

# 基于深度学习的新抗原智能筛选

## 一、实验背景.

新抗原 (Neoantigen) 是一类在癌细胞中表达，但在正常细胞中不表达或低表达的特异性蛋白质。它们由突变或异常表达的基因来编码，这些基因通常与癌症的发展有关。新抗原的出现是由于癌细胞的遗传不稳定性和高突变率导致的正常蛋白结构变化。

新抗原的重要性在于它们对免疫系统的作用。由于这些抗原在正常细胞中不存在或表达水平很低，因此免疫系统可以将其识别为外来物质，激发免疫应答以对抗癌细胞。这使得新抗原成为癌症疫苗和免疫治疗研究中的重要目标，特别是在个性化医疗和精准医疗的背景下。

新抗原的识别和利用是一个复杂的过程，涉及对癌细胞基因组的深入分析，以及对个体免疫系统的详细了解。通过这些分析，科学家可以设计出特定的免疫疗法，针对个体的癌症特征进行治疗。在现代生物技术领域，新抗原疫苗 (Neoantigen vaccine) 是一种对抗癌症的新手段。如何在众多的新抗原以及其序列中选择最合适的作为疫苗，便是我们需要研究的方向。

本次实验从患者原始肽池中提取的肽序列开始，采用生成器 (Generator) + 过滤器 (Filters) 的架构，采用蒙特卡罗方法，结合深度学习进行评估，最终对一个给定的九肽池进行扩张，再通过过滤模块进行 17 次过滤，最终形成一个完善的目标多肽池选出最适合作为新抗原疫苗的九肽。

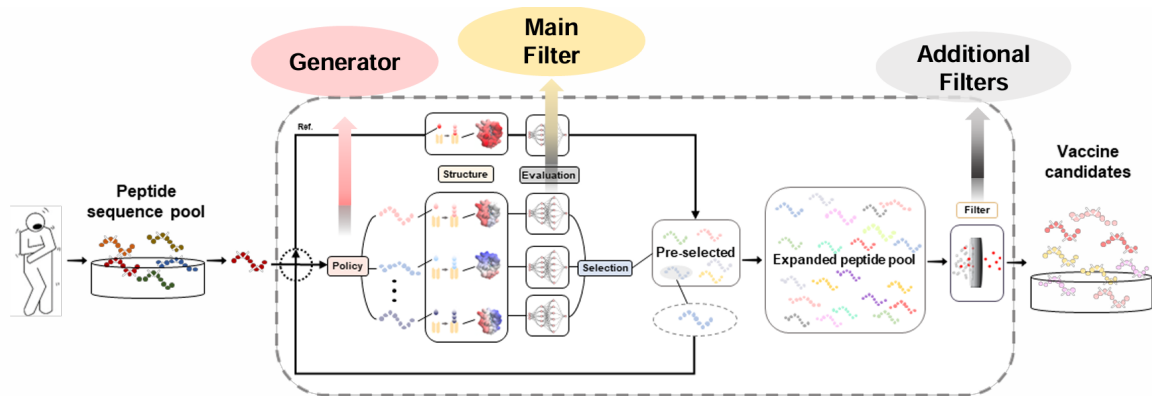


图 1. 整体实验框架

## 二、实验目的.

- 1、理解本次实验背后的生物学原理，掌握蒙特卡洛方法的原理。
- 2、学习如何采用 RNN、LSTM、GRU 三种模型，对 structure loss 进行深度学习。
- 3、完成从给定九肽序列的扩张、过滤过程，以及最终的数据分析。

## 三、实验方法和实验材料.

### 3.0 实验环境.

- 操作系统: Windows 2019.
- 电脑配置: 8 核, 26GB 内存.
- 显卡配置: TeslaT4, 16GB 显存.
- cuda 版本: 11.2
- python 版本: 3.10

### 3.1 RNN、LSTM、GRU 模型的训练.

打开 `RNN_model.ipynb` 文件，启动预先配置的环境，开始模型训练。我调整了超参，将 `num_epochs` 调整至 20，以 RNN 为例：

```
num_epochs, lr = 20, 0.01
dt = datetime.now()
print(f'现在是: {dt.year}年{dt.month}月{dt.day}日 {dt.hour}:{dt.minute}:
{dt.second}')
```

```
train_rnn_kfold(net, train_iter, 5, lr, num_epochs, try_gpu(), batch_size)
dt = datetime.now()
print(f'现在是: {dt.year}年{dt.month}月{dt.day}日 {dt.hour}:{dt.minute}:
{dt.second}')
```

训练完毕后保存参数，但是值得注意的是，训练不同的模型时需要将 `save` 函数的保存文件名修改。

```
torch.save(net.state_dict(), './rnn.params')
print(f'现在是: {dt.year}年{dt.month}月{dt.day}日 {dt.hour}:{dt.minute}:
{dt.second}, 任务已结束。')
```

