Visualising Twitch.tv emote semantic similarity through pre-trained language models.

732A92 Text Mining - Project

Salvador Martí Román

January 15, 2022

Abstract

Online subcultures throughout the internet have, over time, developed unique variations of common languages. One such way in which these differences manifest themselves is through the use completely platform and community specific emotes.

This work aims to produce a visual representation of how these sometimes obscure expressions relate to more widely spread ones through learnt word embeddings. Said embeddings are obtained by further pre-training a widely available language model[5] on a wide variety of publicly available Twitch.tv chat logs[3].

Contents

1 Introduction

1.1 Motivation

Languages experience constant change through continuous use and cultural shifts. Since the start of the internet many of these changes, specially seen in teen culture[9], have in origin online subcultures shared by a number of relatively small communities.

As of writing, a great number of pretrained language models exist but none of them have been specifically trained on Twitch.tv's community jargon. The fist of this works outputs is a visual representation of semantic similarity between commonly used emotes and platform specific ones. This in enough itself is of some academic interest as almost all of the new emotes are extremely platform specific words are not in the pre-training stage. The visualization will demonstrate how language models are able to generalise in open vocabulary scenarios.

1.2 Groundwork laid

This project builds a foundation for further studies concerning the effectiveness of pre-training language models on Twitch data before fine-tuning them to tasks such as spam detection for moderation tools.

This works second contribution, in adition to the visualization, are the weights for the twitch pre-trained distilRoberta-base[8] model. This could serve as a starting point for further performance comparisons of the effectiveness of domain specific pre-training.

2 Theory

2.1 Tokenizers

2.1.1 Open vocabulary tokenizers

The role of a tokenizer is to split a continuous stream of data into small repeating groups of elements. In natural language processing this atomization of language elements, be it at word or character level, allow the models to compress the input representation. This compression can have a great effect in the length of context a given model can capture as well as its computational efficiency while training and infering.

Tokens that are composed of longer strings of information are better at compressing representations. However, the longer these tokenised sequences are, the more likely they are to occur infrequently which poses a problem in enough itself. In modern NLP approaches, it is common to represent each token with a V dimensional vector where V is the size of the vocabulary. This vocabulary contains a entry for every type of token that a given string of information can be broken down into.

From a compression perspective the most effective tokenizer is that which minimises $L * V$ where L is the amount of tokens into which the data stream is broken into. This approach can work well when dealing with a problem with closed vocabulary where the model is never going to encounter an unknown token but often fails to generalise outside of this.

Since current state of the art approaches revolve around the costly pre-training of a general language model and the subsequent fine-tuning for specific tasks, it is extremely likely the model will encounter out of vocabulary sequences.

A naive solution to this would be including smaller and more generic building blocks into which these new sequences could be tokenised as a back-up. For example, an English text tokenizer could contain

the 26 letter of the English alphabet in the vocabulary. This would allow the tokenizer to break down an unknown word into its known components.

2.1.2 Byte-Pair Encodings (BPE)

Originally conceived as a data compression technique[1], BPE has become a common hands-off approach to vocabulary building.

At character level, the vocabulary is initialised with a list of every chracter contained in a corpus. Note that this not only contains letters. It may potentially include other single unicode characters like spaces. Once done, the corpus is read and the adjacent occurences of vocabulary tokens are counted. Those two tokens which occur most frequently together are then added to the vocabulary and the count begins once more. This process is repeated until an arbitrary maximum vocabulary size is reached or a minimum adjacent token amount is observed.

The following table shows a practical example of how the vocabulary increases in size over a series of 5 iterations. For simplicity, the space character is considered a hard delimiter instead of a token to be counted.

Token vocabulary	**Sentence tokenization**
{s, h, e, l, a}	S-h-e s-e-l-l-s s-e-a-s-h-e-l-l-s
{s, h, e, l, a, sh}	Sh-e s-e-l-l-s s-e-a-sh-e-l-l-s
{s, h, e, l, a, sh, he, se}	Sh-e se-l-l-s se-a-sh-e-l-l-s
{s, h, e, l, a, sh, he, se, ll}	Sh-e se-ll-s se-a-sh-e-ll-s
{s, h, e, l, a, sh, he, se, ll, she}	She se-ll-s se-a-she-ll-s

Table 1: Trace of 5 iterations of the BPE algorithm at character level.[7]

2.1.3 Byte-level Byte-Pair Tokenizers

As language models become increasingly popular, flexibility becomes a main concern when building when building a tokenizer for them. The need for multilingual-ready models make it particularly challenging to continue a character level approach to BPE tokenization. One widely used solution, is to consider the smallest possible unit of meaning to be a single byte.

