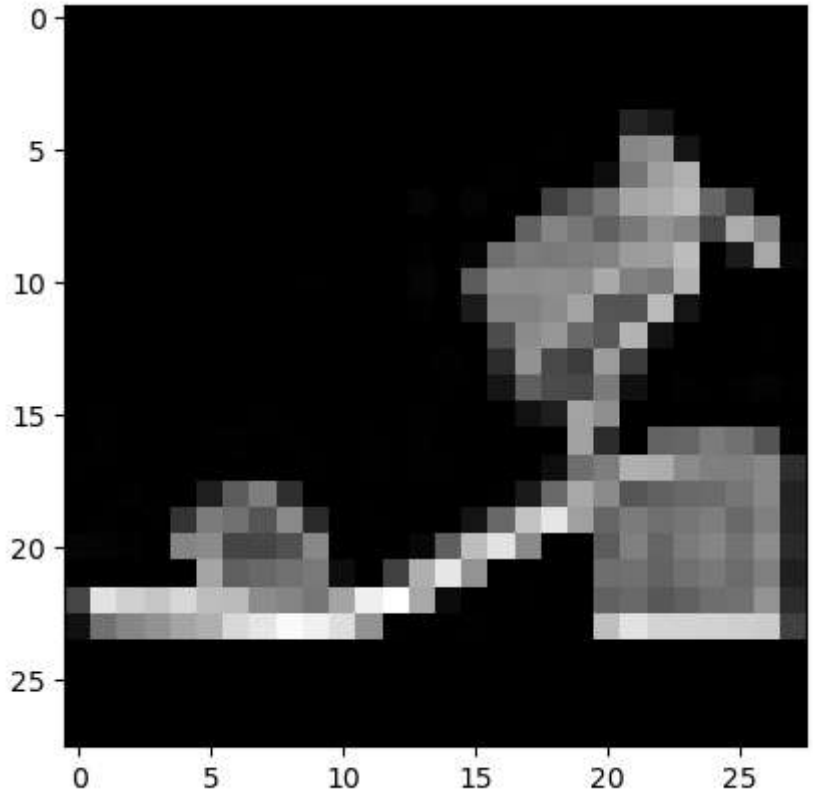
### Download → Split → Scale → Reshape → Shuffle → Verify

🧹 Data Pre-processing

In [12]:
```
"""
Features = 784
Samples = 60,000
class labels = 10 (0 - 9)
"""

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

X, y, X_test, y_test = create_data_mnist('fashion_mnist_images')

"""
Current range 0 - 255
Scale in range -1 to 1 centered to 0
"""
X = (X.astype(np.float32) - 127.5) / 127.5
X_test = (X_test.astype(np.float32) - 127.5) / 127.5

print(X.shape)
print(y.shape)

"""
pixel size - 28 x 28 - 784 features
images are reshaped to be next to each other
or
Reshape to vectors (1 - D Array aka list)
"""
X = X.reshape(X.shape[0], -1)
X_test = X_test.reshape(X_test.shape[0], -1)
print(X.shape[1])

"""
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()
```

```
(60000, 28, 28)
(60000,)
784
[0 1 2 3 4 5 6 7 8 9]
```




## Dense Layer aka Fully Connect Layer

In [13]:
```
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 that they are close to zero and NN doesn't e

        biases is set to zero initially and if we encounter error where the entire out
        NN is zero we can 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):
```

```
        """
        used in backpropagation
        dvalues is gradient from next layer
        """
        self.dweights = np.dot(self.inputs.T, dvalues)
        self.dbiases = np.sum(dvalues, axis=0, keepdims=True)

        # Gradients - 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)
```

## 🏃 Activation Functions

In [14]:
```
class Activation_ReLU:
    """
    Almost Linear - fast & accurate
    mostly used in hidden layers
    """
    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
```

In [15]:
```
class Activation_Softmax:
    """
    exponent & normalize
    common classification activation function
    mostly used in output layer
    """
    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
```

## 📉 Loss Function

In [16]:

```
class Loss:
    """
    common loss class
    calculates mean after sample loss is calculated
    calculates regularization if specifies during layer Instantiation
    """
    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):
    """
    common loss function for multi - class classification problems
    """
    def forward(self, y_pred, y_true):
        """
        takes in predicted and expected output
        accounts for zero values -log(0) = inf and then remove biasis caused by its in
        returns - log (correct_confidences)
        """
        samples = len(y_pred)

        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):
        """
        takes gradient from next layer
        returns gradient wrt to inputs
        """

        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
```

## 🤝 Aggregate - Softmax & Categorical Cross Entropy

In [17]:

```
class Activation_Softmax_Loss_CategoricalCrossentropy():
    """
    forward - calculates activation output
    backpropagation - calculate gradient of softmax output wrt loss
    """
    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
```

## Optimizers: 🙂 SGD 😃 SGD Momentum 😁 Adam

In [18]:
```
class Optimizer_SGD:
    """
    stochastic gradient descent with learning rate decay
    """
    def __init__(self, learning_rate=1.0, decay=0):
        self.learning_rate = learning_rate
        self.current_learning_rate = learning_rate
        self.decay = decay
        self.iterations = 0

    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):
        weight_updates = -self.current_learning_rate * layer.dweights
        bias_updates = -self.current_learning_rate * layer.dbiases

        layer.weights += weight_updates
        layer.biases += bias_updates

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

class Optimizer_SGD_Momentum:
    """
    SGD with momentum
    """
    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 *

    def update_params(self, layer):
        # 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_learnir
        layer.weight_momentums = weight_updates

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

        layer.weights += weight_updates
        layer.biases += bias_updates

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

```
class Optimizer_Adam:
    """
    Adam - RMSProp + SGD with momentum
    """
    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 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

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

In [19]:

```
def train_Model(epochs, X, y, dense1, dense2, dense3, activation1, activation2, loss_a
    """
    training data - traing expected predictions
    utility function takes in three layers
    2 activation function
    output activation + loss function
    optimizer of choice
    """
    accuracies = []
    losses = []
    learning_rate = []

    for epoch in range(epochs):
        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_
            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.append(accuracy)
            losses.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()

    return accuracies, losses, learning_rate
```

## Model - SGD

In [21]:
```
"""
2 hidden layers - ReLU    - 128 neurons each
1 output layer  - Softmax - 10 output neurons

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)
```

