1

a) if $q = k_j$ and $q \perp k_i$ $(i \neq j)$

then $c \approx v_j$

(b) $q = M(k_a + k_b)$

M → big$^+$ number

$$C = \frac{e^{M(k_a+k_b)k_a} v_a + e^{M(k_a+k_b)k_b} v_b + v_c + \cdots}{e^{M(k_a+k_b)k_a} + e^{M(k_a+k_b)k_b} + 1 + 1 + \cdots}$$

$$\approx \frac{e^{M\overbrace{\|k_a\|^2}^{\to 1}} v_a + e^{M\overbrace{\|k_b\|^2}^{\to 1}} v_b}{e^{M\underbrace{\|k_a\|^2}_{\to 1}} + e^{M\underbrace{\|k_b\|^2}_{\to 1}}}$$

$$= \frac{e^{M}(v_a + v_b)}{e^{M}(1+1)} = \frac{v_a + v_b}{2}$$

c

Intuitively, we can assume that the following statements are roughly true:

- If matrices A and B are both from a normal distributions with the mean μ and $\Sigma_i = \alpha I$ (like mentioned in the question) then their dot product will approximately equal to $||\mu||^2$. (Because A and B roughly point to the same direction and they have roughly the same norm.)
- If matrices A and B are from two normal distributions with the means μ_1 and μ_2 and $\mu_1 \perp \mu_2$ and $\Sigma_i = \alpha I$ for both (like mentioned in the question) then their dot product will approximately equal to 0. (Because A and B roughly point to the same direction as μ_1 and μ_2 hence A is roughly perpendicular to B)

i.

$$q = M(\mu_a + \mu_b)$$

$$c = \frac{e^{M(\mu_a+\mu_b)k_a} v_a + e^{M(\mu_a+\mu_b)k_b} v_b + e^{M(\mu_a+\mu_b)k_c} v_c + \dots}{e^{M(\mu_a+\mu_b)k_a} + e^{M(\mu_a+\mu_b)k_b} + e^{M(\mu_a+\mu_b)k_c} + \dots}$$

$$\simeq \frac{e^{M||\mu||^2} v_a + e^{M||\mu||^2} v_b + v_c + \dots}{e^{M||\mu||^2} + e^{M||\mu||^2} + 1 + \dots}$$

$$\simeq \tfrac{1}{2}(v_a + v_b)$$

The terms "$k_i^T.\mu_a$", "$k_i^T.\mu_b$" and hence "$\exp(\log(M)(k_i^T.\mu_a + k_i^T.\mu_b)).v_i$" are approximately zero according to the above statements.

ii. In this case the term “$k_a^T.\mu_a$” will vary roughly between 1/2 and 3/2. Thus, c will have a value roughly between $(v_a+2v_b)/3$ and $(3v_a+v_b)/5$. In the first extreme case c will lean towards v_b and in the second case it will lean towards v_a. This phenomenon shows an inconsistency between different samplings of k_i.

(d)

(i)

$$q_1 = M\mu_a$$

$$q_2 = M\mu_b$$

$$c_1 = \frac{e^{M(k_a^T \mu_a)} v_a + e^{M(k_b^T \mu_a)} v_b + \dots + e^{M(k_i^T \mu_a)} v_i + \dots}{e^{M(k_a^T \mu_a)} + e^{M(k_b^T \mu_a)} + \dots + e^{M(k_i^T \mu_a)} + \dots}$$

$$\approx \frac{e^{M\|\mu_a\|^2} v_a}{e^{M\|\mu_a\|^2}}$$

$$= v_a$$

$$c_2 = \frac{e^{M(k_a^T \mu_b)} v_a + e^{M(k_b^T \mu_b)} v_b + \dots + e^{M(k_i^T \mu_b)} v_i + \dots}{e^{M(k_a^T \mu_b)} + e^{M(k_b^T \mu_b)} + \dots + e^{M(k_i^T \mu_b)} + \dots}$$

$$\approx \frac{e^{M\|\mu_b\|^2} v_b}{e^{M\|\mu_b\|^2}}$$

$$= v_b$$

$$c = \frac{c_1 + c_2}{2} = \frac{v_a + v_b}{2}$$

ii

$$c_1 \approx v_a$$
$$c_2 \approx v_b \Rightarrow c \approx \frac{v_a + v_b}{2}$$

ii. In this case always c_1 equals to v_a approximately and c_2 approximately equals to v_b because all other terms are approximately zero and the remaining coefficients in the numerator and denominator cancel out. In other words, variance in the magnitude of k_a, unlike part (c), does not have much effect.

e

i $q_2 = u_a$

$$c_2 = \frac{e^{k_1^T q_2} v_1 + e^{k_2^T q_2} v_2 + e^{k_3^T q_2} v_3}{e^{k_1^T q_2} + e^{k_2^T q_2} + e^{k_3^T q_2}}$$

$$= \frac{v_1 + e^{\|u_a\|^2} v_2 + v_3}{2 + e^{\|u_a\|^2}}$$

$$\approx V_2 = u_a$$

No, because x_1 and x_3 have the terms u_d or u_c and hence the softmax of $k_1^T q_2$ or $k_3^T q_2$ will not be near zero.

