人工智能期末项目

基于开放环境的文本分类任务

中山大学计算机学院　计算机科学与技术

施天予　　孙奥远

19335174　　19335177

目录

一、概述

本次期末项目的内容为基于开放环境的文本分类问题。开放环境学习范式主要有辅助数据、微调、知识蒸馏、参数重构、元学习等。在本次项目中我们小组使用参数重构的策略进行了基于开放环境的文本分类问题的求解。我们实现了 TextCNN、LSTM 两个深度学习模型, 尝试了 EWC、MAS、SI 三种不同的参数重构方法。

二、实验原理

1. 模型原理

(i) 卷积神经网络

卷积神经网络用于文本分类的核心思想是抓取文本的局部特征，通过不同的卷积核尺寸来提取文本的 N-gram 信息，然后通过最大池化操作来突出各个卷积操作提取的最关键信息，拼接后通过全连接层对特征进行组合，最后通过交叉熵损失函数来训练模型。

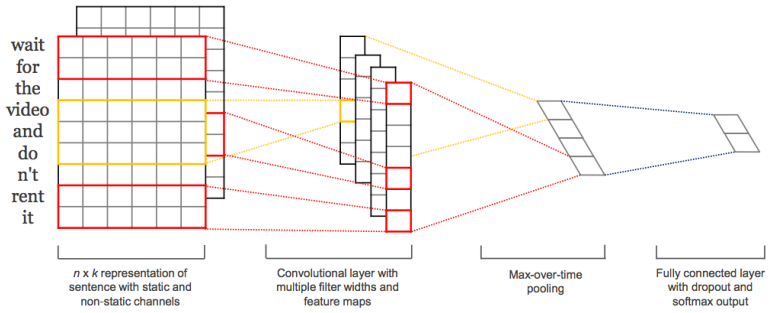

Figure 1: Model architecture with two channels for an example sentence.

图 1: textCNN 模型

模型的第一层是 Embedding 层，预训练的词嵌入可以利用其他语料库得到更多的先验知识，经过模型训练后能够得到和当前任务最相关的文本特征。

模型的第二层是卷积层，不同的核可以获得不同范围内词的关系，获得的是纵向的差异信息，也就是在一个句子中不同范围的词出现会带来什么信息，相当于 n-gram。每个卷积尺寸下又有多个相同的卷积核，原因是卷积神经网络学习的是卷积核中的参数，每个核都有自己的关注点，这些卷积核可以在同一个窗口中学习到互补的特征，多个卷积核就能学习到不同的信息。

模型的第三层是最大池化层，即为从每个滑动窗口产生的特征向量中筛选出一个最大特征，然后将这些特征拼接起来构成向量表示。在短文本分类中，每条文本中都会有一些对分类无用的信息，而最大池化可以突出最重要的关键词以帮助模型更容易找到对应的类别。

(ii) 循环神经网络

对于一个分类问题，我们使用的 RNN 循环神经网络是一种`Sequence-to-Vector`结构，即输入一个序列，输出一个单独的数据，通常在最后一个序列上进行输出变换。

然而传统的 RNN 结构共享一组参数，梯度在反向传播过程中不断连乘，数值不是越来越大就是越来越小，就会出现**梯度爆炸**或是**梯度消失**的情况。LSTM 长短期记忆网络解决了这个问题，其神经元加入了输入门、遗忘门、输出门和内部记忆单元。因此本次项目我们使用了 LSTM 模型进行文本分类，模型结构如图2所示。

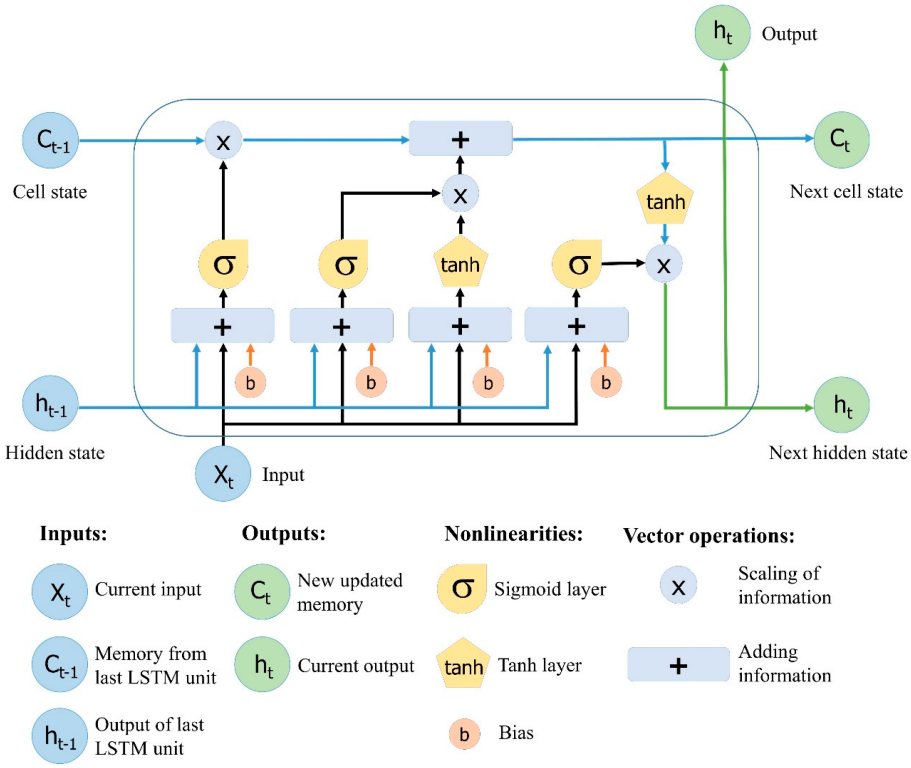

图 2: LSTM 单元

2. 参数重构

终身学习被定义为一种能够从连续的信息流中学习的自适应算法，随着时间的推移，这些信息逐渐可用，并且所要学习的任务数量不是预先定义的。关键的是，新信息的容纳应该在没有灾难性遗忘或干扰的情况下发生。为避免发生灾难性遗忘，我们可以使用参数重构（类似正则化）的方法：通过对神经网络学习的参数加以约束来减轻灾难性遗忘。实际上，通过在损失函数中加惩罚项使模型参数受限地变动，阻碍在旧任务上重要参数的变化。

