京东-贪心 NLP 项目实验手册

项目 2：基于京东电商的营销文本生成

项目设计和编写: 林培鑫 姜冰钰

后期编辑: 李文哲

单 位: 贪心科技

2020 年 7 月 17 日

目 录

1 项目描述与目标

文本生成（Text Generation）具体可以细分成文本摘要、机器翻译、故事续写等几种任务。本项目主要用到的是**文本摘要（Summarization）**的技术。文本摘要技术通常分为两大类，**抽取式（Extractive）**方法和**生成式**方法。抽取式摘要指的是对于原文（称之为 source），我们生成摘要的方式是选取其中关键的句子摘抄下来。相反，生成式摘要则是希望通过学习原文的语义信息后相应地生成一段较短但是能反应其核心思想的文本作为摘要。生成式摘要相较于抽取式摘要更加灵活，但也更加难以实现。本项目我们将会先用生成式摘要的方法构建一个 Seq2seq+Attention[3] 的模型作为 baseline，然后构建一个结合了生成式和抽取式两种方法的 Pointer-Generator Network 模型 [6]。

文本生成任务中，我们作为输入的原文称之为 **source**，待生成的目标文本称之为 **target** 或者 **hypothesis**，用来作为 target 好坏的参考文本称之为 **reference**。在本项目的数据源来自于京东电商的发现好货栏目，source 主要由三部分构成：1 是商品的标题，2 是商品的参数，3 是商品宣传图片里提取出来的宣传文案。

通过本项目的练习，你能掌握文本生成的主要流程：

- **文本预处理与特征提取**：预处理与文本分类的做法差不多，而深度学习中特征提取基本都是通过 embedding 来做。
- **模型搭建**：近些年深度学习的模型迭代很快，很多模型提出后还未被工业界广泛应用，因此不像经典的机器学习模型如逻辑回归、随机森林等有成熟的实现方案。很多时候我们需要自己去阅读论文之后，根据论文以及作者提供的代码去复现（reproduce）一个模型，并且结合到我们的业务需求和数据进行调整。我们将详细解释如何构建一个深度学习模型的各个模块，包括实现前向传导、定义损失函数等。模型的搭建将是本项目最重要的一个部分。
- **Debug 和调参**：深度学习模型的训练时间一般较长（少则几个小时，多则好几天）。复杂的网络结构下 debug 的难度较大，超参数的选择

如果使用网格搜索也比较费时。我们可以对训练过程损失的变化、权重的分布变化进行记录，然后通过可视化工具来分析模型训练的过程是否发生过拟合、哪一层的权重更新缓慢，从而进行 debug、选择超参数。

- **模型输出的控制**：文本生成任务，往往需要对 Decoder 的输出进行一定的控制（inference），最常用的方法是 Beam Search。项目中将详解如何实现 Beam Search，并围绕 Beam Search 讲解如何让对文本生成模型的输出进行控制。
- **模型的评估**：使用 Rouge 对模型的效果进行评估，同时了解如何使用 Python 的装饰器来评估模型预测的用时。

同时，通过本项目

- 1. 熟练掌握如何使用 Pytorch 框架搭建复杂神经网络结构。
- 2. 熟练掌握深度学习中 Tensor 的变换操作和矩阵的运算。
- 3. 熟练掌握（Seq2seq、Attention、LSTM、PGN 等模型）。
- 4. 熟练掌握如何训练神经网络（调参、debug、可视化）。
- 5. 熟练掌握如何实现 Beam Search 算法来生成文本。
- 6. 熟练掌握文本生成任务的评估方法。
- 7. 掌握复现论文的技巧。
- 8. 掌握深度学习训练的一些优化技巧（Scheduled sampling[1]、Weight tying[4] 等）。
- 9. 了解如何使用多种文本增强技术处理少样本问题。
- 10. 了解深度学习中的常用优化方法的原理及区别。

作为第二个项目，我们的目标是帮助大家完成一个文本生成的任务，通过这个任务让大家熟悉 NLP 常用的模型架构，培养深度学习模型搭建必要的编码能力，并且培养作为一个算法工程师所必备的论文复现能力。完成这一阶段的学习，相信大家将有能力自行去深入探索文本生成的技术以及更多的 NLP 相关领域。