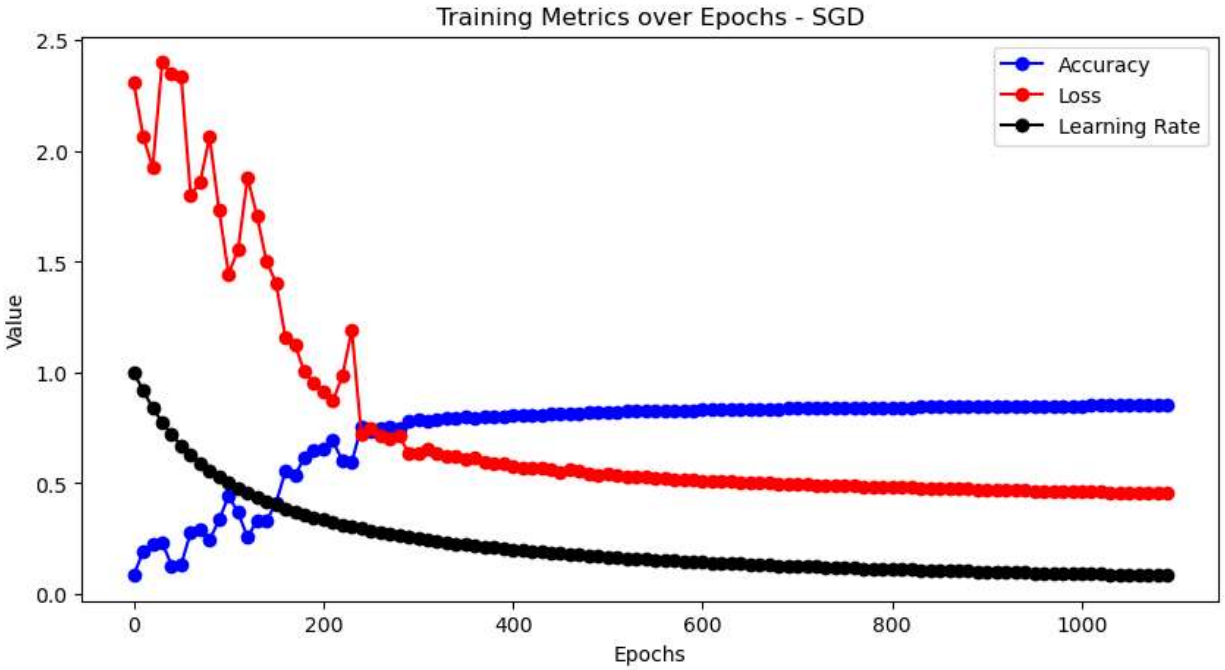
In [22]:
```
accuracies, losses, learning_rate = train_Model(1100, X, y, dense1, dense2, dense3, ac
```

```
epoch: 0, acc: 0.085, loss: 2.308, lr: 1.0 Rr: 0.005059542213208445
epoch: 10, acc: 0.194, loss: 2.065, lr: 0.9174311926605504 Rr: 0.005503636788984646
epoch: 20, acc: 0.227, loss: 1.926, lr: 0.8403361344537815 Rr: 0.03145679680319871
epoch: 30, acc: 0.232, loss: 2.402, lr: 0.7751937984496123 Rr: 0.04135866634962637
epoch: 40, acc: 0.127, loss: 2.349, lr: 0.7194244604316546 Rr: 0.05008126218330518
epoch: 50, acc: 0.133, loss: 2.338, lr: 0.6711409395973155 Rr: 0.049696640433813374
epoch: 60, acc: 0.277, loss: 1.802, lr: 0.628930817610063 Rr: 0.050752140383866655
epoch: 70, acc: 0.292, loss: 1.858, lr: 0.591715976331361 Rr: 0.05075627429585336
epoch: 80, acc: 0.244, loss: 2.066, lr: 0.5586592178770949 Rr: 0.05143420192205202
epoch: 90, acc: 0.334, loss: 1.733, lr: 0.5291005291005291 Rr: 0.05380842142232751
epoch: 100, acc: 0.443, loss: 1.446, lr: 0.5025125628140703 Rr: 0.053444498565793
epoch: 110, acc: 0.368, loss: 1.553, lr: 0.47846889952153115 Rr: 0.05338609579708597
epoch: 120, acc: 0.255, loss: 1.881, lr: 0.4566210045662101 Rr: 0.053284520572558315
epoch: 130, acc: 0.331, loss: 1.709, lr: 0.4366812227074236 Rr: 0.05320135733977414
epoch: 140, acc: 0.329, loss: 1.502, lr: 0.41841004184100417 Rr: 0.05335201613844296
epoch: 150, acc: 0.411, loss: 1.406, lr: 0.4016064257028112 Rr: 0.05420008103564568
epoch: 160, acc: 0.553, loss: 1.157, lr: 0.3861003861003861 Rr: 0.054209729735093
epoch: 170, acc: 0.534, loss: 1.126, lr: 0.3717472118959108 Rr: 0.053972659817800084
epoch: 180, acc: 0.617, loss: 1.003, lr: 0.35842293906810035 Rr: 0.053832210480692914
epoch: 190, acc: 0.649, loss: 0.951, lr: 0.3460207612456747 Rr: 0.053664087947193996
epoch: 200, acc: 0.657, loss: 0.915, lr: 0.33444816053511706 Rr: 0.053500429150790785
epoch: 210, acc: 0.691, loss: 0.872, lr: 0.3236245954692557 Rr: 0.05335549121287524
epoch: 220, acc: 0.601, loss: 0.984, lr: 0.31347962382445144 Rr: 0.053264830510683654
epoch: 230, acc: 0.595, loss: 1.191, lr: 0.303951367781155 Rr: 0.053298408382477805
epoch: 240, acc: 0.755, loss: 0.721, lr: 0.2949852507374631 Rr: 0.053434235420635676
epoch: 250, acc: 0.734, loss: 0.751, lr: 0.28653295128939826 Rr: 0.05336250957596532
epoch: 260, acc: 0.748, loss: 0.717, lr: 0.2785515320334262 Rr: 0.05330357510881777
epoch: 270, acc: 0.753, loss: 0.700, lr: 0.2710027100271003 Rr: 0.053197261278141054
epoch: 280, acc: 0.750, loss: 0.715, lr: 0.2638522427440633 Rr: 0.05304842722547707
epoch: 290, acc: 0.784, loss: 0.634, lr: 0.2570694087403599 Rr: 0.05293645468079071
epoch: 300, acc: 0.785, loss: 0.638, lr: 0.2506265664160401 Rr: 0.052859614150939746
epoch: 310, acc: 0.779, loss: 0.657, lr: 0.24449877750611249 Rr: 0.05271432668251726
epoch: 320, acc: 0.788, loss: 0.634, lr: 0.23866348448687355 Rr: 0.05258927993687847
epoch: 330, acc: 0.791, loss: 0.621, lr: 0.2331002331002331 Rr: 0.05246939186167195
epoch: 340, acc: 0.792, loss: 0.622, lr: 0.2277904328018223 Rr: 0.052343847316209224
epoch: 350, acc: 0.797, loss: 0.611, lr: 0.22271714922048996 Rr: 0.052231206760561276
epoch: 360, acc: 0.793, loss: 0.613, lr: 0.2178649237472767 Rr: 0.05212467419236713
epoch: 370, acc: 0.800, loss: 0.597, lr: 0.21321961620469085 Rr: 0.05200199863924819
epoch: 380, acc: 0.803, loss: 0.588, lr: 0.20876826722338204 Rr: 0.0518754455146933
epoch: 390, acc: 0.803, loss: 0.586, lr: 0.20449897750511245 Rr: 0.0517568183184564
epoch: 400, acc: 0.808, loss: 0.576, lr: 0.2004008016032064 Rr: 0.0516490218120781
epoch: 410, acc: 0.810, loss: 0.570, lr: 0.19646365422396858 Rr: 0.051535852135708354
epoch: 420, acc: 0.810, loss: 0.567, lr: 0.19267822736030826 Rr: 0.05142181496872441
epoch: 430, acc: 0.810, loss: 0.568, lr: 0.1890359168241966 Rr: 0.05130790027668203
epoch: 440, acc: 0.813, loss: 0.561, lr: 0.1855287569573284 Rr: 0.051193382606677086
epoch: 450, acc: 0.816, loss: 0.552, lr: 0.18214936247723132 Rr: 0.05107596803867855
epoch: 460, acc: 0.813, loss: 0.559, lr: 0.17889087656529518 Rr: 0.05096167956633924
epoch: 470, acc: 0.815, loss: 0.556, lr: 0.17574692442882248 Rr: 0.05086985256254349
epoch: 480, acc: 0.822, loss: 0.541, lr: 0.17271157167530224 Rr: 0.050771318476832264
epoch: 490, acc: 0.822, loss: 0.539, lr: 0.16977928692699493 Rr: 0.0506628090514307
epoch: 500, acc: 0.819, loss: 0.542, lr: 0.1669449081803005 Rr: 0.050559328152655715
epoch: 510, acc: 0.823, loss: 0.534, lr: 0.16420361247947454 Rr: 0.05045582301670324
epoch: 520, acc: 0.824, loss: 0.531, lr: 0.16155088852988692 Rr: 0.05035021379565129
epoch: 530, acc: 0.825, loss: 0.529, lr: 0.1589825119236884 Rr: 0.05024595308957669
epoch: 540, acc: 0.826, loss: 0.527, lr: 0.1564945226917058 Rr: 0.050144578874197154
