深度学习与神经网络实验报告 1

信科 2201 孙彧旸

2022310220107, frontsea040320@gmail.com

摘要 本实验实现了一个简单的神经网络模型用于实现 MINST 手写数字数据集的识别任务，在原代码的基础上对网络结构进行简要修改，增加了隐藏层并支持残差连接，并增加了衰减学习率功能，重新修改代码结构，将模型定义、训练过程与测试过程分离，提供了简单的权重保存与加载机制，并支持使用预训练权重进行训练或断点重续，提供了 lr = 1e-4/epoch = 50 的预训练权重，对结果进行可视化，支持 loss 与 accuracy 曲线图绘制，测试过程支持可视化与 groundtruth 进行对比，最终训练出的神经网络达到了 84% 的准确率，主要模型与代码已在附录给出，完整实现可在 Github 获取：https://github.com/frontsea320/Deep-Learning-and-Neural-Networks-Lab-Session/tree/main/ex1

1 实验目标

本实验旨在构建并训练一个基于全连接神经网络的手写数字识别模型，实验内容包括构建带残差连接的多层神经网络结构，利用 MNIST 数据集对模型进行训练和性能评估，实现学习率的自适应调整以优化模型的收敛性能（后续发现预训练 + 固定学习率效果更好）。此外，本实验还实现了模型权重的保存与加载机制，便于权重复用和迁移学习，并通过绘制训练损失与准确率变化曲线，以及随机抽样预测结果的可视化分析，直观地展示模型性能，以期综合掌握神经网络模型的搭建与性能评估方法。

2 实验背景、应用场景与价值

本实验基于 MNIST 手写数字数据集，构建并训练一个简单的全连接神经网络模型，以研究神经网络在图像分类任务中的应用。近年来，深度学习在计算机视觉领域取得了显著进展，手写数字识别作为图像分类的基础任务，广泛应用于智能字符识别（OCR）、金融票据处理、自动化办公系统以及身份认证等多个领域。例如，在银行与税务行业，手写字符识别技术可用

于票据、支票等文档的自动录入与验证，提高工作效率并降低人工成本；在交通管理领域，手写识别可应用于车牌自动识别，提高自动化程度；在教育行业，智能阅卷系统利用手写识别技术可实现自动评分，提高批改效率。本实验不仅研究了深度神经网络在手写数字识别中的应用，还通过引入残差连接与自适应学习率优化方法，提高了模型的稳定性和泛化能力。此外，实验实现了权重存储与加载功能，使得模型可进行断点续训和迁移学习，提高了模型的复用性和扩展性，为更复杂数据集上的应用提供了良好的基础，同时也是深度学习初学者入门的良好开端。

3 数据预处理

本实验使用 MNIST 手写数字数据集进行训练与测试，该数据集由 28×28 像素的灰度图像组成，涵盖 0 至 9 共 10 个手写数字类别。MNIST 数据集包含 60,000 张训练图像和 10,000 张测试图像，每张图像的像素值范围为 0 到 255。为提高模型训练的稳定性和收敛速度，在数据预处理阶段，我们将所有像素值归一化到 $[0, 1]$ 区间。此外，数据集中提供的标签为 0-9 的整数值，为适应神经网络的计算，我们将其转换为独热编码（one-hot encoding）格式，使其在分类任务中更具可计算性。通过该数据预处理步骤，确保输入数据格式统一，为模型的高效训练奠定基础。

4 实验方法

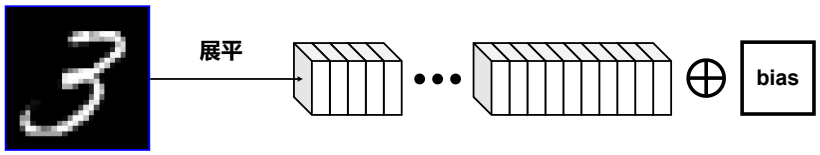

图 1: 28 * 28 的灰度图像被展平为长度为 784 的一维向量

本实验采用全连接神经网络进行 MNIST 手写数字识别任务，网络结构由输入层、两个隐藏层和输出层组成。如图1所示，输入层接收尺寸为 28×28 的灰度图像，并展平为长度为 784 的一维向量，在输入数据后附加一个偏置项，使得输入维度为 785。第一隐藏层为 100 维全连接层，采用 ReLU 激活

