Credit: This homework is based on Stanford cs224n materials.

Honor code notice: By turning in this assignment, you agree to abide by the UMass Lowell rules on Academic Integrity and declare that this is your own work. You also agree not to share your solution outside this class.

Homework due date: April 6 2020 (code), April 8 2020 (written)

Starter code url: `http://web.stanford.edu/class/cs224n/assignments/a4.zip`

Hardware note

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 CS department machine or (especially) GCP/AWS/Azure virtual machine. It takes approximately **4 hours** to train the NMT system on GPU (about 40 hours on CPU). So **start early**, because even if you finish writing the code 1 hour before the deadline, you won't have time to train it.

We recommend using GPU workstations provided by the CS department. You can find the guide on our GitHub [1]. Please check that you are able to connect to them and run code on GPU (`torch.cuda.is_available()` should return True). We found out that some machines (e.g., dan417-01) do not have a properly installed CUDA. Just don't use them - find one that works (e.g., dan417-02). If you can't or don't want to use them, we provided a bunch of alternative options at the end of this document.

As you can see, this homework is not notebook-based. There is a good reason for this. When your experiments and models become complex enough, using one notebook is hard, because you need modularized structure in your code in order to keep up with the code complexity. This homework is close in its complexity to a small class project, and you can base your project on it. Seq2seq is a pretty general text generation set up, and you can build a dialogue generation/summarization model with a few modifications.

If you have any questions, feel free to ask them in the Google Group or Zoom the class TAs during their office hours. Good luck!

[1] `https://github.com/text-machine-lab/uml_nlp_class_2020/blob/master/homeworks/homework5/README.md`

Introduction

This assignment is split into two parts: *Neural Machine Translation with RNNs* and *Analyzing NMT Systems*. The first is primarily coding, and implementation focused, whereas the second entirely consists of written, analysis questions. If you get stuck on the first section, you can always work on the second as the two sections are independent of each other. Note that 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 that 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 contact TAs and meet with them during their office hours so that they can support you.

We discussed the topics required for this homework in class. If you struggle with them, ask the TAs during their office hours or send an email to our Google Group. You are also encouraged to use Stanford lecture 8 on attention and machine translation, especially if you're stuck with the notation: `youtu.be/XXtpJxZBa2c`

Part 1: Neural Machine Translation with RNNs (45 points)

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.

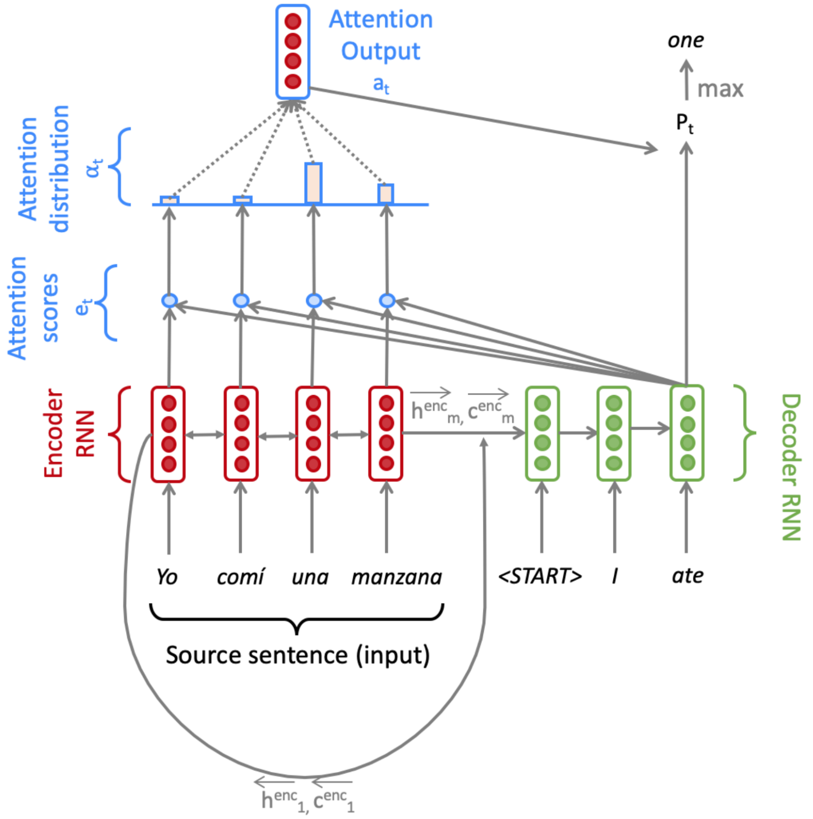

Figure 1: Seq2Seq Model with Multiplicative Attention, shown on the third step of the decoder.

