Real-time prediction of popularity and trends on Reddit

Luis Fernández González

Final Year Dissertation
Escuela de Ingeniería de Telecomunicación
Degree in Telecommunications Technologies Engineering

Tutor
Carmen García Mateo

Year 2019-2020
Real-time prediction of popularity and trends on Reddit

Author: Luis Fernández González
Advisor: Carmen García Mateo
Year: 2019-2020

1 Introduction

Predicting the popularity of social media posts and detecting new trends is a incredibly powerful tool for any business with a measure of internet presence. Being the first to capitalize on viral content is a really competitive game and, machine learning, has proven to be one of the best approaches for detecting those viral gold nuggets hidden inside the massive stream of new content that flows through the net.

Reddit has been selected to form the basis of a representative sample of updated and potentially trending content. Publicized as “The front page of the internet”, Reddit is one of the most popular discussion and content rating websites. Users can post, comment and evaluate other users posts through a system of upvoting and downvoting. The votes determine the popularity of a given submission and, even more, its visibility on the site. The most popular submissions are displayed first on the site’s content feed, thereby benefiting the explosive growth of viral content.

Predicting the popularity of social media and news is a well charted endeavour, but not many attempt to do so by using modern natural language processing (NLP) techniques such as using the language modelling (LM) capabilities of transformer models and their contextual word embeddings. Deep learning has taken the machine learning (ML) world by storm and this new transformer architectures greatly outperform previous approaches like recurrent neural networks (RNN) or bi-LSTM [1]. All the more, transformers allow end users to leverage massive pre-trained models that can be fine-tuned to accomplish a variety of downstream tasks.

In this work, we will attempt to design the building blocks of a complete pipeline able to predict the popularity of new Reddit submissions in real time. The three main components of this project are:

- A dataset to train and test our model.
- A Reddit scraper to get new submissions.
- A trained predictive model.

The focus will be on building a working foundation that serves as a prove of concept and that can be further improved with more data, resources and research.
2 Related Work

There are two different outlooks when it comes to the prediction of popularity on Reddit and similar websites. You can either try to get your prediction on a cold start using only the content features available at the time of posting (which has historically achieved mixed results [2]) or you can leverage meta-features and time-series data for the forecasting.

A cold start approach is taken by Lakkaraju et al. [3], who built a virality prediction model trained over re-posts of the same content while varying title, community and time as the features achieving promising results.

Recent research has started to take advantage of NLP. Ahmed Shuaibi [4] uses GloVe [5] embeddings and sentiment classification to expand their feature set, while other approaches add even wilder features like using structural network features to augment the linguistic data like Makov et al. [6].

Transformers models (further discussed on section A.8) are sparsely used in prediction literature but recent works manage to leverage their capabilities like Weller et al. [7] who use Bidirectional Encoder Representations from Transformers (BERT) [8] to identify trolls using a small supervised dataset.

3 Goals

From a research perspective, this project’s objective is to study the performance of different transformer based neural network architectures on modelling such an abstract feature as popularity.

On a practical level, the end goal is developing a machine learning project from the ground up, implementing the building blocks of a real world pipeline. Framing it as a business, this work aims to explore the possibilities of detecting popular content on Reddit just by using the text title as a feature, thereby allowing the detection of popular content in real time or, even helping business and users alike to decide whether or not the content they are planning to post has chances of becoming viral.

4 Implementation

The project has been developed in Python using the deep learning framework SimpleTransformers which is built on top of HuggingFace’s Transformers library [9] and Pythorch [10]. Weights & Biases [11] was used for experiment tracking while the dataset was built using the Reddit API and its Python wrapper PRAW [12].

4.1 Building the dataset

In order to properly frame your model, the first step in any any ML project is to understand the data you are working with. Our only feature will be the titles of Reddit submissions, while our target is the score (calculated from upvotes and downvotes). The dataset was built querying the Reddit API following the directives described below, resulting in almost half a million valid submissions.

4.1.1 Scope and selection bias

Computational constraints forces us to incur in two main forms of selection bias, the first one being subreddit selection. Reddit is divided into a variety of topic-specific subsections called subreddits. The chosen subreddit was AskReddit, whose only rule is that each submission has
to be a single question, with no topic limitation whatsoever and no extra content or media attached. This allows us to not limit our dataset to a specific topic while obtaining unbiased submissions titles (no image or body content will affect the popularity of the submissions).