Since every unicode character is 4 bytes in length, this approach allows for the creation of what is an essentially universal tokenizer. This comes at the cost of a potentially reduced compression rate which can limit sequence lengths. This is then mitigated by the BPE vocabulary building technique covered above whose purpose is to minimise this compression loss.

2.2 Language Models

As of a few years ago, large transformer-based pre-trained language models have become very common in state of the art approaches for a wide variety of tasks. The main reason behind this is that, while data is available in massive quantities, annotated data is much harder to come by and expensive produce. As a result, there is a huge incentive to leverage available data even though it might not be directly related to the task at hand.

Language models are pre-trained in self-supervised token unmasking tasks. In this pre-training stage, given a series of tokens W, the model must then predict a token w^t which can has been masked or swapped for another token. These tasks vary slightly depending the LM being used. Encoder based models such as BERT prefer predicting masked tokens given their sorrounding context while decoder based models like GPT-2 mask every token then predict from only the previous ones.

In both cases the objective is to learn the probabilty of a word given its context. For example, for a bidirectional encoder that considers 4 surrounding words the probability distribution learnt can be written as follows:

$$P(w^t \mid w^{t-2}, w^{t-1}, w^{t+1}, w^{t+2})$$

The model weights learnt from this task can then be fine-tuned to any task with minor changes to the last layers of the architecture.

2.3 Transformers

Before explaining the architecture used to produce the results of this work, it is first important to understand the characteristics of the building blocks that make it.

2.3.1 Self attention mechanisms

Transformers were first proposed in the 2017 paper "Attention is all you need"[10]. In this paper the authors propose a new kind of layer which builds linear combinations of inputs, the weights of which could be learnt through backpropagation.

The resulting architecture is able to learn which inputs to give importance to and which ones to ignore depending on the values it gets as input. A number of more efficient ways to compute self-attention have been created since the publishing of this paper but at their core they all achieve a similar purpose.

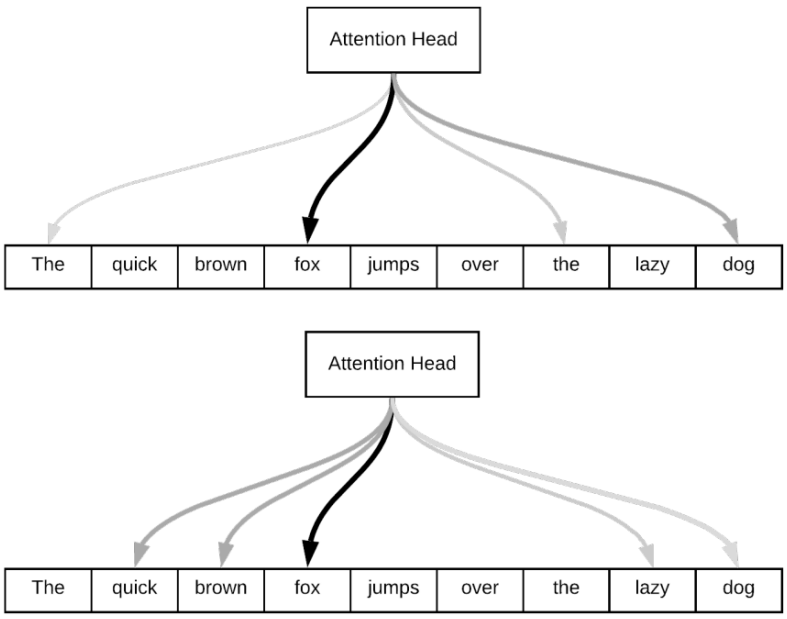


Figure 1: Illustration of the attention concept in neural networks. Line thickness and darkness indicate more importance[7].