epoch: 550, acc: 0.827, loss: 0.524, lr: 0.15408320493066255 Rr: 0.05004520836738017
epoch: 560, acc: 0.828, loss: 0.521, lr: 0.15174506828528073 Rr: 0.04994675489741125
epoch: 570, acc: 0.829, loss: 0.519, lr: 0.14947683109118085 Rr: 0.04984925930673661
epoch: 580, acc: 0.830, loss: 0.517, lr: 0.14727540500736377 Rr: 0.04975310463527012
epoch: 590, acc: 0.830, loss: 0.515, lr: 0.14513788098693758 Rr: 0.04965805710131912
```

```
epoch: 600, acc: 0.831, loss: 0.512, lr: 0.14306151645207438 Rr: 0.04956403289995318
epoch: 610, acc: 0.832, loss: 0.510, lr: 0.14104372355430184 Rr: 0.049471158396725834
epoch: 620, acc: 0.833, loss: 0.509, lr: 0.13908205841446453 Rr: 0.04937963819210265
epoch: 630, acc: 0.834, loss: 0.507, lr: 0.13717421124828533 Rr: 0.049289398204054306
epoch: 640, acc: 0.835, loss: 0.505, lr: 0.13531799729364005 Rr: 0.049200443972726035
epoch: 650, acc: 0.835, loss: 0.503, lr: 0.13351134846461948 Rr: 0.04911256938427884
epoch: 660, acc: 0.836, loss: 0.501, lr: 0.13175230566534915 Rr: 0.049025666203234376
epoch: 670, acc: 0.836, loss: 0.500, lr: 0.13003901170351104 Rr: 0.04893976581712262
epoch: 680, acc: 0.837, loss: 0.498, lr: 0.12836970474967907 Rr: 0.04885501454732028
epoch: 690, acc: 0.837, loss: 0.497, lr: 0.12674271229404308 Rr: 0.04877143762732927
epoch: 700, acc: 0.838, loss: 0.495, lr: 0.1251564455569462 Rr: 0.04868899164188452
epoch: 710, acc: 0.838, loss: 0.494, lr: 0.12360939431396786 Rr: 0.048607603997474186
epoch: 720, acc: 0.839, loss: 0.492, lr: 0.12210012210012208 Rr: 0.04852732691150793
epoch: 730, acc: 0.840, loss: 0.491, lr: 0.12062726176115804 Rr: 0.048448152227598386
epoch: 740, acc: 0.840, loss: 0.490, lr: 0.11918951132300357 Rr: 0.04837011145555063
epoch: 750, acc: 0.841, loss: 0.488, lr: 0.11778563015312131 Rr: 0.048293031373434256
epoch: 760, acc: 0.841, loss: 0.487, lr: 0.11641443538998836 Rr: 0.048216792848320984
epoch: 770, acc: 0.841, loss: 0.486, lr: 0.1150747986191024 Rr: 0.04814150532404577
epoch: 780, acc: 0.842, loss: 0.485, lr: 0.11376564277588169 Rr: 0.04806721843837615
epoch: 790, acc: 0.842, loss: 0.483, lr: 0.11248593925759279 Rr: 0.04799388488068481
epoch: 800, acc: 0.843, loss: 0.482, lr: 0.11123470522803114 Rr: 0.047921355530426146
epoch: 810, acc: 0.843, loss: 0.481, lr: 0.11001100110011001 Rr: 0.04784962684020312
epoch: 820, acc: 0.844, loss: 0.480, lr: 0.1088139281828074 Rr: 0.047778766418919
epoch: 830, acc: 0.844, loss: 0.479, lr: 0.1076426264800861 Rr: 0.047708798884719544
epoch: 840, acc: 0.844, loss: 0.478, lr: 0.10649627263045792 Rr: 0.04763978239240448
epoch: 850, acc: 0.845, loss: 0.477, lr: 0.1053740779768177 Rr: 0.04757164841105672
epoch: 860, acc: 0.845, loss: 0.475, lr: 0.10427528675703858 Rr: 0.04750428297066195
epoch: 870, acc: 0.845, loss: 0.474, lr: 0.10319917440660475 Rr: 0.04743761987422008
epoch: 880, acc: 0.846, loss: 0.473, lr: 0.10214504596527067 Rr: 0.04737171666723524
epoch: 890, acc: 0.846, loss: 0.472, lr: 0.10111223458038422 Rr: 0.04730657367016781
epoch: 900, acc: 0.846, loss: 0.471, lr: 0.10010010010010009 Rr: 0.04724215020885169
epoch: 910, acc: 0.847, loss: 0.470, lr: 0.09910802775024777 Rr: 0.04717839953366351
epoch: 920, acc: 0.847, loss: 0.469, lr: 0.09813542688910697 Rr: 0.04711534654126841
epoch: 930, acc: 0.848, loss: 0.468, lr: 0.09718172983479105 Rr: 0.04705294418297152
epoch: 940, acc: 0.848, loss: 0.467, lr: 0.09624639076034648 Rr: 0.04699115630195838
epoch: 950, acc: 0.848, loss: 0.466, lr: 0.09532888465204957 Rr: 0.04693003191616469
epoch: 960, acc: 0.849, loss: 0.465, lr: 0.09442870632672333 Rr: 0.04686948894996785
epoch: 970, acc: 0.850, loss: 0.464, lr: 0.09354536950420955 Rr: 0.04680952994955936
epoch: 980, acc: 0.850, loss: 0.463, lr: 0.09267840593141798 Rr: 0.04675018685097274
epoch: 990, acc: 0.850, loss: 0.462, lr: 0.09182736455463728 Rr: 0.04669143219670495
epoch: 1000, acc: 0.850, loss: 0.461, lr: 0.09099181073703366 Rr: 0.04663323116835195
epoch: 1010, acc: 0.850, loss: 0.461, lr: 0.09017132551848513 Rr: 0.04657554933387046
epoch: 1020, acc: 0.851, loss: 0.460, lr: 0.08936550491510277 Rr: 0.046518436429789735
epoch: 1030, acc: 0.851, loss: 0.459, lr: 0.08857395925597873 Rr: 0.046461945581617974
epoch: 1040, acc: 0.851, loss: 0.458, lr: 0.08779631255487269 Rr: 0.046406092749757
epoch: 1050, acc: 0.852, loss: 0.458, lr: 0.08703220191470844 Rr: 0.046350930249400 4
epoch: 1060, acc: 0.852, loss: 0.457, lr: 0.08628127696289906 Rr: 0.04629642482850592
epoch: 1070, acc: 0.852, loss: 0.456, lr: 0.0855431993156544 Rr: 0.046242541021367
epoch: 1080, acc: 0.852, loss: 0.455, lr: 0.08481764206955046 Rr: 0.046189190902431544
epoch: 1090, acc: 0.853, loss: 0.455, lr: 0.08410428931875526 Rr: 0.046136343660691255
```