函数进行非线性变换；第二隐藏层为 66 维全连接层，并同样使用 ReLU 进行激活。为了优化梯度传播，提高模型的学习能力，在第一隐藏层和第二隐藏层之间引入了残差连接，残差连接通过一个 100×66 的变换矩阵将第一隐藏层的输出直接映射到第二隐藏层的维度，并与其输出相加。最终，模型的输出层为 10 维全连接层，使用 Softmax 计算每个类别的概率分布，并采用交叉熵损失函数进行优化。

在训练过程中，采用手动实现的随机梯度下降（SGD）方法进行参数更新，每次迭代计算梯度后，按照以下方式更新模型权重：

$$W_i \leftarrow W_i - \eta \cdot \nabla W_i \tag{1}$$

其中，η 表示学习率，∇W_i 表示当前梯度。在本实验中，初始学习率设置为 5×10^{-4}，并使用固定学习率策略进行优化，而未引入学习率衰减。实验过程中，模型的训练轮次设定为 200 轮，并采用全数据批量梯度下降的方法进行训练，即每次梯度更新使用整个训练集样本。在测试阶段，加载训练好的权重，对 10,000 张测试集样本进行前向传播，计算交叉熵损失和分类准确率，并随机抽取若干张测试图像进行预测可视化，以直观展示模型的分类效果。

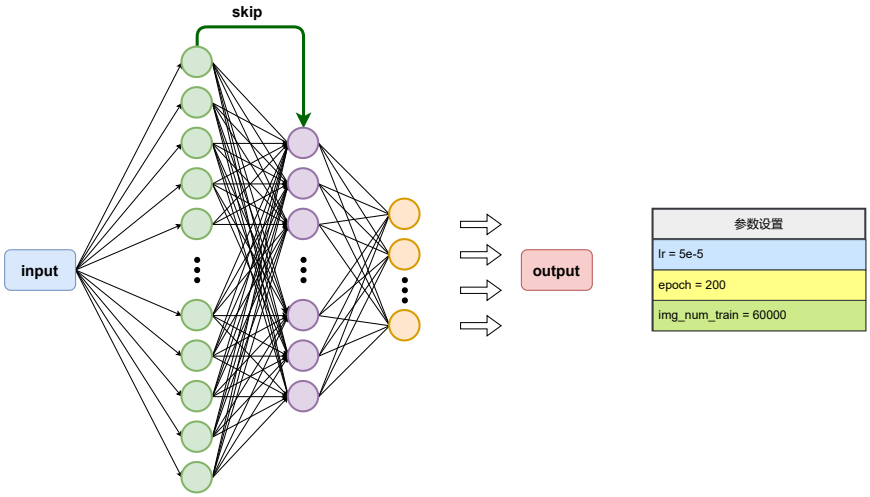

图 2: 网络整体架构与参数设置

4.1 残差连接

本实验在第一隐藏层与第二隐藏层之间引入了残差连接（Residual Connection），以优化梯度传播，提高网络的训练稳定性。传统的前馈神经网络结构依赖逐层传递信息，而残差连接通过直接跳过一个隐藏层，将较低层的特征信息引入到更高层，缓解深层网络可能出现的梯度消失问题。

假设第一隐藏层的输出为 $\mathbf{h}_1$，经过 ReLU 激活后记作：

$$\mathbf{h}_1^{\text{relu}} = \text{ReLU}(\mathbf{h}_1). \tag{2}$$

正常情况下，该输出会作为第二隐藏层的输入，计算如下：

$$\mathbf{h}_2 = \mathbf{h}_1^{\text{relu}} W_2 + b_2. \tag{3}$$

然而，在本实验的残差连接中，我们引入一个额外的映射矩阵 $\mathbf{W}_{\text{skip}}$，用于调整第一隐藏层的输出维度，使其与第二隐藏层匹配：

$$\mathbf{h}_1^{\text{skip}} = \mathbf{h}_1^{\text{relu}} W_{\text{skip}}. \tag{4}$$

最终，我们将 $\mathbf{h}_1^{\text{skip}}$ 直接加到第二隐藏层的输出 $\mathbf{h}_2^{\text{relu}}$ 上，形成残差连接：

$$\mathbf{h}_2^{\text{res}} = \mathbf{h}_2^{\text{relu}} + \mathbf{h}_1^{\text{skip}}. \tag{5}$$

