CS 539: Natural Language Processing with Deep Learning **(Due: March 11th, 2021)**

Submission Homework 4

Instructor: Prof. Stefan Lee *Name:* Rashmi Jadhav, *Student ID:* 934-069-574

1. Scaled Dot-Product Attention

1. In scaled dot-product attention, when we come across an input word which is not present in the model's vocabulary, we end up copying its value from the attention distribution.

   For the output to be equal to one of the input values v_j, we would have produced a peaky attention distribution with the attention value α_j corresponding to input v_j being the maximum compared to the rest of the α_i's, at the same time these α_i's would be way smaller ≈ 0 compared to $\alpha_j \approx 1$. This would result into picking just the input word v_j which had the maximum value in its attention due to Eq. (2).

   For the attention calculation in Eq. (1), in order to get higher attention for j'th input, the score (query q x candidate key k_j) for the j'th input would be much higher than that of all other i inputs.

2. $$k_i^T k_j = 0, \forall ij \;\; and \;\; ||k_i|| = 1 \tag{0.1}$$

   Let $v_a, v_b \in \{v_1, v_2, ..., v_m\}$ be two value vectors and the key vectors corresponding to v_a and v_b are k_a and k_b respectively.

   For output to be approximately $a \approx \frac{1}{2}(v_a + v_b)$, the attention values for the vectors a and b should be very high compared to the rest of the attentions; meaning $\alpha_a \approx 0.5; \alpha_b \approx 0.5;$ *Rest of* $\alpha_i's \approx 0$. The query q vector would be:

   $$q = c(k_a + k_b)$$

   The score for input a:

   $$qk_a^T = c(k_a + k_b)k_a^T$$

   $$\implies qk_a^T = c(k_a k_a^T + k_b k_a^T)$$

   From Eq. (0.1), $k_b k_a^T = 0$ and $||k_a|| = 1$

   $$\implies qk_a^T = c \tag{0.2}$$

   Similarly,

   $$\implies qk_b^T = c \tag{0.3}$$

   The scores for rest of the input i's where $i \neq a$ and $i \neq b$ would be:

   $$qk_i^T = c(k_a + k_b)k_i^T$$

   $$qk_i^T = c(k_a k_i^T + k_b k_i^T)$$

   And due to orthogonal behavior from Eq. (0.1),

   $$qk_i^T = 0 \tag{0.4}$$

   The attentions:

   $$\alpha_i = \frac{exp(qk_i^T/\sqrt{d})}{\sum_{j=1}^{m} exp(qk_j^T/\sqrt{d})} \tag{0.5}$$

Substituting Eq. (0.2) and Eq. (0.3) in (0.5) for α_a, α_b:

$$\implies \alpha_a = \alpha_b = \frac{exp(c/\sqrt{d})}{\sum_{j=1}^{m} exp(qk_j^T/\sqrt{d})} \tag{0.6}$$

$\forall i \neq a$ and $\forall i \neq b$ substituting (0.4) in (0.5):

$$\implies \alpha_i = \frac{exp(0)}{\sum_{j=1}^{m} exp(qk_j^T/\sqrt{d})} \quad \forall i \neq a \ \ and \ \ \forall i \neq b$$

$$\implies \alpha_i = \frac{1}{\sum_{j=1}^{m} exp(qk_j^T/\sqrt{d})} \quad \forall i \neq a \ \ and \ \ \forall i \neq b \tag{0.7}$$

From (0.6) and (0.7), we notice that scaling constant affects only α_a and α_b and α_i's can be made to reach 0 by making $\alpha_a \approx \alpha_b \approx \frac{1}{2}$. Thus, we have,

$$a \approx \frac{1}{2}(v_a + v_b) \tag{0.8}$$

3. Key vectors $\{k_1, k_2, ..., k_m\}$ are randomly scaled such that $k_i = \mu_i * \lambda_i$ and $\mu_1, \mu_2, ..., \mu_m$ are orthogonal unit vectors. Then using the same strategy as previous answer prompt,

$$q = c(k_a + k_b)$$

$$\implies q = c(\mu_a\lambda_a + \mu_b\lambda_b)$$

The score for input a:

$$qk_a^T = c(\mu_a\lambda_a + \mu_b\lambda_b)k_a^T$$

$$\implies qk_a^T = c(\mu_a\lambda_a + \mu_b\lambda_b)\mu_a^T\lambda_a$$

$$\implies qk_a^T = c(\mu_a\mu_a^T\lambda_a^2 + \mu_b\mu_a^T\lambda_b\lambda_a)$$