Each attention mechanism A receives a matrix of values V and a Key matrix K which indexes the values in V. Each head then poses a query Q to the key matrix. Calculating the dot product between K and Q yields the cosine of the angle between the queries and the keys. Since the keys index the values, the distances calculated can be used to see which of the values should be given more importance in the

weighted linear combination. This can be written as follows:

$$A(Q, K, V) = softmax(QK^T)V$$

2.3.2 Scaling properties of the transformer

The following section will attempt to explain how the scaling advantages of transformers have made large scale language models computationally viable. The main 3 scaling factors are the sequence length n, representation size d and for convolutional networks kernel size k.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Classic Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(\log_k(n))$

Table 2: Comparison of layer properties. [10]

From the table above we can see every approach has a different trade-off.

Recurrent neural networks RNNs excel due to their excelent complexity scaling caused by sequence length. This allows RNNs to theoretically have extremely long context windows, however, this potential rarely manifests due to gradient vanishing problems caused by the poor maximum path length scaling.

Convolutional networks similarly excel at sequence length complexity scaling with the added benefit of no sequential operations, allowing for bigger models to be trained in masively parallelised clusters. The main drawback is the rather poor complexity scaling with regards to representation size. As covered in the BPE section of this work representation size is often the vocabulary size which is often much larger than the sequence length.

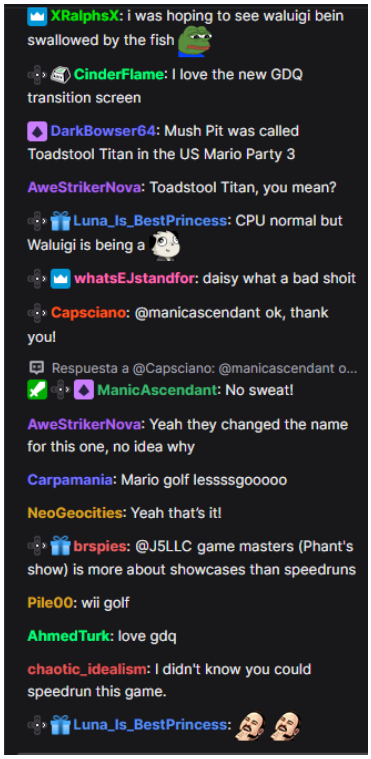
The main reason transformers dominate, in spite of the fact that classic self-attention mechanisms scale poorly with its limited sequence lengths, is due to its massively parallelisable potential and short maximum path length. This allows 1.5 billion parameter models like GPT-2 or smaller ones like BERT to be trained on massive clusters of GPUs or TPUs at the same time greatly accelerating the process. The better scaling with vocabulary length can partially alleviate problems caused by the limited context windows thanks to better compression at the tokenization stage.

2.4 RoBERTa

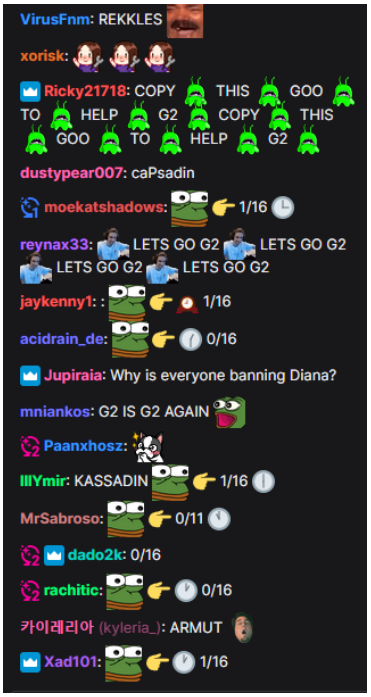
First proposed in 2019, RoBERTa[5] is an English retraining of the original BERT architecture covered in lecture 4 of the text Mining course[4]. In its proposal article, they found the original model to be undertrained.

These new model weights were a product of a much longer training stage over a significantly larger and more varied dataset. In addition to the 16GB bookcorpus[12], this refined training also adds an additional 144GB of text data in the form of the CC-news[6] and Openwebtext[2] corpora.

In addition to these changes, the next sentence prediction NSP task was dropped and left intrasentence token masking as the sole self-surpervised task. RoBERTa also leverages data better, with different tokens being masked every epoch for some additional data augmentation.



(a) A game speedrunning charity event.



(b) A League of Legends tournament chatlog.

Figure 2: Chat comparison of different communities.

3 Data

The corpus used on this project[3] is published under a CC0 license and is free to use while accompanied with a citation.

