

1.

a)

i)

since α is actually soft-max function with each value somewhat between 0 to 1. Summing up all of them result in 1. And this means soft-max function is probability distribution. And the categorical case is we have n number of elements with their own probability ($0 < p_i < 1$).

ii)

query for a token gets dot product with all of other tokens *key*. This value receives high amount whenever this dot product gets high ($k_j^T \cdot q \gg k_i^T \cdot q$). This case happens when all *key* values of j tokens are large numbers and they're same sign with q element-wise. And in this case we have much weight on j token.

iii)

The output of previous question is when α put much weight on specific token called j . in this case c will get higher attention on j token and approximately focus nothing to other tokens. This will cause c to get almost v_j value.

iv)

for example, if we have sentence which is tokenized to ["I", "sat", "by", "the", "river", "side"] and the dot production of "side" *query*(q) with "river" *key* is very high but with other keys are very low, it means that "side" token just focus on the "river" token and gets "river" value(copied) but other values aren't considered much.

b)

i)

as it was mentioned:

$v_a = c_1 a_1 + c_2 a_2 + \dots + c_m a_m \rightarrow$ in which c is just constant vector where each element has constant value

should show that:

$M s = v_a \rightarrow M(v_a + v_b) = v_a \rightarrow M v_a + M v_b = v_a \rightarrow$ so two deduction should be made :

1) $M v_b = 0$

2) $M v_a = v_a$

So we should consider M in some way that Matrix Product with v_b get 0. And Matrix Product with v_a act like that its eye(I) matrix.

If we consider M like this: $M = \begin{bmatrix} a_1^T \\ \dots \\ a_m^T \end{bmatrix} = A^T$

$Mv_a = [a_1^T a_1 c_1, a_2^T a_2 c_2, \dots, a_m^T a_m c_m] \rightarrow$ we know from the fact that a 's are orthogonal and normalized so $a_i^T \cdot a_j^T = 0$, $a_i^T \cdot a_i^T = 1$

$Mv_b = [a_1^T b_1 x_1, a_2^T b_2 x_2, \dots, a_m^T b_m x_m] \rightarrow$ we also know that a and b are orthogonal so the whole matrix get 0.

$Mv_a + Mv_b = [c_1, c_2, \dots, c_m] + [0, 0, \dots, 0] = [c_1, c_2, \dots, c_m] \rightarrow$ this matrix should be equal to v_a and we know that v_a is some constant element of c .

So we deduce that $M = A^T$.

ii)

we want at the end this equation : $c \approx \frac{1}{2} (v_a + v_b)$

it means that values of other keys should be ignored and just consider a and b key.

So for winning of v_a and v_b we should make their α very large to other keys so then we should have

$K_a q \approx K_b q \rightarrow$ and this value should be very much large

$K_i q \approx 0 \rightarrow$ for every i not equal to a, b .

So we should pick q in such way that it contains both k_a and k_b so that the dot product gets large.

We can consider $q = L(k_a + k_b) \rightarrow L$ is some large number

And as the question said all keys are orthogonal and normalized to 1 so for every i not equal to a, b we have $k_i \cdot q = k_i \cdot L(k_a + k_b) = 0 \rightarrow$ because i is orthogonal to a, b

And $k_a q = k_b q = k_a \cdot L(k_a + k_b) = L \rightarrow$ because a to a is normalized and then get 1 at matrix product

So when we calculate alpha:

$$\alpha_a = \alpha_b = \frac{\exp(L)}{2 \cdot \exp(L) + n - 2} \approx \frac{\exp(L)}{2 \cdot \exp(L)} \approx 0.5$$

c)

i)

here question is almost same as what we have proposed in the previous question. Since that covariance matrix is multiplied by vanishingly small α , the effect of covariance gets almost 0. So we can almost say

that the equation $k_i \approx \mu_i$ holds true. And all μ are perpendicular we can result that we should consider q like previous question.

$$q = L(\mu_a + \mu_b) \rightarrow L \text{ is some large number}$$

ii)

as in the question said we should assume $\Sigma_a = \alpha I + \frac{1}{2}(\mu_a \mu_a^T)$

and α is vanishingly small and since μ is normalized to 1 we have $\Sigma_a \approx 0 + \frac{1}{2} = \frac{1}{2}$ and $\mu = 1$

so we have:

$$k_a = \varepsilon \mu_a, \varepsilon \sim N(1, 0.5)$$

and k for other i is just same as before equal μ_i .