最终文件夹下生成三个模型文件：







|  |                  |             |        |
|--|------------------|-------------|--------|
|  .ipynb_checkpoints | 2023/12/13 20:07 | 文件夹         |        |
|  gru.params         | 2023/12/18 0:36  | PARAMS 文件   | 101 KB |
|  lstm.params      | 2023/12/15 14:42 | PARAMS 文件   | 123 KB |
|  rnn.params       | 2023/12/13 20:39 | PARAMS 文件   | 58 KB  |
|  RNN_model.ipynb  | 2023/12/18 0:38  | Jupyter 源文件 | 630 KB |
|  RNN-raw.ipynb    | 2023/12/3 22:05  | Jupyter 源文件 | 8 KB   |

图 2. 模型文件

### 3.2 计算皮尔森系数.

通过计算皮尔森系数，我们可以探究标准答案和模型预测值之间的相关性。运行以下代码：

```
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
import numpy as np

def predict_single(seq, net, vocab, device):
    net.eval()
    state = net.begin_state(batch_size=1, device = device)
    seq = torch.tensor(vocab[[i for i in seq]], device = device).unsqueeze(0)
    # print(seq)
    with torch.no_grad():
        y, _ = net(seq, state)
    return y.squeeze(0).numpy()

def read_file(filename):
    scores, peptides = [], []
    with open(filename, 'r') as file:
        for line in file:
            parts = line.strip().split()
```

```

        scores.append(float(parts[0]))
        peptides.append(parts[1])
    return scores, peptides

def pearson_coefficient(x, y):
    n = len(x)
    sum_x, sum_y = sum(x), sum(y)
    sum_x2 = sum(xi**2 for xi in x)
    sum_y2 = sum(yi**2 for yi in y)
    sum_xy = sum(xi*yi for xi, yi in zip(x, y))
    numerator = n * sum_xy - sum_x * sum_y
    denominator = ((n * sum_x2 - sum_x**2) * (n * sum_y2 - sum_y**2))**0.5
    if denominator == 0:
        return 0
    return numerator / denominator

def calculate_pearson_coefficient(filename, plot_graph=False):
    original_scores, peptides = read_file(filename)
    predicted_scores = [predict_single(peptide, net, vocab, try_gpu()) for
    peptide in peptides]
    pearson_coeff = pearson_coefficient(original_scores, predicted_scores)
    if plot_graph:
        plot_regression(original_scores, predicted_scores)
    return pearson_coeff

def plot_regression(x, y):
    x = np.array(x).reshape(-1, 1)
    y = np.array(y)
    model = LinearRegression()
    model.fit(x, y)
    y_pred = model.predict(x)
    plt.scatter(x, y, color='blue')
    plt.plot(x, y_pred, color='red')
    plt.xlabel('Original Scores')
    plt.ylabel('Predicted Scores')
    plt.title('Linear Regression of Scores')
    plt.show()

filename = '../data/student_corr_test.txt'
print(calculate_pearson_coefficient(filename, True))

```

由于之前已经训练过模型，我没有选择 `load the params`，而是选择直接将代码中所有 `clone` 替换为 `net`。

保存生成的图片，进行数据分析。

### 3.3 将模型嵌入主程序.

以嵌入 GRU 为例，我们对 `loss.py` 进行嵌入。我们需要在这段代码中嵌入 `predict_single` 的模块。

```

def compute_loss(
    iedb_freqmat,
    prob_ref_freqmat,
    pep_seq,
    logfile,

```

```

w1=10,
w2=-0.1,
w3=None,
w4=args.weight_iedb,
):# tes
    if args.cdr_sequence == None:
        w3 = args.weight_cdr_dis
    else:
        w3 = args.weight_cdr
    cdr_loss = pep_TCR_loss(
        prob_ref_freqmat,
        pep_seq=pep_seq,
        cdr_seq=args.cdr_sequence,
        TCRpos=args.TCR_loss_pos,
    )
    iedb_loss = IEDB_loss(iedb_freqmat, pep_seq)
    print("", file=logfile)
    print("LOSS RESULTS of the {}".format(pep_seq), file=logfile)
    if args.cdr_sequence == None:
        print("****cdr loss --> {} given no cdr".format(w3 * cdr_loss),
file=logfile)
    else:
        print(
            "****cdr loss --> {} with cdr sequence as {}".format(
                w3 * cdr_loss, args.cdr_sequence
            ),
            file=logfile,
        )
    print("****iedb loss --> {}".format(w4 * iedb_loss), file=logfile)
    return (
        w3 * cdr_loss + w4 * iedb_loss
    ) # DOPE is negative, thus w2 should be negative

```

可以发现现在只考虑了 `cdr_loss` 和 `iedb_loss`。因此我们将 `predict_single` 所需要的各种参数嵌入这段代码之前。包括 `class vocab`、`def counter_corpus`、`class PositionwiseFFN`、`class RNNModel`、`def try_gpu`、`def read_sequence`、`def tokenize`。然后修改代码如下：

```

num_hiddens, num_layers = 64, 1
directory = './data/'
file = 'student_train.log'
lines = read_sequence(directory, file)
tokens = tokenize(lines)
tokens = [list(_) for _ in np.array(tokens)[: , 1]]
vocab = vocab(tokens)
rnn_layer = nn.GRU(len(vocab), num_hiddens, num_layers)
ffn_num_hiddens, linear1_num_hiddens, linear2_num_hiddens, linear3_num_hiddens =
32, 32, 32, 8
net = RNNModel(rnn_layer, len(vocab), ffn_num_hiddens,
                linear1_num_hiddens, linear2_num_hiddens, linear3_num_hiddens,
                dropout=0.5)
net.load_state_dict(torch.load('./gru.params'))

def predict_single(seq, net, vocab, device):
    net.eval()

