



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



Escola Tècnica
Superior d'Enginyeria
Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Application of Machine Learning Techniques for Text Generation

DEGREE FINAL WORK

Degree in Computer Engineering

Author: Salvador Martí Román

Tutor: Lluís Felip Hurtado
Ferran Pla Santamaría

Course 2019-2020

Resum

Els avanços en l'àrea del processament del llenguatge natural i l'aprenentatge automàtic permeten l'anàlisi, la comprensió i la generació de textos de forma cada vegada més precisa. Aquest treball de final de grau es marca com a objectiu la generació de text que tracte de simular l'estil d'un membre del parlament de Regne Unit. Per a això s'haurà de recopilar i analitzar les transcripcions dels debats en el parlament anglès des de 2016. A partir d'aquestes dades s'entrenarà un model estadístic que genere text imitant l'estil dels membres del parlament més rellevants. Finalment, es realitzarà una avaluació a diferents nivells dels textos generats.

Paraules clau: Aprenentatge Automàtic, Aprenentatge Profund, Processament del Llenguatge Natural, Generació de Textos

Resumen

Los avances en el área del procesamiento del lenguaje natural y el aprendizaje automático permiten el análisis, la comprensión y la generación de textos de manera cada vez más precisa. Este trabajo final de grado se marca como objetivo la generación de texto que intente simular el estilo de algún miembro del parlamento de Reino Unido. Para ello se deberá recopilar y analizar las transcripciones de los debates en el parlamento inglés desde 2016. A partir de estos datos se entrenará un modelo estadístico que genere texto imitando el estilo de los miembros del parlamento más relevantes. Finalmente, se realizará una evaluación a diferentes niveles de los textos generados.

Palabras clave: Aprendizaje Automático, Aprendizaje Profundo, Procesamiento del Lenguaje Natural, Generación de Texto

Abstract

Advances in the field of natural language processing have led to the ever increasing precision of automatic text analysis, understanding and generation processes. This work has as an objective the generation of text in the style of speakers in the United Kingdom's houses. To this end debate transcripts from 2016 onward will be collected and analyzed. An statistical model will then be trained from the resulting text corpus that will generate text in the style of different speakers. To conclude, the generated texts then be evaluated with several metrics.

Key words: Machine Learning, Deep Learning, Natural Language Processing, Text Generation

Contents

Contents	v
List of Figures	vii
List of Tables	vii

1 Introduction	1
1.1 Motivation	1
1.2 Objectives	1
1.3 Essay structure	2
2 Theoretical background	5
2.1 Machine Learning in Natural Language Processing (NLP)	5
2.1.1 Pattern recognition and machine learning	5
2.1.2 Classification and regression	5
2.1.3 Gradient and optimizers	6
2.1.4 Artificial Neural Networks (ANN)	8
2.1.5 Backpropagation	9
2.1.6 Recurrent Neural Networks (RNN)	10
2.1.7 Convolutional Neural Networks (CNN)	12
2.2 Local and Distributed Representations in NLP	14
2.2.1 Local Representation: Bag of words (BOW)	14
2.2.2 Distributed Representations: Word2Vec	15
2.2.3 Sub-word Representations: Byte Pair Encodings (BPE)	16
2.3 Language Models (LM)	17
2.3.1 N-gram based Language Models	17
2.3.2 Feed forward Neural language models	18
2.3.3 Recurrent Neural Network Language Models	19
2.3.4 Convolutional Language Models	19
2.3.5 Language Model Sampling	21
2.4 State of the art results: Transformers	23
2.4.1 Attention is all you need	23
2.4.2 Encoder blocks	24
2.4.3 Decoder blocks	25
2.4.4 Transformer scaling	25
2.4.5 Generative Pretrained Transformer 2: (GPT-2)	26
2.4.6 BERT and RoBERTa	28
3 Design	31
3.1 Chosen approach	31
3.2 Compute Resources	32
3.3 Tensor Processing Units (TPU) for Deep Learning	32
3.4 Environment	33
4 Case Study	37
4.1 Corpus creation: cleaning and web-scraping	37
4.1.1 Source Data	37

4.1.2	Legality	38
4.1.3	Environment and technologies	38
4.1.4	Webcrawling	38
4.1.5	Corpus preparation	39
4.1.6	Text pre-processing	40
4.2	Testing methodology.	40
4.3	Results and evaluation	43
4.4	Costs and efficiency	44
5	Relation with studies	49
6	Conclusions	51
6.1	Further work	51
	Bibliography	53

Appendix		
A	Additional Language Model Samples	57

List of Figures

2.1	Step by step optimisation of a weight for a given gradient [19].	7
2.2	Depiction of an Artificial Neuron.	9
2.3	Unfolded recurrent neural network.	11
2.4	A Sequence to Sequence architecture. [34]	12
2.5	Two dimensional convolution with kernel size 2 and windows placed every 1 step.	12
2.6	Effect of pooling on feature dimensions.	13
2.7	Word2Vec architectures [30]	15
2.8	Vector operations with word2vec embeddings. [30]	16
2.9	Diagram of model proposed in Neural Probabilistic Language Models paper. [32]	18
2.10	Simplified Recurrent Language model diagram. [34].	19
2.11	Illustration of a standard convolution (top) versus causal convolution (bottom)[36].	20
2.12	Gated convolutional language model architecture. [35].	21
2.13	Example of broad and narrow distributions. Using $k = 3$ would perform favourably in the narrow distribution but limit the range of options in the broad one. [37]	22
2.14	Illustration of the attention concept in neural networks. Line thickness and darkness indicate more importance.	23
2.15	Transformer encoder block. [39]	24
2.16	The Transformer - model architecture. Encoder to the left, decoder to the right. [38]	25
2.17	GPT-2 released model architectures.[42]	27
2.18	Bert masking prediction.	28
4.1	Search Request	37
4.2	In webtext the detection accuracy becomes higher for longer text. The figure also shows that training on random-length training examples has significant positive effect on the accuracy for short length texts[56]	43
4.3	1.5B GPT-2 model TPU fine-tuning on Hansard Transcripts. Loss over time.	44
4.4	Cloud TPU diagnostic tools. [55]	46

List of Tables

2.1	Trace of 5 iterations of the BPE algorithm.	16
2.2	Comparison of layer properties. [38]	26
2.3	GPT-2 zero shot translations quoted in Language Models are Unsupervised Multitask Learners[1]	27

4.1	RoBERTa large accuracy with respect to three sampling methods (Temperature = 1 no truncation, Top-K = 40, and nucleus sampling with the Top-P sampled uniformly between 0.8 and 1.0). Subsection corresponding to training with GPT-2 1.5B parameter model outputs. The accuracy is obtained by testing 510-token test examples comprised of 5,000 samples from the WebText dataset and 5,000 samples generated by a GPT-2 model, which were not used during the training. [56]	41
4.2	Comparison between real and model generated responses for a given prompt. See Appendix A for more model output examples.	42
4.3	Fine-tuned GPT-2 1.5B detection rate on publicly available pre-trained GPT-2 output detectors	43
4.4	Potential costs of replicating the fine-tuning of the 1.5B GPT-2 model using TPUs in GCP.	45
4.5	Potential costs of replicating the fine-tuning of the 1.5B GPT-2 model using GPUs in GCP.	46
4.6	Potential costs of replicating the fine-tuning of the 1.5B GPT-2 model using GPUs in AWS.	47
A.1	Comparison between real and model generated responses for a given prompt.	57
A.2	Comparison between real and model generated responses for a given prompt.	58
A.3	Comparison between real and model generated responses for a given prompt.	59
A.4	Comparison between real and model generated responses for a given prompt.	60

CHAPTER 1

Introduction

1.1 Motivation

The generation of coherent language even in text form has proven to be a challenging task. Thanks to recent advances we might soon see applications that rely on custom text generation come to life. The recent raise in popularity of massive language models, i.e statistical models pre-trained with extremely large generic corpora like wikipedia, and their public availability is an extremely important event. Applications of text natural language processing (NLP) can be rapidly experimented on without the need of weeks worth of training or the allocation extremely high amounts compute resources. This lowered barrier to entry means research groups with lower amounts of resources development of NLP related functionality by fine-tuning a pre-trained model to make is suitable for a given task.

The implications of these advances are still largely unknown. Being able to produce convincingly human-like works of text can prove to be useful in the form of chatbots or extremely detrimental in the form of automated generation of misinformation.

Before releasing their new language model architecture [1] openAI[2] documented their concerns in their article “Better Language Models and their Implications”[3] and temporarily refrained from publicly releasing the full-scale 1.5 billion parameter pre-trained model until further research into the effects of such models not only in the field of artificial intelligence but also in the context of lawmaking.

Since that statement, a number of reasonably competent detectors using similar architectures have been created and the generator model has been released. In one such solution created by openAI, they encouraged research on the impacts of detection rates in models fine-tuned with different datasets[4]. These detectors in the past have mostly been evaluated in best case scenarios where training and evaluation datasets both originated from the exact same model. The concern is that slightly modified model weights may render the detectors ineffective.

1.2 Objectives

As a follow up to these concerns this work aims to assess the accuracy of the GPT-2 output detectors released by open-AI for a fine-tuned model.

The specific objectives of this work will be the creation of:

1. A consolidated corpus composed by the United Kingdom's post-2016 house of commons transcripts. Instructions on how these texts are collected and processed will be included in this same work.
2. A version of the GPT-2 1.5B parameter model fine-tuned with the political corpus collected. The training of such a large model can be costly in terms of both time and monetary resources. This model could be used to improve future performance of detectors for purely political texts. The model weights will be made available upon request.
3. A short evaluation comparing the performance of detectors prior to being fine-tuned against prompted outputs from the model. The real responses will also be tested to assess the false positive rate.

1.3 Essay structure

Chapter 2: Theoretical Background

Section 2.1, *Machine Learning in Natural Language Processing (NLP)* begins laying the groundwork for basic machine learning concepts used in the evolution of natural language processing such as gradients, optimisers and neural networks. The topics covered in this introduction to the field are not only important to understanding the evolution of language models but also form part of the state of the art techniques that will be covered later on.

Section 2.2, *Local and Distributed Representations in NLP* focuses on the evolution of text transformations for computer understanding and how problems of limited vocabulary can be mitigated through algorithms like BPE. These representations are most essential for use in any kind of machine learning application.

Section 2.3, *Language Models (LM)* covers the progression of models whose objective is to comprehend the underlying structures and meanings of text. Several types of language models are covered in chronological order of invention and compared. At the end of the chapter a small section is dedicated to techniques for better generation of text, one of the possible downstream tasks of a language model.

Section 2.4, *The Transformer* dives into the current state of the art architecture commencing by the paper that started it all *Attention is all you need*. The intuition behind the attention concept is explained and is used as a stepping stone to cover the basic building blocks that compose the different variations of the transformer. The inner workings of the GPT-2 and BERT architectures, the transformers used in the experimental component of this work, are compared with each other and their predecessors.

Chapter 3: Design

Section 3.1, *Chosen approach* introduces how a large model like the 1.5B parameter version can be trained and served using cloud resources.

Section 3.2, *Compute Resources* describes the resources that were used to train the generator model.

Section 3.3, *Tensor Processing Units (TPU) for Deep Learning* explains what tensor processing units are and how TPU and GPU computing differ.

Section 3.4, *Environment* does a breakdown of all the software used during the training of both the generators and detectors.

Chapter 4: Case-study

Section 4.1, *Corpus creation: cleaning and web-scraping* details the techniques used to extract data from crawling through websites when the information is not directly obtainable from its source code. The specific process and requests used to collect the House-of-commons corpus are detailed. In addition this chapter includes steps on how to adequately prepare a GPT-2 input for training.

Section 4.2, *Testing Methodology* sets down the procedure used to evaluate the detectors including the parameters and sampling methods used.

Section 4.3, *Results and evaluation* assesses the performance of the detection models used and compares the results with other studies done by the creators.

Section 4.4, *Costs and efficiency* provides a breakdown of the costs of fine-tuning the GPT-2 model on TPUs. Inefficiencies in the code implementation are explained. A basic cost comparison is made with GPUs taking into account the pricing of two cloud compute providers.

Chapter 5: Relation with studies

This chapter names specific degree subjects relevant to the topics covered in this work as well as some of the transversal competencies that played a role in its making.

Chapter 6: Conclusions

In this chapter conclusions are drawn on the viability of the detectors as a tool for the automatic detection of computer generated political discourse. Additionally, number of possible follow-up questions resulting from this work's findings are posed for further study.

CHAPTER 2

Theoretical background

2.1 Machine Learning in Natural Language Processing (NLP)

2.1.1. Pattern recognition and machine learning

Pattern recognition is the field “concerned with the automatic discovery of regularities in data through the use of computer algorithms” (p.21) [15] and the use those patterns to take different actions like estimation or classification of data. This can be through explicitly written instructions or the use of machine learning (ML) techniques.

The process of learning in ML can be broadly described as occurring whenever a model “changes its structure, program, or data (based on its inputs or in response to external information) in such a manner that its expected future performance improves” [16]. Due to similarities in the objectives of both areas, pattern recognition and ML are often regarded “as two facets of the same field” (p.7)[15]. The resulting combined field’s objective is the automated taking of decisions that maximize or minimize a defined utility function.

Through the decades the combined fields have surpassed a large variety of ground-breaking milestones ranging from self-driving cars[17] to generation of machine designed art [18]. NLP has in grand part, but not exclusively, been forwarded by advancements in ML techniques and hence it is necessary to introduce a number of common machine learning concepts to adequately explain the current state of the field.

2.1.2. Classification and regression

The process of learning revolves around the estimation of a function f given a value or a set of features x . A simple example of this would be estimating the gradient m of a linear function where $f(x) = mx$ so as to minimize a given cost function C that is calculated using the output of $f(x)$. In a supervised regression problem a set of features X and true value Y pairs are used to minimize the error for the cost function by changing $f(x)$.

$$y' = f(m, x) = mx$$

$$C(y', y) = (y' - y)^2$$

The best value for m is the one that minimizes the sum of the cost function for N number of x, y value pairs.

$$m^* = \operatorname{argmin}_m \sum_{i=1}^{N+1} C(f(m, x_i), y_i)$$

This value m of our simple linear regressor is called a weight. The weights of a ML model are those values in f that are modified in order to improve how the function models the

relationship between the x, y value pairs. These weights are modified using an optimization algorithm like gradient descent using a learning rate α to converge to a local minimum or maximum for the given cost/loss function.

At its most basic gradient descent is an algorithm that computes the derivative of a cost function given a set of weights and modifies the weights using a step of size α in a downwards trajectory so as to minimize the original function.

In a classification problem the objective is to predict the correct label y given a set of values X . Unlike regression, labels in classification problems are discrete and often non-numerical. As a result, it is first necessary to encode classes into vectors using one hot encoding. E.g when we have two classes (Tall, Small) they can be encoded as $[1, 0]$ and $[0, 1]$ respectively.

This can now be split into two simpler classifications: The likelihood of X being Tall p_1 and the likelihood of X being Small p_2 where $p_1, p_2 \in [0, 1]$.

The class chosen is the one with the highest activation, i.e output value, of the two. In this case, the cost function can be calculated using categorical cross entropy for the model which is defined as such:

$$Y = (y_1, \dots, y_n) : y_i \in \{0, 1\}, \forall i \in \{1, \dots, n\}$$

$$Y' = (y'_1, \dots, y'_n) : y'_i \in \{0, 1\}, \forall i \in \{1, \dots, n\}$$

$$C(Y', Y) = - \sum_{i=1}^n y_i * \log y'_i$$

Using this modified cost function or one similar facilitates the usage of an optimizer like gradient descent to update the weights with the objective of reducing the cost function for subsequent examples.

Quite often, the softmax function is applied to the outputs of a classification model to transform them into a probability distribution of the same size. This not only scales the output of the model between 0 and 1 aiding in the learning of the class representation but also makes the output far more readable as each node corresponds to the estimated probability of the class for a given input. The softmax function for a given output y'_i is calculated as follows:

$$y'_i = \text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_j^{j=n} e^{x_j}}$$

The cost function C may now be used compare the costs between the probability distribution and the one-hot label as detailed above.

2.1.3. Gradient and optimizers

Machine learning models are mostly used in non-convex optimization problems. These present a number of issues due to the nature of the resulting gradients. The presence of local minima and maxima makes training the weights significantly more challenging.

Finding the global optimum of the weights that minimize the cost function of a complex model like a neural network has been shown to be an NP-complete problem. [21]

It is therefore why optimizers in machine learning revolve around converging to local minimums while balancing exploration of new parts of the gradient that might lead to an even lower cost. This poses a number of issues to overcome. Notably:

- Failure to converge
- High training cost
- Low performance

Over time a number of different optimizers and techniques have been applied to alleviate some of the problems suffered but for simplicity we choose to only cover a simple version of the stochastic gradient descent algorithm (SGD).