(ii)

$$v_1 = u_b = V x_1 = V(u_b + u_d) \rightarrow V \text{ should have } \frac{1}{\|u_b\|^2} u_b u_b^T$$
$$v_3 = u_b - u_c = V x_3 = V(u_b + u_c) \rightarrow V \text{ should have } \frac{-1}{\|u_c\|^2} u_c u_c^T$$

$$V = \frac{1}{\beta^2}\left(u_b u_b^T - u_c u_c^T\right)$$

$$c_2 = u_b \rightarrow \frac{e^{\overbrace{k_1^T q_2}} v_1 + e^{\overbrace{k_2^T q_2}^{0 \,\rightarrow\, k_2 \perp q_2}} v_2 + e^{\overbrace{k_3^T q_2}^{0 \,\rightarrow\, k_3 \perp q_2}} v_3}{e^{k_1^T q_2} + e^{k_2^T q_2} + e^{k_3^T q_2}} = v_1 = u_b$$

$$c_1 = u_b - u_c \rightarrow \frac{e^{\overbrace{k_1^T q_1}^{0 \,\rightarrow\, k_1 \perp q_1}} v_1 + e^{\overbrace{k_2^T q_1}^{0 \,\rightarrow\, k_2 \perp q_1}} v_2 + e^{k_3^T q_1} v_3}{e^{k_1^T q_1} + e^{k_2^T q_1} + e^{k_3^T q_1}} = v_3 = u_b - u_c$$

*

*

$$k_i = x_i \rightarrow K = I$$

$$q_1 = u_c = Q x_1 = Q(u_b + u_d) \rightarrow Q \text{ should have } \frac{1}{\beta^2}(u_c u_d^T)$$
$$q_2 = u_d = Q x_2 = Q u_a \rightarrow Q \text{ should have } \frac{1}{\beta^2}(u_d u_a^T)$$
$$q_3 = \vec{0}$$

**

** $$Q = \frac{1}{\beta^2}\left(u_c u_d^T + u_d u_a^T\right)$$

2 c & d

```
epoch 64 iter 7: train loss 0.34248. lr 5.335229e-04: 100%|██████████| 8/8 [00:04<00:00,  1.68it/s]
epoch 65 iter 7: train loss 0.32717. lr 5.315115e-04: 100%|██████████| 8/8 [00:04<00:00,  1.66it/s]
epoch 66 iter 7: train loss 0.30116. lr 5.294740e-04: 100%|██████████| 8/8 [00:04<00:00,  1.67it/s]
epoch 67 iter 7: train loss 0.30572. lr 5.274107e-04: 100%|██████████| 8/8 [00:04<00:00,  1.68it/s]
epoch 68 iter 7: train loss 0.29130. lr 5.253217e-04: 100%|██████████| 8/8 [00:04<00:00,  1.68it/s]
epoch 69 iter 7: train loss 0.27051. lr 5.232074e-04: 100%|██████████| 8/8 [00:04<00:00,  1.67it/s]
epoch 70 iter 7: train loss 0.25012. lr 5.210680e-04: 100%|██████████| 8/8 [00:04<00:00,  1.68it/s]
epoch 71 iter 7: train loss 0.23997. lr 5.189037e-04: 100%|██████████| 8/8 [00:04<00:00,  1.67it/s]
epoch 72 iter 7: train loss 0.23392. lr 5.167147e-04: 100%|██████████| 8/8 [00:04<00:00,  1.68it/s]
epoch 73 iter 7: train loss 0.22486. lr 5.145014e-04: 100%|██████████| 8/8 [00:04<00:00,  1.66it/s]
epoch 74 iter 7: train loss 0.21690. lr 5.122639e-04: 100%|██████████| 8/8 [00:04<00:00,  1.67it/s]
epoch 75 iter 7: train loss 0.20826. lr 5.100024e-04: 100%|██████████| 8/8 [00:04<00:00,  1.67it/s]
```

```
data has 418352 characters, 256 unique.
number of parameters: 3323392
number of parameters: 3323392
500it [00:41, 12.01it/s]
Correct: 9.0 out of 500.0: 1.7999999999999998%
```

dev acc

```
data has 418352 characters, 256 unique.
number of parameters: 3323392
number of parameters: 3323392
437it [00:42, 10.31it/s]
No gold birth places provided; returning (0,0)
Predictions written to vanilla.nopretrain.test.predictions; no targets provided
```

```
data has 418352 characters, 256 unique.
number of parameters: 3323392
437it [00:35, 12.27it/s]
Correct: 63 out of 437: 14.416475972540047%
```