3.1 Chat Log Data

The data used in this project originates from the livestreaming platform `Twitch.tv`. This website is primarily known for its videogame related content and for its live audience interaction through chat. The content creators in this platform may choose to archive their live performances so that their viewers may access the content on demand. Since live audience reactions are considered a meaningful part of the experience for many users, moderated chat comments are also archived alongside the video component.

The corpus used in this work takes advantage of this archival feature to obtain a log of the comments in videos. This corpus, collected between the dates of 2018-04-24 and 2018-06-24, contains 2162 logs from 52 of the platform's most popular and as a result, most culturally defining, streamers. After discarding the metadata, the corpus is left with 2.1GB worth of text. A example of what these transcripts can look like is seen in figure 2.

3.2 Twitch Emotes

Unlike unicode emoji, Twitch emotes do not have a reserved 4 byte character. Instead, users type in the name of the emote using a series of characters. For instance, the name of the emote present in the first line of figure 2a is *"FeelsBadMan"*. The user client reads the name and exchanges it for the corresponding image. It is important to note that these names are space and case-sensitive.

4 Methodology

For the sake of reproducibility, all code for each of the different stages is licensed under GPLv3 and made publicaly available in separate notebooks at `https://github.com/TheClassyPenguin/Twitch_Emote_Visualization`

4.1 Data preparation

The original corpora are pickeled pandas dataframes split by content creators. The first step in the data pre-preprocessing process is the merging of all dataframes into a single Hugginface dataset. This data structure is designed to preprocess and facilitate the use of large corpora for training. The main advantages are its robust parallelised filtering and operation mapping aswell as on-disk caching of intermediate results.

This last feature proved to be invaluable when preprocessing the tokenised corpus as the sheer size of it made it impossible to load into memory. Pandas chunk reading dataframes and python generators can be used to a similar end, but the close integration with Hugginface's transformers library made it by far the most efficient and robust option.

Models in the transformer library come bundled with their tokenizers, making it simple to preprocess text. The language model used for this project, ´DistilRoBERTa-base´, uses a case-sensitive byte level BPE tokenizer. This tokenizer is a strong fit for the use case as, while twitch is a predominantly english platform, there is a strong international presence and other languages with different character sets can be easily encountered. Additionally, the case sensitive nature of the model will in theory help distinguish when an emote is being used, making their representations distinct.

A map operation that tokenizes then truncates sequences longer than the maximum sequence length of 512 tokens is applied over every message. Sequences shorter than the maximum length are batched together into a single sample to help contextualize chat messages. The entire data pipeline operation takes around 25 minutes on a 20 core 40 thread Google colab machine.

A relatively small (1%) testing set is set apart to monitor the models loss improvement over time.

4.2 Model training

Due to the limited availability of computing resources, a distilled model seemed the best compromise between performance and training time. Learner model weights in knowledge distillation are tipically much smaller than their full-sized teachers and can perform better than models of the same size trained from scratch[11]. Of all the possible BERT models RoBERTa is chosen with the hypothesis that its training on Openwebtext could help expose it to internet jargon better than book collections. If this hypothesis holds true training time and results obtained could be improved over using a similarly sized BERT model.

This language model is trained for just under 2 days on a P100 colab GPU with a batch size of 8 fully leveraging the GPU's memory. Additional parameters used in the training configuration are available in the model training notebook.

4.3 Evaluation

The quality of the visualization produced is subjectively judged by the author of this work leveraging its intuitive understanding of the emotes achieved through repeated exposure to them. 98 of the most popular emotes during December 2021 are chosen for comparison.

The representations are obtained by mean pooling together the final attention values for each token composing an emote. PCA is then used to obtain the two most important principal components which are plotted in a labelled scatterplot.

Perplexity on the test set is used as a more academically rigorous measure of the models learning over time. This loss is defined as one over the probability of each word in the test set one after the other. This probability is given by the model trained. The lower it is the better the predictions are.

$$P(W) = \sqrt[n]{\frac{1}{P(w^1, w^2, ..., w^n)}}$$

5 Discussion

As can be observed on figure 3 clusters seem to coalesce as training progresses. After some additional scrutiny, some emotes that seem oddly placed were discovered to be created after the chat data was collected. This meant that emotes such as *"pcrowDoodle"* wouldn't have have any mentions in the text which justifies their odd placement.