2　项目数据描述

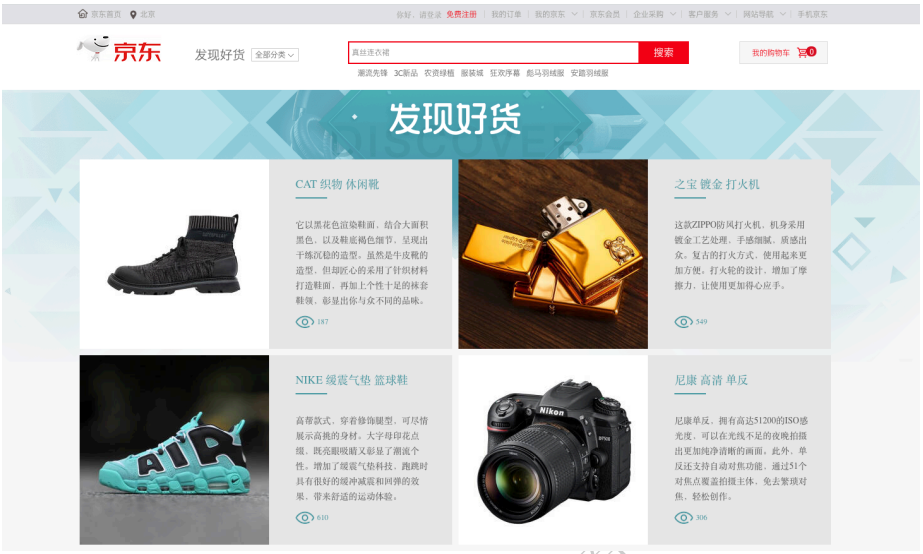

图 1: 京东发现好货主页[1]