london baseline

f

2 f (1)

```
epoch 642 iter 22: train loss 0.47495. lr 6.000000e-04: 100%|██████████| 23/23 [00:07<00:00,  3.04it/s]
epoch 643 iter 22: train loss 0.50243. lr 6.000000e-04: 100%|██████████| 23/23 [00:07<00:00,  3.04it/s]
epoch 644 iter 22: train loss 0.47090. lr 6.000000e-04: 100%|██████████| 23/23 [00:07<00:00,  3.04it/s]
epoch 645 iter 22: train loss 0.45273. lr 6.000000e-04: 100%|██████████| 23/23 [00:07<00:00,  3.04it/s]
epoch 646 iter 22: train loss 0.45603. lr 6.013186e-04: 100%|██████████| 23/23 [00:07<00:00,  3.05it/s]
epoch 647 iter 22: train loss 0.49273. lr 6.296935e-04: 100%|██████████| 23/23 [00:07<00:00,  3.04it/s]
epoch 648 iter 22: train loss 0.46037. lr 6.586443e-04: 100%|██████████| 23/23 [00:07<00:00,  3.04it/s]
epoch 649 iter 22: train loss 0.49022. lr 6.881640e-04: 100%|██████████| 23/23 [00:07<00:00,  3.04it/s]
epoch 650 iter 22: train loss 0.48776. lr 7.182453e-04: 100%|██████████| 23/23 [00:07<00:00,  3.04it/s]
```

2 f (2)

[redacted]

```
data has 418352 characters, 256 unique.
number of parameters: 3323392
number of parameters: 3323392
epoch 1 iter 7: train loss 0.71179. lr 5.999844e-04: 100%|██████████| 8/8 [00:04<00:00,  1.70it/s]
epoch 2 iter 7: train loss 0.56339. lr 5.999351e-04: 100%|██████████| 8/8 [00:04<00:00,  1.80it/s]
epoch 3 iter 7: train loss 0.45983. lr 5.998521e-04: 100%|██████████| 8/8 [00:04<00:00,  1.79it/s]
epoch 4 iter 7: train loss 0.40272. lr 5.997352e-04: 100%|██████████| 8/8 [00:04<00:00,  1.79it/s]
epoch 5 iter 7: train loss 0.33338. lr 5.995847e-04: 100%|██████████| 8/8 [00:04<00:00,  1.74it/s]
epoch 6 iter 7: train loss 0.29572. lr 5.994004e-04: 100%|██████████| 8/8 [00:04<00:00,  1.66it/s]
epoch 7 iter 7: train loss 0.22406. lr 5.991823e-04: 100%|██████████| 8/8 [00:04<00:00,  1.77it/s]
epoch 8 iter 7: train loss 0.18956. lr 5.989306e-04: 100%|██████████| 8/8 [00:04<00:00,  1.76it/s]
epoch 9 iter 7: train loss 0.15669. lr 5.986453e-04: 100%|██████████| 8/8 [00:04<00:00,  1.76it/s]
epoch 10 iter 7: train loss 0.12842. lr 5.983263e-04: 100%|██████████| 8/8 [00:04<00:00,  1.76it/s]
```

2 f (3)

[redacted]

```
data has 418352 characters, 256 unique.
number of parameters: 3323392
number of parameters: 3323392
500it [00:39, 12.69it/s]
Correct: 137.0 out of 500.0: 27.400000000000002%
```

← dev acc

2 f (4)

[redacted]

```
data has 418352 characters, 256 unique.
number of parameters: 3323392
number of parameters: 3323392
437it [00:40, 10.77it/s]
No gold birth places provided; returning (0,0)
Predictions written to vanilla.pretrain.test.predictions; no targets provided
```

g

$$\underbrace{Y_i}_{l\times d/h} = \underbrace{\mathrm{softmax}\Big(\overbrace{\underbrace{\mathrm{ReLU}(\underbrace{\overbrace{X}^{l\times d}\overbrace{A_i}^{d\times d/h}}_{l\times d/h} + b_1)}_{l\times d/h}\underbrace{B_i}_{d/h\times l}}^{l\times l} + b_2\Big)}_{l\times l}\big(\underbrace{\overbrace{X}^{l\times d}\overbrace{V_i}^{d\times d/h}}_{l\times d/h}\big),$$