(i) EWC

EWC（Elastic Weight Consolidation）是一种加了相对 Previous Task 参数的 L2 正则化方法。其核心思想是找到不同参数不同的重要性，对 Previous Task 重要的参数就更新幅度小一些，不重要的参数就可以多更新一些，这相对全部都加同样约束的 L2 正则就有了先进之处。对于计算模型每个参数的重要程度，EWC 算法使用 Fisher 矩阵的对角线元素来进行估计：计算当前模型在 Previous Task 上的 loss（损失函数使用交叉熵），然后将反向传播的梯度取平方再除以在 Previous Task 的总任务数。然而 EWC 算法在任务差异较大时不适用，因为其忽略了参数之间的关联性。

$$\mathcal{I}(\theta) = \frac{1}{N} \sum_{(x,y)_i \in \mathcal{A}} \left(\underbrace{\frac{\partial l_{\mathbf{LL}}(\theta|(x,y)_i)}{\partial \theta}}_{\text{Gradient}} \right)^2$$

图 3: Fisher 矩阵对角线元素的计算公式

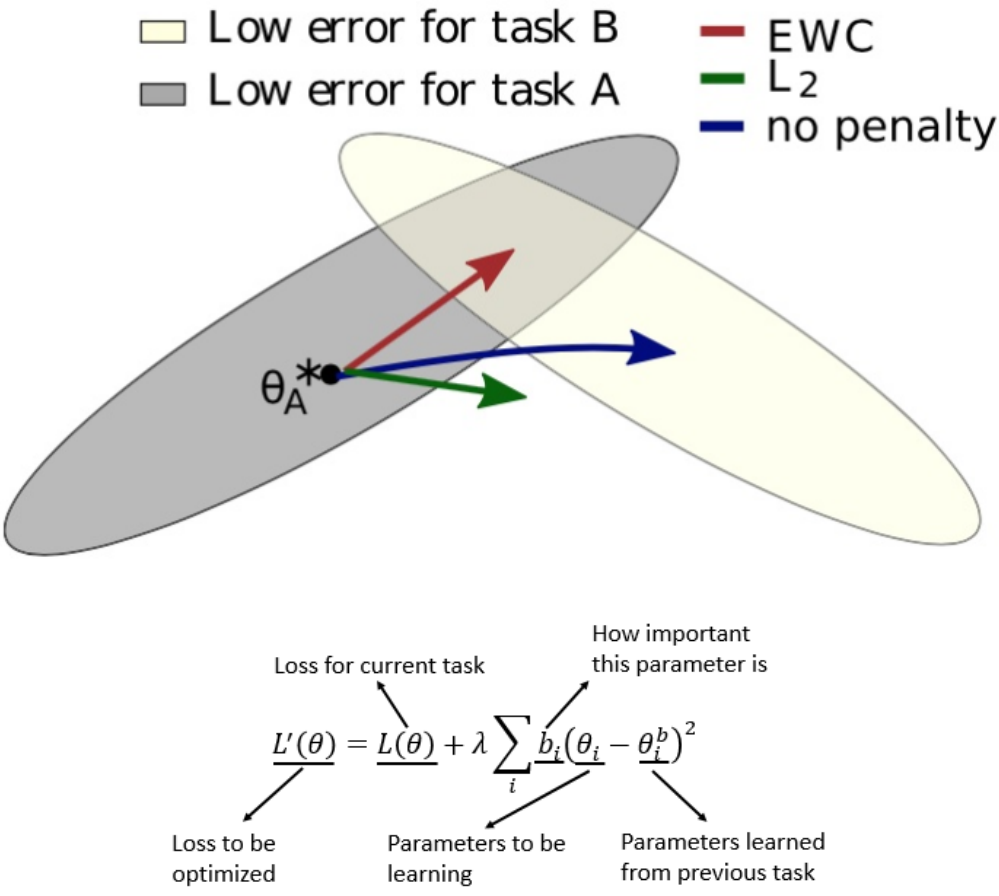

图 4: EWC 算法

(ii) MAS

MAS（Memory Aware Synapses）的大体思想与 EWC 算法基本一致，都是找到不同参数不同的重要性，重要的参数更新幅度小一些，不重要的参数更新幅度大一些。不过不同的是，它不是用 Fisher 矩阵计算参数的重要程度，而是使用 Omega 矩阵。

$$L(\theta) = L_n(\theta) + \lambda \sum_{i,j} \Omega_{ij}(\theta_{ij} - \theta^*_{ij})^2$$

图 5: MAS 算法

MAS 算法将当前模型在 Previous Task 上每个输出结果取平方和，再求均值作为 loss（不需要 label），然后取反向传播的梯度的绝对值除以在 Previous Task 的总任务数，即可估计每个参数的重要程度。

$$\Omega_{ij} = \frac{1}{N} \sum_{k=1}^{N} \| g_{ij}(x_k) \|$$

图 6: Omega(Ω) 权重的计算公式

(iii) SI

为了解决神经网络中的持续学习的问题，突触智能是给神经网络中的每个突触一个重要性参数，当训练一个新的任务时，突触智能惩罚重要参数的变化，避免旧的记忆被覆盖。

神经网络的训练过程可以由一条在参数空间的轨迹 $\theta(t)$ 来描述，如果参数轨迹的最终点落在损失函数 Loss 的最小值附近，则说明训练成功。

神经网络由若干神经元以及和神经元相关的参数组成。若神经网络的参数在时间 t 有一个微小的变化 $\sigma(t)$, 损失函数值的变化能用 $g = \frac{\partial L}{\partial \theta}$ 表示。

$$L(\theta(t) + \sigma(t)) - L(\theta(t)) \approx \sum_k g_k(t) * \sigma_k(t)$$

其中 $g_k(t)$ 为损失函数对第 k 个参数的梯度,$\sigma_k(t)$ 表示第 k 个参数的变化量。

为了计算参数在参数空间中按某条轨迹对损失函数造成的所有微小变化的总和，需要计算参数轨迹在梯度上的路径积分，假设初始点为 t_0，终点为 t_1，从 t_0 到 t_1，损失函数的变化为

$$\int_C g(\theta(t))d\theta = \int_{t_0}^{t_1} g(\theta(t)) * \theta'(t)dt = L(\theta(t_1)) - L(\theta(t_0))$$

在任务 u 下，单个权值的变化对损失函数的影响可以表示如下

$$\int_{t^{u-1}}^{t^u} g(\theta(t)) * \theta'(t)dt = \sum_k \int_{t^{u-1}}^{t^u} g_k(\theta(t))\theta'_k(t)dt = -\sum_k \omega_k^u = L(\theta(t^u)) - L(\theta(t^{u-1}))$$

