# XCS224N Assignment 4 Neural Machine Translation with RNNs

---

**Due Thrusday, April 1 at 11:59pm PT.**

### Guidelines

1. These questions require thought, but do not require long answers. Please be as concise as possible.
2. If you have a question about this homework, we encourage you to post your question on our Slack channel, at http://xcs224n-scpd.slack.com/
3. Familiarize yourself with the collaboration and honor code policy before starting work.
4. For the coding problems, you may not use any libraries except those defined in the provided started code. In particular, ML-specific libraries such as `scikit-learn` are not permitted.

### Submission Instructions

**Coding Submission:** Some questions in this assignment require a coding response. For these questions, you should submit **all files indicated in the question** to the online student portal. Your code will be autograded online using `src/grader.py`, which is provided for you in the `src/` subdirectory. You can also run this autograder on your local computer, although some of the tests will be skipped (since they require the instructor solution code for comparison).

### Honor code

We strongly encourage students to form study groups. Students may discuss and work on homework problems in groups. However, each student must write down the solutions independently, and without referring to written notes from the joint session. In other words, each student must understand the solution well enough in order to reconstruct it by him/herself. In addition, each student should write on the problem set the set of people with whom s/he collaborated. Further, because we occasionally reuse problem set questions from previous years, we expect students not to copy, refer to, or look at the solutions in preparing their answers. It is an honor code violation to intentionally refer to a previous year's solutions.

### Writing Code and Running the Autograder

All your code should be entered into `src/submission.py`. When editing `src/submission.py`, please only make changes between the lines containing `### START_CODE_HERE ###` and `### END_CODE_HERE ###`. Do not make changes to files other than `src/submission.py`.

The unit tests in `src/grader.py` (the autograder) will be used to verify a correct submission. Run the autograder locally using the following terminal command within the `src/` subdirectory:

```
$ python grader.py
```

There are two types of unit tests used by the autograder:

- `basic`: These unit tests will verify only that your code runs without errors on obvious test cases. These tests so not require the instructor solution code and can therefore be run on your local computer.
- `hidden`: These unit tests will verify that your code produces correct results on complex inputs and tricky corner cases. Since these tests require the instructor solution code to verify results, only the setup and inputs are provided. When you run the autograder locally, these test cases will run, but the results will not be verified by the autograder. When your run the autograder online, these tests will run and you will receive feedback on any errors that might occur.

For debugging purposes, you can run a single unit test locally. For example, you can run the test case `3a-0-basic` using the following terminal command within the `src/` subdirectory:

```
$ python grader.py 3a-0-basic
```

Before beginning this course, please walk through the Anaconda Setup for XCS Courses to familiarize yourself with the coding environment. Use the env defined in `src/environment.yml` to run your code. This is the same environment used by the online autograder.

---

In this assignment you will write code for a Neural Machine Translation (NMT) model using RNNs. The NMT system is more complicated than the neural networks we have previously constructed within this class and takes about **4 hours to train on a GPU**. Thus, we strongly recommend you get started early with this assignment. Finally, the notation and implementation of the NMT system is a bit tricky, so if you ever get stuck along the way, please post a question in the Slack workspace or contact your Course Facilitator

# 1 Neural Machine Translation with RNNs

In Machine Translation, our goal is to convert a sentence from the *source* language (e.g. Spanish) to the *target* language (e.g. English). In this assignment, we will implement a sequence-to-sequence (Seq2Seq) network with attention, to build a Neural Machine Translation (NMT) system. In this section, we describe the **training procedure** for the proposed NMT system, which uses a Bidirectional LSTM Encoder and a Unidirectional LSTM Decoder.

