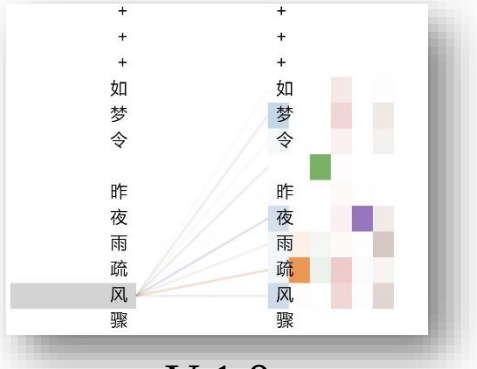
媒体与认知

Media and Cognition



V 1.0

清华大学电子工程系

姚刚 助教

Email: yg19@mails.tsinghua.edu.cn

第四次作业的考察目标

理论部分：

- 隐含马尔可夫模型HMM
  - **HMM的定义**
  - **HMM的评估问题和解码问题**
- 循环神经网络RNN
  - **传统RNN**
  - **引入门控机制的RNN**
- 自注意力机制Transformer

编程部分：

- 文本生成
  - **构建基于Transformer的GPT（Generative Pre-training）模型，在中文文本数据集（全宋词）上进行语言模型的训练、文本生成和可视化**

作业说明

- 任务一：单选题

  共5题：隐含马尔可夫模型、循环神经网络和注意力机制相关内容。

- 任务二：计算题

  共1题：隐含马尔可夫模型的解码问题。

- 任务三：实现基于GPT的文本生成程序代码

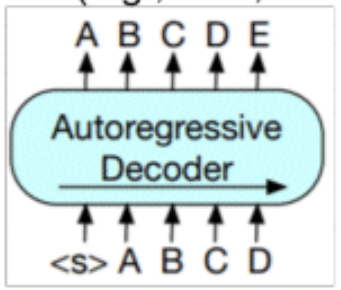
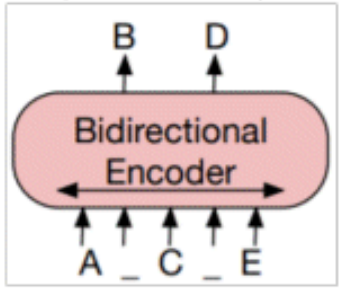
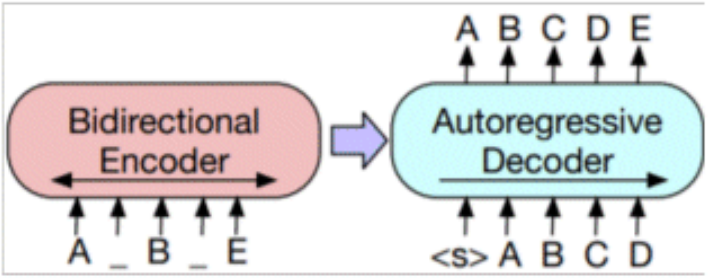
  需要同学们补全model.py中的代码，包括带掩码的多头自注意力机制和GPT的前向计算过程。

- 任务四：训练/文本生成/可视化

编程部分

基于Transformer的预训练模型

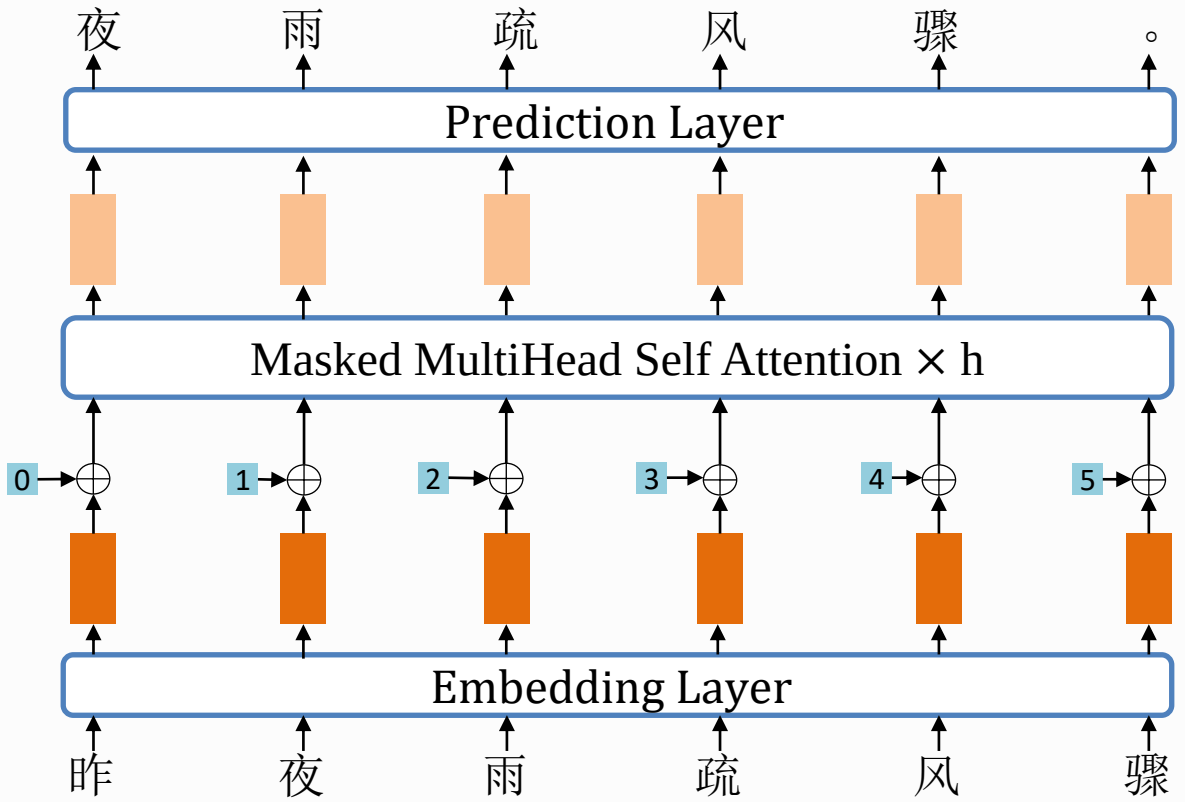
➢ Decoder-only & Encoder-only & Encoder-Decoder