0.1 Model description (training procedure)

Given a sentence in the source language, we look up the word embeddings from an embeddings matrix, yielding $\mathbf{x}_1, \ldots, \mathbf{x}_m \left(\mathbf{x}_i \in \mathbb{R}^{e \times 1}\right)$, 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 (→) and backwards (←) LSTMs. The forwards and backwards versions are concatenated to give hidden states $\mathbf{h}_i^{\text{enc}}$ and cell states $\mathbf{c}_i^{\text{enc}}$:

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

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

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].

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

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

$$\tag{5}$$

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} 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.

$$\mathbf{h}_t^{\text{dec}}, \mathbf{c}_t^{\text{dec}} = \text{Decoder}(\overline{\mathbf{y}_t}, \mathbf{h}_{t-1}^{\text{dec}}, \mathbf{c}_{t-1}^{\text{dec}}) \text{ where } \mathbf{h}_t^{\text{dec}} \in \mathbb{R}^{h \times 1}, \mathbf{c}_t^{\text{dec}} \in \mathbb{R}^{h \times 1} \tag{6}$$

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

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

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

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

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 $\mathbf{o}_t$.

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

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

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

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

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

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 one-hot vector of the target word at 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 step t of the decoder. Now that we have described the model let's try implementing it for Spanish to English translation!

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

0.2 Setting up the environment

We recommend making a new virtual environment [3] for this project.

Create a virtual environment `virtualenv -p python3 venv`

Activate the virtual envrironment

```
source venv/bin/activate
```

Install required dependencies

```
python -m pip install torch==1.3 tqdm docopt nltk
```

If you work on a GPU machine, check that it works (this code should print True).

```
python -c "import torch; print(torch.cuda.is_available())"
```

Note that you need to activate your virtual environment (`source venv/bin/activate`) every time you ssh to the remote machine or create a new terminal window/tab.

0.3 Implementation and written questions

1. (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 `utils.py`, which shall produce these padded sentences.

2. (3 points) (coding) Implement the `__init__` function in `model_embeddings.py` to initialize the necessary source and target embeddings.

3. (4 points) (coding) Implement the `__init__` function in `nmt_model.py` to initialize the necessary model embeddings (using the `ModelEmbeddings` class from model embeddings.py) and layers (LSTM, projection, and dropout) for the NMT system.

4. (8 points) (coding) Implement the encode function in `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. You can run a non-comprehensive sanity check by executing:

   ```
   python sanity_check.py 1d
   ```

5. (8 points) (coding) 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 a non-comprehensive sanity check by executing:

   ```
   python sanity_check.py 1e
   ```

6. (10 points) (coding) 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 word $\mathbf{h}_t^{\text{dec}}$, 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
   ```

7. (3 points) (written) The `generate_sent_masks()` function in `nmt_model.py` produces a tensor called enc-masks. It has shape (batch size, max source sentence length) and contains 1s in positions corresponding to 'pad' tokens in the input, and 0s for non-pad tokens. Look at how the masks are used during the attention computation in the `step()` function (lines 295-296).

   First, explain (in around three sentences) what effect the masks have on the entire attention computation. Then explain (in one or two sentences) why it is necessary to use the masks in this way.

Now it's time to get things running! Execute the following to generate the necessary vocab file:

```
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 environment and then execute the following command to train the model on your local machine:

[3] `https://docs.python-guide.org/dev/virtualenvs`

```
        sh run.sh train_local
```

Once you have ensured that your code does not crash (i.e., let it run till iter 10 or iter 20), power on your VM / Colab / etc, upload your code. Your machine probably already has all the requirements, but if it's not, run

```
        pip install -r gpu_requirements.txt
```

If you're using ssh to connect to your virtual machine or CS department GPU station, execute [4].

```
        tmux new -s nmt
```

If you are using Colab, just don't close its browser tab.