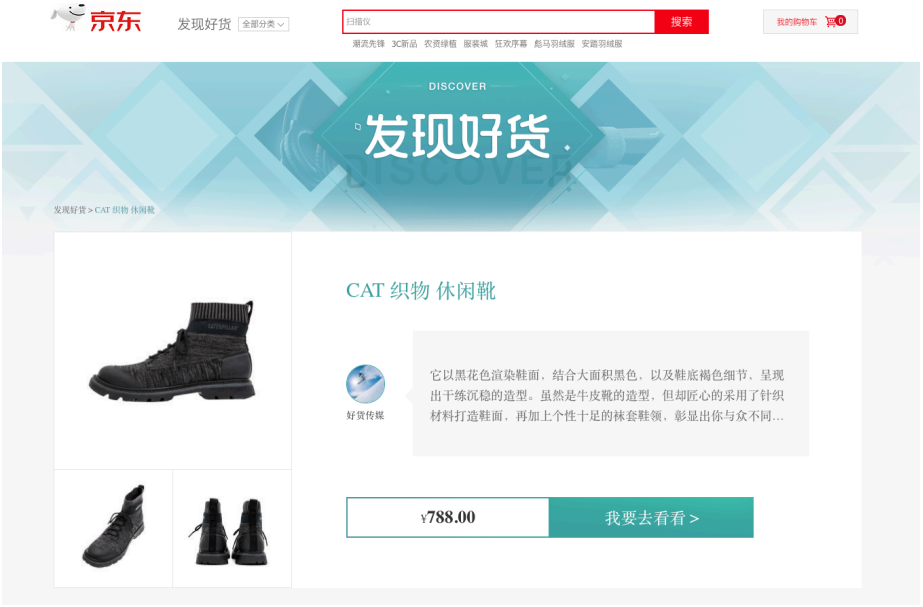

图 2: 京东发现好货详情页

在本项目中，我们使用的是京东商品的数据以及营销文案。在发现好货栏目

里，每一件推荐的商品都会有一段专业写手撰写的营销文案，这段文案将作为我们的 reference。如图 1、 2所示。

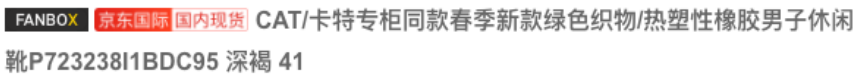

图 3: 商品标题

图 4: 商品参数

图 5: 商品营销图片文案

作为生成依据的 source，主要由三部分数据组成，分别为商品的标题（如图 3所示），商品的属性（如图 4所示）以及营销图片的文案（如图 5所示）。

本项目使用的数据集中，原始的样本包括约 5 万个文本对（source-reference pairs），经过文本增强后，可以成倍扩展。

3 项目事宜

本项目相对于上一个项目，将更加注重对大家动手能力的提升，让大家完整的去了解一个 NLP 项目的整体框架，并且实现各个环节的重要细节。

3.1 项目的整个框架

整个项目框架如图 6所示。下面对于图中每个模块做简要的描述，具体的细节请参考本文章后续的内容。

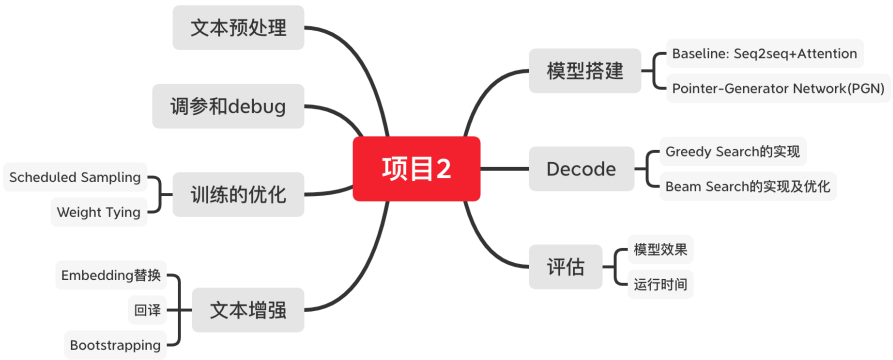

图 6: 京东营销文本生成项目模块架构图

- **文本预处理**：这一块没有太多的工作量，代码将直接提供给大家，大家可以根据自己的想法进行调整优化。
- **模型搭建**：从这个项目开始，需要大家具体的来实现深度学习的模型。我们将使用 Pytorch 框架，先构建一个经典的 Seq2seq+Attention 模型，然后在此基础上构建一个 Pointer-Generator Network(PGN) 模型。

- **Decode**：Decode 部分将先实现最 naive 的 Greedy Search，了解如何将模型的输出转化为最终的文本；然后实现在文本生成任务中很常用的 Beam Search 算法，并加上一些相关的优化。

- **评估**：文本生成任务常用的评估指标有 BLEU 和 ROUGE 等，摘要生成一般用 ROUGE 进行评估，我们将构建一个评估的模块来衡量模型效果的好坏；同时，由于我们的模型基于 Seq2seq 结构，decode 阶段是一个一个生成 token，往往 decode 阶段的速度会是性能的瓶颈，我们需要对我们实现的预测模块的运行速度进行评估，具体会介绍如何使用 Python 装饰器写一个通用的计时器。

- **调参和 debug**：训练深度学习模型往往需要对超参数进行多次调整才能训练出一个相对理想的模型，同时由于我们要自己实现模型的细节，可能有些实现错误导致模型运行不如预期，我们可能需要进行多次 debug。这里我们会介绍一些可视化的方法帮助大家了解训练过程中模型内部的变化。

- **训练的优化**：Seq2seq 模型的训练往往会出现”Exposure bias”[5] 的问题，即训练阶段与预测阶段的输入数据分布差别过大，导致模型在预测阶段的表现不如训练阶段。我们将介绍 Scheduled sampling 来应对这一问题；除此之外，还将介绍权重共享（Weight tying）等一些训练技巧来帮助模型更快的收敛。

- **文本增强**：缺乏高质量的标注数据是 NLP 任务中非常常见的情况，相对于英文，中文数据更是匮乏。我们将带大家了解一些常用的文本增强技术，并实现其中几种，来扩充我们的样本数量。

4 项目作业描述

本次项目依旧分成三次作业给大家进行练习。第一次作业将带大家构建一个 Seq2seq+Attention 的经典模型结构作为我们的 baseline；第二次作业将让大家在第一次作业的基础上，实现一个 Pointer-Generator Network；第三次作业将让大家实现一些在课上学到的优化技术（训练优化、Beam Search 优化、数据增强等）。

4.1 第一次作业

任何机器学习任务都需要构建一个 baseline 作为后续优化效果的参考依据，第一个作业需要大家构建本项目的 Baseline——经典的 Seq2seq+Attention 模型。同时搭建一个文本生成任务的各个基本模块。第一次作业的框架图如图7所示。

图 7: 第一次作业架构图

4.2 第一次作业描述

对于本次任务，需要完成如下的部分：

- 第一：构建 Seq2seq+Attention 的模型
- 第二：实现训练模块。
- 第三：实现预测模块，包括 Greedy Search 和 Beam Search。
- 第四：实现评估模块。

