论文学习心得

NLP研究综述Pre-trained models: Past, present and future

这篇文章主要介绍了NLP领域的发展历史，发展进程，以及如今的一些研究前沿。同时讲解了NLP中三个提升性能的方式，**其一是改变模型的结构架构；其二是改进训练数据，收集更多有效的数据；其三是改进训练任务，即训练的方法**

研究背景

1. NLP的发展历史
    - 在目前，使用神经网络模型代替了传统基于统计的自然语言处理模型，能够得到词的低维表示。
    - 使用神经网络强大的拟合能力，能够自己发现数据中的特征，减少特征工程。
    - 词向量的表示，通过上下文预测词的方法得到神经网络的词向量表示
    - 目前的模型的研究趋势变得越来越大，参数变得越来越多，尽可能在下游进行few-shot训练
2. 迁移学习与监督预训练
    - 迁移学习和预训练其实是差不多的，迁移学习主要有两种表现形式，首先是**预训练特征表示**，其次是**预训练参数迁移。**比如像词向量就是前者，预训练模型就是后者。
    - 几种迁移方法可以进行一个分类，从训练集是否有监督和测试集是否有监督可以分成四种迁移学习方法，归纳学习，推导学习，自主学习（训练集无标注），无监督迁移学习（均无标注）。在加上自监督学习（自己生成等价于带有标注的数据）
    - 预训练特征表示如何解决一词多义的问题**使用基于上下文的训练词向量表示**

transformer结构与预训练模型基础（PTM）

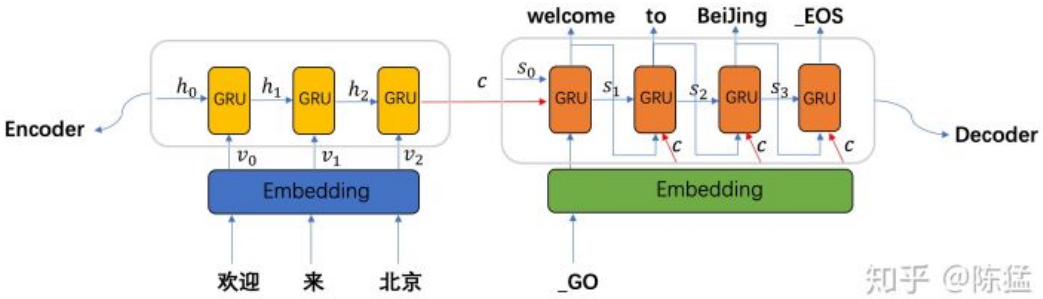
1. transformer结构
    - 其多头注意力机制已经在原理学习中详解
    - transformers结构中包括有多头注意力层，前馈层，正则层（**Residual Connection and Normalization**）
    - 对于多头注意力有几种变种
        - 传统类型，对于一个给定的词，计算他在整个input中需要关注的那些其他的词
        - seq2seq类型，将后面的词的权重mask掉，只能根据前面的词预测后面的词。
        - 交叉注意力（cross-attention），比如在传统的transformer里面把来自前一层的decoder的output1和来自本层的encoder的output2的都输入进入本层的decoder中，具体的，output1是作为Q，output2作为K/V，本层进行查询。
2. GPT 自回归训练的代表，只能看到上文
    - GPT主要运用于文本生成任务，因为这个正好就是只能看到上文。
    - 其损失函数是对数几率的相加，要求得到一组参数使得k词预测下一个词的对数概率之和尽可能小
3. bert模型，使用mask机制，上下文预测，实现自监督
    - 注意，bert在上游预训练任务不只是有MLM，还有NSP，其实是给出两个句子，判断这两个句子是不是相邻关系，有观点认为这是一个比较简单的任务。

基于bert与GPT的改进模型

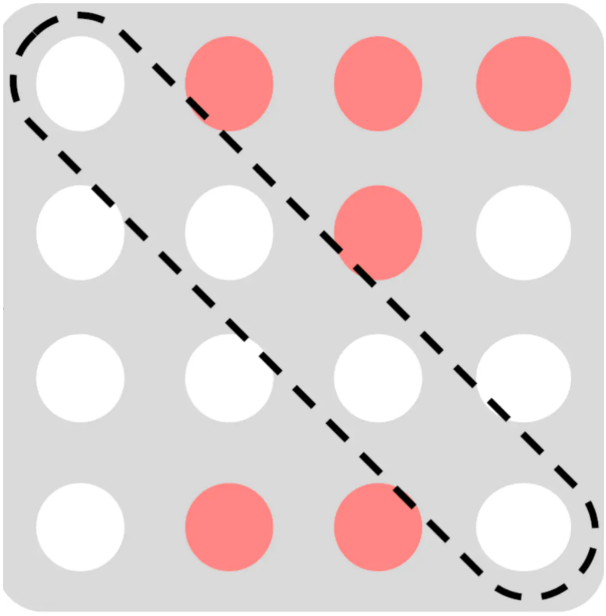
1. roberta模型
   - roberta模型除去了NSP任务，增加了更多的训练周期，使用更大的批次和更多数据
   - 使用了更长的训练句子，并且动态改变mask标签。
2. albert
   - 实现了多层的transformer块之间的参数共享，事实上，多个transformers模块只用了一层的参数，相当于把参数量降低了12分之1
   - albert对于NSP任务进行了替换，变成了SOP任务，即判断句子的先后顺序。