For a model function f and a cost function C defined as:

$$y' = f(m, x) = mx$$

$$C(y', y) = (y' - y)^2$$

SGD calculates the derivative of the cost function for a single data point as follows:

$$\frac{\delta C}{\delta m} = 2 * (y' - y) * \frac{\delta}{\delta m} (y' - y)$$

$$= 2 * (y' - y) * \frac{\delta}{\delta m} (mx - y)$$

As the data point X and label Y are constants the resulting gradient is:

$$\frac{\delta}{\delta m} = 2x$$

Once this is calculated the direction of the step to be taken is known and the magnitude can be adjusted using a learning rate hyper-parameter commonly referred to as α .

$$m^{i+1} = m^i - 2 * (y' - y) * x * \alpha$$

The new model weight m is then used with the next set of values. Iterating on these calculations takes small steps towards a local minimum.

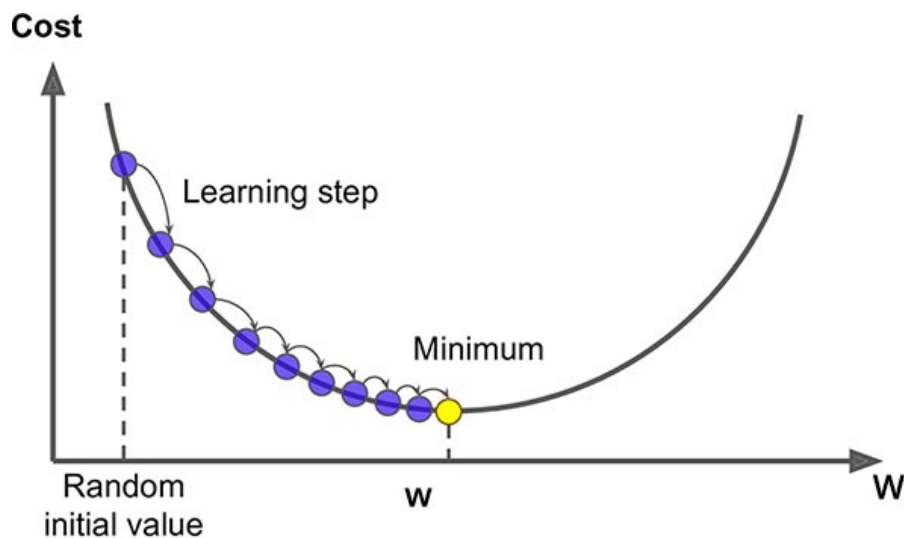


Figure 2.1: Step by step optimisation of a weight for a given gradient [19].

In most real world applications, several weights are used to increase the expressiveness of the model. Partial derivatives are used to update additional weights to be considered and to update them separately, e.g:

$$y' = f(m, x) = mx + b$$

$$C(y', y) = (y' - y)^2$$

$$\begin{aligned}\frac{\delta C}{\delta m} &= 2 * (y' - y) * \frac{\delta}{\delta m}(y' - y) \\ \frac{\delta C}{\delta m} &= 2 * (y' - y) * \frac{\delta}{\delta m}(mx + b - y)\end{aligned}$$

$$\begin{aligned}\frac{\delta C}{\delta b} &= 2 * (y' - y) * \frac{\delta}{\delta b}(y' - y) \\ \frac{\delta C}{\delta b} &= 2 * (y' - y) * \frac{\delta}{\delta b}(mx + b - y)\end{aligned}$$

$$\begin{aligned}\frac{\delta}{\delta m}(y' - y) &= 2x \\ \frac{\delta}{\delta b}(y' - y) &= 1\end{aligned}$$

$$\begin{aligned}m^{i+1} &= m^i - 2 * (y' - y) * x * \alpha \\ b^{i+1} &= b^i - (y' - y) * \alpha\end{aligned}$$

A number of variants of this algorithm exist that increase speed and stability of training by adding features like momentum, clipping and accumulation of costs along several data points. Some notable examples of algorithms that include some of these features are batch gradient descent [22], Adam [23] and RMSprop [24].

2.1.4. Artificial Neural Networks (ANN)

Neurons in artificial neural networks are in many ways similar to the linear regressor introduced in the previous section. Both techniques multiply their input by a weight and optionally add a bias to produce an output [25]. The main distinguisher is that neurons are organized in layers and those layers can be stacked in a variety of ways to increase the expressive power of the model.

The input of a neuron in a standard multi-layered perceptron is the sum of the outputs of all the neurons in the previous layer. This allows neural networks to express non-linear discriminant functions and, as a result, to model more complex systems. These outputs are often fed through an activation function G to solve problems with exploding/vanishing gradients and accelerate the speed of convergence.

The output of a neuron j in a fully connected layer L_i takes all N_{i-1} outputs from the previous layer L_{i-1} .

$$L_{i-1} = (x_{i-1,1}, \dots, x_{i-1,N_{i-1}}) : x_{i-1,k} \in \mathbb{R} \forall k \in \{1, \dots, N_{i-1}\}$$

The weights W_i and bias b_i connecting the neurons in two fully connected layers are defined as follows:

$$\begin{aligned}W_i &= (w_{i,1,1}, \dots, w_{i,N_{i-1},N_i}) : w_{i,j,k} \in \mathbb{R}, \forall j \in \{1, \dots, N_i\}, \forall k \in \{1, \dots, N_{i-1}\} \\ b_i &\in \mathbb{R}\end{aligned}$$

The vectorial representation of all weights connected to the neuron in layer i in position j is defined as:

$$W_{i,j} = (x_{i,j,1}, \dots, x_{i,j,N_{i-1}}) : x_{i,j,k} = L_{i-1,k} * w_{i,j,k} \forall k \in \{1, \dots, N_{i-1}\}$$

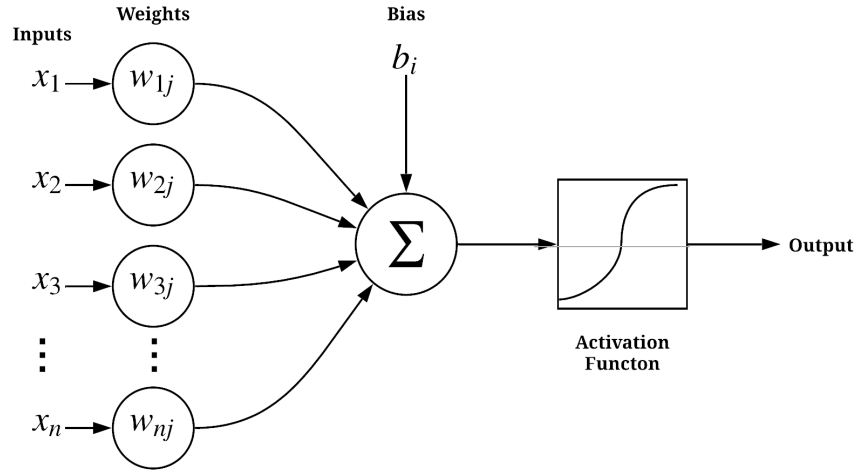


Figure 2.2: Depiction of an Artificial Neuron.

$$X_i = (x_{i,1}, \dots, x_{i,N_i}) : x_{i,j} = \left(\sum_{k=1}^{N_{i-1}} W_{i,j,k} \right) + b_i \quad \forall j \in \{1, \dots, N_i\}$$

For this sample layer G will be the commonly used sigmoid activation function defined as:

$$G(x) = \frac{1}{1 + e^{-x}}$$

As such, the outputs for L_i can be generalised as:

$$L_i = (x_{i,1}, \dots, x_{i,N_i}) : x_{i,j} = G(X_{i,j}) \quad \forall j \in \{1, \dots, N_i\}$$

These series of operations constitute only the feed-forward part of the model which is in charge of inference. Using the outputs and a given cost function a gradient can be computed with the objective of updating the weights of the network in a similar way to what is done with simpler models. Commonly referred to as back-propagation, this approach is quite computationally intensive as it requires the calculation of the partial derivative for every single weight in the network.

This step is why while the concept of an artificial neural network dates back to 1958[26] with the creation of single layer perceptrons the idea wouldn't popularize until much later when backpropagation of errors through the network, first introduced in 1974[27], became much more feasible thanks to improvements in computational technology.

2.1.5. Backpropagation

This approach propagates the errors of the output layer backwards through the network. The backwards pass begins by calculating the partial derivatives of the errors for each output neuron.

The cost function C previously used is calculated for every single node in the output layer. The cost for a given sample C_s is defined as:

$$C_s = (C_{s_1}, \dots, C_{s_{N_i}}) : C_{s_j} = (L_{i,j} - y_j)^2, \quad \forall j \in \{1, \dots, N_i\}$$

The computed loss using a cost function C for a given training sample S is defined as the sum of all the square differences C_s

$$\frac{\delta C_{s_j}}{\delta w_{i,j,k}} = \left(\frac{\delta C_{s_j}}{\delta L_{i,j}} \right) \left(\frac{\delta L_{i,j}}{\delta X_{i,j}} \right) \left(\frac{\delta X_{i,j}}{\delta w_{i,j,k}} \right)$$

If the weight for which the partial derivative is being calculated is in the last layer L_m then the gradient for the loss with respect to that weight only takes into account itself. However if the weight is not in the output layer it is first necessary to calculate the effect the weights in the later layers have on the loss function. In this model layers are fully connected. As such, a node must sum the effects of all the nodes for which it is responsible.

$$\frac{\delta C_{s_j}}{\delta w_{i,j,k}} = \{2(L_{i,j} - y_j) \text{ if } i = m, \sum_{k=1}^{N_{i+1}} \left(\frac{\delta C_{s_j}}{\delta L_{i+1,k}} \right) \left(\frac{\delta L_{i+1,k}}{\delta X_{i+1,k}} \right) \left(\frac{\delta X_{i+1,k}}{\delta w_{i,j}} \right) \text{ if } i = m - 1\}$$

$$\frac{\delta X_{i,j}}{\delta w_{i-1,k}} = w_{i,j,k}$$

$$\frac{\delta L_{i+1,k}}{\delta X_{i+1,k}} = g(X_{i,j})(1 - g(X_{i,j}))$$

$$\frac{\delta X_{i,j}}{\delta w_{i,j,k}} = L_{i-1,k}$$

$$\frac{\delta C_{s_j}}{\delta w_{i,j,k}} = 2(L_{i,j} - y_j)g(X_{i,j})(1 - g(X_{i,j}))L_{i-1,k}$$

2.1.6. Recurrent Neural Networks (RNN)

Recurrent Neural Networks are in many ways similar to standard ANNs with one small architectural change that allows them to propagate its values through inputs.

As detailed in section 2.1.4 cells in a standard ANN only take its values from the previous layers. As such, the output is only ever a function of the current forward-pass input. This comes with a several limitations. One of the main limitations, being the network's inability to deal with variable length inputs and outputs.

In RNNs this limitation is solved by propagating network values through forward-passes. This propagation through time means that the output of a recurrent cell is not only a function of the values provided by it at a given time t but also of its own state at a time $t - i$ where i corresponds to the amount of passes the connection bridges.

$$y'_t = f_t(x, y'_{t-i})$$

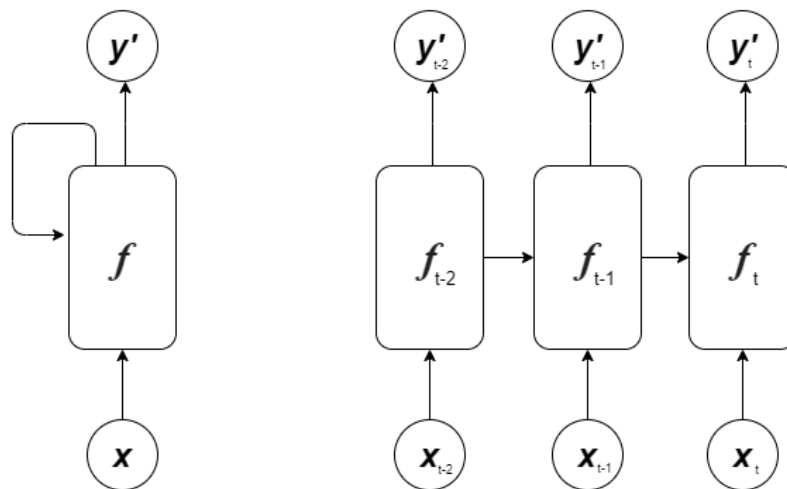


Figure 2.3: Unfolded recurrent neural network.

As the output of a cell changes based on its previous forward passes it is necessary to take into consideration how the network outputs affect themselves through time at the back-propagation stage. While it is still possible to generate an output at every forward-pass of the network, back-propagation and weight updating in most cases only occurs at the end when the complete result is obtained. For exceptionally long sequences, this approach can prove to be too memory intensive and fail to converge. As a result these sequences may be broken down into truncated sections at the end of which the network's weights are updated but the states are still carried through to the next piece.

A major computational drawback of recurrent models is that the loss for the hidden states must be calculated sequentially as there's a dependency through time t . Modern hardware like GPUs or TPUs typically excel at neural network training because of their vast multiprocessing capabilities. This lack of parallelism makes RNN backpropagation significantly slower.

Another issue inherent to recurrent units are exploding and vanishing gradients [29]. This gradient instability is caused by the repeated multiplication of the network's weight matrix W at every single step t in the backpropagation stage. These issues may prevent the models from learning relevant information over long training sequences.

Refinements over time led to the creation of long short term memory (LSTM) Networks in 1997[28] which included several connections that allowed models to retain fading long term dependencies for longer. These LSTMs featured the addition of several gates that control how the flow of information through time.

The first one, often referred to as the forget gate decides which information to discard by taking into consideration y_{t-1} and x_t . A second gate called the input gate takes the same values and considers which candidates are worth updating. These modifications are then used to update the cell state. The main advantage of these cells is that the forgetting and replacing of previous states can be trained to the task hence learning what to forget and what to remember.

The multi-input and multi-output capability of RNNs made encoder (many-to-one) and decoder (one to many) architectures possible. When both were combined the result was the birth of the sequence-to-sequence model architecture. This many-to-many model type is essential to many NLP tasks including, but not limited to, machine translation and text generation.

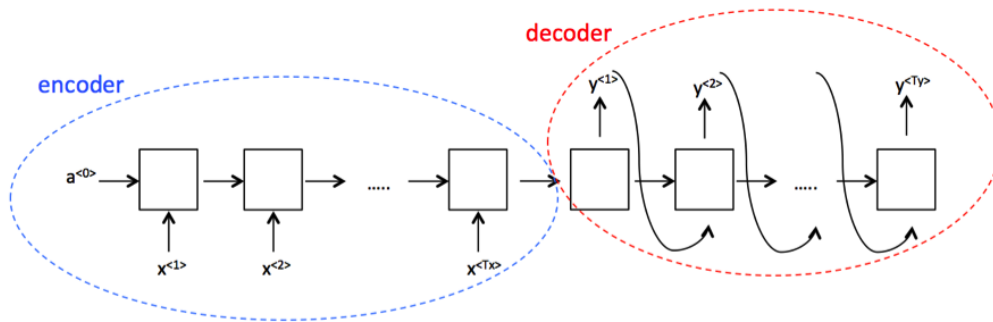


Figure 2.4: A Sequence to Sequence architecture. [34]

2.1.7. Convolutional Neural Networks (CNN)

CNNs are most frequently used in the realm of computer vision where 2 dimensional convolutions are applied to a pixel array. A convolution in machine learning is the application of a filtering function f over a series of activations given by another function g . The filters of size h are applied to a series of windows of the same size. These windows are subsections of the input shifted by a given amount for every time step until all the input is represented. The parameters of the filtering function are learnt through back-propagation of errors as any other weight in a neural network. The next figure illustrates how 2D convolutions work:

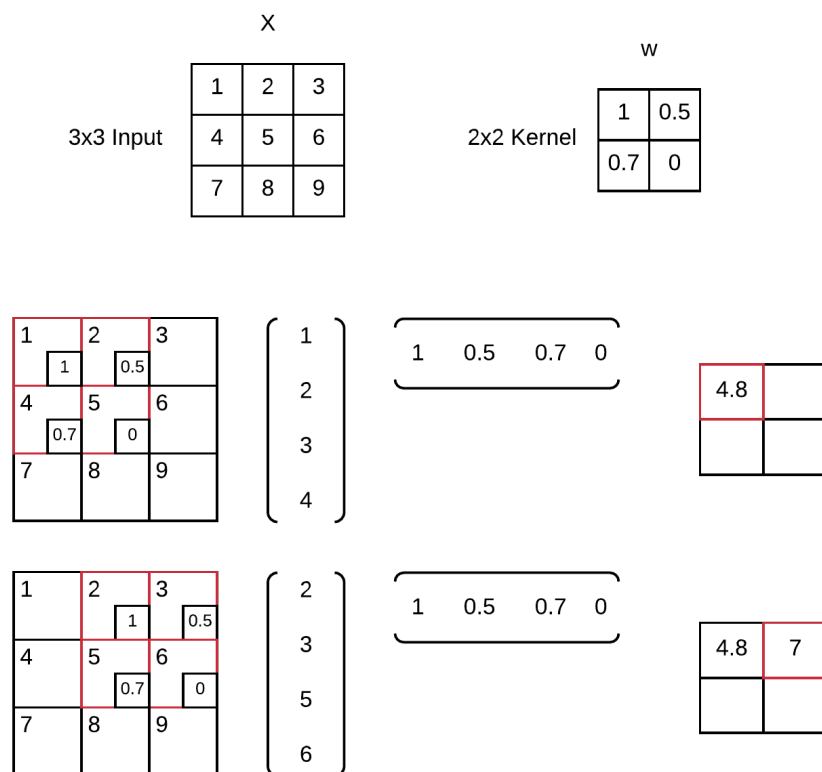


Figure 2.5: Two dimensional convolution with kernel size 2 and windows placed every 1 step.