Secondly, we decided on limiting the time frame to a single month (April 2020) while giving each submission a lifespan of at least one month.

It is also vital to note that all the submissions are written in English and catered an English speaking audience, so this could be also considered a cultural and language bias. This bias helps us in the implementation because all the vocabularies used by the models tested on the project are based on the English language and are completely language dependant.

4.1.2 Data exploration and framing the problem

By studying the score values of the different submissions as represented in table 1, the virality potential of some content is easily noticed by observing the disparity between the standard deviation and the mean values.

<table>
<thead>
<tr>
<th>count</th>
<th>490992</th>
</tr>
</thead>
<tbody>
<tr>
<td>mean</td>
<td>3.70</td>
</tr>
<tr>
<td>std</td>
<td>293.82</td>
</tr>
<tr>
<td>max</td>
<td>78525</td>
</tr>
</tbody>
</table>

Table 1: Dataset score statistical data.

A reasonable classification for a popular post is one that has a score above the mean. This criteria is met by 5.53% of the submissions as depicted in table 2.

<table>
<thead>
<tr>
<th>score&lt;mean(score)</th>
<th>score&gt;mean(score)</th>
</tr>
</thead>
<tbody>
<tr>
<td>463793</td>
<td>27199</td>
</tr>
</tbody>
</table>

Table 2: Label distribution.

With the data available and, taking into account the computational limitations of this research, the prediction task is framed as a binary classification problem where the labels “0” and “1” represent whether the score is under or over the mean value for the whole dataset. With a bigger dataset is possible to harden the criteria, looking for even more popular submissions.

4.2 Validation sets

Hold-out validation is used instead of K-fold cross-validation mostly for simplicity and computational reasons. It is not an easy task to properly implement K-fold cross-validation on this imbalanced dataset (section 4.6.1) while still being able to fairly compare the performance of the five different used models. At the end, the final dataset is split into training (60%), validation (20%) and test (20%) sets.

4.3 New submission scrapping

A similar script as the one used for creating the dataset is running 24/7 on an Amazon Web Services (AWS) instance, storing each new submission in a csv file.

Using the trained model, it is possible to predict whether or not the new submissions will be popular.
4.4 Transformer models

Transformer model based architectures currently hold the crown in the vast majority of NLP tasks. They use transfer learning, where a first step, called pre-training, is used to train the model on a self-supervised task on huge amounts of unlabeled text data. After this step, the language modelling part is complete and the model can be trained again on a smaller labelled dataset. This is called fine-tuning and it yields a far superior performance compared to other deep learning approaches.

Five different architectures, based in different transformer models, were trained and compared for this task. The models utilized are: BERT base, DistilBERT, RoBERTa, ELECTRA base and ELECTRA small (further described on section A.9 and section A.10).

4.5 Neural network architecture

The neural network architecture (Fig. 1) used for all the variations will remain almost exactly the same during the five different experiments (some minor modifications will be needed and explained for certain models).

Figure 1: High level example view of the network architecture.

Thanks to the language modelling realized during pre-training, network architectures that use transformers for common NLP tasks (like classification) are really straightforward. First, the text is inputted into the transformer which is connected to a dropout layer to reduce overfitting. Finally, the dropout layers is fully connected to a final classifying layer.

4.6 Model Training

With no powerful hardware was available for the research, the solution was performing all the training using Google Colab’s free online environment. It offers 12 hours long run-times with Nvidia K80 GPUs, which have 12Gb GDDR5 VRAM and are CUDA compatible (2496 CUDA cores) while also allowing for fp16, reducing the training time by a big margin.

The hyperparameter values are set to general, sensible values without doing hyperparameter optimization. The training and evaluation batch sizes were 128, the learning rate was $4 \times 10^{-5}$ and every model trained for 3 epochs. Adam algorithm [13] is used for gradient optimization.

4.6.1 Data imbalance

In order to deal with the imbalance of the dataset, class weights are used while evaluating during the training process. An up-sampling of the minority class is used during loss calculation to avoid overfitting to the majority class.
4.6.2 Text pre-processing

No pre-processing like removing stop words, lemmatization or stemming is applied to the text. All the pre-trained models used are case sensitive in order to take into account attention grabbing usage of capitalization.

