EECS 182 Deep Neural Networks
Fall 2022 Anant Sahai

Homework 8

This homework is due on Monday, Nov 14, 2022, at 10:59PM.

Deliverables: Please submit the code/notebooks/saved model as a zip file to the code gradescope assignment. Submit your written answers in the written gradescope assignment, and attach a pdf printout of the notebooks.

1. Coding Question: Summarization

Implement the Summarization.ipynb notebook and answer the following questions below.

(a) In the notebook, visualize attention weights for randomly-chosen Q, K, and V matrices. Would you expect weights to sum to 1 if we sum over the queries dimenstion, or over the keys dimension? **Solution:** Weight should sum to 1 if we sum over the key dimension. Intuitively, you can think of the attention weights for a query as probabilities of which value it will retried from a lookup table, and these probabilities should sum to 1.

(b) Run the notebook cell which plots attention weights where Q and K are identical random matrices. Explain why we see the attention weights and outputs we do. **Solution:** The attention map looks close to the identity. This is because the dot product between q_i and k_i is high, since they are identical, but the dot product between q_i and k_j is low, since the two vectors are random and completely independent, so the dot product is likely close to 0. This means that output $o_i \approx v_i$, so the output matrix looks the same as the value matrix.

(c) (Optional) In the notebook, we showed that positional encodings make it easy to attend to relative positions. Derive why the transformation matrix used achieves the desired effect - i.e. derive a matrix $M \in \mathbb{R}^{d\times d}$ such that $p_{t+1} = Mp_t$, where p_t is the d-dimensional positional encoding for timestep t.

Solution: consider a pair of features in the positional encoding i and $i+1$, where i is even. Their formulas are given by

$$k = i/2$$
$$\omega_k = \frac{1}{10000^{2k/d}}$$
$$p_t^i = \sin(\omega_k t)$$
$$p_t^{i+1} = \cos(\omega_k t)$$

Let M be a 2x2 matrix. We want to find u_1, v_1, u_2, and v_2 so that

$$\begin{bmatrix} u_1 & v_1 \\ u_2 & v_2 \end{bmatrix} \begin{bmatrix} \sin(\omega_k t) \\ \cos(\omega_k t) \end{bmatrix} = \begin{bmatrix} \sin\big(\omega_k(t+1)\big) \\ \cos\big(\omega_k(t+1)\big) \end{bmatrix}$$

By applying the addition theorem, we can expand the right hand side as follows.

$$\begin{bmatrix} u_1 & v_1 \\ u_2 & v_2 \end{bmatrix} \begin{bmatrix} \sin(\omega_k t) \\ \cos(\omega_k t) \end{bmatrix} = \begin{bmatrix} \sin(\omega_k t)\cos(\omega_k) + \cos(\omega_k t)\sin(\omega_k) \\ \cos(\omega_k t)\cos(\omega_k) - \sin(\omega_k t)\sin(\omega_k) \end{bmatrix}$$

Which results in the following two equations

$$u_1 \sin(\omega_k t) + v_1 \cos(\omega_k t) = \cos(\omega_k)\sin(\omega_k t) + \sin(\omega_k)\cos(\omega_k t)$$
$$u_2 \sin(\omega_k t) + v_2 \cos(\omega_k t) = -\sin(\omega_k)\sin(\omega_k t) + \cos(\omega_k)\cos(\omega_k t)$$

Solving for u_1, v_1, u_2, and v_2 gives

$$u_1 = \cos(\omega_k)$$
$$v_1 = \sin(\omega_k)$$
$$u_2 = -\sin(\omega_k)$$
$$v_2 = \cos(\omega_k)$$

This gives us the desired 2x2 matrix m_k

$$m_k = \begin{bmatrix} \cos(\omega_k) & \sin(\omega_k) \\ -\sin(\omega_k) & \cos(\omega_k) \end{bmatrix}$$

To shift an entire p_t vector to the next position, we can create a block diagonal matrix M where each block is m_k.
Solution reference: `https://kazemnejad.com/blog/transformer_architecture_positional_encoding/`

(d) In this model, the encoder and decoder share the same dimensions. Is this necessary, or could we choose to make them different? **Solution:** They can be different. The only requirement is that the decoder cross-attention can ingest outputs from the final encoder layer.

(e) Run the notebook cell which visualizes attention masks, and explain the patterns you see.

Solution: The encoder and decoder input maps show how we handle sequences of different lengths. Pad tokens are masked and unpadded tokens are unmasked. The cross attention mask shows that every query can attend to every key except for the pad tokens. The decoder attention mask shows that all pad tokens are masked out, along with all future tokens.

