CS7643: Deep Learning
Assignment 4

Instructor: Zsolt Kira

Deadline: 8:00am ET April 1st, 2024

- This assignment is due on the date/time posted on canvas. We will have a 48-hour grace period for this assignment. However, no questions regarding the assignment are answered during the grace period in any form.
- Discussion is encouraged, but each student must write his/her own answers and explicitly mention any collaborators.
- Each student is expected to respect and follow the GT Honor Code. **We will apply anti-cheating software to check for plagiarism**. Anyone who is flagged by the software will automatically receive 0 for the homework and be reported to OSI.
- Please **do not change the filenames and function definitions** in the skeleton code provided, as this will cause the test scripts to fail and you will receive no points in those failed tests. You may also **NOT** change the import modules in each file or import additional modules.
- It is your responsibility to make sure that all code and other deliverables are in the correct format and that your submission compiles and runs. We will not manually check your code (this is not feasible given the class size). Thus, **non-runnable code in our test environment will directly lead to a score of 0**. Also, your entire programming parts will **NOT** be graded and given 0 score if your code prints out anything that is not asked in each question.

Theory Problem Set

1. Given a 4 length sequence RNN, compute $\frac{\partial L_2}{\partial b}$ and show how you derived it. In addition, please provide its computation graph.

   The following equations might be helpful:

$$\begin{aligned} a_t &= Ux_t + Wh_{t-1} + b \\ h_t &= tanh(a_t) \\ o_t &= Vh_t + c \\ \hat{y}_t &= Softmax(o_t) \\ L_t &= CE(\hat{y}_t, y_t) \\ f(x) = tanh(x) &\longrightarrow f^{'}(x) = 1 - tanh^2(x) \end{aligned}$$

   Hint: You can combine softmax and CE into a single derivative as follows:

$$\frac{\partial L_t}{\partial o_t} = \hat{y}_t - y_t$$

Paper Review

In this section, you must choose **one** of the papers below and complete the following:

1. provide a short review of the paper,
2. answer paper-specific questions,

Guidelines: Please restrict your reviews to no more than 350 words and answers to questions to no more than 350 words per question. The review part (1) should include answers to the following:

3. What is the main contribution of this paper? In other words, briefly summarize its key insights. What are some strengths and weaknesses of this paper?
4. What is your personal takeaway from this paper? This could be expressed either in terms of your perceived novelty of this paper compared to others you've read in the field, potential future directions of research in the area that the authors haven't addressed, or anything else that struck you as being noteworthy.

Paper Choice 1:

Up until recently, convolutional neural networks (CNN) have been the state of the art for computer vision (CV) tasks. With the recent introduction of vision transformers (ViT), there has been research into using this new architecture for CV. There have been instances where ViTs have either achieved similar or even superior accuracy to CNNs. The biggest question is, how do they see? Is it similar to CNNs? Is it different? This paper attempts to answer these questions.
The paper can be viewed here.

Questions for this paper:

- Compare and contrast the learned features of ViTs and CNNs? For differences between the two, please provide explanations in terms of network architecture and training.
- What is meant by spatial localization? And why might we consider the use of ViTs better for object detection?

Paper Choice 2:

The second paper focuses on the generalization ability of pre-trained language models, specifically zero-shot learning (i.e., performing a new task with no labels at all). It turns out these models (like GPT-3) are surprisingly effective zero-shot learners. The paper can be viewed here.

Questions for this paper:

- How might we approach the technical limitations (e.g., changes in architecture, other context/data, optimization, etc.) mentioned in sections 5/6?
- What are the social implications of deploying these models for various uses (e.g., to generate image captions, answer questions as chatbots, etc.)?

