* 为什么要复用：底层特征的可复用性
* 为什么要fine-tuning：高层特征任务的相关性
* Frozen：浅层加载的数据在训练任务中不动
* fine-tuning：网络中所有参数不断变化
* Word2Vec

word2vec是Google在2013年推出的一个工具。word2vec通过训练，可以将所有的词向量化，这样就可以定量的去度量词与词之间的关系，挖掘词之间的联系；同时还可以将词向量输入到各种RNN网络中进一步处理。因此，word2vec 输出的词向量可以被用来做很多自然语言处理相关的工作，比如聚类、找同义词、词性分析等等、文本分析等，是自然语言处理的重要基础。

**为什么要进行词嵌入(word embedding)，而不是用one-hot？**

1. One-hot编码导致**维独灾难**，在实际应用中，词向量维度太大，会造成网络参数量大、网络推理速度慢、网络运行占用内存高等问题。
2. One-hot**编码过于稀疏**，在one-hot编码的词向量中，数值几乎全部是0，非常稀疏，很可能导致实际中网络难以收敛。
3. **无法表示词间的关系，**对于one-hot编码，任意两个词间的相似度都为0，这是违背实际情况的。

**word2vec提出了CBOW与skip-gram结构：**

图示

描述已自动生成

* CBOW结构：根据输入周围 2c 个词来预测出这个词本身（即通过上下文预测词语）;
* skip-gram结构：根据输入词来预测周围 2c个词（即预测词语的上下文）。

**Huffman:**

一般来说，训练用的语料库都非常大。而在语料库中，有一些词出现频率很高，还有一些词出现频率很低，而且这种频率差异是非常巨大的。那么采用Huffman树后，出现频率很高的常用词路径短，计算量小，从而降低了整个word2vec模型在训练时的计算开销。

Huffman树的构建：

1. 根据给定的n个权值{w1, w2, ... wn}，构造n棵只有根节点的二叉树。
2. 在森林中选取两棵根节点权值最小的树作为左右子树，构造一颗新的二叉树，置新二叉树根节点权值为其左右子树根节点权值之和。注意，左子树的权值应小于右子树的权值。
3. 从森林中删除这两棵树，同时将新得到的二叉树加入森林中。

换句话说，之前的2棵最小的根节点已经被合并成一个新的结点了。

1. 重复上述两步，直到只含一棵树为止，这棵树即是“哈弗曼树”。

假设我们从某书籍中收集了一段语料，统计在语料中出现的词及每个词出现的次数，生成词汇表。然后通过每个词在语料中的出现次数建立Huffman树。

另外在word2vec中约定：**从Huffman树根节点(root)开始，每次父节点向左子叶遍历编码为1，向右子叶遍历为0**。**在word2vec中使用Huffman树的重要原因就是降低训练时的计算量。**

**CBOW(Continuous Bag of Words)结构：**

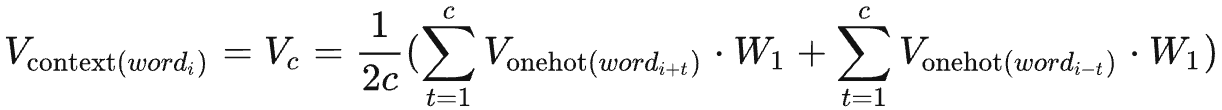
图示

描述已自动生成

在之前提到过，CBOW根据输入词周围 2c 个词来预测出这个词本身。如果当前网络已经充分训练，那么输入you、are、 wise、and四个词，则应该输出词very。那么CBOW结构的word2vec网络是如何训练的？

* + 从INPUT->PROJECTION层

CBOW结构首先会取中心词的 2c 个上下文词，然后用这些上下文词的one-hot编码向量乘以 W1 权重再求和：



其中代表词的one-hot编码向量；代表的上下文词的词向量之和（为了便于书写记为）。

* + 从PROJECTION->OUTPUT层(Hierarchical Softmax)

以词"very"为例在Huffman树中需要分类4次（编码为0100），第一次分类结果  概率为：

, 图示, 示意图

描述已自动生成

形状, 矩形

描述已自动生成

第二次分类为 概率为：



第三次分类为 的概率为：



第四次分类为的概率为：



所以词"very"最终Hierarchical Softmax最终概率为：

文本

描述已自动生成

图示

描述已自动生成

推广一下，词最终Hierarchical Softmax最终概率为

图片包含 文本

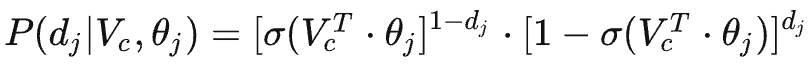
描述已自动生成

其中为 的Huffman树路径长度（如"very"为4），而 为：

手机屏幕截图

低可信度描述已自动生成

写成一个表达式：



* + 从OUTPUT->PROJECTION->INPUT训练

在训练时，我们显然希望**输入是的**2c **个上下文词时输出是**，即概率 越大越好。那么最终优化目标就是对语料中每个词 有 L 最大：

文本, 信件

描述已自动生成

其中C代表训练使用的语料库，代表词的Huffman树路径长度。而优化目标是乘法形式，所以取对数log将优化目标转化为加法：

文本

中度可信度描述已自动生成

由于优化目标是使 L′ 最大，那么训练采用梯度上升算法，即每当获取新的训练中心词 时都会通过梯度更新一次权重。记：



这里 是中心词为 时的优化目标，即希望通过调整  和  使  最大。

所以计算  对 的偏导数：

文本

描述已自动生成

注意。

那么  中的参数 的更新公式为：

图示

描述已自动生成

除了，训练还需更新，所以还要计算对  的梯度：

文本

描述已自动生成

即可更新 中的上下文词的词向量 （注意这里的  ）：

图示, 示意图

描述已自动生成

其中 η 代表梯度上升学习率。

对于上式一个比较通俗且不严谨的理解：

误差传给了谁，谁就会把梯度返回给传它误差的节点，即“原路送回“。在前传中通过 节点将误差传递给了后续网络，那么在反传中后续网络也要把自己所有的梯度还给  节点，然后 又会把通梯度返还给 2c 个上下文词的词向量。

图示

描述已自动生成

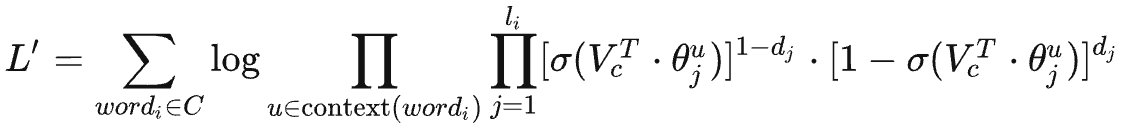
**skip-gram结构**

图表

描述已自动生成

在之前提到过，skip-gram根据输入词词来预测出周围 2c 个上下文词。如果当前网络已经充分训练，那么输入very，则应该输出you、are、 wise、and四个上下文词。

从INPUT->PROJECTION计算  的词向量 ，然后PROJECTION直接向后续分层softmax输出 （CBOW是求和）。skip-gram的优化目标与CBOW稍微不同：



其中多出的 符号代表skip-gram要通过 2c 个Huffman树分别输出 2c 个上下文词， 中的 u 代表每个输出词对应的  参数。skip-gram与CBOW非常接近，考虑篇幅这里不再介绍。