Finally, launch training using

```
        sh run.sh train
```

If you're using ssh, detach from tmux session

```
        tmux detach
```

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

```
        tmux a -t nmt
```

1. (4 points) Once your model is done training (this should take about 4 hours on GPU), execute the following command to test the model:

   ```
   sh run.sh test
   ```

   Please report the model's corpus BLEU Score. It should be larger than 21.

2. (3 points) (written) In class, we learned about dot product attention, multiplicative attention, and additive attention. Please explain one advantage and one disadvantage of dot product attention compared to multiplicative attention. Then explain one advantage and one disadvantage of additive attention compared to multiplicative attention. As a reminder, dot product attention is $\mathbf{e}_{t,i} = \mathbf{s}_t^T \mathbf{h}_i$ multiplicative attention is $\mathbf{e}_{t,i} = \mathbf{s}_t^T \mathbf{W} \mathbf{h}_i$, and additive attention is $\mathbf{e}_{t,i} = \mathbf{v}^T \tanh\left(\mathbf{W}_1 \mathbf{h}_i + \mathbf{W}_2 \mathbf{s}_t\right)$

Part 2: Analyzing NMT Systems (30 points)

Analyzing your data and your model outputs is a huge part of real-world ML applications. It allows you to check that it actually works. Do not rely solely on automated metrics - sometimes, you just missed a bug in metric computation, or the metric itself may be non-descriptive.

1. (12 points) Here, we present a series of errors we found in the outputs 2 of our NMT model (which is the same as the one you just trained). For each example of a Spanish source sentence, reference (i.e., 'gold') English translation, and NMT (i.e., 'model') English translation, please:

   (a) Identify the error in the NMT translation.

   (b) Provide the possible reason(s) why the model may have made the error (either due to a specific linguistic construct or a specific model limitation).

   (c) Describe one possible way we might alter the NMT system to fix the observed error. There are more than one possible fixes for an error. For example, it could be tweaking the size of the hidden layers or changing the attention mechanism.

   Below are the translations that you should analyze, as described above. Note that out-of-vocabulary words are underlined. Rest assured that you don't need to know Spanish to answer these questions. You just need to know English! The Spanish words in these questions are similar enough to English that you can mostly see the alignments. If you are uncertain about some words, please feel free to use resources like Google Translate to look them up.

[4] If you are not familiar with tmux, we recommend this guide `https://www.hamvocke.com/blog/a-quick-and-easy-guide-to-tmux/`

(a) (2 points) **Source Sentence**: Aquí de mis favoritos, "La noche estrellada".
Reference Translation: So another one of my favorites, "The Starry Night".
NMT Translation: Here's another favorite of my favorites, "The Starry Night".

(b) (2 points) **Source Sentence**: Ustedes saben que lo que yo hago es escribir para los niños, y, de hecho, probablemente soy el autor para niños, ms ledo en los EEUU.
Reference Translation: You know, what I do is write for children, and I'm probably America's most widely read children's author, in fact.
NMT Translation: You know what I do is write for children, and in fact, I'm probably the author for children, more reading in the U.S.

(c) (2 points) **Source Sentence**: Un amigo me hizo eso - Richard Bolingbroke.
Reference Translation: friend of mine did that - Richard Bolingbroke.
NMT Translation: friend of mine did that - Richard ⟨ unk ⟩

(d) (2 points) **Source Sentence**: Solo tienes que dar vuelta a la manzana para verlo como una epifanía.
Reference Translation: You've just got to go around the block to see it as an epiphany.
NMT Translation: You just have to go back to the apple to see it as an epiphany.

(e) (2 points) **Source Sentence**: Ella salvó mi vida al permitirme entrar al baño de la sala de profesores.
Reference Translation: She saved my life by letting me go to the bathroom in the teachers' lounge.
NMT Translation: She saved my life by letting me go to the bathroom in the women's room.