In [29]:
```
epochs = range(0, 1100, 10)
plt.figure(figsize=(10, 5))
plt.plot(epochs, accuracies, label='Accuracy', marker='o', color="blue")
plt.plot(epochs, losses, 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()
```




In [31]:
```
# Validate the model - SGD

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: {data_loss:.3f}')
```

```
validation - SGD, acc: 0.837, loss: 0.449
```

## Model - SGD with momentum

In [45]:
```
"""
2 hidden layers - ReLU   - 64 neurons each
1 output layer - Softmax - 10 output neurons

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

dense1_m = Layer_Dense(X.shape[1], 64, weight_regularizer_l2=5e-4, bias_regularizer_l2
activation1_m = Activation_ReLU()
```

```
dense2_m = Layer_Dense(64, 64)
activation2_m = Activation_ReLU()
dense3_m = Layer_Dense(64, 10)
loss_activation_m = Activation_Softmax_Loss_CategoricalCrossentropy()
optimizer_m = Optimizer_SGD_Momentum(decay=0.01, momentum=0.5)
```

In [46]:
```
accuracies_m, losses_m, learning_rate_m = train_Model(1100, X, y, dense1_m, dense2_m,
```

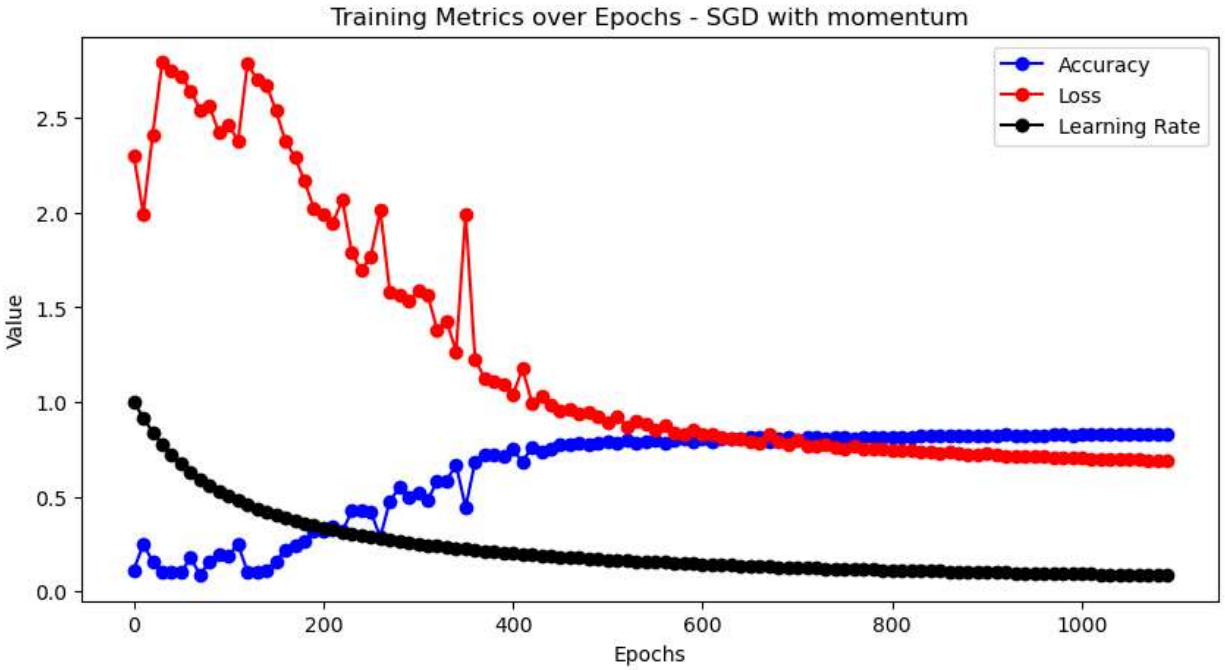
```
epoch: 0, acc: 0.112, loss: 2.305, lr: 1.0 Rr: 0.002484665630943879
epoch: 10, acc: 0.247, loss: 1.995, lr: 0.9174311926605504 Rr: 0.0026700369411095997
epoch: 20, acc: 0.158, loss: 2.413, lr: 0.8403361344537815 Rr: 0.20302829400902353
epoch: 30, acc: 0.098, loss: 2.795, lr: 0.7751937984496123 Rr: 0.4235824716767303
epoch: 40, acc: 0.100, loss: 2.748, lr: 0.7194244604316546 Rr: 0.4434652824293732
epoch: 50, acc: 0.102, loss: 2.722, lr: 0.6711409395973155 Rr: 0.4312927289779493
epoch: 60, acc: 0.175, loss: 2.643, lr: 0.628930817610063 Rr: 0.4215606725648712
epoch: 70, acc: 0.087, loss: 2.544, lr: 0.591715976331361 Rr: 0.4139536349044341
epoch: 80, acc: 0.155, loss: 2.561, lr: 0.5586592178770949 Rr: 0.4144528218211565
epoch: 90, acc: 0.196, loss: 2.421, lr: 0.5291005291005291 Rr: 0.4068437320960858
epoch: 100, acc: 0.190, loss: 2.463, lr: 0.5025125628140703 Rr: 0.399321322549662
epoch: 110, acc: 0.247, loss: 2.381, lr: 0.47846889952153115 Rr: 0.3915290564201556
epoch: 120, acc: 0.101, loss: 2.788, lr: 0.4566210045662101 Rr: 0.39291626180545136
epoch: 130, acc: 0.102, loss: 2.704, lr: 0.4366812227074236 Rr: 0.3861069498973918
epoch: 140, acc: 0.107, loss: 2.675, lr: 0.41841004184100417 Rr: 0.37955078673422415
epoch: 150, acc: 0.158, loss: 2.540, lr: 0.4016064257028112 Rr: 0.37336453450144175
epoch: 160, acc: 0.217, loss: 2.377, lr: 0.3861003861003861 Rr: 0.36757782250065757
epoch: 170, acc: 0.240, loss: 2.295, lr: 0.3717472118959108 Rr: 0.3624800623560582
epoch: 180, acc: 0.263, loss: 2.167, lr: 0.35842293906810035 Rr: 0.3577638702603576
epoch: 190, acc: 0.320, loss: 2.024, lr: 0.3460207612456747 Rr: 0.3529929680428951
epoch: 200, acc: 0.318, loss: 1.988, lr: 0.33444816053511706 Rr: 0.34937606999440546
epoch: 210, acc: 0.341, loss: 1.946, lr: 0.3236245954692557 Rr: 0.3452587854167675
epoch: 220, acc: 0.315, loss: 2.070, lr: 0.31347962382445144 Rr: 0.3413188449220515
epoch: 230, acc: 0.424, loss: 1.787, lr: 0.303951367781155 Rr: 0.33784215280880636
epoch: 240, acc: 0.429, loss: 1.696, lr: 0.2949852507374631 Rr: 0.33396291200126743
epoch: 250, acc: 0.419, loss: 1.768, lr: 0.28653295128939826 Rr: 0.3303546898852958
epoch: 260, acc: 0.286, loss: 2.014, lr: 0.2785515320334262 Rr: 0.3269569636693239
epoch: 270, acc: 0.474, loss: 1.585, lr: 0.2710027100271003 Rr: 0.32373956456555175
epoch: 280, acc: 0.547, loss: 1.563, lr: 0.2638522427440633 Rr: 0.3203815117321416
epoch: 290, acc: 0.499, loss: 1.535, lr: 0.2570694087403599 Rr: 0.31737022425326705
epoch: 300, acc: 0.519, loss: 1.585, lr: 0.2506265664160401 Rr: 0.31445814798336297
epoch: 310, acc: 0.480, loss: 1.569, lr: 0.24449877750611249 Rr: 0.3116720553008786
epoch: 320, acc: 0.583, loss: 1.379, lr: 0.23866348448687355 Rr: 0.3088965881131083
epoch: 330, acc: 0.583, loss: 1.425, lr: 0.2331002331002331 Rr: 0.3063128615837939
epoch: 340, acc: 0.667, loss: 1.267, lr: 0.2277904328018223 Rr: 0.30371297251215484
epoch: 350, acc: 0.443, loss: 1.991, lr: 0.22271714922048996 Rr: 0.3011850675589693
epoch: 360, acc: 0.684, loss: 1.225, lr: 0.2178649237472767 Rr: 0.29907317732408367
epoch: 370, acc: 0.721, loss: 1.126, lr: 0.21321961620469085 Rr: 0.29670561976914916
epoch: 380, acc: 0.723, loss: 1.106, lr: 0.20876826722338204 Rr: 0.29445036507517963
epoch: 390, acc: 0.710, loss: 1.096, lr: 0.20449897750511245 Rr: 0.2922569982517731
epoch: 400, acc: 0.749, loss: 1.039, lr: 0.2004008016032064 Rr: 0.2902112115385834
epoch: 410, acc: 0.681, loss: 1.177, lr: 0.19646365422396858 Rr: 0.28812300801086066
epoch: 420, acc: 0.763, loss: 0.995, lr: 0.19267822736030826 Rr: 0.2861876391094511
epoch: 430, acc: 0.735, loss: 1.027, lr: 0.1890359168241966 Rr: 0.2842650754266305
epoch: 440, acc: 0.753, loss: 0.985, lr: 0.1855287569573284 Rr: 0.2823637292496999
epoch: 450, acc: 0.778, loss: 0.956, lr: 0.18214936247723132 Rr: 0.28054577196405733
epoch: 460, acc: 0.773, loss: 0.960, lr: 0.17889087656529518 Rr: 0.278715556115756
epoch: 470, acc: 0.780, loss: 0.936, lr: 0.17574692442882248 Rr: 0.27703132184928675
epoch: 480, acc: 0.775, loss: 0.943, lr: 0.17271157167530224 Rr: 0.27522369593054835
epoch: 490, acc: 0.783, loss: 0.924, lr: 0.16977928692699493 Rr: 0.27356242238018247
epoch: 500, acc: 0.792, loss: 0.895, lr: 0.1669449081803005 Rr: 0.27188061829688226
epoch: 510, acc: 0.780, loss: 0.924, lr: 0.16420361247947454 Rr: 0.2702628019861075
epoch: 520, acc: 0.797, loss: 0.871, lr: 0.16155088852988692 Rr: 0.2687160051983622
epoch: 530, acc: 0.781, loss: 0.901, lr: 0.1589825119236884 Rr: 0.2670989052243736
epoch: 540, acc: 0.791, loss: 0.881, lr: 0.1564945226917058 Rr: 0.26560178798051326
epoch: 550, acc: 0.798, loss: 0.849, lr: 0.15408320493066255 Rr: 0.26410373910656204
epoch: 560, acc: 0.780, loss: 0.872, lr: 0.15174506828528073 Rr: 0.2626231078537758
epoch: 570, acc: 0.801, loss: 0.840, lr: 0.14947683109118085 Rr: 0.2611645229886741
epoch: 580, acc: 0.804, loss: 0.832, lr: 0.14727540500736377 Rr: 0.25973905435677785
epoch: 590, acc: 0.793, loss: 0.852, lr: 0.14513788098693758 Rr: 0.258341593215243
```