设计更有效的模型结构

1. 模型框架
   - 自回归模型：单向预测下一个词的出现，从左至右或者从右至左预测下一个词。典型为GPT
   - 自编码模型：就是用mask将句子中15%的单词随机屏蔽掉，这样的话通过上下文来预测单词。典型为bert。
   - 编码器-解码器模型：代表模型是T5，通过encoder层先将句子转化为词向量，并且通过序列模型得到句子的大致含义。通过decoder重新解码，decoder中的每一个单元接受decoder前一时刻和来自上一个预测词还有encoder的输入。encoder的输入相当于代入了一个注意力分配权重。



2. 架构的结合
   - XLNet通过一定的算法打乱句子中的词的顺序，比如c_1, c_2, c_3, c_4,我重新排列成c_3, c_2, c_4, c_1，这样的话模型在预测第四个词的位置时可以看到上下文（下文）的信息，并且不用通过mask机制来实现（因为下游任务中，并没mask，所以使用mask机制会导致上下游任务的不匹配）。这样的重新排序是通过改变模型的attention矩阵实现的。



3	2	4	1

- GLM通过自回归生成mask的机制实现对于mask机制的放弃，将两种架构结合。并通过二维位置编码的方法保证了mask位置的信息
- ELECTRA模型，有点像是使用了生成对抗的方法，构建一个生成器与判别器，生成器不断地去生成mask去掩盖住句子中的词语，并且通过context重新预测出这个词（有可能是同一个词，也有可能被替换了），然后判别器去判断这个token是否被替换了。这种训练方法最大程度地获取了监督信息，因为模型需要判断所有的词是否被替换过了。
- 还有许多模型等待学习。

使用更多来源的数据

1. 多语言预训练
    - 比如说像m-bert模型就是在多语料库上面预训练得到的模型。
    - 语料库又分为平行与非平行，平行语料库是意思一一对应的语料库，这一点在机器翻译领域非常重要。
    - 常见的训练方法有将两种语料合并成为一句话，然后随机mask，结合两种语料的信息去预测结果。
2. 多模态数据预训练
    - 最常见的是图像-文本学习，有两种模型类型：**双流与单流。**其中双流模型有分为：把图像和文本分别编码与交叉编码的模型，单流则是将两种数据（文本的embedding与图像的特征区域向量合并，送到transformer层里面）
3. 知识增强预训练（利用知识图谱）

提高计算效率

1. 体系结构的优化
    - 精度的损失换取存储空间的减小；使用并行化计算（多GPU，将模型与数据分块）；设置梯度断点方法，使得模型不用保持那么多梯度
2. 高效的训练方法
    - 在预训练任务中，挑选合适的mask位置（通过梯度信息与重要性信息）**为什么这样能加速?**
    - 调整学习率和批次，使用一个能够变化的学习率，有一种方法是在训练的开始阶段线性地增加学习率。
    - warm-up：先直接训练一个浅层的模型，比如我就训练一层，然后用这一层的参数去初始化全部模型的参数，比如其他相同的层

3. 高效的模型结构
    - 使用低秩核
    - 使用稀疏注意力机制
    - 使用全局注意力与**局部注意力**相结合的方法，比如一个句子或者一个段落中有些词他是可以看到这个段落的，其他的词只能关注到身边的词，固定window大小，这样实现稀疏注意力。
4. 模型压缩
    - 首先是参数共享，在介绍ALBert的时候已经分析过了
    - 知识蒸馏：有一个教师模型和一个学生模型，同时喂入数据，将教师模型的输出结果或者是隐藏层向量与学生模型的隐藏层向量进行对比，去监督学生模型的参数，最后的训练目标是让学生模型尽可能像老师模型。
    - 模型量化：模型中参数的低精度或者是混合精度的表示

关于预训练模型的理论研究

1. 研究模型是否真正获得了语言知识和世界知识，研究方法如下
    - 固定模型的参数，初始化一个新的线性层，判断模型是否能够完成特定的任务（如果能，说明之前得到的知识是有用的），这是一种比较表面但操作简单的研究方法
    - 表示分析：使用PTM的隐藏层去计算一些性质，比如相似度，空间距离等
    - 注意分析：观察模型的注意力矩阵，是否对于一些关键的句子成分有把握
    - 生成分析：即传统的生成任务
2. 模型的鲁棒性
    - 受到对抗攻击时鲁棒性不强
    - 对抗攻击数据集的有效性的验证方法并不是很好
3. 模型的稀疏性
    - 观察发现在一些任务当中其实并不需要用到transformer结构中的多头注意力机制（不用效果会更好），在机器翻译、抽象总结等任务中多头注意力多余
4. 理论分析
    - 预训练的好处，在于更好的优化与更好的正则化。

