人工智能基础第二次实验

实验概述

背景与目标

人工智能基础课程的第二次实验旨在通过实践加深对经典机器学习、深度学习及对抗攻击算法的理解。本实验分为三部分：

- **经典机器学习（SVM）**：实现自定义支持向量机算法，理解核函数、支持向量和二次规划的核心概念，并验证其在不同数据集上的性能。
- **深度学习（RNN）**：通过调整超参数，训练并比较RNN、LSTM和GRU模型在IMDB情感分析任务中的表现，分析超参数对性能的影响。
- **附加实验（GCG攻击）**：基于Greedy Coordinate Gradient（GCG）算法，攻击预训练语言模型，绕过安全限制生成指定输出，探索对抗性攻击原理。

实验目标是通过代码实现和实验分析，掌握算法设计、优化、评估及对抗攻击的技能，为后续人工智能学习打下基础。

实验环境

- **依赖库**：
  - SVM任务：NumPy, CVXOPT, scikit-learn, Matplotlib
  - RNN任务：datasets, PyTorch, NumPy, Matplotlib, tqdm, scikit-learn
  - GCG任务：PyTorch, transformers, NumPy, livelossplot
- **运行环境**：Python 3.11，建议使用GPU加速RNN和GCG任务（支持CUDA或MPS）。
- **文件管理**：
  - 数据缓存：`./data/`（RNN任务，IMDB数据集缓存）
  - 模型权重：`./models/`（RNN任务，保存模型权重）
  - 可视化图表：`./figures/`（RNN和GCG任务，保存损失和准确率曲线），`./results/`（SVM任务，保存决策边界图）

实验步骤

1. **安装依赖**：

```
conda create -n ai_lab2 python=3.11
conda activate ai_lab2
pip install numpy cvxopt scikit-learn matplotlib datasets torch tqdm
transformers livelossplot
```

2. **运行代码**：
   - SVM任务：运行`./SVM/svm.py`，实现占位符部分，观察输出和图表。
   - RNN任务：运行`./RNN/rnn.py`，设置超参数，训练模型并分析结果。
   - GCG任务：运行`./Bonus/gcg.py`，实现占位符部分，验证攻击效果。
3. **分析结果**：
   - 比较不同实现和参数配置的性能。
   - 提交实验报告，包含代码、结果和分析。

第一部分：经典机器学习 - 支持向量机（SVM）

1.1 背景

支持向量机（Support Vector Machine, SVM）是一种经典的监督学习算法，广泛应用于分类和回归任务，尤其在高维数据分类中表现出色。SVM通过寻找一个最优超平面，将不同类别的数据点分开，最大化类别之间的间隔。这种最大间隔的特性使得SVM具有良好的泛化能力。通过核函数的引入，SVM能够处理非线性分类问题，将数据映射到高维空间实现线性可分。本实验要求你实现自定义SVM算法，理解其核心原理，并通过实验验证其性能。

1.2 原理

SVM的核心思想是找到一个超平面，使得最近的数据点（支持向量）到超平面的距离（间隔）最大化。其关键概念包括：

- **最大间隔**：SVM寻找一个超平面，使得支持向量到超平面的距离最大化。
- **支持向量**：位于间隔边界或误分类的数据点，决定了超平面的位置。
- **核函数**：通过核函数（如线性核、多项式核、高斯核），SVM将数据映射到高维空间，解决非线性问题。
- **软间隔**：通过惩罚参数C，允许部分误分类以提高模型泛化能力。
- **二次规划**：SVM的训练过程通过求解拉格朗日乘子（alphas）优化目标函数，使用二次规划方法（如CVXOPT).