	Autoregressive language model (e.g., GPT, GPT-2/3)	**Masked** language model (e.g., BERT, RoBERTa, XLM-R)	**Encoder-Decoder** (e.g., BART, T5)
Model & illustration			
Training objective	Predicting what word comes next given previous words	Predicting masked words given other words in the sequence	Corrupting a sequence and then predicting the original sequence

基于Transformer的预训练模型

➢ GPT

- Decoder-only
- 单向自回归语言模型：根据输入文本预测下一字符



基于GPT的文本生成任务

➢ 数据集

- 本次作业采用中文文本数据集（全宋词）作为训练和验证的数据集
- 数据预处理 tokenization：将文本分词得到token（可称为词元）并进行量化表示

python prepare.py –data_root data/quansongci

- 构建词汇表，为简便起见采用字符级的token

```
# get all the unique characters that occur in this text
chars = sorted(list(set(data)))
vocab_size = len(chars)
```

- 用词汇表集合中的字符序号表示token

```
# create a mapping from characters to integers
stoi = { ch:i for i,ch in enumerate(chars) }
itos = { i:ch for i,ch in enumerate(chars) }
```

- 划分训练集与验证集

```
# create the train and test splits
n = len(data)
train_data = data[:int(n*0.9)]
val_data = data[int(n*0.9):]
```

带掩码的多头自注意力机制

➢ 需要完成的代码

- 假设输入特征为 $X \in \mathbb{R}^{B\times L\times C}$，其中$B$为批次大小 (batch_size)，$L$为每个样本序列的长度 (seq_len), C为特征维度，$C = h \times d$，h为多头注意力机制的头数 (num_heads), d为单头注意力所用的特征维度；
- 语言模型采用decoder-only的架构，注意力掩码为 M
- Step 1：输入$X \in \mathbb{R}^{B\times L\times C}$，计算$q, k, v \in \mathbb{R}^{B\times L\times C}$

$$q = \text{Concat}(q_1, ..., q_h) = \text{Concat}(XW_1^Q, ..., XW_h^Q) = XW^Q$$
$$k = \text{Concat}(k_1, ..., k_h) = \text{Concat}(XW_1^K, ..., XW_h^K) = XW^K$$
$$v = \text{Concat}(v_1, ..., v_h) = \text{Concat}(XW_1^V, ..., XW_h^V) = XW^V$$

  - 第11讲编程教程示例如下，其中d_model对应于单头注意力所用的特征维度d:

```
self.K = nn.Linear(d_model, d_model)
self.V = nn.Linear(d_model, d_model)
self.Q = nn.Linear(d_model, d_model)
```

```
keys = self.K.forward(x)
values = self.V.forward(x)
queries = self.Q.forward(x)
```

带掩码的多头自注意力机制

➢ 需要完成的代码

- Step 2：对 q,k,v进行变形，变形后尺寸为$B \times L \times h \times d$
  - 变形操作便于多头注意力的并行计算，第11讲编程教程采用在for循环中计算单头注意力
  - 使用Tensor.view()或者Tensor.reshape()

$$q = q.view(?,?,?,?)$$
$$k = k.view(?,?,?,?)$$
$$v = v.view(?,?,?,?)$$

带掩码的多头自注意力机制

➢ 需要完成的代码

- Step 3：计算每一个注意力头的输出

$$head_i = \text{Attention}(q_i, k_i, v_i) = \text{Softmax}\left(q_i k_i^T / \sqrt{d} + M\right) v_i$$

  - Step 3.1：当前q, k, v的尺寸均为$B \times L \times h \times d$，为便于进行多头自注意力机制的张量乘法计算，需要进行维度交换，使得q, v尺寸为$B \times h \times L \times d$，$k^T$的尺寸为$B \times h \times d \times L$
    - 使用Tensor.transpose()或者Tensor.permute()
    - 示例：

```
>>> x = torch.zeros(3, 4, 5)
>>> x.shape
torch.Size([3, 4, 5])
>>> y = x.transpose(0,1)
>>> y.shape
torch.Size([4, 3, 5])
>>> z = x.permute(2,1,0)
>>> z.shape
torch.Size([5, 4, 3])
```

