

MSc in Data Science

Spring 2021   ITC6010A1 - Natural Language Processing

Instructor: Dr. Lazaros Polymenakos

**CHIT CHAT CHATBOT A Sequence-to-Sequence Model Implementation using Star Wars Conversations**

Eleni Fengou (127409)

Michael Koinakis (247358)

Eleni Ntokou (249253)

Table of Contents

[Abstract 2](#_Toc77548706)

[Introduction 2](#_Toc77548707)

[Background 3](#_Toc77548708)

[Early Implementations 3](#_Toc77548709)

[Recurrent Neural Networks 4](#_Toc77548710)

[The Encoder-Decoder Model 5](#_Toc77548711)

[Sequence To Sequence Model (Seq2Seq) 5](#_Toc77548712)

[Attention Layer 6](#_Toc77548713)

[Decoding and Vocabulary 7](#_Toc77548714)

[Architecture and Development 9](#_Toc77548715)

[Long Short-Term Memory Networks (L.S.T.M.) and Gated Recurrent Unit Networks (G.R.U.) 9](#_Toc77548716)

[Attention Mechanism 12](#_Toc77548717)

[Methods 15](#_Toc77548718)

[Dataset Description 15](#_Toc77548719)

[Data Preprocessing 15](#_Toc77548720)

[Implementation Details 18](#_Toc77548721)

[Configurations 18](#_Toc77548722)

[Training Model 18](#_Toc77548723)

[Results 20](#_Toc77548724)

[Discussion and Future Work 22](#_Toc77548725)

[References 23](#_Toc77548726)

# Abstract

This project aims to build a chit-chat chatbot trained with a sequence-to-sequence model using Star Wars movies and series scripts. Although the hyperparameters of a deep learning neural network algorithm, such as the encoder-decoder, are plenty, it was decided to focus on the attention mechanism and the training of three different attention approaches in order to compare how each one affects not only execution time but also the chatbot’s efficiency in terms of engagingness and making sense. The conclusion is that concatenated attention mechanism might need more time to train, but overall, it produces better results.

# Introduction

With the arrival of the information age, the field of Artificial Intelligence (A.I.) has seen tremendous growth over the past few years in various forms and applications (Bevacqua, 2015). Research and further implementation over ways to program computers to fruitfully process large amounts of raw data and interact with humans are constantly increasing, and conversational modeling is now used in a wide variety of services and for many different scopes. Chatbots, in particular, can be found in a variety of settings, serving many purposes. They are widely known as personal assistants and commonly used as customer support. However, they are also used in marketing to automate interactions with prospect customers, in the financial services industry, offering banking and account services, in e-commerce websites to better engage with the customers, and even for leisure and entertainment purposes. Following the trends and in parallel satisfying our interests, in this project, we will build a chatbot that simulates Star Wars conversations, trained with the movies and series scripts. This chatbot is implemented for entertainment purposes, given some genuine enthusiasm for the Star Wars world, to attract other fans of the Star Wars community who will be able to embrace fun conversations with a bot answering like a Star Wars character.

Lately, the I.T. industry and Developer companies are creating more and more applications, offering chatbot solutions ranging from plug-n-play to fully developed and personalized applications. Though most applications serve primarily as questioning answering systems (Frequently Asked Questions – F.A.Q.), adoption from major companies excites interest and makes the future seem promising.

# Background

## Early Implementations

Chatbots are software programs that create responses based on some input and imitate human conversations, either in text or voice recognition. The first-ever idea of the chatbot was proposed in 1950 as The Turing Test and called the imitation game by Alan Turing. It was a human evaluator that would judge conversations between humans and machines designed to generate conversations. The test result would be an evaluation that would distinguish whether the participant is a human or a machine. Moving on to 1966, the chatbot E.L.I.Z.A. was developed, destined for psychotherapy, designed to match the users' answers with a list of possible scripted responses. Following E.L.I.Z.A.'s purpose, another chatbot psychiatrist was launched in 1972, named Parry, which was designed to interact with patients with paranoid schizophrenia. Another milestone in chatbot history was the creation of Jabberwacky in 1988, which was the first one using A.I. technologies to provide entertainment and companionship interacting with users by talking in a human voice. The revolution of using A.I. to engage conversations between bots and humans, other chatbots were also created, such as A.L.I.C.E in 1995 and SmarterChild in 2001, which gained much popularity. At a later day, in 2006, I.B.M.'s Watson was launched with new revolutionary technology, being able to process unstructured data and improve the tasks of various industries ranging from healthcare to finance and retail. Later on, in 2010, intelligent assistants, like Siri, Google Now, Cortana and Alexa made their appearance to help automate everyday human tasks and provide information. Soon after that, bots for messengers were designed, and users await the next significant innovation, which will ease their everyday lives.

