作业：从零开始构建三层神经网络分类器，实现图像分类

一．模型介绍

```python
# 模型模块
class NeuralNetwork:
    def __init__(self, input_size, hidden_size, output_size, activation='relu'):
        # 初始化权重和偏置
        self.params = {
            'W1': np.random.randn(input_size, hidden_size) * np.sqrt(2. / input_size),
            'b1': np.zeros((1, hidden_size)),
            'W2': np.random.randn(hidden_size, output_size) * np.sqrt(2. / hidden_size),
            'b2': np.zeros((1, output_size))
        }
        if activation == 'relu':
            self.activation_function = relu
            self.activation_derivative = relu_derivative
```

模型分为三层：input 层、hidden 层和 output 层，其中 output 层层数要根据数据的类数给定。

模型的计算和反向传播计算梯度的函数如下图所示：

```python
# 向前传播，激活函数默认设置为relu
def forward(self, X):
    Z1 = np.dot(X, self.params['W1']) + self.params['b1']
    A1 = self.activation_function(Z1)
    Z2 = np.dot(A1, self.params['W2']) + self.params['b2']
    A2 = softmax(Z2)
    self.cache = {'Z1': Z1, 'A1': A1, 'Z2': Z2, 'A2': A2}
    return A2

# 通过反向传播计算给定损失的梯度
def backward(self, X, y):
    m = y.shape[0]
    output_error = self.cache['A2'] - y
    dW2 = np.dot(self.cache['A1'].T, output_error) / m
    db2 = np.sum(output_error, axis=0, keepdims=True) / m
    hidden_error = np.dot(output_error, self.params['W2'].T) * self.activation_derivative(self.cache['Z1'])
    dW1 = np.dot(X.T, hidden_error) / m
    db1 = np.sum(hidden_error, axis=0, keepdims=True) / m
    return {'W1': dW1, 'b1': db1, 'W2': dW2, 'b2': db2}
```

训练部分使用交叉熵损失：

```python
# 交叉熵损失
def cross_entropy(predictions, labels):
    return -np.sum(labels * np.log(predictions + 1e-10)) / predictions.shape[0]
```

0.6s

通过 SGD 优化器、L2 正则化和验证集自动寻优来寻找最优参数：

```python
# 训练
def train(model, X_train, y_train, X_val, y_val, epochs, batch_size, learning_rate, lambda_reg=0.001, lr_decay=0.95):
    history = {'train_loss': [], 'val_loss': [], 'train_acc': [], 'val_acc': [], 'grad_norms': []}
    best_val_acc = 0
    best_params = {}
    # SGD优化器
    for epoch in range(epochs):
        current_lr = learning_rate * (lr_decay ** epoch)
        permutation = np.random.permutation(X_train.shape[0])
        X_train_shuffled = X_train[permutation]
        y_train_shuffled = y_train[permutation]
        epoch_grads = []
        for i in range(0, X_train.shape[0], batch_size):
            X_batch = X_train_shuffled[i:i + batch_size]
            y_batch = y_train_shuffled[i:i + batch_size]
            outputs = model.forward(X_batch)
            loss = cross_entropy(outputs, y_batch)
            grads = model.backward(X_batch, y_batch)
            model.update_params(grads, current_lr, lambda_reg)
            grad_norm = np.linalg.norm(np.concatenate([grad.flatten() for grad in grads.values()]))
            epoch_grads.append(grad_norm)
        history['grad_norms'].append(np.mean(epoch_grads))
        train_loss, train_acc = evaluate(model, X_train, y_train)
        val_loss, val_acc = evaluate(model, X_val, y_val)
        history['train_loss'].append(train_loss)
        history['train_acc'].append(train_acc)
        history['val_loss'].append(val_loss)
        history['val_acc'].append(val_acc)
        print(f'Epoch {epoch+1}/{epochs}, Train Loss: {train_loss:.4f}, Validation Loss: {val_loss:.4f}, Validation Accuracy: {val_acc:.4f}')

        # 根据验证集指标自动保存最优的模型权重
        if val_acc > best_val_acc:
            best_val_acc = val_acc
            best_params = {k: v.copy() for k, v in model.params.items()}
    model.params = best_params
    return history
```

