

Report for final project:

Machine - Translation

Members:

Moran Oshia - 313292633

Alex Chagan - 206262123

Raam Banin - 204702344

Moderator:

Or Haim Anidjar

Purpose

In this document we are going to describe how our final project in the subject of “Machine Translation” is going to work at the course “Deep Learning Methods for Natural Language Processing & Speech Recognition”, a little bit on the algorithm of seq2seq, preparations of the data sets, how we used OOP principle, how to train the models and the user interface and how to use it.

Our main goal is to program a user-friendly GUI web application to translate a word or sentences with the seq2seq model after training.

Short explanation about seq2seq

The most common sequence-to-sequence (seq2seq) models are encoder-decoder models, which commonly use a recurrent neural network (RNN) to encode the source (input) sentence into a single vector.

We can think of the context vector as being an abstract representation of the entire input sentence. This vector is then decoded by a second RNN which learns to output the target (output) sentence by generating it one word at a time.

How does it work?

The idea being that you get one model encoder to take a sequence of words and turn them into an encoder vector which represents all the information in the sentence you want to translate. Then you get a decoder that takes this vector and outputs the translated words using a softmax function.

Attention Layers

The attention layer allows the decoder to focus on the parts of the sequence needed for getting the output correct at that time step, therefore shortening the path between the input word to its translation, thereby alleviating some of the memory limitations that LSTMs can have. It does this by providing a way of scoring tokens in the target sequence against all the tokens on the source sequence and using this to change the input to the decoder sequence. The scores are then fed into a softmax activation to make an attention distribution.

Object Oriented

The training and testing process was divided into 3 main classes: Pre Process, Training and Inference. The advantage of having a class for each stage is that we can save the results inside pickle objects and this way we don't need to repeat the process each time we want to train a new model or test an existing model.

Pre Process

The pre-process class gets a text file of sentences in a certain language and their translation in another language as input. It takes the input and creates a pre-process object that contains dictionaries for both languages and pairs of sentences and their translations. In order to make the data more efficient, the sentences are trimmed by removing certain symbols and filtering them by a max length parameter. It saves the pre-process object as a pickle.

Training

The training class gets a pre-process object in the form of a pickle as input.

At the beginning of the process It has 2 options:

1. If we want to create a new training model from scratch, it initializes an encoder and a decoder.
2. If we want to train an existing training model, we load a pickle that contains an updated encoder and decoder.

Next, we randomly pick pairs based on the number of iterations, and for each pair we run the input sentence through the encoder, and keep track of every output and the latest hidden state. Then the decoder is given the SOS token as its first input, and the last hidden state of the encoder as its first hidden state.

It creates a training object with updated encoder and decoder and saves it inside a pickle.

Inference

The inference class gets a training object in the form of a pickle as input. We simply feed the decoder's predictions back to itself for each step. Every time it predicts a word we add it to the output string, and if it predicts the EOS token we stop there.

Additionally we have a python file that contains global variables that are used throughout the entire process. Because we want the project to be dynamic with every 2 languages, the only changes that we need to implement are the parameters inside this file.

Levenshtein Algorithm

In order to deal with spelling mistakes inside the user's input sentence, we implemented an algorithm that iterates over all words in the sentence and turns unrecognized words into the "closest" words from the dictionary based on the levenshtein distance.

Requirements

- Python 3.8
- Ubuntu
- torch== 1.9.0
- numpy==1.21.1
- levenshtein~=0.12.0
- Flask~=2.0.0
- matplotlib~=3.4.2

You can run the command in the terminal

```
$ pip install -r requirements.txt
```

and it will install the necessary packages.

Preparations of the data sets:

We got the data set of English - French that contains above hundred thousand sentences.

We create the data set of Hebrew Aramic from the website of [Bible and Mishneh Torah for All - Jews and Gentiles / Mechon Mamre](#) , and arrange the verses to short sentences of hebrew Aramic separated by tab. This data set contains about ten thousand sentences.

Creating the model step by step

The model is created with the help of Google's Colab notebook:

<https://research.google.com/colaboratory>

1. Connect to GPU:

Runtime -> change runtime type -> GPU -> save.

2. clone Github. activation of the folder by name Machine_Translation:

```
!git clone https://github.com/MoranOshia/Machine_Translation
%cd Machine_Translation
```

3. Install requirements: Installs versions of torch and numpy:

```
!pip install -r requirements.txt
```

4. Pre-process

global vars:

lang1 and lang2 choosing languages for the translation.

reverse switches between lang1 to lang2

example: lang1='eng', lang2='fra', reverse=True

The translation will be from French to english.