In order to fine-tune a pre-trained model it is necessary to use exactly the same tokenization, vocabulary, and index mapping as used during the pre-training. The processes of tokenization involves splitting the input text into list of tokens that are available in the transformer model vocabulary. To deal with out of vocabulary (OOV) words, BPE based WordPiece tokenisation is used [14].

All the vocabulary sizes (these are the different tokens that can be represented by the inputs passed to transformer and not the actual words) are 30522 except for DistilRoBERTa which has a size of 50265 tokens.

4.7 BERT

The first architecture implements BERT with the objective of getting a good ground truth to compare other models. It follows the design described on 1 and achieves an accuracy of 0.7479. Further discussion of evaluation metrics will be reserved for section 5.

In order to help the reader visualize and compare the transformer models to other deep learning architectures, a summary of the dimensions of the BERT model is included:

- There are 12 hidden layers in the Transformer encoder.
- There are 12 attention heads for each attention layer in the Transformer encoder.
- There are 768 hidden layers.
- There are 110 million parameters.

4.8 Model size and distillation

To be able to deploy the model in production, a compromise between performance and complexity has to be reached. Distillation [15] allows this, reducing these large models into smaller yet almost as efficient and more production-friendly models. A distilled version of BERT (DistilBERT [16]) is also implemented and tested.

<table>
<thead>
<tr>
<th># Parameters (millions)</th>
<th>BERT base</th>
<th>DistilBERT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training Time</td>
<td>$8 \times v100 \times 12\text{days}$</td>
<td>$8 \times v100 \times 3.5\text{days}$</td>
</tr>
</tbody>
</table>

Table 3: Model size comparison.

Is important to note that there is a slight variation network architecture used with distil-BERT. Following Huggingface’s implementation, it includes a pre-classifier layer in between the transformer output and the dropout layer.

In paper, by distilling BERT base is possible to achieve around 95% of its performance while running a 60% faster and using 40% less parameters (table 3). The actual achieved accuracy is 0.706, which represents a 0.05 decrease compared to the non-distilled version.
4.9 DistilRoBERTa

RoBERTa: A Robustly Optimized BERT pre-training [17], is one of the first iterations over BERT’s pre-training procedure. It manages to outperform BERT by using longer training batches and increasing the pre-training dataset by tenfold. The pre-training tasks are also changed; it removes the next sentence prediction and dynamically changes the masking pattern applied to the training data.

The distilled version is used in the project, and the neural network architecture used mirrors the one used for distilBERT, consisting in the transformer, a pre-classifier layer, a dropout layer and a final fully connected classifier layer. The resulting accuracy is 0.6547.

4.10 ELECTRA

Pre-training Text Encoders as Discriminators Rather Than Generators (ELECTRA) [18] pushes forward effective computing, outperforming the previously discussed models given the same compute budget. The base and small versions are used with 110 million and 14 million parameters respectively. Looking at the network architecture it follows the one used in BERT but adds a Pooling layer between the model and the dropout layer.

The base version yields an accuracy of 0.7421 (a good result compared to BERT’s 0.7479) while the small version gets an accuracy of 0.6554.

5 Evaluation

Backed by a discussion about used metrics, a comparison between the 5 architectures will be presented.

5.1 Overfitting

In order to avoid overfitting, all models are evaluated during training using the validation set where an early stopping mechanism is applied based on validation loss.

5.2 Weights & Biases

All the experiments have been tracked using Weights & Biases so they can be further analyzed and easily replicated. A complete track of all the relevant metrics, hardware performance and logs is available online [19].

5.3 Model Comparison

Taking into account the final performance of each model and its computational demands, a final conclusion has to be reached about which model will be used in production. A good first approach when comparing the performance of classification models is the accuracy metric, which offers the most intuitive look as represented in Fig. 2.
On the other hand, even in a model with high accuracy some problem may arise. By using accuracy, we are assigning equal cost to false positives and false negatives and, in an imbalanced dataset like the one at hand, this can result in a model incapable of detecting true positives of the minority label.

5.3.1 Matthews correlation coefficient

Matthews correlation coefficient (equation 1) is the metric used to determine the best model of the lot.

