

**实 验 报 告**

**（2024 / 2025 学年 第 一 学期）**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **课程名称** | 机器学习（双语） | | | | | |
| **实验名称** | Python usage in deep learning methods | | | | | |
| **实验时间** | 2024 | **年** | 12 | **月** | 31 | **日** |
| **指导单位** | 物联网学院 | | | | | |
| **指导教师** |  | | | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **学生姓名** |  | **班级学号** |  |
| **学院(系)** | 物联网学院 | **专 业** | 网络工程 |

**实 验 报 告**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **实验名称** | | Python usage in deep learning methods | | | | | **指导教师** | | |  |
| **实验类型** | | 上机 | | **实验学时** | | 2 | **实验时间** | | | 2024年12月31日 |
| 1. **实验目的和要求**   实验要求：Learning to use Python for deep learning approaches.  实验目的：1）Knowing the basic usage of Python on designing deep learning structures;  2）Understanding the theory of deep networks;  3）Knowing how to use machine learning approaches to process complex data. | | | | | | | | | | |
| **二、实验环境**  机器硬件配置：Windows 11 24H2  软件安装：Miniconda3, Python 3.6, Python 3.8 | | | | | | | | | | |
| **三、实验原理及内容**  **Task 1: 直接使用TensorFlow实现MNIST数据集分类**  工具包/库：tensorflow。  训练集及训练参数：28x28图像，10类（手写数字），batch\_size=100，学习率(learning rate)=0.001，轮数(epochs)=10。  训练样本(x\_train)数55000，测试样本(x\_test)数10000。  网络结构（两层卷积）：conv1+pool1+conv2+pool2+flatten+fc+output    图1 最后一轮（ep10）的训练结果及模型验证结果  最终训练结果为：模型在测试集上的测试准确率达到了98.9%          图2 模型训练及验证结果  调节网络结构和参数，新增第三层卷积层和池化层，卷积核大小减少至3x3，卷积核数量增加至64，卷积步长默认为1。修改后的网络结构：conv1+pool1+conv2+pool2+ conv3+pool3+flatten+fc+output，新增代码如下：  # 1st Convolutional Layer  filter\_size1 = 5  # Convolution filters are 5 x 5 pixels.  num\_filters1 = 16  # There are 16 of these filters.  stride1 = 1  # The stride of the sliding window  # 2nd Convolutional Layer  filter\_size2 = 5  # Convolution filters are 5 x 5 pixels.  num\_filters2 = 32  # There are 32 of these filters.  stride2 = 1  # The stride of the sliding window  **# 3rd Convolutional Layer (Newly Added)**  **filter\_size3 = 3   # Convolution filters are 3 x 3 pixels.**  **num\_filters3 = 64  # There are 64 of these filters.**  **stride3 = 1  # The stride of the sliding window**  # Fully-connected layer.  h1 = 128  # Number of neurons in fully-connected layer.  ## Network graph  # Placeholders for the inputs (x) and corresponding labels (y)  with tf.name\_scope('Input'):      x = tf.placeholder(tf.float32, shape=[None, img\_h, img\_w, n\_channels], name='X')  y = tf.placeholder(tf.float32, shape=[None, n\_classes], name='Y')  # Create the network layers  conv1 = conv\_layer(x, filter\_size1, num\_filters1, stride1, name='conv1')  pool1 = max\_pool(conv1, ksize=2, stride=2, name='pool1')  conv2 = conv\_layer(pool1, filter\_size2, num\_filters2, stride2, name='conv2')  pool2 = max\_pool(conv2, ksize=2, stride=2, name='pool2')  **conv3 = conv\_layer(pool2, filter\_size3, num\_filters3, stride3, name='conv3')**  **pool3 = max\_pool(conv3, ksize=2, stride=2, name='pool3')**  # Flatten layer  layer\_flat = flatten\_layer(pool3)  # Fully connected layer  fc1 = fc\_layer(layer\_flat, h1, 'FC1', use\_relu=True)  # Output layer  output\_logits = fc\_layer(fc1, n\_classes, 'OUT', use\_relu=False)  修改网络后，重新训练模型。