```

```

state = net.begin_state(batch_size=1, device = device)
seq = torch.tensor(vocab[[i for i in seq]], device = device).unsqueeze(0)
# print(seq)
with torch.no_grad():
    y, _ = net(seq, state)
return (y.squeeze(0).numpy())[0]

def compute_loss(
    iedb_freqmat,
    prob_ref_freqmat,
    pep_seq,
    logfile,
    w1=10,
    w2=-0.1,
    w3=None,
    w4=args.weight_iedb,
):
    # test
    if args.cdr_sequence == None:
        w3 = args.weight_cdr_dis
    else:
        w3 = args.weight_cdr
    cdr_loss = pep_TCR_loss(
        prob_ref_freqmat,
        pep_seq=pep_seq,
        cdr_seq=args.cdr_sequence,
        TCRpos=args.TCR_loss_pos,
    )
    iedb_loss = IEDB_loss(iedb_freqmat, pep_seq)
    structure_loss = predict_single(pep_seq, net, vocab, try_gpu())
    print("", file=logfile)
    print("LOSS RESULTS of the {}".format(pep_seq), file=logfile)
    if args.cdr_sequence == None:
        print("****cdr loss --> {} given no cdr".format(w3 * cdr_loss),
            file=logfile)
    else:
        print(
            "****cdr loss --> {} with cdr sequence as {}".format(
                w3 * cdr_loss, args.cdr_sequence
            ),
            file=logfile,
        )
    print("****iedb loss --> {}".format(w4 * iedb_loss), file=logfile)
    print("****structure loss --> {}".format(structure_loss), file=logfile)
    return (
        w3 * cdr_loss + w4 * iedb_loss + structure_loss
    ) # DOPE is negative, thus w2 should be negative

```

至此，我们完成了 GRU 模型的嵌入。我们把 `gru.params` 和 `student_train.log` 导入工程所在目录即可。

### 3.4 从给定的九肽开始运行代码.

在配置的环境下执行命令：

```
python main.py -o out -start_pep VMNILLQYV
```

```
(AI4LS) C:\Users\xieji\Desktop\P3>python main.py -o out -start_pep VMNILLQYV
# Namespace(l=9, mhc='HLA-A*0201', start_pep='VMNILLQYV', start_fre=None, frequency_weight='1_1', rest_mut_pos=None, fre
_mut_pos='1_2_3_4_5_6_7_8_9', o='out', verbose=False, tolerance=None, mutation_rate='3-1', T_init=0.025, half_life=1000,
metropolis_hasting=False, steps=10, add_booster=False, booster_num=3, frequency_change_rate=0.3, mutant_times=10, TCR_l
oss_pos='1_2_3_4_5_6_7_8_9', cdr_sequence=None, weight_cdr_dis=0.1, weight_cdr=2, weight_iedb=0.05, nomemory=False, freq
notchange=False)
```

图 3. 运行结果

在文件目录下产生了 `out` 文件夹，其中包括生成的数据文件。



|   |                  |      |        |
|---|------------------|------|--------|
|  VMNILLQYV_Details_logfile.txt | 2023/12/24 15:26 | 文本文档 | 104 KB |
|  VMNILLQYV_Frequencymap.txt    | 2023/12/24 15:26 | 文本文档 | 4 KB   |

