[CS224N Winter 2021] Assignment 5: Self-Attention, Transformers, and Pretraining

1. Attention exploration (21 points)

1. Attention Exploration

$\boxed{A.}$ Suppose we want $c = v_j$ for some $j \in \{1, \ldots, n\}$

This means that $v_j = \sum_{i=1}^{n} v_i \alpha_i$ which means that v_j needs to be a linear combination of the value vectors $\{v_1, \ldots, v_n\} = V$. The simplest case of this is whe $\alpha_j = 1$ and $\alpha_i = 0$ for $i \neq j$. This is approximately achieved when $k_j^T q >> k_i q$ for $i \neq j$. This will often be the case when q is "similair" to k_j (both having large vector entries in the same direction):

$$v_{jk} \approx q_k$$

$$|v_{jk}| \approx |q_k| >> 0$$

$\boxed{B.}$ Two key vectors k_i and k_j are said to be perpendicular, $k_i \perp k_j$, if $k_i^T k_j = 0$. The length, norm, or magnitude of a key vector k_i, $||k_i|| = \sqrt{k_{i0}^2 + k_{i1}^2 + \ldots + k_{in}^2}$.

Suppose we have key vectors $k = \{k_1, \ldots, k_n\}$ such that $k_i \perp k_j$ for $i \neq j$ and $||k_i|| = 1$ for all

i. Suppose we have $V = \{v_1, \dots, v_n\}$ and our objective is to find a value q such that

$$c \approx \frac{1}{2}(v_a + v_b)$$

for arbitrary $a, b \in \{1, \dots, n\}$. By the definition of c we get

$$\frac{1}{2}(v_a + v_b) \approx \sum_{i=1}^{i=n} v_i \alpha_i \qquad \alpha_i = \frac{e^{k_i q}}{\sum_{j=1}^{j=n} e^{k_j^T q}}$$

Combining this all together we get that

$$\frac{1}{2}(v_a + v_b) \approx \sum_{i=1}^{i=n} \frac{v_i e^{k_i q}}{\sum_{j=1}^{j=n} e^{k_j^T q}}$$

Let $q = c\,(k_a + k_b)$. Then we have

$$k_i^T q = \frac{1}{2}(k_i^T k_a + k_i^T k_b) \quad \text{for all } i$$

when $i \neq a, b$ then we have

$$k_i^T q = \frac{1}{2}(0 + 0) \quad \text{as} \quad \begin{matrix} k_i \perp k_a \\ k_i \perp k_b \end{matrix}$$

when $i = a$ we have

$$k_a^T q = \frac{1}{2}(k_a^T k_a + 0)$$

Because $\|k_a\| = 1$, we know that $k_a^T k_a = 1$:

$$1 = \sqrt{k_{i1}^2 + \dots k_{in}^2}$$

$$1 = k_{i1}^2 + \dots k_{in}^2$$

Therefore $k_a^T q = \gamma/2 \cdot 1 = \gamma/2$.

Similarity $k_b^T q = \gamma/2$

Therefore $\alpha_i = \begin{cases} \frac{e^0}{n-2+2e^{\gamma/2}} = \frac{1}{2e^{\gamma/2}+n-2} \\ \end{cases}$

Lemma 1) If $\|v\| = 1$, then that implies $v^T v = 1$.

Proof:

$$\|v\| = 1$$

$$\sqrt{v_1^2 + \dots + v_n^2} = 1$$

$$v_1^2 + \dots + v_n^2 = 1$$

$$v^T v = 1$$

Proof

Let $q = \hat{c}(k_a + k_b)$ where $\hat{c}$ is a very large constant: $\hat{c} \Rightarrow \infty$. First, let's compute the attention scores α with this choice of q. If $i \neq a,b$ then we have

$$\begin{aligned} k_i^T q &= k_i^T \hat{c}(k_a + k_b) \\ &= \hat{c}(k_i^T k_a + k_i^T k_b) \\ &= \hat{c}(0+0) \qquad \text{as } \begin{matrix} k_i \perp k_a \\ k_i \perp k_b \end{matrix} \\ &= 0 \end{aligned}$$

If $i = a$ we have

$$\begin{aligned} k_a^T q &= k_a^T \hat{c}(k_a + k_b) \\ &= \hat{c}(k_a^T k_a + k_a^T k_b) \\ &= \hat{c}\, k_a^T k_a \qquad \text{as } k_a \perp k_b \\ &= \hat{c} \end{aligned}$$