**word2vec缺点**

* + OOV (Out of vocabulary)

在word2vec中，词汇表从开始训练就已经是确定的。那么在使用时，必然会有词不在词汇表中。一般使用<UNKNOW>特殊标志符解决OOV问题，但是当句子中<UNKNOW>过多时必然严重影响精度。

后续ELMO使用char CNN、Bert使用word piece embedding，基本解决了OOV问题。

* + 无法处理多义词

很多词在不同语境是有含义不同，即多义词。而word2vec中所有词的embedding向量都是训练好即固定的，无法在使用时根据上下文调整，导致处理多义词效果差。

ELMO和Bert使用训练language model，动态生成embedding向量，解决多义词问题。

* ELMO: Embeddings from Language Models

ELMO是个根据当前上下文对 Word Embedding 动态调整的思路，是基于特征融合（Feature-based Pre-Training）的预训练方法。

**解决的问题：**

* + 建模词语复杂的语法、语义用法。
  + 使词语的表征根据输入的上下文而变化。

双层LSTM网络：

图示

描述已自动生成

双向LSTM网络：

图示

描述已自动生成

图示

描述已自动生成

图示, 示意图

描述已自动生成

前向语言模型：

文本

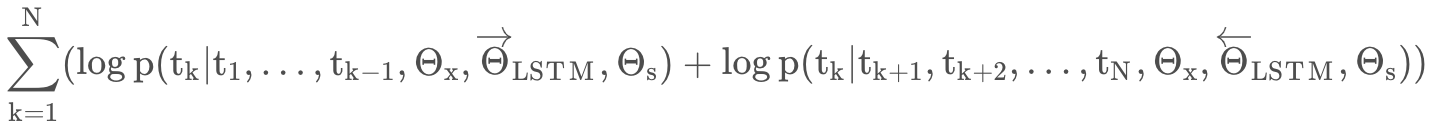
描述已自动生成

后向语言模型：

文本

中度可信度描述已自动生成

双向语言模型：



由于ELMo模型使每一层的向量都使用，因此对于一个token通过一个L层的网络会产生2L+1个表征（每一层双向两个向量，初始输入一个向量，共有3个）

文本

描述已自动生成

下游任务可表达为：

图示

描述已自动生成

其中是每层向量的权重，是跟任务相关的可训练的缩放系数。

**总结**：ELMo 是task specific上biLM 中间层表示的线性组合。基于此，我们应该认识到相比于word2vec 预训练的到的产物是词向量，这里的预训练产物是biLM，而ELMo 是最终根据task specific 学习得到的结果。ELMo向量是多个输出层及输入层，按照一定权重相乘得到的。这个权重怎么来的？针对具体的nlp任务，我们用的时候，需要再次训练去得到的这个权重。最简单的方法，就是权重都设为一样。

**ELMo核心步骤**

1. 第一层是普通的word embedding 可以用wrod2vec或者glove来得到，或者使用character level得到token embedding 这部分是general embedding，上下文无关。文中使用的character level的CNN+Highway.
2. 后面连接两个biLSTM 去encode 输入（同时也有残差连接), 每一层LSTM得到的输出（隐状态）作为每个词的上下文相关的word vectors.
3. 这样每个词就会有（L+1）个词向量，L为biLSTM的层数.
4. 词向量的线性组合，针对不同的任务，不同层的向量做不同的权重加和。 其中权重s是一个维度为L的vector，参与训练。

**优缺点**

1. 考虑上下文，针对不同的上下文生成不同的词向量。表达不同的语法或语义信息。如“活动”一词，既可以是名词，也可以是动词，既可以做主语，也可以做谓语等。针对这种情况，ELMo能够根据不同的语法或语义信息生成不同的词向量。使用LSTM提取特征，而LSTM提取特征的能力（远）弱于Transformer
2. 使用向量拼接方式融合上下文特征，这种方式获取的上下文信息效果不如想象中好（偏弱）
3. 训练时间长，这也是RNN的本质导致的，和上面特征提取缺点差不多

图示

描述已自动生成

* BERT：Pre-training of Deep Bidirectional Transformers for Language Understanding

假如给一个句子 “能实现语言表征[mask]的模型”，遮盖住其中“目标”一词。从前往后预测[mask]，也就是用“能/实现/语言/表征”，来预测[mask]；或者，从后往前预测[mask]，也就是用“模型/的”，来预测[mask]，称之为单向预测 unidirectional。单向预测，不能完整地理解整个语句的语义。于是研究者们尝试双向预测。把从前往后，与从后往前的两个预测，拼接在一起 [mask1/mask2]，这就是双向预测 bi-directional。 BERT 的作者认为，bi-directional 仍然不能完整地理解整个语句的语义，更好的办法是用上下文全向来预测[mask]，也就是用 “能/实现/语言/表征/../的/模型”，来预测[mask]。BERT 作者把上下文全向的预测方法，称之为 deep bi-directional。如何来实现上下文全向预测呢？BERT 的作者建议使用 Transformer 模型。 这个模型的核心是聚焦机制，对于一个语句，可以同时启用多个聚焦点，而不必局限于从前往后的，或者从后往前的，序列串行处理。不仅要正确地选择模型的结构，而且还要正确地训练模型的参数，这样才能保障模型能够准确地理解语句的语义。

BERT损失函数仅考虑被屏蔽词的预测，而忽略非屏蔽词的预测。结果，该模型的收敛速度比方向模型慢，不过这一特征被上下文感知能力的提高所抵消。

图示

描述已自动生成

图形用户界面, 图示, 应用程序

描述已自动生成

图片包含 游戏机, 键盘

描述已自动生成

**input = token\_embedding + segment\_embedding + positional\_embedding**

**token\_embedding:**可以是随机初始化，也可以使用 word2vec；

**segment\_embedding :**用于对两个句子进行区分，CLS到中间SEP，全部都是EA（整个句子都是一样的），也可以全部用0；SEP到SEP，全部用EB，也可以全部用1；

**positional\_embedding**:区别于transformer中的 positional\_encoding，使用随机初始化。

• **MLM**: Randomly masks some of the tokens from the input, and the objective is to predict the original word based only on its context.

• **NSP**: To predict whether sentence B is the next sentence of A.