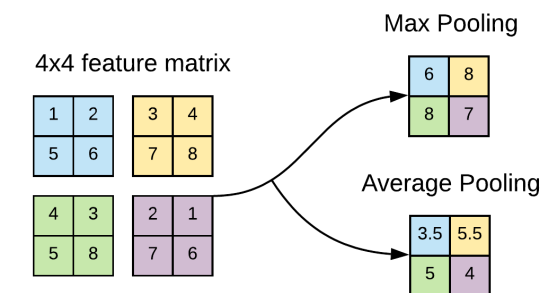
As seen in Figure 2.5, for a given window the dot product between the context of the sub-matrix and the filter is calculated. The output of this calculation is the similarity between the two vectors. Kernel weights are learnt so they may specialise on picking up certain local characteristics of the data. As the same kernel weights w are used for all windows for a given input size n and filter size h , each kernel produces a feature vector as follows where g is an activation function:

$$c = \{c_1, \dots, c_{n-h+1}\} : c_i = g(w^T x_{i:i+h-1} + b)$$

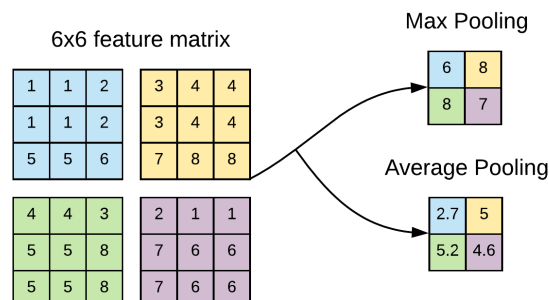
Each kernel used produces a separate feature vector c . Different filters may have differently sized windows even within the same layer. This helps to capture a wide range of different features. Using these kernels as they are provides information on each of the windows taking into account their position. This approach restricts sequence length which is problematic when dealing with inputs like sentences. To solve this flaw we use a technique called pooling.

Several kinds of pooling exist but the two most used are max and average pooling. As the name suggests, max pooling takes the feature of maximum value while average pooling conducts the mean of the features. The choice of which one to use has special repercussions for gradient flow at the back-propagation stage.

In max pooling, as only one of the features is responsible for the loss, the kernel weights w are only updated taking into consideration the window with the highest similarity. The average pooling gradient flow updates the weights taking into consideration the contribution from every window.



(a) Pooling a 4x4 matrix.



(b) Pooling a 6x6 matrix.

Figure 2.6: Effect of pooling on feature dimensions.

In either case, this allows a convolutional neural network to take a variable length input as the amount of windows calculated, i.e the length of c , may vary. The advantage convolutional layers have over its traditional feed-forward counterparts is their capability to take such variable lengths much like recurrent networks. However, unlike RNNs convolutions may be calculated in parallel. This greatly speeds up training and allows the use of bigger models. Additionally, the amount of steps required for two different parts of an input to interact is reduced as the convolution size increases. This path shortening can sometimes translate to better dependency modelling between sections.

A significant drawback of these kinds of layers is that by default the features produced have no way of knowing their relative position. As such, order is only taken into account within the same convolution. This can be an issue in NLP as word order is critical in capturing meaning. That said, CNNs can also produce good results. The convolutions in such applications are 1 dimensional but work in the same way. Instead of pixels, each window contains the distributed representations of a series tokens.

2.2 Local and Distributed Representations in NLP

Machine Learning can be applied to many domains, including ones where the subject matter being treated is not typically represented numerically. Understanding and generating text is one such domain where inputs and outputs need to be transformed into numeric representations that aren't human-readable in order to apply statistical models.

A number of options exist on how to transform the written word into a single or series of numbers while retaining the original meaning. In a local representation a concept such as a word is represented by a single node. Meanwhile, in a distributed representation, the same concept can be expressed as a pattern in the activation seen throughout a series of numerical representations.

2.2.1. Local Representation: Bag of words (BOW)

Assuming the vocabulary for a given problem is finite, each division of text, which can range from letters to whole sentences, can be considered a possible class in a classification problem.

A one-hot encoding representation of a word with an assigned index W_{id} in a problem with a limited vocabulary of size N can be encoded as a zero vector of that size with a one in the position W_{id} e.g:

$$W_1 = the | V_{the} = [1, 0]$$

$$W_2 = car | V_{car} = [0, 1]$$

This approach, while simple, has several problems. The first one being its poor scaling with vocabulary size. Due to the nature of local representations each word has to be represented by a single activation. For a model to have the same basic vocabulary as a native English speaker each vector representing a word would be of size 20000. With most of the numbers in those vectors being 0 this sparse representation is very memory inefficient.

Another of its issues lies in its inability to represent the semantics of a word. Each vectorial representation of a word is as distant from any other without taking into account the context. All of the relationship building is left to the model and has to be learnt again for each application.

Furthermore, this approach is unable to deal with unknown tokens, i.e words or symbols, outside its vocabulary and has no way of learning them.

Some of BOW shortcomings can be addressed by changing what each node represents. N-grams combine N number of tokens together in an attempt to better contextualize the words. This approach acknowledges that sentences are more than the sum of the meaning of its words by paying attention to combinations of them in a certain order. This approach, however, has one notable shortcoming. The higher the number of N , the more context it can capture but this comes at a steep cost as the vector size needed to encode all combinations of words is V^N .

So, following the example above, using 3-grams for a model with the average basic vocabulary of a native English speaker the vectorial representation of each word would be of size 20000^3 .

2.2.2. Distributed Representations: Word2Vec

Over the years neural networks have been proposed as a solution to the dimensionality problems stemming from large vocabulary sizes. These solutions often vaguely resembled the training of a fixed window feed-forward language model 2.3.2 with the notable difference that the objective of such process was instead the hidden layer representations of the network for each word.

Feeding the resultant word vectors to a model dedicated to an unrelated task can be seen as an exercise in transfer learning.

While several other architectures with the same objective exist, Word2vec's strengths lie in its comparatively fast training time, low memory use and simplicity. Word2vec uses a single shallow neural network to generate word embeddings.

Two tasks are often seen in the training of these kind of distributed representations: Continuous bag-of-words (CBOW) and Skip Gram.

In CBOW, the network receives the words surrounding the target w_t to predict. The context provided as input is comprised of words $[w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2}]$. The Skip Gram task is similar but works in the opposite way. Given a word the model tries to predicts its context.

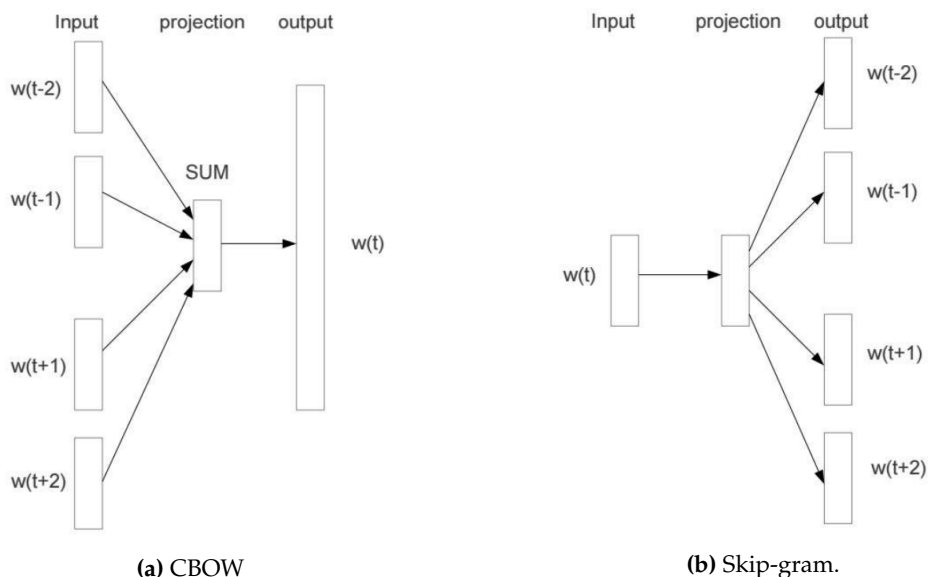


Figure 2.7: Word2Vec architectures [30]

The resulting N-dimensional vector activations contain the syntax and semantics of the word it corresponds to. These vectors have an interesting property that allows for the subtraction and addition of words to form analogous ones. In the following example it is shown how subtracting the singular word from its plural counterpart and adding an unrelated singular noun approximately yields the plural of said noun.

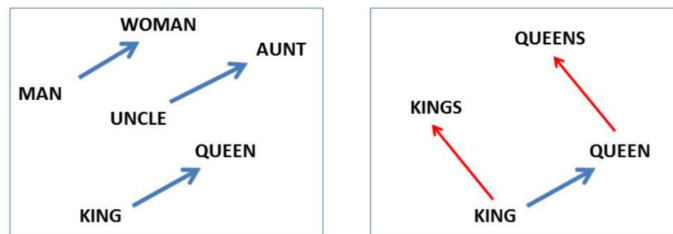


Figure 2.8: Vector operations with word2vec embeddings. [30]

Using the resulting embeddings solves the issues stemming from large vocabularies as its size is unrelated to the amount of words in it. Additionally any model fed such vectors may better generalise previously unseen words contained in the vocabulary using the semantic information contained in the representations.

A problem still remains when an out of vocabulary word is presented. Many of these methods assign a dedicated unknown token for said occasions. While the model may still be able to predict correctly from its context, no amount of training would improve its performance in such cases as the word meaning is lost at the moment of conversion.

2.2.3. Sub-word Representations: Byte Pair Encodings (BPE).

BPE[31] was originally conceived as a data compression technique which worked by replacing the most frequent byte pairs with a single byte. In the context of NLP, the same principle may be applied to individual or series of characters to achieve an automated corpus-specific vocabulary selector.

The token vocabulary commences as the unique character list of the text. All token pairs are iterated upon and counted. Once this is done the most frequent pair is replaced by a token representing the couple combined. The space character is often treated as a delimiter so that tokens won't comprise several words. This is often done for the sake of efficiency. This process is repeated until the desired vocabulary size is reached. E.g

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 2.1: Trace of 5 iterations of the BPE algorithm.

As the token vocabulary is only ever added to, the smaller sequences are maintained. This makes possible the tokenization of sentences not contained in the original text e.g He-ll, S-a-l-e. Being able to recognise out of vocabulary words by combining smaller tokens is essential in some NLP tasks like machine translation where the vocabularies are often open and full of rarely occurring words.

Alternatives to the algorithm explained above exist and may sometimes perform better. Some state-of-the-art models use slightly different approaches to achieve the same objective. Wordpiece, for instance, prefers grouping by the likelihood given by a pre-trained language model instead of pair frequency count.

Once the tokenization is concluded the vocabulary might be fed to algorithms like word2vec to generate distributed representations for other models to use or alternatively can be directly fed to a language model at training time.

2.3 Language Models (LM)

A language model's purpose is the estimation of the probability distributions of tokens in a vocabulary for a given context. These models are often trained with conditioned next token prediction tasks with the objective of forming an understanding of language through the processing of large amounts of written text. This task is selected because of written text's sequential nature and the lack of need for supervised samples. These language models may then be used as is for next-token prediction, as seen in predictive text keyboards, or may be used as a base for a model trained to perform another task such as summarisation or text classification.

At its most basic a language model's task is to compute the probability distribution of next word p_{t+1} given a sequence of words x^1, x^2, \dots, x^t where x^t is any token in a vocabulary V .

$$P(x^{t+1} | x^t, \dots, x^1)$$

These kinds of models may also assess the probability for a given sequence of text of length T by multiplying the probability of each of its tokens given the previous ones.

$$P(x^1, \dots, x^T) = P(x^1) \times P(x^2 | x^1) \times \dots \times P(x^T | x^{T-1}, \dots, x^1) = \prod_{t=1}^T P(x^t | x^{t-1}, \dots, x^1)$$

2.3.1. N-gram based Language Models

Before the advent of neural networks, count-based language models were considered the norm. The likelihoods described above were determined by the amount of times a token was observed after a given context. For a 3-gram language model the probability distribution would be calculated in the following way:

$$p(w_3 | w_1, w_2) = \frac{\text{count}(w_1, w_2, w_3)}{\text{count}(w_1, w_2)}$$

This approach has one main flaw. If a combination is not seen at training time the LM will always predict the likelihood of that sequence as 0. These combinations with 0 probability are often numerous, specially as the amount of context tokens increase. This sparsity problem may be attempted to be solved with smoothing techniques. Smoothing gives every combination at least a small likelihood of occurring, even combinations that are not grammatically correct. Another flaw of this approach is that window which the model views is always the same size as the N-gram length. This makes language models unable to identify long-term dependencies in text even with n-gram overlapping.

In addition, this method much like other local representations is unable to assess the semantic similarity of sequences that might partially share meaning e.g "The child ran

towards its mother" and "The infant sprinted in the direction of her parent" are seen as equally distant as "The quick brown fox jumps over the lazy dog".

2.3.2. Feed forward Neural language models

In 2003, neural probabilistic language models [32] were proposed as a way to simultaneously solve the curse of dimensionality stemming from large vocabularies and the inability of other methods to express relationships between words. The representations resulting from such approach serve both as distributed representations of words and probability functions for word sequences.

Every word i in a vocabulary V is associated with a unique feature vector $C(i)$ of size M unrelated to the vocabulary size. This function C is in fact a weight matrix of size $V \times M$ where the row i corresponds to word i in the vocabulary. When the tokens $\{w_{t-n+1}, \dots, w_{t-1}\}$ are fed into the model their corresponding vectorial representations $\{C(w_{t-n+1}), \dots, C(w_{t-1})\}$ are retrieved and concatenated in the next layer.

The concatenated vectors are multiplied by the neural network's weights and fed through a tanh activation function. The resulting vector of size V contains the statistical distribution of for every word in the vocabulary given the context provided.

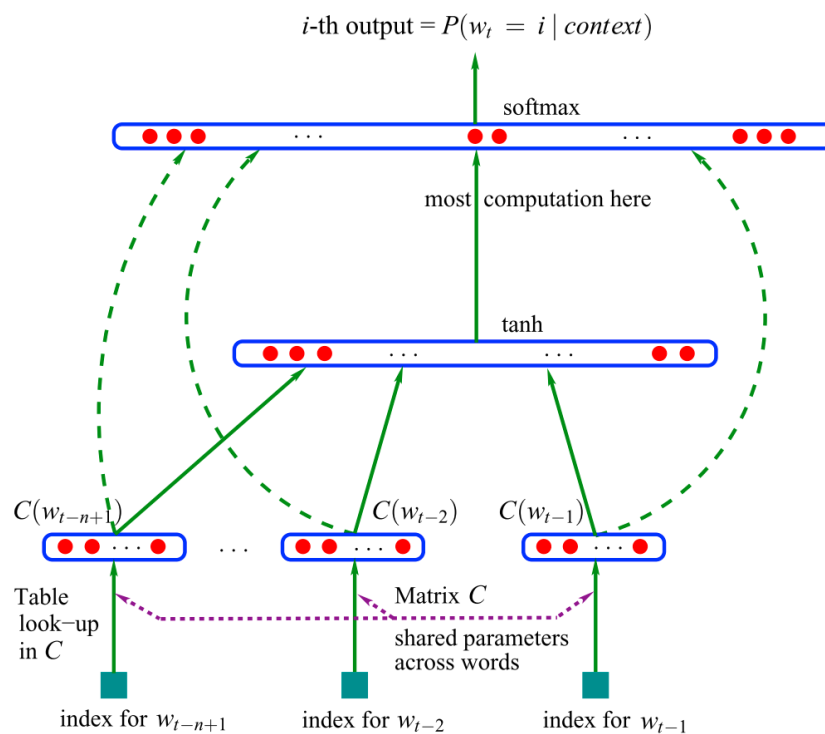


Figure 2.9: Diagram of model proposed in Neural Probabilistic Language Models paper. [32]

This final output is compared to the one-hot representation of the word it was attempting to predict. The loss is then calculated and backpropagated through the network including the weight matrix C containing the distributed representations.