three cases :

- $i \neq a, b \rightarrow q.k = 0$
- $i = a \rightarrow q.k = \varepsilon L$
- $i = b \rightarrow q.k = L$

finally we have $c = \alpha_a.v_a + \alpha_b.v_b$ and :

$$c = \frac{\exp(\varepsilon L)}{(\exp(\varepsilon L) + \exp(L))} . v_a + \frac{\exp(L)}{(\exp(\varepsilon L) + \exp(L))} . v_b$$

when ε is minimum i.e 0.5 $\rightarrow c \approx v_b$

when ε is maximum i.e 1.5 $\rightarrow c \approx v_a$

d)

i)

there can be multiple solution for this question:

$$- \quad \frac{1}{2} (c_1 + c_2) = \frac{1}{2} (v_a + v_b) \rightarrow c_1 + c_2 = v_a + v_b$$

We can design in a way that $c_1 = v_a$, $c_2 = v_b$

And this can be achieved by $q_1 = L.\mu_a$, where L is large number and same for $q_2 = L.\mu_b$

Because μ are mutually orthogonal so for c_1 we just have value of q_1 which is just copy of μ_a and same happens for q_2

- Or we can just use combination of them just like previous question
 $q_1 = q_2 = L(k_a + k_b) \rightarrow \alpha_a = \alpha_b = \frac{1}{2} \rightarrow c_1 = \frac{1}{2} (v_a + v_b)$, $c_2 = \frac{1}{2} (v_a + v_b)$
 $\frac{1}{2}(c_1+c_2) = \frac{1}{2}(v_a + v_b)$

ii)

as in the question *c part ii* we discussed there are the fact that we have

we should assume $\Sigma_a = \alpha I + \frac{1}{2}(\mu_a \mu_a^T)$

and α is vanishingly small and since μ is normalized to 1 we have $\Sigma_a \approx 0 + \frac{1}{2} = \frac{1}{2}$ and $\mu = 1$

so we have:

$$k_a = \epsilon \mu_a, \epsilon \sim N(1, 0.5)$$

and k for other i is just same as before equal μ_i .

and what we have designed for query vector in the previous question was

$q_1 = L \cdot \mu_a, q_2 = L \cdot \mu_b$ then :

$$q_1 \cdot k_a = q_1 \cdot \epsilon \mu_a = L \cdot \mu_a \cdot \epsilon \mu_a = L \cdot \epsilon \rightarrow \alpha_1 \approx \exp(L \cdot \epsilon) / \exp(L \cdot \epsilon) \approx 1$$

$$q_2 \cdot k_b = q_2 \cdot \mu_b = L \cdot \mu_b \cdot \mu_b = L \rightarrow \alpha_2 \approx \exp(L) / \exp(L) \approx 1$$

so we can see that $c_1 = v_a$ and $c_2 = v_b$

and in multi head attention we should take average : $c = \frac{1}{2}(c_1 + c_2) = \frac{1}{2}(v_a + v_b)$

2.