1 Code: RNN, LSTM, Seq2Seq, and Transformer models

In this assignment, we will work with **Python 3**. If you do not have a python distribution installed yet, we recommend installing Anaconda (or mini-conda) with Python 3 (3.8.10 recommended).We have provided you with a conda environment YAML file to assist you in setting up the code environment. Alternatively, there is a requirements.txt file if you decide to use pip to install the required packages. You should make sure you have the following packages installed

```
$ pip install torchtext
$ pip install torch
$ pip install spacy
$ pip install tqdm
$ pip install numpy
```

Additionally, you will need the Spacy tokenizers in English and German language, which can be downloaded as such:

```
$python -m spacy download en_core_web_sm
$python -m spacy download de_core_news_sm
```

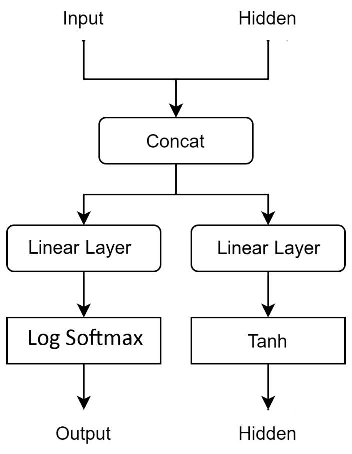
In your assignment you will see the notebook `Machine_Translation.ipynb` which contains test cases and shows the training progress. You can follow that notebook for instructions as well.

2 RNNs and LSTMs

In `models/naive` you will see files necessary to complete this section. In both of these files you will complete the initialization and forward pass.

2.1 RNN Unit

You will be using PyTorch Linear layers and activations to implement a vanilla RNN unit. Please refer to the following structure and complete the code in `RNN.py`:

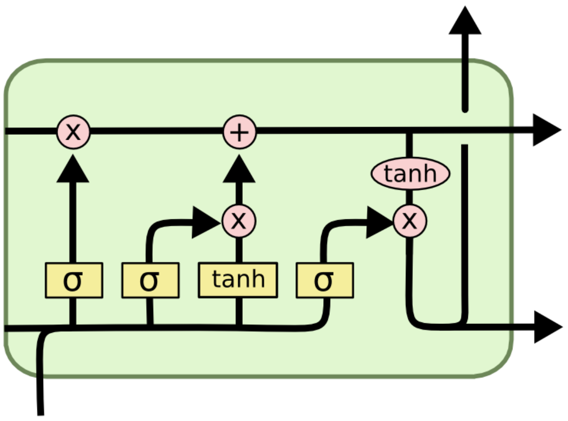


2.2 LSTM

You will be using PyTorch nn.Parameter and activations to implement an LSTM unit. You can simply translate the following equations using nn.Parameter and PyTorch activation functions to build an LSTM from scratch:

$$
\begin{aligned}
i_t &= \sigma(x_t.W_{ii} + b_{ii} + h_{t-1}.W_{hi} + b_{hi}) \\
f_t &= \sigma(x_t.W_{if} + b_{if} + h_{t-1}.W_{hf} + b_{hf}) \\
g_t &= \tanh(x_t.W_{ig} + b_{ig} + h_{t-1}.W_{hg} + b_{hg}) \\
o_t &= \sigma(x_t.W_{io} + b_{io} + h_{t-1}.W_{ho} + b_{ho}) \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned}
$$

Here's a great visualization of the above equation from Colah's blog to help you understand LSTM unit.



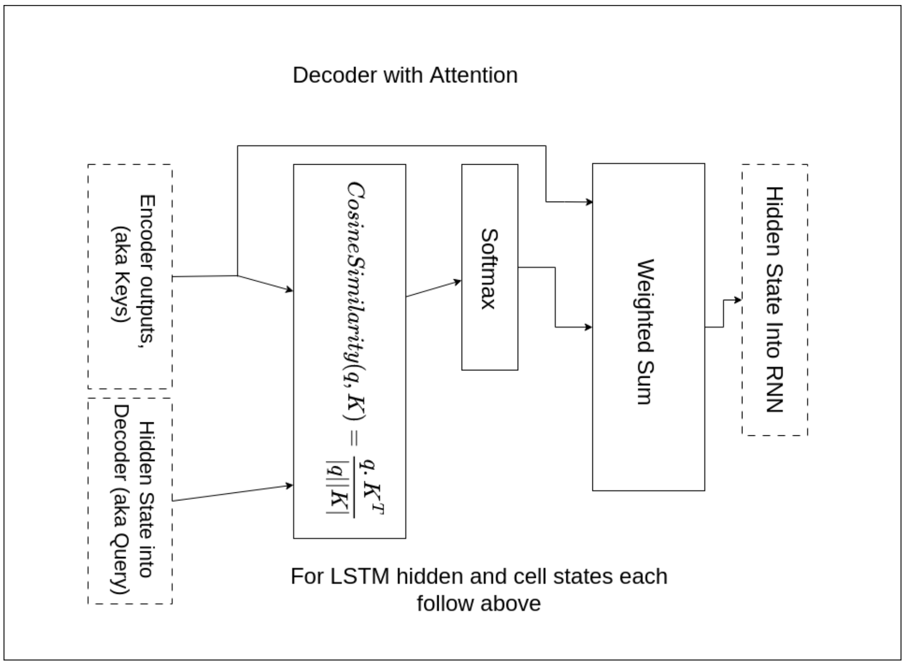
If you want to see nn.Parameter in example, check out this tutorial from PyTorch.