2 g (1)

```
epoch 637 iter 22: train loss 0.58126. lr 6.000000e-04: 100%|██████████| 23/23 [00:07<00:00,  3.05it/s]
epoch 638 iter 22: train loss 0.53698. lr 6.000000e-04: 100%|██████████| 23/23 [00:07<00:00,  3.06it/s]
epoch 639 iter 22: train loss 0.60296. lr 6.000000e-04: 100%|██████████| 23/23 [00:07<00:00,  3.07it/s]
epoch 640 iter 22: train loss 0.56730. lr 6.000000e-04: 100%|██████████| 23/23 [00:07<00:00,  3.06it/s]
epoch 641 iter 22: train loss 0.58239. lr 6.000000e-04: 100%|██████████| 23/23 [00:07<00:00,  3.06it/s]
epoch 642 iter 22: train loss 0.57320. lr 6.000000e-04: 100%|██████████| 23/23 [00:07<00:00,  3.07it/s]
epoch 643 iter 22: train loss 0.58172. lr 6.000000e-04: 100%|██████████| 23/23 [00:07<00:00,  3.06it/s]
epoch 644 iter 22: train loss 0.55105. lr 6.000000e-04: 100%|██████████| 23/23 [00:07<00:00,  3.06it/s]
epoch 645 iter 22: train loss 0.56613. lr 6.000000e-04: 100%|██████████| 23/23 [00:07<00:00,  3.06it/s]
epoch 646 iter 22: train loss 0.57172. lr 6.013186e-04: 100%|██████████| 23/23 [00:07<00:00,  3.07it/s]
epoch 647 iter 22: train loss 0.58965. lr 6.296935e-04: 100%|██████████| 23/23 [00:07<00:00,  3.06it/s]
epoch 648 iter 22: train loss 0.56178. lr 6.586443e-04: 100%|██████████| 23/23 [00:07<00:00,  3.05it/s]
epoch 649 iter 22: train loss 0.57799. lr 6.881640e-04: 100%|██████████| 23/23 [00:07<00:00,  3.06it/s]
epoch 650 iter 22: train loss 0.53707. lr 7.182453e-04: 100%|██████████| 23/23 [00:07<00:00,  3.07it/s]
```

2 g (2)

[redacted]

```
data has 418352 characters, 256 unique.
number of parameters: 3076988
epoch 1 iter 7: train loss 0.82974. lr 5.999844e-04: 100%|██████████| 8/8 [00:04<00:00,  1.67it/s]
epoch 2 iter 7: train loss 0.68206. lr 5.999351e-04: 100%|██████████| 8/8 [00:04<00:00,  1.76it/s]
epoch 3 iter 7: train loss 0.62294. lr 5.998521e-04: 100%|██████████| 8/8 [00:04<00:00,  1.75it/s]
epoch 4 iter 7: train loss 0.57534. lr 5.997352e-04: 100%|██████████| 8/8 [00:04<00:00,  1.76it/s]
epoch 5 iter 7: train loss 0.54958. lr 5.995847e-04: 100%|██████████| 8/8 [00:04<00:00,  1.75it/s]
epoch 6 iter 7: train loss 0.52497. lr 5.994004e-04: 100%|██████████| 8/8 [00:04<00:00,  1.74it/s]
epoch 7 iter 7: train loss 0.47789. lr 5.991823e-04: 100%|██████████| 8/8 [00:04<00:00,  1.74it/s]
epoch 8 iter 7: train loss 0.43871. lr 5.989306e-04: 100%|██████████| 8/8 [00:04<00:00,  1.73it/s]
epoch 9 iter 7: train loss 0.40346. lr 5.986453e-04: 100%|██████████| 8/8 [00:04<00:00,  1.73it/s]
epoch 10 iter 7: train loss 0.35365. lr 5.983263e-04: 100%|██████████| 8/8 [00:04<00:00,  1.73it/s]
```

2 g (3)

```
data has 418352 characters, 256 unique.
number of parameters: 3076988
500it [00:39, 12.50it/s]
Correct: 40.0 out of 500.0: 8.0%
```

← dev acc

2 g (4)

```
data has 418352 characters, 256 unique.
number of parameters: 3076988
437it [00:35, 12.28it/s]
No gold birth places provided; returning (0,0)
Predictions written to synthesizer.pretrain.test.predictions; no targets provided
```

ii

The synthesizer method does not take contextual and relative similarity of tokens into consideration. (Does not have a mechanism similar to Q and K)

3

i

The pretrained model captured low level information about the data using the corrupted span strategy.

ii

1. Models of this kind which can output some wrong results that seem very similar to a true result are sometimes really hard to debug.

2. They can be misleading and sometimes dangerous if they are used in an important decision making process.

3. They may learn unwanted biases.

iii

The model may learn a correlation between a name (and it's similar forms) and a location which is not always true. In such cases the model can use this similarity to make up a somewhat realistic birthplace if a name similar (not identical) to a previously learnt name is given.

In some use cases it may raise a subtle issue: information leakage. For example consider an application in which a user X has the authority to query about a name A but does not have permission to query about a name B which is very similar to A. This type of model can leak some info about B to the user X.