(f) (2 points) **Source Sentence**: Eso es más de 100,000 hectáreas.
Reference Translation: That's more than 250 thousand acres.
NMT Translation: That's over 100,000 acres.

2. (4 points) Now, it is time to explore the outputs of the model that you have trained! The test-set translations your model produced in question 1-i should be located in outputs/test outputs.txt. Please identify 2 examples of errors that your model produced. [5]. The two examples you find should be different error types from one another and different error types than the examples provided in the previous question. For each example, you should:

   (a) Write the source sentence in Spanish. The source sentences are in the `en_es_data/test.es`.

   (b) Write the reference English translation. The reference translations are in the `en_es_data/test.en`.

   (c) Write your NMT model's English translation. The model-translated sentences are in the `outputs/test_outputs.txt`.

   (d) Identify the error in the NMT translation.

   (e) Provide a reason why the model may have made the error (either due to a specific linguistic construct or specific model limitations).

   (f) Describe one possible way we might alter the NMT system to fix the observed error.

3. (14 points) BLEU score is the most commonly used automatic evaluation metric for NMT systems. It is usually calculated across the entire test set, but here we will consider BLEU defined for a single example. [6] Suppose we have a source sentence s, a set of k reference translations $\mathbf{r}_1, \dots, \mathbf{r}_k$, and a candidate translation c. To compute the BLEU score of c, we first compute the modified n-gram precision p_n of $\mathbf{c}$, for each of $n = 1, 2, 3, 4$, where n is the n in n-gram: Next, we compute the brevity

[5] An 'error' is not just an NMT translation that doesn't match the reference translation. There must be something wrong with the NMT translation, in your opinion.

[6] This definition of sentence-level BLEU score matches the `sentence_bleu()` function in the `nltk` Python package. Note that the NLTK function is sensitive to capitalization. In this question, all text is lowercased, so capitalization is irrelevant. `http://www.nltk.org/api/nltk.translate.html#nltk.translate.bleu_score.sentence_bleu`

penalty BP. Let len(c) be the length of **c** and let len(r) be the length of the reference translation that is closest to len (c) (in the case of two equally-close reference translation lengths, choose len(r) as the shorter one).

$$BP = \begin{cases} 1 & \text{if } \operatorname{len}(c) \geq \operatorname{len}(r) \\ \exp\left(1 - \frac{\operatorname{len}(r)}{\operatorname{len}(c)}\right) & \text{otherwise} \end{cases} \tag{15}$$

Lastly, the BLEU score for candidate c with respect to $\mathbf{r}_1, \ldots, \mathbf{r}_k$ is:

$$BLEU = BP \times \exp\left(\sum_{n=1}^{4} \lambda_n \log p_n\right) \tag{16}$$

where $\lambda_1, \lambda_2, \lambda_3, \lambda_4$ are weights that sum to 1. The log here is natural log.

(a) (5 points) Please consider this example: Source Sentence **s**: el amor todo lo puede
Reference Translation **r1**: love can always find a way Reference Translation **r2**: love makes anything possible NMT Translation **c1**: the love can always do NMT Translation **c2**: love can make anything possible

Please compute the BLEU scores for $\mathbf{c}_1$ and $\mathbf{c}_2$. Let $\lambda_i = 0.5$ for $i \in \{1, 2\}$ and $\lambda_i = 0$ for $i \in \{3, 4\}$ (this means we ignore 3-grams and 4-grams, i.e., don't compute p_3 or p_4). When computing BLEU scores, show your working (i.e., show your computed values for $p_1, p_2, \operatorname{len}(c)$ $\operatorname{len}(r)$ and BP). Note that the BLEU scores can be expressed between 0 and 1 or between 0 and 100. The code is using the 0 to 100 scale while in this question we are using the 0 to 1 scale.

Which of the two NMT translations is considered the better translation according to the BLEU Score? Do you agree that it is a better translation?

(b) (5 points) Our hard drive was corrupted, and we lost Reference Translation **r2**. Please recompute BLEU scores for **c1** and **c2**, this time with respect to **r1** only. Which of the two NMT translations now receives the higher BLEU score? Do you agree that it is a better translation?