$-\omega_k^u$ 就是第 k 个权值的调整造成的损失函数的变化，前面加一个负号的原因在于，在训练的过程中，损失函数值是在减小。由上式得到

$$\omega_k^u = -\int_{t^{u-1}}^{t^u} g_k(\theta(t))\theta'_k(t)dt$$

因此可以利用训练过程中产生的梯度与参数更新量的积进行累加，再取负来近似 ω_k^u。

那么假定在训练任务 u 时，第 k 个参数的路径积分为 ω_k^u，在训练过程中，ω_k^u 越大，说明第 k 个参数在任务 u 的训练过程中调整越大，因此说明第 k 个参数对任务 u 越重要。论文中权值的重要性为

$$\Omega_k^u = \sum_{v<u} \frac{\omega_k^v}{(\Delta_k^v)^2 + \xi}$$

上式中 $\Delta_k^v = \theta_k(t^v) - \theta_k(t^{v-1})$ 是为了确保正则项具有和损失函数相同的单位尺度，参数 ξ 是为了防止 Δ_k^v 为 0。将权值的重要性加入损失函数中。

$$L'_u = L_u + c\sum_k \Omega_k^u(\hat{\theta}_k - \theta_k)^2$$

$\hat{\theta}_k$ 为上一个任务训练结束时得到的第 k 个任务的取值。新的损失函数根据参数的重要程度抑制重要参数的调整，越重要的参数，让它的变化越小。

三、方法

1. 流程图

本次实验的整体流程图如图7所示。

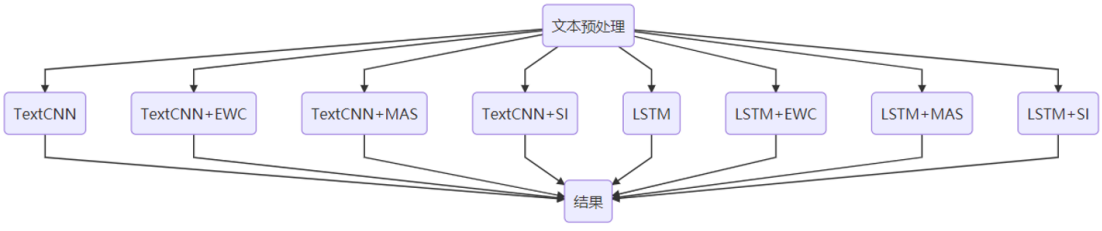

图 7: 流程图

2. 文本处理

本次项目的文本处理与期中项目大同小异，但不同的是本次的数据集共有 16 个文件夹，还要每 4 个结合在一起再划分训练集和测试集。另外，本次的数据集的格式相对复杂，脏数据也特别多。与之前相同的文本处理过程就不再详细介绍了，主要步骤如下：

- 脏数据清洗
- 分词
- 统一大小写
- 去除停用词

值得注意的是，本次项目我们没有使用词形还原、词干提取等操作，因为我们使用了斯坦福的 glove.6B.300d.txt 预训练词向量 http://nlp.stanford.edu/data/glove.6B.zip，其对于一个单词的不同形式已经给出了不同的预训练词向量。经检验发现如果使用词形还原、词干提取等方法反而会使模型得到更差的结果。

使用 Glove 预训练词向量，需要经过以下几个步骤：

1. 给训练集中的每个单词一个序号，构建一个 word-id 的词典。
2. 将每个文档的单词转为序号。
3. 统一每个文档的长度。通过对数据集的观察，最后选择 80 作为每个文档的长度进行 padding。将过长的文档截断，过短的文档用 0 补齐。
4. 使用词典和 Glove 预训练词向量构建一个权重矩阵。使每个 id 对应该单词的 word vector。

最后，在 CNN 和 RNN 中使用 nn.Embedding 载入 Glove 预训练权重模型，就可以完成整个词嵌入的过程。

3. 模型结构

(i) 卷积神经网络

搭建的 TextCNN 模型如下：

词嵌入层-> 卷积层-> 激活层-> 最大池化层-> 全连接层

1. 词嵌入层使用预训练的词向量，词向量维度为 300，词表大小为 50107。输入为 (batch_size, seq_len) 的二维 Tensor，经过词嵌入层，每个句子的一维向量表示转换为二维向量表示，二维 Tensor 转变成 (batch_size，seq_len, embedding_dim) 的三维 Tensor。

2. 卷积层使用一维卷积，使用卷积核大小分别为 (3,4,5) 的三种卷积核，每种卷积核有 128 个。这样经过卷积层之后，对应每种卷积核都会有 128 个输出，对每个输出进行激活函数激活。

3. 第三层为最大池化层，将卷积激活层的每个输出向量进行最大池化，利用一个值代表一个向量，将最大池化层的输出拼接起来，得到一个 384 维的向量。

4. 最后一层为全连接层, 输入一个 384 维的向量，输出一个 4 维的向量，代表属于四个种类的概率。

TextCNN 模型的部分结构如下, 没有包括激活层以及最大池化层

```
TextCNN(
  (embedding): Embedding(50107, 300)
  (conv1d_list): ModuleList(
    (0): Conv1d(300, 128, kernel_size=(3,), stride=(1,))
    (1): Conv1d(300, 128, kernel_size=(4,), stride=(1,))
    (2): Conv1d(300, 128, kernel_size=(5,), stride=(1,))
  )
  (linear): Linear(in_features=384, out_features=4, bias=True)
  (dropout): Dropout(p=0.5, inplace=False)
)
```

图 8: TextCNN 模型

TextCNN 模型的主要参数如表1所示。

wordlist size	50107
embedding dim	300
kernel num	128
kernel size	(3,4,5)
layer num	1
dropout	0.5

表 1: TextCNN 模型参数

(ii) 循环神经网络

我们搭建的 LSTM 模型如下：

1. 首先使用 Glove 预训练词向量通过 nn.Embedding 词嵌入，输入的二维 Tensor 变成了一个 `(batch_size, seq_len, embedding_dim)` 的三维 Tensor。
2. 单层隐藏层大小为 64 的 LSTM
3. 最后通过一层全连接层完成 4 分类

中间加入了 dropout，使模型的参数以一定概率不工作，防止过拟合。

```
RNN(
  (word_embed): Embedding(50107, 300)
  (rnn): LSTM(300, 64, batch_first=True)
  (dropout): Dropout(p=0.5, inplace=False)
  (out): Linear(in_features=64, out_features=4, bias=True)
)
```