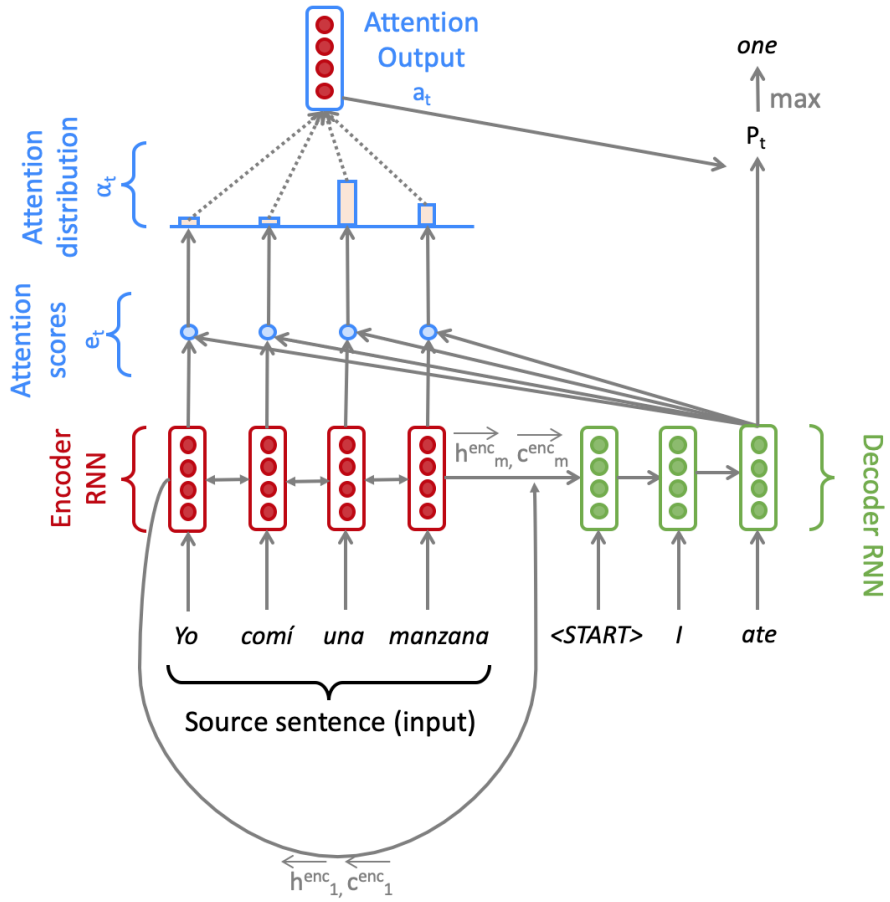


Figure 1: Seq2Seq Model with Multiplicative Attention, shown on the third step of the decoder. Note that for readability, we do not picture the concatenation of the previous combined-output with the decoder input.

## Model description (training procedure)

Given a sentence in the source language, we look up the word embeddings from an embeddings matrix, yielding $\mathbf{x}_1, \ldots, \mathbf{x}_m \mid \mathbf{x}_i \in \mathbb{R}^{e \times 1}$, 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 give hidden states $\mathbf{h}^{\text{enc}}_i$ and cell states $\mathbf{c}^{\text{enc}}_i$:

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

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

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.[1]

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

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

With the Decoder initialized, we must now feed it a matching sentence in the target language. On the $t^{th}$ step, we look up the embedding for the $t^{th}$ word, $\mathbf{y}_t \in \mathbb{R}^{e\times 1}$. 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 word (i.e. the start token) $\mathbf{o}_0$ is a zero-vector. We then feed $\overline{\mathbf{y}_t}$ as input to the Decoder LSTM.

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

$$\tag{6}$$

We then use $\mathbf{h}_t^{\text{dec}}$ to compute multiplicative attention over $\mathbf{h}_0^{\text{enc}}, \ldots, \mathbf{h}_m^{\text{enc}}$:

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

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

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

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 $\mathbf{o}_t$.

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

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

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

Then, we produce a probability distribution $\mathbf{P}_t$ over target words 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 then compute the softmax cross entropy loss between $\mathbf{P}_t$ and $\mathbf{g}_t$, where $\mathbf{g}_t$ is the 1-hot vector of the target word at timestep $t$:

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

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

## Setting up your Virtual Machine

Follow the instructions in the XCS224N Azure Guide in order to create your VM instance. Though you will need the GPU to train your model, we strongly advise that you first develop the code locally and ensure that it runs, before attempting to train it on your VM. GPU time is expensive and limited. It takes approximately **4 hours** to train the NMT system. We don't want you to accidentally use all your GPU time for the assignment, debugging your model rather than training and evaluating it. Finally, **make sure that your VM is turned off whenever you are not using it.**

In order to run the model code on your VM, please run the following command to create the proper virtual environment (You did this at the beginning of the course on your local computer):

[1] If it's not obvious, think about why we regard $[\overleftarrow{\mathbf{h}_1^{\text{enc}}}, \overrightarrow{\mathbf{h}_m^{\text{enc}}}]$ as the 'final hidden state' of the Encoder.

```
$ conda env create --file environment.yml
```

Next, you need to install GPU-specific dependencies on your VM. First, activate the `XCS224N` environment you just created. Then install the dependencies:

```
$ conda activate XCS224N
(XCS224N)$ conda install --file gpu_requirements.txt
```

For local development and testing, you can feel free to continue to using the same `XCS224N` environment you've used for all the assignments so far.