对抗攻击方法，基于attention-distribution

主要背景

- 针对于双语翻译的模型的对抗攻击训练，以提升模型的稳定性，如何提升这种对抗攻击的力度，使其对模型影响更限制，是本文的研究内容
- 常见的针对双语平行数据集（每一个数据是一个句子对，分为source与target文本）的扰动，主要是修改source文本，也有只修改target文本，也有少量研究修改两种文本（修改时要一一对应，source文本修改的地方，target文本只能并且必须修改对应的地方）。
- 同时，对于生成攻击的位置也是十分重要。

研究结果

- 研究发现，对于两种文本同时对应修改的攻击要更加有力一些。
- 同时，对于一段文本，修改位置靠前的单词对于模型精度的影响更大。（**这个是否要看他采用什么模型，如果是双向的话是不是没啥用**（笔者在文中提到了这一个讨论仅限于自回归模型））
- 事实上，双语文本中source文本的第一个单词往往对应的是target文本的第一个单词，这个笔者使用了注意力分数的大小来证明。

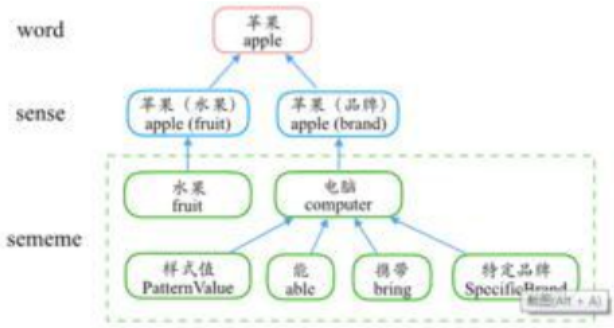
提出的新方法

- 本文针对得到的尽可能攻击token靠前的位置的结论，提出了一种基于**attention-distribution**攻击方法,首先我们对于多头注意力机制中的每一个头，我们关注这个注意力头的输出（模型内部的输出），这种输出中包含着source中所有单词对于target文本中的第一个单词的关注程度，选取这几个较高的token对于source文本进行改变，再对应改变target。
- 证明了这种方法生成的鲁棒性模型在评价标准中表现更优。

词语级别的攻击算法Word-level Textual Adversarial Attacking as Combinatorial Optimization

义原

在自然语言处理领域，义原知识库的构建旨在将语素应用于实际的场景中，其中义原通常被认为是单词的语义标签。HowNet是最著名的一个义原知识库，它用预定义的大约2000个语义集注释了超过10万个英语和中文单词。**一个单词可以由多个义原组成，也可以本身就是一个义原**。需要具体的语言知识。



An example of words in HowNet

在这篇文章中，我们认为进行词语替换的时候，当两个词语具有相同的义原组成时，就可以认为这两个词属性相近，可以进行替换。当我们采用这种方法时，**所找到的替换词要比同义词替换或者上下文替换更加多并且更加精确。**

粒子群算法

- PSO利用一个相互作用的个体群体，在特定的空间中迭代地寻找最优解。种群称为群，个体称为粒子。每个粒子在搜索空间中都有一个位置，并以可适应的速度移动。

  形式上，当在包含N个粒子的群中搜索D维连续空间$S \subset R^D$时，每个粒子的位置和速度可以分别用$x^n \in S$和$v^n \in R^D$表示,$n \in \{1, \ldots, N\}$。接下来，我们将逐步描述PSO算法。

  （1）**初始化。**在一开始，每个粒子被随机初始化为搜索空间中的位置和速度，初始速度的每个维度$v_d^n \in [-V_{max}, V_{max}], d \in \{1, \ldots, D\}$

  （2）**记录。**搜索空间中的每个位置都对应着一个优化分数。粒子达到优化得分最高的位置被记录为其个体最佳位置。在所有粒子的各个最佳位置中的最佳位置被记录为全局最佳位置。

  （3）**终止。**如果当前的全局最佳位置达到了期望的优化分数，则算法终止并输出全局最佳位置作为搜索结果。

  （4）**更新。**否则，每个粒子的速度和位置将根据其当前位置和个体最佳位置以及全局最佳位置进行更新。更新公式为：

$$v_d^n = \omega v_d^n + c_1 r_1 (p_d^n - x_d^n) + c_2 r_2 (p_d^g - x_d^g)$$
$$x_d^n = x_d^n + v_d^n$$

其中，ω为惯性权重，p_d^n和p_d^g分别为第n个粒子的个体最佳位置和全局最佳位置的第d个维数。c_1和c_2是加速度系数，它们是正值常数，控制粒子向其个体最佳位置和全局最佳位置移动的速度，r_1和r_2是随机系数。更新后，算法返回到**记录**步骤。

