作业 5　　自注意力 Transformer 预训练

一、简述

1. 数学探索

自注意力容易实现哪些类型的操作？为什么需要使用多头自注意力？本部分将使用一些数学以说明自注意力和 Transformer 网络的几点动机。

2. 研究型基础代码扩展

通过预训练教 NLP 模型关于世界的事实；通过微调评估这些知识。训练一个 Transformer 模型，使其在没有提供任何可以提取答案的输入文本的情况下尝试回答如“Where was person [x] born?”之类的简单问题。可以发现模型能够通过预训练学到一些关于人们出生地的知识，并在微调过程中访问出生地信息以回答相关问题。

二、探索自注意力（20 分）

本部分关注自注意力方程以及为何多头注意力相对单头注意力更受青睐。

q, {v1,…,vn}, {k1,…,kn}，d 维向量

$$c = \sum_{i=1}^{n} v_i \alpha_i \tag{1}$$

$$\alpha_i = \frac{\exp(k_i^\top q)}{\sum_{j=1}^{n} \exp(k_j^\top q)} \tag{2}$$

“注意力权重”alpha={a1,…an}

输出 c 是值向量关于 alpha 的加权平均，d 维向量

(a) (5 分) 注意力中的拷贝 注意力的优势之一是相对容易将一个值向量拷贝给输出 c。

i) 解释为何 alpha 能理解为一个类别概率的分布？

My answer:

如果将查询 q 视为待分类的样本，将键{k1,k2,…kn}视为 n 类，则 alpha 可以理解为根据相似度对查询分类的概率；经过 softmax 运算得到的 alpha 各维分量非负且和为 1。

ii) 用一句话描述在何种情况下分布 alpha 将几乎所有权重放在某一个 alpha_j 上；为此查询 q 和/或 键{k1,…,kn}应该满足什么条件？

My answer:

$\boldsymbol{q}^T \boldsymbol{k}_j \gg \boldsymbol{q}^T \boldsymbol{k}_i,\ \forall i \neq j,\ i,j \in \{1,...,n\}$

iii) 基于 ii)中的情况，描述输出 c

My answer:

$\boldsymbol{c} \approx \boldsymbol{v}_j$

iv) 用不多于两句话直观解释 ii)和 iii)中的答案意味着什么？

My answer:
某一个键向量由于与查询向量非常相似而近似等于注意力加权的输出。

(b) (7 分) 两项平均
Transformer 模型希望聚合来自多个源向量信息。考虑源向量 va 和 vb，对应键向量 ka 和 kb。

i) 将两个向量 va 和 vb 结合成一个输出向量 c 的一种方式：$c = \frac{1}{2}(v_a + v_b)$

从结果向量 c 中提取出原始向量 va 和 vb 的信息似乎是困难的，但在特定情况下能够做到。假定 va 或 vb 未知，可以确定 va 存在于由 m 个基向量{a1,…,am}形成的子空间 A 中，vb 存在于由 p 个基向量{b1,…,bp}形成的子空间 B 中（每个基向量长度为 1 且相互正交）。另外，假定两个子空间正交。使用基向量{a1,…,am}构造一个矩阵 M，使得对 A，B 中的任意向量 va 和 vb，可以用 M 从和向量 s=va+vb 中提取 va，换言之，$Ms = v_a$。

My answer:

$$\boldsymbol{v}_a = [\boldsymbol{a}_1, \boldsymbol{a}_2, ..., \boldsymbol{a}_m]\begin{bmatrix} k_1 \\ k_2 \\ \vdots \\ k_m \end{bmatrix} = \boldsymbol{Ak}, \quad \boldsymbol{v}_b = [\boldsymbol{b}_1, \boldsymbol{b}_2, ..., \boldsymbol{b}_p]\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_p \end{bmatrix} = \boldsymbol{Bw}$$

$$\boldsymbol{M}(\boldsymbol{Ak} + \boldsymbol{Bw}) = \boldsymbol{Ak}$$

$$\boldsymbol{A}^T = \begin{bmatrix} \boldsymbol{a}_1^T \\ \boldsymbol{a}_2^T \\ \vdots \\ \boldsymbol{a}_m^T \end{bmatrix}, \quad \boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{E}, \quad \boldsymbol{A}^T\boldsymbol{B} = 0$$

$$\boldsymbol{M} = \boldsymbol{A}\boldsymbol{A}^T, \quad \boldsymbol{Ms} = \boldsymbol{M}(\boldsymbol{Ak} + \boldsymbol{Bw}) = \boldsymbol{Ak} = \boldsymbol{v}_a$$

ii) 假定(1)所有键向量相互正交；(2)所有键向量长度为 1。找到一个查询向量 q 的表达式使得$c \approx \frac{1}{2}(v_a + v_b)$，证明结论

