CSCI-GA 2590: Natural Language Processing

HW2 - Machine Translation

Chuanyang Jin
cj2133

Collaborators:
By turning in this assignment, I agree by the honor code of the College of Arts and Science at New York University and declare that all of this is my own work.

Acknowledgement: This HW is based off of Annotated Transformers from Sasha Rash.

Before you get started, please read the Submission section thoroughly.

Submission

Submission is done on Gradescope.

Written: You can either directly type your solution in the released `.tex` file, or write your solution using pen or stylus. A `.pdf` file must be submitted.

Programming: Questions marked with "coding" at the start of the question require a coding part. Each question contains details of which functions you need to modify. We have also provided some unit tests for you to test your code. You should submit all `.py` files which you need to modify, along with the generated output files as mentioned in some questions.

Machine Translation

The goal of this homework is to build a machine translation system using sequence-to-sequence transformer models `https://arxiv.org/abs/1706.03762`. More specifically, you will build a system which translates German to English using the Multi30k dataset (`https://arxiv.org/abs/1605.00459`) You are provided with a code skeleton, which clearly marks out where you need to fill in code for each sub-question.

First go through the file `README.md` to set up the environment required for the class.

1. Attention

Transformers use scaled dot-product attention — given a set of queries Q (each of dimension d_k), a set of keys K (also each dimension d_k), and a set of values V (each of dimension d_v), the output is a weighted sum of the values. More specifically,

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{1}$$

Note that each of Q, K, V is a matrix of vectors packed together.

1. (2 points, written) The above function is called 'scaled' attention due to the scaling factor $\frac{1}{\sqrt{d_k}}$. The original transformers paper mentions that this is needed because dot products between keys and queries get large with larger d_k.

   For a query q and key k both of dimension d_k and each component being an independent random variable with mean 0 and variance 1, compute the mean and variance (with steps) of the dot product $q.k$ to demonstrate the point.

$$\begin{aligned}
\mathbb{E}[q.k] &= \mathbb{E}\left[\sum_{i=1}^{d_k} q_i k_i\right] \\
&= \sum_{i=1}^{d_k} \mathbb{E}[q_i k_i] \\
&= \sum_{i=1}^{d_k} \mathbb{E}[q_i] \cdot \mathbb{E}[k_i] && \text{(since } q \text{ and } k \text{ are independent)} \\
&= \sum_{i=1}^{d_k} 0 \cdot 0 \\
&= 0
\end{aligned}$$

$$\begin{aligned}
\mathrm{Var}[q.k] &= \mathbb{E}[(q.k)^2] - \mathbb{E}[(q.k)]^2 \\
&= \mathbb{E}[(q.k)^2] \\
&= \mathbb{E}\left[\left(\sum_{i=1}^{d_k} q_i k_i\right)^2\right] \\
&= \sum_{i=1}^{d_k} \mathbb{E}\left[(q_i k_i)^2\right] + 2\sum_{i\neq j}\sum \mathbb{E}\left[(q_i k_j)^2\right] && \text{(since } q \text{ and } k \text{ are independent)} \\
&= \sum_{i=1}^{d_k} 1 && \text{(since } \mathbb{E}\left[(q_i k_i)^2\right] = \mathbb{E}\left[q_i^2\right] \cdot \mathbb{E}\left[q_i^2\right] = 1 \cdot 1 = 1\text{)} \\
&+ \sum_{i\neq j}\sum 0 && \text{(since } \mathbb{E}\left[(q_i k_j)^2\right] = \mathbb{E}\left[q_i k_i q_j k_j\right] = 0 \text{ if } i \neq j\text{)} \\
&= d_k
\end{aligned}$$

2. (2 points, coding) Implement the above scaled dot-product attention in the `attention()` function present in `layers.py`. You can test the implementation after the next part.

Implemented.

3. (2 point, coding) In this part, you will modify the `attention()` function by making use of the parameters `mask` and `dropout` which were input to the function. The `mask` indicates positions where the attention values should be zero (e.g. when we have padded a sentence of length 5 to length 10, we do not want to attend to the extra tokens). `dropout` should be applied to the attention weights for regularization.

   To test the implementation against some unit tests, run `python3 test.py --attention.`

Implemented.

4. (3 points, coding) Instead of a single attention function, transformers use multi-headed attention function. For original keys, queries and values (each of dimension say d_{model}), we use h different projection matrices to obtain queries, keys and values of dimensions d_k, d_k and d_v respectively. Implement the function `MultiHeadedAttention()` in `layers.py`. To test the implementation against some unit tests, run `python3 test.py --multiheaded_attention.`

Implemented.

2. Positional Encoding

Since the underlying blocks in a transformer (namely attention and feed forward layers) do not encode any information about the order of the input tokens, transformers use 'positional encodings' which are added to the input embeddings. If d_{model} is the dimension of the input embeddings, pos is the position, and i is the dimension, then the encoding is defined as:

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}}) \tag{2}$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}}) \tag{3}$$

1. (2 points, written) Since the objective of the positional encoding is to add information about the position, can we simply use $PE_{pos} = sin(pos)$ as the positional encoding for pos position? Why or why not?

No, simply using $PE_{pos} = sin(pos)$ will give the same values for many different positions, and the model will not be able to differentiate between those positions. We need the positional encoding to have some properties such as being continuous and able to represent relative distances between positions.

2. (2 points, coding) Implement the above positional encoding in the function `PositionalEncoding()` in the file `utils.py`. To test the implementation against some unit tests, run `python3 test.py --positional_encoding`.