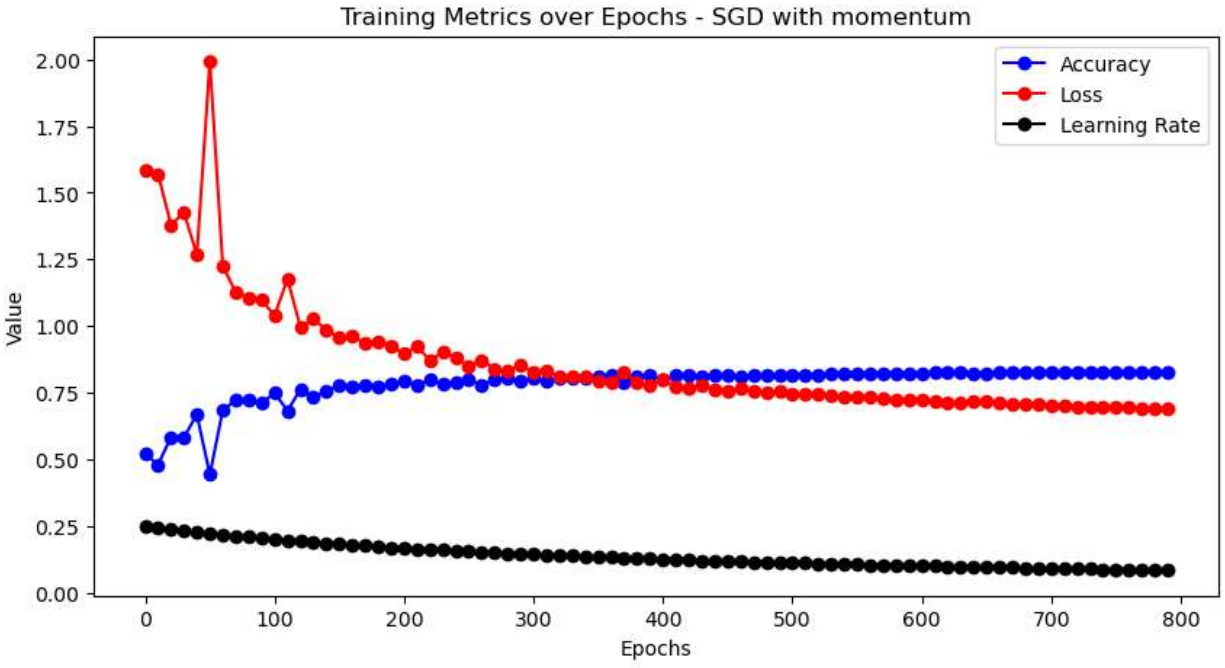
```
epoch: 600, acc: 0.803, loss: 0.829, lr: 0.14306151645207438 Rr: 0.2569695981766394
epoch: 610, acc: 0.794, loss: 0.832, lr: 0.14104372355430184 Rr: 0.25561712877598963
epoch: 620, acc: 0.804, loss: 0.811, lr: 0.13908205841446453 Rr: 0.2543025991602842
epoch: 630, acc: 0.804, loss: 0.808, lr: 0.13717421124828533 Rr: 0.2529770871639039
epoch: 640, acc: 0.803, loss: 0.809, lr: 0.13531799729364005 Rr: 0.25170638842015913
epoch: 650, acc: 0.812, loss: 0.791, lr: 0.13351134846461948 Rr: 0.2504264522279249
epoch: 660, acc: 0.813, loss: 0.787, lr: 0.13175230566534915 Rr: 0.24915821416116585
epoch: 670, acc: 0.790, loss: 0.827, lr: 0.13003901170351104 Rr: 0.2479217531971363
epoch: 680, acc: 0.809, loss: 0.791, lr: 0.12836970474967907 Rr: 0.246762625124566
epoch: 690, acc: 0.814, loss: 0.776, lr: 0.12674271229404308 Rr: 0.24557026855452627
epoch: 700, acc: 0.798, loss: 0.800, lr: 0.1251564455569462 Rr: 0.2443975282330444
epoch: 710, acc: 0.815, loss: 0.770, lr: 0.12360939431396786 Rr: 0.2432745972829236
epoch: 720, acc: 0.816, loss: 0.765, lr: 0.12210012210012208 Rr: 0.24213306945506166
epoch: 730, acc: 0.808, loss: 0.777, lr: 0.12062726176115804 Rr: 0.24101400153801375
epoch: 740, acc: 0.814, loss: 0.763, lr: 0.11918951132300357 Rr: 0.23994140069597186
epoch: 750, acc: 0.817, loss: 0.756, lr: 0.11778563015312131 Rr: 0.23885864921746194
epoch: 760, acc: 0.809, loss: 0.768, lr: 0.11641443538998836 Rr: 0.23779540275519565
epoch: 770, acc: 0.815, loss: 0.755, lr: 0.1150747986191024 Rr: 0.2367636804048836
epoch: 780, acc: 0.817, loss: 0.751, lr: 0.11376564277588169 Rr: 0.2357351463125329
epoch: 790, acc: 0.814, loss: 0.754, lr: 0.11248593925759279 Rr: 0.23472494331818622
epoch: 800, acc: 0.817, loss: 0.746, lr: 0.11123470522803114 Rr: 0.23373201483568715
epoch: 810, acc: 0.817, loss: 0.744, lr: 0.11001100110011001 Rr: 0.2327503961754034
epoch: 820, acc: 0.818, loss: 0.743, lr: 0.1088139281828074 Rr: 0.23178473107982417
epoch: 830, acc: 0.819, loss: 0.737, lr: 0.1076426264800861 Rr: 0.23083327001461593
epoch: 840, acc: 0.821, loss: 0.733, lr: 0.10649627263045792 Rr: 0.2298926837012805
epoch: 850, acc: 0.821, loss: 0.732, lr: 0.1053740779768177 Rr: 0.2289648198614936
epoch: 860, acc: 0.819, loss: 0.733, lr: 0.10427528675703858 Rr: 0.22805213180539827
epoch: 870, acc: 0.821, loss: 0.728, lr: 0.10319917440660475 Rr: 0.22715376019207978
epoch: 880, acc: 0.823, loss: 0.724, lr: 0.10214504596527067 Rr: 0.2262649482417491
epoch: 890, acc: 0.822, loss: 0.723, lr: 0.10111223458038422 Rr: 0.22538812411038456
epoch: 900, acc: 0.821, loss: 0.725, lr: 0.10010010010010009 Rr: 0.22452618617987635
epoch: 910, acc: 0.824, loss: 0.718, lr: 0.09910802775024777 Rr: 0.22367870122260952
epoch: 920, acc: 0.826, loss: 0.714, lr: 0.09813542688910697 Rr: 0.22283634800871477
epoch: 930, acc: 0.825, loss: 0.713, lr: 0.09718172983479105 Rr: 0.22200444086160323
epoch: 940, acc: 0.823, loss: 0.715, lr: 0.09624639076034648 Rr: 0.22118593270828665
epoch: 950, acc: 0.822, loss: 0.716, lr: 0.09532888465204957 Rr: 0.22038335989268035
epoch: 960, acc: 0.825, loss: 0.710, lr: 0.09442870632672333 Rr: 0.21958800956685395
epoch: 970, acc: 0.826, loss: 0.707, lr: 0.09354536950420955 Rr: 0.21880098074855067
epoch: 980, acc: 0.826, loss: 0.705, lr: 0.09267840593141798 Rr: 0.21802315264812674
epoch: 990, acc: 0.826, loss: 0.706, lr: 0.09182736455463728 Rr: 0.2172565084095476
epoch: 1000, acc: 0.826, loss: 0.703, lr: 0.09099181073703366 Rr: 0.21650124512087596
epoch: 1010, acc: 0.828, loss: 0.700, lr: 0.09017132551848513 Rr: 0.21575329773077795
epoch: 1020, acc: 0.829, loss: 0.698, lr: 0.08936550491510277 Rr: 0.21501268127710074
epoch: 1030, acc: 0.829, loss: 0.696, lr: 0.08857395925597873 Rr: 0.21428086414401543
epoch: 1040, acc: 0.827, loss: 0.697, lr: 0.08779631255487269 Rr: 0.21355882675148047
epoch: 1050, acc: 0.827, loss: 0.697, lr: 0.08703220191470844 Rr: 0.21284812869033237
epoch: 1060, acc: 0.828, loss: 0.694, lr: 0.08628127696289906 Rr: 0.2121446918542278
epoch: 1070, acc: 0.828, loss: 0.692, lr: 0.0855431993156544 Rr: 0.21144855688464148
epoch: 1080, acc: 0.828, loss: 0.691, lr: 0.08481764206955046 Rr: 0.21076086572437666
epoch: 1090, acc: 0.829, loss: 0.690, lr: 0.08410428931875526 Rr: 0.21008169349813444
```