3 Seq2Seq Implementation

In `models/seq2seq` you will see the files needed to complete this section. In these files you will complete the initialization and forward pass in `__init__` and `forward` function. `Encoder.py` `Decoder.py` `Seq2Seq.py`

3.1 Seq2Seq with attention

We will be implementing a simple form of attention to evaluate how it impacts the performance of our model. In particular, we will be implementing cosine similarity as the attention mechanism in decoder.py per this diagram. Please pay attention to comments in TODO sections of the code for more detail.


Decoder with Attention
Encoder outputs, (aka Keys)
Hidden State into Decoder (aka Query)
$CosineSimilarity(q, K) = \frac{q.K^T}{|q||K|}$
Softmax
Weighted Sum
Hidden State Into RNN
For LSTM hidden and cell states each follow above

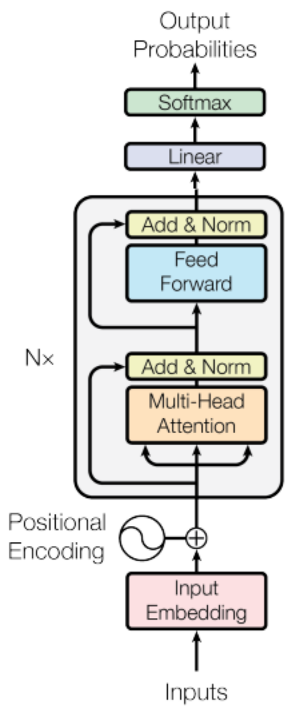
To learn more about attention and how it is used in Neural Machine Translation, we recommend (but not required) that you read "Neural machine translation by jointly learning to align and translate (Bahdanau et al)" and "Effective Approaches to Attention-based Neural Machine Translation (Luong et al)".

3.2 Training and Hyperparameter Tuning

Train seq2seq on the dataset with the default hyperparameters. Then perform hyperparameter tuning and include the improved results in a report explaining what you have tried. Do **NOT** just increase the number of epochs or change the model type (RNN to LSTM) as this is too trivial.