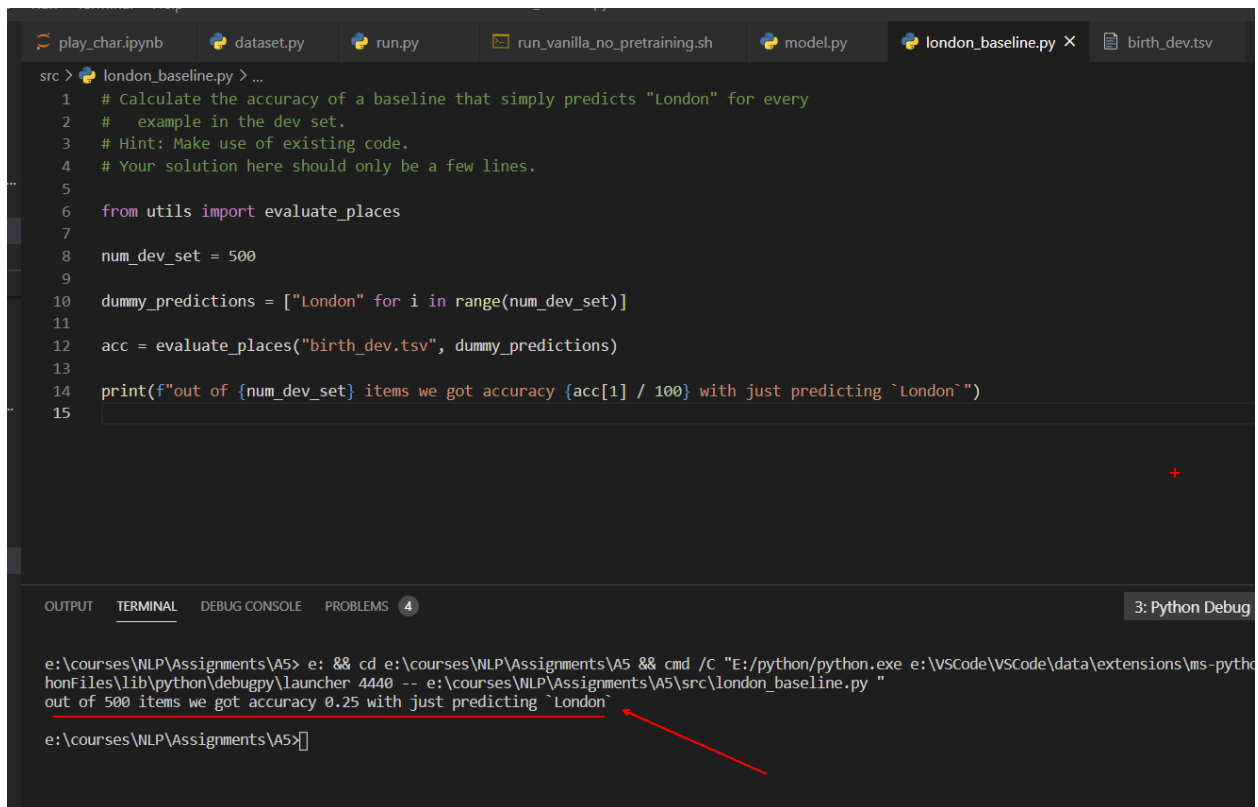
d)

after complete *TODO* code I received accuracy of :

```
epoch 75 iter 7: train loss 0.21425. lr 5.100024e-04: 100% 8/8
2023-04-21 11:58:21.715694: I tensorflow/core/platform/cpu_feat
To enable the following instructions: AVX2 FMA, in other operat
2023-04-21 11:58:22.573575: W tensorflow/compiler/tf2tensorrt/ut
data has 418352 characters, 256 unique.
number of parameters: 3323392
500it [00:55, 9.08it/s]
Correct: 10.0 out of 500.0: 2.0%
2023-04-21 11:59:25.474786: I tensorflow/core/platform/cpu_feat
To enable the following instructions: AVX2 FMA, in other operat
2023-04-21 11:59:26.340861: W tensorflow/compiler/tf2tensorrt/ut
data has 418352 characters, 256 unique.
number of parameters: 3323392
437it [00:48, 8.97it/s]
```

It's 2% which is highly low.

So I just make dummy prediction with London for each evaluation dataset I got following result :