(f) How would you modify the summary sampling procedure if you wanted to generate multiple summaries per article rather than just one? **Solution:** There are multiple valid answers, including.

- Use beam search and return all the stored summaries.
- Sample next tokens from the output probability distribution rather than taking the argmax. You can do this repeatedly to generate different summaries.
  - Variant: rescale the logits with a temperature parameter before sampling to control how diverse the generated summaries are.
  - Variant: use top-p or top-k sampling, which both only sample from the most probable next tokens, to avoid sampling very unlikely next tokens.

(g) (Optional) How does the Performer model's performance match the performance with standard softmax attention? Explain how you reached this conclusion (Did you test with multiple sets of hyperparameters? Include plots to support your answer.)

Solution: Answers may vary, but they should perform similarly.

2. Coding Question: Visualizing Attention

Please run the cells in the Visualizing_BERT.ipynb notebook, then answer the questions below.

(a) Attention in GPT: Run part a of the notebook and generate the corresponding visualizations

i. What similarities and differences do you notice in the visualizations between the examples in this part? Explore the queries, keys, and values to identify any interesting patterns associated with the attention mechanism.

Solution: Notice that GPT is paying attention to previous words in the sequence. Specifically, in the first example, the word 'ran' attends to the token 'dog' because it is important for the model to know who or what is doing the running. In the second example, notice that GPT is paying more attention to the subject of the sentence 'dog', rather than the object of the prepositional phrase 'car'. In the example where GPT is tasked with keeping track of past information, the model is indeed appropriately making a connection between the name mentioned previously and the vague pronoun reference in the second sentence. The queries, keys, and values further support this fact that GPT is successful in identifying linguistic patterns.

ii. How does attention differ between the different layers of the GPT model? Do you notice that the tokens are attending to different tokens as we go through the layers of the netowork?

Solution: Students may notice any number of patterns. Here's an example. As you look through the layers of the network, you may notice that GPT tends to place a lot of attention on the first word in the sentence. This is because when the model doesn't know what linguistic patterns are interesting or relevant to the current setting, it looks to the start of the sentence for answers. If you are interested in seeing some more interesting patterns, feel free to play around with this tool some more, or read this interesting blog post.

(b) BERT pays attention: Run part b of the notebook and generate the corresponding visualizations.

i. Look at different layers of the BERT model in the visualizations of part (b) and identify different patterns associated with the attention mechanism. Explore the queries, keys, and values to further inform your answer. For instance, do you notice that any particular type of tokens are attended to at a given timestep?

Solution: At some layers, words attend to the immediate next or previous token. At others, there are a lot of attentions to the CLS token. Another occurence is that that words will attend to other instances of the word in the sentence.

ii. Do you spot any differences between how attention works in GPT vs. BERT? Think about how the model architectures are different.

Solution: BERT is a bidirectional model, whereas GPT only attends to tokens one at a time. When looking at the words that the attention heads are paying atteniont to this difference becomes obvious.

iii. For the example with syntactically similar but definitionally different sentences, look through the different layers of the two BERT networks associated with sentence a and sentence b, and take a look at the queries, keys, and values associated with the different tokens. Do you notice any differences in the embeddings learned for the two sentences that are essentially identical in structure but different in meaning?

Solution: The word "play" in this context has significantly different meanings based on the context. While the overall attention patterns look very similar, in some layers you can see that keys corresponding to "play" look different between the two examples.

iv. For the pre-training related examples, do you notice BERT's bi-directionality in play? Do you think pre-training the BERT helped it learn better representations?

Solution: BERT uses bidirectionality in order to attend backwards. For example, SEP tokens look for CLS tokens even though the attention head at layer 2, head 0 mostly consists of words that look at the next work. And yes! BERT is very powerful because the training is self-supervised, and so we do not need to manually label data–pretraining BERT gives it access to larges amounts of data that it can generalize from to form useful representations. If you are interested in seeing some more interesting patterns, feel free to play around with this tool some more, or read this interesting blog post.

(c) BERT has multiple heads!: Run part c of the notebook and generate the corresponding visualizations.

i. Do you notice different features being learned throughout the different attention heads of BERT? Why do you think this might be?

Solution: Students should notice different patterns being learned by the different attention heads in the rows of the visualization. Some interesting patterns include the following: an attention head focuses on the special SEP and CLS tokens in the text; an attention head focuses on all words in the text equally; an attention head focuses on the word that was directly before it; etc. The attention heads are able to focus on different patterns as their queries, keys, and values are calculated independently and then combined together to produce a final attention score that is eventually used downstream.

ii. Can you identify any of the different features that the different attention heads are focusing on?