In [47]:

```
epochs = range(0, 1100, 10)
plt.figure(figsize=(10, 5))
plt.plot(epochs, accuracies_m, label='Accuracy', marker='o', color="blue")
plt.plot(epochs, losses_m, label='Loss', marker='o', color='red')
plt.plot(epochs, learning_rate_m, label='Learning Rate', marker='o', color='black')
plt.title('Training Metrics over Epochs - SGD with 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_m[30:], label='Accuracy', marker='o', color="blue")
plt.plot(epochs, losses_m[30:], label='Loss', marker='o', color='red')
plt.plot(epochs, learning_rate_m[30:], label='Learning Rate', marker='o', color='black
plt.title('Training Metrics over Epochs - SGD with momentum')
plt.xlabel('Epochs')
plt.ylabel('Value')
plt.legend()
plt.savefig('training_metrics_plot.png')
plt.show()
```




In [51]:
```
# Validate the model - SGD with momentum
```

```
dense1_m.forward(X_test)
activation1_m.forward(dense1_m.output)
dense2_m.forward(activation1_m.output)
activation2_m.forward(dense2_m.output)
dense3_m.forward(activation2_m.output)
data_loss_m = loss_activation_m.forward(dense3_m.output, y_test)

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

```
validation - SGD, acc: 0.812, loss: 0.527
```

## Model - Adam

In [58]:
```
"""
2 hidden layers - ReLU   - 64 neurons each
1 output layer - Softmax - 10 output neurons

learning rate decay - 1e-5
momentum = 0.5
l2 regularization - lambda - 5e-4 (0.0005)
"""

dense1_a = Layer_Dense(X.shape[1], 64, weight_regularizer_l2=5e-4, bias_regularizer_l2
activation1_a = Activation_ReLU()
dense2_a = Layer_Dense(64, 64)
activation2_a = Activation_ReLU()
dense3_a = Layer_Dense(64, 10)
loss_activation_a = Activation_Softmax_Loss_CategoricalCrossentropy()

optimizer_a = Optimizer_Adam(learning_rate=0.05, decay=1e-4)
```

In [ ]:
```
accuracies_a, losses_a, learning_rate_a = train_Model(1100, X, y, dense1_a, dense2_a,
```