4.2.1 第一次作业详解

Seq2seq+Attention

Baseline 的架构遵循论文 [6] 中的定义（如图8所示）：Encoder 是 Embedding 层之后接 BiLSTM

- Encoder: Embedding + BiLSTM + Softmax

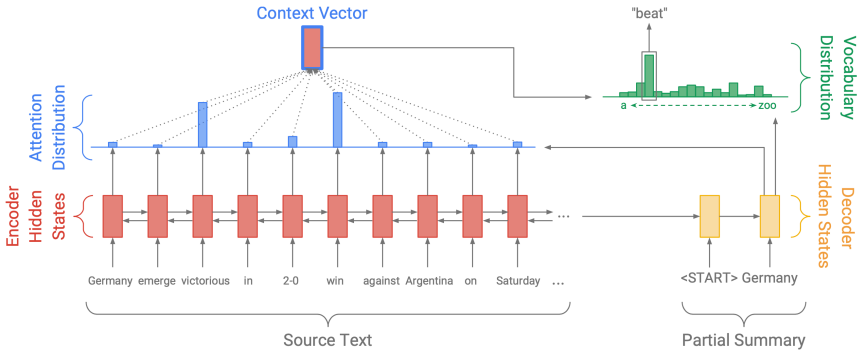

图 8: Seq2seq+Attention 架构图

- Attention: Feed Forward + Softmax (具体公式见图9，即论文中的公式 (1), (2))，之后使用计算到的 attention 权重和 encoder 输出的隐状态计算 context vector（详见论文中的公式 (3))。
- Decoder: Embedding + LSTM + Feed Forward + Softmax（详见论文中的公式 (4))。

搞清楚结构之后，我们就可以实现整个模型的前向传导部分。

$$e_i^t = v^T \tanh(W_h h_i + W_s s_t + b_{\text{attn}}) \quad (1)$$

$$a^t = \text{softmax}(e^t) \quad (2)$$

图 9: Attention 计算方式

实现训练模块

实现一个 Seq2seq 模型的训练模块，我们主要需要做以下几步：

- 初始化模型或加载现有模型。
- 定义一个 DataLoader 类来处理训练样本，实现包括字典的构建、token 的索引以及 batch 的划分等功能。

- 定义损失函数。Baseline 模型的损失函数定义为负对数概率（Negative Log Likelihood），具体公式见论文中的公式 (6), (7)。
- 定义优化方法。本模型中使用 Adagrad 作为优化算法，定义优化算法时需要指定学习率等参数，建议参数会在代码中给出，大家可以根据自己的实际训练情况进行调整。
- 计算整体平均损失。具体步骤为，计算 step loss（当前 time step 的损失），然后计算 sample loss（当前样本的平均损失），然后计算 bacth loss（当前 batch 的平均损失），最后计算 epoch loss（当前 epoch 的平均损失）。

Pytorch 模块

这里给大家介绍这个作业主要用到的 Pytorch 模块，建议大家在具体使用哪一个模块时详细查阅Pytorch 官方文档中该模块的用法。

- **torch.nn.Module**：用以定义网络各层以及前向传导逻辑的模块。我们将整个 Seq2Seq 模型定义为一个 Module，模型下的各部分（Encoder、Decoder、Attention）也都分别定义为一个 Module，并作为 Seq2seq 的属性。
- **torch.nn.Linear**：可以用于定义一个前馈层，初始化参数需指定输入和输出的维度，可以指定是否需要 bias 项，会自动生成可更新的参数，即这一层的权重。
- **torch.nn.LSTM**：用于定义 LSTM 层，可以通过参数选择单向或者双向。
- **torch.Tensor**：神经网络中一切计算的基础，使数据能够应用到 GPU 计算中。我们的数据全部都要以 tensor 的形式传入。学习如何对 Tensor 操作是学习搭建神经网络的一个重要部分。有一个注意事项是，非必要情况下尽量不要使用 *+=*、**=* 这样的 *in-place* 操作符，否则可能会报错。最好将 *tensor1 += tensor2* 写成 *tensor1 = tensor1 + tensor2*。
- **torch.squeeze**：删掉 Tensor 中大小为 1 的维度，作用类似于 numpy 中的 flatten()。