Solution: Students should be able to see the different patterns in the visualization. Some are listed in the solution for the part above.

(d) Visualizing untrained attention weights

i. What differences do you notice in the attention patterns between the randomly initialized and trained BERT models?

Solution: All of the words pay equal attention to all of the other words. The untrained BERT has no way of distinguishing which words might be more important, and which words are less so.

ii. What are some words or tokens that you would expect strong attention between? What might you guess about the gradients of this attention head for those words?

Solution: As seen from the previous parts of the notebook, BERT pays particular attention to word before, the word after, separators, and words that have similar meanings. We expect gradients at these steps to be higher, since these are the features that the model eventually learns.

3. Kernelized Linear Attention

This is a continuation of Problem 3 on HW 6 (https://inst.eecs.berkeley.edu/~cs182/fa22/assets/assignments/hw6_sol.pdf). Please refer to this part for notation and context. In Part 1 of this problem, we considered ways to efficiently express the attention operation when sequences are long (e.g. a long document). Attention uses the following equation:

$$V_i' = \frac{\sum_{j=1}^{N} \text{sim}\left(Q_i, K_j\right) V_j}{\sum_{j=1}^{N} \text{sim}\left(Q_i, K_j\right)}. \tag{1}$$

We saw that when the similarity function is a kernel function (i.e. if we can write $\text{sim}\left(Q_i, K_j\right) = \Phi(Q_i)^T \Phi(K_j)$ for some function Φ), then we can use the associative property of matrix multiplication to simplify the formula to

$$V_i' = \frac{\phi\left(Q_i\right)^T \sum_{j=1}^{N} \phi\left(K_j\right) V_j^T}{\phi\left(Q_i\right)^T \sum_{j=1}^{N} \phi\left(K_j\right)}. \tag{2}$$

If we use a polynomial kernel with degree 2, this gives a computational cost of $\mathcal{O}\left(ND^2M\right)$, which for very large N is favorable to the softmax attention computational cost of $\mathcal{O}\left(N^2 \max\left(D, M\right)\right)$. ($N$ is the sequence length, D is the feature dimension of the queries and keys, and M is the feature dimension of the values. Now, we will see whether we can use kernel attention to directly approximate the softmax attention:

$$V_i' = \frac{\sum_{j=1}^N \exp(\frac{Q_i^T K_j}{\sqrt{D}}) V_j}{\sum_{j=1}^N \exp(\frac{Q_i^T K_j}{\sqrt{D}})}. \tag{3}$$

(a) Approximating softmax attention with linearized kernel attention

i. As a first step, we can use Gaussian Kernel $\mathcal{K}_{\text{Gauss}}(q,k) = \exp(\frac{-||q-k||_2^2}{2\sigma^2})$ to rewrite the softmax similarity function, where $\text{sim}_{\text{softmax}}(q,k) = \exp(\frac{q^Tk}{\sqrt{D}})$. Assuming we can have $\sigma^2 = \sqrt{D}$, **rewrite the softmax similarity function using Gaussian Kernel.**. (*Hint: You can write the softmax* $\exp(\frac{-||q-k||_2^2}{2\sigma^2})$ *as the product of the Gaussian Kernel and two other terms.)*

Solution:

$$\begin{aligned}\text{sim}_{\text{softmax}}(q,k) = \exp(\frac{q^Tk}{\sqrt{D}}) &= \exp(\frac{||q||_2^2}{2\sigma^2}) * \exp(\frac{-||q-k||_2^2}{2\sigma^2}) * \exp(\frac{||k||_2^2}{2\sigma^2}) \\ &= \exp(\frac{||q||_2^2}{2\sigma^2}) * \mathcal{K}_{\text{Gauss}}(q,k) * \exp(\frac{||k||_2^2}{2\sigma^2})\end{aligned}$$

ii. However, writing softmax attention using a Gaussian kernel does not directly enjoy the benefits of the reduced complexity using the feature map. This is because the feature map of Guassian kernel usually comes from the Taylor expression of $\exp(\cdot)$, whose computation is still expensive [1]. However, we can approximate the Guassian kernel using random feature map and then reduce the computation cost. (Rahimi and Recht, 2007)[2] proposed random Fourier features to approximate a desired shift-invariant kernel. The method nonlinearly transforms a pair of vectors q and k using a **random feature map** $\phi_{\text{random}}()$; the inner product between $\phi\left(q\right)$ and $\phi\left(k\right)$ approximates the kernel evaluation on q and k. More precisely:

$$\phi_{\text{random}}(q) = \sqrt{\frac{1}{D_{\text{random}}}}\left[\sin\left(\mathbf{w}_1 q\right), \ldots, \sin\left(\mathbf{w}_{D_{\text{random}}} q\right), \cos\left(\mathbf{w}_1 q\right), \ldots, \cos\left(\mathbf{w}_{D_{\text{random}}} q\right)\right]^\top. \tag{4}$$