该残差连接确保了梯度可以通过跳跃路径向网络的更低层传播，提高了训练的稳定性和收敛速度，同时增强了模型对低层特征的利用能力。

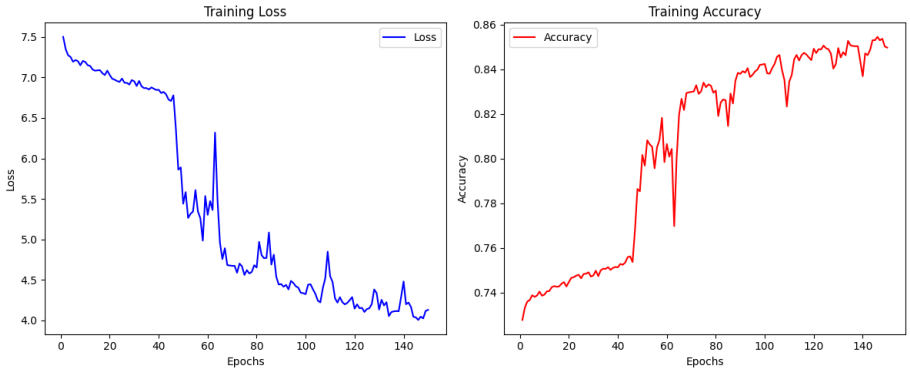

图 3: 损失曲线与准确率曲线

5 实验结果

本实验训练了一个全连接神经网络用于 MNIST 手写数字识别任务，训练过程的损失（Loss）和准确率（Accuracy）如图 3 所示。从损失曲线可以观察到，损失函数随着训练轮次的增加呈现出整体下降趋势，表明模型的优化过程较为稳定。然而，在某些阶段（如前 50 轮），损失下降较为缓慢，可能是由于学习率较小或数据特征较为复杂。同时，训练准确率曲线表明，模型的准确率随着迭代次数的增加逐步提升，在训练后期趋于稳定，并最终收敛至 0.84 左右，表明模型具有一定的分类能力。

为了进一步评估模型的识别能力，我们在测试集上随机选取 5 张样本进行预测，并将预测结果与真实标签进行对比（如图 4 所示）。从可视化结果来看，模型在大多数情况下能够准确识别输入的手写数字，分类结果与真实标签一致。但模型在某些情况下仍可能存在一定的识别偏差，可能需要进一步优化网络结构或增加数据增强技术来提高泛化能力。

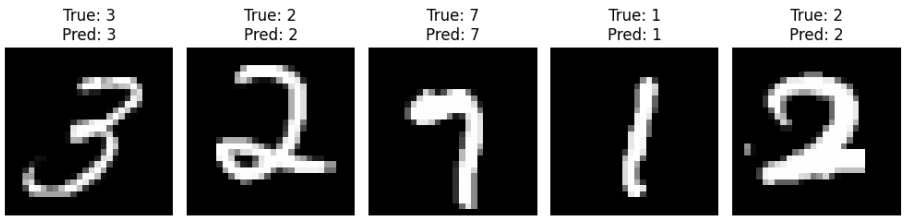

图 4: 测试结果样本可视化

6 讨论与结论

6.1 讨论

从实验结果来看，神经网络在 MNIST 数据集上取得了较好的分类性能，最终训练准确率达到 0.84，且测试样本的可视化结果表明模型在大多数情况下能够正确分类。然而，我们仍然观察到了一些值得讨论的问题。

首先，从损失曲线（图 3）可以看出，训练过程中损失并非单调下降，而是在部分阶段出现了较大的震荡。这可能是由于模型使用了固定学习率，未采用动态调整策略，使得优化过程在某些区域振荡，而未能迅速收敛。此外，由于本实验使用的是全批量梯度下降（Batch Gradient Descent），权重更新依赖于整个训练集，这可能导致收敛速度较慢。未来可以尝试 **小批量梯度下降**（Mini-Batch Gradient Descent）或**自适应优化算法**（如 Adam、RMSprop）以提升训练稳定性。其次，测试集的分类结果（图 5）显示，某

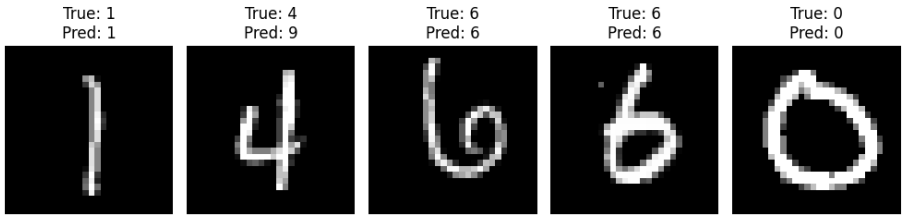

图 5: 测试结果样本可视化 (包含错误分类)

