Assignments

M3224.000300 Natural Language Processing with Neural Networks

Fall 2025

Graduate School of Data Science
Seoul National University

Instructor: Jay-yoon Lee (lee.jayyoon@snu.ac.kr)
TA: Sunah O (ec_osa16586@snu.ac.kr)
Yeonsung Kim (yeonsungkim@snu.ac.kr)

Due on Wednesday Nov. 5, 2025 by 23:59 pm

(Assignment 3: Recurrent Neural Networks)

Honor Pledge for Graded Assignments

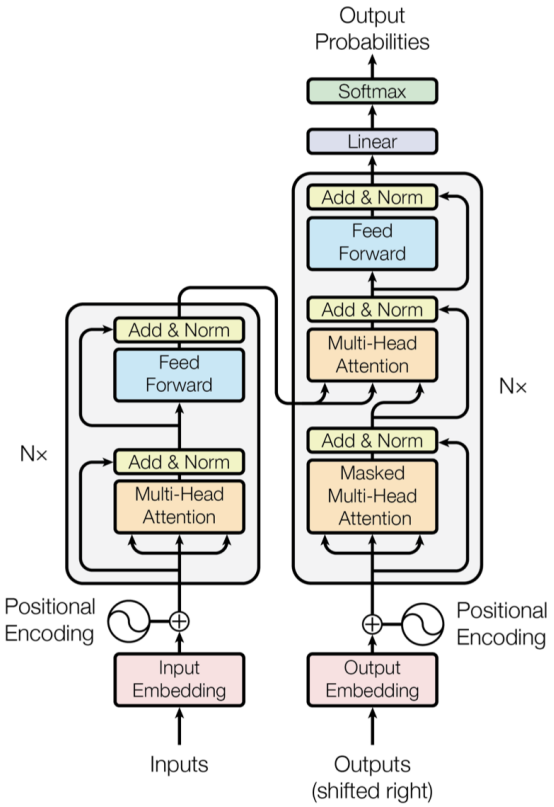
"I, YOUR NAME HERE , affirm that I have not given or received any unauthorized help on this assignment, and that this work is my own."

0 Instructions

- Total score cannot exceed 100 points. For example, if you score 98 points from non-bonus questions and 3 points are added from bonus questions, your score will be 100 points, not 101 points.
- Skeleton codes for problem 3 are at the directory `/q3`.
- Run the `bash collect_submission.sh` script to produce your 2000_00000_coding.zip file. Please make sure to modify collect_submission.sh file before running this command. (**2000_00000** stands for your student id.)
- Modify this tex file into `2000_00000_written.pdf` with your written solutions.
- Upload both `2000_00000_coding.zip` and `2000_00000_written.pdf` to etl website.
- **If the submission instructions are not followed,** `4 points` **will be deducted.**

1 Transformer (34 pts)

1.1 Transformer Encoder Block Parameter Counting (23 pts)



Consider a single Transformer encoder block with the following settings:

- Model dimension: $d_{\text{model}} = 512$
- Multi-Head Attention:
  - Number of heads: 8
  - Dimension of each head (Query, Key, Value): 64
  - Output projection dimension: d_{model}
- Feed-Forward Network (FFN):
  - First linear: $d_{\text{model}} \to d_{ff} = 2048$
  - Second linear: $d_{ff} \to d_{\text{model}}$
- All linear layers include bias terms.
- Layer Normalization appears twice, each with scale (γ) and bias (β) parameters.

(a) Compute the total parameters for Query, Key, and Value projections in Multi-Head Attention. **(3 pts) Answer:**

(b) Compute the total parameters of the Multi-Head Attention module when including the output projection as well. **(3 pts) Answer:**

(c) Compute the parameters of the Feed-Forward Network:

i) First linear layer ($512 \to 2048$). **(2 pts)**

ii) Second linear layer ($2048 \to 512$). **(2 pts)**

iii) Report the total parameters of the FFN. **(3 pts)**

Answer:

(d) Compute the total parameters of the two LayerNorm layers combined. **(3 pts) Answer:**

(e) Extended Case: Suppose the architecture is modified as follows:

- The Feed-Forward dimension is expanded to $d_{ff} = 4 \times d_{\text{model}} = 4096$.
- The number of attention heads is increased to 16, while $d_{\text{model}} = 512$ remains unchanged.
- All other conditions remain the same.