图 4. 输出文件

打开 `Extract_peptide_script` 目录，运行以下命令：

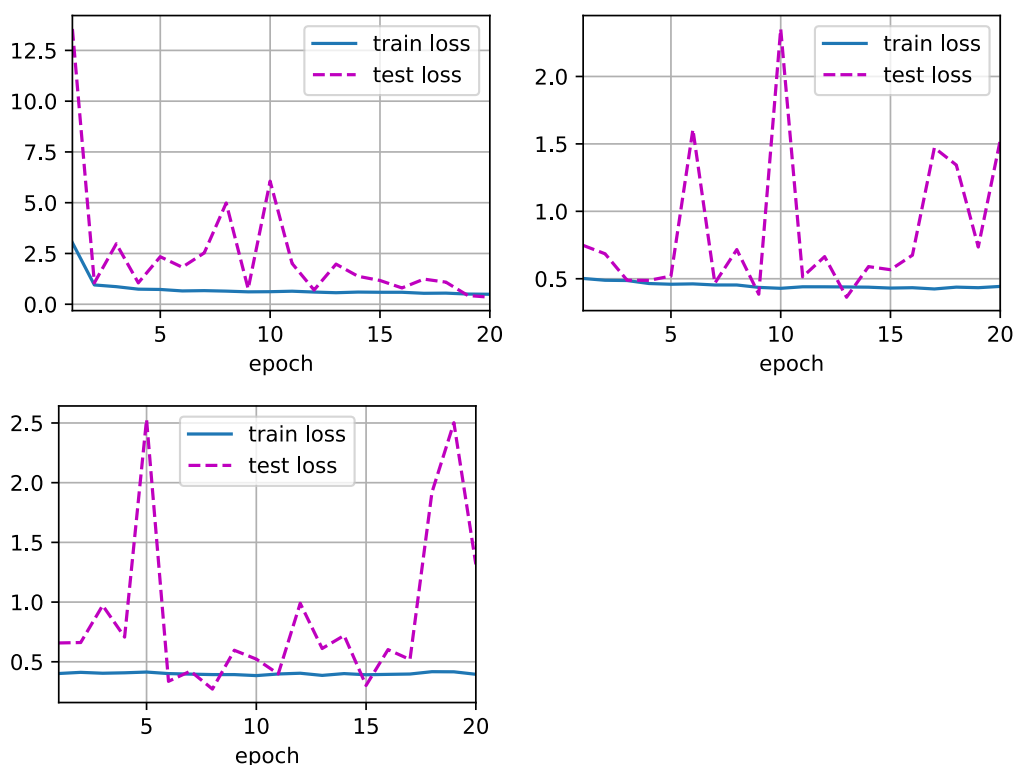
```
python extract_step_loss_sequence_version7.py C:\Users\xieji\Desktop\P3\out
VMNILLQYV_Details_logfile.txt
```