Even though there are various metrics widely used in classification tasks like precision, recall or F1 score, when performing binary classification MCC is a more reliable statistical rate. It uses the four confusion matrix categories for binary classification: true positives (TP), false negatives (FN), true negatives (TN), and false positives (FP).

\[
MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}
\]

(1)

MCC results in a high score only if the prediction obtained good results in all of the four confusion matrix categories while taking into account the balance of the dataset.

Looking at figure 3 we can appreciate that DistilRoBERTa achieves the highest MCC.
5.4 Results

Taking into account the previously discussed computational demands and comparing the metrics of the five experiments DistilRoberta comes on top. By it being a more modern iteration of BERT, it does not come as a surprise seeing it achieving better results than both BERT based implementations, while both ELECTRA approaches end up lacking in performance compared to the bigger DistilRoBERTa model.

The final metrics obtained after evaluating the models on the test set can be seen in table 4.

<table>
<thead>
<tr>
<th></th>
<th>BERT</th>
<th>DistilBERT</th>
<th>DistilRoBERTa</th>
<th>ELECTRA base</th>
<th>ELECTRA small</th>
</tr>
</thead>
<tbody>
<tr>
<td>accuracy</td>
<td>0.7479</td>
<td>0.706</td>
<td>0.655</td>
<td>0.7421</td>
<td>0.6554</td>
</tr>
<tr>
<td>precision</td>
<td>0.09922</td>
<td>0.09554</td>
<td>0.094</td>
<td>0.0968</td>
<td>0.09012</td>
</tr>
<tr>
<td>recall</td>
<td>0.4398</td>
<td>0.5092</td>
<td>0.597</td>
<td>0.4346</td>
<td>0.5651</td>
</tr>
<tr>
<td>f1</td>
<td>0.1619</td>
<td>0.1609</td>
<td>0.162</td>
<td>0.1583</td>
<td>0.1554</td>
</tr>
<tr>
<td>mcc</td>
<td>0.1094</td>
<td>0.1137</td>
<td>0.125</td>
<td>0.1032</td>
<td>0.1088</td>
</tr>
</tbody>
</table>

Table 4: Evaluation metrics.

Taking a deeper look into DistilRoBERTa’s confusion matrix (Fig. 4) we can see that the model barely classifies over more than 50% of the popular submissions correctly. Even more, it is important not to omit that the number of false negatives is ten times the number of true positives, which makes us question the quality of the predictor.

![Figure 4: DistilRoberta’s confusion matrix.](image)

6 Future Work

Further work into this research can follow two routes: increasing the model performance and pipeline optimization.

A proof of concept has been achieved using very scarce resources. Increasing the dataset time frame to a whole year should be the first step to increase the model performance. Hyperparameter optimization and bigger models would the natural continuation.

Another interesting approach would be adding more features (like the time of publication) which could drastically increase the overall performance. During the development of this project there was an attempt use named-entity recognition (NER) in combination with Google Trends to add new features to the model. Due API limitations and time constraints it was impossible to get enough data for the whole dataset but it remains an interesting avenue to pursue.
Finally, the pipeline as a whole could be greatly enhanced. Improving on the current storage system using a database, automating the interaction between the different scripts or adding an audit framework to the whole pipeline would be required improvements for an efficient real world setting.

7 Ethical Considerations

The ability to predict popular content raises the concern about blindly abusing viral content without regard of the transmitted message. Misinformation and plagiarised or harmful content are prime examples or potentially trending topics that should be filtered if an automated system is implemented using similar ideas as the ones described in this article.

8 Conclusion

A popularity prediction pipeline was developed using unlabelled freely available data. While not good enough to warrant a real implementation yet. The achieved results prove the possibility of implementing a popularity prediction system. As is the norm with deep learning approaches, more data and more computational power usually yield better results, which hints of promising results waiting ahead should this research continue as discussed in section 6.

On a final note, the performance and easiness of implementation of various transformer models paints a bright future for the NLP field. No dedicated hardware was used during the research, making use of freely available cloud services. Even though a proper setup would be needed before achieving a production ready product, the current ecosystem proves that we are in a gold age for machine learning.

References


Appendix

A State of the Art

A.1 Introduction

NLP has developed by leaps and bounds during the recent years and the interest keeps increasing [20]. High level implementations of deep learning architectures allow a newcomer to achieve similar results to the SOTA easily, but understanding how and why they work or trying to come with new approaches proves to be a daunting task.