Similarily if $i = b$ we have

$$\begin{aligned} k_b^T q &= k_b^T \hat{c}(k_a + k_b) \\ &= \hat{c}\, k_b^T k_b \qquad \text{as } k_a \perp k_b \\ &= \hat{c} \end{aligned}$$

Therefore using this we get that the quantity

$$\sum_{j=1}^{j=n} e^{k_i^T q} = e^{\hat{c}} + e^{\hat{c}} = 2e^{\hat{c}}$$

Now we can compute the attention scores α:

$$\alpha_i = \begin{cases} \frac{e^{0}}{2e^{c^{*}}} & \text{if } i \neq a,b \\ \frac{e^{c^{*}}}{2e^{c^{*}}} & \text{if } i = a,b \end{cases}$$

$$\alpha_i \approx \begin{cases} 0 & \text{if } i \neq a,b \\ 1/2 & \text{if } i = a,b \end{cases}$$

Therefore this gives us the following

$$c = \sum_{i=1}^{n} v_i \alpha_i \approx \frac{1}{2} v_a + \frac{1}{2} v_b = \frac{1}{2}(v_a + v_b)$$

as desired.

$\boxed{C}$

I. Suppose we have $K = \{k_1, \ldots, k_n\}$ where:

$$k_i \sim N(u_i, \Sigma_i) \quad u_i \in \mathbb{R}^n \quad \Sigma_i \in \mathbb{R}^{n^2}$$

and u_i are known but Σ_i are not. Also suppose that:

$$u_i^T u_j = 0 \text{ if } i \neq j \text{ and } \|u_i\| = 1$$

meaning that all u_i are perpendicular to each other and have unit norm. Also assume that the covariance matrices are of the form:

$$\Sigma_i = \alpha I \qquad \alpha \Rightarrow 0$$

As α approches 0, k_i appoaches the mean vectors

u_i as $k_i \sim N(u_i, \Sigma_i \Rightarrow 0)$. Therefore, if we want $c = \frac{1}{2}(v_a + v_b)$, then we can let:

$$q = c^* (k_a + k_b) \quad c^* >> 0$$

using the results from above as:

$k_i \perp k_j \; i \neq j$ because $k_i \sim u_i$ and $u_i \perp u_j \; i \neq j$

$\|k_i\| = 1$ because $k_i \sim u_i$ and $\|u_i\| = 1$

Because $k_a \sim u_a$ and $k_b \sim u_b$ this reduces to

$$q = c^* (u_a + u_b)$$

as desired.

II Suppose that $\Sigma_a = \alpha I + \frac{1}{2}(u_a u_a^T)$. Because $\|u_a\| = 1$, we have

$$\|u_a\| = 1$$
$$\sqrt{u_{a_1}^2 + \dots + u_{a_d}^2} = 1$$
$$u_{a_1}^2 + \dots + u_{a_d}^2 = 1$$
$$u_a u_a^T = 1$$

This means that as $\alpha \Rightarrow 0$ $\Sigma_a = \frac{1}{2} I$. Now

$$k_a \sim N(u_a, \tfrac{1}{2} I) \quad k_i \sim (u_i, 0) \; i \neq a$$

That means that depending on the sample,

$\|k_a\| > 1$ or $\|k_a\| < 1$ while $\|k_i\| \sim 1$ $i \in a$.

Eq. 1 are equally likely. This means that

$$\begin{cases} \alpha_a > \alpha_b & \text{when } \|k_a\| > 1 \\ \alpha_a < \alpha_b & \text{when } \|k_a\| < 1 \\ \alpha_i = 0 & \text{when } i \neq a, b \end{cases}$$

This means that our output c will look more like v_a when $\|k_a\| > 1$ and more like v_b when $\|k_a\| < 1$.