$$\boldsymbol{q} = \frac{C}{2}(\boldsymbol{k}_a + \boldsymbol{k}_b),\ C\text{是一个较大的正数}$$

$$\boldsymbol{c} = \sum_{i=1}^{n} \alpha_i \boldsymbol{v}_i$$

$$= \sum_{i=1}^{n} \frac{\exp(\boldsymbol{k}_i^T \boldsymbol{q})}{\sum_{j=1}^{n} \exp(\boldsymbol{k}_i^T \boldsymbol{q})} \boldsymbol{v}_i$$

$$\because \boldsymbol{k}_a^T \boldsymbol{q} = \boldsymbol{k}_b^T \boldsymbol{q} = \frac{C}{2},\ \boldsymbol{k}_o^T \boldsymbol{q} = 0, o \neq a, b, \exp\left(\frac{C}{2}\right) \gg n$$

$$\therefore \boldsymbol{c} = \frac{\exp\left(\frac{C}{2}\right)}{2\exp\left(\frac{C}{2}\right) + (n-2)} \boldsymbol{v}_a + \frac{\exp\left(\frac{C}{2}\right)}{2\exp\left(\frac{C}{2}\right) + (n-2)} \boldsymbol{v}_b + \sum_{o \neq a,b} \frac{1}{2\exp\left(\frac{C}{2}\right) + (n-2)} \boldsymbol{v}_o$$

$$\approx \frac{1}{2}(\boldsymbol{v}_a + \boldsymbol{v}_b)$$

(c) (5 分) 单头注意力的缺陷
上一部分可以看到单头注意力可以平均地关注两个值向量。相同的概念容易扩展到任意值向

量的子集。本问题中将看到为什么这不是一个可行的解决方案。

给定键向量集{k1,…,kn}，ki 通过随机采样得到且服从正态分布$N(\mu_i,\ \Sigma_i)$，假定均值$\mu_i \in R^d$已知，协方差矩阵Σ_i未知，且不同均值向量相互正交，每个均值向量长度为 1。

i) （2 分）假定协方差矩阵$\Sigma_i = \alpha I,\ \forall i \in \{1,2,...,n\}$，$\alpha$是极小的实数。设计一个关于$\mu_i$的查询向量 q，使得$c \approx \frac{1}{2}(v_a + v_b)$，并简述原因。

My answer:

$\boldsymbol{q} = \frac{\lambda}{2}(\boldsymbol{\mu}_a + \boldsymbol{\mu}_b),\ \lambda$是较大的正实数

$E[\boldsymbol{q}^T \boldsymbol{k}_a] = \boldsymbol{q}^T \boldsymbol{\mu}_a = \frac{\lambda}{2},\ E[\boldsymbol{q}^T \boldsymbol{k}_b] = \boldsymbol{q}^T \boldsymbol{\mu}_b = \frac{\lambda}{2},\ E[\boldsymbol{q}^T \boldsymbol{k}_o] = \boldsymbol{q}^T \boldsymbol{\mu}_o = 0 (o \neq a,b)$

$Var[\boldsymbol{q}^T \boldsymbol{k}_i] = \frac{\lambda}{2}(\boldsymbol{\mu}_a + \boldsymbol{\mu}_b)^T \cdot \alpha 1 \approx 0,\ 1 \in R^d$

$\boldsymbol{q}^T \boldsymbol{k}_a \approx \boldsymbol{q}^T \boldsymbol{k}_b \approx \frac{\lambda}{2},\ \boldsymbol{q}^T \boldsymbol{k}_o \approx 0$

$\boldsymbol{c} \approx \frac{1}{2}(\boldsymbol{v}_a + \boldsymbol{v}_b)$

Draft:

$$E[\boldsymbol{k}_i^T \boldsymbol{k}_j] = \boldsymbol{\mu}_i^T \boldsymbol{\mu}_j = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

$$Var(\boldsymbol{k}_i^T \boldsymbol{k}_j) = E[(\boldsymbol{k}_i^T \boldsymbol{k}_j)^2] - (E[\boldsymbol{k}_i^T \boldsymbol{k}_j])^2$$

$$= \sum_p \sum_q E[k_{ip} k_{iq} k_{jp} k_{jq}] - \left(\sum_r \mu_{ir} \mu_{jr}\right)^2$$

$$E[k_{ip} k_{iq} k_{jp} k_{jq}] = \begin{cases} (\mu_{ip}{}^2 + \alpha)(\mu_{jp}{}^2 + \alpha) & p = q \\ \mu_{ip} \mu_{iq} \mu_{jp} \mu_{jq} & p \neq q \end{cases}$$