带掩码的多头自注意力机制

➢ 需要完成的代码

- Step 3.2：计算$q_i k_i^T/\sqrt{d}, i = 1, \ldots, h$，其尺寸为$B \times h \times L \times L$
  - 编程教程中的实现：

```
sqrt_d = self.d_model ** 0.5
att = torch.matmul(queries, keys.transpose(1,2)) / sqrt_d
```

  - 注意在Step 3.1中已对k进行转置
- Step 3.3：计算$q_i k_i^T/\sqrt{d} + M, i = 1, \ldots, h$，其尺寸为$B \times h \times L \times L$
  - 参考课件第11讲P39

```
>>> attn_masked = attn_raw.masked_fill(mask, -np.inf)
>>> attn_masked
tensor([[12., -inf, -inf, -inf],
        [ 6.,  8., -inf, -inf],
        [ 3.,  7., 10., -inf],
        [ 5.,  6.,  9.,  4.]])
```

$$mask = \begin{pmatrix} False & \cdots & True \\ \vdots & \ddots & \vdots \\ False & \cdots & False \end{pmatrix}$$

带掩码的多头自注意力机制

➢ 需要完成的代码

- Step 3.4-3.6：计算attn_out = Softmax$\left(qk^T/\sqrt{d}+M\right)v$，其尺寸为$B \times h \times L \times d$
  - 第11讲编程教程中的实现

```
att_softmax = torch.softmax(att, dim=2)
# shape: [N, SEQ, D_MODEL]
att_out = torch.matmul(att_softmax, values)
```

  - 注意沿着哪一个维度进行Softmax归一化（参数dim的取值）

带掩码的多头自注意力机制

➢ 需要完成的代码

- Step 4：拼接多头注意力的计算结果

$$Concat(head_1, \dots, head_h)$$

  - 本次作业中对Step 3的输出结果进行维度交换（$\mathbb{R}^{B\times h\times L\times d} \rightarrow \mathbb{R}^{B\times L\times h\times d}$）和变形（$\mathbb{R}^{B\times L\times h\times d} \rightarrow \mathbb{R}^{B\times L\times (hd)}$）即可

$$attn\_out = attn\_out.transpose(?,?).\text{reshape}(?,?,?)$$

- Step 5：经过一层线性层完成计算

$$MultiHead(q,k,v) = Concat(head_1, \dots, head_h)W^o$$

  - 第11讲编程教程中的实现：

```
ret = self.linear.forward(out_cat)
```

  - 注意本次作业中还添加了Dropout层

GPT的前向计算

➢ 需要完成的代码

- 假设输入的文本序列为$U = \{u_1, \ldots, u_L\} \in \mathbb{R}^{B\times \mathrm{L}}$，其中$u_i$表示第$i$个单词的字符序号，$B$为批次大小 (batch_size)，$L$为每个样本序列的长度 (seq_len)
  - 第11讲编程教程中采用第11讲课件P26中的正弦、余弦函数位置编码
  - 本次作业采用nn.Embedding()实现可学习的位置编码，即根据输入的位置序号输出对应的位置嵌入表示
- Step 1：生成每个单词的位置序号$P = \{0, 1, \ldots, L-1\}$
  - 使用torch.arange()生成，注意生成张量的数据类型（torch.long）和设备（与输入U位于相同设备：cpu或者gpu）
- Step 2：使用nn.Embeding()得到单词嵌入$E_U \in \mathbb{R}^{B\times \mathrm{L}\times C}$和位置编码$E_P \in \mathbb{R}^{\mathrm{L}\times C}$，其中$C$为特征通道数

GPT的前向计算

➢ 需要完成的代码

- Step 3：初始化Transformer的输入，使用if语句判断是否使用位置编码

$$h_0 = \begin{cases} E_U & if\ no\_pos \\ E_U + E_P & else \end{cases}$$

  - 注意本次作业中添加了Dropout层

- Step 4：生成自注意力系数掩码$M \in \mathbb{R}^{L \times L}$
  - 可参考第11讲课件P39

```
>>> mask = torch.triu(torch.ones(4, 4), diagonal=1).bool()
>>> mask
tensor([[False,  True,  True,  True],
        [False, False,  True,  True],
        [False, False, False,  True],
        [False, False, False, False]])
```

$$mask = \begin{pmatrix} False & \cdots & True \\ \vdots & \ddots & \vdots \\ False & \cdots & False \end{pmatrix}$$

  - 注意参数diagonal的数值以及生成张量的数据类型（torch.bool）和设备（与输入U位于相同设备：cpu或者gpu ）

GPT的前向计算

➢ 需要完成的代码

- Step 5：使用for循环计算多层Transformer的输出，并保存每层的注意力系数用于可视化
  - 每层Transformer会输出计算结果和注意力系数
  - 定义一个列表保存所有层的注意力系数用于可视化
  - 示例：

```
attn_weights = []
for i in range(?):
    out, weights = ?
    attn_weights.append(?)
```

- Step 6：通过一层线性层得到输入文本的下一个单词的预测结果
  - 输入线性层之前还需要通过层归一化（Layer Normalization）
  - 预测结果不需要通过Softmax函数，因为Cross Entropy Loss自带Softmax函数

