در این بخش شما را به چالش می‌کشد تا با پیاده‌سازی تکنیک‌های پیشرفته‌تر، عملکرد و درک خود از شبکه‌های عصبی را عمیق‌تر کنید.

۱. پیاده‌سازی الگوریتم بهینه‌ساز مومنتوم و مقایسه سرعت همگرایی

الگوریتم بهینه‌ساز Momentum را پیاده‌سازی کنید و سرعت همگرایی آن را با گرادیان کاهشی استاندارد در بخش ۳ مقایسه نمایید. نمودار فاصله تا بهینگی یا مقدار تابع هزینه در هر دوره را رسم کنید.

برنامه ۱: پیاده‌سازی Momentum

```
def update_params(self, grads_w, grads_b):
    for i in range(len(self.weights)):
        self.vel_w[i] = self.momentum * self.vel_w[i] - self.lr * grads_w[i]
        self.vel_b[i] = self.momentum * self.vel_b[i] - self.lr * grads_b[i]
        self.weights[i] += self.vel_w[i]
        self.biases[i] += self.vel_b[i]
```

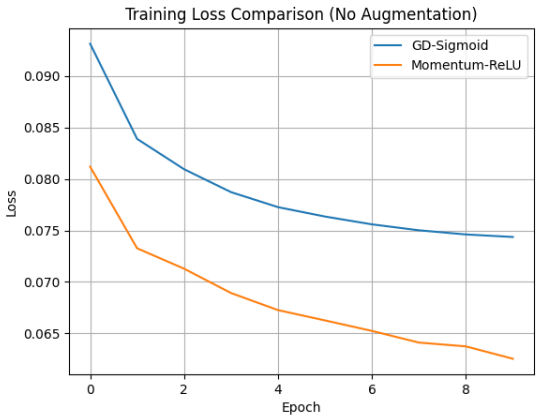
۲. رسم نمودارهای زیان و دقت

نمودارهای زیان (loss) و دقت (accuracy) آموزش را در طول دوره‌ها برای بخش ۳ رسم کنید. در صورت پیاده‌سازی تقسیم اعتبارسنجی (cross-validation split) نمودارهای زیان و دقت اعتبارسنجی را نیز اضافه نمایید.

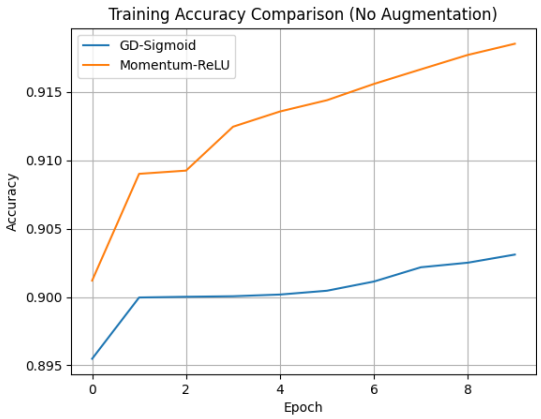
برنامه ۲: رسم نمودار زیان و دقت آموزش و اعتبارسنجی

```
# Plot Loss
    plt.figure()
    for name, hist in results.items():
        plt.plot(hist['train_loss'], label=name)
    plt.title('Training Loss Comparison (No Augmentation)')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()
    plt.grid()
    plt.show()

    # Plot Accuracy
    plt.figure()
    for name, hist in results.items():
        plt.plot(hist['train_acc'], label=name)
    plt.title('Training Accuracy Comparison (No Augmentation)')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.grid()
    plt.show()
```



(آ) نمودار زیان (Loss)



(ب) نمودار دقت (Accuracy)

شکل ۱: مقایسهٔ زیان و دقت آموزش و اعتبارسنجی در طول دوره‌ها

۳. استفاده از تابع فعال‌سازی ReLU و مقایسه با سیگموئید

در لایه‌های پنهان شبکه خود از تابع ReLU استفاده کنید و عملکرد آن را با سیگموئید مقایسه نمایید. مقداردهی اولیه وزن‌ها را با He Initialization تنظیم کنید.

برنامه ۳: مقداردهی اولیه‌ی Relu

```
def init_weights(self):
        self.weights = []
        self.biases = []
        for i in range(len(self.layer_dims) - 1):
            W = np.random.randn(self.layer_dims[i], self.layer_dims[i + 1]) *
    np.sqrt(2. / self.layer_dims[i])
            b = np.zeros((1, self.layer_dims[i + 1]))
            self.weights.append(W)
            self.biases.append(b)
```

۴. ساخت کلاس مدولار NeuralNetwork()

یک کلاس مدولار NeuralNetwork() بسازید که امکانات زیر را فراهم کند:

- تعریف تعداد دلخواه لایه با اندازه‌های مشخص
- انتخاب توابع فعال‌سازی مختلف برای هر لایه (sigmoid, ReLU, tanh, SGD, momentum, Adam)
- پیاده‌سازی بهینه‌سازهای مختلف