Recalculate the total number of parameters under these new settings, and compute the percentage increase compared to the original architecture. **(7 pts) Answer:**

1.2 Dot-Product Attention Exercise (11 pts)

A single-head Transformer processes the sentence [`cat`, `on`, `the`, `mat`]. We compute scaled dot-product attention *from the perspective of the query token* `cat`. Key/query/value dimension is $d_k = d_v = 3$, so the scaling factor is $\sqrt{3} \approx 1.732$. Round final answers to 3 decimals.

You may use code (e.g., Python/NumPy, MATLAB, or R) for all calculations.

Token embeddings (row vectors). Let the sentence be [`cat`, `on`, `the`, `mat`] with

$$e_{\texttt{cat}} = [0.8,\ 0.1,\ 0.5], \qquad e_{\texttt{on}} = [-0.2,\ 0.3,\ 0.0],$$
$$e_{\texttt{the}} = [0.1,\ -0.1,\ -0.2], \qquad e_{\texttt{mat}} = [2.4,\ 0.3,\ 1.6].$$

(Values are chosen so that `cat` tends to attend strongly to `mat`.)

Weight matrices. We use row-vector convention ($Q = EW_Q$, etc.). Let

$$W_Q = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \qquad W_K = I_3, \qquad W_V = \begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0.5 \end{bmatrix}.$$

Thus $d_k = 3$ and the usual scaling factor is $\sqrt{d_k} = \sqrt{3} \approx 1.732$. Unless otherwise noted, round final numeric answers to **3 decimal places**.

(a) Compute Q, K, and V. (3 pts)
Using the definitions $Q = EW_Q$, $K = EW_K$, $V = EW_V$ with the matrices above, compute

$$Q_{\texttt{cat}} = e_{\texttt{cat}} W_Q, \quad K_w = e_w W_K, \quad V_w = e_w W_V \quad (w \in \{\texttt{cat,on,the,mat}\}).$$

Answer:

(b) Compute Attention Scores (Raw and Scaled). (3 pts)
For the query token `cat`, first compute the *unscaled* logits against *all* targets $w \in \{\texttt{cat,on,the,mat}\}$

$$s(\texttt{cat} \to w) \ = \ \langle Q_{\texttt{cat}},\, K_w \rangle \ = \ Q_{\texttt{cat}}\, K_w^{\top}.$$

Then convert to *scaled* logits using $d_k = 3$:

$$\tilde{s}(\texttt{cat} \to w) \ = \ \frac{s(\texttt{cat} \to w)}{\sqrt{3}}, \qquad \sqrt{3} \approx 1.732.$$

Answer:

(c) Compute Normalized Attention Distribution (Softmax)(3 pts)
Apply softmax over the *scaled* logits from (b) to obtain the attention distribution

$$\alpha_{\texttt{cat}\to w} \;=\; \frac{\exp\big(\tilde{s}(\texttt{cat}\to w)\big)}{\sum_{u\in\{\texttt{cat,on,the,mat}\}}\exp\big(\tilde{s}(\texttt{cat}\to u)\big)}\,,$$

for each $w \in \{\texttt{cat,on,the,mat}\}$.

Answer:

(d) Compare Scaling Effects: Scaled vs. Unscaled Attention. (2 pts)
Let $\alpha^{(\text{scaled})}$ be the softmax over the *scaled* logits from (b), and let $\alpha^{(\text{raw})}$ be the softmax over the *unscaled* logits from (b).

$$\alpha^{(\text{scaled})}_{\texttt{cat}\to w} = \frac{e^{\tilde{s}(\texttt{cat}\to w)}}{\sum_u e^{\tilde{s}(\texttt{cat}\to u)}}, \qquad \alpha^{(\text{raw})}_{\texttt{cat}\to w} = \frac{e^{s(\texttt{cat}\to w)}}{\sum_u e^{s(\texttt{cat}\to u)}}.$$

Explain the effect of scaling on the resulting attention distribution.