些手写数字由于笔迹模糊或形状相似，导致模型误判。例如，某些“4”被误识别为“9”，这可能是由于数据集中不同书写风格导致的类间相似性问题。为了解决这一问题，可以引入数据增强（Data Augmentation）方法，如旋转、缩放、形变等，以提升模型对不同书写风格的泛化能力。此外，当前模型仅采用全连接层（Fully Connected Layer, FC Layer），而在手写数字识别任务中，卷积神经网络（CNN）通常能够更好地提取图像特征，提高分类性能。因此，未来可以考虑引入 CNN 结构，如 LeNet、VGG 等，以进一步优化模型性能。

6.2 结论

本实验采用全连接神经网络对 MNIST 手写数字数据集进行了分类研究，网络结构包含两层隐藏层，并引入了残差连接以优化梯度传播。实验结果表明，该模型在 MNIST 数据集上的最终训练准确率达到 0.84，并在测试

集上展现了较好的分类能力。然而，实验过程中仍然存在损失震荡、分类错误等问题，可能受限于固定学习率、数据分布以及网络结构等因素。未来可以通过优化学习率策略、使用更高级的优化器、引入数据增强、采用卷积神经网络（CNN）等方式进一步提升模型性能。

全连接神经网络在手写数字识别任务中具有可行性，我们重新设计了工作并为后续改进提供了方向。本研究的结果不仅有助于深入理解深度学习在计算机视觉领域的应用，同时也为后续更复杂的图像分类任务学习提供了基础。