Implemented.

3. Training

1. (2 points, written) The above questions should complete the missing parts in the training code and we can now train a machine translation system!

   Use the command `python3 main.py` to train your model. For the purpose of this homework, you are not required to tune any hyperparameters. You should submit the generated `out_greedy.txt` file containing outputs. You must obtain a BLEU score of at least 35 for full points.

Implemented.

4. Decoding & Evaluation

In the previous question, the code uses the default `greedy_decode()` to decode the output. In practice, people use algorithms such as beam search decoding, which have been shown to give better quality outputs. (Note: In the following questions, use a model trained with the default i.e. given hyperparameters)

1. (2 points, written) In the file `utils.py` you will notice a function `subsequent_mask()`. What does that function do and why is it required in our model?

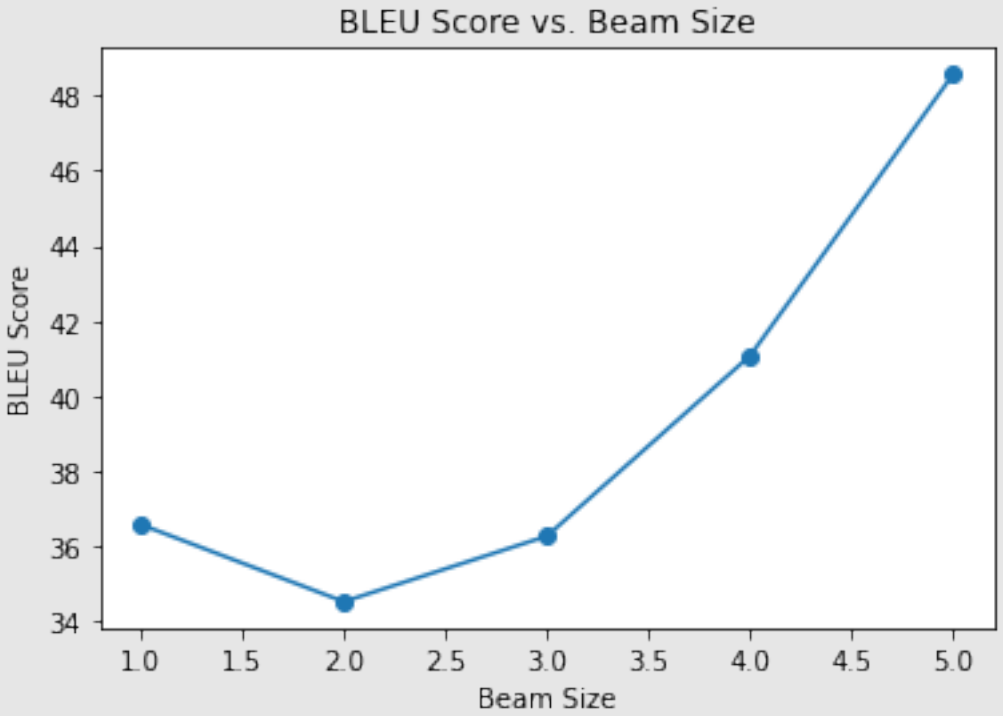
This function generates a mask that has ones in the lower triangular part of the matrix and zeros in the upper triangular part, masking out the subsequent positions. By forcing the model to attend only to previous positions, it lets the model to learn to make predictions based only on the information available at each step in the sequence.

2. (5 points, coding) Implement the `beam_search()` function in `utils.py`. We have provided the main skeleton for this function and you are only required to fill in the important parts (more details in the code). You can run the code using the arguments `--beam_search` and `--beam_size`. You should submit the generated file `out_beam.txt` when `beam_size = 2`.

   To test the implementation against some unit tests, run `python3 test.py --beam_search`.

Implemented.

3. (3 points, written) For the model trained in question 1.3, plot the BLEU score as a function of beam size. You should plot the output from beam size 1 to 5. Is the trend as expected? Explain your answer.



The BLEU score generally increases as the beam size increases. This is generally expected as a larger beam size means that more candidates are being considered, which can increase the likelihood of finding a better result. However, there is a drop in performance when the beam size increased from 1 to 2. This may be because the additional candidates produced by the larger beam size are not as good.

4. (2 points, written) You might notice that some of the sentences contain the '⟨unk⟩' token which denotes a word not in the vocabulary. For systems such as Google Translate, you might not want such tokens in the outputs seen by the user. Describe a potential way to avoid (or reduce) the occurrence of these tokens in the output.

To avoid (or reduce) the occurrence of these tokens in the output, we can

(a) improve the model's vocabulary coverage by adding more words to the vocabulary;

(b) use data augmentation and filtering to reduce the occurrence of unknown words in the input data;

(c) use post-processing techniques to replace '⟨unk⟩' tokens with appropriate words or phrases.

5. (2 points, written) In this homework, you have been using BLEU score as your evaluation metric. Consider an example where the reference translation is "I just went to the mall to buy a table.", and consider two possible model generations: "I just went to the mall to buy a knife." and "I just went to the mall to buy a desk.". Which one will BLEU score higher? Suggest a potential fix if it does not score the intended one higher.

They have the same BLEU score, since they have the same unigram precision.

However, since "desk" is a closer match to "table" than "knife," we would expect "I just went to the mall to buy a desk." to have a higher BLEU score than "I just went to the mall to buy a knife."

To achieve this, one potential fix is to consider the difference in word choice when computing the score, possibly by assigning an appropriate weight to similar words.