基于GPT的文本生成任务

- 本次作业要求python版本至少为3.8，若python版本过低则需要更新
- 本次作业需要安装bertviz库，可以使用pip install bertviz进行安装
- 模型训练
  - 通过--data_root 参数可以指定数据集目录；
  - 通过--ckpt_path 参数可以指定指定模型的保存目录，便于后续测试和可视化；

```
python train.py --data_root data/quansongci --ckpt_path workdirs/quansongci
```

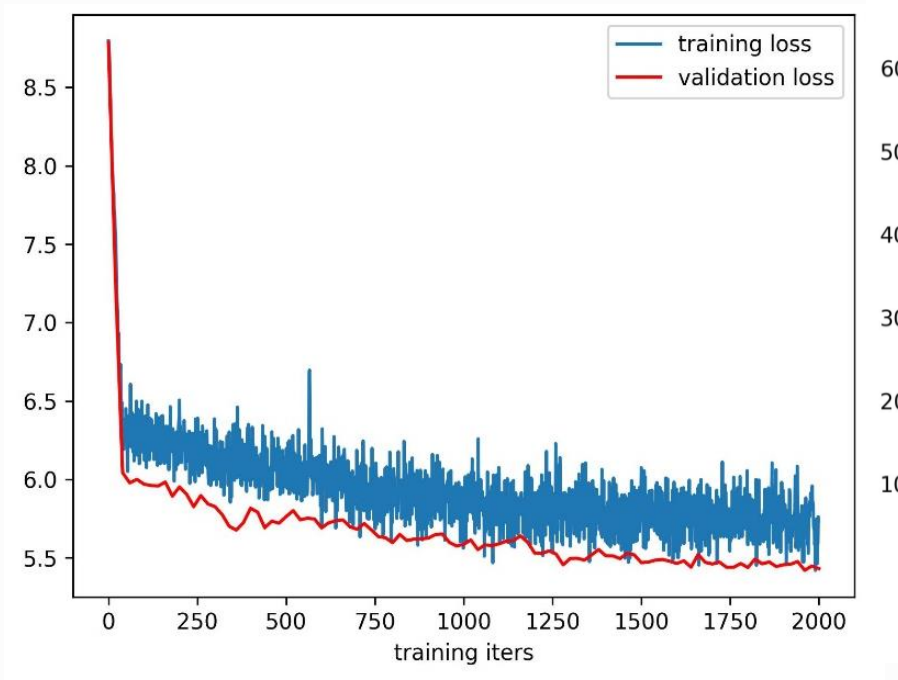
  - 通过--batchsize 参数可以训练的批次大小，默认为16；
  - 通过--iters 参数可以指定迭代次数，默认为2000；
  - 通过--max_seq_len 参数可以指定训练时每个样本的序列长度，默认为64。

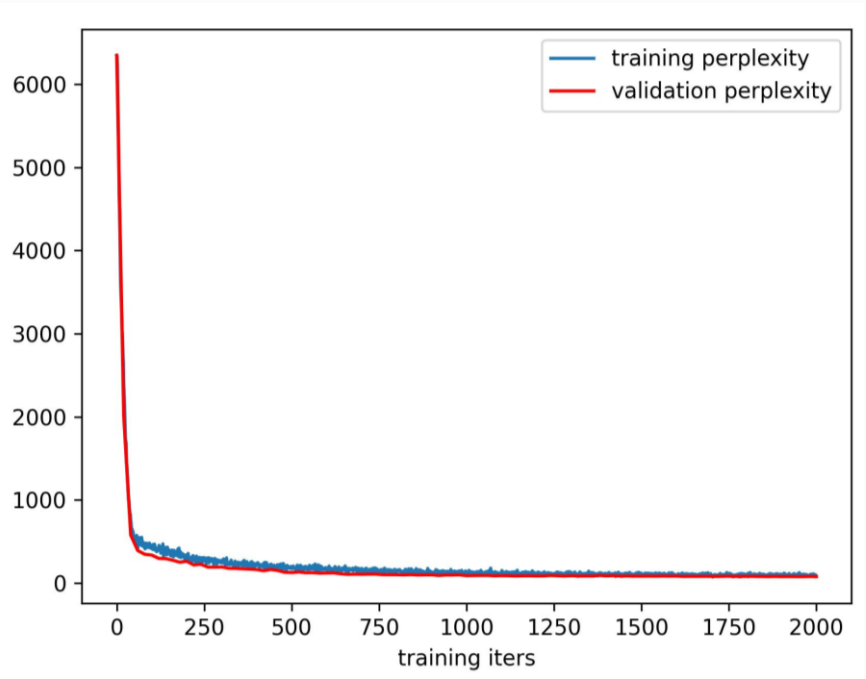
基于GPT的文本生成任务

➢ 模型训练

- 训练过程中在训练集和验证集上的的loss和困惑度（perplexity）变化会保存在workdirs/quansongci/loss&perplexity.jpg中
- 困惑度为交叉熵损失函数（CrossEntropyLoss）的指数：

$$perplexity = e^{CrossEntropyLoss}$$





基于GPT的文本生成任务

➢ 文本生成

- 通过--data_root 参数可以指定数据集目录；
- 通过--ckpt_path 参数可以指定指定模型的保存目录；
- 通过--start 参数指定初始文本（默认为“\n”);
- 通过--num_samples 参数指定文本生成次数，默认为10；
- 通过--max_new_tokens 参数指定每次生成文本的单词数，默认为500。

python sample.py --data_root data/quansongci --workdirs/quansongci

python sample.py --data_root data/quansongci --workdirs/quansongci --start +++清平乐