第一个参数填写生成文件目录，第二个参数填写需要处理的文件名。将生成的文件信息使用 [NetMHC](#) 进行验证和处理分析即可。

## 四、实验数据记录和处理.

### 1、深度学习语言模型训练和模型测试.

RNN 训练结果图：



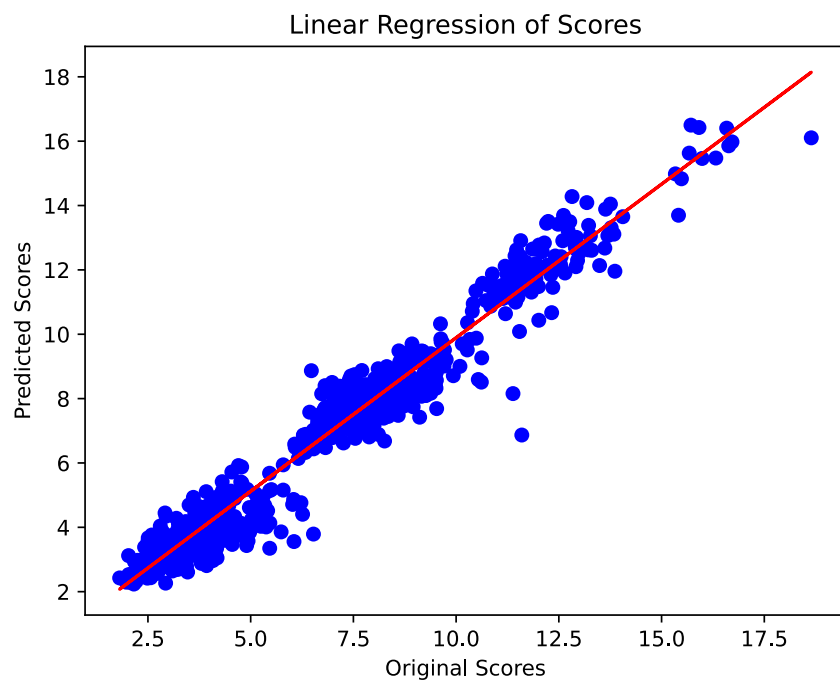
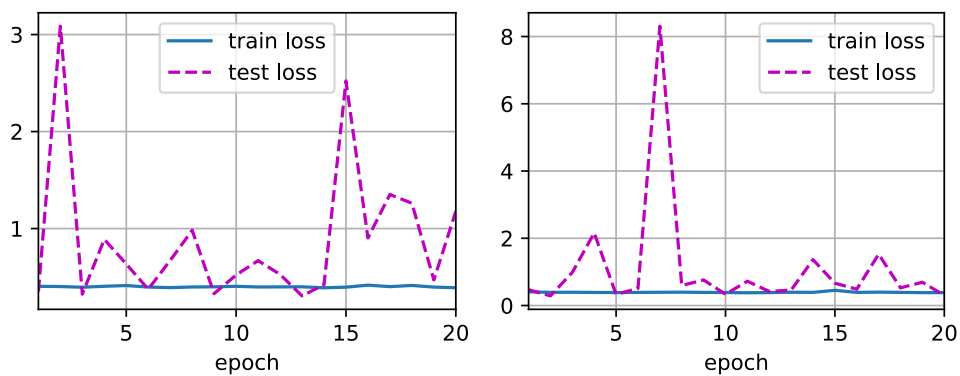
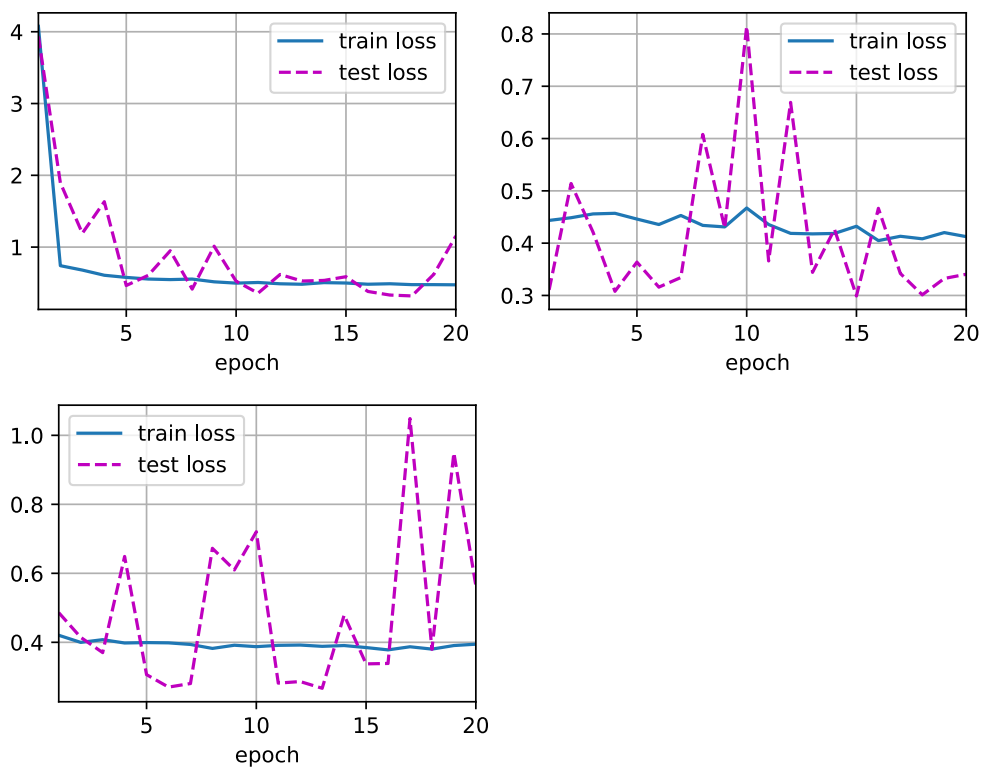