Chatbots are divided into two main categories based on their architecture. First, the Rule-Based Chatbots are mainly like a decision tree, not using any natural language processing techniques. They appear to be more economical but offer limited services as they can only respond to predefined questions based on keywords extracted from user queries. Thus, it is up to the programming work how smart they will be and can only advance in capabilities when fed with more statements by the programmer. On the other hand, there is Artificial Intelligence (A.I.) Chatbots. Using Machine Learning and N.L.P. techniques can process natural language and interact more humanly. It is noticeable that A.I. chatbots are continuing their training while interacting with the users, and thus, they are evolving and are becoming more intelligent over time. This conversational model usually activates when it is engaging in "chit chat" with the user and being asked off-topic questions that do not target to extract specific information from it.

## Recurrent Neural Networks

The basic concept that differentiates rule-based and neural network-based approaches is the presence of a learning algorithm in the latter case.

A Recurrent Neural Network (R.N.N.) is a natural generalization of feed-forward neural networks to sequences. Given a sequence of inputs (x1,...,xT), a standard R.N.N. computes a sequence of outputs (y1,…,yT) by iterating the following equation (vsinghal, 2020):

The R.N.N. can easily map sequences to sequences whenever the alignment between the inputs and the outputs is known ahead of time (Zaytar, 2016). However, it is unclear how to apply an R.N.N. to problems whose input and the output have different lengths with complicated and non-monotonic relationships (Sutskever, 2014).

Diagram, schematic

Description automatically generated

Figure 1: A recurrent neural network and the unfolding time of the computation involved in its forward computation. Source: LeCun, Bengio, and G. Hinton 2015

## The Encoder-Decoder Model

R.N.N. Encoder-Decoder consists of R.N.N.s that act as an encoder and a decoder pair. The encoder maps a variable-length source sequence to a fixed vector, and the decoder maps the vector representation back to a variable-length target sequence. The two networks are trained jointly to maximize the conditional probability of the target sequence, given a source sequence (Cho, 2014).

## Sequence To Sequence Model (Seq2Seq)

Sequence to Sequence model turned out to be the most popular model for Dialogue Systems and Machine Translation. It consists of two R.N.N.s, an Encoder, and a Decoder described briefly in 2.4.

Each hidden state influences the next hidden state, and the final hidden state is seen as the summary of sequences. This state is called the context or thought vector, as it represents the intention of the sequence. From the context, the decoder generates another sequence, one symbol (word – token) at a time. Here, at each step, the decoder is influenced by the context and the previously generated symbols (Ram, 2016).

Diagram

Description automatically generated

Figure 2: Gated Recurrent Unit (G.R.U.) Encoder-Decoder Seq2Seq Architecture

## Attention Layer

An attention mechanism has been used to improve neural machine translation by focusing on parts of the source sentence during translation. There are two simple and practical classes of attention mechanism: a global approach or Luong Attention, which always attends to all source words, and a local one or Bahdanau, which only looks at a subset of source words at a time. Common to these two types of models is that at each time step t in the decoding phase, both approaches first take as input the hidden state ht at the top layer of stacking G.R.U (Luong, 2015). The goal is to derive a context ct that captures relevant source-side information to help predict the current target word yt.

Chart, box and whisker chart

Description automatically generated

Figure 3: Luong Global Attention Mechanism and Bahdanau Local Attention Mechanism

## Decoding and Vocabulary

Once the model is trained, it can start translating (or in this case generating a response) a given sentence by finding a match that maximizes the following:

Where θ = (.