SVM的数学形式基于以下优化问题：

- **硬间隔**：最大化间隔，约束为所有样本正确分类：

$$\min \frac{1}{2}\|w\|^2$$

$$\text{subject to } y_i(w \cdot x_i + b) \geq 1 \quad \forall i$$

- **软间隔**：引入松弛变量，允许误分类：

$$\min \frac{1}{2}\|w\|^2 + C\sum \xi_i$$

$$\text{subject to } y_i(w \cdot x_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0 \quad \forall i$$

- **核函数**：
  - 线性核：$K(x_i, x_j) = x_i \cdot x_j$
  - 多项式核：$K(x_i, x_j) = (x_i \cdot x_j + 1)^d$
  - 高斯核（RBF）：$K(x_i, x_j) = \exp\left(-\gamma\|x_i - x_j\|^2\right)$

1.3 实验要求

你需要完成`svm.py`中`SVM`类的`#TODO`部分，实现以下方法：

1. `__init__`：初始化SVM模型，设置核函数类型、超参数和实例变量。
2. `_kernel_function`：实现线性核、多项式核和高斯核的计算。
3. `fit`：训练SVM模型，求解拉格朗日乘子，提取支持向量和偏置项。
4. `project`：计算决策函数值。
5. `predict`：预测类别标签。

1.4 实验步骤

1. **实现SVM类**：
   - 按照 `#TODO` 注释完成 `SVM` 类的方法。
   - 参考SVM原理和CVXOPT文档.
2. **运行实验**：
   - 执行 `svm.py`，生成决策边界图和性能指标。
3. **评估与分析**：
   - **输出**：检查支持向量数量和测试集准确率。
   - **可视化**：查看 `./results/` 中的图表，验证决策边界和支持向量。
   - **比较**：分析自定义SVM与scikit-learn SVM的性能差异。
   - **报告**：记录实验结果，讨论核函数和参数的影响。

第二部分：深度学习 - 循环神经网络（RNN）

2.1 背景

循环神经网络（RNN）及其变体LSTM和GRU是处理序列数据的核心模型，广泛应用于自然语言处理领域。本实验使用IMDB电影评论数据集进行情感分析，预测评论是正面还是负面。

2.2 原理

RNN通过循环连接处理序列数据，捕捉时间依赖关系：

- **基本RNN**：通过隐藏状态传递信息，但难以捕捉长期依赖。
- **LSTM**：引入记忆单元和门控机制，有效捕捉长序列依赖。
- **GRU**：简化LSTM，合并门控机制，参数更少，计算效率更高。

2.3 实验要求

这里你需要完成 `rnn.py` 中的超参数设置和RNN模型实现：

1. **超参数调整**：
   - 设置批量大小、嵌入维度、隐藏层维度、层数、是否双向、丢弃率和学习率。
   - 尝试至少3组超参数组合，记录性能。
2. **实现RNN模型**：
   - 实现 `RNNModel` 类，支持RNN、LSTM和GRU。
   - 配置嵌入层、RNN层和全连接层。
   - 处理可变长度序列。
3. **运行与评估**：
   - 训练模型并记录损失和准确率。
   - 在测试集上评估模型性能。

2.4 实验步骤

1. **设置超参数**：编辑 `Config` 类中的超参数值。
2. **实现RNN模型**：完成 `RNNModel` 类的方法。
3. **运行实验**：训练模型并观察输出。
4. **评估与分析**：
   - 记录测试集性能指标。
   - 分析训练/验证曲线，判断过拟合情况。

- 比较不同模型和超参数的性能。

附加实验：Transformer及对抗攻击

3.1 背景

对抗攻击是人工智能安全领域的热点，旨在通过精心设计的输入扰动，诱导模型产生错误或特定输出。本实验通过Greedy Coordinate Gradient（GCG）算法攻击TinyStories-33M语言模型，绕过安全限制，生成指定文本："This is great! I love living on the wild side!"。实验模拟越狱（Jailbreaking）场景，探索语言模型的安全性。