基于GPT的文本生成任务

➢ 文本生成

◆ 生成示例

- 注：由于训练文本中含有少量古籍生僻字，使用系统字体显示可能会出现“□”

```
+++踏莎行（和江行作）
水面菱花，莺昏杨柳。年华将晚天容晚。杏花枝上笑开时，红窗外、绿阴庭院。
坐久西凉，尊前被雨和烟暖。酒情不似秦楼梦。应先问、月中春，怎生心事。
+++江城子（再次韵）
一枝梅萼粉香黄。粉香吹。不禁香。十里扬州，又见暮云。向梦魂间，重向朱门。风雨半凝眸芳草字，还又入、夕阳红。
青鞋束带梦中看。两眉尖。也应怪、花枝好在人西。枕上几花，花底闲开。
+++江神子（戊午前一日上元素）
云垂幕日暮如桥。卷帘张。画帘垂。今日去年，芳草伴春心。桃李旧闲闲去客，花片片，梦无情。
楚衣清夜梦入帘栊。酒销迟。背人言。残醉月明花影里，人不见，似人空。
```

```
+++清平乐（九日登高作）
去年今夕。还是佳期夕。红雨低飞芳草。只有双双双翠。
当年燕子丁东。欲如飞絮濛濛。风雨不禁啼鸟。一回红叶，墙头春在何处。
+++清平乐
画帘暮。燕子衔泥絮。已记踏青无觅处。草草幽闲长在否。
旧游闲去程。又还是春知。门外夕阳天末，画桥柳色无情。
+++清平乐
新烟半掩。冷雨残红燕。白白飞花飞不定。一线旧欢罗带。
闲庭两点清星。锦衣初试新妆。莫嫌春思不似，小屏睡过余香。
```

基于GPT的文本生成任务

➢ 探究残差连接和位置编码对模型的影响

- 在默认参数下，位置编码和残差连接都是启用的。请同学们分别关闭位置编码和残差连接训练模型，比较不同参数设置下训练过程中在训练集和验证集上的loss和困惑度的变化以及文本生成的质量。
- 关闭位置编码

```
python train.py --data_root data/quansonci --ckpt_path workdirs/quansongci_no_pos --no_pos
```

- 关闭残差连接

```
python train.py --data_root data/quansonci --ckpt_path workdirs/quansongci_no_res --no_res
```

基于GPT的文本生成任务

➢ 可视化

- 本次作业可视化采用bertviz库可视化注意力系数。
- 打开jupyter文件attnvis.ipynb，检查数据集路径、模型保存路径和可视化文本路径是否正确

```
##################################################
# set the params
ckpt_path = 'workdirs/quansongci'
data_root = 'data/quansongci'
vis_text_path = 'data/vis/vis_1.txt'
##################################################
```

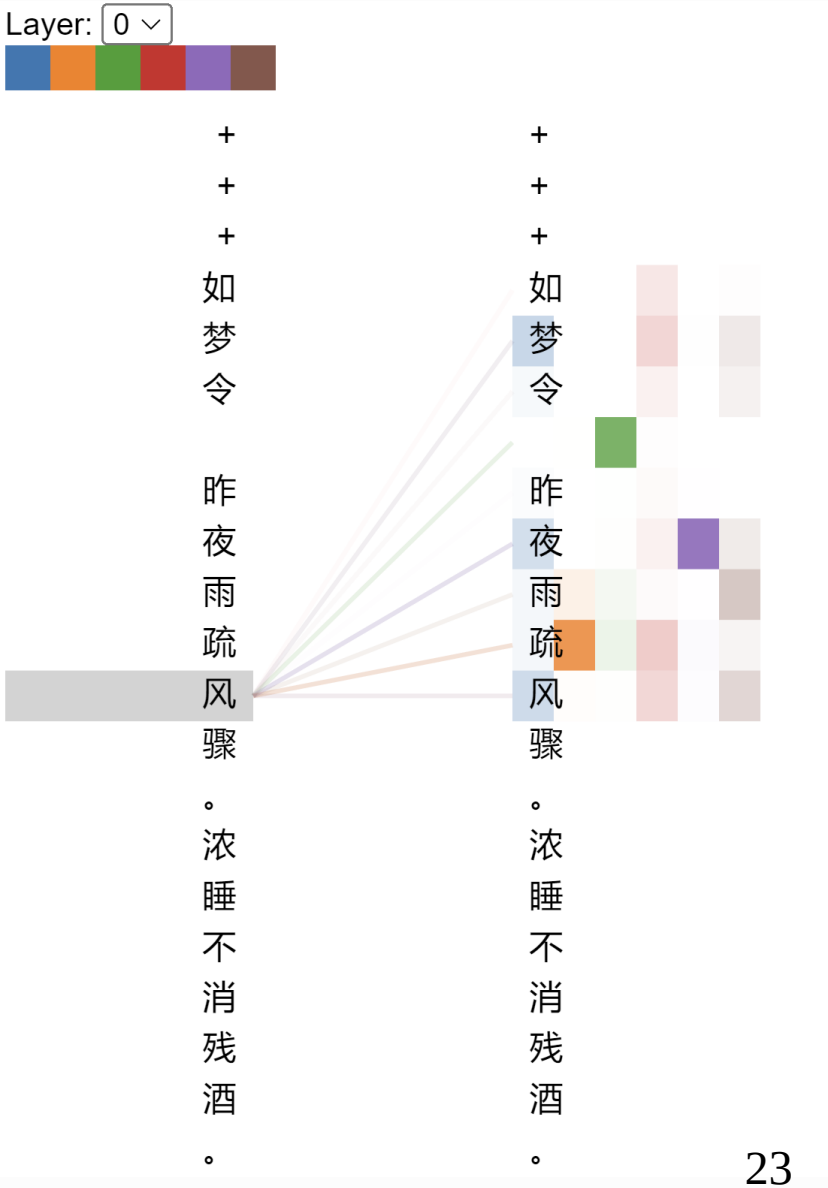
- 运行第一部分读取可视化文本

```
Loading meta from data/quansongci\meta.pkl...
Input texts: +++如梦令
昨夜雨疏风骤。浓睡不消残酒。试问卷帘人，却道海棠依旧。知否。知否。应是绿肥红瘦。
```