一个求解y=-x*(x-1) 在0,2上最大值的粒子群算法演示。**注意其中的r_1, r_2是随机值。注意到这里的速度是一维的，如果是多维的话可以用二维的double数组表示。**

```
public class AlgorithmPSO {
  int n=2; //粒子个数，这里为了方便演示，我们只取两个，观察其运动方向
    double[] y;
    double[] x;
    double[] v;
    double c1=2;
    double c2=2;
    double pbest[];
    double gbest;
    double vmax=0.1; //速度最大值
    //适应度计算函数，每个粒子都有它的适应度
    public void fitnessFunction(){
        for(int i=0;i<n;i++){
            y[i]=-1*x[i]*(x[i]-2);
        }
    }
    public void init(){ //初始化
        x=new double[n];
        v=new double[n];
        y=new double[n];
        pbest=new double[n];
        /***
         * 本来是应该随机产生的，为了方便演示，我这里手动随机落两个点，分别落在最大值两边
         */
        x[0]=0.0;
        x[1]=2.0;
        v[0]=0.01;
        v[1]=0.02;
        fitnessFunction();
        //初始化当前个体最优位置，并找到群体最优位置
        for(int i=0;i<n;i++){
            pbest[i]=y[i];
            if(y[i]>gbest) gbest=y[i];
        }
        System.out.println("算法开始，起始最优解:"+gbest);
        System.out.print("\n");
    }
    public double getMAX(double a,double b){
        return a>b?a:b;
    }
    //粒子群算法
    public void PSO(int max){
        for(int i=0;i<max;i++){
            double w=0.4;
            for(int j=0;j<n;j++){
                //更新位置和速度，下面就是我们之前重点讲解的两条公式。
```

```
                    v[j]=w*v[j]+c1*Math.random()*(pbest[j]-x[j])+c2*Math.random()*
(gbest-x[j]);
                    if(v[j]>vmax) v[j]=vmax;//控制速度不超过最大值
                    x[j]+=v[j];

                    //越界判断，范围限定在[0, 2]
                    if(x[j]>2) x[j]=2;
                    if(x[j]<0) x[j]=0;

                }
                fitnessFunction();
                //更新个体极值和群体极值
                for(int j=0;j<n;j++){
                    pbest[j]=getMAX(y[j],pbest[j]);
                    if(pbest[j]>gbest) gbest=pbest[j];
                    System.out.println("粒子n"+j+": x = "+x[j]+"  "+"v = "+v[j]);
                }
                System.out.println("第"+(i+1)+"次迭代，全局最优解 gbest = "+gbest);
                System.out.print("\n");
            }

        }
        //运行我们的算法
        public static void main(String[] args){
          AlgorithmPSO ts=new AlgorithmPSO();
            ts.init();
            ts.PSO(10);//为了方便演示，我们暂时迭代10次。
        }
    }
```

词语攻击中的义原与粒子群算法

- 在进行词语级别的攻击时，采用义原的方法搜索可能的替换值（**构建搜索集合**），然后采用粒子群搜索的方法筛选替换值（**进行搜索**）

  具体建模如下

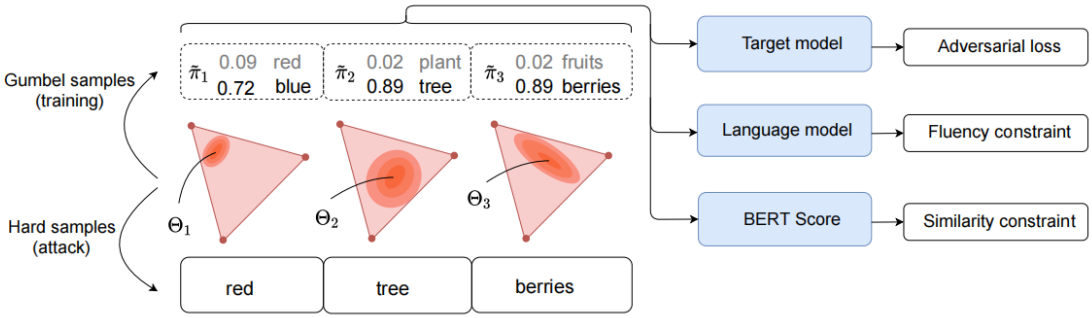
  - 我们把一句话看成是一个粒子，然后把一句话里面的单词数量（已经进行填补）看作是空间的维度D。
  - 通过重复随机替换句子的一个单词得到n个粒子，然后将模型预测出相反标签的概率作为他的适应度。
  - 最后通过上述公式，迭代求解，最后取那个适应度最大的粒子作为求解结果。
  - 一些注意的地方：会存在一个移动概率，这个粒子**会不会向个体最优或者全局最优移动方向移动**；会存在一个突变概率，这个粒子有一定的概率**突变到空间中可能的随机点**。

基于梯度方法寻找攻击样本的优化分布Gradient-based Adversarial Attacks against Text Transformers

基本思路

- 我们希望生成对抗性样本的时候，不是去单独搜索一个样本，而是形成一个对抗性样本的分布，并需要这个分布满足一些性质。能够找到比较多，质量比较高的样本。
  - 首先要成功地攻击模型，生成target-label
  - 构建的对抗样本要有效，不能太多地改变原意。
  - 构建的样本在语义上要流畅。

- 比如在下面这句话，每一个位置都对应着一个选词分布，这个选词分布是根据三个目标训练而来的。



- 一些评价指标，主要是将损失函数加权相加，就实现了模型对于上述三个目标的训练。
  - 首先是对抗样本攻击时产生的损失。(这个损失函数实际上是预测的label和原来一样的时候，损失就是$\kappa > 0$,否则只会更小)，**当然，这只是损失函数的一种形式，也可以是交叉熵**

