

CS7643: Deep Learning

Assignment 4

Instructor: Zsolt Kira

Deadline: 11:59pm AOE April 3, 2022

- 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.

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. Given that you should already have your PyTorch installed in your local anaconda environment in assignment 2, we do not provide environment files for you to setup anaconda environment. You should make sure you have the following packages installed

```
$ pip install torchtext==0.8.1
$ pip install torch==1.7.1
$ pip install spacy==2.3.5
$ 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
$python -m spacy download de
```

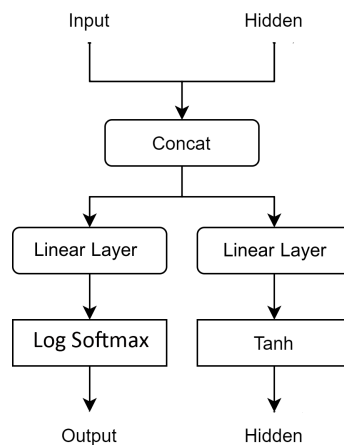
In your assignment you will see the notebook `Machine_Translation.ipynb` which contains test cases and shows the training progress. You can follow that notebook for instructions as well.

1 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.

1.1 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`:

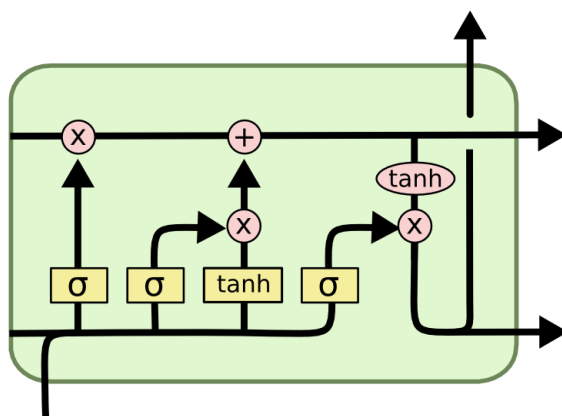


1.2 LSTM

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:

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

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.

2 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`

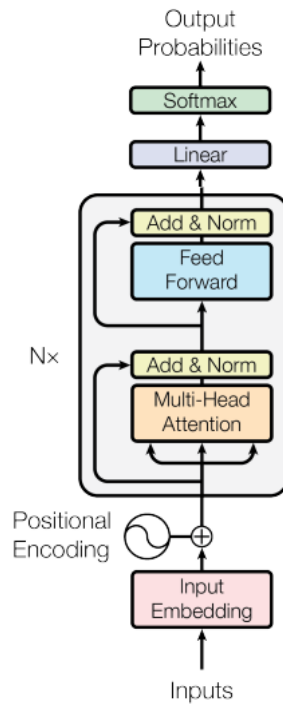
2.1 Training and Hyperparameter Tuning

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.

3 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](#). In `models` you will see the file `Transformer.py`. You will implement the functions in the `TransformerTranslator` class.



3.1 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	$E_{[\text{CLS}]}$	E_{my}	E_{dog}	E_{is}	E_{cute}	$E_{[\text{SEP}]}$	E_{he}	E_{likes}	E_{play}	$E_{\text{##ing}}$	$E_{[\text{SEP}]}$
	+	+	+	+	+	+	+	+	+	+	+
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`.

3.2 Multi-head Self-Attention

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

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

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.

3.3 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.

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

3.4 Final Layer

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

3.5 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.

3.6 Training and Hyperparameter Tuning

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.

4 Theory Problem Set

1. Beam Search is a widely-used technique for decoding the most likely sequence from sequence models. But it is difficult to decide when to stop beam search to obtain optimality because hypotheses can finish in different steps. In this question, we will develop a formal understanding of the stopping criteria in beam search.

Let \mathbf{x} denote the input upon which we condition our sequence model. Let \mathbf{y} denote the output sequence. Let $\mathbf{y}_{<t}$ be a shorthand notation for the sub-sequence

$(y_0, y_1, \dots, y_{t-1})$. We say that a sequence (or hypothesis as they are sometimes referred to in this literature) \mathbf{y} is *completed* ($\text{comp}(\mathbf{y}) = \text{true}$), if its last token is $\langle /s \rangle$, *i.e.*,

$$\text{comp}(\mathbf{y}) = \text{true} \leftrightarrow (\mathbf{y}_{|\mathbf{y}|} = \langle /s \rangle)$$

in which case it will not be further expanded in beam search.

With this notation, we can write down the maximum a-posteriori inference problem as:

$$\mathbf{y}^* = \underset{\mathbf{y}}{\text{argmax}} \prod_{t \leq |\mathbf{y}|} p(y_t | \mathbf{x}, \mathbf{y}_{<t}) \quad (1)$$

$$\text{s.t.} \quad \text{comp}(\mathbf{y}) = \text{true} \quad (2)$$

We use beam search to find the (approximate) best output \mathbf{y}^* . At time t , let B_{t-1} denote the beams so far. Thus, B_{t-1} is a b -length list consisting of $\langle \mathbf{y}, s \rangle$ pairs, *i.e.*, $B_{t-1} = (\langle \mathbf{y}^1, s^1 \rangle, \dots, \langle \mathbf{y}^b, s^b \rangle)$, where \mathbf{y}^i is a $(t-1)$ -length sequence (a beam) and s^i is its associated score (sum of log-conditional probabilities), *i.e.* $s^i = \sum_{j=1}^{t-1} \log p(y_j^i | \mathbf{x}, \mathbf{y}_{<j}^i)$.

Let \circ denote a concatenation operation, *i.e.* $\mathbf{y} \circ y_t$ represents a beam expansion where \mathbf{y} is concatenated with y_t . Beam search can then be formalized via a top^b operator that selects (quite literally) the top- b scoring items in an expanded list of beams:

$$B_t = \text{top}^b \{ \langle \mathbf{y} \circ y_t, s + \log p(y_t | \mathbf{x}, \mathbf{y}_{<t}) \rangle \mid \langle \mathbf{y}, s \rangle \in B_{t-1} \} \quad (3)$$

Let $\text{best}_{\leq t}$ be the best completed hypothesis so far (up to step t), *i.e.*

$$\text{best}_{\leq t} \triangleq \max \{ s \mid \langle \mathbf{y}, s \rangle \in \cup_{j \leq t} B_j, \text{comp}(\mathbf{y}) = \text{true} \} \quad (4)$$

Notice that if there no completed beam till time t , $\text{best}_{\leq t}$ is undefined/empty.

Now, for the proof.

Assuming that $\text{best}_{\leq t}$ is defined at time t and the current highest scoring beam in B_t (*i.e.* \mathbf{y}^1) scores worse than or equal to $\text{best}_{\leq t}$, *i.e.* $s^1 \leq \text{best}_{\leq t}$, prove that there is no need to run beam search out further. That is, prove that the current best completed beam (corresponding to $\text{best}_{\leq t}$) is the overall highest-probability completed beam and future steps will be no better.

5 Deliverables

You will need to submit the notebook as well as your code in the `models` folder. Your report should include the accuracy of the Seq2Seq model and Transformer architecture before and after hyperparameter tuning with explanations of what you did.