基于GPT的文本生成任务

➢ 可视化

- ◆ 运行第二部分对模型的注意力系数进行可视化
  - 可以自行调节层数
  - 鼠标放在单词上即可查看该单词与其他单词的注意力系数
  - 颜色越深表示注意力系数越大
  - 不同颜色表示不同的注意力头



理论部分

隐含马尔可夫模型

- 隐含马尔可夫模型(Hidden Markov Model，HMM)是一个双重随机过程
  - 隐含状态是一个马尔科夫链，观测值由隐含状态决定
  - 我们只能看到观测值，而看不到隐含状态
- 理解HMM模型参数$\lambda = (\pi, A, B)$的定义

参数	含义	实例
A	与时间无关的状态转移概率矩阵	类间转移概率
B	给定状态下，观察值概率分布	给定类别，特征向量分布
π	初始状态空间的概率分布	初始时选择类别的概率

- 理解前向后向算法中的前向变量和后向变量的定义
  - 前向变量$\alpha_t(i) = P(o_1, \ldots, o_t, q_t = S_i|\lambda)$
  - 后向变量$\beta_t(i) = P(o_{t+1}, \ldots, o_T|q_t = S_i, \lambda)$

隐含马尔可夫模型

➢ 如何使用模型参数$\lambda=(\pi,A,B)$和前向、后向变量计算特定概率

- 作业中需计算如下概率：

$$P(q_t=S_i|O,\lambda)$$

$$P(q_t=S_i,q_{t+1}=S_j,O|\lambda)$$

- 利用上述两种概率，我们可以得到许多有意义的信息：
  - 在观测到序列O的条件下，状态S_i出现的概率
  - 在观测到序列O的同时，发生了从状态S_i转移到状态S_j的概率

1.1 给定 HMM 的模型参数 λ，隐含状态总数为 N。设给定观测序列 O 的条件下，在第 t 时刻处于状态 S_i 的概率为 $\gamma_t(i)=P(q_t=S_i|O,\lambda)$。若已经计算得到所有前向变量 $\alpha_t(i)$ 和后向变量 $\beta_t(i)$，则下列计算 $\gamma_t(i)$ 的方法中正确的是哪项？

(A) $\gamma_t(i)=\alpha_t(i)\beta_t(i)$

(B) $\gamma_t(i)=\alpha_t(i)\beta_t(i)\ /\ \sum_{j=1}^{N}\alpha_t(j)\beta_t(j)$

(C) $\gamma_t(i)=\alpha_t(i)+\beta_t(i)$

(D) $\gamma_t(i)=(\alpha_t(i)+\beta_t(i))\ /\ \sum_{j=1}^{N}(\alpha_t(j)+\beta_t(j))$

1.2 对于一个参数为 $\lambda=\{\pi,A,B\}$ 的 HMM，隐含状态总数为 5，且观测序列 O 长度为 10。若已经计算得到所有前向变量 $\alpha_t(i)$ 和后向变量 $\beta_t(i)$，则下列关于 $P(q_5=S_2,q_6=S_4,O|\lambda)$ 的计算方式中正确的是哪项？

(A) $\alpha_5(2)a_{24}$

(B) $b_4(O_6)\beta_6(4)$

(C) $\alpha_5(2)a_{24}b_4(O_6)$

(D) $\alpha_5(2)a_{24}b_4(O_6)\beta_6(4)$

HMM的评估问题

➢ 给定$\lambda = (\pi, A, B)$，计算观测序列$O = \{o_1, o_2, \dots, o_T\}$的概率$P(O|\lambda)$

◆ 前向变量$\alpha_t(i) = P(o_1, \dots, o_t, q_t = S_i|\lambda)$

◆ 若已经计算出$\alpha_t(i), i = 1, \dots, N$，如何求$\alpha_{t+1}(i), i = 1, \dots, N$？

(1) $\alpha_t(j)a_{ji} = P(o_1, \dots, o_t, q_t = S_j\,|\lambda) \cdot P(q_{t+1} = S_i|q_t = S_j, \lambda) = P(o_1, \dots, o_t, q_t = S_j, q_{t+1} = S_i|\lambda)$