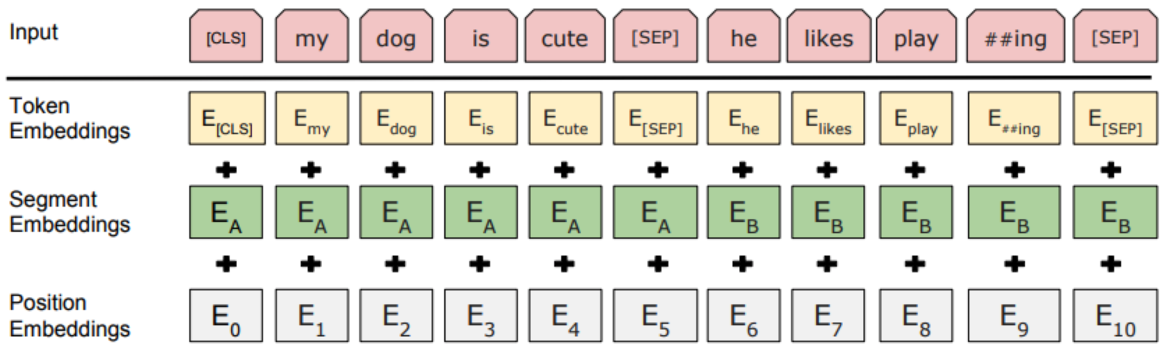
4 Transformers

We will be implementing a one-layer Transformer encoder which, similar to an RNN, can encode a sequence of inputs and produce a final output of possibility of tokens in target language. The architecture can be seen below.
You can refer to the original paper for more detail. In `models` you will see the file `Transformer.py`. You will implement the functions in the `TransformerTranslator` class.



4.1 Embeddings

We will format our input embeddings similarly to how they are constructed in [BERT (source of figure)](https://arxiv.org/pdf/1810.04805.pdf). Recall from lecture that unlike a RNN, a Transformer does not include any positional information about the order in which the words in the sentence occur. Because of this, we need to append a positional encoding token at each position. (We will ignore the segment embeddings and [SEP] token here, since we are only encoding one sentence at a time). We have already appended the [CLS] token for you in the previous step.


Input
[CLS]
my
dog
is
cute
[SEP]
he
likes
play
##ing
[SEP]
Token Embeddings
$E_{[CLS]}$ E_{my} E_{dog} E_{is} E_{cute} $E_{[SEP]}$ E_{he} E_{likes} E_{play} $E_{\#\#ing}$ $E_{[SEP]}$
Segment Embeddings
E_A E_A E_A E_A E_A E_A E_B E_B E_B E_B E_B
Position Embeddings
E_0 E_1 E_2 E_3 E_4 E_5 E_6 E_7 E_8 E_9 E_{10}

Your first task is to implement the embedding lookup, including the addition of positional encodings. Complete the code section for **Deliverable 1**, which will include part of `__init__` and `embed`.

4.2 Multi-head Self-Attention

Attention can be computed in matrix-form using the following formula:

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

We want to have multiple self-attention operations, computed in parallel. Each of these is called a *head*. We concatenate the heads and multiply them with the matrix *attention_head_projection* to produce the output of this layer.
After every multi-head self-attention and feedforward layer, there is a residual connection + layer normalization. Make sure to implement this, using the following formula:

$$\text{LayerNorm}(x + \text{Sublayer}(x))$$

Implement the function `multi_head_attention` for **Deliverable 2**. We have already initialized all of the layers you will need in the constructor.

4.3 Element-Wise Feedforward Layer

Complete code for **Deliverable 3** in `feedforward_layer`: the element-wise feed-forward layer consisting of two linear transformers with a ReLU layer in between.

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

4.4 Final Layer

Complete code for **Deliverable 4** in `final_layer`., to produce probability scores for all tokens in target language.

4.5 Forward Pass

Put it all together by completing the method `forward`, where you combine all of the methods you have developed in the right order to perform a full forward pass.

4.6 Training

Train the transformer encoder architecture on the dataset with the default hyperparameters – you should get a perplexity better than that for seq2seq.

4.7 Implement and train a full transformer (encoder/decoder)

In the previous section, you implemented the encoder module of a transformer from scratch. In this section, you use pytorch built-in transformer module to build your translator model. You will implement the functions in the `FullTransformerTranslator` class. Please follow the instructions in the TODO sections of the code for implementation details. Train your model with hyper-parameter tuning. You are asked to explain your results and compare this model to other models you implemented in the assignment.

5 Deliverables

You will need to submit the notebook as well as your code in the `models` folder.
Run the script collect_submission.py to generate a zip folder with all the required code files. Upload the resulting zip folder to Gradescope.
You will need to follow the guidance and fill in the report template.Your report should include the performance metrics of the Seq2Seq model and Transformer architecture before and after hyper-parameter tuning with explanations of what you did and why. You also need to include the answers to the theory question and the section on paper review in your report.**When submitting to Gradescope, make sure you select ALL corresponding slides for each question. Failing to do so will result in point deductions.**