图 9: LSTM 模型

LSTM 模型的主要参数如表2所示。

wordlist size	50107
embedding dim	300
hidden size	64
layer num	1
dropout	0.5

表 2: LSTM 模型参数

4. 参数重构

参数重构的方法类似于正则化，通过对神经网络学习的参数加以约束来减轻灾难性遗忘。也就是说，在损失函数中加惩罚项使模型参数受限地变动，阻碍在旧任务上重要参数的变化。

(i) Baseline

设置 Baseline 的目的是为了给接下来的三种参数重构方法作为参考，观察我们的方法是否有效。Baseline 的方法也十分简单，只需将模型在一个训练集练完后立即就在下一个训练集上训练，不需要计算参数的重要程度，也不用改变 loss。这种方法得到的平均准确率应为最后模型的下限。

(ii) EWC

EWC 利用 Fisher 矩阵计算模型参数的重要程度，当模型在一个训练集上训练完后，我们需要将模型在之前所有练过的数据集的验证集上进行测试，得到参数重要程度的 EWC 权重矩阵。

我们将 EWC 设置为一个 class，其共有两个重要的函数，calculate_importance 用于计算参数的重要程度，penalty 用于计算惩罚项。在 calculate_importance 中更新 EWC 权重矩阵的过程大致如下:

1. 计算模型的预测结果
2. 利用交叉熵损失函数计算 loss，并反向传播计算每个更新参数的梯度（只计算梯度不对参

数更新）

3. 将每个参数的梯度平方，再除以验证集中数据的个数（否则会因为验证集数据量不同而造成计算结果的差异），结果加到该参数的重要程度矩阵中

最后的加和结果就是参数重要程度的 EWC 矩阵。

penalty 函数将模型训练阶段时当前模型的参数与每个训练集练完时模型参数的差的平方，乘以 EWC 矩阵，再把所有参数的计算结果求和得到 lll_loss。将 lll_loss 乘以超参数 λ，加上原本模型训练时的 loss，就可作为新的 loss 在训练阶段更新模型的参数，达到固定重要参数的目的。EWC 的超参数 λ 最后经检验设置为 100 时效果最佳。

这里还有一个需要关注的点，在验证集计算 loss 时需要将模型改为 eval 模式。然而 RNN 在 eval 模式下不能计算 loss，所以需要设置为 train 模式并把 dropout 关掉，以免影响预测结果。CNN 不会有这个问题，设置为 eval 模式即可。

(iii) MAS

MAS 的方法与 EWC 大同小异，利用 Omega 矩阵计算模型参数的重要程度，当模型在一个训练集上训练完后，我们需要将模型在之前所有练过的数据集的验证集上进行测试，得到参数重要程度的 MAS 权重矩阵。

与 EWC 相同,我们将 MAS 设置为一个 class,其也有两个重要的函数 calculate_importance 和 penalty。在 calculate_importance 中更新 MAS 权重矩阵的过程大致如下:

1. 计算模型的预测结果
2. 计算预测结果（第二维是 4 的 Tensor）的平方和，再求平方和的均值作为 loss，并反向传播计算每个更新参数的梯度。与 EWC 不同的是，MAS 在验证集计算 loss 时不需要 label。
3. 取每个参数梯度的绝对值除以验证集中数据的个数，结果加到该参数的重要程度矩阵中

最后的加和结果就是参数重要程度的 MAS 矩阵。

penalty 函数与 EWC 相同。将模型训练阶段时当前模型的参数与每个训练集练完时模型参数的差的平方，乘以 MAS 矩阵，再把所有参数的计算结果求和得到 lll_loss。将 lll_loss 乘以超参数 λ，加上原本模型训练时的 loss，就可作为新的 loss 在训练阶段更新模型的参数，达到固定重要参数的目的。MAS 的超参数 λ 最后经检验设置为 0.1 时效果最佳。

与 EWC 一样，RNN 在验证集计算 loss 时将模型设置为 train 模式并把 dropout 关掉，CNN 使用 eval 模式。

(iv) SI

由前面的讨论得知，新的损失函数的形式为

$$L'_u = L_u + c\sum_k \Omega_k^u(\hat{\theta}_k - \theta_k)^2$$

实验中选择的 c 的值为 100。定义一个类 si 计算上述损失函数中的惩罚项 $c\sum_k \Omega_k^u(\hat{\theta}_k - \theta_k)^2$，下面讲述类 si 的具体实现思路。

在 SI 中, 参数 $\Omega_k^u = \sum_{v<u} \frac{\omega_k^u}{(\Delta_k^v)^2+\xi}$，只在一个任务训练完成之后才进行更新，$\omega_k^u$ 在训练一个任务的过程中进行更新，其初始值为 0，其等于训练一个任务过程中，损失函数对第 k 个参数的梯度和第 k 个参数的更新量的乘积再取负的累加。

为了实现 SI，为类 SI 实现了三个函数

- compute_importance
- update_W
- compute_penalty

compute_importance

该函数计算 Ω_k^u，在这个函数中，根据 $\Omega_k^u = \sum_{v<u} \frac{\omega_k^v}{(\Delta_k^v)^2+\xi}$ 计算第 k 个参数的重要性权重，其中 Δ_k^v 表示第 v 个任务得到的第 k 的参数的值与第 v-1 个任务得到的第 k 个参数值的差。由于是在线计算 Ω_k^u，因此仅仅需要计算 $\frac{\omega_k^{u-1}}{(\Delta_k^{u-1})^2+\xi}$，然后加到旧的 Ω_k^u 进行更新 Ω_k^u。

compute_penalty

该函数计算改进后的损失函数中的惩罚项，根据公式 $\sum_k \Omega_k^u(\hat{\theta}_k - \theta_k)^2$ 计算损失函数中的惩罚项，Ω_k^u 为第 k 个参数的重要性，$\hat{\theta}_k$ 是前一个任务训练结束之后，第 k 个参数的取值，θ_k 是第 k 个参数的当前值。

update_W

该函数对 ω_k^u 进行更新，对 ω_k^u 进行更新，将参数更新量和损失函数对参数的梯度的乘积的负值加到旧的 ω_k^u 上，得到新的 ω_k^u。

在训练任务 u 时, 每一次训练, 先计算原始的损失函数, 然后通过 SI 类的 compute_penalty 计算惩罚项，最终的损失函数为原始的损失函数 + 惩罚项，之后对模型中的参数进行反向传播更新，每一次训练还需要调用 SI 的 update_W 函数对 ω_k^u 进行更新，在一个任务训练完成之后，需要利用 compute_importance 更新 Ω_k^u。