$\boxed{D}$ We will define q_1 and q_2 such that $c = \frac{1}{2}(v_a + v_b)$ with the added condition that $q_1 \neq q_2$. In order to do this we can do the following:

$$q_1 = c^{*} u_a \qquad c^{*} \gg 0$$

$$q_2 = c^{**} u_b \qquad c^{**} \gg 0$$

If we do that then we have:

$$\alpha_i^1 = \begin{cases} 1 & i = a \\ 0 & i \neq a \end{cases} \qquad \alpha_i^2 = \begin{cases} 1 & i = b \\ 0 & i \neq b \end{cases}$$

This gives us

$$c_1 = v_a \qquad c_2 = v_b$$

$$c = \frac{1}{2}(c_1 + c_2) = \frac{1}{2}(v_a + v_b)$$

II Suppose we have a sample of $k = \{k_1, \dots, k_n\}$

Then we have:

$$k_a = N(\mu_1, \tfrac{1}{2}) \quad k_i = N(\mu_1, 0) \quad i \neq a$$

This means that

$$\alpha_i^1 = \begin{cases} 1 & \text{when } i = a \\ 0 & i \neq a \end{cases}$$

$$\alpha_i^2 = \begin{cases} 1 & \text{when } i = b \\ 0 & i \neq b \end{cases}$$

These are regardless of the variation of the norm $\|k_a\|$ from sampling. Thus

$$c_1 = v_a$$
$$c_2 = v_b$$

And $c = \frac{1}{2}(c_1 + c_2) = \frac{1}{2}(v_a + v_b)$. The key here is that the pertrubation on k_a had no effect on the resulting attention scores since we had multiple queries.

Extended elaboration: The takeaway here is that multiheaded attention allows the network to construct independent queries and take a even average of those individual attention output. This helps the network out when the scale of keys is slightly different.

$\boxed{E}$ If we perform self-attention then we get that

$$c_2 = \sum_{j=1}^{n} \alpha_{2j} v_j$$

In order to compute this we first compute the required quantities:

$$q_2 = x_2 = u_a$$

$$k_1 q_2 = (u_d + u_b)(u_a) = 0 \qquad k_2 q_2 = (u_a)(u_a) = \beta$$

$$k_3 q_2 = (u_c + u_b) u_a = 0$$

$$\alpha_{21} = \frac{e^0}{\sum_{j=1}^{n} e^{k_j q_2}} = \frac{1}{e^\beta} = 0$$

$$\alpha_{22} = \frac{e^\beta}{e^\beta} = 1 \quad \alpha_{23} = \frac{e^0}{e^0} = 0$$

Then we have c^2 to be

$$c_2 = \sum_{j=1}^{n} \alpha_{2j} v_j = v_2 = x_2 = u_a$$

So c^2 will approximate u_a. It would not be possible for c_2 to approximate u_b by adding either u_a or u_c. This is because:

1) If we set x_2 to $u_a + u_b$, then $\alpha_2 = (0,1,0)$ as $e^{2B} >> e^{B}$. So c^2 will be $u_a + u_b$

2) If we set x_2 to be $u_a + u_c$, then again $\alpha_2 = (0,1,0)$ as $e^{2B} >> e^{0}$. So c^2 will be $u_a + u_c$

Neither of these are u_b

II First let's try to find V as recommended by the author. Note that $u_a, u_b, u_c, u_d \in R^{d \times 1}$

$$v_1 = V x_1 = V(u_d + u_b) = V u_d + V u_b$$

So we want $Vu_d = 0$ $Vu_b = u_b$

$$v_3 = V x_3 = V(u_c + u_b) = Vu_c + Vu_b$$

So we want $Vu_c = -u_c$ and $Vu_b = u_b$

Let's focus on the first condition $Vu_b = u_b$. Using the hint from the assignment, note that

$$(u_b u_b^T) u_b = u_b \|u_b\|^2$$

So let's set $V = \frac{1}{B} u_b u_b^T$. Thus we have

$$v_1 = V x_1 = \frac{1}{B} u_b u_b^T (u_d + u_b)$$

$$= \frac{1}{B} u_b u_b^T u_b \quad \text{as } u_b u_b^T u_d = 0$$

$$= \frac{1}{B} u_b B = u_b \checkmark$$

Let's do something similair by adding $-\frac{1}{B} u_c u_c^T$ to our choice of V. Then

$$V = \frac{1}{B} u_b u_b^T - \frac{1}{B} u_c u_c^T$$

Then

$$V z = V(u_c + u_b) = Vu_c + Vu_b$$

$$= \left(0 + -\frac{1}{B} u_c u_c^T u\right) + \left(\frac{1}{B} u_b u_b^T u_b + 0\right) \quad \text{by orthogonality}$$

$$= -u_c + u_b \checkmark$$

$$V z = V(u_d + u_b) = 0 + \frac{1}{B} u_b u_b^T = u_b \checkmark$$

So our choice in V is good. Now let's try to find K and Q. We want $c_2 \approx u_b$. That will only occur when $\alpha_2 = (1,0,0)$ given our choice of V. Similarily $c_1 \approx u_b - u_c$ will only occur when $\alpha_1 = (0,0,1)$.