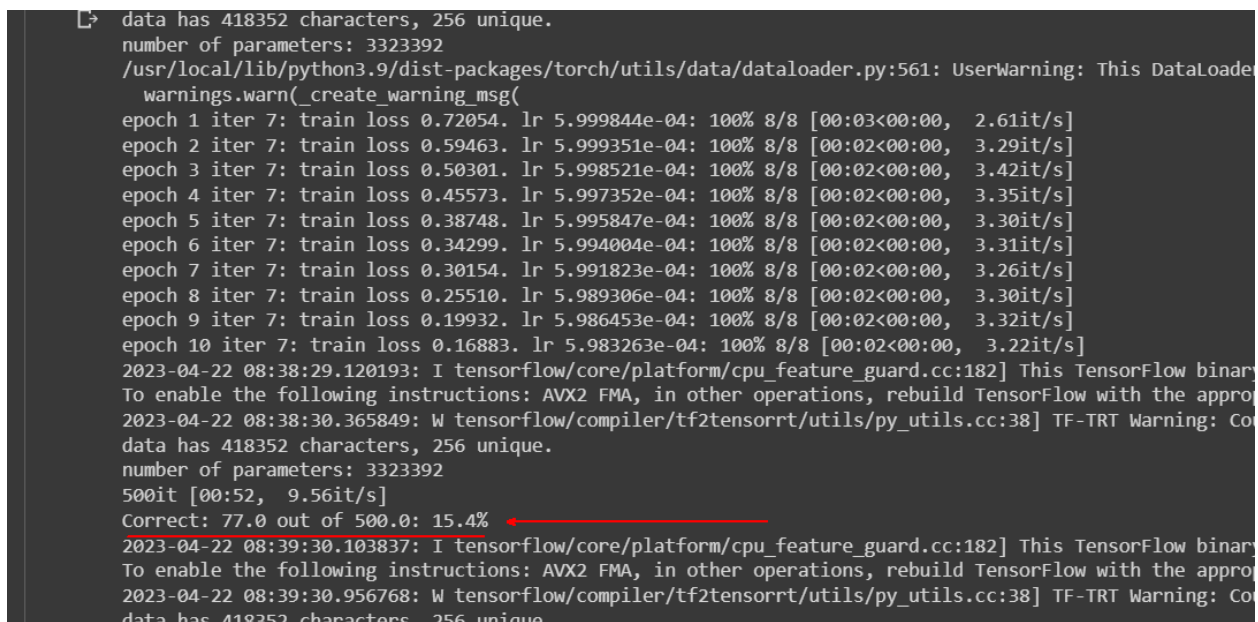
The screenshot shows a VS Code editor with a file explorer at the top containing files like `play_char.ipynb`, `dataset.py`, `run.py`, `run_vanilla_no_pretraining.sh`, `model.py`, `london_baseline.py`, and `birth_dev.tsv`. The main editor window displays the `london_baseline.py` script. The script's purpose is to calculate the accuracy of a baseline model that always predicts "London" for a development set. It imports `evaluate_places` from `utils`, sets `num_dev_set = 500`, creates `dummy_predictions` as a list of 500 "London" strings, and then evaluates the accuracy against `birth_dev.tsv`. The final print statement shows an accuracy of 0.25.

```
src > london_baseline.py > ...
1 # Calculate the accuracy of a baseline that simply predicts "London" for every
2 # example in the dev set.
3 # Hint: Make use of existing code.
4 # Your solution here should only be a few lines.
5
6 from utils import evaluate_places
7
8 num_dev_set = 500
9
10 dummy_predictions = ["London" for i in range(num_dev_set)]
11
12 acc = evaluate_places("birth_dev.tsv", dummy_predictions)
13
14 print(f"out of {num_dev_set} items we got accuracy {acc[1] / 100} with just predicting `London`")
15
```

The terminal at the bottom shows the command used to run the script: `cd e:\courses\NLP\Assignments\A5 && cmd /C "E:/python/python.exe e:\VSCode\VSCode\data\extensions\ms-python\honFiles\lib\python\debugpy\launcher 4440 -- e:\courses\NLP\Assignments\A5\src\london_baseline.py"`. The output confirms the accuracy: `out of 500 items we got accuracy 0.25 with just predicting `London``. A red arrow points from this output line to the corresponding print statement in the code above.

f)

after doing pretraining on *wiki.txt* we have achieved following result:



The terminal output shows the results of pretraining on `wiki.txt`. It starts with a warning from PyTorch's DataLoader and then displays training progress for 10 epochs. Each epoch shows 7 iterations of training loss and learning rate. The final line of the pretraining output, `Correct: 77.0 out of 500.0: 15.4%`, is highlighted with a red arrow. Following this, there are more TensorFlow warnings and the start of a new section.

```
data has 418352 characters, 256 unique.
number of parameters: 3323392
/usr/local/lib/python3.9/dist-packages/torch/utils/data/dataloader.py:561: UserWarning: This DataLoader
warnings.warn(_create_warning_msg(
epoch 1 iter 7: train loss 0.72054. lr 5.999844e-04: 100% 8/8 [00:03<00:00, 2.61it/s]
epoch 2 iter 7: train loss 0.59463. lr 5.999351e-04: 100% 8/8 [00:02<00:00, 3.29it/s]
epoch 3 iter 7: train loss 0.50301. lr 5.998521e-04: 100% 8/8 [00:02<00:00, 3.42it/s]
epoch 4 iter 7: train loss 0.45573. lr 5.997352e-04: 100% 8/8 [00:02<00:00, 3.35it/s]
epoch 5 iter 7: train loss 0.38748. lr 5.995847e-04: 100% 8/8 [00:02<00:00, 3.30it/s]
epoch 6 iter 7: train loss 0.34299. lr 5.994004e-04: 100% 8/8 [00:02<00:00, 3.31it/s]
epoch 7 iter 7: train loss 0.30154. lr 5.991823e-04: 100% 8/8 [00:02<00:00, 3.26it/s]
epoch 8 iter 7: train loss 0.25510. lr 5.989306e-04: 100% 8/8 [00:02<00:00, 3.30it/s]
epoch 9 iter 7: train loss 0.19932. lr 5.986453e-04: 100% 8/8 [00:02<00:00, 3.32it/s]
epoch 10 iter 7: train loss 0.16883. lr 5.983263e-04: 100% 8/8 [00:02<00:00, 3.22it/s]
2023-04-22 08:38:29.120193: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appro
2023-04-22 08:38:30.365849: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Co
data has 418352 characters, 256 unique.
number of parameters: 3323392
500it [00:52, 9.56it/s]
Correct: 77.0 out of 500.0: 15.4%
2023-04-22 08:39:30.103837: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appro
2023-04-22 08:39:30.956768: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Co
data has 418352 characters, 256 unique
```