5. 模型训练

首先使用 Dataloader 装载 4 个训练集的数据和 label，设置 batchsize 并启用 shuffle。因为是分类问题，所以选择交叉熵作为损失函数，并且选择收敛较快的 Adam 作为优化函数。模型单个 epoch 的训练过程如下：

1. 将一个 batch 的数据和 label 装载到 GPU 显存中
2. 用当前模型输出结果
3. 利用交叉熵损失函数计算 loss
4. 利用参数重构的方法根据模型中参数的重要性计算出 lll_loss(在 baseline 中 lll_loss=0)
5. 计算 total_loss=loss+λ*lll_loss
6. 根据 total_loss 反向传播，优化模型的参数

在每个不同的训练集上训练结束后，都要使用参数重构的方法重新计算当前模型每个参数重要程度的矩阵。相关超参数如表3所示。

batch size	256
learning rate	0.001
epochs	35

表 3: 模型训练超参数

6. 模型评估

设置每个 epoch 计算一次模型在验证集的准确率。首先将模型调成 eval 模式，然后用模型预测出在每个数据的 label，分别计算模型在每个验证集的准确率。最后再计算模型在 4 个验证集的平均准确率，观察实验结果。

四、实验结果与分析

结果来源说明: 对于单个 TextCNN 模型和单个 RNN 模型，分别让其在四个测试集上运行五次，得到实验结果。对于 Baseline、EWC、MAS、SI，训练模型顺序：comp->rec->sci->talk，然后在四个测试集上依次测试，每个模型各执行五次得到实验结果。

1. comp_test

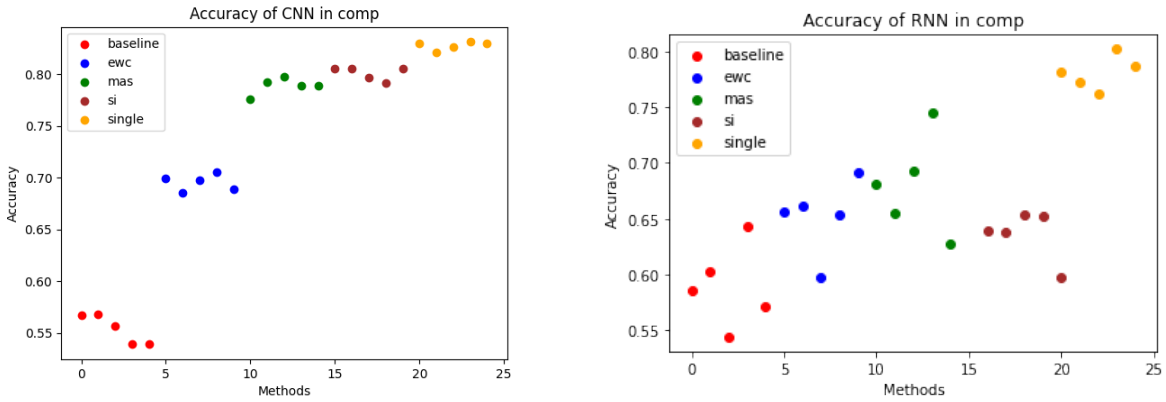

图 10: 所有范式在 comp 任务的结果

TextCNN：

单个 CNN 在 comp 上的正确率能达到 82% 左右，但是 baseline 的正确率仅为 55% 左右，在没有使用任何优化方法的情况下，TextCNN 模型对于第一个任务的遗忘是非常严重的，正确率下降了 27%。

- EWC

  使用 EWC 方法应对灾难性遗忘问题，效果显著，在 comp 测试集上平均正确率在 70% 左右，使用 EWC 方法，使得 TextCNN 在第一个任务上的正确率提升了大约 15%。

- MAS

  利用 MAS 方法，TextCNN 对于第一个任务的记忆几乎没有遗忘，与 baseline 相比，正确率提升了 24%，与单个 TextCNN 在 comp 上的正确率仅相差 3%

- SI

  经过四个任务的学习之后，利用 SI 的 TextCNN 模型仍然对第一个任务学习到的知识记忆犹新，平均正确率在 80%，与 baseline 相比提升了大约 25%，与单个 CNN 模型相比，仅仅差 1%-2%。

由上面所述，对于 TextCNN 模型，三种参数重构方法均对灾难性遗忘问题有了一定的缓解，效果：SI>MAS>EWC。

RNN：

单个 RNN 在 comp 测试集上的平均正确率在 78% 左右，而没有使用任何抗遗忘方法的 baseline，其正确率大概在 58%，由于灾难性遗忘问题的存在，使得 RNN 模型对于已经学习到的知识有了一定的遗忘，经过学习四个任务之后，其在第一个任务的正确率下降了 20%。

- EWC

  使用 EWC 方法，RNN 在对于第一个任务的遗忘有了一定的缓解，相比于 baseline，其正确率大约为 66%，提升了大约 9%，但是距离单个 RNN 模型的正确率还有一定的差距。

- MAS

  利用 MAS 方法，在 comp 测试集上的平均正确率大约为 69%，与 baseline 相比，其平均正确率有了较大的提升，提升了大约 11%。

- SI

  SI 方法下，RNN 模型在第一个任务上的正确率为 64%，与 baseline 方法相比其正确率提升了 6%。

由上面所述，对于 RNN 模型，三种参数重构方法均对灾难性遗忘问题有了一定的缓解，效果：MAS>EWC>SI。

TextCNN 和 RNN 在 comp 上的比较:

对于单个模型，TextCNN 的效果要比 RNN 的效果好，正确率大约高 4%。而 RNN 对于第一个任务的遗忘没有 TextCNN 对于第一个任务的遗忘严重，相比于单个模型，RNN 在 baseline 上的正确率下降 20%，而 TextCNN 下降了 27%。但是从图中也可以看到，对于 TextCNN，其图中的相同颜色点大致在一条直线上，而对于 RNN 来说，点分布的比较散，不同的执行得到的结果相差较大，这说明了 TextCNN 的稳定性要比 RNN 的稳定性好。