图 5. RNN 训练结果

LSTM 训练结果图:



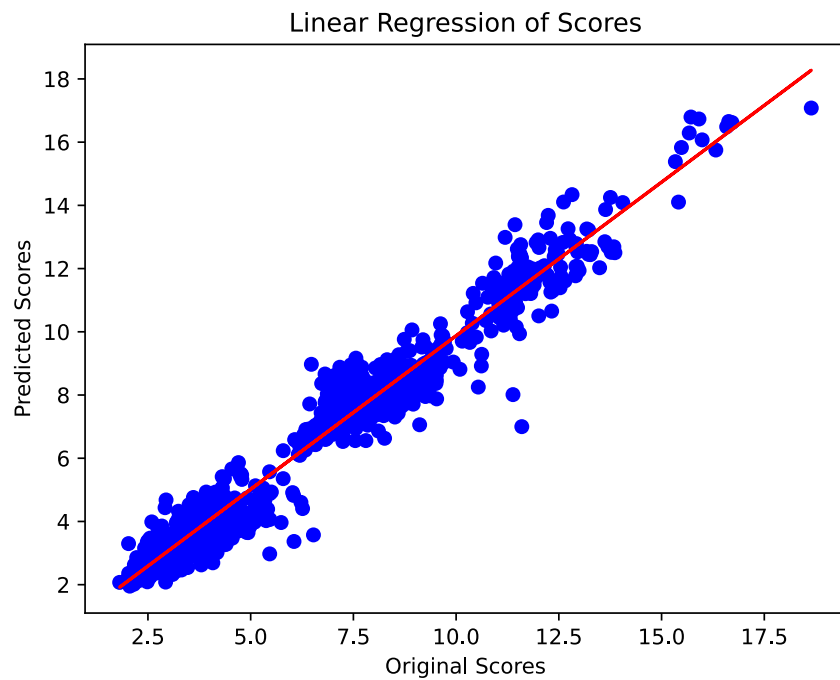
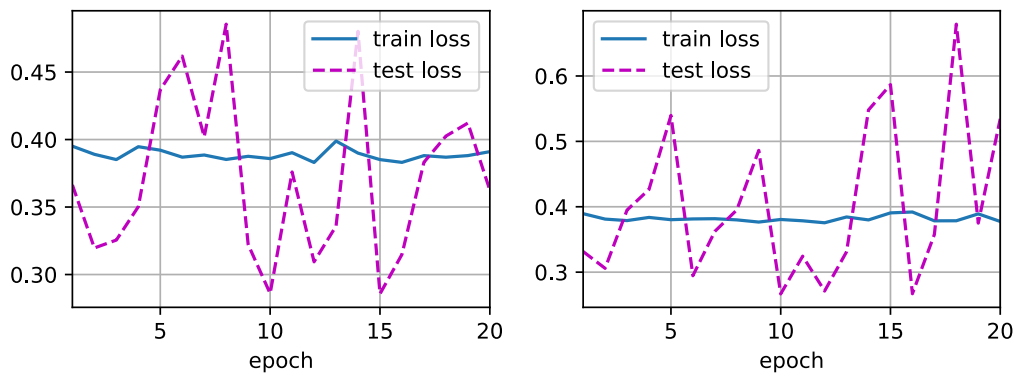
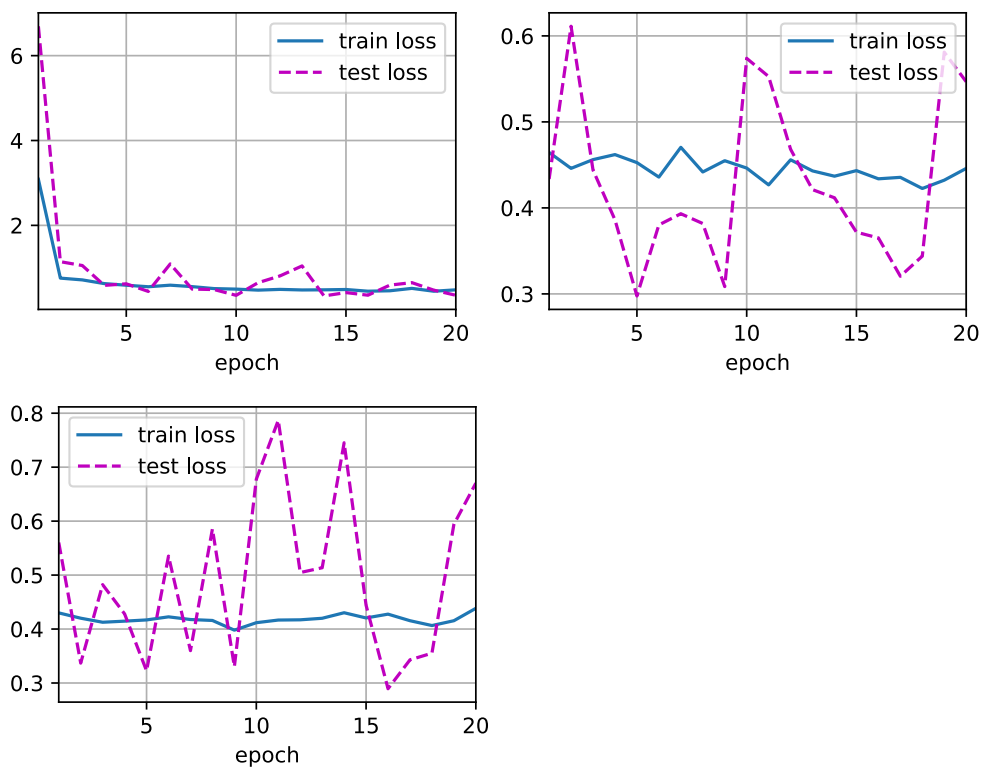