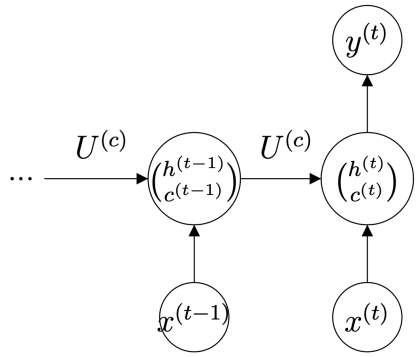
Answer:

2 Backprop on LSTM (12 pts)

In class, we learned about Long Short-Term Memory (LSTM) model. Recall the units of an LSTM cell are defined as

$$\begin{aligned} i_t &= \sigma(W^{(i)}x_t + U^{(i)}h_{t-1}) \\ f_t &= \sigma(W^{(f)}x_t + U^{(f)}h_{t-1}) \\ o_t &= \sigma(W^{(o)}x_t + U^{(o)}h_{t-1}) \\ \tilde{c_t} &= tanh(W^{(c)}x_t + U^{(c)}h_{t-1}) \\ c_t &= f_t \circ c_{t-1} + i_t \circ \tilde{c_t} \\ h_t &= o_t \circ tanh(c_t) \end{aligned}$$

where the final output of the last lstm cell is defined by $\hat{y}_t = softmax(h_t W + b)$. The final cost function J uses the cross-entropy loss. Consider an LSTM for two time steps, t and $t-1$.



(a) Derive the gradient $\frac{\delta J}{\delta U^{(c)}}$ in terms of the following gradients: $\frac{\delta h_t}{\delta h_{t-1}}$, $\frac{\delta h_{t-1}}{\delta U^{(c)}}$, $\frac{\delta J}{\delta h_t}$, $\frac{\delta c_t}{\delta U^{(c)}}$, $\frac{\delta c_{t-1}}{\delta U^{(c)}}$, $\frac{\delta c_t}{\delta c_{t-1}}$, $\frac{\delta h_t}{\delta c_t}$, and $\frac{\delta h_t}{\delta o_t}$. *Not all of the gradients may be used.* You can leave the answer in the form of chain rule and do not have to calculate any individual gradients in your final result. **(8 pts) Answer:**

(b) Using your result from part (a) for $\frac{\delta J}{\delta U^{(c)}}$ in the LSTM *and* the vanilla RNN chain rule below, explain *which multiplicative factor(s)* in the LSTM gradient mitigate the vanishing–gradient problem *and why* . (Comparison) Vanilla RNN chain rule: If $h_t = \phi(Wx_t + Uh_{t-1})$ and J depends only on h_t, then for two successive time steps,

$$\frac{\delta J}{\delta U} = \frac{\delta J}{\delta h_t}\,\frac{\delta h_t}{\delta h_{t-1}}\,\frac{\delta h_{t-1}}{\delta U}.$$

(4 pts) Answer:

3 Neural Machine Translation with LSTM (54+3 pts)

In Neural Machine Translation (NMT), our goal is to convert a sentence from the *source* language to the *target* language. In this assignment, we will implement a sequence-to-sequence (Seq2Seq) network with attention, to build a Neural Machine Translation (NMT) system between Jeju dialect and Korean.