Now we will let Q and K be the following

$$Q = u_d u_a^T + u_c u_d^T \qquad K = I$$

$$q_1 = (u_d u_a^T + u_c u_d^T)(u_d + u_b) = 0 + u_c u_d^T u_d = u_c$$

$$k_1 = Kx_1 = u_d + u_b$$

$$k_2 = Kx_2 = u_a$$

$$k_3 = Kx_3 = u_c + u_b$$

$$\alpha_{11} = \frac{e^{(u_d+u_b)^T u_c}}{e^0} = 0 \quad \alpha_{12} = \frac{e^{u_a^T u_c}}{e^0} = 0 \quad \alpha_{12} = e^{(u_c+u_b)^T u_c} = \frac{e}{e^0} = 1$$

$$\alpha_1 = (0, 0, 1)$$

$$c_1 = v_3 = V(u_c + u_b) = u_b - u_c \checkmark$$

$$q_2 = (u_d u_a^T + u_c u_d^T) u_a = u_d$$

$k_1 = u_d + u_b \quad k_2 = u_a \quad k_3 = u_c + u_b$ this is from the result above

$$\alpha_2 = (1, 0, 0)$$

$$c_2 = v_1 = V(u_d + u_b) = u_b \checkmark$$

2. Pretrained Transformer models and knowledge access (35 points)

D.

On the development test set, I got a 1.401% accuracy score from a model trained from scratch versus a 5% accuracy score from a baseline model that just predicts “London” for each test row.

```
(local_nmt) sh-4.2$ python src/london_baseline.py
500 500
Result if just predicted London:  (500.0, 25.0)
(local_nmt) sh-4.2$ python src/london_baseline.py
Result if just predicted London: correct:25.0, total:500.0: percentage:0.05
(local_nmt) sh-4.2$ 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.pred
ictions
data has 418352 characters, 256 unique.
number of parameters: 3323392
500it [01:03,  7.83it/s]
Correct: 7.0 out of 500.0: 1.4000000000000001%
```

F.

I got a 25.8% accuracy score on the development test set after pretraining and finetuning.

```
epoch 1 iter 7: train loss 0.73257. lr 5.999844e-04: 100%|██████████| 8/8 [00:13<00:00,  1.71s/it]
epoch 2 iter 7: train loss 0.53881. lr 5.999351e-04: 100%|██████████| 8/8 [00:16<00:00,  2.03s/it]
epoch 3 iter 7: train loss 0.44053. lr 5.998521e-04: 100%|██████████| 8/8 [00:17<00:00,  2.18s/it]
epoch 4 iter 7: train loss 0.35198. lr 5.997352e-04: 100%|██████████| 8/8 [00:16<00:00,  2.05s/it]
epoch 5 iter 7: train loss 0.30953. lr 5.995847e-04: 100%|██████████| 8/8 [00:16<00:00,  2.05s/it]
epoch 6 iter 7: train loss 0.26981. lr 5.994004e-04: 100%|██████████| 8/8 [00:16<00:00,  2.04s/it]
epoch 7 iter 7: train loss 0.21797. lr 5.991823e-04: 100%|██████████| 8/8 [00:16<00:00,  2.03s/it]
epoch 8 iter 7: train loss 0.18256. lr 5.989306e-04: 100%|██████████| 8/8 [00:16<00:00,  2.03s/it]
epoch 9 iter 7: train loss 0.14337. lr 5.986453e-04: 100%|██████████| 8/8 [00:16<00:00,  2.03s/it]
epoch 10 iter 7: train loss 0.13564. lr 5.983263e-04: 100%|██████████| 8/8 [00:16<00:00,  2.03s/it]
(local_nmt) sh-4.2$ python src/run.py evaluate vanilla wiki.txt --reading_params_path vanilla.finetune.params --eval_corpus_path b
irth_dev.tsv --outputs_path vanilla.pretrain.dev.predictions
data has 418352 characters, 256 unique.
number of parameters: 3323392
500it [01:05,  7.68it/s]
Correct: 129.0 out of 500.0: 25.8%
(local_nmt) sh-4.2$ 
```

G.