Using orthogonal unit vectors quality of μ's

$$\implies qk_a^T = c\lambda_a^2 \tag{0.9}$$

Similarly,

$$\implies qk_b^T = c\lambda_b^2 \tag{0.10}$$

The scores for rest of the input i's where $i \neq a$ and $i \neq b$ would be:

$$qk_i^T = c(\mu_a\lambda_a + \mu_b\lambda_b)k_i^T$$

$$\implies qk_i^T = c(\mu_a\lambda_a + \mu_b\lambda_b)\mu_i^T\lambda_i$$

$$\implies qk_i^T = c(\mu_a\lambda_a\mu_i^T\lambda_i + \mu_b\mu_i^T\lambda_i\lambda_b) = 0 \tag{0.11}$$

Comparing (0.9), (0.10), (0.11) to the scores in previous answer, we observe that qk_i^T can be zeroed whereas qk_a^T and qk_b^T are now proportional to the square of λ_a and the the square of λ_b respectively. If we keep $\lambda_a = \lambda_b$, we would get the same behavior of $a \approx \frac{1}{2}(v_a + v_b)$ but since λ's are randomly sampled, we cannot guarantee $\lambda_a = \lambda_b$. This results in $a \neq \frac{1}{2}(v_a + v_b)$

4. Now we have two queries defined as (q_1 *and* q_2) leading to two different attended features (a_1 *and* a_2). The output of this computation is $a = \frac{1}{2}(a_1 + a_2)$. Using the keys in previous task, we now design q_1 and q_2:

$$q_1 = c\mu_a\lambda_a \ \ and \ \ q_2 = c\mu_b\lambda_b \tag{0.12}$$

Computing score for input a:

$$q_1 k_a^T = c\mu_a\lambda_a k_a^T$$

$$\implies q_1 k_a^T = c\mu_a\lambda_a\mu_a^T\lambda_a = c\lambda_a^2 \tag{0.13}$$

Computing score for input i:

$$q_1 k_i^T = c\mu_a\lambda_a k_i^T$$

$$\implies q_1 k_i^T = c\mu_a\lambda_a\mu_i^T\lambda_i = 0 \tag{0.14}$$

Similarly,

$$\implies q_2 k_b^T = c\lambda_b^2 \tag{0.15}$$

and,

$$\implies q_2 k_i^T = 0 \tag{0.16}$$

Now,

$$a = \frac{1}{2}(a_1 + a_2)$$

$$\implies a \approx \frac{1}{2}(\alpha_a v_a + \alpha_b v_b)$$

From the scores above needed to calculate α's, we notice that $\alpha_a \approx \alpha_b \approx 1$.

Thus,

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

2. Attention in German-to-English Machine Translation

1. Implementation of Scaled-Dot Product Attention:

```python
class SingleQueryScaledDotProductAttention(nn.Module):

    # kq_dim is the dimension of keys and values.
    # Linear layers should be used to project inputs to these dimensions.
    def __init__(self, enc_hid_dim, dec_hid_dim, kq_dim=512):
        super().__init__()
        self.linear_keys = nn.Linear(enc_hid_dim * 2, kq_dim)
        self.linear_queries = nn.Linear(dec_hid_dim, kq_dim)
        self.kq_dim = kq_dim
        self.softmax = Softmax(dim=-1)

    # hidden is h_t^{d} from Eq. (11) and has dim => [batch_size , dec_hid_dim]
    # encoder_outputs is the word representations from Eq. (6)
    # and has dim => [src_len , batch_size , enc_hid_dim * 2]
    def forward(self, hidden, encoder_outputs):
        # Convert hidden and encoder_outputs to keys, queries and values
        k_t = self.linear_keys(encoder_outputs)
        q = self.linear_queries(hidden)
        v_t = encoder_outputs

        # Compute Eq. (1) for batch
        k = torch.transpose(input=torch.transpose(input=k_t, dim0=0, dim1=1), dim0=1, dim1=2)
        score = torch.bmm(q.unsqueeze(1), k) / math.sqrt(self.kq_dim)
        alpha = self.softmax(score)

        # Compute Eq. (2) for batch
        attended_val = torch.bmm(alpha, torch.transpose(v_t, 0, 1))

        attended_val = attended_val[:, -1, :]
        alpha = alpha[:, -1, :]

        assert attended_val.shape == (hidden.shape[0], encoder_outputs.shape[2])
        assert alpha.shape == (hidden.shape[0], encoder_outputs.shape[0])

        return attended_val, alpha
```

The **perplexity** obtained **on the test set** is **6.178** whereas **BLEU score** for the same is **31.70**:

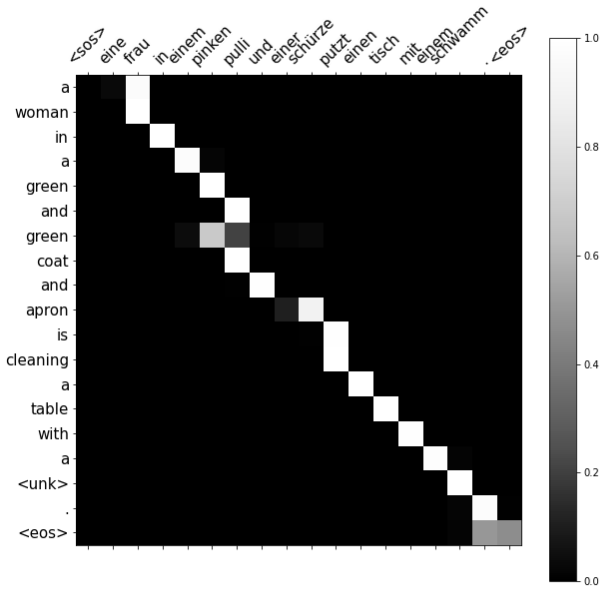
```
Epoch 1: 100%|██████████| 227/227 [01:10<00:00,  3.24batch/s]
2021-03-07 22:13:21 INFO     Epoch: 01  Train Loss: 3.747 | Train PPL:  42.394
2021-03-07 22:13:21 INFO     Epoch: 01   Val. Loss: 2.503 |  Val. PPL:  12.215
Epoch 2: 100%|██████████| 227/227 [01:10<00:00,  3.22batch/s]
2021-03-07 22:14:32 INFO     Epoch: 02  Train Loss: 2.391 | Train PPL:  10.923
2021-03-07 22:14:32 INFO     Epoch: 02   Val. Loss: 1.982 |  Val. PPL:   7.254
Epoch 3: 100%|██████████| 227/227 [01:10<00:00,  3.22batch/s]
2021-03-07 22:15:44 INFO     Epoch: 03  Train Loss: 1.911 | Train PPL:   6.763
2021-03-07 22:15:44 INFO     Epoch: 03   Val. Loss: 1.838 |  Val. PPL:   6.283
Epoch 4: 100%|██████████| 227/227 [01:10<00:00,  3.20batch/s]
2021-03-07 22:16:55 INFO     Epoch: 04  Train Loss: 1.628 | Train PPL:   5.094
2021-03-07 22:16:55 INFO     Epoch: 04   Val. Loss: 1.777 |  Val. PPL:   5.912
Epoch 5: 100%|██████████| 227/227 [01:10<00:00,  3.22batch/s]
2021-03-07 22:18:06 INFO     Epoch: 05  Train Loss: 1.426 | Train PPL:   4.160
2021-03-07 22:18:06 INFO     Epoch: 05   Val. Loss: 1.750 |  Val. PPL:   5.757
Epoch 6: 100%|██████████| 227/227 [01:11<00:00,  3.19batch/s]
2021-03-07 22:19:18 INFO     Epoch: 06  Train Loss: 1.280 | Train PPL:   3.595
2021-03-07 22:19:18 INFO     Epoch: 06   Val. Loss: 1.752 |  Val. PPL:   5.767
Epoch 7: 100%|██████████| 227/227 [01:10<00:00,  3.21batch/s]
2021-03-07 22:20:29 INFO     Epoch: 07  Train Loss: 1.165 | Train PPL:   3.207
2021-03-07 22:20:29 INFO     Epoch: 07   Val. Loss: 1.755 |  Val. PPL:   5.783
Epoch 8: 100%|██████████| 227/227 [01:10<00:00,  3.22batch/s]
2021-03-07 22:21:40 INFO     Epoch: 08  Train Loss: 1.076 | Train PPL:   2.934
2021-03-07 22:21:40 INFO     Epoch: 08   Val. Loss: 1.763 |  Val. PPL:   5.830
Epoch 9: 100%|██████████| 227/227 [01:10<00:00,  3.23batch/s]
2021-03-07 22:22:51 INFO     Epoch: 09  Train Loss: 1.005 | Train PPL:   2.733
2021-03-07 22:22:51 INFO     Epoch: 09   Val. Loss: 1.776 |  Val. PPL:   5.909
Epoch 10: 100%|██████████| 227/227 [01:11<00:00,  3.19batch/s]
2021-03-07 22:24:03 INFO     Epoch: 10  Train Loss: 0.945 | Train PPL:   2.574
2021-03-07 22:24:03 INFO     Epoch: 10   Val. Loss: 1.793 |  Val. PPL:   6.004
2021-03-07 22:24:03 INFO     Running test evaluation:


2021-03-07 22:24:17 INFO     | Test Loss: 1.821 | Test PPL:   6.178 | Test BLEU 31.70
```

The translated sentences are quite decent with Scaled Dot Product Attentions:

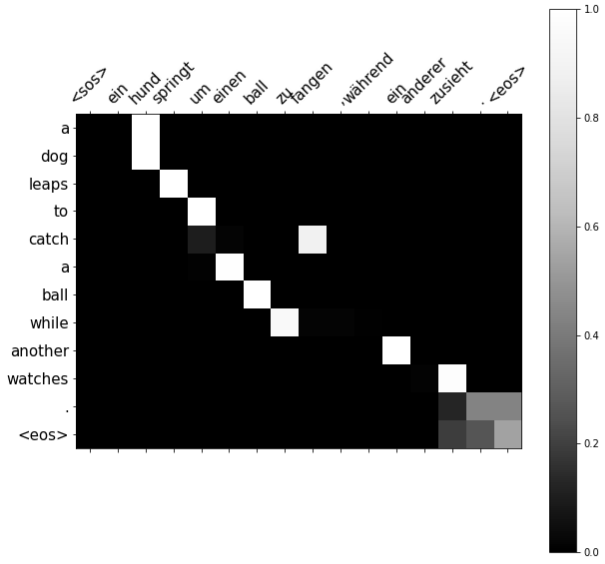
```
--------------------
src = ['ein', 'junge', 'posiert', 'mit', 'einem', 'großen', 'grünen', 'insekt', 'auf', 'der', 'nase', '.']
trg = ['a', 'boy', 'poses', 'with', 'a', 'large', 'green', 'insect', 'on', 'his', 'nose', '.']
prd = ['a', 'boy', 'posing', 'with', 'a', 'large', 'green', 'and', 'large', 'green', 'toy', 'on', 'his', 'nose', '.', '<eos>']

--------------------
src = ['viele', 'menschen', 'sitzen', 'um', 'ein', 'zelt', 'im', 'freien', '.']
trg = ['many', 'people', 'are', 'sitting', 'around', 'a', 'tent', 'outside', '.']
prd = ['many', 'people', 'are', 'sitting', 'around', 'a', 'tent', 'outside', '.', '<eos>']

--------------------
src = ['eine', 'familie', 'spaziert', 'durch', 'einen', 'park', '.']
trg = ['a', 'family', 'going', 'for', 'a', 'walk', 'in', 'a', 'park', '.']
prd = ['a', 'family', 'is', 'walking', 'a', 'park', '.', '<eos>']

--------------------
src = ['diese', 'personen', 'klettern', 'die', 'stufen', 'zum', 'berg', 'hoch']
trg = ['these', 'people', 'are', 'climbing', 'the', 'steps', 'to', 'go', 'the', 'mountain']
prd = ['these', 'people', 'are', 'climbing', 'up', 'the', 'steps', 'to', 'the', 'steps', '.', '<eos>']

--------------------
src = ['eine', 'gruppe', 'klettert', 'bei', 'kaltem', 'wetter', '.']
trg = ['a', 'group', 'of', 'people', 'are', 'climbing', 'in', 'cold', 'weather', '.']
prd = ['a', 'group', 'of', 'people', 'are', 'climbing', 'in', 'a', 'cold', 'weather', '.', '<eos>']
```

2. Translations from one language to another show that there are different kinds of alignments between them like many-to-many, one-to-many, many-to-one, and inversions. The attention maps generated by our model help us in observing the similar trends. For each word in the translated sentence, we can figure out where the model is attending in the source sentence. Following are some of the attention diagrams generated:

   I. One-to-many: From the following example, we notice that **putz** is translated as **is cleaning** which basically mapped one word in German to two words in English.



   II. Inversion: From the following example, we notice that **fangen (to catch)** comes somewhere quite later in the German sentence compared to English. This is the difference of Subject-Verb-Object in English vs Subject-Object-Verb in German.



   With the limited set of examples which were tried out, it was difficult to find any many to one mappings from German to English and majorly, the translated English sentence is longer than the source sentence in

German.

3. Comparisons
   Following are the BLEU scores for the three types of attention mechanisms averaged over three runs:

BLEU	Mean	Variance
Dummy	18.7667	0.202033
MeanPool attention	22.25	0.1525
Scaled-dot-product attention	34.63	0.7617

The Dummy version attends to no words. This means we do not have any attention distribution but only in-vocabulary distribution. Not having attention mechanism would not be able to handle unseen words in the test and produce more UNKs. Thus, Dummy version has got the worst BLEU score. The MeanPool version attends equally to all the words. This is a little enhancement over the Dummy and so we observe an improved BLEU score here. The Single Scaled Dot Product attention mechanism has the quality that it extends the vocabulary dynamically based on inputs and decides when to copy a word and when not to. This gives an edge over prior two methods. Thus we observe an even improved BLEU score here.

Perplexity	Mean	Variance
Dummy attention	10.918	0.006367
MeanPool attention	9.02267	0.000204333
Scaled-dot-product attention	6.103	0.000037

Like how bigger the BLEU score, better the model, smaller the perplexity, better is the model. Having smaller perplexity means there's more probability of generating a sentence. We notice similar trends in the Perplexity scores as well. The model does best with Scaled-dot-product attention followed by MeanPool and Dummy.