These improvements over the LM covered in 2.3.1 help solve the sparseness problem faced by it. The feature vectors which contain the words semantics allow it to better generalize to rare examples. As it only uses a feed forward network, the amount of parameters used is proportional to the context window to be considered. This is why

this system still struggles to capture long term dependencies outside the N-token context given to it.

2.3.3. Recurrent Neural Network Language Models

RNNs ability to retain information throughout feed-forward steps improves the long-term dependency learning of language models without increasing the size of the networks. As detailed in section 2.1.6, recurrent networks can deal with variable length contexts which are a common occurrence when dealing with text data.

The recurrent language model may be fed pre-trained distributed representations of words or localized one-hot vocabulary vectors one at a time. For every step, the model may attempt to predict the word that follows. As hidden layer states are propagated through time previous tokens are remembered at the time of prediction to the best of the network's ability. The models capability to detect long term dependencies may change depending on the kind of recurrent cell used in the architecture.

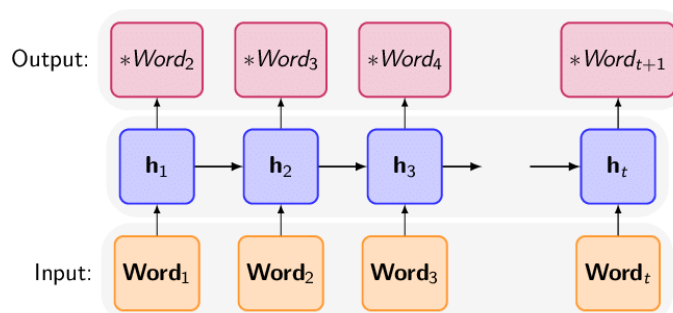


Figure 2.10: Simplified Recurrent Language model diagram. [34].

While a significant improvement over fixed-window language models, it is important to keep in mind the limitations and issues of RNNs covered in section 2.1.6. Their slow training and gradient problems limit their size and keep them from being as effective and scalable as transformers.

2.3.4. Convolutional Language Models

In 2016, a CNN based language model achieved better performance than a classic RNN architecture. This model was proposed in the paper *Language Modeling with Gated Convolutional Networks*[35] and, while it fell behind in performance compared to state of the art LSTM models, it reduced the compute time needed to train a language model significantly.

These kinds of convolutional models over text depend in great part on word embeddings like word2vec to train the filter weights. For a convolution that spans h words and uses embeddings of size m , the weight matrix is $w \in \mathbb{R}^{mh}$. The window step is always a multiple of the size of the word embedding so as to not split a word mid representation.

This work also modifies how convolution context windows capture data. Normal convolutional windows take the context around a given input. E.g for the sentence "The quick brown fox jumped" the context captured for the word "fox" would be "brown fox jumped". This is an issue in language models as the objective is to predict the next word in a sentence. To avoid including the next token in the input causal convolutions are used. This window version captures the previous h tokens instead.

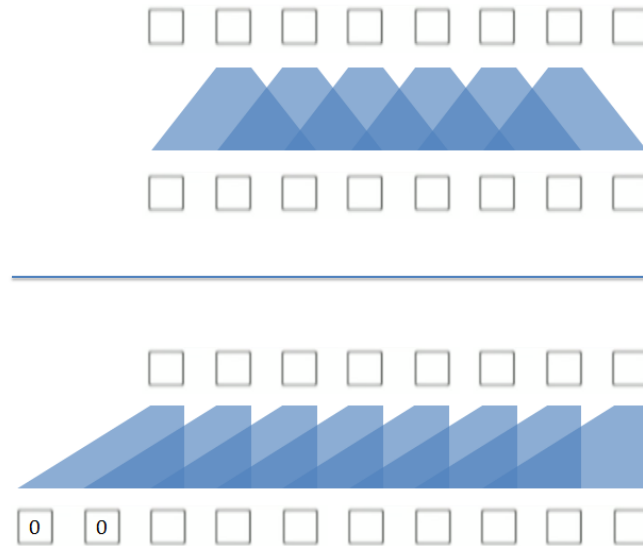


Figure 2.11: Illustration of a standard convolution (top) versus causal convolution (bottom)[36].

Using this modified convolution on the example above yields a context window containing the words "quick brown fox" which does not include the next word "jumped" which is to be predicted.

The convolutional language model featured in the 2016 paper[35] used gates to manage the flow of information through the network. Some recurrent neural network architectures had in the past used gates to control the flow of information between time steps. One such example are the LSTM models mentioned at the end of section 2.1.6 that used remember and forget gates. This gate implementation differed because it controlled the flow of information within the same timestep using a set of convolutional filters V dedicated to the gating of other filters W which contained the information carried.

Both sets of filters receive the same input but have different roles. The result is that the gating convolution weights B learn to assess the relevance of the filters carrying the information A . This relevance is calculated with the tensor product of A and the sigmoid of B . The sigmoid is used to scale inputs between 1 and 0. This way, filters that produce highly relevant activations for a given sentence will have a corresponding value in the gating matrix B close to 1 while irrelevant sections will have lower values.

$$H = (X * W + b) \otimes \sigma(X * V + c)$$

These layers can be stacked to increase model complexity and produce better results. Models can be made larger and trained faster than recurrent neural networks thanks to the parallel nature of convolutions. However, larger networks face a vanishing gradient problem. To solve this skip-connections are placed between layers to aid in gradient flow. It is worth noting that as in the architecture described no pooling was used the context window is fixed unlike RNNs so there is hard upper limit on the length of the dependencies it can capture.

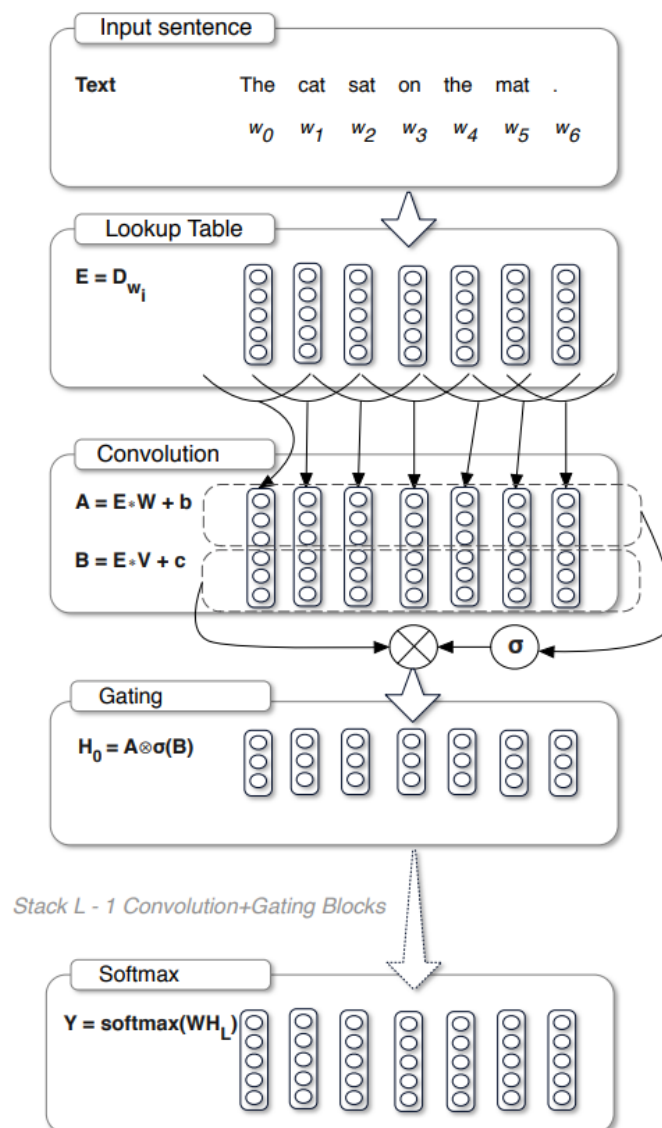


Figure 2.12: Gated convolutional language model architecture. [35].

2.3.5. Language Model Sampling

A LM's output for a given context is the probability distribution for the new token. If the objective of the LM is to generate human-like text just selecting the token with the maximum likelihood might cause problems. These problems stem from the model's attention being primarily centered on the last tokens given as a prompt. This deterministic approach to token sampling can easily get stuck in loops, go off-topic or just produce sentences that are easily identifiable as computer generated.

A solution to these problems has been found with different methods of controlled random sampling. Using a purely random approach is ill-advised as it might cause the model to go off-topic frequently. This is in part because in a vocabulary of 10 thousand tokens, even if the bottom 80% of tokens are extremely unlikely they might still have a high combined probability. Choosing one of those numerous poor next token predictions would derail the topic of the generated output.

Temperature sampling allows fine control over the randomness of predictions in a model and it used to solve high repetition in generated text. Inspired by statistical thermodynamics, the intuition revolves around how higher temperatures make low energy states more likely to be encountered. This translates to a smoothing of the softmax function which makes high probability predictions less likely and assigns a higher probability to poorer predictions. This modified softmax function is equivalent to the one covered in section 2.1.2 when the temperature $\tau = 1$.

$$\text{Softmax}(x_i) = \frac{e^{\frac{x_i}{\tau}}}{\sum_j e^{\frac{x_j}{\tau}}}$$

In top k sampling all probabilities for tokens that are not in the k most likely are set to 0. This method of solving the problem caused by vast amounts of very unlikely tokens works at its best when there is a narrow distribution of probabilities and the number of good options is lower than the value selected for k . When the model encounters a broad selection of equally likely tokens this sampling approach might limit the variety in the models output.

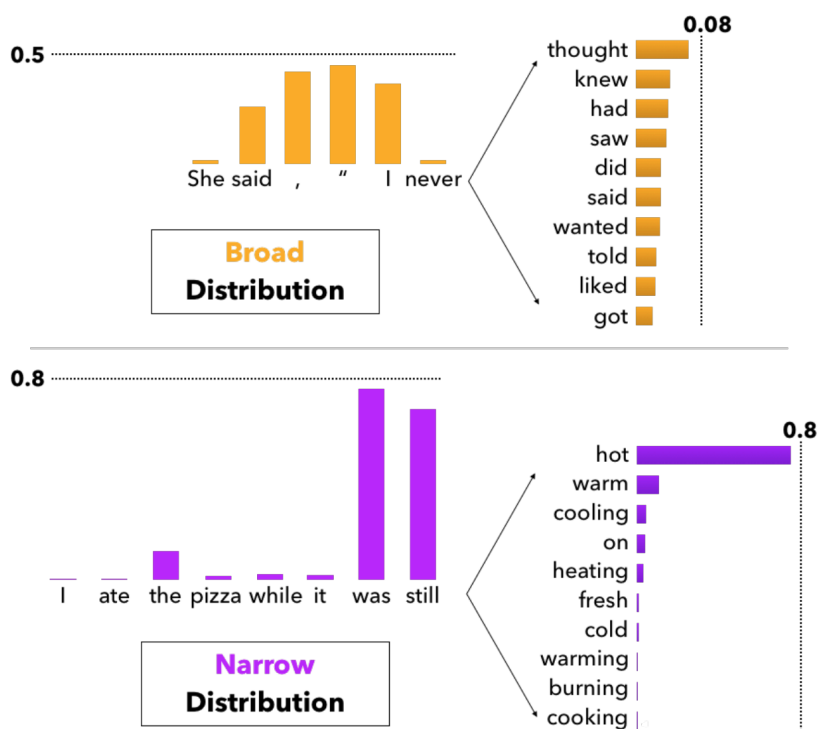


Figure 2.13: Example of broad and narrow distributions. Using $k = 3$ would perform favourably in the narrow distribution but limit the range of options in the broad one. [37]

Nucleus sampling [37], also commonly known as top-p sampling, takes into account the cumulative distribution instead of a fixed number of top samples. The most likely tokens are selected until their cumulative distribution exceeds the parameter. In the example above, for $p = 0.5$ nucleus sampling would select around 8 tokens from the broad distribution and only the top one from the narrow example.

2.4 State of the art results: Transformers

2.4.1. Attention is all you need

The transformer was first proposed in 2017 in a paper by the name "*Attention is all you need*" [38]. This model for use in sequence to sequence tasks forwent recurrent layers in favour of attention.

These attention heads in neural networks produce a vector which is a linear combination of others. In the context of sentences this unit may be described as deliberately ignoring some words in favour of others by adjusting in what proportions their embeddings feature in the resulting layer. This is conceptually similar to how gates operated in section 2.3.4 which described the gated convolutional model architecture.

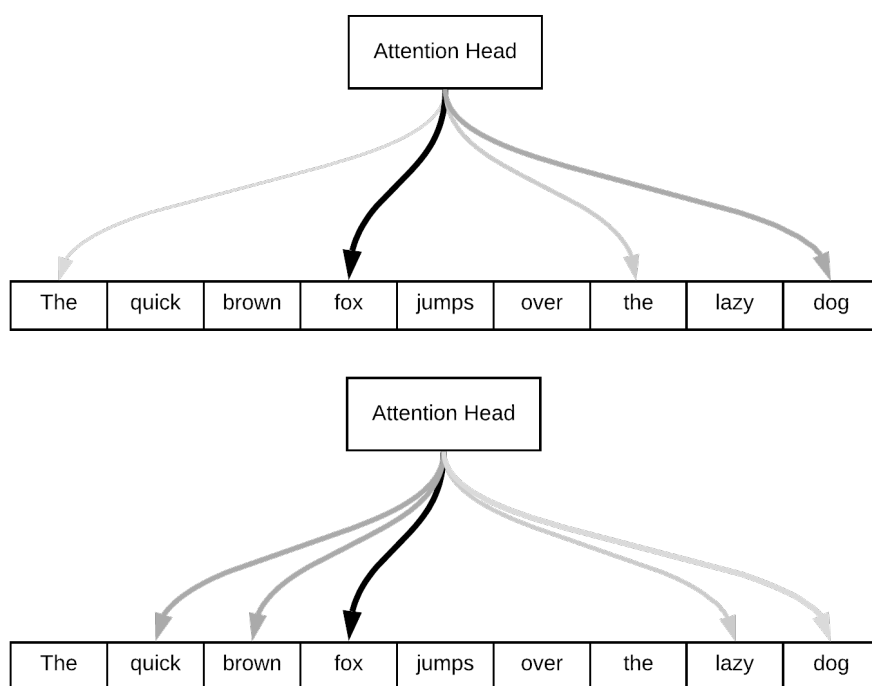


Figure 2.14: Illustration of the attention concept in neural networks. Line thickness and darkness indicate more importance.

Several heads are used within the same layer with the objective of capturing more than one aspect of a sentence. In the example given above the attention of the model is captured by nouns and their articles while the second one fixates on subjects and the adjectives that accompany them. It is important to note that which tokens the head assigns a higher importance is learnt as the model trains. The model needs no prior knowledge of grammatical rules.

The attention function is fed three different parameters: Key K , value V and query Q matrices. The keys index the value matrix, the values correspond to the data being carried and the queries can be thought of as a question to be answered. The relevance of each key to a query must be ascertained in order to assess which of the values is more relevant to Q . This is done by calculating the dot product between the two matrices which results in a vector containing the cosine of the angle between them for every key. The relevance vector is then multiplied by the value matrix which provides the linear combination of

weights mentioned above. The "Scaled Dot-Product Attention" function described in the paper is as follows:

$$A(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

The dot product of Q and K are divided by the square root of K 's dimension to counteract an effect seen for larger dimensions of K . In such cases, the large magnitude of dot product may push the softmax function to the extreme values where gradients are small and hence hinder learning.

2.4.2. Encoder blocks

The attention mechanism covered in section 2.4.1 forms part of an encoder block. The output of the function is concatenated and normalised before being received by a standard feed forward network.