图 6. LSTM 训练结果

GRU 训练结果图：





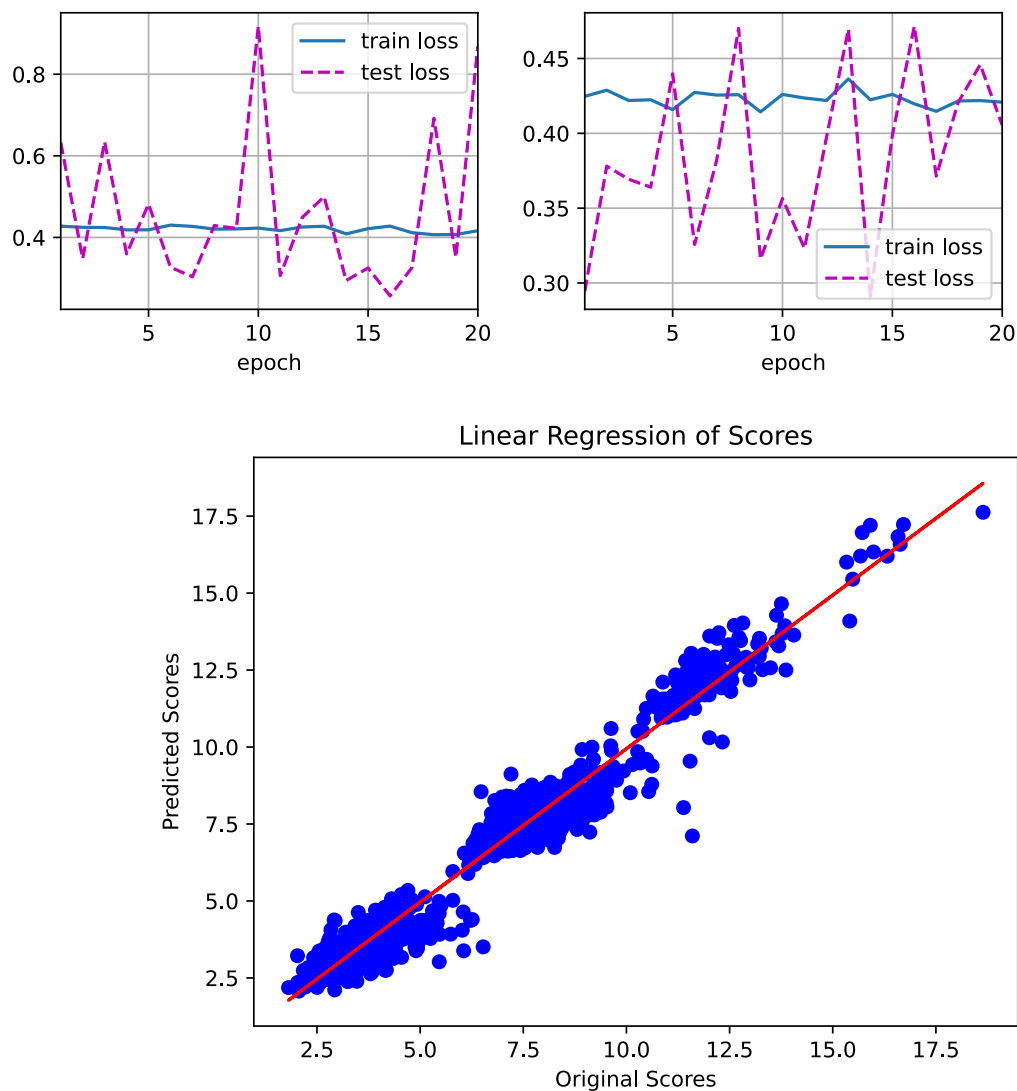


图 7. LSTM 训练结果

## 2、深度学习语言模型嵌入优化系统.

运行后得到的文件 `VMNILLQYV_Details_logfile_step_loss_accept.log` :

```
0 21.072 VMNILLQYV *
1 20.941766959191753 VMNPLAQLV
1 20.16844457763217 VMNPLVQSV
1 19.089615007517356 VMNELIQTV *
1 20.453821000779175 VMNGLIQLV
1 20.170039371550228 VMNDLQQVV
2 14.125407280336722 VMFSLIQVV
2 12.731836890311879 VMLKLIQVV *
3 12.289129757601508 VMLKKIQVL *
4 12.062247461050857 VLLKKVQVL *
4 12.062247461050857 VLLKKVQVL *
5 11.503181154257708 VLLEKVQLL
5 11.096609647956706 VLLGKVQSL *
5 11.144595907525451 VLLDKVQWL
7 10.81791663654113 VLLGKVQQL *
7 10.81791663654113 VLLGKVQQL *
8 10.629766738141642 VLLGTVQQL
8 10.629766738141642 VLLGTVQQL
8 10.629766738141642 VLLGRVQQL *
#Total 19
```