A number of reasonable trends can be observed from the data at 400000 steps. To the bottom left, a cluster of emotes often used in a particular content creator's chat has formed. The streamer associated with some of the emotes found in that tight cluster is called Forsen, a former competitive esport player, and the emotes reflect the high energy, high excitement facets of his streams. The emotes within that cluster belong to community similar to the one seen in figure 2b and have been occasionally banned on other creators chats due to their tipically high spam usage.

From the bottom right an overall positive sentiment trend can be observed with *POGGERS, KEKW, OMEGALUL* and *POGchamp* being some of the most popular positive excitement and laughter emotes in the platform.

To the top left emotes associated with loses, rage and negative excitement are losely positioned together. Notably *ppOverheat, NotLikeThis, BabyRage, MonkaS* and *FeelsbadMan*. However this section is fairly noisy as many other emotes within the are don't quite fit this category.

A number of explanations exist for the slightly underwhelming results observed. The simplest explanation being that the model is undertrained. 2 training days given the available hardware wasn't enough for it even complete one epoch with the distilled model. It is likely that training for a longer amount of time would help the results slightly. However, this model has already been trained to the point of dimishing returns. Every evaluation stage after 400000 steps appeared to reduce perplexity very slowly.

It would be also possible that a clearer emote clustering emerged if the data collected matched the dates the most popular emotes are selected. In the 3 years since the corpus was collected the platform and communities have changed substantially and so has the usage and popularity of different emotes.

One final interesting possibility is that the nature of twitch chat is fundamentally different to other written mediums. While sentences in a text are tipically related to the previous ones, this is not necesarily the case for streaming chats. Chat messages mainly react to the video content, so previous messages are not necessarily related with what is going to be written next. In a way it's like training the model on only one side of a 2-way conversation, a lot of context is potentially missing.

6 Conclusion

Overall the trends and clusters observed in the emote grouping point towards emote use by twitch users not being random. As the model is trained on the data, trends based on co-ocurrance in communities and sentiments seem to become stronger.

As these trends are noisier than otherwise expected a number of questions are proposed for further research:

- Could further training or model model switching produce better results?
- Does mean pooling of token embeddings to obtain word representations lead to a significant loss of information? Would methods like Word2Vec produce better visualizations?
- How time sensitive are emote usage trends on Twitch.tv?
- Is a lot of important context lost looking only at chats when there is a third source influencing the conversation?

References

[1] Philip Gage. "A new algorithm for data compression". In: *The C Users Journal archive* 12 (1994), pp. 23–38.

[2] Aaron Gokaslan and Vanya Cohen. *OpenWebText Corpus*. http://Skylion007.github.io/OpenWebTextCorpus. 2019.

[3] Jeongmin Kim. *Twitch.tv Chat Log Data*. Version V2. 2019. DOI: 10.7910/DVN/VE0IVQ. URL: https://doi.org/10.7910/DVN/VE0IVQ.

[4] Marco Kuhlmann. *732A92/TDDE16 Text Mining (2020) Lecture 4, Word embeddings*. https://www.ida.liu.se/~732A92/commons/TM-2020-4.pdf. Accessed: 2022-01-10.

[5] Yinhan Liu et al. *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. 2019. arXiv: 1907.11692 [cs.CL].

[6] Sebastian Nagel. *CC-News*. http://web.archive.org/save/http://commoncrawl.org/2016/10/newsdataset-available.. 2016.

[7] Salvador Marti Roman. "Application of Machine Learning Techniques for Text Generation". In: *riunet* http://hdl.handle.net/10251/149583 (2020).

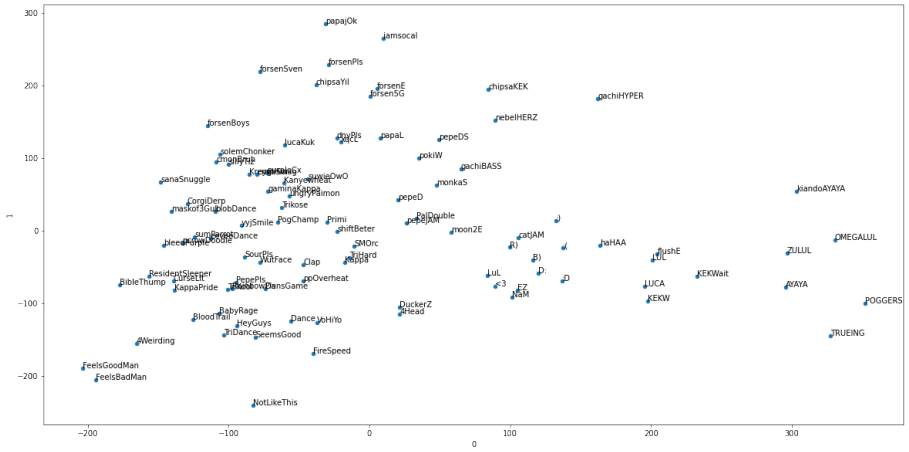
[8] Victor Sanh et al. "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter". In: *ArXiv* abs/1910.01108 (2019).