3.1 Training Procedure

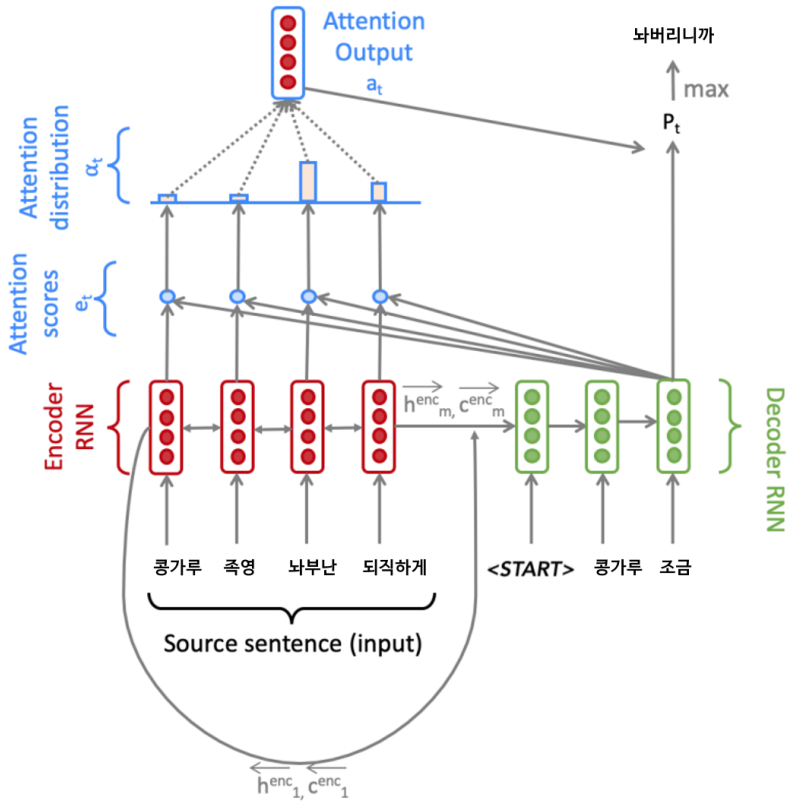


Figure 1: Seq2Seq Model with Multiplicative Attention, shown on the third step of the decoder. (NOTE: Embedding of NMT in the assignment differs from described above.)

In this section, we describe the **training procedure** for the proposed NMT system, which uses a Bidirectional LSTM Encoder and a Unidirectional LSTM Decoder. The model is trained and evaluated on JIT (Jejueo interview transcripts) dataset[1]. Given a sentence in the source language (Jeju dialect), we look up the subword embeddings from an embeddings matrix, yielding $\mathbf{x}_1, \ldots, \mathbf{x}_m$ ($\mathbf{x}_i \in \mathbb{R}^{e \times 1}$), where m is the length of the source sentence and e is the embedding size. We feed these embeddings to the bidirectional encoder, yielding hidden states and cell states for both the forwards ($\rightarrow$) and backwards ($\leftarrow$) LSTMs. The forwards and backwards versions are concatenated to make hidden states $\mathbf{h}_i^{enc}$ and cell states $\mathbf{c}_i^{enc}$:

$$\mathbf{h}_i^{\text{enc}} = [\overleftarrow{\mathbf{h}_i^{enc}}; \overrightarrow{\mathbf{h}_i^{enc}}] \text{ where } \mathbf{h}_i^{\text{enc}} \in \mathbb{R}^{2h \times 1}, \overleftarrow{\mathbf{h}_i^{enc}}, \overrightarrow{\mathbf{h}_i^{enc}} \in \mathbb{R}^{h \times 1} \qquad 1 \leq i \leq m \tag{1}$$

$$\mathbf{c}_i^{\text{enc}} = [\overleftarrow{\mathbf{c}_i^{enc}}; \overrightarrow{\mathbf{c}_i^{enc}}] \text{ where } \mathbf{c}_i^{\text{enc}} \in \mathbb{R}^{2h \times 1}, \overleftarrow{\mathbf{c}_i^{enc}}, \overrightarrow{\mathbf{c}_i^{enc}} \in \mathbb{R}^{h \times 1} \qquad 1 \leq i \leq m \tag{2}$$

We then initialize the decoder's first hidden state $\mathbf{h}_0^{\text{dec}}$ and cell state $\mathbf{c}_0^{\text{dec}}$ with a linear projection of the encoder's final hidden state and final cell state.[2]

[1] https://www.kaggle.com/datasets/bryanpark/jit-dataset

[2] Note that we regard $[\overleftarrow{\mathbf{h}_1^{enc}}, \overrightarrow{\mathbf{h}_m^{enc}}]$ as the 'final hidden state' of the Encoder.

$$\mathbf{h}_0^{\text{dec}} = \mathbf{W}_h[\overleftarrow{\mathbf{h}_1^{enc}}; \overrightarrow{\mathbf{h}_m^{enc}}] \text{ where } \mathbf{h}_0^{\text{dec}} \in \mathbb{R}^{h\times 1}, \mathbf{W}_h \in \mathbb{R}^{h\times 2h} \tag{3}$$

$$\mathbf{c}_0^{\text{dec}} = \mathbf{W}_c[\overleftarrow{\mathbf{c}_1^{enc}}; \overrightarrow{\mathbf{c}_m^{enc}}] \text{ where } \mathbf{c}_0^{\text{dec}} \in \mathbb{R}^{h\times 1}, \mathbf{W}_c \in \mathbb{R}^{h\times 2h} \tag{4}$$