图 8. VMNILLQYV\_Details\_logfile\_step\_loss\_accept.log

NetMHC 的评估结果：

| pos | HLA        | peptide   | Core      | Offset | I_pos | I_len | D_pos | D_len | iCore     | Identity | 1-log50k (aff) | Affinity(nM) | %Rank | BindLevel |
|-----|------------|-----------|-----------|--------|-------|-------|-------|-------|-----------|----------|----------------|--------------|-------|-----------|
| 0   | HLA-A*0201 | VMNELIQTV | VMNELIQTV | 0      | 0     | 0     | 0     | 0     | VMNELIQTV | PEPLIST  | 0.797          | 9.01         | 0.10  | <= SB     |
| 0   | HLA-A*0201 | VMFSLIQVV | VMFSLIQVV | 0      | 0     | 0     | 0     | 0     | VMFSLIQVV | PEPLIST  | 0.796          | 9.08         | 0.10  | <= SB     |
| 0   | HLA-A*0201 | VLLDKVQWL | VLLDKVQWL | 0      | 0     | 0     | 0     | 0     | VLLDKVQWL | PEPLIST  | 0.787          | 9.97         | 0.12  | <= SB     |
| 0   | HLA-A*0201 | VMNGLIQLV | VMNGLIQLV | 0      | 0     | 0     | 0     | 0     | VMNGLIQLV | PEPLIST  | 0.750          | 14.90        | 0.20  | <= SB     |
| 0   | HLA-A*0201 | VLEKVVQLL | VLEKVVQLL | 0      | 0     | 0     | 0     | 0     | VLEKVVQLL | PEPLIST  | 0.743          | 16.05        | 0.25  | <= SB     |
| 0   | HLA-A*0201 | VMNPLVQSV | VMNPLVQSV | 0      | 0     | 0     | 0     | 0     | VMNPLVQSV | PEPLIST  | 0.718          | 21.16        | 0.30  | <= SB     |
| 0   | HLA-A*0201 | VMLKLIQVV | VMLKLIQVV | 0      | 0     | 0     | 0     | 0     | VMLKLIQVV | PEPLIST  | 0.717          | 21.35        | 0.30  | <= SB     |
| 0   | HLA-A*0201 | VMNILLQYV | VMNILLQYV | 0      | 0     | 0     | 0     | 0     | VMNILLQYV | PEPLIST  | 0.710          | 22.96        | 0.30  | <= SB     |
| 0   | HLA-A*0201 | VLLGKVQSL | VLLGKVQSL | 0      | 0     | 0     | 0     | 0     | VLLGKVQSL | PEPLIST  | 0.681          | 31.39        | 0.40  | <= SB     |
| 0   | HLA-A*0201 | VMNPLAQLV | VMNPLAQLV | 0      | 0     | 0     | 0     | 0     | VMNPLAQLV | PEPLIST  | 0.639          | 49.71        | 0.60  | <= WB     |
| 0   | HLA-A*0201 | VMNDLQQVV | VMNDLQQVV | 0      | 0     | 0     | 0     | 0     | VMNDLQQVV | PEPLIST  | 0.619          | 61.78        | 0.80  | <= WB     |
| 0   | HLA-A*0201 | VLLGTVQQL | VLLGTVQQL | 0      | 0     | 0     | 0     | 0     | VLLGTVQQL | PEPLIST  | 0.603          | 73.62        | 0.90  | <= WB     |
| 0   | HLA-A*0201 | VLLGTVQQL | VLLGTVQQL | 0      | 0     | 0     | 0     | 0     | VLLGTVQQL | PEPLIST  | 0.603          | 73.62        | 0.90  | <= WB     |
| 0   | HLA-A*0201 | VLLGKVQQL | VLLGKVQQL | 0      | 0     | 0     | 0     | 0     | VLLGKVQQL | PEPLIST  | 0.602          | 74.33        | 0.90  | <= WB     |
| 0   | HLA-A*0201 | VLLGKVQQL | VLLGKVQQL | 0      | 0     | 0     | 0     | 0     | VLLGKVQQL | PEPLIST  | 0.602          | 74.33        | 0.90  | <= WB     |
| 0   | HLA-A*0201 | VLLGRVQQL | VLLGRVQQL | 0      | 0     | 0     | 0     | 0     | VLLGRVQQL | PEPLIST  | 0.573          | 101.66       | 1.10  | <= WB     |
| 0   | HLA-A*0201 | VMLKKIQVL | VMLKKIQVL | 0      | 0     | 0     | 0     | 0     | VMLKKIQVL | PEPLIST  | 0.547          | 134.46       | 1.30  | <= WB     |
| 0   | HLA-A*0201 | VLLKKVQVL | VLLKKVQVL | 0      | 0     | 0     | 0     | 0     | VLLKKVQVL | PEPLIST  | 0.488          | 254.34       | 1.90  | <= WB     |
| 0   | HLA-A*0201 | VLLKKVQVL | VLLKKVQVL | 0      | 0     | 0     | 0     | 0     | VLLKKVQVL | PEPLIST  | 0.488          | 254.34       | 1.90  | <= WB     |

图 9. NetMHC 评估结果

得到扩展的九肽池中比初始九肽更好的比率为  $\frac{7}{18} \times 100\% = 38.89\%$ 。

## 五、实验分析.

### 1、深度学习语言模型训练和模型测试.

可以发现，三个模型经过了几十个 epoch 的训练后，train loss 都非常好地收敛在了 0.4 左右，但是观察其 test loss 却别有端倪。尽管后期 train loss 几乎收敛，但是 test loss 却有明显的突变，尤其是 RNN，其 test loss 一度突变到 6。

之所以出现这样的问题，我认为是 RNN 存在梯度爆炸的问题，尽管在训练集上有所收敛，但是其权重在训练集的微小变化可能会对应于测试集的大梯度变化，因此出现损失突变的情况。而在下一个 epoch 中，也可以发现 test loss 又恢复了原状。也正是如此，我没有选择 RNN 作为优化系统的模型。

对于 LSTM 和 GRU，可以发现 LSTM 在训练集上有更好的表现，这是因为其有更多的参数和更强的拟合能力。但是在验证集中，我发现 LSTM 有更大的突变范围，效果也没有 GRU 好。我认为这是 LSTM 的参数更多，出现了过拟合的现象，而 GRU 会表现得更好。因此最终我选了 GRU 作为优化模型。

观察其相关系数，发现都在 0.98 附近，这也直观的说明了我们训练出来的模型的拟合效果都是非常不错的。

### 2、深度学习语言模型嵌入优化系统.

从实验结果中可以看出，structure loss 成功地被嵌入到了算法中，在搜索的过程中，被接受的新九肽大概让 Loss 降低了 10 左右。

扩展的九肽池中比初始九肽更好的比率为 38.89%，这说明了整个算法确实扩大了有效的九肽池，也搜索出了很多更优的抗原序列。

## 六、讨论和心得.

本次实验相对于上次更加容易一些。因为训练内容简单，而且有 jupyter 文档的辅助，我大概只花了不到 1 个小时就完成了训练。这次实验的主要难点在于将训练好的模型嵌入至 Loss.py 中，期间出现了各种各样的错误需要自己一步一步调试和排查。虽然过程曲折，但是收获还是非常丰富的，比如我学会了如何将模型的参数保存至一个本地文件中，以及如何导入模型的参数，还有如何读懂 python 的报错等。

当然，收获最大的，还是我实际操作了三个序列模型的训练，也让我能够直观的观察其特性以及优缺点。这给我未来的科研之路必定会打下深厚的基础。