When doing research about a specific topic within the field of deep learning NLP is usual to find that all the available information assumes previous knowledge of different concept and architectures. Understanding the top performing models can seem, at first, impossible because of the huge amount of new terms and concepts present.

Even thought BERT has been outperformed by recent models, this article assumes that it will remain as a landmark model because its role and as an stepping stone for understanding other models and further advancements. Following this principle this text will dive into the field following the long line of breakthroughs that resulted in the current SOTA, as represented in Fig 5.

Figure 5: Building blocks of current deep learning models.

Understanding the state of the art by working up from a simple single neural cell to the complex pre-trained models used today is the aim of this section. This will give the reader to gain a better understanding of NLP and deep learning while providing strong foundations to further its research.

A.2 NLP and Deep Learning Basics

Before getting into how neural networks evolved into the complex architectures we see today we have to have a firm grasp of the basics. Even though this section aims to help newcomers to the field understand new concepts, it assumes a minimal understanding of machine learning requiring at least a intuition of it to properly follow its contents.

NLP is a branch of artificial intelligence that aims to help computers understand, interpret and manipulate human language. The main focus of this discussion will be written text, but do not ignore that speech is also deeply studied and many of the models and methods are applied parallelly in both areas.

Before understanding deep learning we have to know what is a neural network. A neural network is an architecture inspired by biological neurons and designed to fit a function that allows us to make prediction over a set of features. Generally speaking, a neural network
structure includes three fundamental elements; an input layer, a hidden layer, and an output layer. The training step consist in modifying the network parameters so we reduce the difference between the prediction and the actual label.

The basic unit of a neural network is the neuron and, in broad terms, can be defined by two elements; input and output. When the neuron activates, it computes its state by adding all the incoming inputs multiplied by its corresponding parameter or weight as referenced in equation 2.

$\Sigma = W_1 x_1 + W_2 x_2 + b \quad (2)$

After computing its state, the neuron passes it through its activation function, which normalizes the output.

$H_{out} = \sigma(\Sigma) \quad (3)$

Figure 6: Basic Neuron diagram.

A layer of a neural network consists of an arbitrary number of neurons. When a neural network has just one hidden layer is called a shallow neural network. In order to step into the realm of deep learning we need more than one hidden layer. The intuition behind the benefits of adding more layers is that a wide enough network can fit any function. Is useful to know that when defining a neural network, width refers to the number of layers while height refers to the number of neurons on a layer.
A.3 Recurrent Neural Networks

Now that we have the basis of a neural network, it is time to focus on the task at hand; NLP. By its nature, language has a defined ordering. The sequence in which we deliver or receive words (even letters or phonemes) matter. In the simple neural network presented before, all the inputs and outputs are independent of each other. Imagine now, using aforementioned network to try to predict the next word of a sentence. You will realize that knowledge of the previous words is required, hence there is a need to remember previous states. To fix this problem and work with sequential data Recurrent Neural Network (RNN) are introduced. RNNs are a type of Neural Network where the output from previous step are fed as input to the current step such as shown in Fig 8.

However RNNs have various problems. They are slow to train and have problems with long sequences due to vanishing/exploding gradients. Due the expansive nature of language, the slowness of training supposes a problem when we want to try using the huge amounts of text and parameters required to model a language. On the other hand, not being able to keep relationships between long sequences does not allow the network to model dependencies between different paragraphs or even sentences.

Long Short Term Memory (LSTM) networks [21] were explicitly designed to remember information for long periods of time and fix the long-term dependency problem. Their shortcoming is that they are even slower to train than a simpler RNN. How to solve this will be tackled on further sections.
A.4 Word Embeddings

We understand the basics of neural networks and we have proposed an architecture capable of learning the sequential nature of language but, how is our model supposed to understand words? A way to represent words as numbers without losing their meaning is required to be able to input them in our deep learning network. Word embeddings are numeric vector inputs that solve this problem.