## Implementation Assignment

(a) **[2 points (Coding)]** In order to apply tensor operations, we must ensure that the sentences in a given batch are of the same length. Thus, we must identify the longest sentence in a batch and pad others to be the same length. Implement the `pad_sents` function in `submission/utils.py`, which shall produce these padded sentences.

(b) **[3 points (Coding)]** Implement the `__init__` function in `submission/model_embeddings.py` to initialize the necessary source and target embeddings.

(c) **[4 points (Coding)]** Implement the `__init__` function in `submission/nmt_model.py` to initialize the necessary layers (LSTM, projection, and dropout) for the NMT system.

(d) **[9 points (Coding)]** Implement the `encode` function in `submission/nmt_model.py`. This function converts the padded source sentences into the tensor $\mathbf{X}$, generates $\mathbf{h}_1^{\text{enc}}, \ldots, \mathbf{h}_m^{\text{enc}}$, and computes the initial state $\mathbf{h}_0^{\text{dec}}$ and initial cell $\mathbf{c}_0^{\text{dec}}$ for the Decoder.

(e) **[9 points (Coding)]** Implement the `decode` function in `submission/nmt_model.py`. This function constructs $\bar{\mathbf{y}}$ and runs the `step` function over every timestep for the input.

(f) **[11 points (Coding)]** Implement the `step` function in `submission/nmt_model.py`. This function applies the Decoder's LSTM cell for a single timestep, computing the encoding of the target word $\mathbf{h}_t^{\text{dec}}$, the attention scores $\mathbf{e}_t$, attention distribution $\alpha_t$, the attention output $\mathbf{a}_t$, and finally the combined output $\mathbf{o}_t$.

Now it's time to get things running! Execute the following to generate the necessary vocab file (you can do this on your local computer):

```
(XCS224N)$ sh run.sh vocab
```

As noted earlier, we recommend that you develop the code on your personal computer. Confirm that you are running in the proper conda environment and then execute the following command to train the model on your local machine:

```
(XCS224N)$ sh run.sh train_local
```

Once you have ensured that your code does not crash (i.e. let it run until `iter 10` or `iter 20`), power on your VM from the Azure Web Portal. Then read the *Practical Guide for Using the VM* section of the XCS224N Azure Guide for instructions on how to upload your code to the VM. Next, turn to the *Managing Processes on a VM* section of the Practical Guide and follow the instructions to create a new tmux session. Concretely, run the following command to create tmux session called `nmt`.

```
(XCS224N)$ tmux new -s nmt
```

Once your VM is configured and you are in a tmux session, reactivate your `XCS224N` environment and execute:

```
$ conda activate XCS224N
(XCS224N)$ sh run.sh train
```

Once you know your code is running properly, you can detach from session and close your ssh connection to the server. To detach from the session, run:

```
(XCS224N)$ tmux detach
```

You can return to your training model by ssh-ing back into the server and attaching to the tmux session by running:

```
(XCS224N)$ tmux a -t nmt
```

(g) **[3 points (Coding)]** Once your model is done training (**this should take about 4 hours on the VM**), execute the following command to test the model:

```
(XCS224N)$ sh run.sh test
```

To achieve credit for this portion of the assignment (i.e., training a large NMT model using a GPU), you must use your trained model to translate a Spanish test set into English. Your results will be compared to the correct translation and a BLEU score of 21 will be required to achieve full credit. To generate the gradescope test data, execute the following (local computer or VM):

```
(XCS224N)$ python evaluation_output.py
```

### Deliverables

For this assignment, please submit all files within the `src/submission` subdirectory. This includes:

- `src/submission/__init__.py`
- `src/submission/model_embeddings.py`
- `src/submission/nmt_model.py`
- `src/submission/utils.py`
- `src/submission/gradescope_test_outputs_(soln).txt`

This handout includes space for every question that requires a written response. Please feel free to use it to handwrite your solutions (legibly, please). If you choose to typeset your solutions, the `README.md` for this assignment includes instructions to regenerate this handout with your typeset LaTeX solutions.

---

# THERE IS NO WRITTEN SUBMISSION FOR THIS ASSIGNMENT.

# YOU ARE NOT REQUIRED TO SUBMIT ANYTHING.