2. rec_test

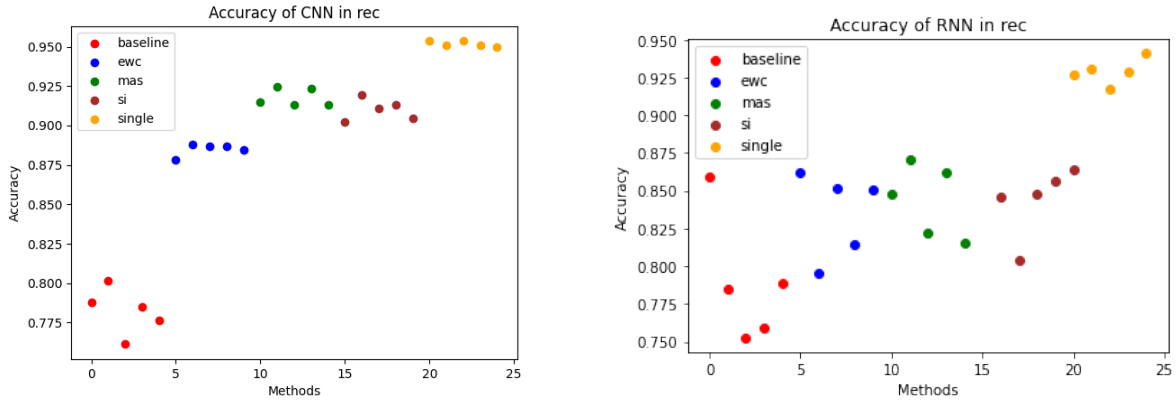

图 11: 所有范式在 rec 任务的结果

TextCNN:

单个 TextCNN 模型在 rec 任务上的学习效果较好，五次执行，其在测试集上的正确率均稳定在 95% 左右，说明 rec 任务比较简单。没有使用任何抗遗忘方法的 baseline，五次运行的平均正确率也在 79%，正确率下降了 16%，遗忘显著。

- EWC

利用 EWC 方法，TextCNN 模型对于 rec 任务的遗忘有了一定的缓解，使用 EWC 方法之后，TextCNN 在 rec 任务上的正确率在 88% 左右，和 baseline 相比有了 9% 的提升，但与单个 TextCNN 模型相比，还差了 7%。

- MAS

利用 MAS 方法之后，TextCNN 在 rec 任务上的平均正确率大约在 92% 左右，几乎完全解决了 TextCNN 对于 rec 任务的遗忘问题，正确率比 baseline 提升了 13%。

- SI

利用 SI 方法，TextCNN 在 rec 任务上的平均正确率大约在 91% 左右，抗遗忘效果良好，正确率比 baseline 提升了大约 12%。

在 rec 任务上的抗遗忘效果：MAS>SI>EWC；

RNN：

单个 RNN 模型在 rec 任务上的正确率为 93% 左右，baseline 下，其正确率大概在 78%，由于灾难性遗忘，其正确率下降了 15%。

- EWC

利用 EWC 方法，RNN 模型对于 rec 任务的遗忘有了一定的缓解，使用 EWC 方法之后，RNN 在 rec 任务上的正确率在 83% 左右，和 baseline 相比有了 5% 的提升，但与单个 RNN 模型相比，还差了 10%。

- MAS

利用 MAS 方法之后，RNN 在 rec 任务上的平均正确率大约在 84.4% 左右，正确率比 baseline 提升了 6.4%。

- SI

利用 SI 方法，RNN 在 rec 任务上的平均正确率大约在 84% 左右，正确率比 baseline 提升了大约 6%。

在 rec 任务上的抗遗忘效果：MAS>SI>EWC；

TextCNN 和 RNN 在 rec 上的比较:

- 正确率

  在 baseline 和单个模型下，TextCNN 模型在 rec 测试集上的正确率与 RNN 模型的正确率相比，略高 1%-2%。在三种参数重构方法下，TextCNN 模型的正确率比 RNN 模型略好。

- 抗遗忘效果

  三种参数重构方法下，对 TextCNN 的提升效果分别为 9%、13%、12%。对 RNN 模型的提升效果为 5%、6.4%、6%，可以看到三种参数重构方法，TextCNN 模型的提升效果要好于 RNN 模型。

- 稳定性

  由图中得知，在不同的执行下，TextCNN 模型的准确率变化不大，变化范围 2% 上下，RNN 模型的准确率变化范围大概为 5%-7%。

三种参数重构方法的比较：

在 TextCNN 模型下，三种参数重构方法的抗遗忘效果为 MAS 约等于 SI>EWC；在 RNN 模型下，三种参数重构方法的抗遗忘效果为 EWC 约等于 MAS 约等于 SI。

3. sci_test

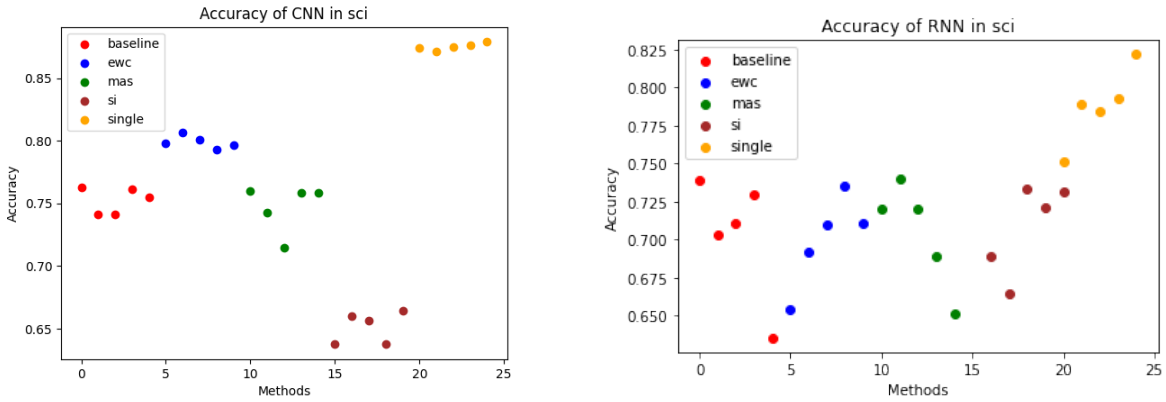

图 12: 所有范式在 sci 任务的结果

TextCNN：

单个 TextCNN 模型在 sci 测试集上的正确率大概在 87%，baseline 下，TextCNN 模型在 SCI 测试集上的正确率大概在 75%，由于灾难性遗忘，TextCNN 模型在 SCI 任务上的正确率下降了 12%。

- EWC

使用了 EWC 方法，TextCNN 模型在 SCI 测试集上的正确率大概在 80%，与 baseline 相比，正确率提升了 5%。

- MAS

使用 MAS 方法，TextCNN 模型在 SCI 测试集上的正确率大概在 74%，与 baseline 相比，正确率下降了 1%，这是因为为了避免遗忘别的任务，在学习 SCI 任务时，参数没有太大调整，对 sci 的学习效果不好。

