Get Inspired - an Inspirobot replication

Course project for Machine learning for statistical NLP: Advanced LT2326

Minerva Suvanto

2.11.2021

1 Introduction

The course project aims to replicate Inspirobot, an AI generating inspirational and motivational quotes for entertainment. The particular topic for the project was chosen to reach personal goals of practising text generation further and gaining knowledge on Long Short Term Memory (LSTM). LSTMs are a powerful method for various natural language processing tasks and with a fairly large dataset, I hoped to achieve at least sensible, if not motivational, quotes. Since the methods of Inspirobot are not public, I relied on knowledge about natural language generation in general.

The task of language generation aims to produce language which is understandable by humans. It does so by learning patterns from the data that the model is trained on. A language generation model can be tweaked to generate a specific type of text - be it the next book by George R. R. Martin or an epic hit song by your favorite artist. The output depends on the type of data the model is trained on. With this in mind, I searched for a large amount of motivational quotes to work with.

Using LSTMs for various natural language processing tasks has been very successful in the past years. The reason that they produce successful results for tasks such as text generation is that they are able to go back in time to look at relevant information. (Olah, 2015) The encouraging information of the benefits of using LSTMs is why I decided to explore their usage for the project. However, other popular methods for text generation are Variational Autoencoders and Generative Adversarial Networks, which are methods worth considering for such a task.

In this report I describe the process of completing the project. I start by introducing the dataset and explaining how I preprocessed it. I go over the model architecture and the training process. In the final sections I show some samples of the results from the trained model and analyse the results. I also consider some factors on why the model performs the way that it does and a few ideas on how the model could be developed further.

2 Data resource

For the dataset I used a large dataset of quotes created by Goel et al. from 2018. The dataset includes nearly five hundred thousand quotes from various websites. The dataset includes among the quotes the author of the quote and some descriptive category tags. The tags could be useful for generating motivational quotes with an attribute, for example to create a motivational quote about specifically friendship. However, this was not in the scope of my project, but rather an interesting possibility to extend the model in the future.

I downloaded the data as a csv file to use for my training. I process the data in the data.py file. After extracting the data from the file into a list and lower casing everything, I tokenize it using the

nltk-packages functions for sentence tokenization and then word tokenization. As the names suggest, the sentence tokenizer tokenizes a quote into sentences. This is done because a quote can have more than one sentence and we need to separate it in order to do word tokenization. Once a quote is split into sentences, each sentence is tokenized to words. In word tokenization for example contractions are considered as individual words, such as the word “I’m” will become “i” and “‘m”. At this point the data might be considered to be processed further, such as removing punctuation and filler words or lemmatizing words.

Once the data is tokenized it is then turned into n-grams. For the value of n I chose 10 to give the model more information to learn on. The n-grams are paired with their target values so that the loss can be calculated during training.

In the final step the data is converted from string values to integer values. This is done by creating a list of unique words and a dictionary of words with an index value from the list of unique words. The dictionary is used to convert string values to integer values.

The final output after the data processing returns the training data, the length of the vocabulary, the data maps for converting a word to an integer and vice versa, and the full tokenized data. The first two values are used for training the model and the remaining values are used in the generation and testing parts.

The dataset is truncated to contain 50 000 quotes. This is done because the whole dataset would take several hours to train just on one epoch, even with a higher batch size. I believe the truncated dataset should still give some insightful results for the sake of this project.

All of the data is used for training and a separate test set or evaluation set is not instantiated, because it was not necessary for the task. The reasoning for this decision is explained in chapter 4.

The data was downloaded from the following source: https://github.com/ShivaliGoel/Quotes-500K

3 Implementation

3.1 Model

For the text generation task I chose to use an LSTM model. The resulting model architecture is fairly simple and I have explained the structure in detail in this chapter. The code for the model is in the model.py file.

The first layer is the embedding layer which converts the input tensors into embeddings. The value for input size (*num_embeddings*) is the length of the vocabulary and the output embedding size (*embedding_dim*) is 200. Various data sources stated that embedding size is typically between 100 and 300, since longer and shorter vectors do not provide sufficient representations (wacax, 2018). Therefore I chose an embedding size in the middle of the range, 200. I also found options of calculating the embedding dimensions based on the vocabulary, but for simplicity I decided to stick to a fixed value.

The next part is the key component which is the LSTM layer. The input to the layer (*input_size*) is the embedding size. For the number of features in the hidden state (*hidden_size*) I chose 256. This value was arbitrary and experimental based on examples I saw online. In the training data, the batches are the first dimension. To avoid having to reshape my data I set the LSTM to recognise this (*batch_first = True*). I change the dropout value to introduce a dropout layer next. Since the dropout layer adds dropout after all but the last recurrent layer, the number of layers (*num_layers*) should be more than 1, so I change it to 2.

The dropout layer is used to prevent overfitting of the data. It ignores randomly selected elements in the input tensor. A good default value for it is to use 20%, which is what I chose to use in my model. (Eckhardt, 2018)

To get the final result, the output is passed through a fully connected linear layer. The input size (*in_features*) is the hidden state size from the LSTM layer. The output size (out_features) is the length of the vocabulary.

3.2 Training

Once the model was completed it was time to train it. The code for the training is in the train.py file. The very first thing I did was get batches of the training data using the *DataLoader* class from *torch.utils.data package*. The *DataLoader* creates batches of size 128 and shuffles the data.

I experimented with the batch size for some time, since I was training the model on the full dataset at first. The training loop was running for several hours and had not reached the second epoch when I decided to start tweaking the batch size. I attempted to make the training loop faster by increasing the batch size, first from 4 to 32, then to 64, and finally to 128. Increasing the batch size did not speed up the training sufficiently so eventually I decided to truncate the data set, like mentioned in the first chapter. After doing so I managed to train a model in about an hour.

In the training loop I use Adam optimizer and cross-entropy loss as the loss function. The learning rate for Adam is 0.01. The model is trained for 3 epochs. In the training loop I use softmax to get

probabilities of the next word and then compute the loss. The output of the model is permuted to get a shape compatible with the loss function. After this the gradients are computer, parameters updated and the gradients are reset.

After the training has finished the model is saved so it can be loaded for later use. The model is found on mtlgpu in */scratch/gussuvmi/lt2326-project/model.pt*.

4 Results

For the evaluation and testing of the model I contemplated between different possibilities. Different ways of testing text generation models are human-centric evaluation metrics, automatic metrics that require no training and machine-learned metrics. Human-centric evaluation metrics are considered the most valuable metrics for testing language generation models. (Celikyilmaz et al., 2021) Since I could generate multiple quotes easily, I decided to evaluate the meaningfulness of the quotes myself. Furthermore I chose to calculate a cosine similarity between a generated quote and a real quote to see how automatic metrics would evaluate the model and if the metrics matched my own judgements.

As mentioned in chapter 2, the dataset is not split into a separate test set. This is because the text generation is done by initialising a random seed extracted from the training data and evaluation is done by computing cosine similarities between the generated quote and the original quote that the seed was extracted from. Therefore I did not find it necessary to get a separate test set. Generating a separate test set would have also introduced a problem with some words missing from the vocabulary that the model was trained on. This would have required some form of a solution, such as removing out-of-vocabulary words from the test set.

The code for generating quotes and testing can be found in the test.py file. To generate a quote, a random seed is generated. This is done by selecting a sentence from the data at random. Then a seed length is generated at random between the range of 1 and 5. I decided to use a fairly short seed length because with a shorter seed, the output from the model would be mostly predictions, and therefore give more insight on how the model performs. The seed is converted to integers and the original quote is returned to use for measuring cosine similarity later.

To generate a quote, a random quote length is generated between the range of 10 and 20. The dataset contained a lot of long quotes with multiple sentences, so I tried to match the length to those. For an actual inspirational quote, such as those produced by Inpirobot, I would argue that the length should be shorter, but as mentioned before, I wanted to get a larger quantity of produced text. The seed is passed to the model and the last word of the output is appended to the final quote until the quote has reached

the given length. The quote converted to words is returned along with the latest output of the model, which is used for calculating the cosine similarity.

The cosine similarity is computed between the original quote and and the output of the model. The seed is removed from the vector to get a more realistic result. If the vector lengths do not match, the longer vector is truncated. Another option would be to add padding to the shorter vector, but this distorts the results too much, so I decided to go with truncating. The cosine similarity is computed by getting the dot product of the vectors which is divided by the product of the lengths of the vectors. The result is a percentage between 1 and 0, meaning the closer the similarity is to, 1 the more similar the two vectors are.

Some sample results are in the table below.

Generated quote	**Original quote**	**cos.sim.**
i will find deuteronomy primroses shadows confounded playful emptiness insincerely ممكنة	i will find you.in the farthest corner , i will find you .	0.881
there lives.god надад solmn adobe cielo haycame youjust homosexuals science.and growling over-cautious self-slavery flirt nakata	there is no logical reason to believe in god . there are emotional reasons , certainly , but i can not have faith that nothing is something simply because it would be reassuring . i can no more believe in god than i can believe an invisible monkey lives in my ass ; however , i would believe in both if they could be scientifically proven .	0.798
it kızartmalık solmn loomed shadows cruising mind.translation oops opens over-cautious davanti likes	it is a common sentence that knowledge is power	0.788

we should haycame moonthat soapy -each dell'ufficio relegated сейчас mind.translation rojo pivot himtypingin moonthat loquacious playful end.because	we should be cautiously open to the spiritual and non-rational , and skeptical of the more invisible magical thinking—what we might call “ magical reason ” —pervading secular thought and experience in modern society . science and technology are for most people a new religion , and their orthodoxies are believed with the same fervor .	0.845
try to peterbilt сейчас unwonted worse.her shadows confounded tandaan not.tom haycame mentiras mischievous pomarańczowe over-cautious polak playful mind.translation	try to remember it always , " he said once gogol had reached him , leading him slowly back across the breakwater , to where his mother and sonia stood waiting . `` remember that you and i made this journey together to a place where there was nowhere left to go .	0.775
men 's eyes were haycame ლურსმნით mind.translation mischievous self-slavery not.tom humbly—to mind.translation whispered.magnus cruising ho	men 's eyes were made to look , let them gaze , i will budge for no man 's pleasure .	0.816
nobody wants pomarańczowe blizzard playful not.tom 127 надад mind.translation frieden end.because tener mâine youjust	nobody wants to look like a fool . nine times out of ten , thatreason alone keeps people from allowing themselves to believe .	0.853
art is relegated playful outyour luckier mind.translation linz 1988 infectious-spreaders	art is the overflow of emotion into action .	0.884
oh , walked thaumaturgy jeziku over-cautious disappointment playful shadows ლურსმნით mind.translation frieden fikir grimm.stories	oh , i have a habit of letting myself be lectured on the things i know best . i like to see if they are understood in the same way i understand ; for there are many ways of knowing the same thing	0.776

you can tell agethat davranır minimize frowned paida whorenot ai rivalled remake traceof nell socialization formulations event	you can tell people of the need to struggle , but when the powerless start to see that they really can make a difference , nothing can quench the fire .	0.867

5 Analysis and discussion of results

5. 1 Analysis of results

Originally, I trained my model on a dataset that contains 10 000 quotes. I wanted to try and improve the results of the model by training it with a larger data set. For this reason, the final model is trained with a dataset that contains 50 000 quotes. When training with the dataset that has more data, the cross entropy loss increases: after the first epoch the total loss is 10.74 and after the last epoch it is 10.98. This means that the loss goes up slightly during training, whereas with the smaller data set, the loss goes down: after the first epoch the loss is 9.52 and in the final epoch the loss has gone down to 9.47. Adding more data seems to make the loss go up. I believe this is explained with the learning rate being too large. To fix it the learning rate could be decreased from 0.01 to for example 0.001.

Since I trained the model twice I managed to get results from both models. However, only results from the final model are included in the examples in the report. For both models the cosine similarities between generated and actual quotes are not so bad, ranging from between 0.7 and 0.9. This however does not give a realistic evaluation on whether the model works well when compared to the evaluation of the textual representations myself.

With the model trained with less data, the results do not make a lot of sense to a human. The quotes are grammatically incorrect. Punctuation is in random places, so in retrospect removing punctuation may have been useful.

For the model trained on a larger dataset the results were similar. It seems that the dataset contained quotes in other languages, despite the source stating the set having only quotes in English. Due to the multilingual dataset the generated quotes are a mixture of different languages in one quote. This is of course unwanted and the model has not learned to categorise words to one language. Perhaps this could be solved by increasing the learning rate as mentioned before, and additionally adding more epochs to train the model for longer.

Something that I think works reasonably well is that there is variety in the words and the model does not predict the same words constantly. This means that the model has learned at least something,

though not necessarily what was intended. However, in the examples it can be seen that some words are predicted more often than others, such as *mind.translation* which is in itself incorrectly processed, since it should in fact be three separate tokens. Another word that seems to repeat often is *сейчас* which translates to *now* in Russian. The reason why these words are generated more often seems illogical and I can not think of a reason why they occur randomly in the quotes.

The reason why the automatic metrics suggest the model to work is probably because a computer does not recognise for example syntactic structures of words. This was an important indication of why human evaluation is essential in text generation. If this model was deployed into a real world situation based on the computational metrics only, the results would not have been great.

5.2 Possible improvements

Besides removing punctuation and quotes which are not in English, it could have been useful to add start and end tokens to the data in preprocessing. Currently the quotes generated will just end randomly. With an end token in the data set, the quotes could be generated until an end token is hit. The data is also not processed in the best way possible and requires a lot of cleaning up. For example, many words seem to be missing a space between them.

Other ways to to improve the model would be to implement bidirectionality in the LSTM. Doing so would use context from both the left and right side of the word to help predict the next word. This would result in more accurate results.

The simplest improvement to be made is to train the model with the entire dataset and use more epochs, so the model trains for longer and thus learns to generalize better. Also the learning rate should be adjusted to create a better model.

6 Conclusion

As a learning experience this project has been very beneficial. I have learned much more about especially LSTMs but also about machine learning architecture in general. I was able to troubleshoot problems in the performance of my models and research for solutions on how to make the model perform better, even though. I implemented a dropout layer, which was originally not something I intended to do, but once figuring out the purpose of it decided it would be useful for my task.

As an actual working model the final outcome is not a very working solution. My expectations of the outcome were not met, since I did assume I would get a little bit more comprehensible outputs from the model. The mixed language data was disappointing since it made the final results look even messier. This also confirms the fact that in machine learning, data is the key to any problem. The

improvements discussed in the previous chapter would be a good first step to start to improve the model. Among these changes, more individual research on text generation would be beneficial. From what I have gathered, machine learning is a lot of trial and error so further testing and tweaking and simply just trying out things would be interesting. For the time being, Inspirobot can provide external motivation to work on these improvements.

Overall this project has inspired me to learn more about machine learning and trying out different methods to solve tasks for NLP. There are various interesting challenges that I can not wait to face and use the skills learned to solve.

References

Celikyilmaz et al. (2021, 5). Evaluation of Text Generation: A Survey. https://arxiv.org/pdf/2006.14799.pdf

wacax. (2018, May 2). *The ratio of vocabulary vs embedding length to determine the size of other layers in a neural network doesn't really matter* [Comment on the online forum post *Ratio between embedded vector dimensions and vocabulary size*]. StackExchange. https://datascience.stackexchange.com/questions/31109/ratio-between-embedded-vector-dimensions-and-vocabulary-size

Eckhardt, E. (2018, 11). Choosing the right Hyperparameters for a simple LSTM using Keras. Towards Data Science. https://towardsdatascience.com/choosing-the-right-hyperparameters-for-a-simple-lstm-using-keras-f8e9ed76f046

Olah, C. (2015, 8). Understanding LSTM Networks. Colah's blog. https://colah.github.io/posts/2015-08-Understanding-LSTMs/