Machine Learning and Deep Learning I Homework 4

Instructor: Joonseok Lee

Deadline: 2024/06/14 Fri, 18:00

- No unapproved extension of deadline is allowed. Late submission will result in 0 credit.
- Optimize your code as much as you can. We do not guarantee to run unreasonably inefficient codes for grading. Remember, vectorization is important for efficient computation!
- You will be given skeleton files for doing your assignment. Detailed instructions are given in the comments below each method to complete. Please read them carefully before jumping into implementation!
- Each assignment is built and tested under Google Colaboratory. If you work on a local machine, you need to handle version issue on your own.
- Explicitly mention your collaborators or reference (e.g., website) if any. If we detect a copied code without reference, it will be treated as a serious violation of student code of conduct.

1 Recurrent Neural Networks [30 pts]

We are going to implement the recurrent neural network model from scratch. You have to work on the skeleton code `hw4_rnn.ipynb` provided on the ETL.

(a) Complete the `collate_batch` function. [3 pts]
For more details, please refer to the inline comments of the skeleton code, lab slides and class materials.

(b) Fill in the the shapes of the parameters in `__init__(self, vocab_size, input_size, hidden_size, num_class)`. [2 pts]

(c) Implement `forward(self, inputs, length)`. Do **NOT** use pre-defined PyTorch layers. [10 pts]
For more details, please refer to the inline comments of the skeleton code, lab slides and class materials.

(d) Implement `compute_loss(self, prediction, label)` to compute the cross-entropy loss and count the number of correct predictions. Do **NOT** use loss function APIs provided by `torch.nn` library (*e.g.*, `torch.nn.CrossEntropyLoss()`). [5 pts]

(e) Compared to practice RNN Text Classification in the lab session 4, there has been minor modification with the scheduler and loss computation. Check what should have been changed and complete the `train` and `evaluate` functions to make them work for the current training pipeline. [5 pts]

(f) When training your customized RNN model, search for the best model hyperparameters and try different combinations of training options such as optimizer types, learning rates and learning rate schedulers. Then visualize the experiment logs with TensorBoard. Upon successful training, the test accuracy should be above 70%. [5 pts] For more details, please refer to the inline comments of the skeleton code, lab slides and class materials.

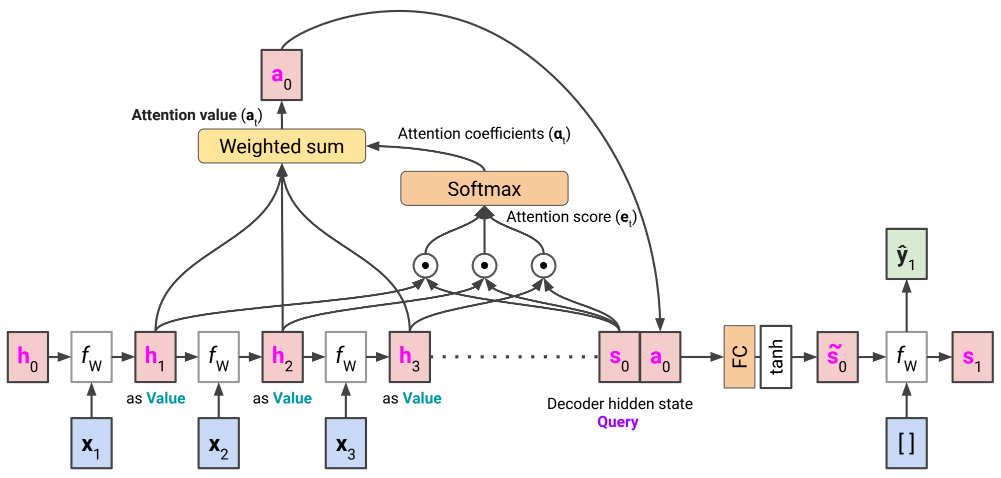

Figure 1: Attention Mechanism of encoder-decoder model

2 Translation with Attention Seq2Seq Model [30 pts]

In this exercise, we are going to implement an LSTM-based Seq2Seq translator with attention mechanism. In the decoder module, dot product attention should be implemented, according to Figure 1. If you want more detail, please refer to class and practice session materials, or original papers.

You have to work on the skeleton code `hw4_seq2seq.ipynb` provided on the ETL. Make sure to download the `data` folder, which includes skeleton code `data.py`.

(a) Implement `forward(self, ...)` with reference to the `__init__(self, ...)` for `LSTMEncoder` and `AttnLSTMDecoder` modules. For both Encoder and Decoder, what to be returned depends on your implementation of `LSTMSeq2Seq` (See Q2-(b)). So feel free to return any output you need. [10 pts]
For more details, please refer to the inline comments of the skeleton code, lab slides and class materials.