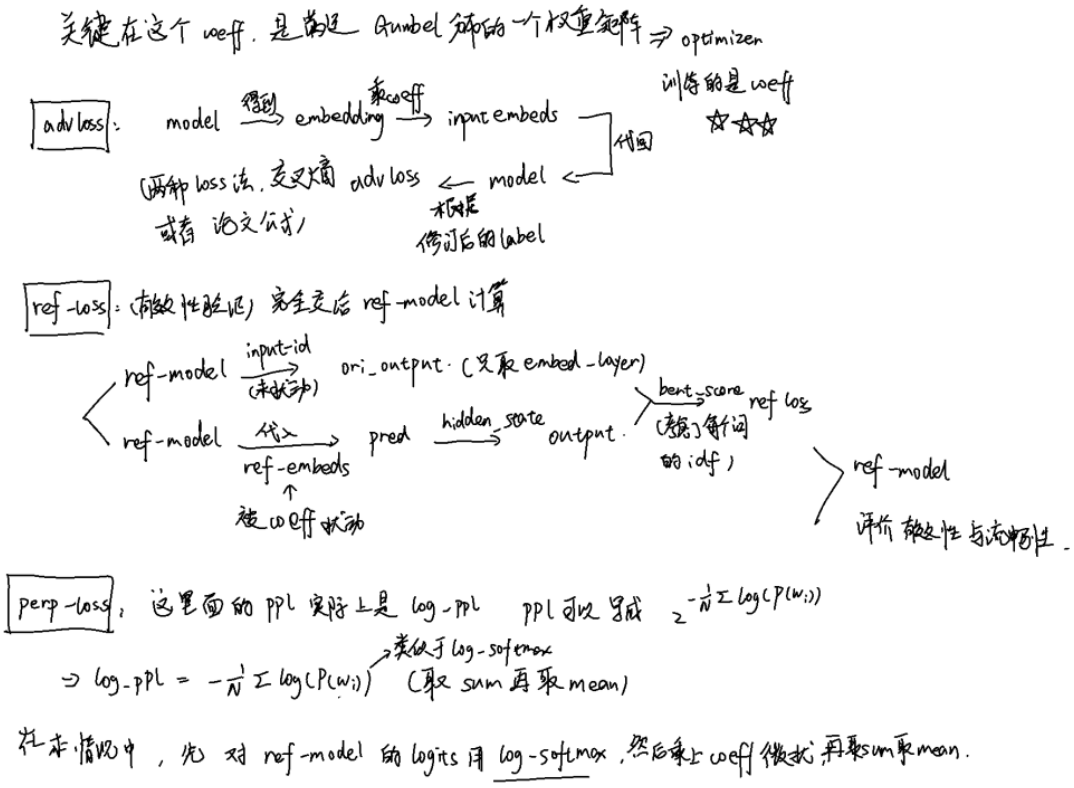
$$l_{margin}(\mathbf{x}, y; h) = max(\Phi_h(\mathbf{x})_y - max_{k \neq y}\Phi_h(\mathbf{x})_k + \kappa, 0)$$

    然后总体的对抗攻击损失是由分布决定的，我们希望取到这样的一个分布使得这个损失最小。其中$\mathbf{z}$是一个序列输入，$\mathbf{z} = z_1 \dots z_n$，**找到一个序列的分布，在这个分布下损失函数的期望取到最小值**

$$\min_{\Theta \in R^{n \times V}} E_{\mathbf{z} \sim P_\Theta} l(\mathbf{z}, y; h)$$

  - 其次是相似性的验证，也就是有效性的验证，主要是使用了bert-score方法。
    - 首先是得到一个数据集的逆文本频率idf，这个是一个weight
    - 然后将扰动前后的样本使用bert-score进行计算。

- 整体的思路如下：



PWWS攻击算法（Generating Natural Language Adversarial Examplesthrough Probability Weighted Word Saliency）

PWWS算法流程

1.词替代算法

- 首先我们需要明确替代过程的问题，选哪些词进行替代
- 对于那些词进行替代，我们使用wordnet词库。考虑的是同义词替代和命名实体替代的两种方法。
- 对于一句话$x = w_1 \dots w_i \dots w_n$，进行一个修改$x_i^{'} = w_1 \dots w_i^{'} \dots w_n$

  根据如下的挑选标准

  $w_i^* = argmax_{w_i^{'} \in L_i} \{P(y_{true}|x) - P(y_{true}|x_i^{'})\}$

  我们选取让y_{true}改变最大的改动词，其中L_i是备选的改动词集合
- 根据这个改动方法，我们挑选得到了对抗攻击样本$x_i^* = w_1 \dots w_i^* \dots w_n$
- 得到的最佳攻击效果为：$\Delta P_i^* = P(y_{true}|x) - P(y_{true}|x_i^*)$