There are two main approaches to generate word embeddings. The first methods tried a deterministic approach, where the embeddings were created based on different word frequency measurement techniques such as count vectors or TF-IDF. If the method focuses instead on modelling the probability distribution of to the words, it will be a prediction based vector. Prediction based approaches such as Word2vec \[22\] and GloVe \[5\] efficiently create word embeddings while capturing some of the meaning of the words or, at least, the relationship between similar words.

![Figure 9: Simplified representation of word2vec word embedding.](image)

Figure 9: Simplified representation of word2vec word embedding.

Even though word2vec or GloVe achieve pretty good results they doesn’t capture contextual information. Language is really complex and, to create a good embedding, the vector needs to capture pronouns, synonyms, long-term dependencies, etc. A mere hidden layer of a shallow neural network cannot model all this relationships. But, why do not we take word embeddings one step further? Is it possible to pre-train a whole model to learn the fundamentals of language and then fine tune this very model for specific downstream tasks? Once we model the basics of a language is easier to apply this knowledge to different problems. This is the main goal that deep learning in NLP has striven to address in the recent years.

A.5 Language Models

What NLP task captures the fundamentals of language? There are various possible tasks such as machine translation, natural language inference, or reading comprehension but these tasks usually result in the model learning patterns in the data instead of the language itself. The current consensus is that language modelling (LM), where the task is predicting the next word given its previous word, is the best suited task to better capture languages.

![Figure 10: Language Model next word prediction.](image)

Figure 10: Language Model next word prediction.

Once we pre-train a language model we can use transfer learning to fine tune it on a wide variety of NLP tasks. Using pre-trained models makes possible to achieve the same or even better performance much faster and with much less labeled data. The Universal Language
Model Fine-Tuning method (ULMFiT) [23], is likely the first effective approach to fine-tuning the language model. The authors demonstrate the importance of several novel techniques, including discriminative fine-tuning, slanted triangular learning rate, and gradual unfreezing, for retaining previous knowledge and avoiding catastrophic forgetting during fine-tuning.

There have been various breakthrough architectures that have allowed us to implement bigger and better language models. The following sections will review the most important but remember, the focus is to use them to create a LM.

A.6 Embedding from Language Models

Embedding from Language Models (ELMo) [24] managed to create embeddings that preserve context. That means that the same word can have a different vector representation depending of its context. The model is trained on the task of next word prediction. This allows the model to use the vast amounts of unlabelled text data available instead of having to work with small labelled datasets.

ELMo word representations, are generated taking the entire context into consideration. At its core it contains two bi-LSTMs, trained for Language Modelling. In particular, they are created as a weighted sum of the internal states of a deep bi-directional language model using bi-LSTMs, pre-trained on a large text corpus. Furthermore, ELMo representations are based on characters so that the network can understand even out-of-vocabulary tokens unseen in training.

![Figure 11: Single ELMo layer](image)

An input vector is generated for every word that its passed to the first layer capturing the left to right context and then to the second layer, capturing the right to left context. These two outputs are concatenated to get a vector that has bidirectional context knowledge of each word. Two bi-LSTMs such as this one conform ELMo. The final output consists on the weighted summation of two vectors generated by the bi-LSTMs and the original input vector.

As discussed in section A.3, LSTM are really slow to train. The input data needs to be passed sequentially, one word or token at a time while the output suffers the same problem. One way to avoid this bottleneck is trying to make the whole process parallel. This would speed up the training while allowing the use of modern GPUs, which are specifically designed for parallel computation. So the question is, can we parallelize sequential data? In order to answer this, let’s first introduce the concept of attention.

A.7 Attention

Imagine you are preparing for an exam using a book as studying material. You know that the exam will only contain questions described in a specific chapter of the book, how will you study? One option will be reading the whole book, trying to remember everything, while the
more sensible choice would be to just focus on the relevant chapter. This attention mechanisms [25] is quite intuitive for us humans and allows us to achieve faster and more concrete results.

Is possible to achieve the same behaviour with neural networks by focusing on part of a subset of the information they’re given and determining its relevance to the task at hand. In order to make a network able to learn where to focus the tricks is making the network focus everywhere with different intensities, imitating Neural Turing Machines [26].

On a high level, we can define attention as a vector of importance weights. In order to predict a new word, we use this attention vector to estimate how strongly it attends to other elements, taking the sum of their values weighted by the attention vector.

![Alignment matrix of attention weights for machine translation](image1.png)

Figure 12: Alignment matrix of attention weights for machine translation. A darker color represents a higher weight value.