Where we have D_{random} of D-dimensional random vectors $\mathbf{w}_i$ independently sampled from $\mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I}_D)$,

$$\mathbb{E}_{\mathbf{w}_i}\left[\phi\left(q\right)\cdot\phi\left(k\right)\right] = \exp\left(-\|q-k\|^2/2\sigma^2\right). \tag{5}$$

Use ϕ_{random} to approximate the above softmax similarity function with Gaussian Kernel and derive the computation cost for computing all the V' here.

Solution:

$$\text{sim}_{\text{softmax}}(q,k) = \exp(\frac{||q||_2^2}{2\sigma^2}) * \mathcal{K}_{\text{Gauss}}(q,k) * \exp(\frac{||k||_2^2}{2\sigma^2})$$

[1] https://www.csie.ntu.edu.tw/~cjlin/talks/kuleuven_svm.pdf
[2] https://people.eecs.berkeley.edu/~brecht/papers/07.rah.rec.nips.pdf

$$= \exp(\frac{||q||_2^2}{2\sigma^2}) * \phi_{\text{random}}(q)^T \phi_{\text{random}}(k) * \exp(\frac{||k||_2^2}{2\sigma^2})$$

The computation cost of ϕ_{random} is $\mathcal{O}\left(N(M+D)D_{\text{random}}\right)$.)

iii. However, we should note this approximation is not perfect, **use the HELPER.PY function to plot the approximation error of ϕ_{random} attention and softmax attention scales in terms of D and D_{random}, summarize what you find below.**

Solution: The larger the D and D_{random}, the less the error will be.

(b) (Optional) Beyond self-attention, an autoregressive case will be masking the attention computation such that the i-th position can only be influenced by a position j if and only if $j \leq i$, namely a position cannot be influenced by the subsequent positions.

Derive how this causal masking changes equation 1 and 2. After deriving the masked causal attention, using S_i and Z_i as follows,

$$S_i = \sum_{j=1}^{i} \phi\left(K_j\right) V_j^T, \; Z_i = \sum_{j=1}^{i} \phi\left(K_j\right), \tag{6}$$

to simplify the causal masking kernel attention and derive the computational complexity of this new causal masking formulation scheme.

Solution: The causal masking changes equation 1 as follows,

$$V_i' = \frac{\sum_{j=1}^{i} \text{sim}\left(Q_i, K_j\right) V_j}{\sum_{j=1}^{i} \text{sim}\left(Q_i, K_j\right)}. \tag{7}$$

We linearize the masked attention as described below,

$$V_i' = \frac{\phi\left(Q_i\right)^T \sum_{j=1}^{i} \phi\left(K_j\right) V_j^T}{\phi\left(Q_i\right)^T \sum_{j=1}^{i} \phi\left(K_j\right)}. \tag{8}$$

we can simplify equation 8 to

$$V_i' = \frac{\phi\left(Q_i\right)^T S_i}{\phi\left(Q_i\right)^T Z_i}. \tag{9}$$

Note that, S_i and Z_i can be computed from S_{i-1} and Z_{i-1} in constant time hence making the computational complexity of linear transformers with causal masking linear with respect to the sequence length.

4. Homework Process and Study Group

Citing sources and collaborators are an important part of life, including being a student!
We also want to understand what resources you find helpful and how much time homework is taking, so we can change things in the future if possible.

(a) **What sources (if any) did you use as you worked through the homework?**

(b) **If you worked with someone on this homework, who did you work with?**
List names and student ID's. (In case of homework party, you can also just describe the group.)

(c) **Roughly how many total hours did you work on this homework? Write it down here where you'll need to remember it for the self-grade form.**

Contributors:

- Olivia Watkins.
- Sheng Shen.
- Hao Liu.
- Allie Gu.
- Anant Sahai.
- CS182 staff from past semesters.
- Shivam Singhal.
- Kevin Li.
- Bryan Wu.
- Shaojie Bai.
- Angelos Katharopoulos.
- Hao Peng.