2. 替换顺序策略

- 我们考虑每个位置的显著性S，即该位置的权重

  $S(\mathbf{x}, w_i) = P(y_{true}|x) - P(y_{true}|\hat{x}_i)$

  其中$x = w_1 \dots w_i \dots w_n$，进行一个修改$\hat{x}_i = w_1 \dots < unknown > \dots w_n$，相当于把单词mask掉，然后看真值label的改变。
- 根据这个S，我们最后得到一个评分函数。其中Φ是一个softmax函数
  $H(\mathbf{x}, \mathbf{x}_i^*, w_i) = \Phi(S(\mathbf{x}))_i \cdot \Delta P_i^*$
- 根据评分函数排序，得到词语的修改顺序。

Algorithm 1 PWWS Algorithm

Input: Sample text $\mathbf{x}^{(0)}$ before iteration;
Input: Length of sample text $\mathbf{x}^{(0)}$: $n = |\mathbf{x}^{(0)}|$;
Input: Classifier F;
Output: Adversarial example $\mathbf{x}^{(i)}$

for all $i = 1$ to n **do**
  Compute word saliency $S(\mathbf{x}^{(0)}, w_i)$
  Get a synonym set $\mathbb{L}_i$ for w_i
  if w_i is an NE **then** $\mathbb{L}_i = \mathbb{L}_i \cup \{\text{NE}_{adv}\}$
  end if 把新的name entity并进来
  if $\mathbb{L}_i = \varnothing$ **then** continue
  end if
  $w_i^* = R(w_i, \mathbb{L}_i)$;
end for
Reorder w_i such that
  $H(\mathbf{x}, \mathbf{x}_1^*, w_1) > \cdots > H(\mathbf{x}, \mathbf{x}_n^*, w_n)$
for all $i = 1$ to n **do**
  Replace w_i in $\mathbf{x}^{(i-1)}$ with w_i^* to craft $\mathbf{x}^{(i)}$
  if $F(\mathbf{x}^{(i)}) \neq F(\mathbf{x}^{(0)})$ **then** break
  end if
end for

具体算法如上。

TextFooler算法 [Is BERT Really Robust? A Strong Baseline for Natural Language Attack on Text Classification and Entailment]

目标

构建满足三个条件的对抗样本

- 首先对于人类来说是能够被正确分类
- 和原来的文本要语义上相似（有效性）
- 对于人类来说语法上要正确

公式化表示如下

- $F(X_{adv}) \neq F(X), and \quad Sim(X_{adv}, X) \geq \epsilon$
- 相似度肯定是越大越好

算法流程

1. 首先有一个计算词语权重的方法，这种方法与PWWS不同，是通过删除这句话的某个词来确定的，PWWS是通过改为标签来计算权重的。TextFooler的计算方法如下。

$$I_{w_i} = \begin{cases} F_Y(X) - F_Y(X_{\setminus w_i}), & \text{if } F(X) = F(X_{\setminus w_i}) = Y \\ (F_Y(X) - F_Y(X_{\setminus w_i})) + (F_{\bar{Y}}(X_{\setminus w_i}) - F_{\bar{Y}}(X)), & \\ \quad \text{if } F(X) = Y, F(X_{\setminus w_i}) = \bar{Y}, \text{ and } Y \neq \bar{Y}. & \end{cases}$$

   得到这个权重之后，构造一个输入句子X的词语集合W，W中的单词按照权重降序排列。

2. 与PWWS不同，PWWS是从WordNet里面挑选同义词作为备选库，TextFooler是从词汇表里面计算余弦相似度最大的N个词作为这个词的备选库。
3. 对于W中的每一个词进行二次筛选。
   - 第一次筛选是：如果这个候选词加入之后，句子已经修改，修改前后的两个句子之间的相似性如果满足一定要求，大于给定值，就把它加入**最后的候选名单**。
   - 第二次筛选的标准是：对于最后的候选名单中，首先寻找能够使得模型预测标签发生改变的样本。遵循以下的原则。如果有能够使得标签改变的，找相似度最高的；如果没有，就找使得ground_true概率改变程度最大的样本。

DeepWordBug（DeepWordBug）

评分机制

计算两个指标，TS和TTS，对于这两个指标取加权，得到某个位置的权重。具体公式如下：

$$TS(x_i) = F(x_1, \ldots, x_i) - F(x_1, \ldots, x_{i-1})$$

其中F是最后$output$的$logit$分数.

当然只看前面然后预测并不全面，需要看后面的单词。所以

$$TTS(x_i) = F(x_i, \ldots, x_n) - F(x_{i+1}, \ldots, x_n)$$

最后的分数为

$$score = \lambda \cdot TTS + TS$$

转换函数

(1)用一个随机字母替换单词中的一个随机字母，(2)从单词中删除一个随机字母，(3)在单词中插入一个随机字母，(4)在单词中交换两个相邻的字母。替换、删除和插入操作的编辑距离分别为1和2。转换过程相似性的判断使用编辑距离来衡量。

算法流程

Algorithm 1 DeepWordBug algorithm with the combined score

Input: Input sequence $\mathbf{x} = x_1 x_2 \dots x_n$, RNN classifier $F(\cdot)$, maximum allowed number of words changed m, hyperparameter λ.