(c) (2 points) Due to data availability, NMT systems are often evaluated with respect to only a single reference translation. Please explain (in a few sentences) why this may be problematic.

(d) (2 points) List two advantages and two disadvantages of BLEU, compared to human evaluation, as an evaluation metric for Machine Translation.

Submission Instructions

You shall submit this assignment as two submissions – one coding and one written.

1. Run the `collect_submission.sh` script on a remote machine / notebook to produce your `assignment5.zip` file. You can use `scp` to transfer files between virtual machines and your local computer.

2. Upload your `assignment5.zip` file to CS server and execute
`submit arum hw5code assignment5.zip`

3. Upload your written submittion as a PDF file to CS server and execute
`submit arum hw5written <your_pdf_filename>`

Additional Readings

1. Original BLEU paper: "BLEU: a Method for Automatic Evaluation of Machine Translation", Papineni et al., 2002, `www.aclweb.org/anthology/P02-1040.pdf`

2. Original seq2seq paper: "Sequence to Sequence Learning with Neural Networks", Sutskever et al., 2014, `arxiv.org/abs/1409.3215`

3. Original attention paper: “Neural Machine Translation by Jointl

4. Blog post: “The Annotated Encoder Decoder. A PyTorch tutorial implementing Bahdanau et al. (2015)”, `bastings.github.io/annotated_encoder_decoder`

   y Learning to Align and Translate”, Bahdanau et al., `arxiv.org/abs/1409.0473`

5. Dot product attention paper (and just a better explanation of attention): “Effective Approaches to Attention-based Neural Machine Translation”, Luong et al., 2015, `arxiv.org/abs/1508.04025`

6. Blog post: “The Annotated Encoder Decoder. A PyTorch tutorial implementing Bahdanau et al. (2015)”, `bastings.github.io/annotated_encoder_decoder`

Credit: This homework is based on Stanford cs224n materials.

Alternative GPU options

Otherwise, you can either use **free GPU** on Google Colab [7], Kaggle Kernel [8], Paperspace Gradient [9] or similar resources. To run your python script from a notebook, follow this tutorial [10]. It should be similar for all providers, but if you have problems, you can search for a specific tutorial (e.g. ”how to run a python script using kaggle kernel”). The main problem of this approach is that you don’t have any computation guarantee - your training can be interrupted. However, this is a rare problem - just do not close your browser tab while it is training, and everything will be ok.

The third option, which is the most advanced one is to use cloud services. Main cloud providers are Amazon Web Services, Microsoft Azure, and Google Cloud Platform. All of them have free welcome credits ($100 - $300), and some of them can provide additional free credits for students. Setup may be complicated, and most of them require a credit card. You won’t be charged until you used all or your free credits, but be careful and make sure that your virtual machine is turned off whenever you are not using it. GPU time is very expensive (˜$2-10 / hour), so you can run out of free credits if you forget to turn off your VM in a few days. This is a good tutorial on how to use Google Cloud Platform: `https://github.com/cs231n/gcloud`. Note that you need to request GPU quota, and it may take up to 2 business days, so do this asap. Paperspace [11] also provides free GPU tier VMs and may be easier to use. You can also find tutorials on how to use free credits on Azure or AWS for deep learning.

Learning how cloud services work may be painful, and we won’t be able to help with all of your problems. On the bright side, knowing how to work with cloud services is a useful skill, and it will also help you with your final project because their GPUs are significantly faster than Colab GPUs.

The last option is to use your local GPU / CPU. We do not recommend using the CPU for this homework. If you do, please start it at least a week before the deadline because you need $\tilde{2}$ days to train your network, and you may need to iterate on this. For example, if there is a bug you missed during the debug phase or if the hyper-parameters are bad. It is deep learning, stuff like this happens every day. If you have a GPU on your computer, make sure to install CUDA-enabled PyTorch and check this using torch.cuda.is_available() at least a week before the deadline. You need an Nvidia GPU to support CUDA.

[7] `https://colab.research.google.com`

[8] `https://www.kaggle.com/kernels`

[9] `https://gradient.paperspace.com/notebooks`

[10] `https://towardsdatascience.com/google-colab-59ad8510eb7e`

[11] `www.paperspace.com`