Bert采用transformer的encode有两个训练任务，**一个**是NSP（next sentence prediction），用来判断两个句子之间的关系，使用SEP对句子进行分割。前接一个CLS，训练时，将CLS输出后面接一个二分类器。当选择句子A和B作为预训练例子时， 50%的可能性B是A的下一句，50%的可能性是语料中的随机句子；**另一个**是MLM（Masked Language Model）掩码语言模型：随机选择15%的位置用于预测, 被选中的token被替换为:80%的概率替换成[MASK]；10%的概率随机token；10%不变动；然后, 每个位置用交叉熵即（CE loss）预测token。

图示

描述已自动生成

从这里可以看出，NLP 四大类任务都可以比较方便地改造成 Bert 能够接受的方式。除了生成类任务外，Bert 其它都覆盖到了，而且改造起来很简单直观。**这其实是 Bert 的非常大的优点，这意味着它几乎可以做任何 NLP 的下游任务，具备普适性，这是很强的。**

在QA任务中，我们将输入问题和片段组成一个序列，问题采用A embedding，片段采用B embedding。在fine-tuning阶段，我们只引入了一个初始向量 和一个终止向量。词是回答范围的开始的概率，通过和S的点乘再加上对段落中所有词的softmax获得：来实现。类似的公式同样用于回答范围的结尾。位置 到位置 j 的候选范围的分数则通过 来计算，在时最大的分数范围用于预测。训练目标是正确的开始和结束的对数似然之和。

例如，微博情感分类任务：

1. 获取通用的预训练模型，比如谷歌中文BERT
2. 在相同领域上继续做模型训练，比如在微博文本上进行训练（**Domain tansfer 领域自适应，或 领域迁移）**
3. 在任务相关的小数据上继续训练，在微博情感任务文本上进行训练（**task transfer 任务自适应，或 任务迁移）**
4. 在任务相关数据上做具体的任务 （**fine-tune 微调**）

图示

描述已自动生成

**总结：**BERT 的 Encoder 架構很適合做自然語言理解 NLU 任務，但如文章摘要等自然語言生成 NLG 的任務就不太 okay

**模型尺寸**：用L表示层数, H为隐藏层数,，A为self-attention的头数。

BERTBASE​(L=12，H=768，A=12，TotalParameters=110M)

BERTLARGE​(L=24，H=1024，A=16，TotalParameters=340M)

* GPT：Generative Pre-Training

图示

描述已自动生成

GPT1: Generative Pre-Training。是一种基于Fine-tuning的模式的生成式的预训练模型，由Open AI团队发表自论文《Improving Language Understanding by Generative Pre-Training》。一种半监督学习方法，它致力于用大量无标注数据让模型学习“常识”，以缓解标注信息不足的问题。适配多**分类**、**文本蕴含**、**相似度**、**多项选择**这几类问题。

GPT的训练过程分为两个阶段：无监督Pre-training 和有监督Fine-tuning。在Pre-training阶段使用单向 Transformer 学习一个语言模型，对句子进行无监督的 Embedding，在fine-tuning阶段，根据具体任务对 Transformer 的参数进行微调，目的在于学习一种通用的 Representation 方法，针对不同种类的任务只需略作修改便能适应。

**选用Decoder的理由：**

语言模型是利用上文预测下一个单词的，因为 Decoder 使用了 Masked Multi Self-Attention 屏蔽了单词的后面内容，所以 Decoder 是现成的语言模型。又因为没有使用 Encoder，所以也就不需要 encoder-decoder attention 了。

**GPT与基本的Transformer相比，进行了以下修改：**

* + 在GPT中，作者对position embedding矩阵进行随机初始化，并让模型自己学习，而不是采用正弦余弦函数进行计算。
  + GPT目标是服务于单序列文本的生成式任务，所以放弃了关于Encoder部分，包括Decoder的 Encoder-Decoder Attention层。
  + 将GLUE（Gaussian Error Linear Unit）作为误差函数 GLUE可视为ReLU的改进方法，ReLU将小于0的数据转换成0，大于0的部分不变，而GELU对其稍做调整，如下图所示:

图表, 折线图

描述已自动生成

图形用户界面

中度可信度描述已自动生成

GPT改造施工图如上：对于分类问题，加上一个起始和终结符号即可；对于句子关系判断问题，比如 Entailment，两个句子中间再加个分隔符即可；对文本相似性判断问题，把两个句子顺序颠倒下做出两个输入即可，这是为了告诉模型句子顺序不重要；对于多项选择问题，则多路输入，每一路把文章和答案选项拼接作为输入即可。从上图可看出，这种改造还是很方便的，不同任务只需要在输入部分施工即可。

模型实现：

1. 在预训练Pretrain部分，用u表示每一个token(词)，当设置窗口长度为k，根据超参数Θ，用前面的k个词预测后面的词i。

文本

描述已自动生成

具体方法如下：输入为词编码U乘词嵌入参数We加上位置参数Wp；We是token的Embedding矩阵，Wp是位置编码的 Embedding 矩阵。用voc表示词汇表大小，pos 表示最长的句子长度，dim表示Embedding维度，则 Wp 是一个pos×dim矩阵，We 是一个voc×dim的矩阵。在GPT中，作者对position embedding矩阵进行随机初始化，并让模型自己学习，而不是采用正弦余弦函数进行计算。

文本

描述已自动生成

1. 模型训练阶段，以判断句子感情色彩为例：



文本

描述已自动生成

1. 正常来说，我们应该调整参数使得 L2 最大，但是为了提高训练速度和模型的泛化能力，我们使用 Multi-Task Learning，GPT 在微调的时候也考虑预训练的损失函数，同时让它最大似然 L1 和 L2



这里使用的L1还是之前语言模型的损失（似然），但是使用的数据不是前面无监督的数据U，而是使用当前任务的数据C，而且只使用其中的X，而不需要标签y。

**GPT-2:** **Zero-shot Learning(零样本学习)**

GPT-2的核心思想并不是通过二阶段训练模式（预训练+微调）获得特定自然语言处理任务中更好的性能。而是**彻底放弃微调阶段**，仅通过大规模多领域的数据预训练，让模型在Zero-shot Learning的设置下**自己学会解决多任务**的问题。与之相对的是，在特定领域进行监督微调的专家模型并不具备多任务场景下的普适性。GPT-2在Zero-shot Learning设置下依然能够很好执行各种任务，证明了自然语言处理领域通用模型的可能性。

对于Zero-Shot问题，则需要考虑目标的风格以及分布情况，并实现一些训练集到测试集的映射（如处理特殊符号、缩写等），从而实现从已知领域到未知领域的迁移学习。GPT-2在Zero-Shot（尤其是小数据集Zero-Shot）以及长文本（长距离依赖）中都表现优异。

与之前的实现方法最大的不同是：GPT-2的训练数据在数量、质量、广泛度上都有大幅度提高：抓取了大量不同类型的网页，并且经过筛选去重生成高质量的训练数据，同时训练出体量更巨大的模型。

与第一代GPT模型相比，GPT-2在模型结构上改动极小、在复用GPT的基础上，GPT-2做了以下修改：

* 1. LN层被放置在Self-Attention层和Feed Forward层前，而不是像原来那样后置。
  2. 在最后一层Transformer Block后增加了LN层。
  3. 修改初始化的残差层权重，缩放为原来的sqrt（N）。其中，N是残差层的数量。
  4. 特征向量维数从768扩展到1600，词表扩大到50257。
  5. Transformer Block的层数从12扩大到48。

**GPT3: Few-shot Learning的优秀表现**

与GPT-2在Zero-shot Learning设置下的惊喜表现相比，GPT-3在Few-shot Learning设置下的性能足以震惊所有人。在自然语言处理下游任务性能评测中，GPT-2在Zero-shot Learning设置下的性能表现远不如SOTA模型，而GPT-3在Few-shot Learning设置下的性能表现与当时的SOTA模型持平，甚至超越了SOTA模型。

GPT-3 的训练过程与 GPT-2 类似，但对模型大小、数据集大小与多样性、训练长度都进行了相对直接的扩充。关于语境学习，GPT-3 同样使用了与 GPT-2 类似的方法，不过 GPT-3 研究团队系统地探索了不同的语境学习设定。