[9] Sali A Tagliamonte et al. "So sick or so cool? The language of youth on the internet". In: *Language in Society* 45.1 (2016), pp. 1–32.

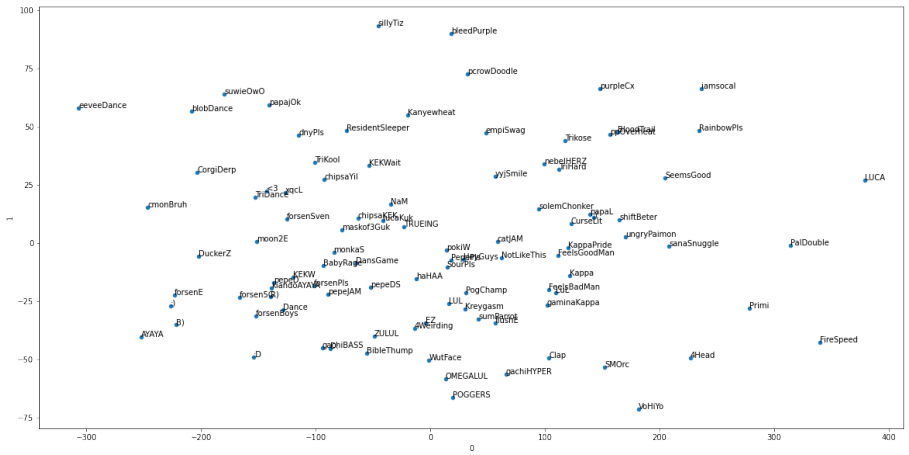
[10] Ashish Vaswani et al. *Attention Is All You Need*. 2017. arXiv: 1706.03762 [cs.CL].

[11] Xing Wu et al. *Distilling Knowledge from Pre-trained Language Models via Text Smoothing*. 2020. arXiv: 2005.03848 [cs.CL].

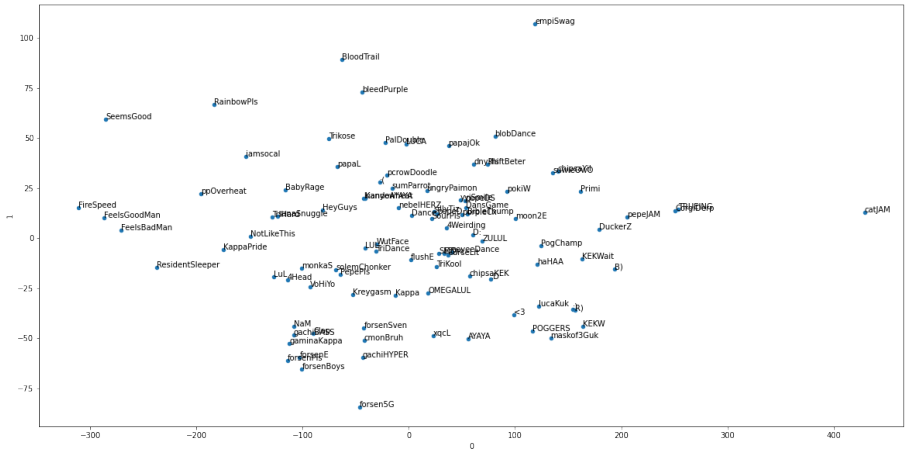
[12] Yukun Zhu et al. *Aligning Books and Movies: Towards Story-like Visual Explanations by Watching Movies and Reading Books*. 2015. arXiv: 1506.06724 [cs.CV].



(a) Representations before fine-tuning (Perplexity: >5.565)



(b) Representations at 160000 steps (Perplexity: 2.918)



(c) Representations at 400000 steps. (Perplexity: 2.573)

Figure 3: Plotting of the first two principal components of the representations obtained for each emote at different stages.