(b) Implement `forward(self, ...)` with reference to the `__init__(self, ...)` for `LSTMSeq2Seq`. The Decoder module should attend the Encoder's outputs using dot product attention method. [10 pts]
For more details, please refer to the inline comments of the skeleton code, lab slides and class materials.

(c) Train your Seq2Seq model and plot perplexities and learning rates. Upon successful training, the test perplexity should be less than 7. Briefly report your hyperparameters and results on test dataset. Make sure your results are printed in your submitted file `hw4_seq2seq.ipynb` [10 pts]

3 Vision Transformer [40 pts]

In this exercise, we are going to implement a Vision Transformer for classification task on CIFAR-10 dataset, using all the techniques you have learned. Refer the Figure 2 and part 3.1 in the paper **An Image Is Worth 16x16 Words: Transformers For Image Recognition at Scale** (Link) for the overall architecture of ViT. You have to implement attention mechanism and MLP layer (*i.e.*, Position-wise Feed-Forward layer) based on part 3.2 and part 3.3 in paper **Attention is All You Need** (Link) respectively.

Carefully read the comments in the skeleton code `hw4_vit.ipynb` to successfully implement all the functions. You are **not allowed** to import any other additional modules nor are you allowed to use any built-in layers

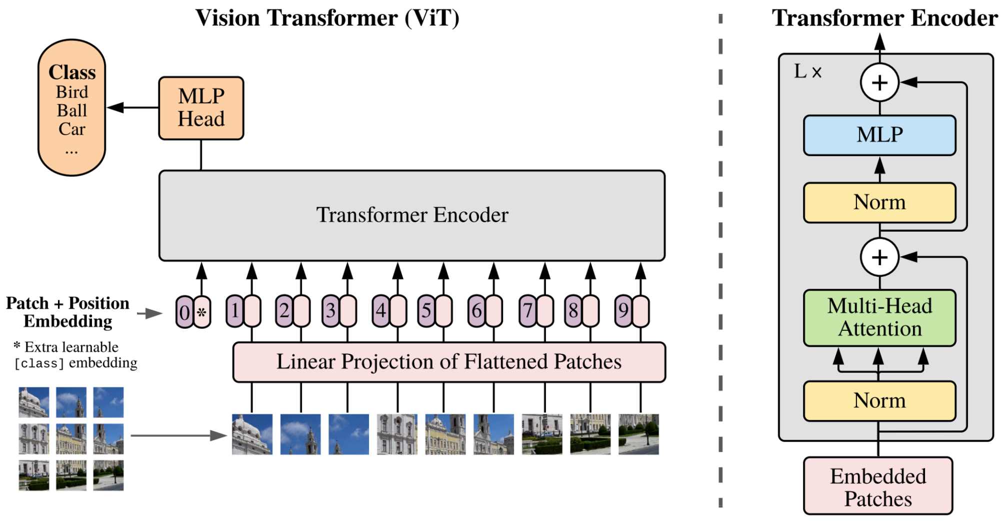

Figure 2: ViT Architecture, An Image Is Worth 16x16 Words: Transformers For Image Recognition at Scale (2021, ICLR)

involved in Transformer model, otherwise you will get 0 point. Specifically, the only built-in layers allowed to use are `torch.nn.Conv2d`, `torch.nn.Linear`, activation layers and normalization layers.

Remember, all your functions should be implemented based on Vision Transformer paper not the original Transformer paper (*e.g.*, activation layer). The original Transformer paper should be referred only for the details of the architecture (*e.g.*, multi-head attention mechanism), which may not be elaborated in the Vision Transformer paper.

(a) Implement `Patchification()` class which processes image to non-overlapping patches using **convolution** layer. You are only allowed to use `torch.nn.Conv2d`. [5 pts]

(b) Implement `Attention()` class referring part 3.2 in the paper Attention is All You Need. You may not consider the *heads* in the attention mechanism, *i.e.*, you are allowed to implement Vanilla Attention not the *Multi-Head* Attention, however your maximum score will be 5 points. [10 pts]

(c) Implement `Block()` class. Attention block is defined as gray colored box in the `Transformer Encoder` shown in the rightmost side of the Figure 2. [10 pts]

(d) Implement `self.patchify`, `self.pos_embedding`, `self.cls_token` in `__init__(self, ...)` and `forward()` method in `ViT` class. Use a **class token** for the classification. [10 pts]

(e) Train your ViT model to achieve **65% of accuracy** on CIFAR10 dataset. Training log and final accuracy should be shown in the output of the cell, otherwise you will get 0 point. [5 pts]

What to Submit

Please upload a single zip file named with your student ID (e.g., `2024-20000.zip`) on eTL, containing

- Your complete `hw4_rnn.ipynb`, `hw4_seq2seq.ipynb` and `hw4_vit.ipynb` files.

- Please erase any unnecessary print codes or comments that you have wrote before submission.