OpenAI 团队明确地定义了用于评估 GPT-3 的不同设定，包括 zero-shot、one-shot 和 few-shot。

* Fine-Tuning (FT)：微调是近几年来最为常用的方法，涉及在期望任务的特定数据集上更新经过预训练模型的权重；
* Few-Shot (FS)：在该研究中指与 GPT-2 类似的，在推理阶段为模型提供少量任务演示，但不允许更新网络权重的情形；
* One-Shot (1S)：单样本与小样本类似，不同的是除了对任务的自然语言描述外，仅允许提供一个任务演示；
* Zero-Shot (0S)：零次样本除了不允许有任何演示外与单样本类似，仅为模型提供用于描述任务的自然语言指示。

以英-法翻译任务为例，展示四种方法。该研究将重点放在 zero-shot、one-shot 和 few-shot 上，其目的并非将它们作为竞品进行比较，而是作为不同的问题设置。OpenAI 团队特别强调了 few-shot 结果，因为其中许多结果仅仅略微逊色于 SOTA 微调模型。



**GPT-3的不足：**不具备真正的推理；存在不良内容出现；在高度程序化问题上表现不佳。

ChatGPT/InstructGPT详解

### 前言

GPT系列是OpenAI的一系列预训练文章，GPT的全称是Generative Pre-Trained Transformer，顾名思义，GPT的目的就是通过Transformer为基础模型，使用预训练技术得到通用的文本模型。目前已经公布论文的有文本预训练GPT-1，GPT-2，GPT-3，以及图像预训练iGPT。据传还未发布的GPT-4是一个多模态模型。最近非常火的ChatGPT和今年年初公布的InstructGPT是一对姐妹模型，是在GPT-4之前发布的预热模型，有时候也被叫做GPT3.5。ChatGPT和InstructGPT在模型结构，训练方式上都完全一致，即都使用了指示学习（Instruction Learning）和人工反馈的强化学习（Reinforcement Learning from Human Feedback，RLHF）来指导模型的训练，它们不同的仅仅是采集数据的方式上有所差异。所以要搞懂ChatGPT，我们必须要先读懂InstructGPT。

### 1. 背景知识

在介绍ChatGPT/InstructGPT之前，我们先介绍它们依赖的基础算法。

### ChatGPT的发展历程。

1. GPT-1用的是无监督预训练+有监督微调。
2. GPT-2用的是纯无监督预训练。
3. GPT-3沿用了GPT-2的纯无监督预训练，但是数据大了好几个量级。
4. InstructGPT在GPT-3上用强化学习做微调，内核模型为PPO-ptx。
5. ChatGPT沿用了InstructGPT，但是数据大了好几个量级。

### 1.1 GPT系列

基于文本预训练的GPT-1，GPT-2，GPT-3三代模型都是采用的以Transformer为核心结构的模型（图1），不同的是模型的层数和词向量长度等超参，它们具体的内容如表1。

图示

描述已自动生成

表1：历代GPT的发布时间，参数量以及训练量

表格

描述已自动生成

GPT-1比BERT诞生略早几个月。它们都是采用了Transformer为核心结构，不同的是GPT-1通过**自左向右生成式**的构建预训练任务，然后得到一个通用的预训练模型，这个模型和BERT一样都可用来做下游任务的微调。GPT-1当时在9个NLP任务上取得了SOTA的效果，但GPT-1使用的模型规模和数据量都比较小，这也就促使了GPT-2的诞生。

对比GPT-1，GPT-2并未在模型结构上大作文章，只是使用了更多参数的模型和更多的训练数据（表1）。GPT-2最重要的思想是提出了“**所有的有监督学习都是无监督语言模型的一个子集**”的思想，这个思想也是提示学习（Prompt Learning）的前身。GPT-2在诞生之初也引发了不少的轰动，它生成的新闻足以欺骗大多数人类，达到以假乱真的效果。甚至当时被称为“AI界最危险的武器”，很多门户网站也命令禁止使用GPT-2生成的新闻。

GPT-3被提出时，除了它远超GPT-2的效果外，引起更多讨论的是它1750亿的参数量。GPT-3除了能完成常见的NLP任务外，研究者意外的发现GPT-3在写SQL，JavaScript等语言的代码，进行简单的数学运算上也有不错的表现效果。GPT-3的训练使用了情境学习（In-context Learning），它是元学习（Meta-learning）的一种，元学习的核心思想在于通过少量的数据寻找一个合适的初始化范围，使得模型能够在有限的数据集上快速拟合，并获得不错的效果。

通过上面的分析我们可以看出从性能角度上讲，GPT有两个目标：

* 1. 提升模型在常见NLP任务上的表现效果；
  2. 提升模型在其他非典型NLP任务（例如代码编写，数学运算）上的泛化能力。

另外，预训练模型自诞生之始，一个备受诟病的问题就是预训练模型的偏见性。因为预训练模型都是通过海量数据在超大参数量级的模型上训练出来的，对比完全由人工规则控制的专家系统来说，预训练模型就像一个黑盒子。没有人能够保证预训练模型不会生成一些包含种族歧视，性别歧视等危险内容，因为它的几十GB甚至几十TB的训练数据里几乎肯定包含类似的训练样本。这也就是InstructGPT和ChatGPT的提出动机，论文中用3H概括了它们的优化目标：

* 有用的（Helpful）;
* 可信的（Honest）;
* 无害的（Harmless）。

### 1.2 指示学习（Instruct Learning）和提示（Prompt Learning）学习

指示学习是谷歌Deepmind的Quoc V.Le团队在2021年的一篇名为《Finetuned Language Models Are Zero-Shot Learners》文章中提出的思想。指示学习和提示学习的目的都是去挖掘语言模型本身具备的知识。不同的是Prompt是激发语言模型的**补全能力**，例如根据上半句生成下半句，或是完形填空等。Instruct是激发语言模型的理解能力，它通过给出更明显的指令，让模型去做出正确的行动。我们可以通过下面的例子来理解这两个不同的学习方式：

1. 提示学习：给女朋友买了这个项链，她很喜欢，这个项链太\_\_\_\_了。
2. 指示学习：判断这句话的情感：给女朋友买了这个项链，她很喜欢。选项：A=好；B=一般；C=差。

指示学习的优点是它经过多任务的微调后，也能够在其他任务上做zero-shot，而提示学习都是针对一个任务的。泛化能力不如指示学习。我们可以通过图2来理解微调，提示学习和指示学习。**（个人觉得这块内容总结的不太对）**

图示

描述已自动生成

**Prompt**：Prompt的核心思想是更好更充分的使用预训练模型，一般在自然语言处理中，有3种类型的预训练模型可以进行选择，他们分别是：

第一种是以Bert为典型的，使用Mask机制进行训练，类似于完形填空。

第二种是以GPT-3为典型，自回归模型，预测下一个token。

第三种是以T5为典型，Seq2Seq，序列到序列模型，比如机器翻译

### 1.3 人工反馈的强化学习

因为训练得到的模型并不是非常可控的，模型可以看做对训练集分布的一个拟合。那么反馈到生成模型中，训练数据的分布便是影响生成内容的质量最重要的一个因素。有时候我们希望模型并不仅仅只受训练数据的影响，而是人为可控的，从而保证生成数据的有用性，真实性和无害性。论文中多次提到了对齐（Alignment）问题，我们可以理解为模型的输出内容和人类喜欢的输出内容的对齐，人类喜欢的不止包括生成内容的流畅性和语法的正确性，还包括生成内容的有用性、真实性和无害性。