训练参数保持不变。在测试集上的准确率：98.9%。与修改前的训练结果相同。    图3 修改网络后的训练结果    图4 修改网络后的验证结果  **Task 2: 使用Keras实现MNIST数据集分类**  工具包/库：tensorflow.keras。  训练集及训练参数：28x28图像，10类（手写数字），batch size=128，初始学习率=0.01，轮数(epochs)=15。  训练/验证样本(x\_train)数：60000(54000+6000)；测试样本(x\_test)数：10000。  网络结构（两层卷积）：conv1+pool1+conv2+dropout+pool2+flatten+fc+output    图5 训练结果  最终训练结果如上：测试集loss：0.023362266795919278，准确率：99.23%。  同样，调节网络结构和参数，新增第三层卷积层和池化层，卷积核大小为3x3，卷积核数量增加至128，并在卷积层后添加池化层以进行降维。修改后的关键代码如下：  model = keras.Sequential()  # 1st Convolutional Layer  model.add(layers.Conv2D(32, kernel\_size=(3, 3), activation='relu', input\_shape=input\_shape))  model.add(layers.MaxPooling2D(2, 2))  # 2nd Convolutional Layer  model.add(layers.Conv2D(64, kernel\_size=(3, 3), activation='relu'))  model.add(layers.MaxPooling2D(2, 2))  **# 3rd Convolutional Layer (Newly Added)**  **model.add(layers.Conv2D(128, kernel\_size=(3, 3), activation='relu'))**  **model.add(layers.MaxPooling2D(2, 2))**  # Dropout and Flatten Layer  model.add(layers.Dropout(0.5))  model.add(layers.Flatten())  # Fully Connected Layer  model.add(layers.Dense(num\_classes, activation="softmax"))  model.summary()  修改完成后重新训练模型，训练结果如下。测试集loss：0.033847148244070195，准确率：99.11%。训练同等的epoch，与修改前相比，loss略微变高且acc值下降，因此修改后需要训练更多的epoch才能得到更好的结果。    图6 新增第三层卷积层后的训练结果  给验证新增更多的评价指标：在model.compile的metrics中增加Precision和Recall。通过tensorflow.keras.metrics.Precision()和tensorflow.keras.metrics.Recall()计算精确率和召回率。关键代码如下：  # Evaluate the model  score = model.evaluate(x\_test, y\_test, verbose=0)  print("Test loss:", score[0])  print("Test accuracy:", score[1])  # Compute precision, recall, and F1-score  y\_pred = model.predict(x\_test, verbose=0)  y\_pred\_classes = np.argmax(y\_pred, axis=1)  y\_true\_classes = np.argmax(y\_test, axis=1)  **precision = precision\_score(y\_true\_classes, y\_pred\_classes, average='weighted')**  **recall = recall\_score(y\_true\_classes, y\_pred\_classes, average='weighted')**  **f1 = f1\_score(y\_true\_classes, y\_pred\_classes, average='weighted')**  **print("Test precision:", precision)**  **print("Test recall:", recall)**  **print("Test F1-score:", f1)**  重新训练模型，观察新增指标如下：  Test loss: 0.04018177319267997  Test accuracy: 0.9893  Test precision: 0.9893690550743258  Test recall: 0.9893  Test F1-score: 0.989302210934031    图7 新增评价指标后的训练结果  **Task 3: 使用Keras实现销售量预测**  工具包/库：keras。  训练集：一维时间序列，316个点训练，12个点测试；训练中每12点作为训练样本，后1点作为标签；测试中每次测试加入前1点的预测值（预测标签）作为样本。  训练样本数304，测试样本数12。  网络结构（三层LSTM）：lstm1+lstm2+lstm3+output    图8 网络结构  训练结束以后进行预测，预测的结果如下：    图9 销售量预测训练结果及预测结果  调整网络结构和参数，将LSTM的units全部从128增加到256，并且另外添加两层LSTM，修改后的网络如下（五层LSTM）：lstm1+lstm2+lstm3+ lstm4+lstm4+output。