Going back to the sequential nature of language, this may be helpful to better model relationships between words in one sentence or close context as depicted in Fig 13. This example is a variation of attention called self-attention and focuses on learning the correlation between the current words and the previous part of the sentence instead of the relation between input and output. The general mechanism derives from the data-structure of a key-value store where the input is the various values that are then matched to a concrete key by an algorithm (with trainable weights). Afterwards, a different algorithm (that also uses a different set of trainable weights) assigns a query to each output.

![Self-attention conceptual representation](image2.png)

Figure 13: Self-attention conceptual representation. “Strong” is directly related to “Rakija” while "He" has little to do in the current context with it.
This mechanism was applied to Sequence-to-sequence models [27], which are deep learning models take a sequence of items and output another sequence. Instead of directly feeding the output sequence of the previous layer directly to the next, an attention mechanism that decides which elements are relevant for each particular output. This obtained great results in tasks such as machine translation and text summarization.

A.8 Transformer

The great breakthrough derived from self-attention is that attention, by itself, is a strong enough mechanism to do all the learning. The famous paper Attention is All You Need [28] introduced the architecture and broke away from previous RNN focused approaches. The transformer expands of the encoder decoder architecture used in Sequence-to-sequence models stripping it and leaving the learning to the Attention mechanism.

It manages to outperform the previous approach in various tasks and speeds up the training process allowing parallelization, solving one of the biggest problems of previous sequential approaches. Self-attention works by having key, query and value be all the same vectors. They attend to themselves and the non linearity provided by stacking them provides enough capacity to fit complex functions.

The Transformer consist on a stack of identical encoders connected to a stack of identical decoders (the Attention is All You Need paper used 6 of each but the number is not set in stone).

The encoder stack receives the input sequence, where is mapped to a single latent vector that represents the whole sequence. Afterwards, the latent vector is then passed to the decoder stack outputting the desired target sequence.

![Figure 14: Simplified Transformer architecture.](image)

The input is passed to the encoder stack where it flows through a self-attention layer. This takes context into account allowing the encoder to attend to other words of the sentence while in encodes each specific word. Afterwards the self-attention outputs are passed through a feed-forward layer. The decoders contain both layers plus another attention layer in between that focuses on the relevant part of the input sentence. Is important to note that even though each encoder and decoder are identical, they all learn their own set of weights.
This solves some of the problems that were holding ELMo back. The goal now is to use this Transformer as a pre-trained model and, in order to do that, we need to train a language model. A solution was proposed by OpenAI [29]. By removing the decoders and stacking the decoders we achieve a deep network were is possible to pass the input words in parallel a have the decoder output the next word successfully training a language model on the next sentence prediction task.

The core of each decoder is a masked attention unit. Is called masked attention because it obfuscates some of the words of the sentence during the training not paying attention to them while generating the next word. In the OpenAI Transformer only the left to right context is taken into account, however, the actual context of a word is bidirectional. The next architecture solves this problem.

A.9 BERT

Even the word embeddings produced by ELMo, which boasted of bi-directionality, had it left-to-right and right-to-left context were determined separately and then concatenated. The next step is to be able to get context from both directions simultaneously.

This was achieved by BERT, which took the leader-boards of the field by storm and showed that transformers were able to reach human-level performance on a variety of language based tasks.

The network structure of BERT and the previously discussed Transformer are identical, with BERT being a stack of transformer blocks. This is then pre-trained on a massive general-domain corpus consisting of English books and Wikipedia articles. The pre-training can be separated into two different tasks: masked language modelling and next sentence prediction.
After tokenization, a certain number of words in the input sequence can be either, replaced with another random word, masked using a special <mask> token or kept without change. Then, the model is tasked with predicting them. This is a variation on the next word prediction task and allows the model to use context from both directions simultaneously. Note that the model doesn’t know which words it will be tasked to predict, so it learns a representation for every input word.

For next sentence classification there are two possible cases: either both follow each other directly in the corpus or they were taken randomly and don’t follow each other. This allows the model to learn long-term relationships between sequences.