我们知道强化学习通过奖励（Reward）机制来指导模型训练，奖励机制可以看做传统模训练机制的损失函数。奖励的计算要比损失函数更灵活和多样（AlphaGO的奖励是对局的胜负），这带来的代价是奖励的计算是不可导的，因此不能直接拿来做反向传播。强化学习的思路是通过对奖励的大量采样来拟合损失函数，从而实现模型的训练。同样人类反馈也是不可导的，那么我们也可以将人工反馈作为强化学习的奖励，基于人工反馈的强化学习便应运而生。

RLHF最早可以追溯到Google在2017年发表的《Deep Reinforcement Learning from Human Preferences》，它通过人工标注作为反馈，提升了强化学习在模拟机器人以及雅达利游戏上的表现效果。

图示

描述已自动生成

InstructGPT/ChatGPT中还用到了强化学习中一个经典的算法：OpenAI提出的最近策略优化（Proximal Policy Optimization，PPO）。PPO算法是一种新型的Policy Gradient算法，Policy Gradient算法对步长十分敏感，但是又难以选择合适的步长，在训练过程中新旧策略的变化差异如果过大则不利于学习。PPO提出了新的目标函数可以在多个训练步骤实现小批量的更新，解决了Policy Gradient算法中步长难以确定的问题。其实TRPO也是为了解决这个思想但是相比于TRPO算法PPO算法更容易求解。

### 2. InstructGPT/ChatGPT原理解读

有了上面这些基础知识，我们再去了解InstructGPT和ChatGPT就会简单很多。简单来说，InstructGPT/ChatGPT都是采用了**GPT-3**的网络结构，通过**指示学习**构建训练样本来训练一个反应预测内容效果的奖励模型（RM），最后通过这个奖励模型的打分来指导强化学习模型的训练。InstructGPT/ChatGPT的训练流程如图4所示。

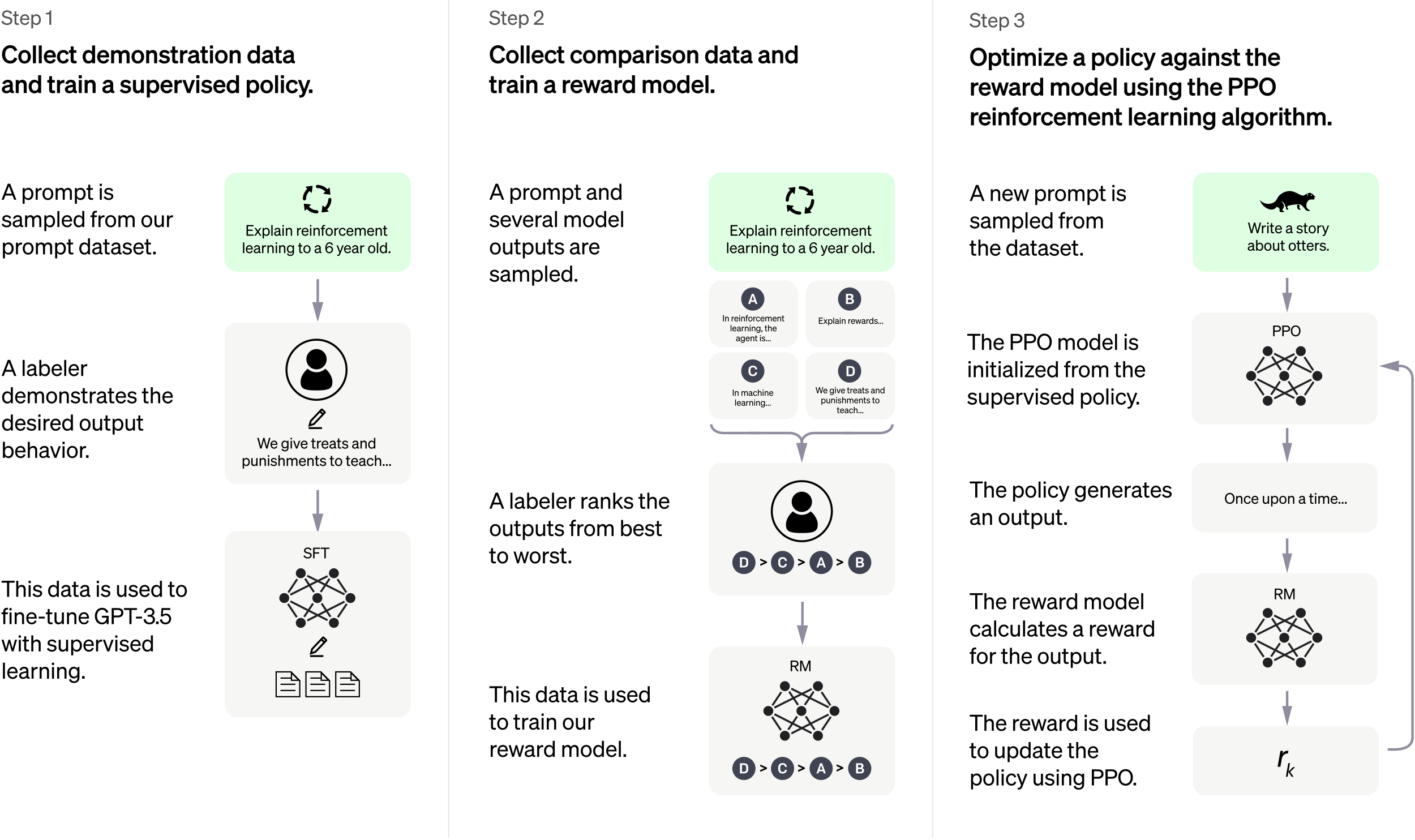


图4：InstructGPT的计算流程：（1）有监督微调（SFT）；（2）奖励模型（RM）训练；（3）通过PPO根据奖励模型进行强化学习。

从图4中我们可以看出，InstructGPT/ChatGPT的训练可以分成3步，其中第2步和第3步使得奖励模型和强化学习的SFT模型可以反复迭代优化。

1. 根据采集的SFT数据集对GPT-3进行有监督的微调（Supervised FineTune，SFT）；
2. 收集人工标注的对比数据，训练奖励模型（Reword Model，RM）；
3. 使用RM作为强化学习的优化目标，利用PPO算法微调SFT模型。

### Chat\_GPT/instruct\_GPT的流程：

前提：你现在有一个GPT-3模型。

**第一步**，先简单地微调一下GPT-3，需要人工标注。

1. 工程师团队设计了一个prompt dataset，里面有大量的提示文本，这些文本介绍了任务是啥。
2. 把这个prompt dataset发给人类标注员进行标注，其实就是回答问题。
3. 用这个标注过的数据集微调GPT-3。

**第二步**，训练一个奖励模型，需要人工标注。

1. 拿微调过的GPT-3去预测prompt dataset里面的任务，获得一系列结果，图中是四个。
2. 把这四个结果交给人类标注员进行标注，把预测的结果从好到坏进行标注。
3. 用这些标注的结果训练一个奖励模型（reward model），这个模型在下一步会用到。