3.2 原理

GCG算法通过优化对抗性前缀，最大化目标输出的生成概率。其核心思想包括：

- **越狱场景**：在用户输入后添加对抗性前缀，诱导模型生成特定响应。
- **优化目标**：最小化目标序列的负对数似然（NLL）损失：

$$\mathcal{L}(x_{1:n}) = -\log p(x^{\star}_{n+1:n+H}|x_{1:n})$$

  其中，$x_{1:n}$为输入序列，$x^{\star}_{n+1:n+H}$为目标序列，H为目标长度。
- **梯度引导**：通过梯度计算每个token位置的候选替换，随机采样top-k候选，迭代优化前缀。
- **TinyStories特点**：仅支持文本补全，攻击目标为生成指定补全文本。

3.3 实验要求

这里你需要完成`gcg.py`中的`#TODO`部分，实现以下内容：

1. `token_gradients`：计算对抗性前缀的token梯度：
   - 创建one-hot向量表示输入token。
   - 计算前缀嵌入并替换输入嵌入。
   - 计算目标序列损失并反向传播。
2. `sample_control`：基于梯度采样候选token：
   - 选择top-k候选token。
   - 随机采样batch size次替换，生成新前缀。
3. `is_success`：验证攻击成功：
   - 使用前缀推理，检查输出是否包含目标文本。
4. **切片定义**：
   - 定义`adv_slice`（前缀token范围）。
   - 定义`target_slice`（目标文本token范围）。
   - 定义`loss_slice`（损失计算的logits范围）。

3.4 实验步骤

1. **准备环境**：
   - 下载TinyStories-33M模型权重（Hugging Face或云盘链接）。
   - 更新`model_path`为实际路径。
2. **实现GCG算法**：
   - 按照`#TODO`注释完成`gcg.py`中的函数和切片定义。
   - 参考GCG原理和PyTorch文档。
3. **运行实验**：

- 执行`gcg.py`，优化对抗性前缀。
- 使用`livelossplot`监控损失曲线，保存至`./figures/`。

4. **评估与分析**：
   - **输出**：检查最终前缀和攻击是否成功（输出是否包含目标文本）。
   - **可视化**：分析`./figures/`中的损失曲线，评估优化过程。
   - **比较**：尝试调整`batch_size`、`topk`和`num_steps`，分析对攻击成功率和速度的影响。
   - **报告**：记录实验结果，讨论GCG算法的效率和局限性。

实验要求

1. **代码**：提交完整的`svm.py`、`rnn.py`和`gcg.py`，确保可运行。
2. **结果**：
   - SVM任务：记录支持向量数量、测试集准确率，附上`./results/`中的图表。
   - RNN任务：记录至少3组超参数的测试集准确率和F1分数，附上`./figures/`中的曲线。
   - GCG任务：记录最终对抗性前缀、攻击是否成功，附上`./figures/`中的损失曲线。
3. **分析**：
   - SVM：比较不同核函数和C值的性能，讨论支持向量数量和决策边界的变化。
   - RNN：比较RNN、LSTM和GRU的性能，分析超参数对训练时间、过拟合和性能的影响。
   - GCG：分析GCG算法的优化过程，讨论超参数（如`batch_size`、`topk`）对攻击效果的影响，探讨TinyStories模型的局限性。
4. **总结**：总结实验收获，提出改进建议（如优化算法、尝试其他超参数或模型）。

注意事项

- 确保代码注释清晰，遵循`#TODO`要求。
- 保存所有输出和图表，确保实验可重现。
- 若遇到运行问题，检查依赖版本或硬件兼容性（如GPU支持）。
- GCG任务需注意TinyStories仅支持文本补全，输入格式需正确。
- 提交截止前测试代码，确保无语法错误。
- 最后，如果你不喜欢现在这个框架，在实验任务不变的情况下，可以完全自己实现，但是要保证你的代码能正常运行。

实验提交及评分标准

1.截止日期：2025 年 6 月 28 日 晚23：59

2.提交方式：通过 bb 平台

3.评分标准：经典机器学习 40 分，深度学习 60 分，附加实验 20 分，总分上限 100 分

4.提交的目录树结构如下所示：

```
PB22000001_ 张 三_lab2/
        |-- src
                |-- (your code)
        |-- results
                |-- ...
                |-- ...
```

5.请务必按时提交实验