In order to complete this tasks and to further improve the performance of the model BERT tokenizes the input sequence following some considerations. <CLS> and <SEP> tokens are added at the beginning and end of the input to allow for the next sentence classification task as well as other sequence classification task that the model could be requested to perform. The aforementioned <MASK> token is also introduced here. WordPiece tokenization is utilized to allow the model to make some inferences based on word structure and to alleviate the problem of out-of-vocabulary (OOV) words. This works between word-level and character level sequences where the words are broken into smaller tokens. A verb such as singing might be split in “sing” and “##ing”.

Once the pre-training is done, a final task-specific layer can be added to the end of the network in order to get the proper output for the task at hand. If we are working with a classification task, a simple softmax layer that maps the classes probabilities of the first output token is enough. In the BERT paper, other more complex layers were introduced for more specific tasks. Finally the model is trained again in the task dataset in a process we call fine tuning. Another option is to simply use the word embeddings generated by the pre-trained BERT model as part of our machine learning pipeline. The authors of the paper presented one ablation study of different features and hyper-parameters used in BERT, facilitating the adoption of the model to other researchers.

One of the most important results is that the largest margin of improvement compared to previous models comes from BERT’s bidirectional nature. Despite that, being a deep learning architecture, is difficult to really understand what does BERT exactly learns [30]. It is also worth of noting that with BERT we are stepping in the realm of huge models. The large BERT model has 340M parameters, product of 24 transformer blocks, an embedding dimension of 1024 and 16 attention heads.

A.10 The State of the Art after BERT

The transformer architecture and pre-trained models have become the norm in recent years. Being able to leverage huge unlabelled text corpus in what we know call self-supervised learning
allow us to, in part, forget about the lack of labelled text data and focus on training bigger and better models.

The masked language modelling used in BERT is further improved in RoBERTa [17], where the masking process is made dynamic. On the other hand the huge size of the models such as as BERT supposes a problem when deploying it to production in high demand, low latency tasks. A proposed solution is Distillation [31] [32] where the model is compressed while retaining most of its performance. DistilBERT [16] is a prime example of this technique.

Parallel to BERT, openAI continued working on generative pre-trained transformer releasing GPT-2[33]. One of the most interesting tricks used in this new model was the creation of a new dataset. Instead of using curated Wikipedia articles and well edited books, the social media site Reddit was used to capture a more diverse and "real" dataset. It used Reddit’s karma as metric for "quality" content. On the other hand, the recent publication of GPT-3 [34] tries to demonstrate that a language model trained on enough data can solve NLP tasks that it has never seen. That is, GPT-3 studies the model as a general solution for many downstream jobs without fine-tuning.

XLNet: Generalized Autoregressive Pretraining for Language Understanding [35] solves some of BERT problems that appear when the masked words are dependent on each other, resulting on low quality embeddings. It introduces Permutation Language Modeling which is used instead of masking and uses many more predictions per sentence. This is used in tandem with relative position embeddings to improve on BERT. It is important to take into account that the published XLNet model also uses more data and bigger models, but the paper showed that its innovations improved on BERT even when using equivalent data and model size.

ELECTRA [18] was recently presented by Google to tackle the cost and difficulty of pre-training bigger and bigger models. ELECTRA trains the model to discriminate locally plausible text from real text but finds it difficult to match SOTA results with less compute.

The current trend seems to mostly be to train bigger, more expensive models while using more data. This makes it hard for researchers to compete with big tech corporations on the leader boards. There is still a lot of room to growth and only time will tell which direction does the field take.

A.11 Conclusion

The transformer is one of the simplest machine learning architectures to dominate the field in decades and the path taken to arrive to it is filled with interesting breakthroughs and research. Pre-trained models have resulted in a renowned interest in NLP for developers and researchers and never has it been so easy to develop a working pipeline for related tasks.

The current performance limit is purely limited by hardware and time. Bigger and better models keep appearing and only how big of a model we can fit in GPU memory with a reasonable training time seems to be the limiting factor. Even though some concerns have been raised about the astronomic price of training new models (The Cost of Training NLP Models) [36]
the benefits still out value the costs and we can expect to keep seeing improvements in this
directions in the recent future.

Understanding the building blocks and the concepts that gave rise to this new era would
become a great tool for anyone working on the NLP field. Further research is surely required
and encouraged by the author if one wants to achieve more than a superficial understanding
but text this should give you a good foundation to build your research on.