- **torch.unsqueeze**：与 squeeze 进行相反的操作，在参数指定的维度添加一个大小为 1 的维度。
- **torch.bmm**：对同一个 batch 里的两个矩阵求 dot product。
- **torch.gather**：根据指定的 dimension 以及 index，从一个 tensor 里取出相应位置的值。

预测部分的实现

由于 Greedy Search，较为简单，就不多讲解，大家跟着代码里的提示实现即可。这一部分重点讲解一下 Beam Search 的实现方法。

- 首先定义一个 **Beam** 类，作为一个存放候选序列的容器，属性需维护当前序列中的 token 以及对应的对数概率，同时还需维护跟当前 time step 的 Decoder 相关的一些变量。此外，还需要给 Beam 类实现两个函数：一个 extend 函数用以扩展当前的序列（即添加新的 time step 的 token 及相关变量）；一个 score 函数用来计算当前序列的分数（在后面的作业中，将主要围绕这个函数对 Beam Search 进行优化）。
- 接着我们需要实现一个 **best_k** 函数，作用是将一个 Beam 容器中当前 time step 的变量传入 Decoder 中，计算出新一轮的词表概率分布，并从中选出概率最大的 k 个 token 来扩展当前序列（在此过程中我们还会过滤掉一些不允许出现的 token，例如 <UNK>, <SOS> 等），得到 k 个新的候选序列。
- 最后我们实现主函数 **beam_search**，作用一是控制最大的 decode step；二是在每个 step 对候选序列进行排序，选出 top k 的候选序列，并对剩下的进行剪枝；三是对最终的 k 个候选结果进行排序取分数最高的一个作为最终结果。

评估部分的实现

- 模型效果的评估我们使用ROUGE来实现，
- 模型预测时间用 Python 的 time 模块来实现，为了对多个函数进行计时，一般我们可以定义一个装饰器来实现这一功能。

4.3 第二次作业

第二个任务的主要工作是在第一次作业的基础上，实现 [6] 论文中提出的 PGN 模型以及 coverage 机制，同时加上一些优化方法提高我们的模型效果。第二次作业的框架如图 10所示。

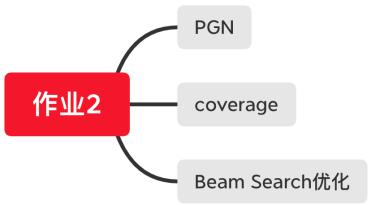

图 10: 第二次作业思维导图

4.3.1 第二次作业描述

对于本次任务，需要完成如下的思维导图部分。

- **第一**:将我们的 baseline 模型改造成一个 Pointer-Generator Network(PGN)。
- **第二**：加上 coverage 机制。
- **第三**：实现对 Beam Search 的优化以达到对输出更好的控制。

4.3.2 第二次作详解

实现 PGN 模型

对比图 11的 PGN 架构和我们作业 1 实现的图 8中的 baseline 模型架构，可以发现主要的变化在于多了一个控制摘抄和生成的 P_gen 以及一个将 Pointer 和 Generator 的输出进行合并计算的 final distribution。这里重点讲一下这两个部分的实现：

- **P_gen**：P_gen 的概念其实很好理解，通过一个 sigmoid 函数的来计算一个阈值以决定给 pointer 和 generator 的输出各自分配多大的权重，有点类似于 LSTM 或者 GRU 中的门机制。这个门的输入是

context vector 、Decoder 当前 time step 的隐状态和输入（详见论文中的公式 (8), (9)）。

- **Final distribution**：所谓的 pointer 本质是根据 attention 的分布（source 中每个 token 的概率分布）来挑选输出的词，是从 source 中挑选最佳的 token 作为输出；所谓的 generator 的本质是根据 decoder 计算得到的字典概率分布 P_vocab 来挑选输出的词，是从字典中挑选最佳的 token 作为输出。所以大家应该能发现：Attention 的分布和 P_vocab 的分布的长度和对应位置代表的 token 是不一样的，所以在计算 final distribution 的时候应该如何对应上呢?