```
epoch: 0, acc: 0.036, loss: 2.305, lr: 0.05 Rr: 0.0024967681283024293
epoch: 10, acc: 0.575, loss: 1.648, lr: 0.04995504046358278 Rr: 0.4112581017789141
epoch: 20, acc: 0.729, loss: 1.324, lr: 0.049905180157700374 Rr: 0.5820227704218323
epoch: 30, acc: 0.742, loss: 1.197, lr: 0.04985541928407619 Rr: 0.5031808012661925
epoch: 40, acc: 0.792, loss: 0.980, lr: 0.04980575754557227 Rr: 0.410429526293498
epoch: 50, acc: 0.818, loss: 0.794, lr: 0.04975619464623346 Rr: 0.3003797224226835
epoch: 60, acc: 0.800, loss: 0.729, lr: 0.04970673029128144 Rr: 0.20649047696751544
epoch: 70, acc: 0.789, loss: 0.746, lr: 0.049657364187108956 Rr: 0.16614415347804806
epoch: 80, acc: 0.840, loss: 0.640, lr: 0.04960809604127394 Rr: 0.19558278126826878
epoch: 90, acc: 0.698, loss: 1.214, lr: 0.04955892556249381 Rr: 0.2851761605263652
epoch: 100, acc: 0.613, loss: 1.436, lr: 0.049509852460639665 Rr: 0.43178923448012074
epoch: 110, acc: 0.602, loss: 1.742, lr: 0.04946087644673064 Rr: 0.46683331646790854
epoch: 120, acc: 0.642, loss: 1.545, lr: 0.04941199723292816 Rr: 0.5535333853491166
epoch: 130, acc: 0.697, loss: 1.484, lr: 0.04936321453253037 Rr: 0.5521458306075728
epoch: 140, acc: 0.733, loss: 1.246, lr: 0.04931452805996647 Rr: 0.5059805845937395
epoch: 150, acc: 0.796, loss: 0.963, lr: 0.04926593753079122 Rr: 0.3968383342197349
epoch: 160, acc: 0.817, loss: 0.782, lr: 0.0492174426616793 Rr: 0.2790234189037479
epoch: 170, acc: 0.833, loss: 0.665, lr: 0.04916904317041991 Rr: 0.20228493686894095
epoch: 180, acc: 0.788, loss: 0.706, lr: 0.049120738775911194 Rr: 0.14775163260282634
epoch: 190, acc: 0.828, loss: 0.637, lr: 0.049072529198154885 Rr: 0.16006255276481574
epoch: 200, acc: 0.839, loss: 0.588, lr: 0.04902441415825081 Rr: 0.1494010104590642
epoch: 210, acc: 0.834, loss: 0.633, lr: 0.048976393378391624 Rr: 0.16722558329818826
epoch: 220, acc: 0.848, loss: 0.582, lr: 0.04892846658185732 Rr: 0.16022700865375197
epoch: 230, acc: 0.854, loss: 0.530, lr: 0.04888063349301008 Rr: 0.1286453546736015
epoch: 240, acc: 0.842, loss: 0.525, lr: 0.0488328938372888 Rr: 0.10204023993857016
epoch: 250, acc: 0.823, loss: 0.589, lr: 0.04878524734120403 Rr: 0.12024301494888323
epoch: 260, acc: 0.760, loss: 0.977, lr: 0.04873769373233258 Rr: 0.2652797575280533
epoch: 270, acc: 0.788, loss: 1.005, lr: 0.0486902327393125 Rr: 0.3900068594806866
epoch: 280, acc: 0.824, loss: 0.833, lr: 0.048642864091837726 Rr: 0.334380989323197
epoch: 290, acc: 0.805, loss: 0.745, lr: 0.04859558752065313 Rr: 0.2290191370827202
epoch: 300, acc: 0.853, loss: 0.609, lr: 0.04854840275754928 Rr: 0.19995211061454998
epoch: 310, acc: 0.860, loss: 0.528, lr: 0.04850130953535746 Rr: 0.14569404655646412
epoch: 320, acc: 0.278, loss: 3.919, lr: 0.04845430758794457 Rr: 0.399233450038623
epoch: 330, acc: 0.307, loss: 3.192, lr: 0.048407396650208157 Rr: 1.2529244185813755
epoch: 340, acc: 0.303, loss: 3.135, lr: 0.04836057645807138 Rr: 1.339577694569845
epoch: 350, acc: 0.177, loss: 3.179, lr: 0.04831384674847812 Rr: 1.045038447714114
epoch: 360, acc: 0.243, loss: 2.885, lr: 0.04826720725938797 Rr: 0.8569594432551194
epoch: 370, acc: 0.232, loss: 2.775, lr: 0.04822065772977144 Rr: 0.7662425753682547
epoch: 380, acc: 0.267, loss: 2.657, lr: 0.048174197899604976 Rr: 0.6594996419072967
epoch: 390, acc: 0.261, loss: 2.528, lr: 0.04812782750986621 Rr: 0.60013112950332
epoch: 400, acc: 0.279, loss: 2.387, lr: 0.04808154630252909 Rr: 0.5612463448688086
epoch: 410, acc: 0.338, loss: 2.178, lr: 0.04803535402055914 Rr: 0.45836741054526436
epoch: 420, acc: 0.380, loss: 2.419, lr: 0.04798925040790863 Rr: 0.3395198330322595
epoch: 430, acc: 0.115, loss: 2.918, lr: 0.04794323520951194 Rr: 0.4525159740014834
epoch: 440, acc: 0.114, loss: 3.088, lr: 0.04789730817128077 Rr: 0.715771552678741
epoch: 450, acc: 0.141, loss: 2.849, lr: 0.04785146904004953 Rr: 0.6481066548277531
epoch: 460, acc: 0.124, loss: 2.750, lr: 0.04780571756382063 Rr: 0.4874640138275878
epoch: 470, acc: 0.167, loss: 2.524, lr: 0.04776005349125992 Rr: 0.372002379035451
epoch: 480, acc: 0.171, loss: 2.414, lr: 0.047714476572192 Rr: 0.28206854274700704
epoch: 490, acc: 0.214, loss: 2.366, lr: 0.0476689865573458 Rr: 0.33078825761856623
epoch: 500, acc: 0.174, loss: 2.310, lr: 0.04762358319839985 Rr: 0.2931932551923462
epoch: 510, acc: 0.141, loss: 2.505, lr: 0.04757826624797924 Rr: 0.23471087886531788
epoch: 520, acc: 0.141, loss: 2.464, lr: 0.04753303545964445 Rr: 0.2486279896196702
```

In [ ]:

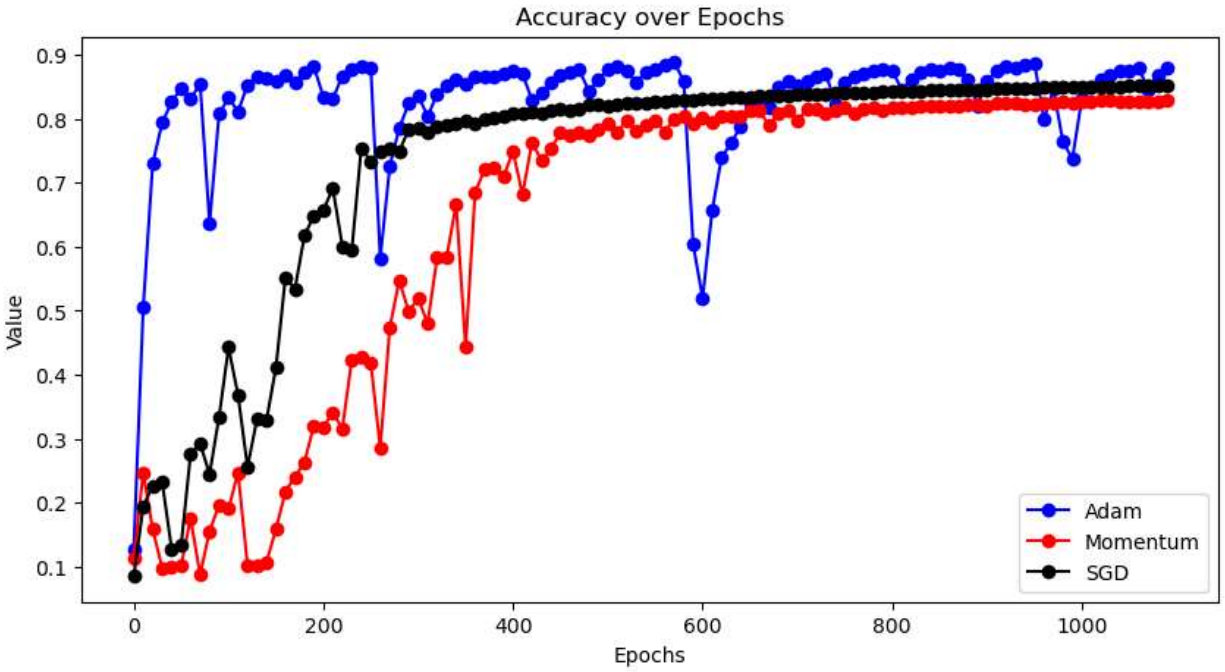
```
epochs = range(0, 1100, 10)
plt.figure(figsize=(10, 5))
plt.plot(epochs, accuracies_a, label='Accuracy', marker='o', color="blue")
plt.plot(epochs, losses_a, label='Loss', marker='o', color='red')
plt.plot(epochs, learning_rate_a, label='Learning Rate', marker='o', color='black')
plt.title('Training Metrics over Epochs - SGD with 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_a[30:], label='Accuracy', marker='o', color="blue")
plt.plot(epochs, losses_a[30:], label='Loss', marker='o', color='red')
plt.plot(epochs, learning_rate_a[30:], label='Learning Rate', marker='o', color='black
plt.title('Training Metrics over Epochs - SGD with momentum')
plt.xlabel('Epochs')
plt.ylabel('Value')
plt.legend()
plt.savefig('training_metrics_plot.png')
plt.show()
```

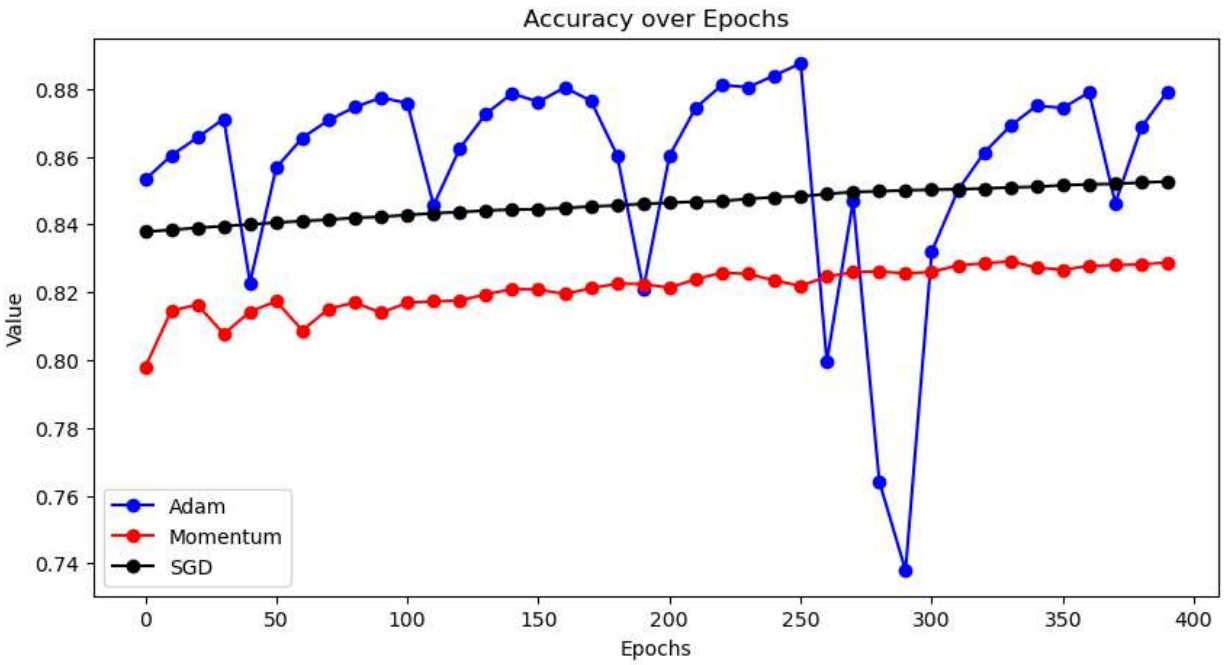
In [55]:
```
# Validate the model - Adam

dense1_a.forward(X_test)
activation1_a.forward(dense1_a.output)
dense2_a.forward(activation1_a.output)
activation2_a.forward(dense2_a.output)
dense3_a.forward(activation2_a.output)
data_loss = loss_activation_a.forward(dense3_a.output, y_test)