$$Var(\boldsymbol{k}_i^T \boldsymbol{k}_j) = \sum_p (\mu_{ip}{}^2 + \alpha)(\mu_{jp}{}^2 + \alpha) - \sum_r (\mu_{ir} \mu_{jr})^2$$

$$= \alpha \sum_p (\mu_{ip}{}^2 + \mu_{jp}{}^2) + d\alpha^2$$

$$\approx 0$$

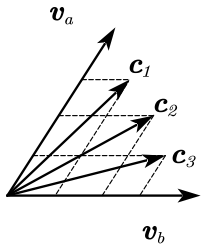
$\boldsymbol{q}^T \boldsymbol{k}_a \approx \boldsymbol{q}^T \boldsymbol{k}_b \approx \frac{\lambda}{2}$

$\boldsymbol{c} \approx \frac{1}{2}(\boldsymbol{v}_a + \boldsymbol{v}_b)$

ii) （3 分）尽管单头注意力对键向量中的小扰动具有一定抵抗力，但某些较大的扰动可能会

带来更大的问题。具体来说，在某些情况下，一个键向量 ka 的范数可能比其他键的范数更大或更小，但仍然指向与 μa 相同的方向。作为一个例子，考虑一个关于项 a 的协方差 $\Sigma_a = \alpha I + \frac{1}{2}(\mu_a \mu_a^T)$，$\alpha$ 是极小的实数，如下图所示。这导致 ka 指向几乎和 μa 相同的方向，但在幅值上具有较大的方差。进一步，对所有不等于 a 的 i，令 $\Sigma_i = \alpha I$。多次采样{k1,k2,...,kn}，使用在第一部分定义的 q 向量，定性估计向量 c 在不同样本上的形状；考虑此处的 c 与第一部分的区别，以及 c 的方差将如何受到影响。

My answer:
当 ka 的模长较大时，c 将更靠近 va，反之更靠近 vb；与第一部分相比，输出向量 c 的变化范围明显增大，方差将增大。



$$\boldsymbol{q} = \frac{\lambda}{2}(\boldsymbol{\mu}_a + \boldsymbol{\mu}_b)$$

$$Var(\boldsymbol{q}^T \boldsymbol{k}_i) = \frac{\lambda}{2}(\boldsymbol{\mu}_a + \boldsymbol{\mu}_b)^T [Var(k_{i1});\ Var(k_{i2});...;Var(k_{id})]$$

$$\because \Sigma_a = \alpha \boldsymbol{I} + \frac{1}{2}\boldsymbol{\mu}_a \boldsymbol{\mu}_a^T,\ \Sigma_i = \alpha \boldsymbol{I},\ i \neq a$$

$$\therefore Var(\boldsymbol{q}^T \boldsymbol{k}_i) = \begin{cases} \frac{\lambda}{2}\sum_{j=1}^{d}(\mu_{aj} + \mu_{bj})\left(\frac{\mu_{aj}^2}{2} + \alpha\right) \approx \frac{\lambda}{4}\sum_{j=1}^{d}(\mu_{aj} + \mu_{bj})\mu_{aj}^2, & i = a \\ \frac{\lambda\alpha}{2}\sum_{j=1}^{d}(\mu_{aj} + \mu_{bj}) \approx 0, & i \neq a \end{cases}$$

(d) (3 分) 多头注意力的优势
考虑两个查询向量 q1 和 q2，基于单头注意力可以得到输出向量 c1 和 c2，多头注意力的最后输出是二者的平均，即 $\frac{1}{2}(\boldsymbol{c}_1 + \boldsymbol{c}_2)$。和问题 1 的(c)部分一样，考虑一个键向量集合{k1,...,kn}, ki 通过随机采样得到且服从正态分布 $N(\mu_i,\ \Sigma_i)$，均值 $\mu_i \in R^d$ 已知，协方差矩阵 Σ_i 未知，不同均值向量相互正交，每个均值向量长度为 1。

i) （1 分）假定协方差矩阵 $\Sigma_i = \alpha I,\ \forall i \in \{1, 2, ..., n\}$，$\alpha$ 是极小的实数，设计 q1 和 q2 使得 $c \approx \frac{1}{2}(v_a + v_b)$，注意 q1 和 q2 应该具有不同的表达式。

My answer:

$\boldsymbol{q}_1=\lambda\boldsymbol{\mu}_a$, $\boldsymbol{q}_2=\lambda\boldsymbol{\mu}_b$, λ是较大的正实数

$$E(\boldsymbol{q}_1^T\boldsymbol{k}_i)=\begin{cases}\lambda & (i=a)\\ 0 & (i\neq a)\end{cases},\ E(\boldsymbol{q}_2^T\boldsymbol{k}_i)=\begin{cases}\lambda & (i=b)\\ 0 & (i\neq b)\end{cases}$$

$$Var(\boldsymbol{q}_1^T\boldsymbol{k}_i)=\lambda\alpha\sum_{j=1}^{d}\mu_{aj}\approx 0,\ Var(\boldsymbol{q}_2^T\boldsymbol{k}_i)=\lambda\alpha\sum_{j=1}^{d}\mu_{bj}\approx 0$$

$$\boldsymbol{c}=\frac{1}{2}(\boldsymbol{c}_1+\boldsymbol{c}_2)\approx\frac{1}{2}(\boldsymbol{v}_a+\boldsymbol{v}_b)$$

ii) （2 分）假定协方差矩阵$\Sigma_a=\alpha I+\frac{1}{2}(\mu_a\mu_a^T)$，$\alpha$是极小的实数，对所有不等于 a 的 i，令$\Sigma_i=\alpha I$，考虑 i)中设计的查询向量 q1 和 q2，定性估计向量 c 在不同样本上的形状；用 c1 和 c2 的方差简述理由。可以忽略$\boldsymbol{k}_a^T\boldsymbol{q}_i<0$的情况。

My answer:

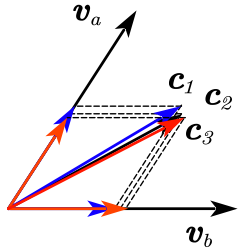
输出向量$\boldsymbol{c}$更接近$\frac{1}{2}(\boldsymbol{v}_a+\boldsymbol{v}_b)$；

$\boldsymbol{q}_1=\lambda\boldsymbol{\mu}_a$, $\boldsymbol{q}_2=\lambda\boldsymbol{\mu}_b$, λ是较大的正实数

$$Var(\boldsymbol{q}_1^T\boldsymbol{k}_i)=\begin{cases}\lambda\alpha\sum_{j=1}^{d}\mu_{aj}\approx 0 & (i\neq a)\\ \lambda\sum_{j=1}^{d}\mu_{aj}\left(\frac{1}{2}\mu_{aj}^2+\alpha\right)\approx\frac{\lambda}{2}\sum_{j=1}^{d}\mu_{aj}^3 & (i=a)\end{cases}$$

$$Var(\boldsymbol{q}_2^T\boldsymbol{k}_i)=\begin{cases}\lambda\alpha\sum_{j=1}^{d}\mu_{bj}\approx 0 & (i\neq a)\\ \lambda\sum_{j=1}^{d}\mu_{bj}\left(\frac{1}{2}\mu_{aj}^2+\alpha\right)\approx\frac{\lambda}{2}\sum_{j=1}^{d}\mu_{aj}^2\mu_{bj} & (i=a)\end{cases}$$

在单头注意力下，$\boldsymbol{c}$的方差来源于$\boldsymbol{v}_a$；而在多头注意力下，$\boldsymbol{c}$的方差来源于$\boldsymbol{v}_a$和$\boldsymbol{v}_b$（$\boldsymbol{c}_1$和$\boldsymbol{c}_2$的方差均小于$\boldsymbol{c}$的方差），此时$\boldsymbol{c}$将更接近$\frac{1}{2}(\boldsymbol{v}_a+\boldsymbol{v}_b)$。



三、预训练 Transformer 模型及知识访问（35 分）

本部分将训练 Transformer 模型以执行一个包含访问世界知识的任务——这些知识并未由任

务的训练数据提供（至少当想要泛化到训练集外）。你将发现模型在这个任务上或多或少地完全失败。由此你将学习如何在包含世界知识的 Wikipedia 文本上预训练 Transformer，并发现在相同的知识密集型任务上微调 Transformer 能够使模型访问在预训练时学到的知识。你将发现这使得模型在测试集上表现出远超随机猜测的水平。

提供的代码基于 Andrej Karpathy 的 minGPT。该代码相对简单且透明，优于绝大多数研究型代码。minGPT 中的 GPT 指的是 OpenAI 的 Transformer 语言模型。

和之前作业相同，需要先在本地开发代码，然后移植到服务器上运行。可以使用和之前作业相同的 conda 环境用于本地开发，在 GPU 上训练的过程同理。大致需要 5 小时训练。在没有 GPU 的本地机器上多进程读取数据集可能失效，因此在本地 debug 时需要将 num_workers 设置为 0。

工作步骤如下：
(a) 检查 demo

在 mingpt-demo/ 文件夹中，有一个名为 play_char.ipynb 的 Jupyter notebook，用于训练和从 Transformer 语言模型中进行采样。请查看该 notebook（本地查看），以便对它如何定义和训练模型有一定了解。您下面要编写的一些代码将受到此笔记本中所见内容的启发。请注意，您不需要为这部分编写任何代码或提交书面答案。