Activating Python code: create dictionary pickle

```
!python /content/Machine_Translation/pre_process.py
```

5. Training

global vars:

Dictionary name: pickle to dictionary. Write the name of the pickle we want.

example: "eng-fra-dictionary.pickle"

Activating Python code: create model pickle

```
!python /content/Machine_Translation/training.py
```

6. Inference

global vars:

Model name: pickle to model. Write the name of the pickle we want.

example: "eng-fra-model.pickle"

Activating Python code: test the training model

```
!python /content/Machine_Translation/inference.py
```

User Interface

We decided to use the Flask for our Front-End in order to connect between the HTML and the Python, by using the Flask we can run the device on localhost.

Install and activate Flask:

1. At first we installed the Flask package by using the following command :

```
$ pip install Flask
```

(Now it is in the requirement.txt included)

2. To run the application, use the flask command or python -m flask. Before you can do that you need to tell your terminal the application to work with by exporting the FLASK_APP environment variable, by using the following command :

```
$ export FLASK_APP=flask_MT.py
```

```
$ flask run
```

3. To enable all DEBUG features, set the FLASK_DEBUG environment variable to 1, by using the following command:

```
$ export FLASK_DEBUG=1
```

Reference Material:

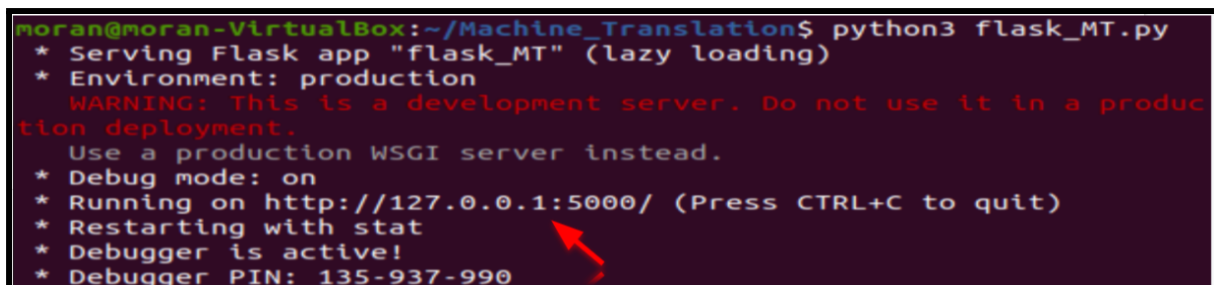
<https://flask.palletsprojects.com/en/2.0.x/installation/#python-version>

<https://flask.palletsprojects.com/en/2.0.x/quickstart/#a-minimal-application>

Running the server in localhost:

Now, you can simply use the following command in order to run the server in localhost
http://127.0.0.1 in port 5000:

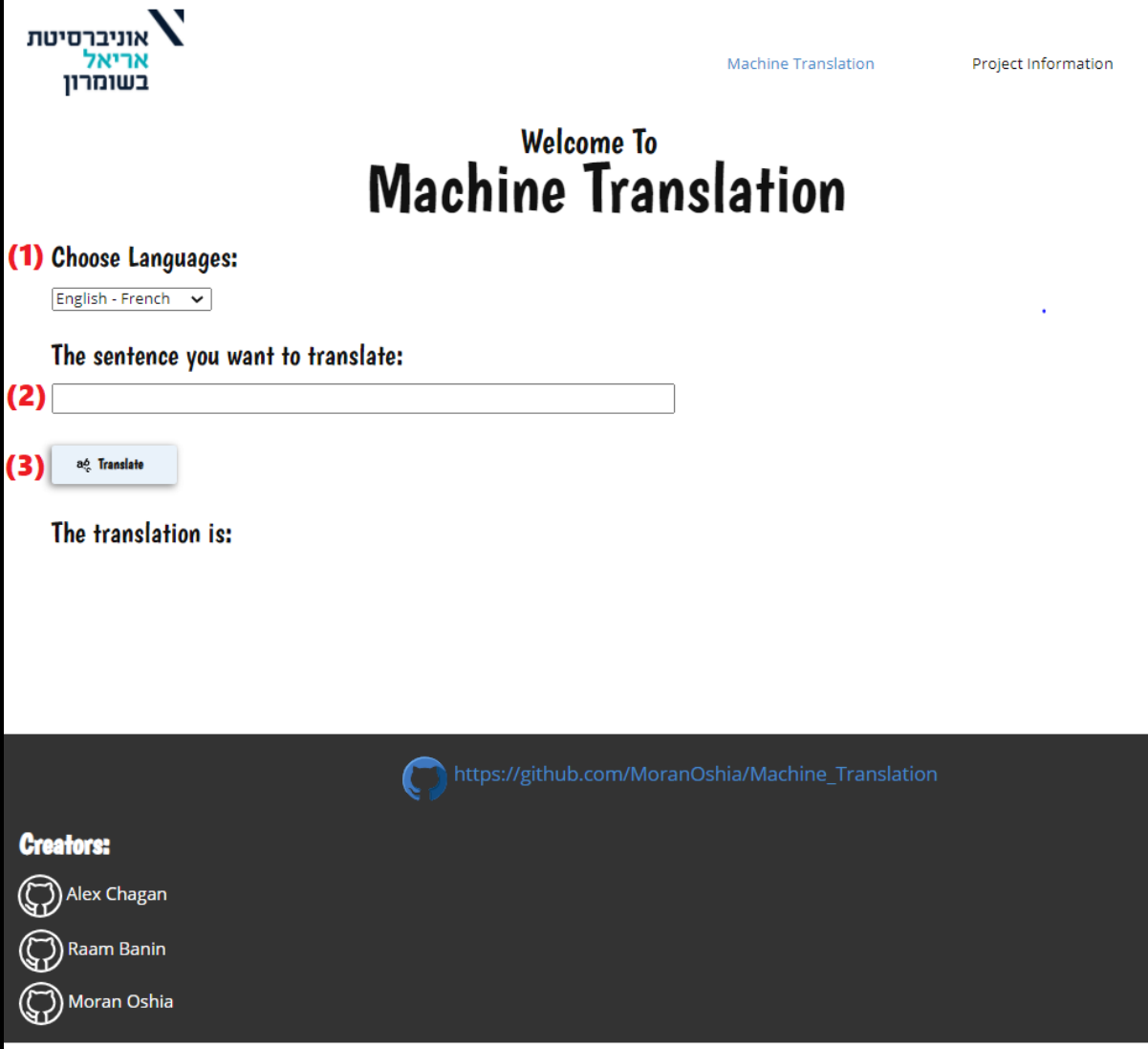
```
$ python flask_MT.py
```

A terminal window screenshot with a dark background and light-colored text. The prompt is 'moran@moran-VirtualBox:~/Machine_Translation\$'. The command 'python3 flask_MT.py' has been executed. The output shows: '* Serving Flask app "flask_MT" (lazy loading)', '* Environment: production', a red 'WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.', '* Debug mode: on', '* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)', '* Restarting with stat', '* Debugger is active!', and '* Debugger PIN: 135-937-990'. A red arrow points to the URL 'http://127.0.0.1:5000/'.

```
moran@moran-VirtualBox:~/Machine_Translation$ python3 flask_MT.py
* Serving Flask app "flask_MT" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production
deployment.
  Use a production WSGI server instead.
* Debug mode: on
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 135-937-990
```

How to use the web application:

After running the localhost and entering the URL <http://127.0.0.1:5000> in browser, it will open the web in the main page:




The screenshot shows a web application interface for machine translation. At the top left is a logo for 'אוניברסיטת אריאל בשומרון' (Ariel University of Samaria). At the top right are links for 'Machine Translation' and 'Project Information'. The main heading is 'Welcome To Machine Translation'. Below this, there are three numbered steps: (1) 'Choose Languages:' with a dropdown menu showing 'English - French'; (2) 'The sentence you want to translate:' with a text input field; and (3) a 'Translate' button. Below the button, it says 'The translation is:'. At the bottom, there is a dark footer section with a GitHub logo and the URL 'https://github.com/MoranOshia/Machine_Translation', followed by a 'Creators:' section listing three people: Alex Chagan, Raam Banin, and Moran Oshia, each with a GitHub profile icon.

In order to translate a word or a sentence you will need:

- Choose the languages you want to translate from to other language in the drop down list (1).
- Enter the sentence from the origin language you want to translate (2).
- Press on the button Translate (3).

Example of translation:



The screenshot shows the initial state of the 'Machine Translation' web application. The header includes the logo of the Hebrew University of Jerusalem (אוניברסיטת אריאל בשומרון) on the left, and navigation links for 'Machine Translation' and 'Project Information' on the right. The main heading is 'Welcome To Machine Translation'. Below this, there is a section titled 'Choose Languages:' with a dropdown menu currently set to 'Hebrew - Aramaic'. Underneath, the prompt 'The sentence you want to translate:' is followed by a text input field containing the Hebrew text 'ויאמר משה'. A blue button with a right-pointing arrow and the text 'Translate' is positioned below the input field. At the bottom, the text 'The translation is:' is displayed, followed by a large empty space for the output.

After clicking on “Translate” button:



The screenshot shows the application after the 'Translate' button was clicked. The language dropdown menu has been changed to 'English - French'. The text input field is now empty. The 'Translate' button remains visible. The output section, labeled 'The translation is:', now displays the Hebrew text 'ואמר משה.' (And he said, Mשה.) in the center of the page.

GitHub Link:

https://github.com/MoranOshia/Machine_Translation