参数查找环节实现了调节学习率、隐藏层大小、正则化强度等超参数:

```python
# 网格搜索
def grid_search(X_train, y_train, X_val, y_val, learning_rates, hidden_sizes, lambdas, input_size, output_size, batch_size, epochs=50):

    # 调节学习率、隐藏层大小、正则化强度
    for lr in learning_rates:
        for hidden_size in hidden_sizes:
            for lambda_reg in lambdas:
                model = NeuralNetwork(input_size, hidden_size, output_size)
                history = train(model, X_train, y_train, X_val, y_val, epochs, batch_size, lr, lambda_reg)
                current_val_acc = max(history['val_acc'])

                # 在验证集上评价模型并保存最佳模型
                if current_val_acc > best_overall_val_acc:
                    best_overall_val_acc = current_val_acc
                    best_overall_params = model.params.copy()
                    best_lr = lr
                    best_hidden_size = hidden_size
                    best_lambda_reg = lambda_reg

    return best_overall_params
```

最后是测试部分，支持导入训练好的模型，并输出在测试集上的分类准确率。

```python
def test_model(model, X_test, y_test):
    test_output = model.forward(X_test)
    test_accuracy = np.mean(np.argmax(test_output, axis=1) == np.argmax(y_test, axis=1))
    print(f'Test Accuracy: {test_accuracy}')
```

二．数据集介绍

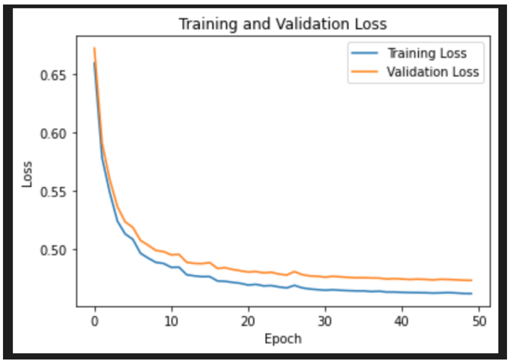
本实验使用的数据集是 Fashion-MNIST 数据集。Fashion-MNIST 是一个经典的机器学习数据集，用于图像分类任务，包含了来自 10 个不同类别的 70,000 张灰度图像，每个类别包含了 7,000 张图像。这些图像代表了服饰和配件的不同类别，例如 T 恤、裤子、运动鞋等。Fashion-MNIST 是对经典 MNIST 数据集的一个现代化替代品，它的图像更加复杂，更贴近真实世界的场景。这个数据集被广泛用于训练和测试机器学习模型，尤其是在图像分类和深度学习领域。数据集中每个图像都是 28x28 像素大小的，

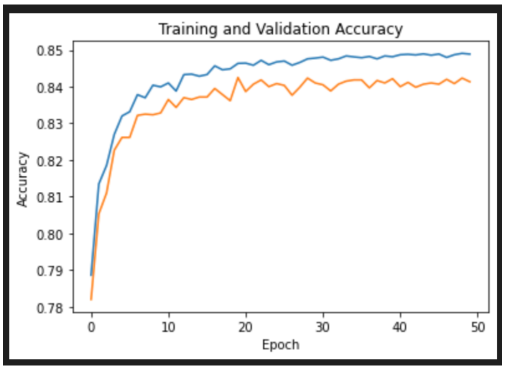
图像较小因此适合用于快速原型设计和实验。

三．实验结果

设置模型训练参数如下：input_size = 784，hidden_size = 128，output_size = 10，epochs = 50，batch_size = 64。