**第三步**，用PPO持续更新策略的参数，不用任何人工标注。

1. 拿GPT-3预测prompt dataset中的文本，这里GPT-3被一个策略包装并且用PPO更新。
2. 用第二步训练好的奖励模型给策略的预测结果打分，用强化学习术语说就是计算reward。
3. 这个计算出来的分数会交给包着GPT-3内核的策略来更新梯度。

总结来看，就是结合了监督学习和强化学习。监督学习先让GPT-3有一个大致的微调方向，强化学习用了AC（Actor-Critic）架构里面的PPO（Proximal Policy Optimization）更新方法来更新GPT-3的参数。PPO是OpenAI的baseline method，可以说使用PPO是InstructGPT非常自然的一个选择。

我们将分别介绍InstructGPT/ChatGPT的数据集采集和模型训练两个方面的内容。

### 2.1 数据集采集

如图4所示，InstructGPT/ChatGPT的训练分成3步，每一步需要的数据也有些许差异，下面我们分别介绍它们。

### 2.1.1 SFT数据集

SFT数据集是用来训练第1步有监督的模型，即使用采集的新数据，按照GPT-3的训练方式对GPT-3进行微调。因为GPT-3是一个基于提示学习的生成模型，因此SFT数据集也是由提示-答复对组成的样本。SFT数据一部分来自使用OpenAI的PlayGround的用户，另一部分来自OpenAI雇佣的40名标注工（labeler）。并且他们对labeler进行了培训。在这个数据集中，标注工的工作是根据内容自己编写指示，并且要求编写的指示满足下面三点：

* **简单任务**：labeler给出任意一个简单的任务，同时要确保任务的多样性；
* **Few-shot任务**：labeler给出一个指示，以及该指示的多个查询-相应对；
* **用户相关的**：从接口中获取用例，然后让labeler根据这些用例编写指示。

### 2.1.2 RM数据集

RM数据集用来训练第2步的奖励模型，我们也需要为InstructGPT/ChatGPT的训练设置一个奖励目标。这个奖励目标不必可导，但是一定要尽可能全面且真实的对齐我们需要模型生成的内容。很自然的，我们可以通过人工标注的方式来提供这个奖励，通过人工对可以给那些涉及偏见的生成内容更低的分从而鼓励模型不去生成这些人类不喜欢的内容。InstructGPT/ChatGPT的做法是先让模型生成一批候选文本，让后通过labeler根据生成数据的质量对这些生成内容进行排序。

### 2.1.3 PPO数据集

InstructGPT的PPO数据没有进行标注，它均来自GPT-3的API的用户。既又不同用户提供的不同种类的生成任务，其中占比最高的包括生成任务（45.6%），QA（12.4%），头脑风暴（11.2%），对话（8.4%）等。

### 2.1.4 数据分析

因为InstructGPT/ChatGPT是在GPT-3基础上做的微调，而且因为涉及了人工标注，它们数据总量并不大，表2展示了三份数据的来源及其数据量。

表格

描述已自动生成

论文的附录A对数据的分布进行了更详细的讨论，这里我列出几个可能影响模型效果的几项：

* 数据中96%以上是英文，其它20个语种例如中文，法语，西班牙语等加起来不到4%，这可能导致InstructGPT/ChatGPT能进行其它语种的生成，但效果应该远不如英文；
* 提示种类共有9种，而且绝大多数是生成类任务，可能会导致模型有覆盖不到的任务类型；
* 40名外包员工来自美国和东南亚，分布比较集中且人数较少， InstructGPT/ChatGPT的目标是训练一个价值观正确的预训练模型，它的价值观是由这40个外包员工的价值观组合而成。而这个比较窄的分布可能会生成一些其他地区比较在意的歧视，偏见问题。

此外，ChatGPT的博客中讲到ChatGPT和InstructGPT的训练方式相同，不同点仅仅是它们采集数据上有所不同，但是并没有更多的资料来讲数据采集上有哪些细节上的不同。考虑到ChatGPT仅仅被用在对话领域，这里我猜测ChatGPT在数据采集上有两个不同：1. 提高了对话类任务的占比；2. 将提示的方式转换Q&A的方式。当然这里也仅仅是猜测，更准确的描述要等到ChatGPT的论文、源码等更详细的资料公布我们才能知道。

### 2.2 训练任务

我们刚介绍到InstructGPT/ChatGPT有三步训练方式。这三步训练会涉及三个模型：SFT，RM以及PPO，下面我们详细介绍它们。

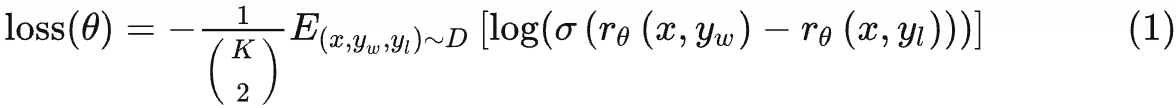
### 2.2.1 有监督微调（SFT）

这一步的训练和GPT-3一致，而且作者发现让模型适当过拟合有助于后面两步的训练。

### 2.2.2 奖励模型（RM）

因为训练RM的数据是一个labeler根据生成结果排序的形式，所以它可以看做一个回归模型。RM结构是将SFT训练后的模型的最后的嵌入层去掉后的模型。它的输入是prompt和Reponse，输出是奖励值。具体的讲，对于每个prompt，InstructGPT/ChatGPT会随机生成 K 个输出，然后它们向每个labeler成对的展示输出结果，也就是每个prompt共展示个结果，然后用户从中选择效果更好的输出。在训练时，InstructGPT/ChatGPT将每个prompt的  个响应对作为一个batch，这种按prompt为batch的训练方式要比传统的按样本为batch的方式更不容易过拟合，因为这种方式每个prompt会且仅会输入到模型中一次。

奖励模型的损失函数表示为式(1)。这个损失函数的目标是最大化labeler更喜欢的响应和不喜欢的响应之间的差值。



其中 是提示 和响应 在参数为的奖励模型下的奖励值， 是labeler更喜欢的响应结果， 是labeler不喜欢的响应结果。 D 是整个训练数据集。

### 2.2.3 强化学习模型（PPO）

强化学习和预训练模型是最近两年最为火热的AI方向之二，之前不少科研工作者说强化学习并不是一个非常适合应用到预训练模型中，因为很难通过模型的输出内容建立奖励机制。而InstructGPT/ChatGPT反直觉的做到了这点，它通过结合人工标注，将强化学习引入到预训练语言模型是这个算法最大的创新点。

如表2所示，PPO的训练集完全来自API。它通过第2步得到的奖励模型来指导SFT模型的继续训练。很多时候强化学习是非常难训练的，InstructGPT/ChatGPT在训练过程中就遇到了两个问题：

1. 问题1：随着模型的更新，强化学习模型产生的数据和训练奖励模型的数据的差异会越来越大。作者的解决方案是在损失函数中加入KL惩罚项  来确保PPO模型的输出和SFT的输出差距不会很大。
2. 问题2：只用PPO模型进行训练的话，会导致模型在通用NLP任务上性能的大幅下降，作者的解决方案是在训练目标中加入了通用的语言模型目标 ，这个变量在论文中被叫做PPO-ptx。