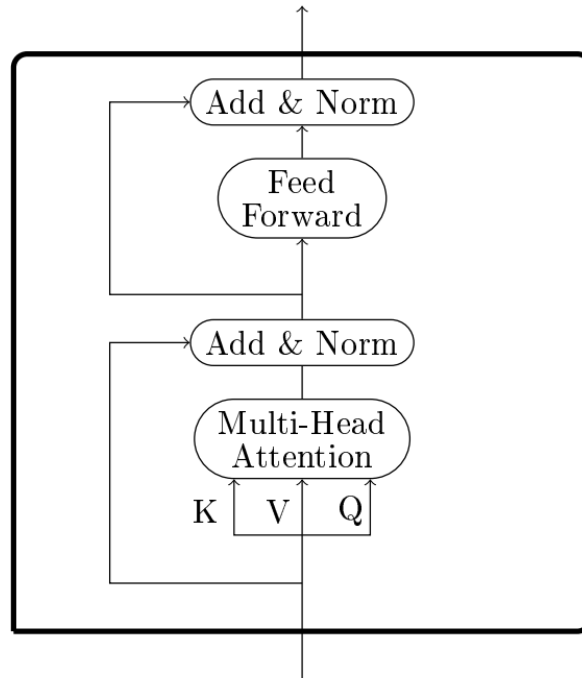


Figure 2.15: Transformer encoder block. [39]

Residual connections are placed between every pair of layers to better allow the flow of information through the net. Blocks without such connections lose accuracy in part because of the loss of the positional information which is added to the word representations at the beginning of model[40]. This is expected as positional encodings are essential to modelling the word order in transformers. Without these locators, the model just perceives an unordered set of tokens. These locating features can be learnt or fixed, both approaches have been observed to perform equally well.

In the paper introducing the transformer, the authors propose using a set of sine and cosine functions with different frequencies. The location representations for tokens in position i of a model of size d_{model} are encoded as follows:

$$PE_{pos,2i} = \sin(pos/10000^{2i/d_{model}})$$

$$PE_{pos,2i+1} = \cos(pos/10000^{2i/d_{model}})$$

This method ensures the model is able to understand the relative positions of the tokens given to it without the need of recurrent architectures.

2.4.3. Decoder blocks

The decoder, in charge of generating the models output, greatly resembles the encoder but with a couple of differences. Two attention functions are stacked on each other with the second only providing the query Q and receiving its keys K and values V from the encoder.

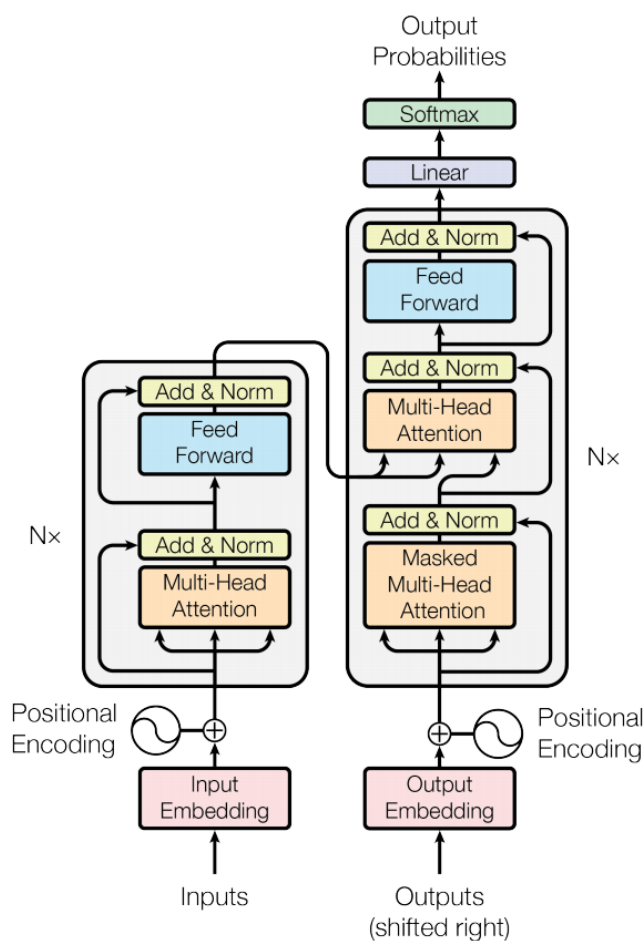


Figure 2.16: The Transformer - model architecture. Encoder to the left, decoder to the right. [38]

When generating text all of the context is fed through the encoder and the decoder is left blank. To ensure each input to be ignored does not affect the output, they are masked by assigning highly negative values to its relevance in the attention function. As words are predicted, they are placed as inputs and the position occupied is unmasked so that it may affect the next output. This masking and unmasking of different inputs is how transformers may handle varying lengths of tokens up to a maximum dimension.

2.4.4. Transformer scaling

In the section introducing recurrent neural networks 2.1.6, this work mentions scaling problems inherent to the serial computation the architecture requires. The following table

compares the complexity of the three kinds of layers covered in terms of the sequence length n and the representation dimension d .

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
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 2.2: Comparison of layer properties. [38]

As can be seen, there is a trade off between the size of the model states and the context the model can handle. The transformer's complexity scales quadratically with the length of its context window, i.e maximum token length it can process. As this kind of model has a fixed window size, it may not capture any context beyond said n tokens. Poor scaling in this area severely limits how long can dependencies be.

While these statements are true, the reality of modern hardware is that it favours vastly parallelisable tasks. Self-attention layers can have each of their outputs calculated independently. This means vast quantities of compute resources can be leveraged at the same time making even 175 billion parameter models with $n = 2048$ trainable by high performance clusters[41]. Convolutional neural network based architectures have similar strengths in terms of parallel computation and scale much better with context size n . Unfortunately, it has been shown to fall far behind LSTMs based architectures in accuracy [35].

These computational strengths are in stark contrast with recurrent networks which, while potentially having the capacity to carry dependencies indefinitely, the serialised nature of the computation makes memory requirements and lack of parallelism an important bottleneck.

This is not aided by the fact that this potential for long term dependency modelling is not fulfilled in current architectures. This is due to the amount of steps information must be propagated through the network and gradient based problems covered in 2.1.6. In RNNs, the path length between two tokens in positions i, j is $|i - j|$ while in transformers any two tokens may interact directly not matter how far apart they are.

2.4.5. Generative Pretrained Transformer 2: (GPT-2)

The encoder-decoder architecture seen in the original transformer paper was well suited to machine translation tasks. Later iterations on the transformers had different objectives. GPT-2's goal, for instance, was the generic training of a LM for multiple tasks including text generation. To this end, the creators forwent the encoding part of the original design and constructed a transformer exclusively formed from stacked decoder blocks like the ones seen in section 2.4.3.

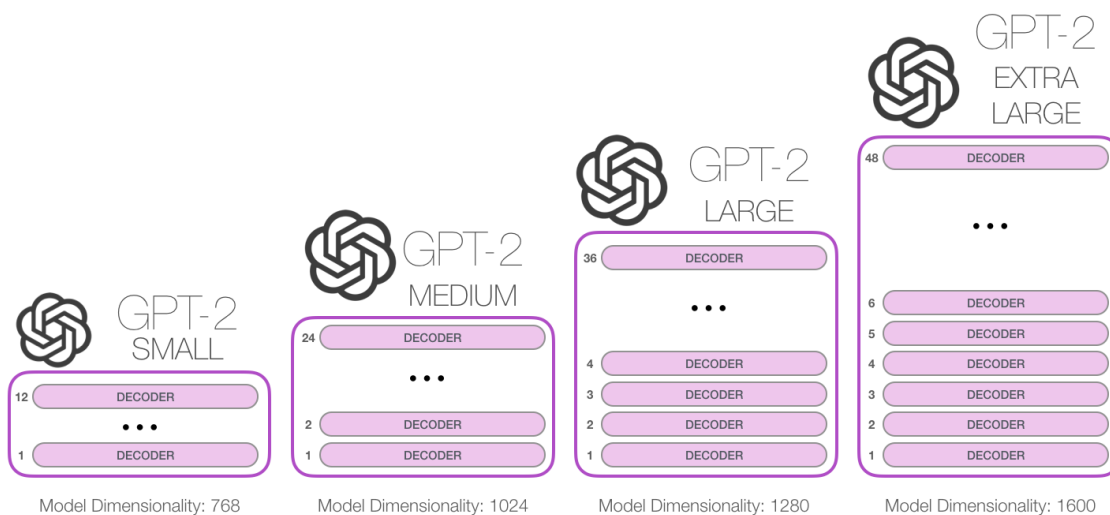


Figure 2.17: GPT-2 released model architectures.[42]

The way sentences are generated is identical to how previous decoders worked. Inputs are progressively unmasked as the outputs are generated and placed in the next position of the input. The extra-large model used in the experimental part of this work has a context window n of size 1600 which is shared between the prompt as the text as it is being generated.

The original GPT-2 model is trained on a 40GB corpus named WebText [14]. While the corpus is still unreleased, it is safe to say that the dataset contains a great variety of text structures which results in the model's ability to perform several different tasks without any further training. As can be seen in the table below, the model is shown to have the capability to do zero shot translation from French to English if an appropriate prompt is given. Additional zero shot performance numbers for other tasks are given in *Language Models are Unsupervised Multitask Learners*[1].

"I'm not the cleverest man in the world, but like they say in French: **Je ne suis pas un imbecile** [I'm not a fool].

In a now-deleted post from Aug. 16, Soheil Eid, Tory candidate in the riding of Joliette, wrote in French: "**Mentez mentez, il en restera toujours quelque chose,**" which translates as, "**Lie lie and something will always remain.**"

"I hate the word 'perfume,'" Burr says. 'It's somewhat better in French: '**parfum.**'

If listened carefully at 29:55, a conversation can be heard between two guys in French: "**-Comment on fait pour aller de l'autre cote? -Quel autre coté?**", which means "**- How do you get to the other side? - What side?**".

If this sounds like a bit of a stretch, consider this question in French: **As-tu aller au cinéma?**, or **Did you go to the movies?**, which literally translates as **Have-you to go to movies/theater?**

"Brevet Sans Garantie Du Gouvernement", translated to English: **"Patented without government warranty"**.

Table 2.3: GPT-2 zero shot translations quoted in *Language Models are Unsupervised Multitask Learners*[1]

2.4.6. BERT and RoBERTa

In the paper *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*[43], the authors propose a modified transformer architecture which consisted on stacking encoder blocks similar to those seen in section 2.4.2. In addition to this, the training tasks proposed differed from the previous iteration of the transformer. Instead of predicting tokens sequentially and building a directional understanding of language, the BERT team proposed using token masking and next sentence prediction (NSP) to build a bidirectional encoder.

In the masked language model (MLM) task, 15% of all tokens in the corpus are selected for replacement. Of this 15%, 80% are replaced by the "[MASK]" token, 10% are replaced with other tokens and the final 10% are left unchanged. The objective of the transformer is to correctly predict the sentence including the representations of the words covered in the input.

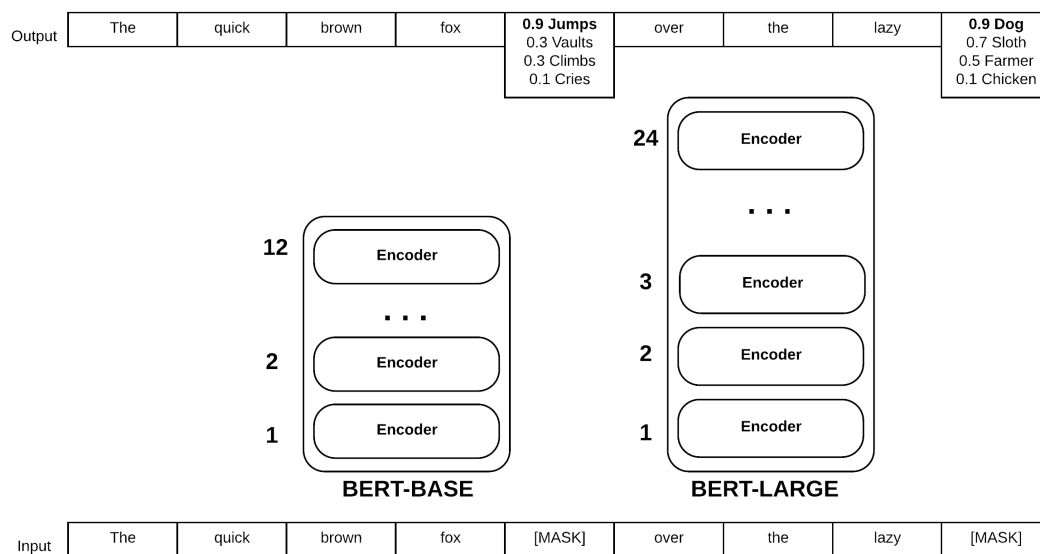


Figure 2.18: Bert masking prediction.

The second task, next sentence prediction (NSP) samples 2 sequences of tokens. The model acts as a binary classifier that determines if those sequences belong together, i.e. are found sequentially, or not. The negative and positive examples are sampled at the same rate. The negative samples are each obtained from different documents. This task is included to increase performance in downstream uses of the model that require understanding of the relationship between two sentences such as question answering.

This model architecture was later re-used in *RoBERTa: A Robustly Optimized BERT Pre-training Approach*[44]. In this study, the authors modify the training procedure of the model to achieve better results across multiple tasks.

The vocabulary was increased from BERT's 30 thousand wordpiece embeddings to a total of 50 thousand. These BPE encodings were made using character bytes instead unicode symbols as seen in section 2.2.3. The model was also trained with a much larger corpus, 160GB compared to the original 16GB. The training data was collected from several different corpora and contained more variety than the original BERT training-set which only included BookCorpus[45] and English wikipedia extracts. Roberta expanded this

training set by adding the openwebtext[46], ccnews[47] and stories[48] corpora which added articles linked on reddit, news and story-like texts respectively.

The training tasks were also slightly altered. RoBERTa uses dynamic masking patterns as opposed to BERT's static approach. This means a given sequence will have different masking patterns every time it is seen by the model. For NSP the sequences being compared are much longer. Instead of sentences, the model takes 2 sequences of 512 tokens unless the end of the document is reached at which case it is just sampled until the end.

CHAPTER 3

Design

3.1 Chosen approach

Training a 1.5 billion parameter model requires an extremely high amount of resources. Memory consumption of the model at training time can easily exceed 84 GB of memory with a low batch size of 2 as well as the use of GPUs to accelerate all of the floating point operations involved in training such a large neural network. Originally the code provided by OpenAI was meant to be used in systems using 4 to 8 high memory GPUs to split the memory usage among them. This approach proved to be quite expensive monetarily as free resources provided by research programmes often offer less compute that would be needed.

Thanks to Tensorflow's support of tensor processing units (TPU) a fork of the previously mentioned GPT-2 project adds the ability to train on them. A TPU is a specialised type of processor developed by Google used to accelerate neural network related computing. These processors are structured in large pods and are made available through Google Cloud Platform and Collab notebooks. The main benefit in our application is that said compute units have a high amount of memory available for large models. Training can also be faster than in GPUs if it is programmed taking every potential bottleneck into account, ranging from data loading bandwidth to distributed compute strategies.

Leveraging Google's Tensorflow Research Cloud (TFRC) programme that provides a month of free TPU access for researchers, the large model was trained for approximately 11 days without GPUs using Google cloud platform.

```
PYTHONPATH=src python ./train.py --dataset plaintext_speaker_sep.txt.npz
↪ --model_name 1558M --batch_size 4 --save_every 500 --sample_every 500
↪ --init_tpu --n_ctx 1024 --sample_length 1023 --n_embd 1600 --n_head 25
↪ --n_layer 48 --restore_from "latest" --run_name 'run2' --learning_rate
↪ 0.00002
```

Once model training subsided, volatile Google Collab notebooks powered by cloud TPUs were used to load the model and produce text samples for testing. TPUs are still required for generation as only performing the feed forward part still requires a significant amount of compute power and memory.

3.2 Compute Resources

The data preprocessing and experiments took place in virtual instances hosted by Google Cloud Platform. Working with virtualized environments meant the compute power available was largely flexible. As data preprocessing and BPE encoding are more CPU intensive tasks a preset hardware configuration called ‘n1-standard-8’ with 8vCPU and 30 GB of RAM was used.

The instance was then powered down to a n1-standard-2 configuration with only 2 vCPUs and 7.5 GB of memory available. This hardware configuration was more than sufficient for training as most of the computation takes place in the reserved TPU pods. The instance only ever sees activity when a model checkpoint is saved.