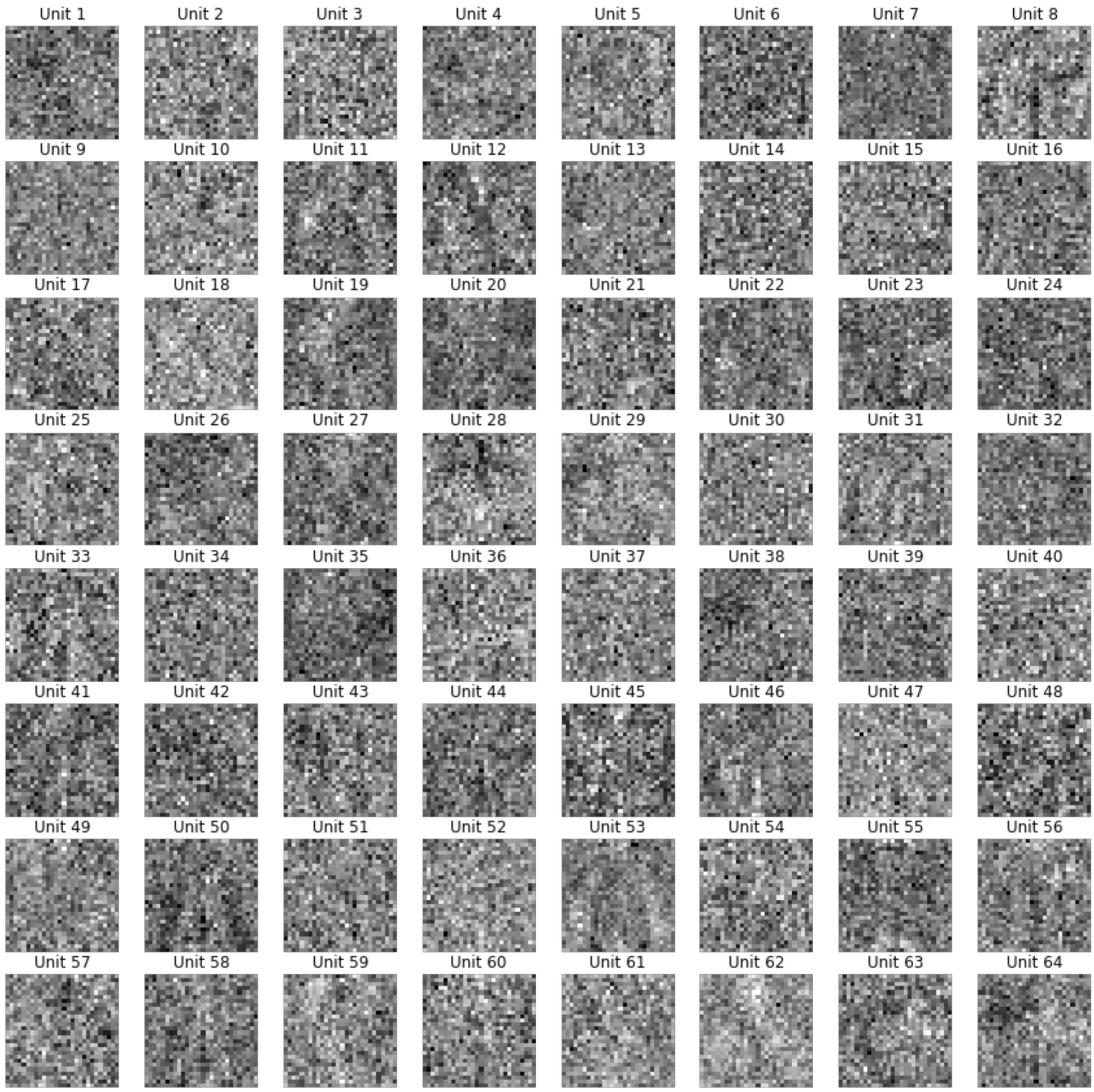
训练 50 个 epoch 后，根据网格搜索参数最优的在训练集和验证集上的 loss 曲线和验证集上的 accuracy 曲线如下所示。模型测试的正确率为 0.8645。





四．参数可视化

将 64 个隐藏层的参数可视化，可以得到下图所示的 64 个图像。其中，较亮的像素表示权重的值较大。



模型权重提取链接:

https://pan.baidu.com/s/1uNx4UpKVOzHt8KzfovBi7g?pwd=4gmj 提取码: 4gmj

GitHub repo 链接：https://github.com/dlwlrmaabaaba/homework1/tree/main