Finished

(b) 阅读 src/dataset.py 中的 NameDataset，这是用于读取姓名-出生年份对的数据集

将使用预训练模型完成的任务是尝试获取知名人物在其维基百科页面中所写的出生地信息。可以将这视为一种非常简单的问答形式：

Q: *Where was [person] born?*
A: *[place]*

从现在开始，您将开始开发 src/文件夹中的代码。**在本次作业中 mingpt-demo/文件夹中的代码不会被更改或评估。**在 dataset.py 文件中，您会找到 NameDataset 类，它用于读取包含姓名/地点对的 TSV 文件并生成上述形式的示例，可以将这些示例提供给 Transformer 模型。

为了对即将处理的示例有所了解，如果运行”python src/dataset.py namedata”，它将在训练集 birth_places_train.tsv 上加载 NameDataset，并打印出一些示例。

```
(base) PS F:\CV\NLP\coding\a5\student_2023> python .\src\dataset.py namedata
data has 418352 characters, 256 unique.
x: Where was Khatchig Mouradian born?⁇Lebanon⁇□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
y: □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□⁇Lebanon⁇□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
x: Where was Jacob Henry Studer born?⁇Columbus⁇□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
y: □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□⁇Columbus⁇□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
x: Where was John Stephen born?⁇Glasgow⁇□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
y: □□□□□□□□□□□□□□□□□□□□□□□□□□□□⁇Glasgow⁇□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
x: Where was Georgina Willis born?⁇Australia⁇□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
y: □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□⁇Australia⁇□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
```

请注意，您不需要为这部分编写任何代码或提交书面答案。

(c) （0 分）实现微调（无预训练）

请查看 run.py 文件。其中包含了一些指定标志的骨架代码，这些标志最终需要作为命令行参数处理。特别地，您可能会希望使用此代码进行预训练、微调或评估模型。目前，我们将关注没有预训练的微调。受 play_char.ipynb 文件中训练代码的启发，编写代码在姓名/出生地数据集上对 Transformer 模型进行微调，通过使用 NameDataset 类中的示例。目前，只实现没有预训练的情况（即从头开始创建一个模型并在第（b）部分的出生地预测任务上进行训练）。您需要修改代码中标记为 [part c] 的两个部分：一个用于初始化模型，一个用于微调模型。请注意，目前只需要在标有“vanilla”的情况下初始化模型（稍后在第（g）部分，我们将探讨模型的变种）。使用在 run.py 代码中指定的 Trainer 的超参数。

另外查看已经为您实现的评估代码。它从经过训练的模型中获取预测结果并调用 evaluate_places() 以获取正确地预测地点的总百分比。在第（d）部分中，您将运行此代码来评估已经训练好的模型。

这是后续部分的中间步骤，包括第（d）部分，其中包含您可以运行以检查实现的命令。此部分不需要书面答案。

(d) 预测（无预训练）

在 birth_places_train.tsv 上训练模型，在 birth_dev.tsv 上评估。特别地，您应该能够运行以下三条命令：

 # Train on the names dataset
python src/run.py **finetune** vanilla wiki.txt **--writing_params_path** vanilla.model.params **--finetune_corpus_path** birth_places_train.tsv

 # Evaluate on the dev set, writing out predictions
python src/run.py **evaluate** vanilla wiki.txt **--reading_params_path** vanilla.model.params **--eval_corpus_path** birth_dev.tsv **--outputs_path** vanilla.nopretrain.dev.predictions

 # Evaluate on the test set, writing out predictions
python src/run.py **evaluate** vanilla wiki.txt **--reading_params_path** vanilla.model.params **--eval_corpus_path** birth_test_inputs.tsv **--outputs_path** vanilla.nopretrain.test.predictions

在服务器上的训练时长少于 10 分钟。汇报模型在 dev 数据集上的准确率（上面第二条命令的打印结果）。和作业 4 类似，在作业 5 中也有用于 debug 的 Tensorboard 日志。Tensorboard 用 tensorboard --logdir expt/命令启动。如果准确率低于 10%，请不要惊讶，我们将在第 3 部分深入探讨。作为参考，我们希望计算当模型为 dev 数据集上的每一个人预测出生地均为”London”时所能达到的准确率。请填充 london_baseline.py 以计算这种方法的准确率并在书面报告中呈现。您应该充分利用现有代码使得该文件的长度仅有几行。

python src/run.py **finetune** vanilla wiki.txt **--writing_params_path** vanilla.model.params **--finetune_corpus_path** birth_places_train.tsv