g)

i)

```
data has 418352 characters, 256 unique.
64
number of parameters: 3339776
500it [00:57, 8.64it/s]
Correct: 43.0 out of 500.0: 8.6%
2023-06-01 20:21:20.538854: I tensorflow/core/platform/cpu_featu
To enable the following instructions: AVX2 FMA, in other operati
2023-06-01 20:21:21.470328: W tensorflow/compiler/tf2tensorrt/ut
data has 418352 characters, 256 unique.
64
number of parameters: 3339776
437it [00:50, 8.65it/s]
No gold birth places provided; returning (0,0)
```

ii)

first let's see computational cost of main transformer(vanilla):

- Self-attention: $\ell^2.d \rightarrow \ell$ is : sequence length , d is : number of dimension
- Attention head: $\ell^2.d.h \rightarrow h$ is : number of heads
- Feed forward: $\ell.d.f \rightarrow f$ is : hidden size
- Number of transformer layer: $L(\ell^2.d.h + \ell.d.f)$

Now let's compare it with Perceiver model:

first and last layer of this model use down Projection and up Projection respectively.

so L becomes $(L - 2). (m^2.d.h + m.d.f) + \text{down projection complexity} + \text{up projection complexity}(m \text{ is bottleneck_dim}).$

Notice that ℓ becomes m because we're doing our computation on lower dimensionality.

$\text{down projection complexity} = \ell.m.d.h + m.d.f$

$\text{up projection complexity} = \ell.m.d.h + \ell.d.f$

Perceiver complexity : $(L - 2). (m^2.d.h + m.d.f) + 2.\ell.m.d.h + d.f(\ell + m)$

3.

a.

pretraining is important phase for our model to become more general and see more common data and casual text on open domain. As we see we trained our model in pretraining phase on 600 epochs to learn whatever it must learn between text correlation and understand the concept. And we can customize our model by fine tuning on our specific task which leads in a far better result.

b.

Information misleading in important situation : our models keep generating answers that we don't know even its correct or not and since it doesn't provide any explanation that how it produced that we have no idea and consider that as correct birth place.

Example : imagine a lecturer who is lecturing about famous people and taking their birth place data from our app when he lectures with some wrong places people will doubt about his/her knowledge and don't pay attention to him.

Losing Trust: if the system frequently presents users with incorrect or unreliable birth places, it can erode trust in the system's overall accuracy and credibility. Users may become suspicious of the system's output and lose confidence in its ability to provide accurate information.

Example: consider an voice assistant which uses our model app api for birth places. When people start to ask birth places of people our voice assistant uses api and then generate fake birth place. And people start to lose trust in our voice assistant and this may lead to bankruptcy of our investment on our voice assistant.

c.

one possible way is that our model analyzes test example and see its embedding tokenization and then map our test example question to the question in training set sample which is most similar and then use that output for test example. Or it may produce some response which is just following general patterns seen at training time or fine-tuning and has some meaning but the answer isn't close to the real answer. And as the last possibility if the test provided isn't close even a little to our training data it may generate some random and non-sense answers without any meaning.

concerns : the possibilities that has been mentioned may cause our app to lose member because making non-sense answers without any explanations. And can be very embarrassing when we're using our app for official paper and this may cause bad reputation for our company.