7　附录

7.1　ex1

```
import os
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, datasets, optimizers
from plot_metrics import plot_metrics
from save_weights import save_weights

os.environ["TF_CPP_MIN_LOG_LEVEL"] = "2"

# 准备数据
def mnist_dataset():
(x, y), (x_test, y_test) = datasets.mnist.load_data()
x = x / 255.0
x_test = x_test / 255.0
return (x, y), (x_test, y_test)

# 定义矩阵乘法类
class Matmul:
def __init__(self):
self.mem = {}

def forward(self, x, W):
h = np.matmul(x, W)
self.mem = {"x": x, "W": W}
return h

def backward(self, grad_y):
x = self.mem["x"]
W = self.mem["W"]
grad_x = np.matmul(grad_y, W.T)
grad_W = np.matmul(x.T, grad_y)
return grad_x, grad_W

# 定义 ReLU 类
class Relu:
def __init__(self):
self.mem = {}

def forward(self, x):
```

```
41          self.mem["x"] = x
42          return np.where(x > 0, x, np.zeros_like(x))
43
44          def backward(self, grad_y):
45          x = self.mem["x"]
46          return (x > 0).astype(np.float32) * grad_y
47
48          # 定义 Softmax 类
49          class Softmax:
50          def __init__(self):
51          self.mem = {}
52          self.epsilon = 1e-12
53
54          def forward(self, x):
55          x = x - np.max(x, axis=1, keepdims=True)  # 防止溢出
56          x_exp = np.exp(x)
57          denominator = np.sum(x_exp, axis=1, keepdims=True)
58          out = x_exp / (denominator + self.epsilon)
59          self.mem["out"] = out
60          return out
61
62          def backward(self, grad_y):
63          s = self.mem["out"]
64          sisj = np.matmul(np.expand_dims(s, axis=2), np.expand_dims(s, axis=1))
65          g_y_exp = np.expand_dims(grad_y, axis=1)
66          tmp = np.matmul(g_y_exp, sisj)
67          tmp = np.squeeze(tmp, axis=1)
68          return -tmp + grad_y * s
69
70          # 定义交叉熵类
71          class CrossEntropy:
72          def __init__(self):
73          self.epsilon = 1e-12
74          self.mem = {}
75
76          def forward(self, x, labels):
77          log_prob = np.log(x + self.epsilon)
78          out = np.mean(np.sum(-log_prob * labels, axis=1))
79          self.mem["x"] = x
80          return out
81
82          def backward(self, labels):
83          x = self.mem["x"]
84          return -1 / (x + self.epsilon) * labels
85
86          # 定义模型
87          class MyModel:
88          def __init__(self, pretrained_dir=None):
89          # 初始化权重
90          self.W1 = np.random.normal(size=[28 * 28 + 1, 100])
91          self.W2 = np.random.normal(size=[100, 66])
92          self.W3 = np.random.normal(size=[66, 10])
93          self.W_skip = np.random.normal(size=[100, 66])
94
95          self.mul_h1 = Matmul()
96          self.relu1 = Relu()
97          self.mul_h2 = Matmul()
98          self.relu2 = Relu()
99          self.mul_h3 = Matmul()
100         self.softmax = Softmax()
101         self.cross_en = CrossEntropy()
102
103         # 如果指定了预训练权重目录，加载权重
```

```
104         if pretrained_dir is not None:
105         self.load_weights(pretrained_dir)
106
107         def load_weights(self, weights_dir):
108         try:
109         self.W1 = np.load(os.path.join(weights_dir, "W1.npy"))
110         self.W2 = np.load(os.path.join(weights_dir, "W2.npy"))
111         self.W3 = np.load(os.path.join(weights_dir, "W3.npy"))
112         self.W_skip = np.load(os.path.join(weights_dir, "W_skip.npy"))
113         print(f"Pretrained weights loaded from '{weights_dir}' successfully!")
114         except FileNotFoundError as e:
115         print(f"Error: {e}. Using randomly initialized weights.")
116
117
118         def forward(self, x, labels):
119         x = x.reshape(-1, 28 * 28)
120         bias = np.ones(shape=[x.shape[0], 1])
121         x = np.concatenate([x, bias], axis=1)
122
123         self.h1 = self.mul_h1.forward(x, self.W1)
124         self.h1_relu = self.relu1.forward(self.h1)
125
126         self.h2 = self.mul_h2.forward(self.h1_relu, self.W2)
127         self.h2_relu = self.relu2.forward(self.h2)
128
129         self.h1_skip = np.matmul(self.h1_relu, self.W_skip)
130         self.h2_res = self.h2_relu + self.h1_skip
131
132         self.h3 = self.mul_h3.forward(self.h2_res, self.W3)
133         self.h3_soft = self.softmax.forward(self.h3)
134         self.loss = self.cross_en.forward(self.h3_soft, labels)
135
136         def backward(self, labels):
137         self.loss_grad = self.cross_en.backward(labels)
138         self.h3_soft_grad = self.softmax.backward(self.loss_grad)
139
140         self.h3_grad, self.W3_grad = self.mul_h3.backward(self.h3_soft_grad)
141
142         self.h2_res_grad = self.h3_grad
143         self.h2_relu_grad = self.h2_res_grad
144         self.h1_skip_grad = np.matmul(self.h2_res_grad, self.W_skip.T)
145
146         self.h1_relu_grad = self.relu1.backward(self.h1_skip_grad)
147
148         self.h2_grad, self.W2_grad = self.mul_h2.backward(self.h2_relu_grad)
149         self.h1_grad, self.W1_grad = self.mul_h1.backward(self.h1_relu_grad)
150         self.W_skip_grad = np.matmul(self.h1_relu.T, self.h2_res_grad)
151
152         # 计算准确率
153         def compute_accuracy(prob, labels):
154         predictions = np.argmax(prob, axis=1)
155         truth = np.argmax(labels, axis=1)
156         return np.mean(predictions == truth)
157
158         def train_one_step(model, x, y, learning_rate):
159         model.forward(x, y)
160         model.backward(y)
161
162         # 更新权重时使用自适应学习率
163         model.W1 -= learning_rate * model.W1_grad
164         model.W2 -= learning_rate * model.W2_grad
165         model.W3 -= learning_rate * model.W3_grad
166         model.W_skip -= learning_rate * model.W_skip_grad
```

```
167
168         loss = model.loss
169         accuracy = compute_accuracy(model.h3_soft, y)
170         return loss, accuracy
171
172         # 测试模型
173         def test(model, x, y):
174         model.forward(x, y)
175         loss = model.loss
176         accuracy = compute_accuracy(model.h3_soft, y)
177         return loss, accuracy
178
179         # 训练模型
180         if __name__ == "__main__":
181         # 当且仅当你直接执行 model.py 时，这段代码才会运行。
182         model = MyModel()
183         train_data, test_data = mnist_dataset()
184         train_label = np.zeros(shape=[train_data[0].shape[0], 10])
185         test_label = np.zeros(shape=[test_data[0].shape[0], 10])
186         train_label[np.arange(train_data[0].shape[0]), train_data[1]] = 1
187         test_label[np.arange(test_data[0].shape[0]), test_data[1]] = 1
188
189         initial_lr = 5e-4  # 初始学习率
190         loss_list, accuracy_list = [], []
191         for epoch in range(200):
192         # 计算当前 epoch 的学习率（指数衰减）
193         learning_rate = initial_lr ** (0.95 ** epoch)
194         loss, accuracy = train_one_step(model, train_data[0], train_label, learning_rate=learning_rate)
195
196         loss_list.append(loss)
197         accuracy_list.append(accuracy)
198         print(f'Epoch {epoch}: Loss {loss:.4f}; Accuracy {accuracy:.4f}; LR {learning_rate:.6f}')
199
200         plot_metrics(loss_list, accuracy_list, save_dir="homework/ex1")
201
202         loss, accuracy = test(model, test_data[0], test_label)
203         print(f'Test Loss {loss}; Accuracy {accuracy}')
204
205         save_weights(model, save_dir="homework/ex1/weight")
206         print('权重文件已保存到 homework/ex1/weight')
```