综上，PPO的训练目标为式(2)，这里参考**重要性采样**。

图示

中度可信度描述已自动生成

其中 是学习到的RL策略，是监督训练模型， 是预训练分布。KL奖励系数 β 和预训练损失系数 γ 分别控制KL惩罚和预训练梯度的强度。 对于“PPO”模型， γ 设置为0。**如无特殊说明，本文InstructGPT均指PPO-ptx模型。**

### 3. InstructGPT/ChatGPT的性能分析

不可否认的是，InstructGPT/ChatGPT的效果是非常棒的，尤其是引入了人工标注之后，让模型的“价值观”的正确程度和人类行为模式的“真实性”上都大幅的提升。那么，仅仅根据InstructGPT/ChatGPT的技术方案和训练方式，我们就可以分析出它可以带来哪些效果提升呢？

### 3.1 优点

* InstructGPT/ChatGPT的效果比GPT-3更加真实：这个很好理解，因为GPT-3本身就具有非常强的泛化能力和生成能力，再加上InstructGPT/ChatGPT引入了不同的labeler进行提示编写和生成结果排序，而且还是在GPT-3之上进行的微调，这使得我们在训练奖励模型时对更加真实的数据会有更高的奖励。作者也在TruthfulQA数据集上对比了它们和GPT-3的效果，实验结果表明甚至13亿小尺寸的PPO-ptx的效果也要比GPT-3要好。
* InstructGPT/ChatGPT在模型的无害性上比GPT-3效果要有些许提升：原理同上。但是作者发现InstructGPT在歧视、偏见等数据集上并没有明显的提升。这是因为GPT-3本身就是一个效果非常好的模型，它生成带有有害、歧视、偏见等情况的有问题样本的概率本身就会很低。仅仅通过40个labeler采集和标注的数据很可能无法对模型在这些方面进行充分的优化，所以会带来模型效果的提升很少或者无法察觉。
* InstructGPT/ChatGPT具有很强的Coding能力：首先GPT-3就具有很强的Coding能力，基于GPT-3制作的API也积累了大量的Coding代码。而且也有部分OpenAI的内部员工参与了数据采集工作。通过Coding相关的大量数据以及人工标注，训练出来的InstructGPT/ChatGPT具有非常强的Coding能力也就不意外了。

### 3.2 缺点

* InstructGPT/ChatGPT会降低模型在通用NLP任务上的效果：我们在PPO的训练的时候讨论了这点，虽然修改损失函数可以缓和，但这个问题并没有得到彻底解决。
* 有时候InstructGPT/ChatGPT会给出一些荒谬的输出：虽然InstructGPT/ChatGPT使用了人类反馈，但限于人力资源有限。影响模型效果最大的还是有监督的语言模型任务，人类只是起到了纠正作用。所以很有可能受限于纠正数据的有限，或是有监督任务的误导（只考虑模型的输出，没考虑人类想要什么），导致它生成内容的不真实。就像一个学生，虽然有老师对他指导，但也不能确定学生可以学会所有知识点。
* 模型对指示非常敏感：这个也可以归结为labeler标注的数据量不够，因为指示是模型产生输出的唯一线索，如果指示的数量和种类训练的不充分的话，就可能会让模型存在这个问题。
* 模型对简单概念的过分解读：这可能是因为labeler在进行生成内容的比较时，倾向于给给长的输出内容更高的奖励。
* 对有害的指示可能会输出有害的答复：例如InstructGPT/ChatGPT也会对用户提出的“AI毁灭人类计划书”给出行动方案。这个是因为InstructGPT/ChatGPT假设labeler编写的指示是合理且价值观正确的，并没有对用户给出的指示做更详细的判断，从而会导致模型会对任意输入都给出答复。虽然后面的奖励模型可能会给这类输出较低的奖励值，但模型在生成文本时，不仅要考虑模型的价值观，也要考虑生成内容和指示的匹配度，有时候生成一些价值观有问题的输出也是可能的。

### 3.3 未来工作

我们已经分析了InstrcutGPT/ChatGPT的技术方案和它的问题，那么我们也可以看出InstrcutGPT/ChatGPT的优化角度有哪些了。

* **人工标注的降本增效**：InstrcutGPT/ChatGPT雇佣了40人的标注团队，但从模型的表现效果来看，这40人的团队是不够的。如何让人类能够提供更有效的反馈方式，将人类表现和模型表现有机和巧妙的结合起来是非常重要的。
* **模型对指示的泛化/纠错等能力**：指示作为模型产生输出的唯一线索，模型对他的依赖是非常严重的，如何提升模型对指示的泛化能力以及对错误指示示的纠错能力是提升模型体验的一个非常重要的工作。这不仅可以让模型能够拥有更广泛的应用场景，还可以让模型变得更“智能”。
* **避免通用任务性能下降**：这里可能需要设计一个更合理的人类反馈的使用方式，或是更前沿的模型结构。因为我们讨论了InstrcutGPT/ChatGPT的很多问题可以通过提供更多labeler标注的数据来解决，但这会导致通用NLP任务更严重的性能下降，所以需要方案来让生成结果的3H和通用NLP任务的性能达到平衡。

### 3.4 InstrcutGPT/ChatGPT的热点话题解答

* **ChatGPT的出现会不会导致底层程序员失业**？从ChatGPT的原理和网上漏出的生成内容来看，ChatGPT生成的代码很多可以正确运行。但程序员的工作不止是写代码，更重要的是找到问题的解决方案。所以ChatGPT并不会取代程序员，尤其是高阶程序员。相反它会向现在很多的代码生成工具一样，成为程序员写代码非常有用的工具。
* **Stack Overflow 宣布临时规则：禁止 ChatGPT**。ChatGPT本质上还是一个文本生成模型，对比生成代码，它更擅长生成以假乱真的文本。而且文本生成模型生成的代码或者解决方案并不能保证是可运行而且是可以解决问题的，但它以假乱真的文本又会迷惑很多查询这个问题的人。Stack Overflow为了维持论坛的质量，封禁ChatGPT也是情理之中。
* **聊天机器人 ChatGPT 在诱导下写出「毁灭人类计划书」，并给出代码，AI 发展有哪些问题需关注？**ChatGPT的「毁灭人类计划书」是它在不可遇见的指示下根据海量数据强行拟合出来的生成内容。虽然这些内容看起来很真实，表达也很流畅，这说明的只是ChatGPT具有非常强的生成效果，并不表示ChatGPT具备毁灭人类的思想。因为他仅仅是一个文本生成模型，并不是一个决策模型。

### 4. 总结

就像很多人们算法刚诞生时一样，ChatGPT凭借有用性，真实性，无害性的效果，引起了业内广泛的关注和人类对AI的思考。但是当我们看完它的算法原理之后，发现它并没有业内宣传的那么恐怖。反而我们可以从它的技术方案中学到很多有价值的东西。InstrcutGPT/ChatGPT在AI界最重要的贡献是将强化学习和预训练模型巧妙的结合起来。而且通过人工反馈提升了模型的有用性，真实性和无害性。ChatGPT也进一步提升大模型的成本，之前还只是比拼数据量和模型规模，现在甚至也引入了雇佣的外包这一支出，让个体工作者更加望而却步。