Greedy Decoding: In greedy decoding, the conditional dependency path is followed, and the symbol with the highest conditional probability is picked at each node. This is equivalent to picking the best symbol one at a time from left to right in conditional language modeling (Gu, 2017). An example of decoded translation of greedy decoding is where:

Beam Search: Beam search keeps K>1 hypotheses, unlike greedy decoding, which keeps only a single hypothesis during decoding. At each time step t, beam search pick K hypotheses with the highest scores (. When all the hypotheses terminate, it returns the hypothesis with the highest log probability (Gu, 2017).

Text, letter

Description automatically generated

Figure 4: Greedy Decoding Architecture pseudocode

# Architecture and Development

In this project, the PyTorch version was used for the full development of the Seq2Seq model. Seq2Seq models are the industry go-to for dialogue and translation systems. To upgrade the performance of the default Seq2Seq model, attention mechanism, greedy search, and bidirectional R.N.N. modules are integrated.

The code was developed entirely on the Google Colab platform, using one Nvidia Tesla T4 GPU accelerator. All the files for the project are uploaded in Google Drive to enhance accessibility.

## Long Short-Term Memory Networks (L.S.T.M.) and Gated Recurrent Unit Networks (G.R.U.)

Recurrent Neural Networks suffer from short-term memory. If a sequence is long enough, they will have a hard time carrying information from earlier steps to later ones. So, in an attempt to process a paragraph of text to make predictions, R.N.N.s may leave out important information from the beginning.

During backpropagation, recurrent neural networks suffer from the vanishing gradient problem. Gradients are values used to update a neural network's weight. The vanishing gradient problem is when the gradient shrinks as it back propagates through time. If a gradient value becomes extremely small, it does not contribute too much learning (Phi, 2018).

So, in recurrent neural networks, layers that get a minor gradient update stops learning. Those are usually the earlier layers. So, because these layers do not learn, R.N.N.s can forget what is seen in longer sequences, thus having a short-term memory (Phi, 2018).

Long Short-Term Memory (L.S.T.M.) and Gated Recurrent Unit (G.R.U.) were created to address this problem. They have internal mechanisms called gates that can regulate the flow of information (Phi, 2018).

Diagram

Description automatically generated

Figure 5: L.S.T.M. and G.R.U. Architecture. Source <https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21>

The central concept of these architectures is the cell state and the various gates. The cell state act as a pipeline that transfers relative information down the chain. It may be considered as the memory of the network. In essence, it can carry relevant information throughout the processing of the sequence. As the cell date goes on, information gets added or removed via gates, which decide which information is allowed on the cell state. They contain sigmoid activations, like tanh activations. Although the tanh activation regulates the values through the network by squishing values between -1 and 1, sigmoid activations squish values between 0 and 1. This is helpful to update or forget data. Everything multiplied with 0 becomes 0 and is forgotten; anything multiplied by 1 remains the same value; therefore value stays the same or is kept in the network. The gates are distinguished to forget gate, input gate, cell state, and output gate for L.S.T.M.s, reset gate, and update gate for G.R.U.s.

The forget gate, as the name indicates, chooses which information is kept or thrown away. Information from the previous hidden state and the current input is passed through a sigmoid function. The values are coming out between 0 and 1. Closer to 0 means forget, and closer to 1 means keep (Phi, 2018) (Pra, 2020).

The input gate is used to update the cell state. First, the current input and the previous hidden state pass through a sigmoid function, which decides which values are forgotten or kept. Also, the same information is passed through a tanh function to help regulate the data. Finally, the outputs from the sigmoid function and tanh function are multiplied, and the final output is the initial decision of the sigmoid function, which decides which data are kept (Phi, 2018).

At this point, there is enough information to calculate the cell state. First, it gets pointwise multiplied by the forget vector, possibly dropping even more values. Then, it gets pointwise addition from the input gate, and now the cell state has been updated with the values that the neural network finds relevant (Phi, 2018).

The fourth gate, the output gate, decides what the next hidden state should be. First, the previous hidden state and the current input into a sigmoid function. Then the updated cell state passes through a tanh function. The output of the tanh and sigmoid functions is multiplied to decide which information the hidden state should carry. The final output is the hidden state (Phi, 2018).

On the other hand, G.R.U.s have a similar performance to L.S.T.M.s, but they get rid of the cell state and use the hidden state to transfer information. Also, instead of having three gates, G.R.U.s utilize two gates: update gate and reset gate (Phi, 2018).

The update gate has a similar function with the forget and input gate of L.S.T.M.s. It decides which information is added or thrown away.

The reset gate is another gate in the G.R.U. architecture, used to decide how much pas information to forget.

Since the G.R.U.s have fewer computations to execute, they are speedier than L.S.T.M.s with the same level of performance. There is not a clear answer whether to use L.S.T.M.s or G.R.U.s. However, there are references that a G.R.U. may perform faster and better with fewer data (Phi, 2018) (Zulqarnain, 2019).

## Attention Mechanism

One particular limitation of the seq2seq framework is that the complete information in the input sentence should be encoded into a fixed-length vector. As the length of the sequence grows, a considerable amount of information is lost. The attention mechanism introduced in (Bahdanau, 2015), allows the decoder to look at the input sequence while decoding. The proposed model from Bahdanau proposes extending the encoder-decoder architecture by allowing a model to automatically search for parts of a source sentence relevant to predicting a target word without having to form these parts like a hard segment explicitly.

As seen in Figure 3, during each time step in the decoder, instead of using a fixed context (last hidden state of the encoder), a distinct context vector ci is used for generating the word yi. This context vector ci is the weighted sum of hidden states of the encoder.

Where n is the length of the input sequence, is the hidden state at time step|j.

eij is the alignment model, which is the function of the decoder's previous hidden state si-1 and the jth hidden state of the encoder. The alignment model is parameterized as a feed-forward neural network jointly trained with the rest of the model (Bahdanau, 2015).

A picture containing text, crossword puzzle

Description automatically generated

Figure 6: Attention visualization example of the alignments between source and target sentences. Source: Neural Machine Translation by Jointly Learning to Align and Translate

In 2015 in (Luong, 2015), another proposition for the attention layer was based on the previous application. Two simple and practical classes of the attentional mechanism were proposed: a global approach that always attends to all source words and a local one that only looks at a subset of source words at a time.

Common to these two types of models is that at each time step t in the decoding phase, both approaches first take as input the hidden state ht at the top layer of a stacking L.S.T.M. The goal is to derive a context vector ct that captures relevant source-side information to help predict the current target word yt. While these models differ in how the context vector ct is derived, they share the same subsequent steps.

Specifically, given the target hidden state ht and the source-side context vector ct, the model employs a simple concatenation layer to combine the information from both vectors to produce a hidden attentional state as follows:

The attentional vector is the fed through a softmax layer to produce the predictive distribution as:

The idea of a global attentional model is to consider all the hidden states of the encoder when deriving the context vector ct. In this model type, a variable-length alignment vector at, whose size equals the number of time steps on the source side, is derived by comparing the current target hidden state ht with each source hidden state :

Here, the score is referred to as a content-based function for which there are considered three different alternatives:

In the project, all three attention mechanisms are implemented, with comparison to the results (Luong, 2015).

# Methods

## Dataset Description

The data used for this project's scope concern dialogues among the heroes of the science fiction sage, Star Wars. The original trilogy movies, Episode IV: A New Hope, Episode V: Empire Strikes Back, and Episode VI: Return of the Jedi, were found in .txt format cleaned from Kaggle. (Xavier, 2018) The prequel trilogy, Episode I: The Phantom Menace, Episode II: Attack of the Clones, and Episode III: Revenge of the Sith, were processed through Python in order to be into a suitable format (IMSDB, n.d.). Finally, the last three movies, all episodes of all six seasons of the TV-Series "The Clone Wars" and all ten chapters of the Mandalorian series, were found in subtitle file formats (.srt files) and were processed in Python locally to bring them in the same format (Open Subtitles, n.d.). The dataset contains 47.047 sentences and 296.605 tokens.

## Data Preprocessing

The data were taken from movie scripts and subtitle files (.srt). Since the sequence of the dialogues was unknown or could not be mapped, various dialogue sequences were created, ranging from groups of two to groups of ten, respecting the order of appearance of each dialogue in each film or series episode.

Some text cleaning was applied, by removing the symbols in each sentence, except the dots, question, and exclamation marks. Also, there was a global lowercase rule applied, and contractions were fixed. Finally, a dictionary with token frequencies was created to remove all those words that are not appearing over a certain frequency threshold, and a dictionary with the unique tokens and their respective indexes. At the same time, basic tokens like the start of the string (S.O.S.), end of the string (E.O.S.), and a padding token (P.A.D.) were created. Only the S.O.S. and E.O.S. tokens are added in the strings at this point.

To prepare the data for the models and GPU parallelization capabilities, it is not enough to simply change the words in the sentences with the corresponding indexes and turn the sentences to tensors. The training is needed to be done in mini-batches. Due to the varying sentence length, all the other sentences are filled with P.A.D. tokens after the E.O.S., to match that length.

An input function is used to convert sentences to tensors, ultimately creating a correctly shaped zero-padded tensor. The outcome is a tensor of lengths for each of the sequences in the batch, which is passed to the decoder layer.

An output function with similar performance to the input function is created, but instead of a length tensor, it returns a binary mask tensor and a maximum target sentence length. The binary mask tensor has the same shape as the output target tensor, but every element is a P.A.D. token is 0, and all the others are 1 (Inkawhich, n.d.).

The last function used in pre-processing takes a batch of sequences of dialogues and returns the input and target tensors using the functions mentioned above.

An example of all the output of the previous functions is the following.

Table

Description automatically generated

Figure 7: Tensors example for the model training

## Implementation Details

|  |  |
| --- | --- |
| Algorithm | Deep Neural Network, Recurrent Neural Network |
| Main Technique | Sequence to Sequence modeling, with encoder-decoder |
| Enhancements | Bidirectional Gated Recurrent Unit (G.R.U.), Embedding Layer, Attention Mechanism |

Table 1: Main Techniques Summary

## Configurations

The table below has the values of the hyperparameters for the model training.

|  |  |  |  |
| --- | --- | --- | --- |
| Parameters | Configuration 1 | Configuration 2 | Configuration 3 |
| Batch size | 128 | 128 | 128 |
| Hidden size | 500 | 500 | 500 |
| Encoder n layers | 8 | 8 | 8 |
| Decoder n layers | 8 | 8 | 8 |
| Dropout | 0.1 | 0.1 | 0.1 |
| Iterations | 48.000 | 48.000 | 48.000 |
| Clip | 50 | 50 | 50 |
| Teacher Forcing Ratio | 1 | 1 | 1 |
| Learning rate | 0.0001 | 0.0001 | 0.0001 |
| Decoder learning ration | 5 | 5 | 5 |
| Attention model | Dot | General | Concat |

Table 2: Training configuration hyperparameters

## Training Model

After pre-processing, the pairs kept for the model training are 374.925 pairs (groups of question-answer – question - …. With varying sequences from 2 to 10).

The number of words kept or unique tokens are 20.224.

The training process is straightforward. An input batch is passed through the encoder. The decoder inputs are initialized as SOS tokens and the hidden state as the encoder’s final hidden state. Then the batch sequence is fed forward through the decoder one-time step at a time. At this step, one trick is used to aid the convergence. It is called teacher forcing. It is a procedure for training RNNs with output to hidden recurrence. It emerges from the maximum likelihood criterion. During training time model receives ground truth output y(t) as input at time t+1 (Srihari, 2016). At some probability, the current target word is used as the decoder’s following input rather than using the decoder’s current guess. This technique acts as training wheels for the decoder, aiding in more efficient training. However, teacher forcing can lead to model instability during inference, as the decoder may not have a good chance to truly craft output sequences during training. Thus, it is mindful of how the threshold probability is set and not be fooled by fast convergence (Goyal, 2016) (Inkawhich, n.d.).

Since the input deals with batches of padded sequences, not all tensor elements are considered when calculating loss. A loss function is defined to calculate the loss based on the decoder’s output tensor, the target tensor, and a binary mask tensor describing the padding of the target tensor. This loss function calculates the average negative log-likelihood of the elements corresponding to a \*1\* in the mask tensor (Chernodub, 2018) (Inkawhich, n.d.).

After performing backpropagation, the second trick that is implemented is gradient clipping. This is a commonly used technique for countering the “exploding gradient” problem. In essence, clipping or thresholding gradients to a maximum value prevent the gradients from growing exponentially and either overflow (NaN) or overshoot cliffs in the cost function (Pascanu, 2013) (Bajaj, 2021) (Inkawhich, n.d.).

Finally, encoder and decoder model parameters are updated.

The whole process described above is an iterative process looping over each time step calculating hidden states. Alternatively, these modules are run one time-step at a time. In this case, it is necessary to manually loop over the sequences during the training process like it must be done for the ``decoder`` model. As long as the correct conceptual model is maintained, implementing sequential models can be very straightforward.

# Results

Since there are no actual evaluation metrics for chatbots, except human judgment, three different models were trained, one for each attention mechanism, and comparison among their results were made in the same conversation scenario.

The Dot-Product Attention needed less training time - almost 6 hours. The Concat Attention mechanism needed 7.5 hours for training, and finally, Generalized Attention needed almost 7 hours. The baseline for the model is the results of the dot method in the attention layer.

By prompting the bot on the same lines, the following results emerged.

Text, letter

Description automatically generated

Figure 8:Results with Luong Dot method

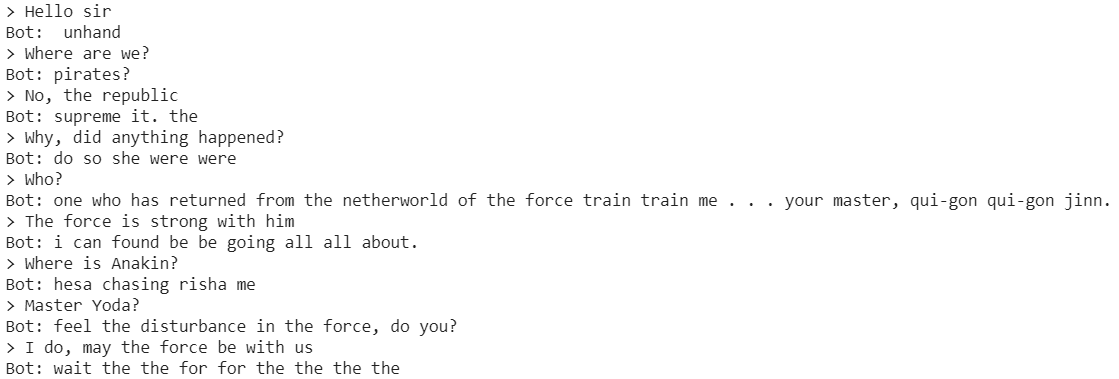


Figure 9:Results with Luong General method

Graphical user interface, text, application, letter, email

Description automatically generated

Figure 10:Results with Luong Concat method

The concat method gives the best results. It provides answers with some meaning, and there are no issues with repeating words or phrases – a common problem with chatbots developed with neural networks and deep learning techniques.

The general method seems to provide the worst results among the three methods since there is evidence of repeated words and phrases. Also, the answers seem to be the least meaningful.

Finally, the benchmark, the dot model, does not give evidence of repeating words or sequences of phrases. Although, the answers provided from the chatbot seem to have less meaningful context than that of the concat method.

All in all, the concat method will provide better results than the other three methods with that amount of data. It will be interesting to check the results with more amount of data.

# Discussion and Future Work

Lastly, some technical remarks about research and implementation. The initial goal was to make a personality chatbot that imitates precisely master’s Yoda personality. This bot would generate responses based on Yoda’s character and allow potential users – Star Wars fans to open conversations with their favorite character asking for his opinion and advice. To achieve this, all conversation sequences with master Yoda lines were extracted from the movies and TV series to train and test the model. By getting the final formatted dataset with Yoda’s lines, it appeared that the number of lines was about 600, a number relatively small to implement this personality. Thus, an alternative of a more generic chatbot was followed, which allows its users to have fun by starting Star Wars related conversations, training our models with all the dialogues of the scripts.