epoch 75 iter 7: train loss **0.18628**. lr 5.100024e-04

python src/run.py **evaluate** vanilla wiki.txt **--reading_params_path** vanilla.model.params **--eval_corpus_path** birth_dev.tsv **--outputs_path** vanilla.nopretrain.dev.predictions

data has 418352 characters, 256 unique.
number of parameters: 3323392
500it [00:54, 9.21it/s]
Correct: 6.0 out of 500.0: **1.2%**

python src/run.py **evaluate** vanilla wiki.txt **--reading_params_path** vanilla.model.params **--eval_corpus_path** birth_test_inputs.tsv **--outputs_path** vanilla.nopretrain.test.predictions

data has 418352 characters, 256 unique.
number of parameters: 3323392
437it [00:43, 10.16it/s]
No gold birth places provided; returning (0,0)
Predictions written to vanilla.nopretrain.test.predictions; no targets provided

python src/london_baseline.py **--eval_corpus_path** birth_dev.tsv

500it [00:00, 1235799.65it/s]
Correct: 25.0 out of 500.0: **5.0%**

(e) （10 分）定义一个用于预训练的 span corruption 函数

在文件 src/dataset.py 中实现数据集类 CharCorruptionDataset 的__getitem__()函数。请遵照 dataset.py 中注释提供的指示。Span corruption 在 T5 的论文中被研究过。它会随机选择文档中的文本片段，并用唯一的标记进行替换（进行数据扰动）。模型获得加噪文本，并需要输出每一个唯一标记及其对应的原始词的模式。在这个问题中，您将实现一个仅掩盖单个句子的字符的简单情况。

这个问题将通过自动评分器进行评分，评分依据是你的文本片段损坏函数是否实现了我们规范中的一些基本属性。我们将使用我们自己的数据实例化 CharCorruptionDataset，并从中获取示例。

为了帮助您 debug，如果运行以下代码，代码将基于您的 CharCorruptionDataset 从预训练数据集 wiki.txt 中采样一些例子并打印出来。

python src/dataset.py charcorruption

本部分不需要手写答案。

```
data has 418352 characters, 256 unique.
x: Kha??chig M??t??□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
y: ha??chig M??t??□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
x: Jacob Henry Studer. Jacob Henry Studer (26 Fe?? Ohio - 2 August 1904 New York Ci??bruary 1840 Columbus,??□□□□□□□□□□□□□□□□□□□□
y: acob Henry Studer. Jacob Henry Studer (26 Fe?? Ohio - 2 August 1904 New York Ci??bruary 1840 Columbus,??□□□□□□□□□□□□□□□□□□□□
x: ??. Born in Glasgow, Stephen became a welder's??John Stephen??□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
y: . Born in Glasgow, Stephen became a welder's??John Stephen??□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
x: Georgina Willis.??a Willis is an award winning film director w?? Georgin??□□□□□□□□□□□□□□□□□□□□□□□□□□□□
y: eorgina Willis.??a Willis is an award winning film director w?? Georgin??□□□□□□□□□□□□□□□□□□□□□□□□□□□□
```

(f) （10 分）预训练，微调，预测（预计 2 小时）

补充 run.py 中的 *pretrain* 部分，这部分将在文段破坏任务上预训练一个模型。另外，修改 *finetune* 部分以处理带预训练的微调的情况。尤其当命令行提供了预训练模型路径时请在微调出生地预测任务的模型之前先加载模型。在 wiki.txt 上预训练模型（预计花费约 2h），在 NameDataset 上微调模型并进行评估。特别地，您应该能够运行以下四条命令：（当损失在预训练中途突然攀升时请不要担忧，损失最终会下降）。

 # Pretrain the model

python src/run.py **pretrain** vanilla wiki.txt **--writing_params_path** vanilla.pretrain.params

 # Finetune the model

python src/run.py **finetune** vanilla wiki.txt **--reading_params_path** vanilla.pretrain.params **--writing_params_path** vanilla.finetune.params **--finetune_corpus_path** birth_places_train.tsv

 # Evaluate on the dev set; write to disk

python src/run.py **evaluate** vanilla wiki.txt **--reading_params_path** vanilla.finetune.params **--eval_corpus_path** birth_dev.tsv **--outputs_path** vanilla.pretrain.dev.predictions

 # Evaluate on the test set; write to disk

python src/run.py **evaluate** vanilla wiki.txt **--reading_params_path** vanilla.finetune.params **--eval_corpus_path** birth_test_inputs.tsv **--outputs_path** vanilla.pretrain.test.predictions

汇报模型在 dev 数据集上的准确率（上面第三条命令的打印结果）。预期在 dev 数据集上的准确率至少为 10%，并且预期在 test 数据集上达到相似的准确率。

pretrain:

epoch 650 iter 22: train loss 0.56488. lr 7.182453e-04

finetune:

epoch 10 iter 7: train loss 0.15702. lr 5.983263e-04

evaluate on dev:

data has 418352 characters, 256 unique.