* UniLM: Unified Language Model

## 图示 描述已自动生成

UniLM是2019年10月微软研究院在BERT的基础上提出的预训练语言模型，**被称为统一预训练语言模型**。从网络结构上看，它的结构是和BERT相同的编码器的结构。但是从它的预训练任务上来看，它不仅可以像自编码语言模型那样利用掩码标志的上下文进行训练，还可以像自回归语言模型那样从左向右的进行训练。甚至可以像Encoder-Decoder架构的模型先对输入文本进行编码，再从左向右的生成序列。该模型使用三种特殊的Mask的预训练目标，从而使得模型可以用于NLG，同时在NLU任务获得和BERT一样的效果

它可以完成单向、序列到序列和双向预测任务，可以说是结合了AR和AE两种语言模型的优点，**UniLM在文本摘要、生成式问题回答等领域取得了SOTA的成绩。**

#### 模型输入

首先对于一个输入句子，UniLM采用了Word Piece的方式对其进行了分词。除了分词得到的token嵌入，UniLM中添加了位置嵌入（和BERT相同的方式）和用于区分文本对的两个段的段嵌入（Segment Embedding）。为了得到整句的特征向量，UniLM在句子的开始添加了[SOS]标志。为了分割不同的段，它向其中添加了[EOS]标志。具体例子可以参考图1中的蓝色虚线框中的内容。

#### 训练策略

* **网络设置**：24层Transformer，1024个hidden size，16个attention heads，vocab\_size：30522，seq\_length：512；
* **参数大小**：340M
* **初始化**：直接采用Bert-Large的参数初始化
* **激活函数**：GELU，与Bert一样
* **dropout比例**：0.1
* **权重衰减因子**：0.01
* **batch\_size**：330
* **混合训练方式**：对于一个batch，1/3时间采用双向语言模型的目标，1/3的时间采用Seq2Seq语言模型目标，最后1/3平均分配给两种单向学习的语言模型，也就是left-to-right和right-to-left方式各占1/6时间
* **MASK方式**：总体比例15%，其中80%的情况下直接用[MASK]替代，10%的情况下随机选择一个词替代，最后10%的情况用真实值。还有就是80%的情况是每次只mask一个词，另外20%的情况是mask掉bi-gram或者tri-gram

#### 其优势总结如下

* 三种不同的训练目标，网络参数共享
* 网络参数共享，使得模型避免了过拟合于某单一的语言模型，使得学习出来的模型更加具有普适性
* 采用了Seq2Seq语言模型，使得其在能够完成NLU任务的同时，也能够完成NLG任务
* T5(2020.7.28 by google)

C[olin Raffel](https://arxiv.org/search/cs?searchtype=author&query=Raffel%2C+C)等人用统一的文本到文本Transformer探索[迁移学习](https://so.csdn.net/so/search?q=%E8%BF%81%E7%A7%BB%E5%AD%A6%E4%B9%A0&spm=1001.2101.3001.7020" \t "_blank)的极限。这次，T5团队想知道Transfomer能在多大程度上理解一门语言。人类学会一门语言后，通过迁移学习可以应用到各种各样的NLP任务上。T5模型的核心想法是找到一个能像人类这样的抽象模型。当我们人类交流时，我们总是从一个序列(A)开始，然后是另一个序列(B)。反过来，B成为另一个序列的起始序列，如图所示:

图示

中度可信度描述已自动生成

T5**（Transfer Text-to-Text Transformer）**模型：一个统一框架，靠着大力出奇迹，**将所有 NLP 任务都转化成 Text-to-Text（文本到文本）任务**。

图示

描述已自动生成

比如英德翻译，只需将训练数据集的输入部分前加上“translate English to German（给我从英语翻译成德语）” 就行。假设需要翻译"That is good"，那么先转换成 "translate English to German：That is good." 输入模型，之后就可以直接输出德语翻译 “Das ist gut.”。 对于需要输出连续值的 STS-B（文本语义相似度任务）， 也是直接输出文本。

通过这样的方式就能将 NLP 任务都转换成 Text-to-Text 形式，也就可以**用同样的模型，同样的损失函数，同样的训练过程，同样的解码过程来完成所有 NLP 任务**。

* **论文中做的对比实验：**

1. 首先作者们先对预训练模型中的多种模型架构（Transformer）进行了比对，最主要的模型架构可以分成下面三种。

图示

描述已自动生成

第一种，**Encoder-Decoder型**，即Seq2Seq常用模型，分成Encoder和Decoder两部分，对于Encoder部分，输入可以看到全体，之后结果输给Decoder，而Decoder因为输出方式只能看到之前的。此架构代表是MASS、BART等；

第二种， 相当于上面的**Decoder部分**，当前时间步只能看到之前时间步信息。典型代表是GPT2 、 CTRL ；

第三种，**Prefix LM（Language Model）型** ，可看作是上面Encoder和Decoder的融合体，一部分如Encoder一样能看到全体信息，一部分如Decoder一样只能看到过去信息， UniLM 便是此结构。

图片包含 图示

描述已自动生成

**Full-visible**全注意力，句中的每个词都可以关注到上下文中的每个词，适用于分类任务；**causal（Language Model）**每个词只能“注意”到之前的词，显然适用于翻译任务；

**prefix LM**一部分词能被句中的所有词“注意”，另一个部分词之间只能“注意”到之前的词，这种最有意思，因为它相当于把Transformer的Encoder和Decoder“挤”成了一个“不伦不类”的Xcoder，它非常适合于这篇论文的文本处理方式，因为任务前缀理应被所有词“注意”，此外在翻译任务中也是适用，因为源端句子x应当具备全注意力，而目标端句子采用causal模式（说到底还是因为它在意义上等同于Encoder和Decoder的混合体）。

1. 然后 是对预训练目标的大范围探索， 总共从四方面来进行比较。

图示

描述已自动生成

* 第一个方面，高层次方法（自监督的预训练方法）对比，总共三种方式，其中发现 Bert-style最好。
* 语言模型式，就是 GPT-2 那种方式，从左到右预测；
* BERT-style 式，就是像 BERT 一样将一部分给破坏掉，然后还原出来；
* Deshuffling （顺序还原）式，就是将文本打乱，然后还原出来。
* 第二方面，对文本一部分进行破坏时的策略，也分三种方法，效果最好的是 Replace Span 法，类似做法如 SpanBERT 也证明了有效性。
* Mask 法，如现在大多模型的做法，将被破坏 token 换成特殊符如 [M]；
* replace span（小段替换）法，可以把它当作是把上面 Mask 法中相邻 [M] 都合成了一个特殊符，每一小段替换一个特殊符，提高计算效率；
* Drop 法，没有替换操作，直接随机丢弃一些字符。
* 第三方面，到底该对文本百分之多少进行破坏呢，挑了4个值，10%，15%，25%，50%，最后发现 BERT 的15% 就很ok了。
* 第四方面，因为 Replace Span 需要决定对大概多长的小段进行破坏，于是对不同长度进行探索，2，3，5，10 这四个值，最后发现 3 结果最好。
* 结论：最终得到完整的 T5 模型，还有它的训练方法。

1. Transformer Encoder-Decoder 模型；
2. BERT-style 式的破坏方法；
3. Replace Span 的破坏策略；
4. 15 %的破坏比；
5. 3 的破坏小段长度。