The TFRC programme includes allocates several kinds of resources:

- 5 on-demand Cloud TPU v2-8 device(s) in zone us-central1-f
- 100 preemptible Cloud TPU v2-8 device(s) in zone us-central1-f
- 5 on-demand Cloud TPU v3-8 device(s) in zone europe-west4-a

The configurations made available allocate a maximum of 8 TPU cores as a single compute resource but allow for several instances to run at a time. The specs for v2 and v3 pods of TPUs are provided by Google[51]:

- TPU v2:
 - 8 GiB of HBM for each TPU core
 - One matrix unit (MXU) for each TPU core
 - Up to 512 total TPU cores and 4 TiB of total memory in a TPU Pod
- TPU v3:
 - 16 GiB of HBM for each TPU core
 - Two MXUs for each TPU core
 - Up to 2048 total TPU cores and 32 TiB of total memory in a TPU Pod

The higher memory per core in v3 TPUs allow the model to train without running out of memory. This allows for the model to train with a higher number of batches without the need to use gradient accumulation. 8v3 TPU cores can handle up to a batch size of 4 for this particular model before running into memory problems. The on-demand property of the resource offered meant the training was able to continue uninterrupted for the entire 11 days.

This is in contrast with preemptible TPU v2-8 resources that are also available as part of Google Collab notebooks. They have a maximum duration of 24 hours after which they have to be spun up again. Preemptible instances are often offered at much a lower price to make up for that fact. This type of instance is used in our case for the generation of samples after training.

3.3 Tensor Processing Units (TPU) for Deep Learning

Domain specific hardware architectures trade the capability of handling general tasks in favour of efficiency in a smaller subsection of them. This is the principle behind complex

instruction set computer (CISC) architectures which utilize long instruction sets to speed up computation on certain tasks.

The concept revolves around more directly implementing instructions that mirror elements of high level languages like a loop, instead of replicating the behaviour using a larger amount of simple operations. Having specific instructions for elements such as loops allowed the processor to execute those elements in a lesser amount of time. The speed-up also came from overall lower memory access times as sometimes results would be saved to the processors cache waiting for the next instruction. As memory speeds in the 1970s were slower, even accessing level 1 cache imposed a significant performance penalty.

These issues are still observed in applications that require large amounts of specific complex instructions. CPUs struggle to keep up with large matrix operations as their instruction sets were primarily designed for general computing. Graphical processing units perform better in deep learning tasks thanks to their large amount of compute cores needed for most of the tasks they are designed for (geometric calculation, polygon rendering, texture mapping...) which involve matrix and vector operations.

When a GPU is used for deep learning the tensors are unfolded into 2D matrices. Typically, the host CPU provides kernel functions to the GPU which can then use a large number of threads to compute the operations. For example, in GPU matrix multiplications the entire operation may be implemented as a function where each thread is used to compute one element of the output matrix. The smallest unit (compute primitive) being used is a vector. Matrix multiplications require a large amount of multiply accumulate operations which are typically the most time consuming part of deep learning.

Application-specific integrated circuits (ASICs) can provide higher performance as they are purpose built for one application. TPUs are neural network ASICs designed by Google to accelerate AI training and serving. One of the most important features of TPUs is the use of systolic arrays for matrix operations.

A systolic array is a collection processing units arranged in a 2-dimensional grids. Each core comprising the array is a multiply accumulate unit plus some additional logic that can store and forward data. This hard-wire approach to matrix calculations allows TPUs to reduce the amount of writing and reading from memory during computation, potentially speeding up performance and being more energy efficient. This same matrix calculation that comprises a great number of different processes with a GPU, is instead done in a single very complex instruction. The compute primitive for these TPUs are entire tensors. Another performance advantage of TPUs as a dedicated piece of hardware is having hard-wired popular activation functions which make calculations much faster.

3.4 Environment

All instances were created using a Debian 9 standard image. Debian is a Linux distribution composed of free and open-source hardware and forms the base for a number of other forks such as Ubuntu.

Jupyter notebooks are interactive documents that are divided in cells. Each cell may run code in processes called kernels which return the output of the code and maintain variable states through cell executions. As the notebooks run on a web based interface, the computation can run on a remote machine which in this case was the cloud instance.

As the scripting for all components of this work are written in Python, Anaconda an open-source distribution of the language is used. It is often used for scientific computing thanks to its simple package management system named conda. Virtual conda environ-

ments can be created to isolate project requirements from one another. This greatly simplified using the same machine for all three components: corpus creation, GPT-2 training and detector training although the later two had some conflicting package dependencies. While it is true pipenv can achieve the same goals and is bundled in with Python, Pip, its package management has one key flaw that makes it not fit for purpose. Conda supports binary packages whereas pip doesn't, requiring the user to compile them. In compute heavy Python libraries this is an issue as most of them implemented their underlying calculations in compiled C code which is much faster than native Python which is an interpreted language.

The main packages used in the practical component of this work are:

- **Regex** is a package for pattern matching that offers more features than the standard regular expression package of python "re".
- **Requests** allows the sending of http requests in Python.
- **Tqdm** is a package that creates progress bars. Extremely useful when monitoring the progress and performance of operations on large datasets.
- **Tensorflow** is an end-to-end open source platform for numerical computation and machine learning. Its API is available for a variety of languages but most of its code is executed in C++ binaries. Tensorflow works by building dataflow graphs where each node represents a mathematical operation. The library offers a large variety of high level abstractions that make the development of machine learning models faster. The library facilitates the execution of code across different kinds of compute resources. CPU and GPU code don't require special adaptations and TPU conversion is greatly accelerated. GPT-2 is implemented in Tensorflow.
- **Tensor2Tensor**. T2T is an an open-source system for training deep learning sequence to sequence models in Tensorflow. Its objective is to facilitate the creation of state of the art models for a number of NLP tasks. It includes a library of datasets and models that include the implementation of the transformer covered in section 2.4.1. At the time of writing this library is in maintenance mode and use of its successor Trax is encouraged.
- **Torch** is another open source machine learning library much like Tensorflow. One of the key differences that distinguish both is that Torch's dataflow graphs are dynamic whereas Tensorflow 1.X's are static by default. The RoBERTa based detectors are implemented with Torch.
- **Tensorboard** is a visualization toolkit that provides measurements and visualizations of a machine learning workflow.
- **Transformers** is an open-source package created by Huggingface that provides state-of-the-art general-purpose architectures. These ready-made NLP models can be used in both Torch and Tensorflow. This includes transformers such as the ones used in this project. This library is only used for the detectors as the GPT-2 implementation used in this project is a fork of the original openAI code modified to run on TPUs.
- **Tokenizers** is a package that offers python bindings to a very fast rust implementation of Bert's WordPiece and other 3 common BPE versions.
- **Fire** is a package that automatically generates command line interfaces for Python files.

- **H5PY** is a python interface to the HDF5 binary data format. It is used to store large amounts of numerical data in a way that it is easy to manipulate with packages like numpy. It is used to save model weights.

Tensorboard was used to monitor training progress through the web interface without having to ssh into the machine. During the development of the scripts used in data pre-processing, a jupyter server was started in the GCP instance and accessed through a web browser in several local machines making the development environment location independent.

At the end of the project, all trained model weights, scripts and corpora are stored in a coldline GCP bucket. This kind of bucket is a cloud storage resource meant for archival of files. Buckets are easily accessible from other GCP services and Collab notebooks.

CHAPTER 4

Case Study

The experimental part of this work will fine-tune the released 1.5 billion parameter language model with a highly political and specialized setting. The outputs of this model will be used to measure the effects of generator fine-tuning on the accuracy of the detectors released [56]. The dataset used to fine-tune will be comprised of post 2016 British House of Commons transcripts. [5]

This setting was chosen for several reasons. The proper etiquette of house of common debates is quite different from most internet discourse in which GPT-2 is pre-trained on. Furthermore, there exists a wide spread concerns that such automated models could be used by politically inclined "malicious actors" [3]. The fine-tuning of such a model could at a later day be used to train for this specific setting the same detectors being evaluated. The final reason is that while all parliamentary debates can be found online [6] there doesn't seem to be a compiled source of recent texts for easy analysis.

4.1 Corpus creation: cleaning and web-scraping

4.1.1. Source Data

The parliament of the United Kingdom of Great Britain and Northern Ireland has historically stored all the transcripts from 1771 in a collection called the 'Hansard'. In its current form, the British Hansard is available online in the form of a searchable website [6].

Debates can be searched for by keyword and filtered by chamber (Commons or Lords) and dates. The search query is then encoded as shown in figure 4.1 where the words between braces correspond to variables.

```
https://hansard.parliament.uk/search/Debates
?endDate={end_date}
&house={houses}
&searchTerm={search_term}
&startDate={start_date}
&page={page_number}
```

Figure 4.1: Search Request

Once the search terms are introduced, a new skeleton page is loaded which is shortly after populated with the search results by an AJAX request [9]. AJAX is a set of web development techniques used for asynchronous loading of information in Javascript. This

allows a webpage to be loaded in a browser while it is querying a server increasing the responsiveness of the website.

These asynchronous requests mean that search result information is not present at the time of request meaning web-scraping libraries without Javascript execution can't access this information. After the AJAX table is populated by the search results, the user can then click and navigate to the website that contains the transcript for the selected debate. The link to said page adheres to the following format:

```
https://hansard.parliament.uk/Commons/{YYYY-MM-DD}/debates/{unique_id}/{Title}
```

Once the user navigates to the above mentioned link, the transcript of the debate can be found within the contents of the website. More importantly, a download link for a ".txt" file is available. This URL follows the format shown below with the variable unique_id corresponding to the debate identifier.

```
https://hansard.parliament.uk/debates/GetDebateAsText/{unique_id}
```

4.1.2. Legality

Parliamentary archives can be used freely for non-commercial research and private study. This work falls into both categories and as such the use of these archives is allowed.

"Parliamentary Archives copies : copies supplied by the Parliamentary Archives may only be used for non-commercial research or private study and/or other exceptions to copyright as outlined within the Copyright Designs and Patents Act 1988, as amended and revised." [8]

As detailed in the same page, any citation of the original records is also allowed as long as the proper acknowledgements are made through the use of the catalogue reference number e.g "*Parliamentary Archives, HL/PO/PB/1/1605/3J1n23*". This work will cite every transcript referenced and will clearly label model generated samples as requested by Open-AI team.

4.1.3. Environment and technologies

Due to the variety of AJAX[9] techniques used in the presentation of data within the Hansard website, commonly used Python HTML/XML scraping libraries like BeautifulSoup [10] can't be used. However, browser automation tools such as Selenium[11] and libraries built on top of it allow Javascript code contained within the site to be executed. Visiting a web page with Selenium is by design the same as accessing through a normal web browser as the main use case for the library is the development of automated tests.

Selenium is compatible with a sizeable selection of different browsers called webdrivers[12]. The standard google chrome driver[13] is used as it allows for an easier debugging of automated actions thanks to its inclusion of a graphical user interface (GUI). Headless webdrivers i.e automated browsers without a GUI are supported by Selenium and could be a potential performance improvement as they can retrieve data faster as long as the request response latency isn't a bottleneck.

4.1.4. Webcrawling

As observed in Figure 4.1, URLs used throughout the Hansard site are modular which greatly simplifies its traversal. The first step is to obtain all of the debate_ids for a given

search. To this end the crawler must save all ids it encounters as it iterates over the search pages.

Our search parameters are as follows:

- **Search_term:** “***”
- **House:** Commons
- **Start_date:** 2016-01-01
- **End_date:** 2019-05-15

In information retrieval systems a wildcard ‘*’ is often used to indicate the retrieval of all information matching the other constraints specified. 3 wildcards are used to get around the minimum character length restriction in place by the browser.

Once the debate ids are obtained, these are subsequently used in the download link for the retrieval of the text files using the ‘GetDebateAsText’ url type.

4.1.5. Corpus preparation

The webcrawling process results in a 757MB worth of unstructured text files. These downloaded transcripts are then structured into a json file with the following schema.

- **Debate_id:** ID List
- **Title:** String
- **Speakers:** String List
- **Votes:** List
 - **Division Name:** String
 - **Ayes:** Integer
 - **Noes:** Integer
- **Interventions:** list
 - **Speaker:** String
 - **Text:** String

Utilizing a JSON file with list objects inside allows the content to be structured while still maintaining order between speaker’s interventions. The structuring of the data is taken care of by a series of regular expressions that capture and clean relevant parts of information.

Some shorthand commonly used in Hansard transcripts are replaced for better readability e.g “hon.” is replaced by “honorable” and transcript notes are removed as they are not part of the interventions by speakers and hence are of no use to the model to be trained.

4.1.6. Text pre-processing

While GPT-2 has been shown to learn a wide variety of text structures at training time, feeding the transcripts as they are found and not filtering irrelevant parts may lead to a lower quality and noisier output. The responses generated could possibly include votes and other undesired housekeeping statements. To increase the quality of the text, the samples given to the model consist solely of the title of the debate followed by the interventions of the speakers. Said interventions are delimited by a special token '<next-speaker>' that will be learnt by the model and will aid it in recognizing the implicit structure of a political debate.

To further increase output quality the original 757MB corpus is combed with a series of filters with the objective of discarding low quality samples. For this specific use the quality of a transcript is defined by the proportion of transcribed dialogue versus noise resulting from imperfect parsing of raw text.

Only sessions with a high amount of interventions are considered. The debate data to be included is greatly narrowed down to a total around 262MB but the quality of it is significantly increased. Other GPT-2 applications such as AI-dungeon have been shown in the past that "a smaller amount of high-quality data is more valuable than a larger amount of low-quality data."^[50]

4.2 Testing methodology.

This work will exclusively focus on the automatic detection of generated text. To this end we will use a GPT-2 output detector released by openAI and based on huggingface's implementation of RoBERTa. This detector was trained on the outputs of the 1.5B generator model which is available online. Two different transformer sizes will be used for detection: a 110M parameter model and a larger 340M one. In this evaluation, the detectors have been trained to detect generic GPT-2 outputs and haven't been fine-tuned with outputs of the generator being evaluated.

The dataset these discriminators were trained on contained in equal parts real WebText data, random samples generated with temperature 1 without truncation and samples generated with Top-K 40 truncation. Open-AI's report on output detection found that the kind of sampling used to train the detectors greatly impacts its accuracy (See Figure 4.1). A best case scenario is hence presented to our generator since the fake transcript samples were generated using Top-P = 0.9 which has shown to be the most difficult to detect unless the discriminator was also trained with examples using Top-P sampling.

In order to ensure a fair comparison of the evaluated texts, both the GPT-2 generated text and the real text will have the same preceding text. This is to compensate for uncertainty resulting from a difference in topic. The real interventions will also be sent for detection to establish a baseline and to establish the rate of false positives.

The prompt used will contain the title of the transcript as well as the previous two interventions by other speakers. The prompt is only used to set the context for the generator and at no point is it included in the text to detect. It is worth noting that at no point does this evaluation take into account the coherency or veracity of the text generated; its sole purpose is to measure the performance of the detection solution available.

The test corpus selected consists of transcripts recorded in late 2019 and early 2020 never seen before by the generator model. These texts cover a range of different issues some of which might include new terms never seen before as the training corpus used only

Trained on:	Temp 1	96.6%	70.8%	77.2%
	Top-K = 40	50.2%	99.1%	77.3%
	Top-P = [0.8-1.0]	94.4%	96.4%	96.0%
		Temp 1	Top-K = 40	Top-P = [0.8-1.0]
		Tested on:		

Table 4.1: RoBERTa large accuracy with respect to three sampling methods (Temperature = 1 no truncation, Top-K = 40, and nucleus sampling with the Top-P sampled uniformly between 0.8 and 1.0). Subsection corresponding to training with GPT-2 1.5B parameter model outputs. The accuracy is obtained by testing 510-token test examples comprised of 5,000 samples from the WebText dataset and 5,000 samples generated by a GPT-2 model, which were not used during the training. [56]

includes transcripts until 15/05/2019. In total, 1366 samples are taken into account for this evaluation as generating a large amount of outputs can easily take upwards of twelve hours. A generation limit of 300 GPT-2 tokens per prompt is placed to accelerate the process.

All relevant samples are placed in a folder. The code for sample generation contained in the repository used until now for model training is modified to iterate through the prompts contained within a folder. The python script is called with the next command:

```
PYTHONPATH=src python ./src/generate_samples.py --model_name 1558M --prompt
→ ../HansardScrapping/prompts/context2 --top_p 0.9 --step 5 --length 1600
→ --maxlen 300 --restore_from ./checkpoint/run2
```

The context is explicitly cleared every time a new prompt is finished so as to not have cross-contamination in samples. The model outputs are then saved labeled according to which prompt they belong to so as to allow easy comparison between real and fake samples.