(2) 对t时刻所有状态求和，$\sum_{j=1}^{N} \alpha_t(j)a_{ji} = P(o_1, \dots, o_t, q_{t+1} = S_i|\lambda)$

(3) 由概率乘法公式，$\alpha_{t+1}(i) = \left[\sum_{j=1}^{N} \alpha_t(j)a_{ji}\right] b_i(o_{t+1})$

$P(o_1, \dots, o_t, o_{t+1}, q_{t+1} = S_i|\lambda) = P(o_1, \dots, o_t, q_{t+1} = S_i|\lambda) \cdot P(o_{t+1}|q_{t+1} = S_i, \lambda)$

(4) 得到$\alpha_T(i) = P(O, q_T = S_i|\lambda)$后，对所有状态求和，有$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i)$

HMM的评估问题

- 后向变量$\beta_t(i) = P(o_{t+1}, \dots, o_T | q_t = S_i, \lambda)$
- 若已经计算出$\beta_{t+1}(i), i = 1, \dots, N$，如何求$\beta_t(i), i = 1, \dots, N$?

(1) $b_j(o_{t+1})\beta_{t+1}(j) = P(o_{t+1}|q_{t+1} = S_j) \cdot P(o_{t+2}, \dots, o_T|q_{t+1} = S_j, \lambda) = P(o_{t+1}, o_{t+2}, \dots, o_T|q_{t+1} = S_j, \lambda)$

(2) $a_{ij}b_j(o_{t+1})\beta_{t+1}(j) = P(q_{t+1} = S_j|q_t = S_i, \lambda) \cdot P(o_{t+1}, o_{t+2}, \dots, o_T|q_{t+1} = S_j, \lambda) = P(o_{t+1}, \dots, o_T, q_{t+1} = S_j|q_t = S_i, \lambda)$

(3) 对t+1时刻的状态求和，可得 $\beta_t(i) = \sum_{j=1}^{N} a_{ij}b_j(o_{t+1})\beta_{t+1}(j)$

(4) 得到$\beta_1(i) = P(o_2, \dots, o_T|q_1 = S_i, \lambda)$后，

有$P(O, q_1 = S_i|\lambda) = \beta_1(i) \cdot P(o_1, q_1 = S_i|\lambda) = \beta_1(i) \cdot P(q_1 = S_i|\lambda) \cdot P(o_1|q_1 = S_i, \lambda) = \beta_1(i)\pi_i b_i(o_1)$

(5) 最终对t=1所有状态求和，有$P(O|\lambda) = \sum_{i=1}^{N} \beta_1(i)\pi_i b_i(o_1)$

HMM的评估问题

➢特定概率计算

- 前向变量$\alpha_t(i) = P(o_1, \dots, o_t, q_t = S_i|\lambda)$
- 后向变量$\beta_t(i) = P(o_{t+1}, \dots, o_T|q_t = S_i, \lambda)$

$$\alpha_t(i)\beta_t(i) = P(o_1, \dots, o_T, q_t = S_i|\lambda) = P(O, q_t = S_i|\lambda)$$

$$\sum_{i=1}^{N} \alpha_t(i)\beta_t(i) = P(O|\lambda)$$

$$P(q_t = S_i, q_{t+1} = S_j, O|\lambda) = ?$$

HMM的解码问题

- 给定$\lambda = (\pi, A, B)$，和观测序列O，求最可能的状态序列
  - Viterbi 算法 – 使每一时刻状态序列出现相应观测值的可能达到最大
  - 定义$\delta_t(i)$：相当于前向变量中的最大值

$$\delta_t(i) = \max_{q_1,\dots,q_{t-1}} P(q_1, q_2 \dots, q_t = S_i, o_1, \dots, o_t|\lambda)$$

$$= \max_{q_1,\dots,q_{t-2},1\le j\le N} P\left(q_1, q_2 \dots, q_{t-1} = S_j, q_t = S_i, o_1, \dots, o_t \,|\lambda\right)$$

其中$P\left(q_1, q_2 \dots, q_{t-1} = S_j, q_t = S_i, o_1, \dots, o_t|\lambda\right)$

$$= P\left(q_1, \dots, q_{t-1} = S_j, o_1, \dots, o_{t-1}|\lambda\right) \cdot P\left(q_t = S_i, o_t|q_{t-1} = S_j, \lambda\right)$$

$$= P\left(q_1, \dots, q_{t-1} = S_j, o_1, \dots, o_{t-1}|\lambda\right) \cdot P\left(q_t = S_i \,|q_{t-1} = S_j, \lambda\right) \cdot P(o_t|q_t = S_i, \lambda)$$

因此$\delta_t(i) = \max_{1\le j\le N}\left[\delta_{t-1}(j)a_{ji}\right]b_i(o_t)$

HMM的解码问题

- 给定$\lambda = (\pi, A, B)$，和观测序列O，求最可能的状态序列
  - 定义$\delta_t(i)$：t时刻状态$q_t = S_i$的所有状态序列可能拥有的最大概率

$$\delta_t(i) = \max_{q_1,\cdots,q_{t-1}} P(q_1, q_2 \ldots, q_t = S_i, o_1, \ldots, o_t|\lambda)$$

$$\delta_t(i) = \max_{1\le j\le N}\left[\delta_{t-1}(j)a_{ji}\right]b_i(o_t)$$

  - 定义$\phi_t(i)$：t时刻状态$q_t = S_i$，且概率最大的状态序列中，q_{t-1}所属的状态类别

$$\phi_t(i) = \operatorname{argmax}_{1\le j\le N}\left[\delta_{t-1}(j)a_{ji}\right]$$

