CS 4644/7643: Deep Learning
Assignment 4

Instructor: Zsolt Kira
TAs: Ting-Yu Lan, Anshul Ahluwalia, Ahmed Shaikh, Aaditya Singh, Yash Jain, Yash Jakhotiya, Hoon Lee, Zach Minot, Sumedh Vijay, Ningyuan Yang, Jan Vijay Singh, Aaditya Singh
Discussions: https://piazza.com/gatech/spring2023/cs46447643

Deadline: 11:59 pm April 8, 2023

- 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: https://osi.gatech.edu/content/honor-code. **We will apply anti-cheating software to check for plagiarism**. We cross-check source code within the class, with previous years' solutions, and with online solutions. Any case that deemed substantial by the teaching team will be reported to OSI and receive a 0.
- 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.

1 Assignment Setup

In this assignment, we will work with **Python 3**. If you do not have a python distribution installed yet, we recommend installing Anaconda (or miniconda) with Python 3 (3.8.10 recommended). You should make sure you have the following packages installed

```
$ pip install torchtext==0.14.1
$ pip install torch==1.13.1
$ pip install spacy==3.4.4
$ 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
```

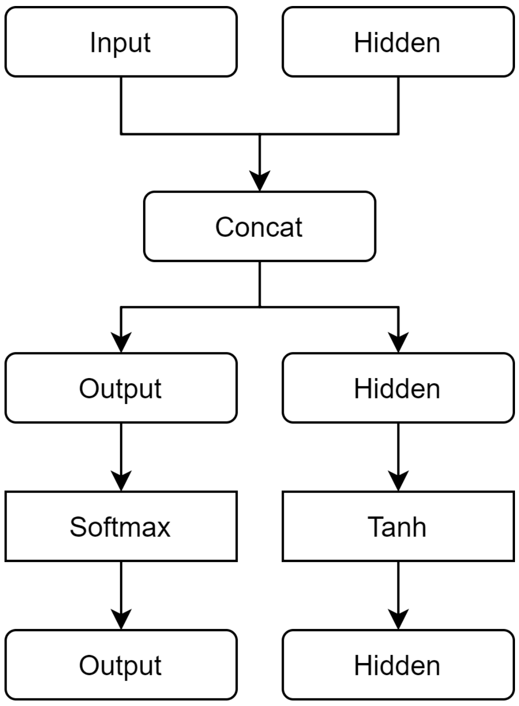
In your assignment you will see the notebook `Machine_Translation.ipynb` which contains test cases and shows the training progress. You can follow that notebook for instructions as well.

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 forward pass.

2.1 RNN Unit (2 points)

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`:

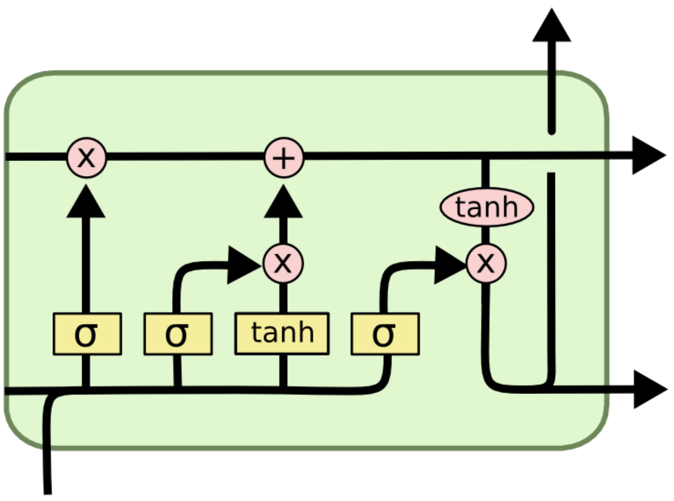


2.2 LSTM (2 points)

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:

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

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.



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

3 Seq2Seq Implementation

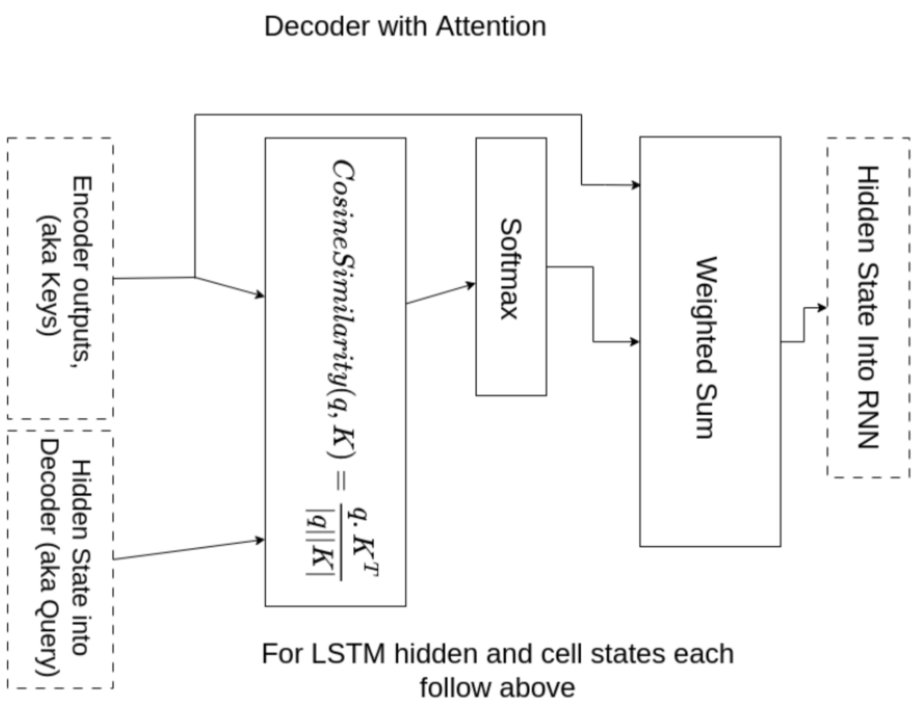
In `models/seq2seq` you will see the files needed to complete this section.