Prompt [61]	
Gender Pay Gap 2019-10-17	
<p>Rachel Maclean (Redditch) (Con)</p> <p>Does the Minister agree that the Conservatives have actually done more than any other Government to tackle the issue of pay inequality at work? What more is she doing to help women in the boardroom who do not earn as much as their male counterparts?</p> <p><nextspeaker></p> <p>Victoria Atkins</p> <p>I am extremely grateful to my honorable Friend, who has done so much work on women and equalities and also on menopause. [Interruption.] I note that Opposition Members are laughing and guffawing, but these issues have a real impact on women who are the lowest paid. I am delighted if it means that the Labour party is supporting gender pay gap regulations, because it was a Conservative coalition Government who introduced the regulations and a Conservative Government who brought them into force two years ago. We need to ensure that employers are treating female employees correctly and properly, and that we are tackling that in the lowest paid sectors. That is why we have the three priority sectors of retail, healthcare and education that are working to bring action plans forward to ensure that we help the lowest paid.</p> <p><nextspeaker></p>	
Real [61]	Model Generated
<p>Angela Crawley (Lanark and Hamilton East) (SNP)</p> <p>May I take this opportunity, Mr Speaker, to congratulate you on your PinkNews award, and your inspirational and outstanding speech? Trans rights are human rights.</p> <p>In the previous Queen’s Speech, the right hon. Member for Maidenhead (Mrs May) pledged to make further progress to tackle the gender pay gap, but that was noticeably absent from this week’s Queen’s Speech. Does that mean that the current Prime Minister does not want to reduce the gender pay gap?</p>	<p>Mike Amesbury (Weaver Vale) (Lab)</p> <p>We know from the London School of Economics that girls from the poorest families in India get between 13% and 18% less than boys for studying maths, which means that it becomes increasingly difficult for them to stay in school and fall in support. What will the Minister do to help these girls not just reach their full potential, but fulfil their potential to take the jobs that we will have in our post-Brexit world?</p>

Table 4.2: Comparison between real and model generated responses for a given prompt. See Appendix A for more model output examples.

4.3 Results and evaluation

The results found it can be challenging to correctly detect if a given text was generated by the model or a human when fine-tuning is used.

GPT-2 1.5B Correct Detection Rate	House of Commons	Fine-tuned GPT2 Outputs	Average Accuracy
RoBERTa base (117M)	30.0%	67.5%	48.8%
RoBERTa large (340M)	81.4%	43.6%	62.5%

Table 4.3: Fine-tuned GPT-2 1.5B detection rate on publicly available pre-trained GPT-2 output detectors

The large difference in size between the generator and discriminator models could contribute to the low accuracies found. The base version of RoBERTa seems to have problems generalising to the domain of house of commons discourse resulting in a large amount of false positives and an averaged accuracy performing under chance.

A dramatic improvement in the amount of false positives is seen when using the larger model. To observe the possible effect fine-tuning has had on the detection rate it is sufficient to average out the accuracy for both the real interventions and the GPT-2 outputs as both test sets have a 50-50 split between real and false samples.

The resulting accuracy on the house of commons test set is **62.5%**. As seen in table 4.1 OpenAI used the same RoBERTa model trained on the 1.5B version of GPT-2 to achieve up to 77.2% webtext accuracy with the sampling technique combination used.

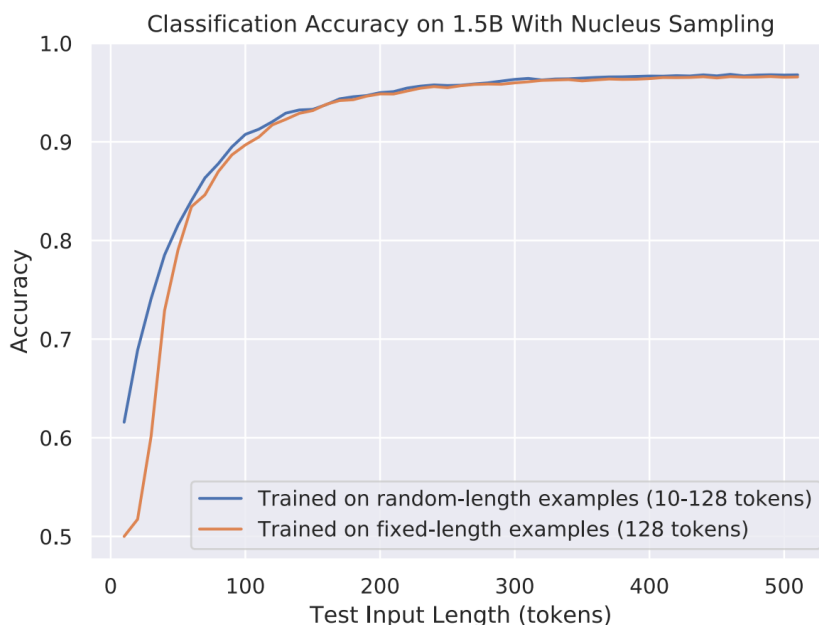


Figure 4.2: In webtext the detection accuracy becomes higher for longer text. The figure also shows that training on random-length training examples has significant positive effect on the accuracy for short length texts[56]

The fine-tuning could partially explain the decrease in accuracy observed but detector sample size might be a confounding factor. The samples used from the house-of-commons dataset are of variable length while the ones used in the webtext test set are 510 tokens long. As 100 RoBERTa tokens roughly equate to 70 words, the average input for the webtext benchmarks is around 1710 characters long given that the average English word is 4.79 characters [57]. The average speaker intervention length generated by the model is 480 characters long, much shorter than its real counterpart which averages at 880.

In release strategies and the social impacts of Language models[56] it is shown how accuracy increases with the amount of RoBERTa tokens in a sample reaching the 90% threshold at around 100 tokens (roughly 70 words or 335 characters) see Figure 4.2. The shorter test samples used might partially contribute to the lower accuracy found.

4.4 Costs and efficiency

The usage of cloud computing platforms removes the upfront costs stemming from acquisition of hardware which greatly lowers the barrier to creation of works such as this one. A DGX-1 which is a prebuilt computer offered by Nvidia used in the creation of AIDungeon, a project of which the model training section is largely similar to this work, can cost up to \$69,000. [50]

The upfront cost is instead replaced by a pay as you go scheme in which the end user gets charged for the amount of time the virtualised hardware is reserved for use. Other paying-schemes are often available at a discounted price. Google Cloud Platform, for instance, offers up to 30% continued use discount depending on the amount of time the instance is used for. Amazon web services also allows the use of on-demand compute resources but largely favours yearly hardware reservations which may be partially or entirely paid upfront for a discounted price.

Platforms incentivise these long-term plans to more accurately determine availability of resources over long periods of time. The flexibility of on demand compute may result in spikes of demand and even a resource shortage as can be occasionally observed in some regions in Google Cloud Platform.

For this particular experiment, on demand compute was the most economical option as the fine-tuning lasted just under 12 days so no discounts applied.

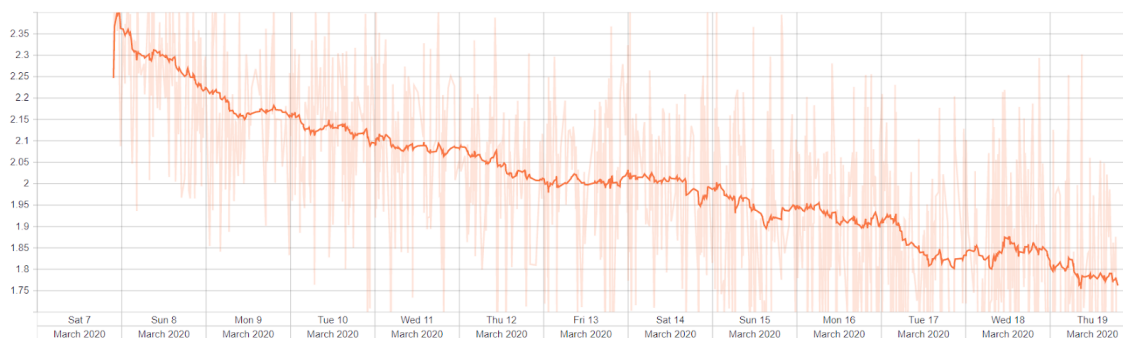


Figure 4.3: 1.5B GPT-2 model TPU fine-tuning on Hansard Transcripts. Loss over time.

Google Cloud Platform TPU Pricing

Thanks to forming part of the TFRC programme the 8 v3 TPUs did not factor into the costs. The only costs charged came from storage, virtualised cores and RAM and were absorbed by promotional Google Cloud Platform credits.

The following cost breakdowns remove all promotions and programmes for comparison. The training costs of the resulting model without taking into account the participation in the TFRC programme are as follows:

GCP Resource	Cost per hour	Hours used	Total
TPU v3-8	\$8.80	285	\$2508.00
N1-standard-2 (2 vCPU, 7,5 Memory, 100 GB disk)	\$0.094	289	\$27.17
			\$2535.17

Table 4.4: Potential costs of replicating the fine-tuning of the 1.5B GPT-2 model using TPUs in GCP.

TPUs can be faster and more economical to use than GPUs in specific cases. For a model as large as the 1.5B parameter version of GPT-2 8 v100 GPUs are needed. We will now see the price needed for a comparable fine-tuning of the model on GPUs. It is important to keep in mind that TPUs speedups highly depend on the operations being done for example: TPU V3-8 achieves more than 3× higher throughput than Tesla V100 on CNNs, while it has only about 1.5× on the original implementation of the transformer[53].

Costly Implementation issues.

The particular implementation of GPT-2 used in this work is known to suffer from some performance issues[54] on TPU. As much as a 10 times slowdown is seen with the smaller models on Collab. The root of the issue was diagnosed to be a severe slowdown in the matrix multiplication operation. As TPUs come with a minimum of 8 cores, correctly applying a parallelisation strategy is essential for performance. One possible explanation for the slowdown observed is that only one of the cores is active when performing the operations.

Debugging TPU code can only be done using the cloud-tpu-profiler plugin for Tensorboard. With these tools the developer can see the operation and performance graphs that may be used to assess bottlenecks in the model. These bottlenecks are quite frequently the origin of poor TPU performance as it can be quite challenging to diagnose the whole data pipeline including the storage medium. The operation graphs also offer a visualization of TPU compatibility as not all Tensorflow operations run in a TPU by default due to the specialised nature of the ASIC. Two tools available that can greatly help in diagnosing poor performance are activity viewers and trace visualizations. While the tpu profiler suite is quite helpful fixing these kind of performance problems, they can require in-depth knowledge on how TPUs operate to solve them which is difficult to obtain online.

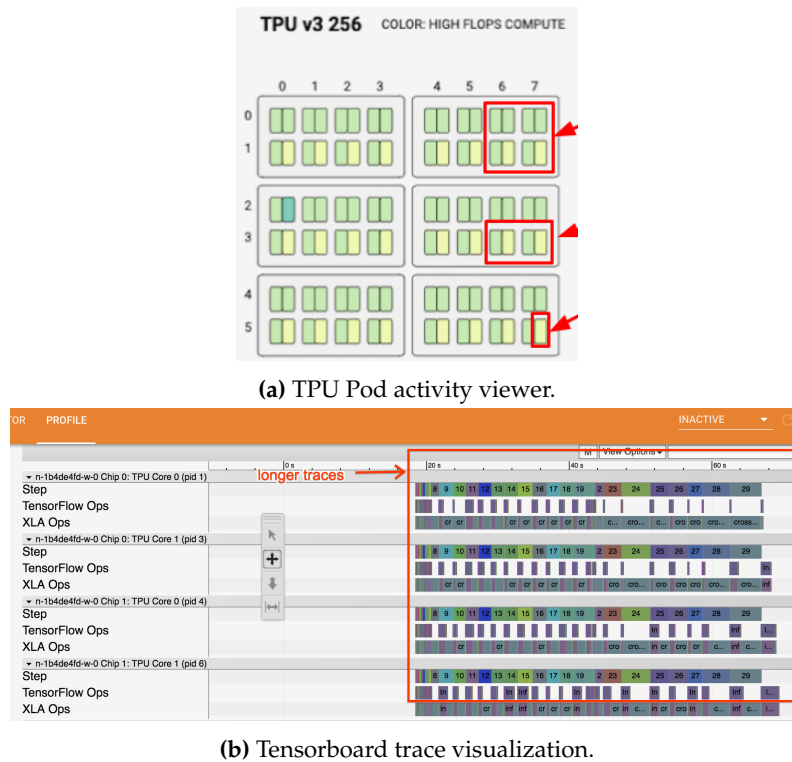


Figure 4.4: Cloud TPU diagnostic tools. [55]

Cost comparison with GPU training.

In order to conduct a rudimentary price comparison, the number of hours trained will be adjusted assuming the slowdown observed in the issue is equally prevalent on the larger model and that the throughput statistic reported above directly translates to convergence speed.

GCP Resource	Cost per hour	Hours used	Total
Nvidia Tesla V100 x8 128GB Total	\$13.888	42.75	\$593.71
8 vCPU, 30GB Memory, 100 GB disk	\$0.267	48.75	\$13.02
			\$606.73

Table 4.5: Potential costs of replicating the fine-tuning of the 1.5B GPT-2 model using GPUs in GCP.

The notable difference in total price indicates that if not for the TFRC programme, which provided the TPUs for free, the usage of 8 Tesla V100s would not only have been more economical but also faster. This does not necessarily reflect on the capabilities of dedicated compute units like TPUs. Instead, it shows a weakness of the software used as a high level of expertise is needed to adapt the models to run efficiently on such specialised compute resources.

Comparison with Amazon Web Services. (AWS)

TPUs were designed by Google specifically for its platform as such, a TPU pricing comparison cannot be made between AWS and GCP. At the time of writing Amazon Web

Services doesn't offer a comparable custom machine learning accelerator. However, it does offer instance templates with GPUs.

AWS approach to compute resources is wildly different from GCP's. In the former, only a range of preset machines can be booted up. This preset approach limits choice for users and may end up having the user pay for resources it does not need. This is definitely the case for this particular experiment as the cheapest machine with 8 Tesla V100 TPUs also has 64 virtualised CPU cores and 488GiB of RAM most of which would go unused. This lack of tailored sizing results in the following cost breakdown.

AWS Resource	Cost per hour	Hours used	Total
p3.16xlarge preset (64 vCPU, 488GiB RAM, 8x Tesla V100)	\$24.48	42.75	\$1046.52
100 GB SSD	\$0.10	Monthly cost	\$10.00
			\$1056.52

Table 4.6: Potential costs of replicating the fine-tuning of the 1.5B GPT-2 model using GPUs in AWS.

GCP's approach to resource management allows for more cost-efficient sizing. The amount of virtualised CPU cores, RAM, GPUs and storage capacities assigned are decoupled from each other and have no hard presets. A small machine can be spun up for setup with no GPUs and then resized to precisely fit the needs of the problem.

CHAPTER 5

Relation with studies

The undertaking of this project wouldn't have been possible if not for the fundamentals learnt throughout the degree. Most of the subjects coursed have contributed in some way to building the knowledge base that made this project feasible. However, a number of them stand out for their direct relevance. Namely:

- **"Sistemas de Almacenamiento y Recuperación de información" (SAR).** This subject teaches how to deal with the storage and retrieval of large quantities of data. The course pays special attention to text. This proved to be quite useful as it introduced many concepts like bag-of-words, tokenizers, regular expressions, embeddings and the practical use of dictionaries. Most of section 2.2 covering text representations builds directly on the syllabus.

The practical component of this subject introduced Python, the language used for all scripting in the project. The laboratory sessions featuring the infinite monkey theorem for text generation has the students build a rudimentary language model based on term-frequencies. The resulting application highly resembled the model covered in the section on n-gram based language models.

- **"Percepción" (PER).** Teaches the basics of pattern recognition. The main takeaway from this subject is how feature extraction is performed for different kinds of data. This subject also introduces the basics of classification models.
- **"Sistemas Inteligentes" (SIN).** The later part of this subject fully introduces the concept of machine learning for the first time in the syllabus. The introduction to discriminant functions and markov models are the main topics of interest of the subject that concern this particular work.
- **"Aprendizaje Automático" (APR).** Builds upon the knowledge obtained in the above mentioned subjects. All of the topics covered in section 2.1 are taught in detail in this course with the exception of recurrent and convolutional neural networks. The practical component has students build a support vector machine classifier as part of its laboratory tasks. This subject provides hands on experience with the implementation of basic machine learning models.
- **"Estructura de Computadores" (ETC).** This subject covers how processors work at low level as well as the interaction between components. The topics covered in ETC and its prerequisite courses provided much needed context to understand how CPU, GPU and TPU computing differ at the hardware level.

In addition to the specific knowledge attained in these courses the degree has also reinforced some transversal competencies that have been essential to the completion of this project.