  这里推荐的方式是，先对 P_vocab 进行扩展，将 source 中的 oov 添加到 P_vocab 的尾部，得到 P_vocab_extend 这样 attention weights 中的每一个 token 都能在 P_vocab_extend 中找到对应的位置，然后将对应的 attention weights 叠加到扩展后的 P_vocab_extend 中的对应位置，得到 final distribution。

  为了做到将 attention weights 这个 tensor 中的值添加到 P_vocab_extend 中对应的位置，你需要使用到 **torch.Tensor.scatter_add** 这个函数，P_vocab_extend 作为添加值的目标 tensor，attention_weights 作为添加值的来源 tensor，index 化后的 source 可以作为 attention_weights 的添加依据。详细用法请查看文档。

实现 Coverage 机制

需要传递一个新的变量 **coverage_vector**，并且每个 time step 更新一次，更新方式详见论文中的公式 (10)。coverage_vector 主要用在 Attention 模块的计算中（详见论文公式 (11)）和 train 模块的 loss 计算中（详见论文公式 (12), (13)）。

此外，由于论文作者提到，训练该模型最好的方式是先不加 coverage 机制训练一个收敛的模型，然后再加上 coverage 机制对该模型进行 fine-tuning，所以需要实现以下几点：

- **实现模型的二次训练**：具体查看 Pytorch 文档中如何保存模型和读取模型）。

- **实现 coverage 机制的开关**：具体的做法是定义一个全局布尔型变量

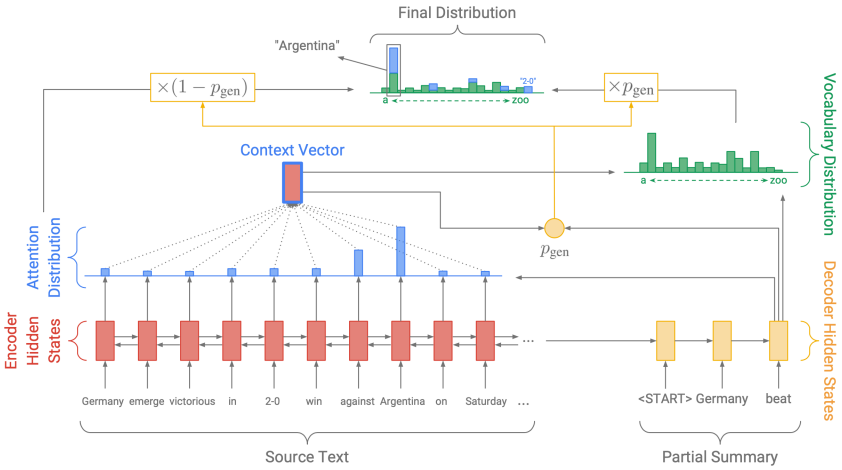

图 11: PGN 模型架构

coverage 决定是否使用 coverage 机制，然后在 Attention 和 train 模块中根据这一变量决定是否加入 coverage 的信息。

- **实现 fine tuning**：同样可以用一个全局变量 fine_tune 来决定是正常训练模式还是 fine-tuning 模式，如果是 fine-tuning 模式，则将与 coverage 无关的权重都固定住，只训练与 coverage 相关的权重，具体可通过以下示例代码实现。注意在 fine-tuning 阶段要以一个比原先小很多的学习率来进行训练才能有效果，建议是原本的学习率的十分之一。

```
if p.fine_tune:
    # In fine-tuning mode, we fix the weights of all parameters
                                        except attention.wc.
    print('Fine-tuning mode.')
    for name, params in model.named_parameters():
        if name != 'attention.wc.weight':
            params.requires_grad=False
```

实现 Beam Search 的优化

主要实现三种优化方式：

- **Length normalization**：根据序列长度对序列的分数做归一化，否则

短序列将对长序列有天然优势。

- **Coverage normalization**：利用我们计算到的 coverage_vector，实现这一功能，目的是鼓励输出序列尽可能全面的涵盖 source 中的信息（这个地方跟 PGN 中的 coverage 机制目的是不同的，注意区分）[7]。
- **EOS normalization**：为了解决 Beam Search 在 decode 的时候总是不会主动生成 <EOS> token 的问题。加入对 EOS token 的概率的 normalization。