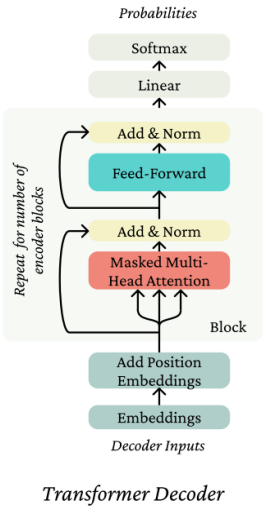
number of parameters: 3323392

500it [00:49, 10.10it/s]

Correct: 107.0 out of 500.0: **21.4%**

(g) （10 分）研究！完成并尝试一种更高效的注意力变种（预留 2h 的预训练时间）

我们将要改变 Transformer 架构——尤其是第一个和最后一个 Transformer 块。Transformer 模型使用一个基于点积的自注意力评分函数，这引入了相对密集的计算（计算复杂度为 $O(l^2)$，l 为句子长度）。这是由于每次需要计算l^2对词向量的内积。如果能够减少传递给自注意力模块的句子长度，就能够显著减少计算量。例如，将句子长度减少到一半，就能节约大约 75%的计算时间。PerceiverAR 提出了一种解决方案：通过减少中间层自注意力模型输入的序列长度，使模型更加高效。在第一层，输入序列被投影到更低维的基向量上。随后，所有的自注意力层在这个更小的子空间上运算。最后一层将输出投影回初始的序列长度。在本次作业中，我们提出了 PerceiverAR transformer 模型的一个更简单的版本。



Transformer Decoder

图 2 transformer 块示意

提供的 CausalSelfAttention 层为多头注意力的每一个头部实现了以下的注意力：令$X \in R^{l \times d}$（l 是块的大小，d 是完整的维度，d/h 是每一个头部的维度）。令Q_i, K_i, $V_i \in R^{d \times d/h}$。自注意力头部的输出是：

$$Y_i = soft\max\left(\frac{(XQ_i)\,(XK_i)^T}{\sqrt{d/h}}\right)(XV_i)$$

其中$Y_i \in R^{l \times d/h}$。然后自注意力的输出是头部向量拼接后的线性变换：

$$Y = [Y_1; ...; Y_h]\,A$$

其中$A \in R^{d \times d}$，$[Y_1; ...; Y_h] \in R^{l \times d}$。代码部分还包括这里未写的 dropout 层。我们建议查看提供的代码并留意在 PyTorch 中这个方程是如何实现的。

我们的模型使用了如图 2 所示的 transformer 块的自注意力层。正如课程中讨论的那样，transformer 块包含残差连接和层归一化。如果我们比较示意图和 model.py 中的 Block 部分的代码，可以注意到实现时并没有在 MLP 的输出上进行层归一化，而是对 Block 的输入进行层归一化。二者可以认为是等价的因为有一系列叠在一起的 transformer 块。

在 Perceiver 模型架构中，我们将模型中的第一个 transformer Block 替换成 DownProjectBlock。

此模块将序列的长度从 l 减少到 m。该模块之后是一系列常规 transformer 模块，这些模块在减少到 m 的序列长度上进行自注意力操作。我们将最后一个 Block 替换成 UpProjectBlock，该模块以上一个模块长度为 m 的输出作为输入，并将其投影回初始序列长度 l。
你需要实现 model.py 中的 DownProjectBlock，作为减少序列维度的第一个模块。为此，以一个可学习的基$C \in R^{m \times d}$作为查询，$m < l$。因此，得到如下方程：

$$Y_i^{(1)} = soft\max\left(\frac{(CQ_i)\,(XK_i)^T}{\sqrt{d/h}}\right)(XV_i)$$

得到$Y_i^{(1)} \in R^{m \times d}$，上标(1)表示输出对应第一层。维数降低后，CausalSelfAttention 层在形状为$R^{m \times d}$的输入上运算。在代码中用 bottleneck_dim 指代 m。注意，为了实现上述方程，需要在可学习的基 C 和输入序列间进行交叉注意力运算。这一部分已经作为 CausalCrossAttention 层提供给你了。我们推荐你完整阅读 attention.py 以理解如何使用交叉注意力层，并查找哪些参数对应键，值和查询输入。用 Xavier Uniform 初始化基向量矩阵 C。
为了回到初始维度，模型的最后一层替换成了 UpProjectBlock。这个模块将在前一层的输出Y^{L-1}和原始输入向量X上执行交叉注意力操作，使得输出序列的长度重新变为和输入序列的长度一致：

$$Y_i^{(L)} = soft\max\left(\frac{(XQ_i)\,(Y^{(L-1)}K_i)^T}{\sqrt{d/h}}\right)(Y^{(L-1)}V_i)$$

