Program1

Yongchao He
07700005711

1.Abtract

This report presents a technical overview of our implementation for a binary classification task based on k-mer feature extraction from sequence data. I employ an advanced neural network with two hidden layers, leveraging Leaky ReLU activation, dropout regularization, and the Adam optimization algorithm. My methodology involves systematic hyperparameter tuning, class balancing, and evaluation using a validation set. Results are analyzed using ranking and MCC scores, with reflections on potential improvements.

2.Introduction

Sequence classification is essential in bioinformatics and computational biology, enabling the analysis of sequence data for various applications. In this project, we extract k-mer features from sequences and utilize a neural network for classification. The objective is to develop a reliable model that can effectively generalize to new, unseen test data. This report provides an in-depth discussion of the approach, methodology, parameter choices, and evaluation criteria used in the study.

3.Approach and method

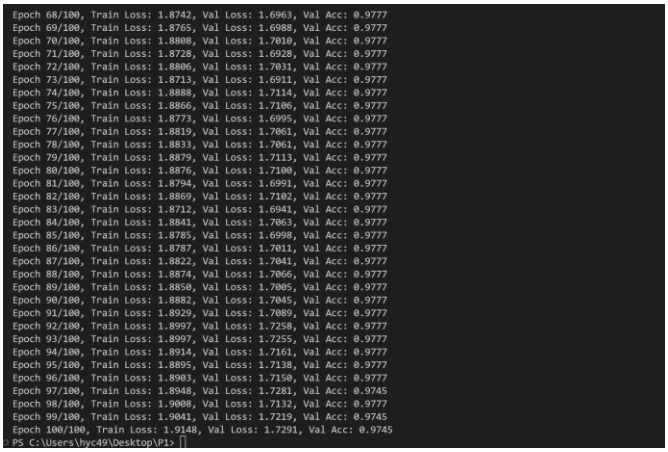
3.1 Early attempt

In the early stages of this project, the primary goal was to establish a foundational model capable of performing sequence classification based on k-mer features.

Initially, **k = 2** was chosen to balance feature representation and computational efficiency. The first model was a simple neural network with a single hidden layer of 256 neurons, utilizing Leaky ReLU activation to mitigate the dying neuron problem. The output layer used a sigmoid activation function for binary classification.

The model was trained with stochastic gradient descent (SGD) and mini-batch updates (batch size = 32), but this led to instability in weight updates, prompting a switch to the Adam optimizer. Overfitting was also observed, leading to the introduction of L2 regularization and dropout (0.2).

Additionally, I encountered class imbalance, which affected performance, so class weights were incorporated to adjust for this. Further refinement led to the final model with two hidden layers (256 and 128 neurons), L2 regularization (1e-5), dropout (0.2), and early stopping based on validation performance. These improvements significantly enhanced stability and generalization, making the model more robust. Therefore, I try to optimize the code.



3.2 Feature Extraction

I chose **k = 2** for k-mer extraction, balancing feature complexity and computational efficiency.

3.3 Neural Network Configuration:
Hidden Layers: Two fully connected layers (256, 128 neurons) with He initialization.
Regularization: L2 weight decay (**1e-5**) and dropout (**0.2**) to prevent overfitting.
Optimizer: Adam optimizer with **learning rate 0.001**.
Batch Size: 32, ensuring stable updates.
Class Weights: Adjusted based on class distribution to handle imbalance.
Epochs: 100, selected based on validation convergence trends.

```
model.eval()
test_preds = []
with torch.no_grad():
    for inputs in test_loader:
        inputs = inputs.to(device)
        outputs = model(inputs).cpu().numpy().flatten()
        preds = np.where(outputs > 0.5, 1, -1)
        test_preds.extend(preds)


output_filename = "prediction.txt"
np.savetxt(output_filename, test_preds, fmt="%d")

print(f"✅ 预测完成，结果已保存为 {output_filename}")
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

/usr/local/bin/python3 /Users/yorkheyongchao/Desktop/342/P1/P2.py
→ P1 /usr/local/bin/python3 /Users/yorkheyongchao/Desktop/342/P1/P2.py
Epoch 1/10, Loss: 0.4020, MCC: 0.0000
Epoch 2/10, Loss: 0.0950, MCC: 0.3646
Epoch 3/10, Loss: 0.0089, MCC: 0.3285
Epoch 4/10, Loss: 0.0028, MCC: 0.3250
Epoch 5/10, Loss: 0.0017, MCC: 0.3211
Epoch 6/10, Loss: 0.0010, MCC: 0.3211
Epoch 7/10, Loss: 0.0009, MCC: 0.3173
Epoch 8/10, Loss: 0.0008, MCC: 0.3173
Epoch 9/10, Loss: 0.0007, MCC: 0.3101
Epoch 10/10, Loss: 0.0007, MCC: 0.3101
✅ 预测完成，结果已保存为 prediction.txt
→ P1
```

prediction.txt

```
-1
-1
1
1
-1
-1
1
-1
-1
1
-1
1
-1
-1
-1
1
-1
-1
1
-1
```

Conclusion

The second code, using pytorch(P2 Program),implements a binary classification neural network for protein sequence classification. It first reads training and test data from "train.dat" and "test.dat", then extracts **k-mer frequency features** (default k=2) to convert sequences into numerical representations. The dataset is split into **80% training and 20% validation**, with **class weights computed** to handle imbalance. A **three-layer neural network** is constructed with **256 and 128 hidden units**, **Leaky ReLU activations**, **dropout (20%)**, **L2 regularization**, and **Adam optimization**. The model is trained for **100 epochs** with a batch size of **32**, using **cross-entropy loss**. After training, it predicts labels for the test set, **mapping 0 to -1 and 1 to 1.**

From this implementation, I have learned that several training parameters can be optimized for better performance. **Epochs (100 epochs)** can improve learning, but excessive training may lead to **overfitting**, especially without early stopping. However, in theory, **a higher number of epochs** could further improve prediction performance, as **100 epochs might still be too low** for the model to fully converge.

Batch size (32) affects training stability—**larger batch sizes** can speed up training and stabilize updates, while **smaller batches** might help generalization but introduce more noise.

Dropout (20%) helps prevent overfitting, but tuning it within a **10%-50% range** could yield better results. Similarly, **L2 regularization (1e-5)** is currently small, and increasing it may further reduce overfitting, though too much could hinder learning.

Other hyperparameters also offer room for improvement. While the **Adam optimizer** is effective, alternatives like **SGD with momentum** could provide better generalization in some cases. However, **SGD may not be ideal for this dataset**, as it generally requires larger datasets and well-tuned learning rates to perform optimally. **Class weights** are currently determined by a simple ratio, but more advanced methods, such as **focal loss**, might handle **imbalanced datasets** more effectively. In **feature extraction (k-mers, k=2)**, using a **larger k** could capture more sequence context but would also **increase feature dimensionality**, requiring careful tuning for the best trade-off between information capture and computational efficiency.