With the decoder initialized, we must now feed it a target sentence. On the t^{th} step, we look up the embedding for the t^{th} subword, $\mathbf{y}_t \in \mathbb{R}^{e\times 1}$. We then concatenate $\mathbf{y}_t$ with the *combined-output vector* $\mathbf{o}_{t-1} \in \mathbb{R}^{h\times 1}$ from the previous timestep (we will explain what this is later down this page!) to produce $\overline{\mathbf{y}_t} \in \mathbb{R}^{(e+h)\times 1}$. Note that for the first target subword (i.e. the start token) $\mathbf{o}_0$ is a zero-vector. We then feed $\overline{\mathbf{y}_t}$ as input to the decoder.

$$\mathbf{h}_t^{\text{dec}}, \mathbf{c}_t^{\text{dec}} = \text{Decoder}(\overline{\mathbf{y}_t}, \mathbf{h}_{t-1}^{\text{dec}}, \mathbf{c}_{t-1}^{\text{dec}}) \text{ where } \mathbf{h}_t^{\text{dec}} \in \mathbb{R}^{h\times 1}, \mathbf{c}_t^{\text{dec}} \in \mathbb{R}^{h\times 1} \tag{5}$$

$$\tag{6}$$

We then use $\mathbf{h}_t^{\text{dec}}$ to compute multiplicative attention over $\mathbf{h}_1^{\text{enc}}, \ldots, \mathbf{h}_m^{\text{enc}}$:

$$\mathbf{e}_{t,i} = (\mathbf{h}_t^{\text{dec}})^T \mathbf{W}_{\text{attProj}} \mathbf{h}_i^{\text{enc}} \text{ where } \mathbf{e}_t \in \mathbb{R}^{m\times 1}, \mathbf{W}_{\text{attProj}} \in \mathbb{R}^{h\times 2h} \qquad 1 \le i \le m \tag{7}$$

$$\alpha_t = \text{softmax}(\mathbf{e}_t) \text{ where } \alpha_t \in \mathbb{R}^{m\times 1} \tag{8}$$

$$\mathbf{a}_t = \sum_{i=1}^{m} \alpha_{t,i} \mathbf{h}_i^{\text{enc}} \text{ where } \mathbf{a}_t \in \mathbb{R}^{2h\times 1} \tag{9}$$

Here, $\mathbf{e}_{t,i}$ is a scalar, the i^{th} element of $\mathbf{e}_t \in \mathbb{R}^{m\times 1}$, computed using the hidden state of the decoder at the t^{th} step $\mathbf{h}_t^{\text{dec}} \in \mathbb{R}^{h\times 1}$, the attention projection $\mathbf{W}_{\text{attProj}} \in \mathbb{R}^{h\times 2h}$, and the hidden state of the encoder at the i^{th} step $\mathbf{h}_i^{\text{enc}} \in \mathbb{R}^{2h\times 1}$.

We now concatenate the attention output $\mathbf{a}_t$ with the decoder hidden state $\mathbf{h}_t^{\text{dec}}$ and pass this through a linear layer, tanh, and dropout to attain the *combined-output* vector $\mathbf{o}_t$.

$$\mathbf{u}_t = [\mathbf{a}_t; \mathbf{h}_t^{\text{dec}}] \text{ where } \mathbf{u}_t \in \mathbb{R}^{3h\times 1} \tag{10}$$

$$\mathbf{v}_t = \mathbf{W}_u \mathbf{u}_t \text{ where } \mathbf{v}_t \in \mathbb{R}^{h\times 1}, \mathbf{W}_u \in \mathbb{R}^{h\times 3h} \tag{11}$$

$$\mathbf{o}_t = \text{dropout}(\tanh(\mathbf{v}_t)) \text{ where } \mathbf{o}_t \in \mathbb{R}^{h\times 1} \tag{12}$$

Then, we produce a probability distribution $\mathbf{P}_t$ over target subwords at the t^{th} timestep:

$$\mathbf{P}_t = \text{softmax}(\mathbf{W}_{\text{vocab}} \mathbf{o}_t) \text{ where } \mathbf{P}_t \in \mathbb{R}^{V_t\times 1}, \mathbf{W}_{\text{vocab}} \in \mathbb{R}^{V_t\times h} \tag{13}$$

Here, V_t is the size of the target vocabulary. Finally, to train the network we compute the cross entropy loss between $\mathbf{P}_t$ and $\mathbf{g}_t$, where $\mathbf{g}_t$ is the one-hot vector of the target subword at the timestep t:

$$J_t(\theta) = \text{CrossEntropy}(\mathbf{P}_t, \mathbf{g}_t) \tag{14}$$

Here, θ represents all the parameters of the model and $J_t(\theta)$ is the loss on the step t of the decoder. Now that we have described the model, let's try implementing it for Jeju dialect to Korean translation!