for $i = 1..n$ **do**
  $S_{temporal}(i) = F(x_1 x_2 ... x_i) - F(x_1 x_2 ... x_{i-1})$
end for
for $i = n..1$ **do**
  $S_{tail}(i) = F(x_{i+1} x_{i+2} ... x_n) - F(x_i x_{i+1} ... x_n)$
end for
$S_{combined} = S_{temporal} + \lambda S_{tail}$
Sort $S_{combined}$ into an ordered index list: L_1 .. L_n
$\mathbf{x}' = \mathbf{x}$
for $i = 1..m$ **do**
  $x'_{L_i} = \text{Transform}(x'_{L_i})$
end for
Return $\mathbf{x}'$

Data augmentation approaches in natural language processing: A survey（数据增强）

数据增强方法分成三类

基于释义的方法

- 使用同义词替换的方法去生成新样本，同义词替换可以是在词语层面的，也可以是在embedding层面的。
- 在句子级别上，可以使用反向翻译的方法，就是翻译过去再翻译回来，得到新样本
- 使用seq2seq的模型直接生成原话的释义

基于噪声的方法

- 就是一些字符级别的扰动，缺点在于解释性较差

基于采样的方法

- rule-base：基于一些特定的规则，比如对于语法树进行操作使得句子的表达方式转化
- Non-pretrained models： seq2seq的模型生成新样本，去训练这个模型
- Pretrained models:使用大模型来生成，缺点在于需要数据量比较多
- Self-training： 自监督，其实是使用数据蒸馏，就是用大模型对于无标注数据（新的数据）进行标注，然后小模型去学习。
- mix-up:本质上是在做一个插值，比如两个词，embedding是x_i, x_j,label是y_i, y_j，我取一个λ对于他们进行插值，就可以得到新的embedding和新的label，然后作为一个新样本进行训练。缺点在于可解释性。

Black-Box Tuning for Language-Model-as-a-Service 使用黑盒优化的方法，在不知道模型梯度的情况下优化prompt

背景

1. 获得模型的参数信息，梯度信息很困难，希望在用户端就能够微调prompt增强模型在特定任务上面的性能。
2. prompt的搜索空间庞大，如何去优化prompt是一个问题

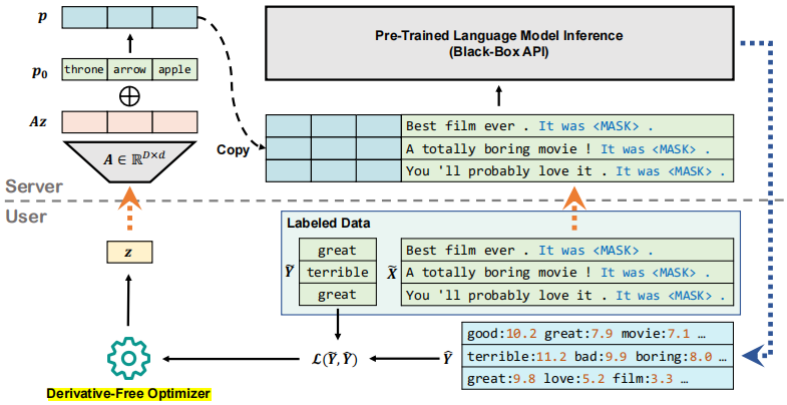
方法

1. 采用**不基于梯度的优化方法**、本文中用户只能获得模型的输出结果，根据这些输出结果，采用DFO算法（Derivative-Free）：比如有贝叶斯优化，进化算法等。
2. 映射到小搜索空间。由于prompt的搜索空间庞大，前人已经证明，在小的搜索空间进行优化已经足够（论文中有提及）。所以本文采用随机矩阵将prompt限制在d维空间，然后用$A \in R^{D \times d}$将其映射回去。

具体的做法是，先优化一个偏移量，然后乘上这个随机矩阵A，把这个偏移量加到原来的模板上面。当然，下面的初始化prompt也可由预训练得到。

$$z^* = \arg\min_{z \in Z} L(f(Az + p_0; \hat{X}), \hat{Y})$$

3. 最后得到的pipeline如图



4. 注意，在这个方法中，优化的是continuous prompt。

Calibrate Before Use:Improving Few-Shot Performance of Language Models 文本校准方法，消除模型在学习中学到的偏置。

背景

1. 文章identify three weakness of model, they are mainly the bias that model learned from training.
    - majority label bias: 模型会更加倾向于预测出样本较多的label。所以说数据的分布对于模型的性能比较重要
    - recency bias：模型会记住他最近学到的东西，就是更加倾向于预测出位于训练后期的样本。比如PPPN四个数据，模型倾向于预测N。这个现象非常严重。
    - common token bias：倾向于预测那些经常出现的词，就是生成一些safe but meaningless的东西。
2. 文本校准方法