3.1 Encoder, Decoder and Seq2seq (4 points)

Please follow the instructions to complete the initialization and forward pass in `__init__` and `forward` function of `Encoder.py`, `Decoder.py` and `Seq2Seq.py`.

3.2 Seq2Seq with Attention (1 point)

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

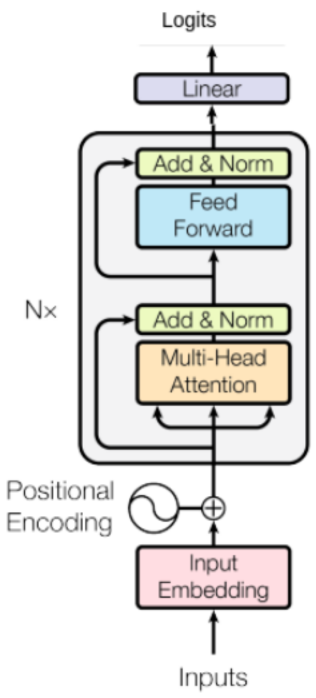
3.3 Training and Hyperparameter Tuning (Report: 4 points)

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. These include but not limited to dropout, embedding size, hidden size, and learning rate.

4 Transformers

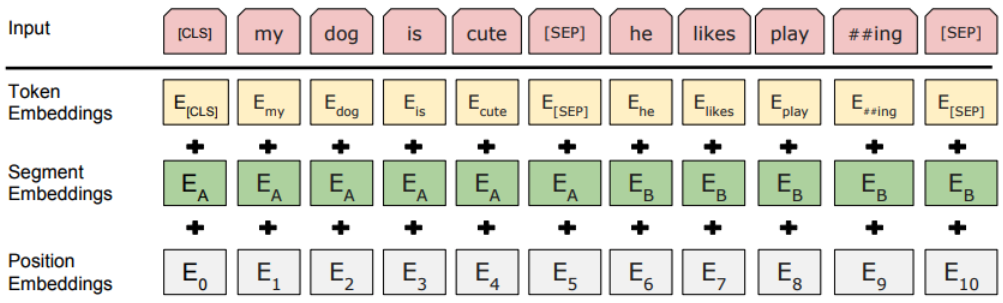
We will be implementing a one-layer Transformer encoder which, similar to an RNN, can encode a sequence of inputs and produce a final output of possibility of tokens in target language. The architecture can be seen below.

You can refer to the original paper for more information. In `models`, you will see the file `Transformer.py`. You will need to implement the functions in the `TransformerTranslator` class.



4.1 Embeddings (1 point)

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] Emy Edog Eis Ecute E[SEP] Ehe Elikes Eplay E##ing E[SEP]
Segment Embeddings
EA EA EA EA EA EA EB EB EB EB EB
Position Embeddings
E0 E1 E2 E3 E4 E5 E6 E7 E8 E9 E10

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 (1 point)

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` to do this. We have already initialized all of the layers you will need in the constructor.

4.3 Element-Wise Feedforward Layer (1 point)

Complete code for **Deliverable 3** in `feedforward_layer`: Include layer norm and addition as per transformer diagram

4.4 Final Layer (1 point)

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

4.5 Forward Pass (1 point)

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 and Hyperparameter Tuning (Report: 5 points)

Train the transformer architecture on the dataset with the default hyperparameters – you should get a perplexity better than that for seq2seq. 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 as this is too trivial. These include but not limited to feedforward size, hidden size, and learning rate.

5 Deliverables

You will need to run the entire `Machine_Translation.ipynb` notebook, save it as a PDF, and submit it to the HW4 Report section on Gradescope. You will need to report the accuracy of the Seq2Seq model and Transformer architecture before and after hyperparameter tuning with explanations of what you did in the given sections of the notebook.

Run the last cell in `Machine_Translation.ipynb` to generate a zip folder with all of the required code files. Upload the zip folder to Gradescope under HW4 Code section.