  - 在得到$\phi_t(i), t = 1, \ldots, T, i = 1, \ldots, N$后，可以从T时刻开始回溯最可能的状态序列

$$q_t^* = \phi_{t+1}(q_{t+1}^*)$$

HMM计算题

某手机专卖店今年元旦新开业，每月上旬进货时，由专卖店经理决策，采用三种进货方案中的一种：高档手机 (H)，中档手机 (M)，低档手机 (L)。

当月市场行情假设分为畅销 (S_1) 和滞销 (S_2) 两种。畅销时，三种进货方案的概率分别为 0.4, 0.4, 0.2；滞销时，三种进货方案的概率分别为 0.2, 0.3, 0.5。

某月份市场行情为畅销，下一个月份为畅销和滞销的概率分别为 0.6 和 0.4；某月份市场行情为滞销，下一个月份为畅销和滞销的概率分别为 0.5 和 0.5。

开业第一个月市场行情为畅销和滞销的可能性均为 0.5。

(1) 如果我们采用隐含马尔可夫模型 (HMM) 对该专卖店进货环节建模，请写出 HMM 对应的参数 $\lambda = \{\pi, A, B\}$。

(2) 在第一季度中，采购业务员执行的进货方案为“高档手机，中档手机，低档手机”，即观测序列为 H, M, L。请利用 Viterbi 算法推测前三个月的市场行情。

传统循环神经网络的梯度消失与梯度爆炸问题

➢ 循环神经网络

- 对序列中的长距离相关信息进行建模是涉及序列的任务中十分重要的一点，例如在阅读理解任务里，题目和正文中的关键词可能相距很远，这就需要模型具备足够好的长距离相关信息建模能力。

- 传统RNN在训练时可能存在梯度消失和梯度爆炸问题，较远的误差无法得到有效传递，因此学习长距离相关信息时面临较大挑战。

传统循环神经网络的梯度消失与梯度爆炸问题

➢ 循环神经网络

◆ 对RNN的计算过程进行简化，考虑一个暂不采用激活函数以及输入h_0的RNN:

$$h_t = Uh_{t-1} = U(Uh_{t-2}) = U^t h_0$$

- 其中U_t为t个U矩阵连乘。若矩阵U存在如下特征值分解：

$$U = Q\Lambda Q^{\top}$$

- 其中Q为单位正交矩阵（每一列为模长为1的特征向量），$Q^{\top}$为Q的转置，Λ为特征值对角矩阵，则上述的RNN计算过程可表示为：

$$h_t = Q\Lambda^t Q h_0$$

- 假设损失函数为$\mathcal{L}$，其对h_t的导数为$\partial\mathcal{L}/\partial h_t$，则可以计算得到：

$$\frac{\partial \mathcal{L}}{\partial h_0} = \frac{\partial \mathcal{L}}{\partial h_t}\frac{\partial h_t}{\partial h_{t-1}}\cdots\frac{\partial h_1}{\partial h_0} = U^t\frac{\partial \mathcal{L}}{\partial h_t} = Q\Lambda^t Q\frac{\partial \mathcal{L}}{\partial h_t}$$

- 试分析矩阵Q的特征值λ_i满足什么条件时，容易造成梯度消失或者梯度爆炸问题。

传统循环神经网络的梯度消失与梯度爆炸问题

1.4 在本题中我们对传统 RNN 难以学习长距离相关信息的问题进行一个简单的讨论:

对 RNN 的计算过程进行简化，考虑一个暂不采用激活函数以及输入 $\boldsymbol{h}_0$ 的 RNN:

$$\boldsymbol{h}_t = U\boldsymbol{h}_{t-1} = U\left(U\boldsymbol{h}_{t-2}\right) = \ldots = U^t\boldsymbol{h}_0$$

其中 U^t 为 t 个 U 矩阵连乘。若矩阵 U 存在如下特征值分解:

$$U = Q\Lambda Q^\top$$

其中 Q 为单位正交矩阵（每一列为模长为 1 的特征向量），$Q^\top$ 为 Q 的转置，Λ 为特征值对角矩阵，则上述的 RNN 计算过程可表示为:

$$\boldsymbol{h}_t = Q\Lambda^t Q^\top \boldsymbol{h}_0$$

通过计算可以得到:

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial h_0} &= \frac{\partial \mathcal{L}}{\partial h_t}\frac{\partial h_t}{\partial h_{t-1}}\cdots\frac{\partial h_1}{\partial h_0} \\ &= U^t\frac{\partial \mathcal{L}}{\partial h_t} = Q\Lambda^t Q^\top\frac{\partial \mathcal{L}}{\partial h_t}\end{aligned}$$

下列说法中正确的是:

(A) 当 Q 的特征值 $\lambda_i < 1$ 时容易造成梯度消失问题

(B) 当 Q 的特征值 $\lambda_i = 1$ 时容易造成梯度消失问题

(C) 当 Q 的特征值 $\lambda_i > 1$ 时容易造成梯度消失问题

谢谢大家！