    文本校准方法是对于模型的学习到的偏好进行修正，**修正是在label层面进行的。**就是对于label，我们把新的label设置为$\hat{q}$，则有

    $$\hat{q} = softmax(W\hat{p} + b)$$

    W,b是我们希望训练得到的参数。

3. 具体实现

    因为这个方法要训练W和b，就需要很多数据。文章提出了一个不需要文本的方法，只是**基于词义**对于W和b进行学习。

    比如我输入一个空字符，如果模型认为这个空字符是积极的概率为0.6，我就让他回归到0.5。一个很naive的W矩阵是

    $$W = diag(\hat{p}_{cf})^{-1}$$

    就是直接取导数让他回归到0.5，当然，文章中不仅对于空字符串进行训练，还对于多种meaningless的单词进行训练，取平均得到W和b。（事实上把b设置为0）

BBTv2: Towards a Gradient-Free Future with Large Language Models

背景

本文是BBT的第二版本，对于black-box tuning进行了优化。在black-box tuning中，需要使用预训练好的模板才能达到很好的效果。在训练数据上，BBT方法存在着过拟合现象。

改进举措

在BBT中，使用的是随机投影矩阵A进行映射，而BBTv2是使用了基于模型特定的正态分布构造参数矩阵A，同时BBTv2将在优化每一层的prompt，**给每一层都构建了一个prompt**，对于每一层的prompt，都使用一个**非梯度优化器进行优化**，通过使用多样化的prompt来进一步提升效果。

对于给定的一个优化次数B,模型的层数为L，整体上进行$\frac{B}{L}$次优化操作。

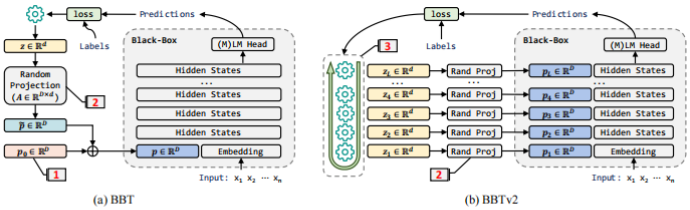


Figure 5: An illustration of (a) BBT (Sun et al., 2022b) and (b) BBTv2. ⚙ is some derivative-free optimizer such as CMA-ES. Compared with BBT, BBTv2 has 3 differences: (1) BBT requires pre-trained prompt embedding $\mathbf{p}_0$ to match the performance of model tuning on entailment tasks, and therefore does not completely get rid of gradients. In contrast, BBTv2 requires no prompt pre-training. (2) BBT generates the random projection using a uniform distribution while BBTv2 adopts model-specific normal distributions. (3) Instead of optimizing the prompt merely in the input layer, BBTv2 uses a divide-and-conquer algorithm to alternately optimize prompt at each layer.

Prototypical Verbalizer for Prompt-based Few-shot Tuning

背景

1. 原型学习

   原型学习是在分类任务中是通过学习特定的label的嵌入表示，比如，对于情感分类，我能够学习到积极情感的嵌入表示，这种表示就相当于这类情感的中心点（中心向量）。

   这种中心向量的学习是基于一个支持集，对于支持集里面特定label进行聚类，然后相同的类里面的词、句的embedding取平均得到原型向量。

   最后在evaluation中，计算查询向量到不同类的原型向量的距离，经过softmax得到输出概率

   原型学习在小样本中非常好，因为我们可以使用小样本的支持集构建原型向量，同时我们也可以通过先验知识，更高层次地构建原型向量（即可以通过学习，也可以通过rule-base）

2. 基于原型向量的verbalizer

   不同于以往使用词作为verbalizer，这里使用的是向量作为verbalizer，就是把模型对于mask位置的输出与各个原型向量进行距离计算，得到最后的label。

实现方法

给模型的最后一层后面加上一个线性层W,其中h是mask位置最后一层隐藏层的输出值，v就是得到的文本的表示。

$$v = E_{\Phi}(x) = Wh_{[mask]}$$

在具体训练中，我们是去训练W，使得模型能够在mask位置进行映射后能够对于文本有很好的表示。

训练中用到两种损失，一种是查询向量之间的损失、另一种是原型向量和查询向量的损失，直观理解，相同类别查询向量之间的相似度应该更高；对于特定类的查询向量，他到该类的原型向量的距离比到其他类的原型向量的距离应该更近一些。

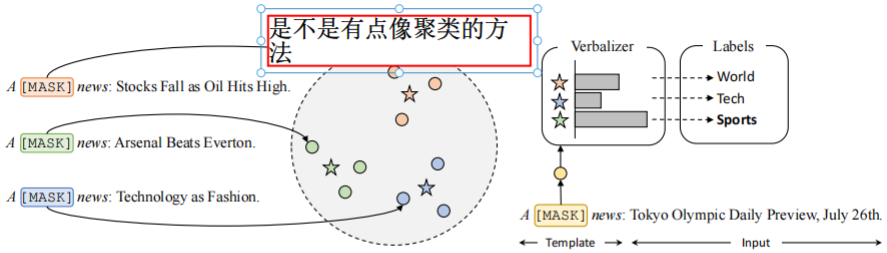


Figure 2: Illustration of ProtoVerb. Left: We project the hidden states of [MASK] tokens to the embedding space and learn prototypes. Right: The learned prototypes constitute the verbalizer and map the PLM outputs to corresponding labels.