I

```
(local_nmt) sh-4.2$ python src/run.py evaluate synthesizer wiki.txt --reading_params_path synthesizer.finetune.params --eval_corpus_path birth_dev.tsv --outputs_path synthesize
s.pretrain.dev.prediction
data has 418352 characters, 256 unique.
number of parameters: 3076988
500it [01:13,  6.82it/s]
Correct: 47.0 out of 500.0: 9.4%
(local_nmt) sh-4.2$ python src/run.py evaluate synthesizer wiki.txt \ --reading_params_path synthesizer.finetune.params \ --eval_corpus_path birth_test_inputs.tsv \r.pretrainr.
pretrain.test.predictions
usage: run.py [-h] [--reading_params_path READING_PARAMS_PATH] [--writing_params_path WRITING_PARAMS_PATH] [--finetune_corpus_path FINETUNE_CORPUS_PATH] [--eval_corpus_path EVA
L_CORPUS_PATH]
              [--outputs_path OUTPUTS_PATH]
              {pretrain,finetune,evaluate} {vanilla,synthesizer} pretrain_corpus_path
run.py: error: unrecognized arguments:  --reading_params_path synthesizer.finetune.params  --eval_corpus_path birth_test_inputs.tsv  --outputs_path r.pretrain.test.predictions
(local_nmt) sh-4.2$ python src/run.py evaluate synthesizer wiki.txt --reading_params_path synthesizer.finetune.params  --eval_corpus_path birth_test_inputs.tsv --outputs_path s
predictionspretrain.test.
data has 418352 characters, 256 unique.
number of parameters: 3076988
437it [01:02,  7.01it/s]
No gold birth places provided; returning (0,0)
Predictions written to synthesizer.pretrain.test.predictions; no targets provided
```

I got a 9.4% accuracy on the development test set (which hopefully translates to > 5% accuracy on the test set, if these are both held out random samples (no hyperparameter tuning was done)).

II

CausalSelfAttention implements attention by first generating a query representation for each token from the input, generating a key representation for each token from the input, and then comparing the two to decide how to weight the value representations in the output. This in the synthesizer paper is referred to as token-token interactions, as the input is involved in a dot product with itself.

The primary purpose of attention is learning self-alignment: figuring out which individual tokens in the input sequence are relevant to the current output. This is especially important if the input sequence is long.

SynthesizerAttention skips token-token interactions to learn self alignment by instead learning a more powerful neural network that ingests the input once to learn pairwise attention weights. This is much faster as it requires 3 matrix multiplications instead of 4 (and compensates a little bit using a relu on top of the query representation).

So if a problem we are working on has self-alignment properties that need to compare token representations exactly to themselves (to see how similar they are), synthesizer attention will more likely struggle with this.

Another way to summarize this is that synthesizer attention can't flexibly create custom query and key representations based on the input sequence. It has to create one general representation for the keys, and then create ideal queries using the input sequence to compute attention from those keys.

3. Considerations in pretrained knowledge (5 points)

A.

The pretrained model was able to achieve an accuracy score much higher than 10% because it was able to learn a lot about language and places associated to people beforehand from the character deletion task. This allowed it much better accuracy to predict people's birthplace after training on the birth-place tasks because both of these learnings are relevant for that task.

B.

The predictions in birth place prediction from the pre-trained model mostly look plausible (regardless of whether they are correct or not). For a user using this system, this might cause a problem of misinformation - where because the prediction looks plausible, the user is most likely to use and share that prediction. This also is less desirable for the user because it makes it difficult for the user to filter for or create confidence intervals around the prediction. If instead the predictions were either 1) correct or 2) gibberish, it would be easy for the user of the system to just throw out the gibberish incorrect predictions and just use the correct ones in their downstream applications.

C.

If the model didn’t see the birthplace in either the pretraining or training task, it’s impossible for the model to have learned the correct birthplace. In this case, the model will most likely default to “average” distributions based on the name origin of the person. For instance, if the model has learned that a name is frequent in the middle east (e.g. Muhammad), it might predict a high population middle eastern city like Mecca. Additionally, if the model has learned that the name John is frequent in the West, it might predict a high population western city like New York City. This can cause a bias in error based on name origin, where people with names that originate from different locations often get a higher error then people with names common to their actual birth-place.