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

```
validation - SGD, acc: 0.864, loss: 0.387
```

In [56]:
```
epochs = range(0, 1100, 10)
plt.figure(figsize=(10, 5))
plt.plot(epochs, accuracies_a, label='Adam', marker='o', color="blue")
plt.plot(epochs, accuracies_m, label='Momentum', marker='o', color='red')
plt.plot(epochs, accuracies, label='SGD', marker='o', color='black')
plt.title('Accuracy over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.savefig('training_metrics_plot.png')
plt.show()
```

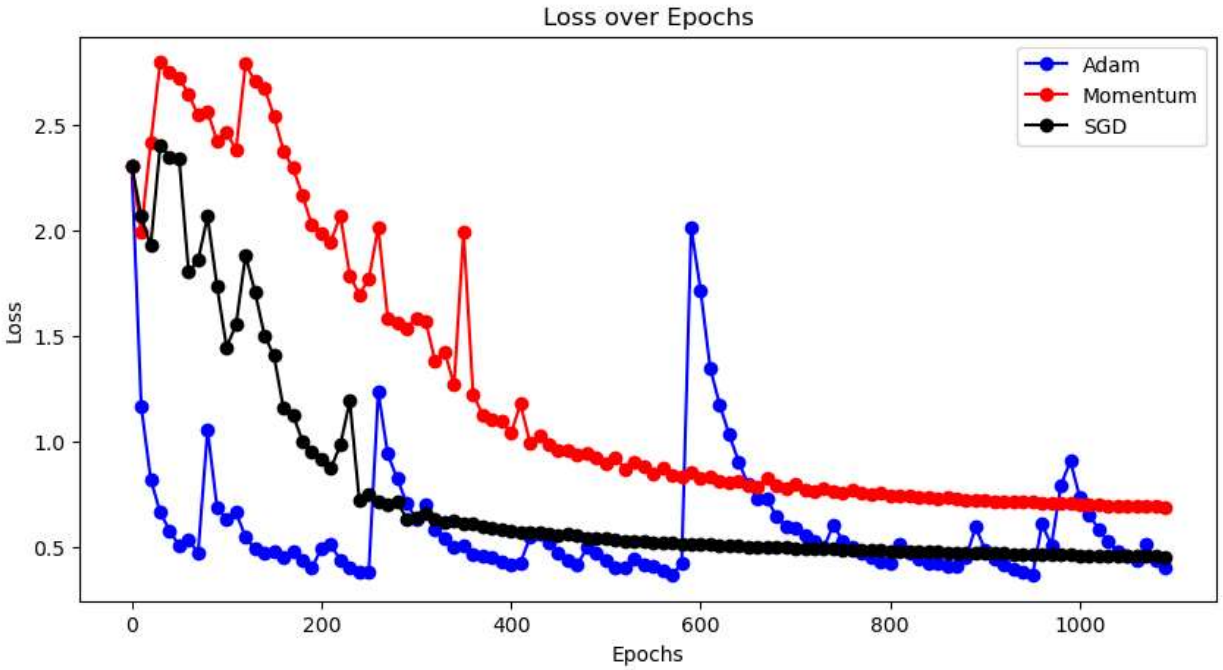



In [57]:
```
epochs = range(0, 400, 10)
plt.figure(figsize=(10, 5))
plt.plot(epochs, accuracies_a[70:], label='Adam', marker='o', color="blue")
plt.plot(epochs, accuracies_m[70:], label='Momentum', marker='o', color='red')
plt.plot(epochs, accuracies[70:], label='SGD', marker='o', color='black')
plt.title('Accuracy over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.savefig('training_metrics_plot.png')
plt.show()
```



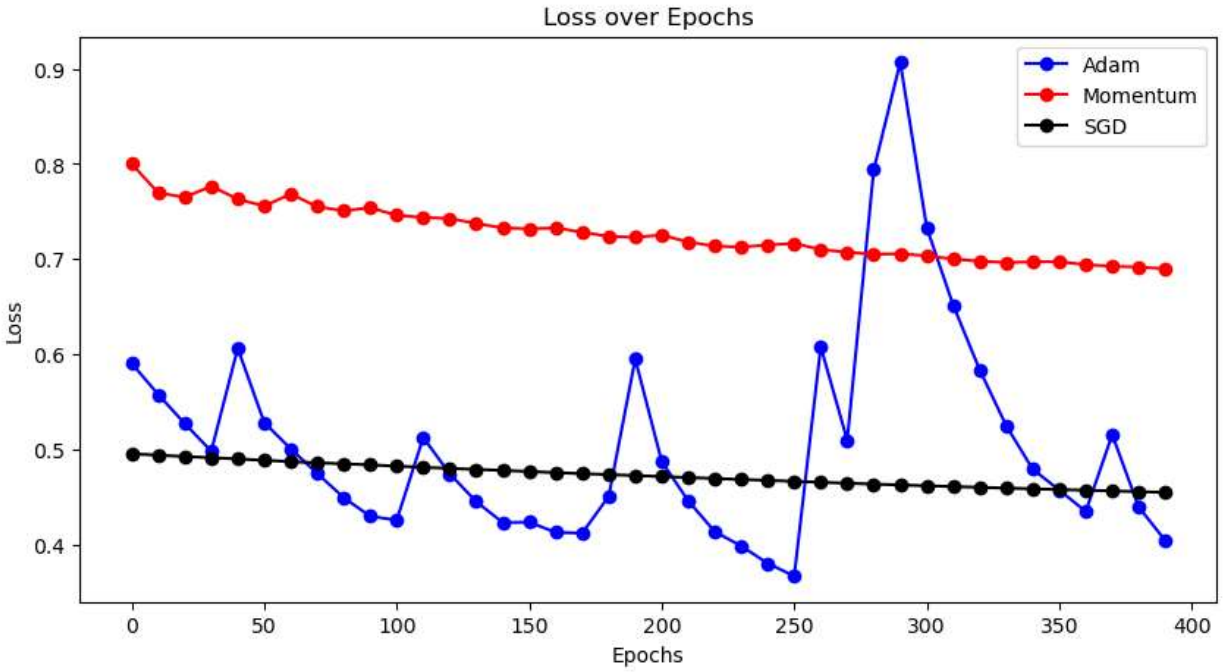

In [60]:
```
epochs = range(0, 1100, 10)
plt.figure(figsize=(10, 5))
plt.plot(epochs, losses_a, label='Adam', marker='o', color="blue")
plt.plot(epochs, losses_m, label='Momentum', marker='o', color='red')
```

```
plt.plot(epochs, losses, label='SGD', marker='o', color='black')
plt.title('Loss over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.savefig('training_metrics_plot.png')
plt.show()
```




In [61]:

```
epochs = range(0, 400, 10)
plt.figure(figsize=(10, 5))
plt.plot(epochs, losses_a[70:], label='Adam', marker='o', color="blue")
plt.plot(epochs, losses_m[70:], label='Momentum', marker='o', color='red')
plt.plot(epochs, losses[70:], label='SGD', marker='o', color='black')
plt.title('Loss over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.savefig('training_metrics_plot.png')
plt.show()
```


Loss over Epochs
Adam
Momentum
SGD
Loss
Epochs
0.9
0.8
0.7
0.6
0.5
0.4
0
50
100
150
200
250
300
350
400

In [ ]: