**Advanced Topics in Deep Learning Course Final Project:**

**“Dialogue Script Generation Using Various Methods”**

**By: Dalia Smyrnov & Michael Rubinfeld**

**Lecturer: Dr. Oren Elisha**

**Abstract**

In this project, we have taken on the challenge of generating a TV script that will be coherent, will make sense, and will have a good flow. In order to generate scripts, we used text generators built with various techniques. Some of which are capable of generating a script.

The TV show on which we built the script is “How I Met Your Mother” (From now on will be referenced as HIMYM). The final result will be a script of a dialogue between two characters which never appeared on the show.

**Text Generation**

Natural Language Generation or NLG is a subfield of NLP which is defined as a model which can produce understandable texts from underlying non-linguistic representation of information.

NLG has many industry uses. Such as:

* Chatbot and voice assistants (Siri, Alexa…).
* Summarizing text (such as news reports) or raw numeric data (such as financial reports, or server logs) to easily understandable content for employees.
* Automating personalized communications such as Emails and chat responses.
* creating product descriptions for e-commerce webpages and customer messaging.

**Popular Methods**

There are 3 popular tools used widely for NLG. All of which will be talked about in our project. The tools are: Markov Models (namely Markov Chains), RNNs (namely LSTM and GRU) and Transformers (namely the GPT family, such as GPT-2, GPT-3, GPT-J, and more…) .

**The Data**

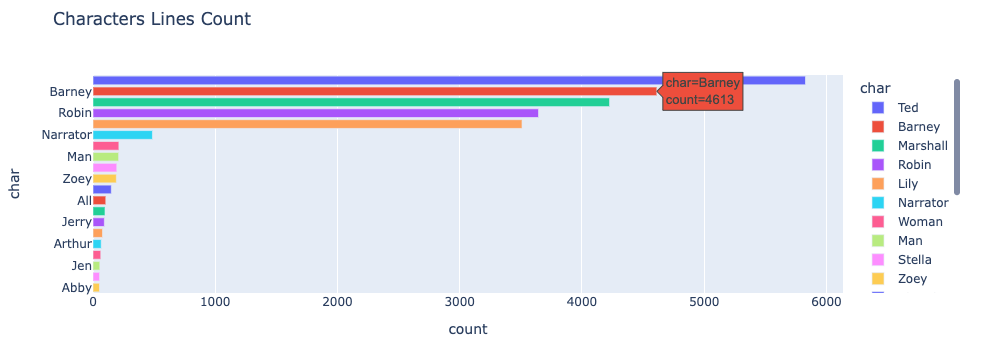
The data used in the project came from a website which is a database for the scripts used for the entirety of HIMYM: <https://transcripts.foreverdreaming.org/viewforum.php?f=177>

The scripts were gathered using a web scraper, powered by Selenium. First we used scrape\_episode\_script\_links.py to create a folder containing a text file for each season, where each file will contain the episode script links. Then we used create\_dataset.py to scrape each script and extract the relevant information for every dialogue line. The output is a .csv file where every line has the text, and the character who said it.

**Preprocessing -** removing director notes and symbols neither numeric or alphabetical.

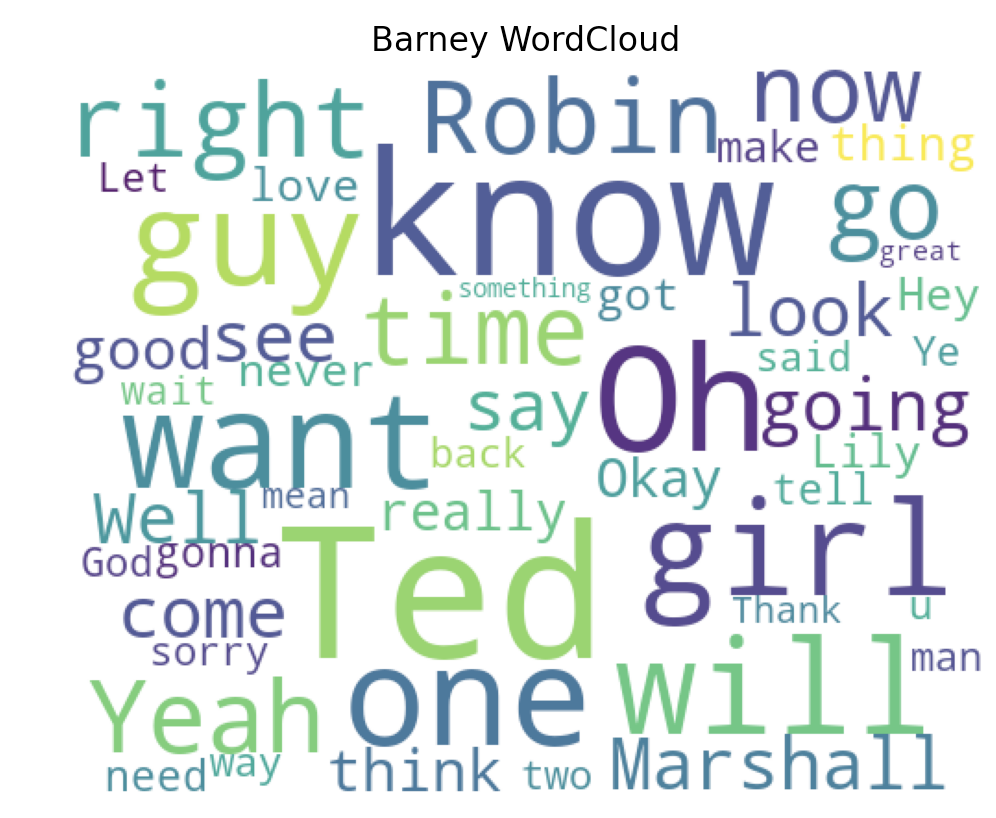
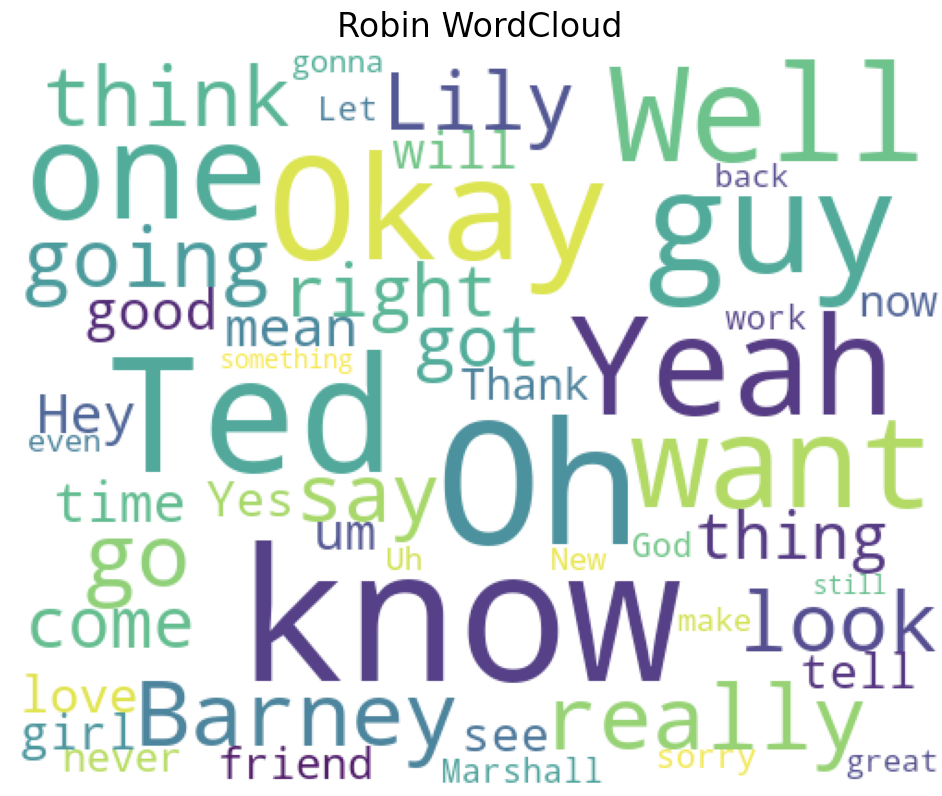
**EDA -**

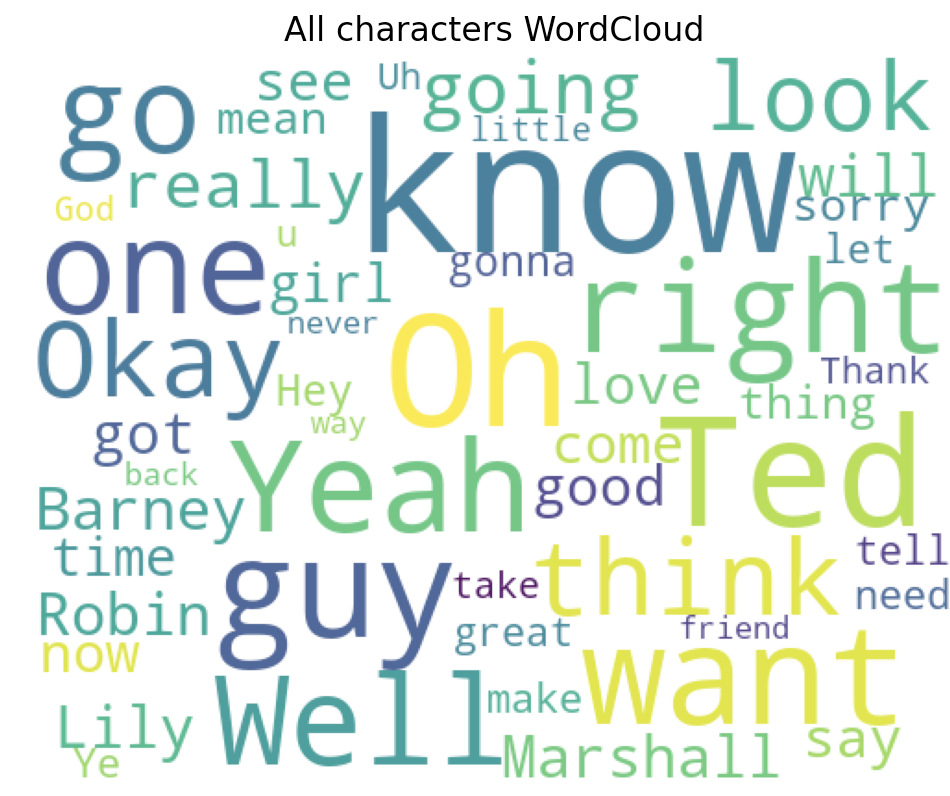
As we can see the five main characters have the most lines. (Narrator is Ted).



**Word Clouds -**

A nice method to see the most common words in each character corpus.

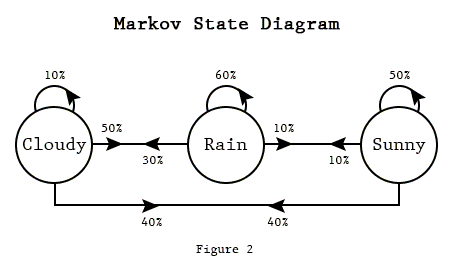




**1st Method - Markov Chains**

Markov Chains is a stochastic process - a mathematical object which is usually used as a model of systems and phenomena that appear to vary in a random manner.

The Markov Chain models a finite set of states, with conditional probabilities to jump from one state to the other.



In order to generate text, we need to define first what our states are going to be. Text is a sequence, so in order to generate the next word we will look at the previous K words as our current state.

**The Naive Method (Our Baseline) -**

The naive method is assuming independence (no conditional probabilities to jump between the states) in the sequence. This means that the current state only needs to be the last word (K=1), and that the next state is the word with the highest probability to come after the word in our previous state.

*Ex: the current state is the word ‘suit’. In the corpus, the word that appears the most times after it is ‘up!’, so the word generated will be ‘up!’ which now will be the current state.*

**Markov Chains for Text Generation -**

Now, we can loosen the assumption of independency. Text is a sequence in which every word is linked not only to the previous word, but to the whole sentence. To try and model that for text generation we will use K > 1.

In order to do that, we will create a vector for each distinct sequence of K words. We’ll create a matrix of all K sequences and all words. Then, we’ll add 1 to the j-th component of the i-th vector, where i is the index of the i-th k-sequence of words, and j is the index of the next word. This will give us a probability distribution for the next word, given the previous k tokens.

**Generating text -**  we start with a seed, which has to be a sequence known to the model. The amount of words in the seed will be K. The model looks at the probability distributions of all the possible next words (all probabilities calculated in advance) and chooses the next word through a weighted choice. This goes on and on until stopping criteria is met.

So, in order to generate text for a character we used all of his/her lines as the corpus.

**Drawback to using Markov Chains -** can take as input only sequences present in the corpus. If the seed contains a word not in the corpus, an exception will be raised.

**2nd Method - RNN**

Similar to what we learned in class. We generate text using a char-level RNN. Similarly to the Markov chain, the model generates the next char using an inference of the previous sequence of chars. This means that we generate text given a context of the previous chars.

The algorithm splits each sample to input text and target text where the input is missing the last char and the target text is missing the first.

The algorithm has three hyperparameters to tune - batch\_size, embedding\_dim and rnn\_units (epochs are constant and we have implemented EarlyStopping, learning rate starts at constant and is automatically reduced at plateaus). We have implemented a pseudo-RandomSearch to find the optimal hyperparameters. The best results were: batch\_size-32, embedding\_dim-512, rnn\_units-1024.

**Generating text -** In order to create the dialogue, we have used two models, one trained on barney’s text and one on Robin’s. First, we loaded each model’s vocabulary and trained weights. Then we’ve generated text in this manner:

1. Barney generates text given the seed, the text is cleaned and saved
2. In a loop, each character generates text given as a seed **all the responses of the adversary character** (an important concept which will be repeated later with transformers)
3. After it is cleaned, each text generated will be saved in the history of the character to be used later by the adversary.

* The cleaning process:

1. Cut the seed from the output
2. The generated text has N chars. This means that the end part of the generated text will have an idea cut short, most of the time in the middle of a word. A solution we came up with was, regardless of the input generation length, cut the text at the last appearance of sentence-ending punctuation: full stop, or question mark. This will only keep generated text which holds a whole idea.
3. Clean noise from generation (random symbols etc…)

**3rd Method - Transformers**

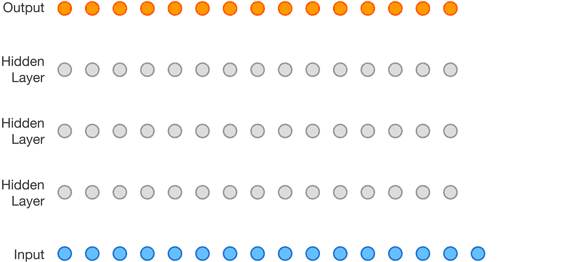
We have generated a dialogue script using DialoGPT. DialoGPT is very similar to GPT-2. It’s a large-scale transformer-based pretrained language model.

**GPT-2 (and how it’s different from BERT)**

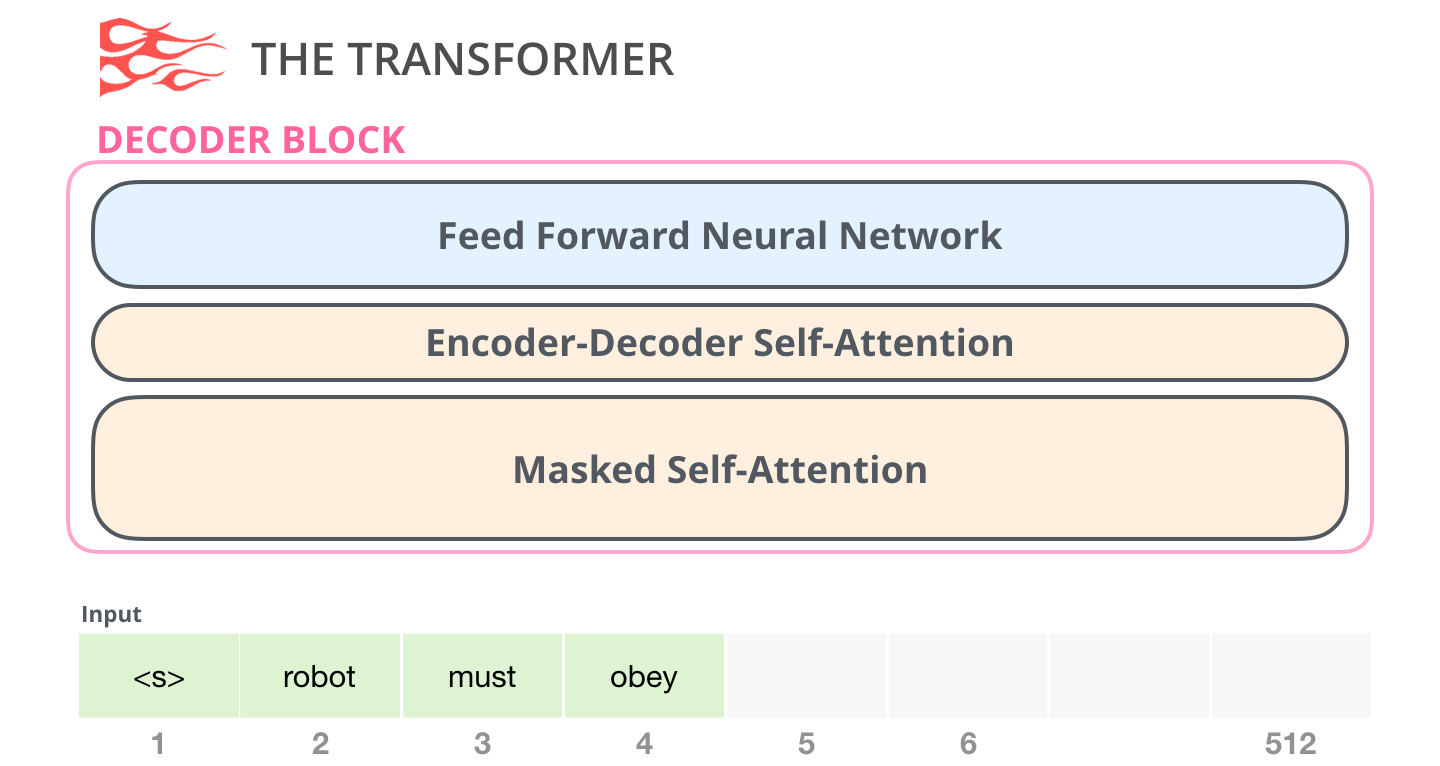
Originally, the transformer model is made up of an encoder and decoder – each is a stack of what we can call transformer blocks. That architecture was appropriate because the model tackled machine translation – a problem where encoder-decoder architectures have been successful. A lot of the subsequent research had shown that the architecture should use either the encoder or the decoder.

**Differences from BERT -** The GPT-2 is built using transformer decoder blocks. BERT, on the other hand, uses transformer encoder blocks. Another key difference is that GPT-2 is an auto-regressive model. BERT is not.

**What is an auto-regressive model -** Similarly to an RNN, an autoregressive model’s output, *h\_t* at time *t*, depends on not just *x\_t* but also *x*’s from previous time steps. Unlike an RNN, the previous *x*’s are not provided from some hidden state: they are given as just another input to the model. In conclusion: in an auto-regressive model, after each token is produced, that token is added to the sequence of inputs. That new sequence becomes the input to the model in its next step, and so on:



**How that difference takes place -**

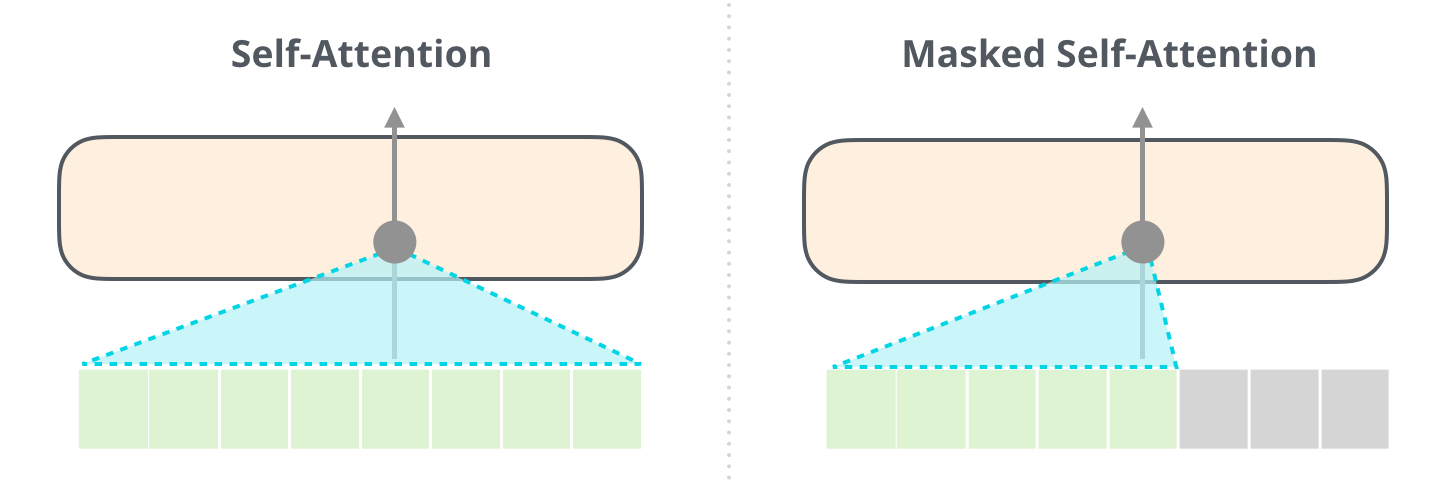


GPT-2's self-attention layer masks future tokens. BERT on the other hand, just changes the words to <mask>, but doesn’t actually mask them (ignores future inputs at each step).



GPT-2's self-attention layer masks future tokens by blocking information from tokens that are to the right of the position being calculated. If, for example, we’re to highlight the path of position i, The masked self-attention layer is only allowed to attend to the present (i) and previous tokens.

In conclusion, the difference between self-attention (what BERT uses) and masked self-attention (what GPT-2 uses) is that a normal self-attention block allows a position to peak at tokens to its right. Masked self-attention prevents that from happening:



**How GPT-2 is trained -** GPT-2 is pretrained on a very large corpus of English data - a massive 40GB dataset called WebText - in a self-supervised fashion. This means it was pretrained on the raw texts only, with no humans labeling them in any way, with an automatic process to generate inputs and labels from those texts.

More precisely, as described before, it was trained to guess the next word in sentences using a mask-mechanism. This method is Similar to how we’ve trained the RNN text generator. For this reason, the RNN text generator gave relatively very good results.

**DialoGPT**

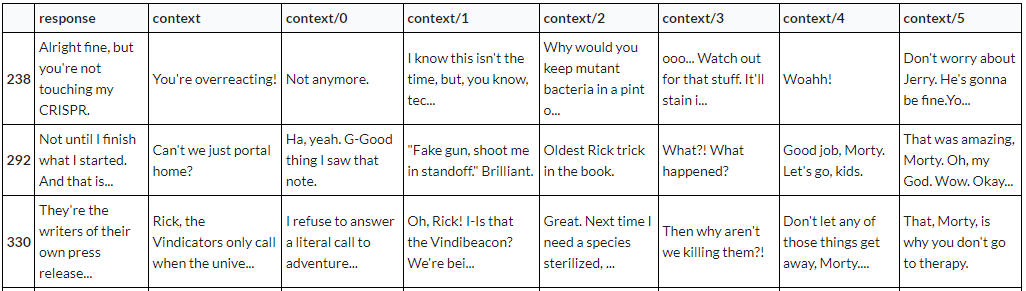
Similarly to GPT-2, DialoGPT is formulated as an auto-regressive language model, and uses a multi-layer transformer as model architecture. Unlike GPT-2, which trains on general text data, DialoGPT draws on 147M multi-turn dialogues extracted from Reddit discussion threads. This is the key difference that makes it more precise in generating responses to dialogue.

DialoGPT produces SOTA response generation. “The [human evaluation” results](https://github.com/dreasysnail/Dialogpt_dev#human-evaluation) indicate that the response generated from DialoGPT is comparable to human response quality under a single-turn conversation Turing test.

**Fine Tuning -** In order to use DialoGPT to generate text, we need to fine tune its weights on data of a character so the text generation will be done by that character. We have fine tuned using Pytorch since it's more compatible with HuggingFace.

The idea we used is to build a dataset of context to feed to the model. Let’s say we want to create a model to generate text as Robin, and we have decided the context length to take into account is 7. We iterate through the data and for every line of the script said by robin: we keep a line in a new dataset.

The target is Robin’s response and the features are the 7 previous lines of text said in the script (the context leading to the response by Robin):



This new dataset is tokenized by the model and is fed to it for fine-tuning.

**Generating text -** Similarly to the RNN, we generate text using two adversary generators, each one generates text given the history of the outputs of the other model. After a small tweak which gave us the ability to generate longer dialogues (mentioned in the notebook), the scripts generated are excellent in our opinion.

\*on the contrary to the RNN model, The transformer didn't need any cleaning of the output.

**Some example scripts made by DialoGPT**

**Seed: What the hell is wrong with you Barney?**

Barney: What do you mean?

Robin: She looks really good.

Barney: All right, let's go.

Robin: What do you mean?

Barney: She's really hot.

Robin: She's not.

Barney: She's great. I'm glad she's here.

Robin: Wait. What?

Barney: I know it's hard to believe, but... I'm really happy for Zoey. I mean, she's had some rough times, but...

Robin: You and me both.

Barney: And me.

Robin: And me.

Barney: Oh, my God.

Robin: Well, I'm glad you're happy.

Barney: Me, too.

**Seed: Why is that girl checking you out?**

Barney: Maybe she's not a girl?

Robin: Let me check my calendar.

Barney: What's that, Barney? It's a calendar.

Robin: What the hell is that?

Barney: It's from last night.

Robin: And you still don't know why the hell she's wearing that?

Barney: Because, b\*\*bs.

Robin: Last night? Oh, my God, that's the worst.

Barney: Well, it's not like it's gonna bother you.

Robin: I don't want it to bother you.

Barney: It's going to bother me. It's going to be like that with everything Robin's done for me.

Robin: You know, I had this really cool theory that I had a secret crush on you. It kind of blew.

Barney: Oh, honey.

Robin: I know.

Barney: And then, one day, I'm walking down the street and seeing a girl walking by with a huge t\*nk on her back.

Robin: It's been a while since I had one of those.

Barney: Maybe we should start dating again.

**Seed: Maclaren's pub tonight??**

Barney: I know right? I mean, it's not like he's gonna be at a strip club with a limp.

Robin: Barney, you think we should go to a strip club with a limp?

Barney: We should go to a strip club with a limp!

Robin: Fine.

Barney: Just the one time we agreed on something, it didn't work out the way we wanted it to.

Robin: It's okay. We'll go back.

Barney: See ya.

Robin: Bye.

Barney: You're such a sweetie.

Robin: Thanks.

Barney: What?

Robin: Oh, I'm sorry. I was so excited to have you. I didn't mean to make you feel bad.

Barney: I didn't mean to make you feel bad. I'm sorry.

Robin: It's okay. I'm not upset about it.

Barney: I'm sorry.

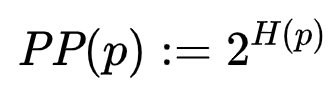
Robin: Don't be. It's okay.

Barney: Yeah. I'm not either.

**Evaluation and Metrics**

In order to evaluate our generators’ performances, we have decided to use perplexity. Only the RNN and transformer can actually generate scripts so we have only evaluated them.

**Perplexity -** a measurement of how well a probability distribution or probability model predicts a sample. It may be used to compare probability models. A low perplexity indicates the probability distribution is good at predicting the sample.

**How we calculated perplexity -** The perplexity PP of a discrete probability distribution p is defined as:  where *H*(*p*) is the entropy (in bits, where bits are defined as the atomic value generated, whether it's a word, or a letter etc…).

Because both the RNN and the transformer measure crossentropy loss, *H*(*p*) can be the loss of the model! So let us define:

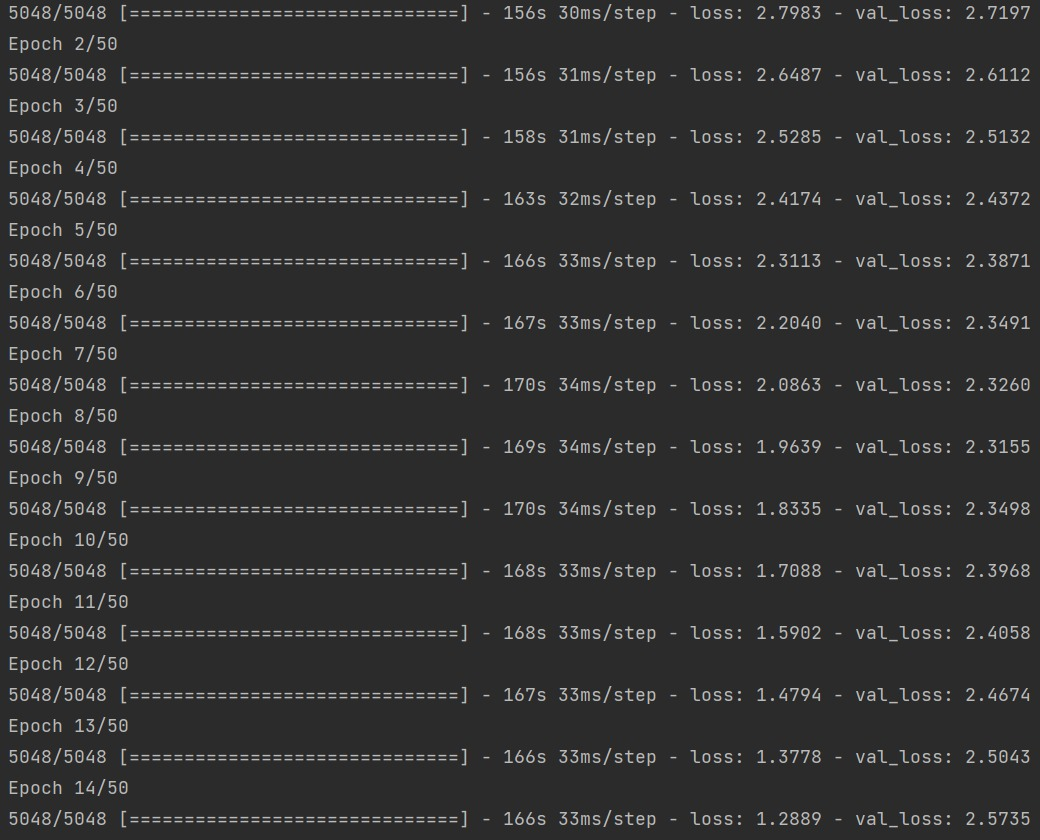
***perplexity = exp(val\_loss)***

* In our case, we have to use e instead of 2 as a base, because TensorFlow and PyTorch measure the cross-entropy loss with the natural logarithm.

**Comparing our models’ results -** As is, we cannot compare the results of the DialoGPT and the RNN, since The GPT is word-level and the RNN is char-level. Since Words are longer than chars, there is a way higher variance in generating words. Due to this fact, the perplexity of char-level models will always be lower and so they’re incomparable to word-level models.

In order to bring them up to par so comparison is possible, we have created a “quick and dirty” version of the RNN that generates at a word-level, with the same rnn\_units and batch\_size as the char-level RNN. Now that both have the same bits of generation, they’re comparable.

Let’s check the training results, and calculate the perplexity based on the best val\_loss we have got (started overfitting at the 9th epoch).



**Final results -**

**RNN’s Perplexity:** exp(2.3155) = 10.13

**DialoGPT’s Perplexity:** 6.3868

**DialoGPT’s (Barney\_Robin) Perplexity:** 4.9319

**Table to compare criteria we think are important for good text generators**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Naive Chain** | **Markov Chain** | **RNN** | **DialoGPT** |
| **Can get a sequence of words as a seed input** |  | **\*** as long as the sequence as a whole is in the corpus |  |  |
| **Can always output a sequence of a sufficient length** |  |  |  |  |
| **Sequence output is understandable and can be easily linked to the character** |  |  |  |  |
| **Seed doesn’t have to appear exactly as-is in the corpus** |  |  |  |  |
| **Output length doesn’t have to be predefined (or does, but we can work around it)** |  |  |  |  |
| **Can be used to generate a dialogue script** |  |  |  |  |
| **Understands context at least to some degree** |  |  |  |  |
| **Always uses punctuation correctly** |  |  |  |  |
| **Always outputs coherent english** |  |  |  |  |
| **Dialogue has somewhat coherent flow** |  |  |  |  |

**Bonus Task - create dialogue between two blends of characters**

Due to the modularity of our code, this task is quite straight-forward: we create a dataset for the DailoGPT where the response column will contain lines from two characters. This will compel the transformer to learn lines of two characters as if they were one. The result of the fine tuning process will be an all new character which is a cool blend of two distinct characters.

**Result:**

**Seed: What are your plans today?**

BarneyRobin: I'm going to eat at Generro.

TedLily: I'm sorry, it's just... I don't feel like cooking right now.

BarneyRobin: Well, you didn't have to.

TedLily: Oh, yeah? Well... I'll see you there.

BarneyRobin: See ya.

TedLily: What?

BarneyRobin: The brunch is here.

TedLily: Yeah, and I have no idea what I'm gonna get.

BarneyRobin: I'll take a burger.

**Conclusions and Summary**

In this project we have looked at various methods of generating text, one stochastic, one using an RNN with a GRU layer and one using a SOTA transformer. It was interesting for us to learn that all 3 methods, while each being **way** more complicated than the last, have the same thought process behind them. Whether its word-level or char-level:

* In the training process, the model needs to guess the next char (or word) in the sequence it trains on (whether its using model weights or just conditional probability)
* In the generation process, after each char (or word) is produced, it’s added to the sequence of inputs, and that new sequence becomes the input to the model in its next step, and so on.

**Sources:**

1. HIMYM Transcripts <https://transcripts.foreverdreaming.org/viewforum.php?f=177>
2. Markov Chains: How to Train Text Generation to Write Like George R. R. Martin - *by: Luciano Strika, published: 29/11/2019*

<https://www.kdnuggets.com/2019/11/markov-chains-train-text-generation.html/>

1. Text generation with an RNN - *by: The Tensorflow team, last updated: 3/5/2022*

[*https://www.tensorflow.org/text/tutorials/text\_generation*](https://www.tensorflow.org/text/tutorials/text_generation)

1. GPT-2 - *by: The HuggingFace team* <https://huggingface.co/gpt2>

# The Illustrated GPT-2 (Visualizing Transformer Language Models) - *by: Jay alammar, published at: 12/8/2019* [htt ps://jalammar.github.io/illustrated-gpt2/](https://jalammar.github.io/illustrated-gpt2/)

## DialoGPT: Toward Human-Quality Conversational Response Generation via Large-Scale Pretraining - *by: Microsoft Research, published: 1/11/2019*[*https://www.microsoft.com/en-us/research/project/large-scale-pretraining-for-response-generation/*](https://www.microsoft.com/en-us/research/project/large-scale-pretraining-for-response-generation/)

1. DIALOGPT : Large-Scale Generative Pre-training for Conversational Response Generation - by: Yizhe Zhang, Siqi Sun, Michel Galley, Yen-Chun Chen, Chris Brockett, Xiang Gao, Jianfeng Gao, Jingjing Liu, and Bill Dolan, last updated: 2/5/2020 <https://arxiv.org/pdf/1911.00536.pdf>
2. Autoregressive Models in Deep Learning - *by: George Ho, published: 9/3/2019* <https://www.georgeho.org/deep-autoregressive-models/>

# Make your own Rick Sanchez (bot) with Transformers and DialoGPT fine-tuning - *by:* [*Rostyslav Neskorozheny*](https://medium.com/@slanjr?source=post_page-----f85e6d1f4e30--------------------------------)*, published at: 8/6/2020* [*https://towardsdatascience.com/make-your-own-rick-sanchez-bot-with-transformers-and-dialogpt-fine-tuning-f85e6d1f4e30*](https://towardsdatascience.com/make-your-own-rick-sanchez-bot-with-transformers-and-dialogpt-fine-tuning-f85e6d1f4e30)

1. Perplexity in Wikipedia <https://en.wikipedia.org/wiki/Perplexity>

# Text Generation With LSTM Recurrent Neural Networks in Python with Keras - *by: Jason Brownlee, last update: 6/7/2022* <https://machinelearningmastery.com/text-generation-lstm-recurrent-neural-networks-python-keras/>