其中L是总层数。这使得最终的输出向量像 CausalSelfAttention 机制中设想的那样具有和原始输入相同的长度。请在 model.py 中实现 UpProjectBlock。
我们提供了将你实现的 DownProjectBlock 和 UpProjectBlock 集成到模型中的代码。当 variant 参数特化成 perceiver 时模型将使用这些层。
以下是你的代码应该支持的 bash 命令，从而实现模型的预训练、微调和在 dev，test 数据集上的预测。请注意预训练过程将花费大约 2h。

 # Pretrain the model
python src/run.py **pretrain** perceiver wiki.txt --bottleneck_dim 64 --pretrain_lr 6e-3
--writing_params_path perceiver.pretrain.params

epoch 650 iter 22: train loss **0.94973**. lr 7.182453e-04

 # Finetune the model
python src/run.py **finetune** perceiver wiki.txt --bottleneck_dim 64
--reading_params_path perceiver.pretrain.params
--writing_params_path perceiver.finetune.params
--finetune_corpus_path birth_places_train.tsv

epoch 10 iter 7: train loss **0.31546**. lr 5.983263e-04

 # Evaluate on the dev set; write to disk
python src/run.py **evaluate** perceiver wiki.txt --bottleneck_dim 64
--reading_params_path perceiver.finetune.params

--eval_corpus_path **birth_dev.tsv**
--outputs_path perceiver.pretrain.dev.predictions

data has 418352 characters, 256 unique.
number of parameters: 3339776
500it [00:51, 9.77it/s]
Correct: 85.0 out of 500.0: **17.0%**

 # Evaluate on the test set; write to disk
python src/run.py **evaluate** perceiver wiki.txt --bottleneck_dim 64
--reading_params_path perceiver.finetune.params
--eval_corpus_path **birth_test_inputs.tsv**
--outputs_path perceiver.pretrain.test.predictions

汇报 perceiver 注意力模型在预训练和微调后在 birth_dev.tsv 上出生地预测的准确度。

保存模型在 birth_test_inputs.tsv 上的预测结果到 perceiver.pretrain.test.predictions。本部分需要你提交：
perceiver.finetune.params, perceiver.pretrain.dev.predictions 以 perceiver.pretrain.test.predictions。
你的模型应该在 dev 数据集上取得至少 6%的准确率。
i.（8 分）我们将根据 test 数据集上是否取得至少 5%的准确率为你的模型打分，分数将公开；
ii.（2 分）提供 Perceiver 模型和 vanilla 模型关于层数 L，输入序列长度 l 和维度 m 的时间复杂度的表达式。

My answer:
Perceiver: l * m * 2 + m * m * (L-2)
vanilla: l * l * L

四、预训练知识方面的考虑（5 分）

(a) （1 分）简要解释为何预训练模型（vanilla）能够达到 10%以上的准确率，而未预训练的模型不能。

My answer:
模型通过预训练过程学习到世界知识的通用特征表示，这些表示有利于预测。

(b) （2 分）观察预训练+微调的 vanilla 模型的一部分正确预测和一部分错误预测。你会发现仅仅通过观察输出来分辨模型是检索到正确的出生地还是编造一个错误的出生地是不可能的。考虑这对于涉及预训练 NLP 组件的用户界面系统的影响。请提出两个不同的原因，解释为什么模型的这种行为（即无法确定是检索还是虚构）可能会引起关于此类应用程序的担忧，并为每个原因提供一个示例。

My answer:
(1) 模型无法满足用户的真实信息需求

示例：用户需要模型推荐特定主题的科学文献，模型返回不存在的结果

(2) 模型的回答可能造成侵权
示例：用户希望模型生成虚构的故事情节，模型将个人真实信息嵌入到情节中

(c) （2 分）如果模型在预训练阶段没有看到某个人的姓名，而且在微调阶段也没有看到该人，那么它不可能已经“学到”该人的出生地。然而，如果被询问，模型将为该人的姓名生成一个预测的出生地。简洁地描述一种模型可能用于为该人的姓名预测出生地的策略，并给出为什么这可能引起这类应用程序的使用担忧的一个原因。

My answer:
该姓名可以由多个其他姓名的子串组合而成，模型将其他姓名对应的出生地“混合”得到预测结果；仅通过模型的回答无法判断是检索还是虚构。