3.2 Setting up Virtual Environment

In this part, we will set up a virtual environment for implementing the NMT machine. Please note that the following instructions are based on the gsds server as announced on the ETL board [3]. Run the following commands within the assignment directory (`/q3`) to create the appropriate conda environment. This guarantees that you have all the necessary packages to complete the assignment.

[3] https://myetl.snu.ac.kr/courses/264449/discussion_topics/202254

```
conda create -n a3q3 python=3.12
conda activate a3q3
srun --gres=gpu:1 bash env.sh
```

3.3 Implementation Questions

(a) To ensure the sentences in a given batch are of the same length, we must pad shorter sentences to be the same length after identifying the longest sentence in a batch. Implement the `pad_sents` function in `utils.py`, which returns padded sentences. **(5 pts)**

(b) Implement the code of class `LSTMCell_assignment` in `assignment_code.py`. LSTMCell contains two functions: initialization `__init__()` and forward `forward()`. You can refer to the PyTorch documentations [4] or `GRUCell_assignment` class which is implemented on the skeleton code. **(10 pts)**

(c) Implement the `__init__` function of `NMT` class in `nmt_model.py` to initialize layers for the NMT system. You can run sanity check by executing `python sanity_check.py 1c` **(5 pts)**

(d) Implement the `encode` function in `nmt_model.py`. This function converts the padded source sentences into the tensor $\mathbf{X}$, generates $\mathbf{h}_1^{enc}...\mathbf{h}_m^{enc}$, and computes the initial state $\mathbf{h}_0^{dec}$ and initial cell $\mathbf{c}_0^{dec}$ for the Decoder. You can run sanity check by executing `python sanity_check.py 1d` **(8 pts)**

(e) Implement the `decode` function in `nmt_model.py`. This function constructs $\overline{\mathbf{y}}$ and runs the step function over every timestep for the input. You can run sanity check by executing `python sanity_check.py 1e` **(8 pts)**

(f) Implement the `step` function in `nmt_model.py`. This function applies the Decoder's LSTM cell for a single timestep, computing the encoding of the target subword $\mathbf{h}_t^{\text{dec}}$, the attention scores $\mathbf{e}_t$, attention distribution α_t, the attention output $\mathbf{a}_t$, and finally the combined output $\mathbf{o}_t$. You can run a non-comprehensive sanity check by executing `python sanity_check.py 1f` **(5 pts)**

(g) Let's train the model! execute the following command:

```
sh run.sh vocab
sh run.sh train
```

Check out the model is running on GPU when training. Training takes within one GPU hour. After training your model, execute the following command to test the model:

```
sh run.sh test
```

Write down the execution time and BLEU score. To get a full credit, BLEU score should be larger than 50. **(5 pts) Answer:**

(h) There are a few different methods to generate text from a decoder model such as greedy decoding, beam search, top-k sampling, and top-p sampling. In this code, beam search with a default beam size of 10 is utilized. You can modify the beam size by passing it as an argument in the following way:

```
sh run.sh test <beam-size>
```

Now, perform the decoding with beam size of 1, 3, 5, 10 and 25. Note that beam search with a beam size of 1 is equivalent to greedy decoding. Compare the execution time and performance with different beam sizes. Explain the observed trends as well as the potential reasons for the trends. Discuss distinctions between beam search (beam size greater than 1) and greedy decoding, considering expected and observed differences. **(8 pts) Answer:**

(i) **(BONUS)** Conduct additional experiments using various beam sizes to determine the best one. Record your chosen beam size and justify your decision. Research established guidelines for beam size selection (or any rule-of-thumb value for beam size) and contrast your choice or reasoning with these conventions. **(3 pts) Answer:**

[4] LSTMCell: https://pytorch.org/docs/stable/generated/torch.nn.LSTMCell.html
GRUCell: https://pytorch.org/docs/stable/generated/torch.nn.GRUCell.html