7.2 train

```
1         import argparse
2         from ex1 import MyModel, compute_accuracy, mnist_dataset, train_one_step, test
3         from plot_metrics import plot_metrics
4         from save_weights import save_weights
5         import numpy as np
6
7         # 解析命令行参数
8         parser = argparse.ArgumentParser(description="Train model with optional pretrained weights")
9         parser.add_argument('--weights_dir', type=str, default=None, help='Path to pretrained weights')
10        args = parser.parse_args()
11
12        # 主训练流程
13        model = MyModel(pretrained_dir=args.weights_dir)
14        train_data, test_data = mnist_dataset()
15        train_label = np.zeros(shape=[train_data[0].shape[0], 10])
16        test_label = np.zeros(shape=[test_data[0].shape[0], 10])
17        train_label[np.arange(train_data[0].shape[0]), train_data[1]] = 1
```

```
        test_label[np.arange(test_data[0].shape[0]), test_data[1]] = 1

        initial_lr = 5e-5  # 初始学习率
        loss_list, accuracy_list = [], []
        for epoch in range(150):
        # 计算当前 epoch 的学习率（指数衰减）
        learning_rate = initial_lr #* (0.99 ** epoch)

        loss, accuracy = train_one_step(model, train_data[0], train_label, learning_rate=learning_rate)

        loss_list.append(loss)
        accuracy_list.append(accuracy)
        print(f'Epoch {epoch}: Loss {loss:.4f}; Accuracy {accuracy:.4f}; LR {learning_rate:.6f}')

        plot_metrics(loss_list, accuracy_list, save_dir="homework/ex1/loss")

        loss, accuracy = test(model, test_data[0], test_label)
        print(f'Test Loss {loss}; Accuracy {accuracy}')

        save_weights(model, save_dir="homework/ex1/weight")
        print('权重文件已保存到 homework/ex1/weight')
```

7.3 test5

```
        import argparse
        import numpy as np
        from ex1 import MyModel, mnist_dataset, compute_accuracy
        from load_and_visualize import load_weights, visualize_predictions

        if __name__ == "__main__":
        parser = argparse.ArgumentParser(description='仅测试模型，不进行训练')
        parser.add_argument('--weights_dir', type=str, default="homework/ex1/weight", help='权重文件夹路径')
        parser.add_argument('--save_path', type=str, default="homework/ex1/test_img", help='预测图片保存路径')
        parser.add_argument('--num_samples', type=int, default=5, help='随机测试图片数量')

        args = parser.parse_args()

        # 加载模型和权重
        model = MyModel()
        load_weights(model, weights_dir=args.weights_dir)

        # 测试数据
        _, test_data = mnist_dataset()
        test_label = np.zeros(shape=[test_data[0].shape[0], 10])
        test_label[np.arange(test_data[0].shape[0]), test_data[1]] = 1

        # 测试
        model.forward(test_data[0], test_label)
        loss = model.loss
        accuracy = compute_accuracy(model.h3_soft, test_label)
        print(f'Test Loss: {loss}, Accuracy: {accuracy}')

        # 随机测试并可视化预测图片
        visualize_predictions(model, num_samples=args.num_samples, save_path=args.save_path)
```

7.4 save__weights

```
import numpy as np
import os

def save_weights(model, save_dir="weights"):
if not os.path.exists(save_dir):
os.makedirs(save_dir)

np.save(os.path.join(save_dir, "W1.npy"), model.W1)
np.save(os.path.join(save_dir, "W2.npy"), model.W2)
np.save(os.path.join(save_dir, "W3.npy"), model.W3)
np.save(os.path.join(save_dir, "W_skip.npy"), model.W_skip)
print(f"Weights have been saved successfully to '{save_dir}'")
```