具体修改的代码如下：  model = Sequential()  model.add(LSTM(**256**, activation='relu', input\_shape=(n\_input, n\_feature), return\_sequences=True))  model.add(LSTM(**256**, activation='relu', return\_sequences=True))  **model.add(LSTM(256, activation='relu', return\_sequences=True))**  **model.add(LSTM(256, activation='relu', return\_sequences=True))**  model.add(LSTM(**256**, activation='relu', return\_sequences=False))  model.add(Dense(1))  model.compile(optimizer='adam', loss='mse')  model.summary()  修改完成后继续训练，训练时间明显变慢（每个epoch所需时间大约15秒）。训练同样的epoch（50轮）后，进行预测，预测结果如下：    图10 修改网络重新训练后的预测结果  **Task 4（选做）：使用Keras实现天气预测**  首先需要解决依赖版本问题，在Python=3.8经过测试以下依赖版本可以成功运行：  Tensorflow==2.3.0，numpy==1.18.5，pandas==1.1.0，protobuf==3.20.0，matplotlib==3.5.0    图11 Feature Correlation Heatmap    图12 Raw Visualization    图13 训练及验证的Loss值      图14 单步验证结果  **Task 5：使用PyTorch CNN架构实现MNIST数据集分类**  在Python3.8中安装依赖，相关依赖及版本如下：scipy==1.9，numpy==1.18.5，torch==1.10.2，scikit-learn，tensorflow==2.3.0，matplotlib==3.5.0。训练30个epoch，最终训练的准确率为：98.73%。    图15 使用PyTorch CNN网络训练过程，ACC值及混淆矩阵    图16 随机验证结果（10000张图片中任意挑选1张）  **Task 6（选做）：使用其他相关算法并比较结果（此处使用SVM）**  使用SVM（支持向量机）训练手写数字识别。从训练结果来看，准确率明显低于上面的其他算法。详细的代码如下：  import numpy as np  from sklearn import svm  from sklearn.metrics import classification\_report, accuracy\_score  from tensorflow.keras.datasets import mnist  import matplotlib.pyplot as plt  rootpath = str(os.getcwd()).replace("\\", "/")  path = rootpath + "/MNIST\_data/mnist.npz"  (X\_train, y\_train), (X\_test, y\_test) = mnist.load\_data(path)  X\_train = X\_train.reshape(-1, 28\*28) / 255.0  X\_test = X\_test.reshape(-1, 28\*28) / 255.0  # 使用SVM进行训练  clf = svm.SVC(kernel='linear')  # 使用线性核  clf.fit(X\_train, y\_train)  # 对测试集进行预测  y\_pred = clf.predict(X\_test)  # 打印结果  print("Accuracy:", accuracy\_score(y\_test, y\_pred))  print("Classification Report:")  print(classification\_report(y\_test, y\_pred))  # 绘制一些预测结果  fig, axes = plt.subplots(1, 5, figsize=(10, 5))  for i in range(5):      ax = axes[i]      ax.imshow(X\_test[i].reshape(28, 28), cmap="gray")      ax.set\_title(f"Pred: {y\_pred[i]}")      ax.axis("off")  plt.show()    图17 使用SVM算法训练手写数字识别的训练结果    图18 SVM算法的一些预测结果 | | | | | | | | | | |
| **四、实验小结**  在本次实验中，我深入学习并实践了Python在深度学习中的应用，尤其是在使用TensorFlow和Keras实现MNIST数据集分类的过程中，收获颇丰。通过实际操作，我不仅掌握了如何搭建卷积神经网络并调整其结构来优化分类精度，还加深了对深度学习网络结构与参数调节之间关系的理解。尤其是通过不断调整卷积层的数量和大小、池化层的配置、以及学习率等参数，我深刻感受到了模型训练中精细调整的重要性。尽管一些参数调整对模型效果有较大影响，但每次改动都需要通过多轮实验验证，才能确定最佳配置。  在使用Keras实现MNIST分类时，我还尝试了加入Dropout层以及额外的卷积层，发现这些变动可以提高模型的准确性，但也需要更多的训练时间。此外，通过使用精准度、召回率和F1值等评价指标，我认识到了单一的准确率可能无法全面反映模型性能，特别是在处理类别不平衡问题时，精确率和召回率的综合考虑尤为重要。  通过对销售量预测和天气预测任务的实践，我体会到了深度学习在时间序列预测中的应用。在这一过程中，我通过LSTM网络实现了对时序数据的建模，感受到了其强大的序列数据处理能力。而在修改LSTM网络的层数和节点数时，虽然训练时间明显增加，但模型的预测能力得到了进一步的提升。  本次实验不仅让我掌握了Python深度学习框架的使用方法，还让我对深度学习模型的原理和调优技巧有了更加全面的理解。每一项任务的实现都让我更加熟悉了模型构建、训练以及优化的全过程，同时也增强了我在未来深度学习领域进一步研究和应用的信心。 | | | | | | | | | | | |
| **五、指导教师评语** | | | | | | | | | | | |
| **成 绩** |  | | **批阅人** | |  | | | **日 期** |  | | |