以上优化方式定义在 https://opennmt.net/OpenNMT/translation/beam_search/中，请详细阅读具体计算公式。

第三次作业

第三次作业的核心目标是让大家了解并且实现一些模型训练的优化技巧以及数据增强的技术。如图 12所示。

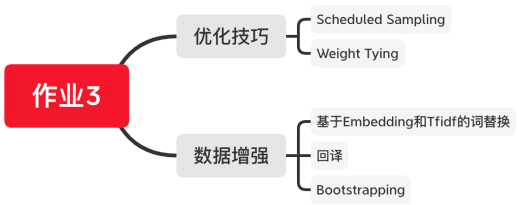

图 12: 第三次作业架构图

4.3.3 第三次作业描述

对于本次任务，需要完成如下的部分：

- **第一**：加入一些优化技巧并评估训练结果。
- **第二**：通过数据增强来增加样本量。

4.3.4 第三次作业详解

优化技巧

在 Seq2seq 模型中，由于 Decoder 在预测阶段需要根据上一步的输出来的生成当前 time step 的输出，所以会面临 “Exposure bias” 的问题：在训练阶段我们使用 ground truth 作为 decoder 的输入（称之为 Teacher forcing），预测阶段却只能使用 decoder 上一步的输出，导致输入样本分布不一样，而影响 decoder 的表现。对此，我们有两种技巧进行优化：

- **Weight tying**：即共享 Encoder 和 Decoder 的 embedding 权重矩阵，使得其输入的词向量表达具有一致性。
- **Scheduled sampling**：即在训练阶段，将 ground truth 和 decoder 的输出混合起来使用作为下一个 time step 的 decoder 的输入，具体的做法是每个 time step 以一个 p 的概率进行 Teacher forcing，以 1-p 的概率不进行 Teacher forcing。p 的大小可以随着 batch 或者 epoch 衰减，即开始训练的阶段完全使用 groud truth 以加快模型收敛，到后面逐渐将 ground truth 替换成模型自己的输出，到训练后期就与预测阶段的输出一致了。

数据增强

少样本问题是 NLP 领域经常面临的，尤其是在金融或者医疗等垂直领域，更是缺乏高质量的标注语料，所以数据增强是一种常用的技术。这一环节我们将实现以下几种数据增强的技术：

- **单词替换**：由于中文不像英文中有 WordNet 这种成熟的近义词词典可以使用，我们选择在 embedding 的词向量空间中寻找语义最接近的词 [2]。通过使用在大量数据上预训练好的中文词向量，我们可以到每个词在该词向量空间中语义最接近的词，然后替换原始样本中的词，得到新的样本。但是有一个问题是，如果我们替换了样本中的核心词汇，比如将文案中的体现关键卖点的词给替换掉了，可能会导致核心语义的丢失。对此，我们有两种解决办法：1. 通过 tfidf 权重对词表里的词进行排序，然后替换排序靠后的词；2. 先通过无监督的方式挖掘样本中的主题词，然后只替换不属于主题词的词汇。
- **回译**：我们可以使用成熟的机器翻译模型，将中文文本翻译成一种外文，然后再翻译回中文，由此可以得到语义近似的新样本。

- **Bootstrapping**：即自助式生成样本。当我们训练出一个文本生成模型后，我们可以利用训练好的模型为我们原始样本中的 reference 生成新的 source，并作为新的样本继续训练我们的模型。

参考文献

[1] Bengio, S., Vinyals, O., Jaitly, N., and Shazeer, N. Scheduled sampling for sequence prediction with recurrent neural networks. *CoRR abs/1506.03099* (2015).

[2] Jiao, X., Yin, Y., Shang, L., Jiang, X., Chen, X., Li, L., Wang, F., and Liu, Q. Tinybert: Distilling bert for natural language understanding. *ArXiv abs/1909.10351* (2019).

[3] Nallapati, R., Xiang, B., and Zhou, B. Sequence-to-sequence rnns for text summarization. *CoRR abs/1602.06023* (2016).

[4] Press, O., and Wolf, L. Using the output embedding to improve language models. *CoRR abs/1608.05859* (2016).

[5] Schmidt, F. Generalization in generation: A closer look at exposure bias. *Proceedings of the 3rd Workshop on Neural Generation and Translation* (2019).

[6] See, A., Liu, P. J., and Manning, C. D. Get to the point: Summarization with pointer-generator networks. *CoRR abs/1704.04368* (2017).

[7] Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Kaiser, L., Gouws, S., Kato, Y., Kudo, T., Kazawa, H., Stevens, K., Kurian, G., Patil, N., Wang, W., Young, C., Smith, J., Riesa, J., Rudnick, A., Vinyals, O., Corrado, G., Hughes, M., and Dean, J. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR abs/1609.08144* (2016).