- SI

使用 SI 方法，TextCNN 模型在 sci 测试集上的正确率大概在 65%，和 baseline 相比，在 sci 测试集上的正确率下降了 10%，原因也和 MAS 中的原因类似。

在 SCI 任务上的抗遗忘效果：EWC>MAS(遗忘程度更加剧烈)>SI(遗忘程度更加剧烈)。

RNN：

单个 RNN 模型在 SCI 测试集上的正确率大概在 78%，baseline 下，RNN 模型在 SCI 测试集上的正确率大概在 70%，正确率下降了 8%。

- EWC

使用 EWC 方法，RNN 模型的正确率在 70% 左右，平均五次运行结果来看，EWC 在缓解 RNN 模型对于 SCI 任务的遗忘上几乎没有效果。

- MAS

使用 MAS 方法，RNN 模型的正确率在 70% 左右，平均五次运行结果来看，EWC 在缓解 RNN 模型对于 SCI 任务的遗忘上几乎没有效果。

- SI

使用 SI 方法，RNN 模型的正确率在 70% 左右，平均五次运行结果来看，EWC 在缓解 RNN 模型对于 SCI 任务的遗忘上几乎没有效果。

在 rec 任务上的抗遗忘效果：EWC、MAS、SI 对于缓解 RNN 对 rec 任务的遗忘上，作用不大。

TextCNN 和 RNN 在 sci 上的比较:

- 正确率/抗遗忘效果

  在 sci 学习任务上，EWC、MAS、SI 三种参数重构方法，无论在 TextCNN 模型还是在 RNN 模型上，缓解灾难性遗忘的作用几乎没有，在 TextCNN 模型上甚至出现了遗忘程度更加剧烈的反常现象。

- 稳定性

  稳定性方面，由图中得知，TextCNN 模型的稳定性要优于 RNN 模型的稳定性。

由上图可以看到，在 sci 测试集上，EWC、MAS、SI 相比于 baseline 来说，抗遗忘效果不是很好，这可能是因为，为了避免遗忘以前的任务，这些参数重构方法选择对一些参数不进行太大的更新，所以导致对当前任务学习效果不好，但是总的来说，四个任务下的总正确率还是在提升的。

4. talk_test

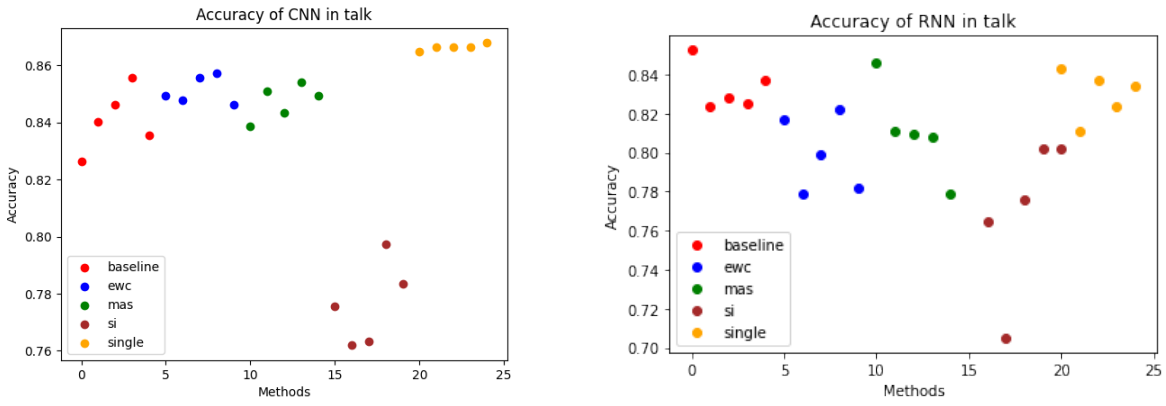

图 13: 所有范式在 talk 任务的结果

TextCNN：

在 talk 任务上，单个 TextCNN 模型的正确率在 86% 左右，baseline 下 TextCNN 模型的正确率在 84%，由于任务的学习顺序为 comp->rec->sci->talk，因此在第四个任务上，TextCNN 模型几乎没有遗忘。

- EWC

使用 EWC 方法，TextCNN 模型在 talk 任务上的正确率在 85% 左右，由于遗忘问题不严重，抗遗忘的效果也没有显示出来。

- MAS

使用 MAS 方法，TextCNN 模型在 talk 任务上的正确率在 85% 左右，同样，由于遗忘问题不严重，抗遗忘的效果也没有显示出来。

- SI

使用 SI 方法，TextCNN 模型在 talk 任务上的正确率在 78% 左右，存在遗忘的问题。

RNN：

单个 RNN 模型在 talk 任务上的正确率大概在 83%，baseline 下在 talk 任务上的平均正确率大概在 83%，几乎没有遗忘。

- EWC

使用 EWC 方法，RNN 模型在 talk 任务上的正确率大概在 80%，和 baseline 相比正确率略微下降。

- MAS

使用 MAS 方法，RNN 模型在 talk 任务上的正确率大概在 81%，和 baseline 相比正确率下降 2%。

- SI

使用 SI 方法，RNN 模型在 talk 任务上的正确率大概在 77%，和 baseline 相比正确率下降了 6%。

TextCNN 和 RNN 在 talk 上的比较：

- 正确率

  单个模型以及 baseline 范式、SI 范式下，TextCNN 和 RNN 模型在 talk 任务上的正确率大致相同，在 EWC、MAS 范式下，TextCNN 模型要比 RNN 模型的正确率高 5% 左右。

- 抗遗忘效果

  对于 TextCNN、RNN 模型，在 talk 任务上，EWC、MAS 范式的抗遗忘效果几乎没有体现，而且在 SI 范式下，TextCNN、RNN 模型在 talk 任务上的准确率还有一定下降。

- 稳定性

  由图中得知，以 talk 测试集上的正确率为指标，总的来看，RNN 模型的稳定性没有 TextCNN

模型的稳定性好。观察图中相同颜色点的分布，对于 RNN 模型来说，五次运行得到的正确率分布比较分散，而 TextCNN 模型五次运行得到的正确率分布比较集中，因此 TextCNN 的稳定性优于 RNN 模型稳定性。

talk 任务上遗忘问题的分析:

由实验结果得知，对一些参数重构方法，在 talk 任务上的正确率反而不及 baseline 效果，原因可能如下:

1. baseline 刚刚学习过第四个任务，因此在第四个测试集上的正确率很高
2. 参数重构方法，要记忆前三个任务，因此需要做一些取舍，为了前面任务的正确率，第四个任务的正确率略微下降，但是保证了四个任务总的正确率在上升。

5. average

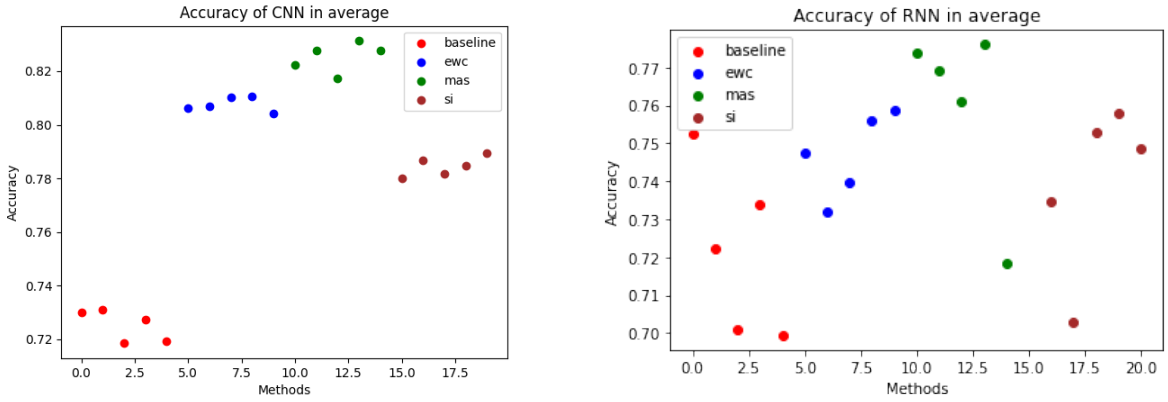

图 14: 所有范式的平均正确率结果

TextCNN:

baseline 下，TextCNN 模型在四个测试集上的平均正确率大概在 73% 左右。

- EWC

利用 EWC 方法，TextCNN 模型在四个测试集上的平均正确率大概在 81% 左右，和 baseline 相比，平均正确率大约提高 8%。

- MAS

利用 MAS 方法，TextCNN 模型在四个测试集上的平均正确率大概在 82% 左右，和 baseline 相比，平均正确率大约提高 9%。

- SI

利用 SI 方法，TextCNN 模型在四个测试集上的平均正确率大概在 79% 左右，和 baseline 相比，平均正确率大约提高 6%。

在 TextCNN 模型下，三种参数重构方法对于缓解灾难性遗忘的效果为：MAS>EWC>SI。

RNN：

baseline 下，RNN 模型在四个测试集上的平均正确率在 71% 左右。

- EWC

利用 EWC 方法，RNN 模型在四个测试集上的平均正确率在 75% 左右，和 baseline 相比，其正确率提高了大约 4%。

- MAS

利用 MAS 方法，RNN 模型在四个测试集上的平均正确率在 76% 左右，和 baseline 相比，其正确率提高了大约 5%。

- SI

利用 SI 方法，RNN 模型在四个测试集上的平均正确率在 74% 左右，和 baseline 相比，其正确率提高了大约 3%。

在 RNN 模型下，三种参数重构方法对于缓解灾难性遗忘的效果为：MAS>EWC>SI。

TextCNN 和 RNN 模型的比较：

- 正确率

  根据上面的分析，可知在正确率方面，三种参数重构方法，TextCNN 的正确率要比 RNN 模型的正确率略好。

- 抗遗忘效果

  三种参数重构方法作用于 TextCNN 和 RNN 模型，抗遗忘的效果，TextCNN 略好于 RNN 模型。

- 稳定性

  根据图示分析，TextCNN 的五次运行的结果，在相同的方法下，准确率的变化不大，相同的颜色点大致在一条直线上；RNN 的五次运行结果，相同方法不同的运行，准确率有时相差大概 5%，因此在稳定性方面，TextCNN 略好于 RNN。

三种参数重构方法比较：

最终的实验结果表明，EWC、MAS、SI 三种参数重构方法在抗遗忘问题上取得了不错的成绩。在 TextCNN 模型上，与 baseline 相比，EWC、MAS、SI 分别让四个测试集上的平均正确率提升 8%、9%、6%；平均正确率分别达到 81%、82%、79%。在 RNN 模型上，与 baseline 相比，EWC、MAS、SI 分别让四个测试集上的平均正确率提升 4%、5%、3%；平均正确率分别达到 75%、76%、74%。

五、总结

人工智能的期末项目终于结束了，这次实验的内容在 AI 中也比较新，无论是方法和编程量上都有一定的难度。

由于我们之前有学过 Pytorch 相关的内容，所以 CNN 和 RNN 模型的搭建不是特别困难，但是多任务学习的知识对我们来说还是十分陌生的。好在我们之前也看过李宏毅老师机器学习的相关课程，所以早早确定了使用参数重构的几种方法。当助教在课上也给我们观看李宏毅老师的视频时，我们顿时懂了，参数重构的方法确实特别适合这次的实验，而最后实验得到的结果也正如我们所愿。

总而言之，这次实验让我们学到了很多新的东西，受益匪浅。《人工智能》的课程即将结束，感谢老师和助教一学期以来的教诲与指导，希望在未来的日子里我们能有机会继续在 AI 的领域遨游。

六、分工

施天予：

- 文本处理
- 循环神经网络
- EWC 和 MAS

孙奥远：

- 卷积神经网络
- SI
- 实验结果分析

实验报告的对应部分由每个人自己完成

参考文献

[1] 李宏毅机器学习，https://speech.ee.ntu.edu.tw/~hylee/ml/2021-spring.html, 2021.

[2] James Kirkpatrick and Razvan Pascanu and Neil Rabinowitz and Joel Veness and Guillaume Desjardins and Andrei A. Rusu and Kieran Milan and John Quan and Tiago Ramalho and Agnieszka Grabska-Barwinska and Demis Hassabis and Claudia Clopath and Dharshan Kumaran and Raia Hadsell. *Overcoming catastrophic forgetting in neural networks*, CoRR, 2017.

[3] Rahaf Aljundi and Francesca Babiloni and Mohamed Elhoseiny and Marcus Rohrbach and Tinne Tuytelaars. *Memory Aware Synapses: Learning what (not) to forget*, ECCV, 2018.

[4] Friedemann Zenke and Ben Poole and Surya Ganguli, *Continual Learning Through Synaptic Intelligence*, ICML, 2017.