BME646 and ECE60146: Homework 9

Spring 2023
Due Date: 11:59pm, Apr 30, 2023
TA: Fangda Li (li1208@purdue.edu)

Turn in typed solutions via BrightSpace. Additional instructions can be found at BrightSpace. Since this is the last HW of the semester and to ensure timely grading, **no late days are allowed for this assignment**.

1 Introduction

This homework has the following goals:

1. To gain insights into the multi-headed self-attention mechanism and to understand the transformer architecture in a deeper level.
2. To understand how the transformer architecture originally for language translation can be readily adopted to process images in Vision Transformer (ViT).
3. To implement your own ViT for image classification.

2 Getting Ready for This Homework

Before embarking on this homework, do the following:

1. Firstly, review Slide 9 through 18 of the Week 14 slides [2] on the topic of self-attention. Make sure you understand the intuition behind the self-attention mechanism and how some of its key logic can be implemented efficiently through matrix multiplications. Furthermore, review Slide 19 through 22 to understand how multiple self-attention heads can work in parallel in multi-headed attention to further capture the inter-word dependencies.
2. Go through Slide 29 through 34 to understand the encoder-decoder structure of a transformer for sequence-to-sequence translation. Especially pay attention to the diagram on Slide 34, which helps you visualize how the self-attention and cross-attention blocks are embedded in the architecture.

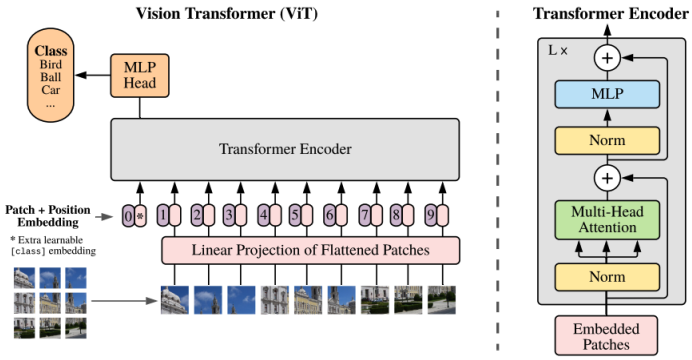

Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable "classification token" to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).

Figure 1: Illustration of the ViT architecture from [3].

3. To play with Prof. Kak's code that uses transformer for English-to-Spanish translation yourself, first download the datasets for transformer-based learning from DLStudio's main documentation page [1], and then uncompress the resulting `.tar.gz` archive it in a local directory. Execute the following two scripts in the `ExamplesTransformers` directory:

   ```
   seq2seq_with_transformerFG.py
   seq2seq_with_transformerPreLN.py
   ```

   Be sure to modify the data paths in both scripts to point correctly to your downloaded archives before running them. Try to read through his implementation on the `SelfAttention` class and the `AttentionHead` class to see how multi-headed self-attention can be coded.

4. Lastly, make sure you understand the basic ideas behind the Vision Transformer (ViT) for image classification. Figure 1 provides a clear illustration of how an image can be converted into a sequence of embeddings and subsequently processed using a transformer encoder. Particularly, you should pay attention to how the class token is prepended to the input patch embedding sequence as well as how the same token is taken from the final output sequence to produce the predicted label.

With these steps, you are ready to start working on the homework described in what follows.

3 Programming Tasks

3.1 Building Your Own ViT

The steps are:

1. Before starting, review the provided `VitHelper.py` file. It contains the necessary classes for you to quickly build a transformer from scratch. The classes are taken from DLStudio's `Transformers.TransformerPreLN` module and are made standalone so you can readily use them in your ViT implementation. More specifically, the `MasterEncoder` class in `VitHelper.py` is what you will use as the "Transformer Encoder" block as depicted in Figure 1.

2. With the tools provided in `VitHelper.py`, you are now ready to build your own ViT. While Figure 1 should already provide you with a good understanding of how ViT functions, here are some additional tips:

   - For converting patch to embedding, while you can use a linear layer like what is done in Figure 1, you can also simply use a `Conv2D` layer. Note that if you are clever about the kernel size and stride, you can apply the `Conv2D` directly on the full image itself without first dividing it into patches.
   - In ViT, suppose your image is 64×64 and you use a patch size of 16×16, you will end up with a patch sequence of length 16. Since a class token will be prepended to your patch sequence in ViT, you will need to set the max sequence length of your transformer to 17. For the class token in the initial input sequence, you can set it as a learnable parameter (*i.e.* use `nn.Parameter`).
   - Different than the sinusoid-based position embedding in language processing, ViT uses learnable position embeddings. Therefore, in addition to the initial class token, you should set the position embeddings as learnable parameters as well.
   - As for the final class prediction, you would simply take the class token from the final output sequence and feed it through a MLP layer to obtain your N-class logits.

3. In your report, designate a code block listing your ViT implementation. Make sure it is well commented.

3.2 Image Classification with Your ViT

The steps are:

1. For this task, you should reuse the training and evaluation scripts you developed in HW4. In place of the HW4 CNN, you would simply use your ViT instead. You should also reuse the COCO-based dataset that you created for HW4, which contains 64×64 images from five classes.

2. In your report, report the confusion matrix on the test set of your HW4-COCO dataset using your ViT. How does it compare with your CNN based network?

3.3 Extra Credit (10 pts)

For the Einsteins reading this, finally this is the opportunity to show off your tremendous PyTorch skills and thoroughly impress your instructors. What you need to do is simply to implement the multi-headed self-attention mechanism using `torch.einsum`. If you can do this within 10 lines of Python code (although 5 lines should be enough), you will get the full extra credit. In your report, simply designate a separate code block listing your implementation with detailed comments.

4 Submission Instructions

Include a typed report explaining how did you solve the given programming tasks.

1. **Make sure your submission zip file is under 10MB.** Compress your figures if needed. **Do NOT submit your network weights nor dataset.**

2. Your pdf must include a description of

   - The figures and descriptions as mentioned in Sec. 3.
   - Your source code. Make sure that your source code files are adequately commented and cleaned up.

3. Turn in a zipped file, it should include (a) a typed self-contained pdf report with source code and results and (b) source code files (only .py files are accepted). Rename your .zip file as hw9_<First Name><Last Name>.zip and follow the same file naming convention for your pdf report too.

4. For all homeworks, you are encouraged to use `.ipynb` for development and the report. If you use `.ipynb`, please convert it to `.py` and submit that as source code.

5. You can resubmit a homework assignment as many times as you want up to the deadline. Each submission will overwrite any previous submission. **If you are submitting late, do it only once on BrightSpace.** Otherwise, we cannot guarantee that your latest submission will be pulled for grading and will not accept related regrade requests.

6. The sample solutions from previous years are for reference only. **Your code and final report must be your own work.**

7. To help better provide feedbacks to you, make sure to **number your figures**.

References

[1] DLStudio. URL https://engineering.purdue.edu/kak/distDLS/.

[2] Transformers: Learning with Purely Attention Based Networks. URL https://engineering.purdue.edu/DeepLearn/pdf-kak/Transformers.pdf.

[3] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.