برنامه ۴: کلاس Neural Network

```
class NeuralNetwork:
    def __init__(self, layer_dims, activations, optimizer_cfg=None):
        self.layer_dims = layer_dims
        self.activations = activations
        self.lr = optimizer_cfg.get('lr', 0.01)
        self.momentum = optimizer_cfg.get('momentum', 0)
        self.init_weights()
        self.init_velocity()
```

```
        self.history = {'train_loss': [], 'train_acc': []}

    def init_weights(self):
        self.weights = []
        self.biases = []
        for i in range(len(self.layer_dims) - 1):
            W = np.random.randn(self.layer_dims[i], self.layer_dims[i + 1]) *
    np.sqrt(2. / self.layer_dims[i])
            b = np.zeros((1, self.layer_dims[i + 1]))
            self.weights.append(W)
            self.biases.append(b)

    def init_velocity(self):
        self.vel_w = [np.zeros_like(W) for W in self.weights]
        self.vel_b = [np.zeros_like(b) for b in self.biases]

    def forward(self, X):
        A = X
        self.zs = []
        self.activs = [X]
        for i in range(len(self.weights)):
            Z = A @ self.weights[i] + self.biases[i]
            self.zs.append(Z)
            if i == len(self.weights) - 1:
                A = sigmoid(Z)
            else:
                act_fn, _ = ACTIVATIONS[self.activations[i]]
                A = act_fn(Z)
            self.activs.append(A)
        return A, self.zs

    def backward(self, X, y, output):
        m = X.shape[0]
        grads_w = [0] * len(self.weights)
        grads_b = [0] * len(self.biases)

        delta = output - y.reshape(-1, 1)

        for i in reversed(range(len(self.weights))):
            A_prev = self.activs[i]
            grads_w[i] = A_prev.T @ delta / m
            grads_b[i] = np.sum(delta, axis=0, keepdims=True) / m

            if i != 0:
                _, d_act = ACTIVATIONS[self.activations[i - 1]]
                delta = (delta @ self.weights[i].T) * d_act(self.zs[i - 1])

        return grads_w, grads_b

    def update_params(self, grads_w, grads_b):
        for i in range(len(self.weights)):
            self.vel_w[i] = self.momentum * self.vel_w[i] - self.lr *
    grads_w[i]
            self.vel_b[i] = self.momentum * self.vel_b[i] - self.lr *
    grads_b[i]
            self.weights[i] += self.vel_w[i]
```

```
            self.biases[i] += self.vel_b[i]

    def train(self, train_loader, epochs=10, print_freq=1):
        for epoch in range(epochs):
            all_loss = []
            all_acc = []
            for x_batch, y_batch in train_loader:
                x_batch = x_batch.numpy()
                y_batch = y_batch.numpy()
                output, _ = self.forward(x_batch)
                loss = np.mean((output - y_batch.reshape(-1, 1)) ** 2)
                preds = (output >= 0.5).astype(int).flatten()
                acc = np.mean(preds == y_batch)

                grads_w, grads_b = self.backward(x_batch, y_batch, output)
                self.update_params(grads_w, grads_b)

                all_loss.append(loss)
                all_acc.append(acc)

            avg_loss = np.mean(all_loss)
            avg_acc = np.mean(all_acc)
            self.history['train_loss'].append(avg_loss)
            self.history['train_acc'].append(avg_acc)

            if epoch % print_freq == 0:
                print(f"Epoch {epoch+1}/{epochs} - Loss: {avg_loss:.4f} - Acc:
{avg_acc:.4f}")
```

۵. آموزش و مقایسه مدل‌ها

مدل طبقه‌بندی خود را با انتخاب ترکیب‌های مختلف از موارد بالا آموزش دهید و نتایج آن‌ها را مقایسه نمایید. می‌توانید جدول یا نمودار مقایسه دقت و زمان آموزش ایجاد کنید.

برنامه ۵: آموزش و مقایسه مدل‌ها

```
if __name__ == '__main__':
    x_train, y_train, x_test, y_test = load_data_no_augmentation()

    configs = [
        {'name': 'GD-Sigmoid', 'activations': ['sigmoid', 'sigmoid'], 'opt':
{'lr': 0.01, 'momentum': 0}},
        {'name': 'Momentum-ReLU', 'activations': ['relu', 'sigmoid'], 'opt':
{'lr': 0.01, 'momentum': 0.9}},
    ]

    trained_models = {}
    results = {}

    for cfg in configs:
        print(f"\n== Training {cfg['name']} Without Augmentation ==")
        model = NeuralNetwork(layer_dims=[3072, 64, 1],
activations=cfg['activations'], optimizer_cfg=cfg['opt'])
        train_loader = SimpleLoader(x_train, y_train)
        model.train(train_loader, epochs=10, print_freq=2)
        results[cfg['name']] = model.history
```

```
        trained_models[cfg['name']] = model

    # Plot Loss
    plt.figure()
    for name, hist in results.items():
        plt.plot(hist['train_loss'], label=name)
    plt.title('Training Loss Comparison (No Augmentation)')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()
    plt.grid()
    plt.show()

    # Plot Accuracy
    plt.figure()
    for name, hist in results.items():
        plt.plot(hist['train_acc'], label=name)
    plt.title('Training Accuracy Comparison (No Augmentation)')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.grid()
    plt.show()

    # Evaluation
    print("\n-- Test Set Evaluation --")
    for name, model in trained_models.items():
        yh, _ = model.forward(x_test)
        yp = (yh >= 0.5).astype(int).flatten()
        print(f"\n{name}:\n", confusion_matrix(y_test, yp))
        print(classification_report(y_test, yp, digits=4))
```