The Star Wars chatbot can be utilized either in a Star Wars-related forum or as a stand-alone application, in which the user can have a pleasant chit-chat. Considering the limited available time and resources (Google Colab has certain usage restrictions), the results are quite satisfactory. However, with further training and a graphical user interface, the user experience would be better and more fun.

When considering another approach for chatbot implementation, this would be using traditional methods, making it Rule-Based. This approach was not attempted due to time restrictions as it would be challenging and time-consuming to make manually crafted rules and keywords.

A future implementation of the current chatbot would mix it up with an information retrieval model or an entity recognition chatbot.

# References

(n.d.). Retrieved from IMSDB: https://imsdb.com/

(n.d.). Retrieved from Open Subtitles: https://www.opensubtitles.org/el/search2/sublanguageid-eng/moviename-star+wars

Bahdanau, D. (2015). Neural Machine Translation By Jointly Learning To Align And Translate. *ICLR.* ICLR.

Bajaj, A. (2021). *Understanding Gradient Clipping (and How It Can Fix Exploding Gradients Problem)*. Retrieved from Neptune Blog: https://neptune.ai/blog/understanding-gradient-clipping-and-how-it-can-fix-exploding-gradients-problem

Bevacqua, P. P. (2015, 5 17). *The PGA of America*. Retrieved from Diversity Journal: https://diversityjournal.com/15242-peter-p-bevacqua-the-pga-of-america/

Chernodub, A. (2018). Sequence Tagging Using Neural Networks. grammarly AI.

Cho, K. (2014). *Learning Phrase Representations using RNN Encoder–Decoder.*

Goyal, A. (2016). *Professor Forcing: A New Algorithm for Training.* Stanford: Stanford University.

Gu, J. (2017). *Trainable Greedy Decoding for Neural Machine Translation.* New York: The University of Hong Kong.

Inkawhich, M. (n.d.). *MatthewInkawhich*. Retrieved from GitHub: https://colab.research.google.com/github/pytorch/tutorials/blob/gh-pages/\_downloads/chatbot\_tutorial.ipynb#scrollTo=PGtXx5GAR7fw

Luong, M.-T. (2015). *Effective Approaches to Attention-based Neural Machine Translation.* Stanford: Stanford University.

Pascanu, R. (2013). *On the difficulty of training Recurrent Neural Networks.*

Phi, M. (2018, 9 24). *Illustrated Guide to LSTM’s and GRU’s: A step by step explanation*. Retrieved from Towards Data Science: https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

Pra, M. D. (2020, 11 2). *Time Series Forecasting with Deep Learning and Attention Mechanism*. Retrieved from Towards Data Science: https://towardsdatascience.com/time-series-forecasting-with-deep-learning-and-attention-mechanism-2d001fc871fc

Ram, S. (2016, 6 28). *Chatbots with Seq2Seq*. Retrieved from GitHub: https://suriyadeepan.github.io/2016-06-28-easy-seq2seq/

Srihari, S. (2016). RNNs: Teacher Forcing. *Deep Learning.* Buffalo: University at Buffalo.

Sutskever, I. (2014). *Sequence to Sequence Learning with Neural Networks.*

vsinghal. (2020, 5 7). *SEQUENCE TO SEQUENCE LEARNING AND ATTENTION IN NEURAL NETWORKS*. Retrieved from CellStrat: https://www.cellstrat.com/2020/05/07/seq-2-seq-attention-neural-networks/

Xavier. (2018). *Star Wars Movie Scripts*. Retrieved from Kaggle: https://www.kaggle.com/xvivancos/star-wars-movie-scripts

Zaytar, M. A. (2016). *Sequence to Sequence Weather Forecasting with Long.* International Journal of Computer Applications.

Zulqarnain, M. (2019). *Efficient Processing of GRU Based on Word Embedding for Text.* INTERNATIONAL JOURNAL ON INFORMATICS VISUALIZATION.