- **CT-01 Comprehension and integration.** This work summarises a long history of advancements in the field of natural language processing, many of which are dispersed between several subjects. The early parts of the written component of this work attempt to integrate the advancements together in a manner where the improvements over time can be easily understood.
- **CT-02 Applied knowledge.** In the completion of this work, theoretical concepts taught at different stages of the degree have been put into practice. One such example is the creation of the corpus in section 4.1 which heavily relied on the efficient use of data structures as well as http requests.
- **CT-10 Knowledge of contemporary problems.** This work started as a direct response to a perceived issue created by state of the art language models.
- **CT-11 Constant Learning.** In the undertaking of this work, a great deal of time was spent researching further material not covered in lectures. Such two examples are the popularisation of transformers and the use of tensor processing units in a cloud environment.
- **CT-13 Practical tool selection.** Without the exploration of the viability of TPUs as a compute resource to train GPT-2 and the Tensorflow research cloud program it would have been prohibitively expensive to train the largest model available.

CHAPTER 6

Conclusions

The detector accuracies found are far too low to be deployed as an automated moderation system due to the high rate of false positives. While it might true that the shorter length texts might have contributed to the poor performance of these detectors, the reality is that false quotes or snippets of text such as short social media messages might be as disruptive as longer form messages. As such, the performance at those sequence lengths is still highly relevant and an area to improve on.

It is possible a larger transformer based language model such as the ones released since GPT-2[41] would have seen better results than the ones observed here. These larger transformers, however, are not only more costly to train but also to use for downstream tasks due to the sheer amount of parameters to calculate and the prohibitively expensive memory requirements. Increasing LM size might not be a viable strategy since the amount of text to verify generated per second might vastly exceed the capability of a moderation system based on large detection models. Even if bandwidth problems are solved by replicating instances in a flexible cloud computing environment it can still prove to be quite costly.

Instead, it may actually be beneficial to increase diversity of text type at training time to deal with more niche tasks such as this one. The results obtained might just be the product of the lack of samples from debate transcripts in the corpora forming the training set for the detectors 2.4.6. Training language models on a wider sets of document types, including shorter ones 4.2, may increase performance for downstream tasks.

6.1 Further work

Several avenues of work are available to increase the lacking performance shown by the detector models in this particular context. The first and most niche of them is to fine-tune the detector models using the outputs of the generator trained. This would likely result in an improvement for this specific context.

The second, more general, potential improvement would be to train the detectors with standard GPT-2 outputs that use Top-P sampling in addition to the data available. Adding these examples might increase general performance in situations like the one encountered in the evaluation of our model.

The generator model used could also be improved in a variety of ways, the first being training the model for an extended period of time. As it can be seen in the plot containing changes in the loss over time, the value still follows a generally downwards trend by the time it is stopped. It is possible that the quality of the text generated might be increased by resuming the training.

In addition to this further training the corpus might be updated with data more recent than 2016. The issues discussed in parliament vary with the concerns at the time. As such, one might see improvements when generating text about recent events if such an event has been included at the training stages at some point. This thought might even be relevant to the training of the original GPT-2 model as it was trained using a corpus that is dated at the latest to august 2019.

Expanding the training set past webtext to include further text types, similarly to what was done in RoBERTa, could increase performance of both detector and generator models alike. A number of parliamentary transcripts originally conceived for other NLP tasks[58] are available and could be repurposed to improve performance in niche downstream tasks like the ones covered in this work.

Bibliography

- [1] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever Language Models are Unsupervised Multitask Learners (accessed April 17, 2020) https://d4mucfpksywv.cloudfront.net/better-language-models/language_models_are_unsupervised_multitask_learners.pdf
- [2] OpenAI is an AI development and deployment company with the mission to ensure that artificial general intelligence benefits all of humanity. <https://openai.com/>
- [3] Alec Radford, Jeffrey Wu, Dario Amodei, Daniela Amodei, Jack Clark, Miles Brundage, Ilya Sutskever, Miles Brundage, Alec Radford, Jeffrey Wu, Jack Clark, Amanda Askell, David Lansky, Danny Hernandez, David Luan Better Language Models and Their Implications See <https://openai.com/blog/better-language-models/>.
- [4] OpenAI's GPT-2 detector code. (accessed June 22, 2020). See: <https://github.com/openai/gpt-2-output-dataset#finetuned-model-samples>
- [5] Hansard Corpus 1803-2005. See: <https://www.clarin.ac.uk/hansard-corpus>
- [6] UK Parliament British Hansard. Transcripts publicly available: <https://hansard.parliament.uk/>.
- [7] Commons Chamber, Coronavirus Volume 672, 03 March 2020. Full transcript available: <https://hansard.parliament.uk/Commons/2020-03-03/debates/4913849A-4E73-4B43-BFA3-16CD4A35BE66/Coronavirus>.
- [8] Parliament Archives Publishing Images, Quotations and Citations, (accessed April 17, 2020). See: <https://archives.parliament.uk/our-services/publishing-images/>
- [9] What is Ajax? (accessed June 3, 2019). See explanation and examples: https://www.w3schools.com/whatis/whatis_ajax.asp
- [10] Beautiful Soup (accessed June 13, 2020). See official documentation: <https://www.crummy.com/software/BeautifulSoup/bs4/doc/>
- [11] Selenium Browser Automation Tool (accessed June 3, 2019). See product page: <http://www.selenium.dev/>
- [12] Selenium Webdriver (accessed June 13, 2020). See definition: <https://www.selenium.dev/documentation/en/webdriver/>
- [13] ChromeDriver - WebDriver for Chrome (accessed June 3, 2019). See: <http://chromedriver.chromium.org/>

- [14] Github issue asking for the original Webtext dataset to be released (accessed June 13, 2020). See: <https://github.com/openai/gpt-2/issues/24>
- [15] Christopher M. Bishop. *Pattern Recognition and Machine Learning* springer, 2006.
- [16] Nilsson, Nils J. *Introduction to machine learning: An early draft of a proposed textbook* 1996.
- [17] Huval, Brody, et al An empirical evaluation of deep learning on highway driving *arXiv preprint arXiv:1504.01716*, (2015).
- [18] Alexander Mordvintsev, Christopher Olah, Mike Tyka Inceptionism: Going Deeper into Neural Networks (June 17, 2015) See <https://ai.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html>.
- [19] Saugat Bhattarai, June 22, 2018 See: <https://saugatbhattarai.com.np/what-is-gradient-descent-in-machine-learning/>
- [20] Irene Solaiman, Miles Brundage, Jack Clark, Amanda Askell, Ariel Herbert-Voss, Jeff Wu, Alec Radford, Gretchen Krueger, Jong Wook Kim, Sarah Kreps, Miles McCain, Alex Newhouse, Jason Blazakis, Kris McGuffie, Jasmine Wang Release Strategies and the Social Impacts of Language Models *arXiv:1908.09203v2*, 13 November 2019
- [21] Brownlee, Jason. Why Training a Neural Network Is Hard (March 1, 2019) <https://machinelearningmastery.com/why-training-a-neural-network-is-hard/>
- [22] Ruder, Sebastian. An overview of gradient descent optimization algorithms *arXiv preprint arXiv:1609.04747*, (2016).
- [23] Kingma, Diederik P., and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, (2014).
- [24] Bengio, Yoshua. Rmsprop and equilibrated adaptive learning rates for nonconvex optimization. *corr abs/1502.04390*, (2015).
- [25] Stanford CS231n Convolutional Neural Networks for Visual Recognition (Accessed June 13, 2020) <https://cs231n.github.io/neural-networks-1/#classifier>
- [26] Rosenblatt, Frank. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review* 65.6 (1958): 386., (1958).
- [27] Werbos, Paul. Beyond regression: new tools for prediction and analysis in the behavioral sciences. *Ph. D. dissertation, Harvard University*, (1974).
- [28] Hochreiter, Sepp, and Jürgen Schmidhuber. Long short-term memory. *Neural computation* 9.8 (1997): 1735-1780, (1997)
- [29] Pascanu, Razvan, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. *International conference on machine learning.*, (2013).
- [30] Mikolov, Tomáš. Slides on Distributed Representations for NLP. *Machine Learning Prague (2016)* <https://www.slideshare.net/mlprague/tom-mikolov-distributed-representations-for-nlp>
- [31] Gage, Philip. A new algorithm for data compression. *C Users Journal* 12.2, (1994): 23-38
- [32] Bengio, Yoshua, et al A neural probabilistic language model. *Journal of machine learning research*, 3.Feb (2003): 1137-1155.

- [33] Cavaioni, Michele (Mar 15, 2018) DeepLearning series: Sequence-to-Sequence Architectures <https://medium.com/machine-learning-bites/deeplearning-series-sequence-to-sequence-architectures-4c4ca89e5654>.
- [34] Kelleher, John & Dobnik, Simon. *What is not where: the challenge of integrating spatial representations into deep learning architectures*. (2017)
- [35] Dauphin, Yann N., et al. Language modeling with gated convolutional networks. *International conference on machine learning*. (2017)
- [36] Bressler, Peter (Dec 11, 2018) Building a convolutional neural network for natural language processing. <https://towardsdatascience.com/how-to-build-a-gated-convolutional-neural-network-gcnn-for-natural-language-processing-nlp-5ba3ee730bfb>.
- [37] Holtzman, Ari, et al. The curious case of neural text degeneration. *arXiv preprint arXiv:1904.09751*, (2019).
- [38] Vaswani, Ashish, et al. Attention is all you need. *Advances in neural information processing systems*. , (2017).
- [39] Iranzo Sánchez, Javier. Online Multilingual Neural Machine Translation. , (2019).
- [40] Christopher Manning, Ashish Vaswani Anna Huang Stanford CS224N: NLP with Deep Learning | Winter 2019 | Lecture 14 – Transformers and Self-Attention <https://www.youtube.com/watch?v=5vcj8kSwBCY>
- [41] Brown, Tom B., et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, (2020).
- [42] Alammar, Jay The Illustrated GPT-2 (Visualizing Transformer Language Models) (Accessed 19-Jun-2020) See: <http://jalammar.github.io/illustrated-gpt2/>
- [43] Devlin, Jacob, et al. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, (2018)
- [44] Liu, Yinhan, et al. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, (2019)
- [45] Yukun Zhu, Ryan Kiros, Richard Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. *arXiv preprint arXiv:1506.06724*, (2015)
- [46] Aaron Gokaslan, Vanya Cohen OpenWebText Corpus. (2019) <https://skylion007.github.io/OpenWebTextCorpus/>
- [47] Nagel, Sebastian. CC-News Corpus. (2016) <https://commoncrawl.org/2016/10/news-dataset-available/>
- [48] Trieu H Trinh and Quoc V Le. A simple method for commonsense reasoning *arXiv preprint arXiv:1806.02847*, (2018)
- [49] Sundermeyer, Martin, et al. Comparison of feedforward and recurrent neural network language models. *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, IEEE, 2013.

- [50] Boog, Jason (Dec 9, 2019) How the Creator of AI Dungeon 2 Used GPT-2 To Create Neverending Adventure Games. <https://towardsdatascience.com/the-creator-of-ai-dungeon-2-shares-gpt-2-finetuning-advice-e5800df407c9>.
- [51] Google Cloud TPU Documentation: System Architecture See: <https://cloud.google.com/tpu/docs/system-architecture>.
- [52] Alcorn, Paul (May 11, 2017) Nvidia Infuses DGX-1 With Volta, Eight V100s In A Single Chassis <https://www.tomshardware.com/uk/news/nvidia-volta-v100-dgx-1-hgx-1,34380.html>.
- [53] Wang, Yuxin, et al. Performance and Power Evaluation of AI Accelerators for Training Deep Learning Models. *arXiv preprint arXiv:1909.06842*, (2019).
- [54] Known GPT-2 TPU performance problems. See: <https://github.com/shawwn/gpt-2/issues/5>.
- [55] Using Cloud TPU Tools. See: <https://cloud.google.com/tpu/docs/cloud-tpu-tools>
- [56] Solaiman, Irene, et al. Release strategies and the social impacts of language models. *arXiv preprint arXiv:1908.09203*, (2019).
- [57] Norvig, Peter English Letter Frequency Counts: Mayzner Revisited <http://norvig.com/mayzner.html>.
- [58] Iranzo-Sánchez, Javier et al. Europarl-ST: A Multilingual Corpus For Speech Translation Of Parliamentary Debates. *ArXiv abs/1911.03167*, (2019).
- [59] Commons Chamber, Northern Ireland Protocol: UK Approach. Volume 676, 20 May 2020. Full transcript available: <https://hansard.parliament.uk/Commons/2020-05-20/debates/07EBAEC6-2593-4969-9A2C-C115FBEEDBBF/NorthernIrelandProtocolUKApproach>.
- [60] Commons Chamber, Special Educational Needs and Disability. Volume 672, 02 March 2020. Full transcript available: <https://hansard.parliament.uk/Commons/2020-03-02/debates/E795037E-80AD-4557-8724-836FDA6C074E/SpecialEducationalNeedsAndDisability>.
- [61] Commons Chamber, Gender Pay Gap. Volume 666, 17 October 2019. Full transcript available: <https://hansard.parliament.uk/Commons/2019-10-17/debates/FAE12CED-EA50-4737-A3F8-B5DD7B69E8FF/GenderPayGap>.
- [62] Commons Chamber, Drug Consumption Room (Glasgow). Volume 663, 24 July 2019. Full transcript available: [https://hansard.parliament.uk/Commons/2019-07-24/debates/AB11514C-0FB9-4390-A1F4-9EBE489ADA1E/DrugConsumptionRoom\(Glasgow\)](https://hansard.parliament.uk/Commons/2019-07-24/debates/AB11514C-0FB9-4390-A1F4-9EBE489ADA1E/DrugConsumptionRoom(Glasgow)).

APPENDIX A

Additional Language Model Samples

Prompt [62]	
Scotland 2019-07-24	
Alison Thewliss (Glasgow Central) (SNP)	
5. What recent discussions he has had with the Home Secretary on the potential merits of establishing a medically supervised drug consumption room in Glasgow. <nextspeaker>	
The Secretary of State for Scotland (David Mundell)	
The causes of drug misuse are complex and need a range of policy responses. I am aware that the Home Secretary has offered to meet Scottish Government Public Health Minister Joe FitzPatrick to discuss a broad range of issues around the tragic matter of drug-related deaths in Scotland. <nextspeaker>	
Real [62]	Model Generated
Alison Thewliss I am glad that the Home Secretary is finally going to meet the Scottish Government on this. When NHS Greater Glasgow and Clyde published its proposals for a supervised drug consumption in 2016, the number of drug-related deaths stood at 257; last year, it was 394. So I ask the Secretary of State for Scotland, how many people would still be alive in the NHS Greater Glasgow and Clyde area if the Home Office had not blocked, for ideological reasons, drug consumption rooms in Glasgow?	Alison Thewliss The UK Government have spent £2.4 billion on their failed war on drugs, which has resulted in the death of 2,000 people and 1,000 babies. Will the UK Government now support medically supervised drug consumption rooms in Glasgow?

Table A.1: Comparison between real and model generated responses for a given prompt.

Prompt[7]	
Coronavirus 2020-03-03	
<p>Matt Hancock</p> <p>We have extensive stockpiles of personal protective equipment. We are not distributing that at this moment because we have to distribute it at the right time. Each individual case can be dealt with at the moment, because they are relatively few, by those who are expert in using that kit. Of course community staff, as well as primary care staff and hospital staff, will be involved in the distribution of that equipment as and when appropriate.</p> <p><nextspeaker></p> <p>Robert Largan (High Peak) (Con)</p> <p>Last week there was a confirmed coronavirus case in Buxton, which led to the temporary closure of a medical centre and a school. Unfortunately, several national newspapers inaccurately reported that Buxton was a town on lockdown. Several local hotels have now reported booking cancellations as a result. Does the Secretary of State agree that the situation calls for responsible journalism and calm reporting of the facts? Will he join me in encouraging people to visit Buxton, Britain's best spa town?</p> <p><nextspeaker></p>	
Real [7]	Model Generated
<p>Matt Hancock</p> <p>Yes. I do not have to check with the chief medical officer before telling you, Mr Speaker, that I love going to Buxton, which is a great place to visit. My honorable Friend makes a serious point. As I said in my statement, there is scientific advice against moving too soon or over-reacting, as there is against moving too slowly or not reacting strongly enough. We need to take the measures that are necessary to protect the public. On taking measures that do not protect the public, the advice is that all of us in a position of responsibility whose communications are heard widely, whether we are Members of this House or members of the media, have a duty of responsibility, because how this is communicated will have a direct impact on how well we as a country cope with this outbreak.</p>	<p>Matt Hancock</p> <p>Yes. My honorable Friend is a keen sportsman, and it would be great if he could get his summer on, but if he reports the facts, we will ensure that the facts are accurate. We need to get the facts straight.</p>

Table A.2: Comparison between real and model generated responses for a given prompt.

Prompt [59]	
Northern Ireland Protocol_ UK Approach 2020-05-20	
<p>Mr William Wragg (Hazel Grove) (Con) [V] The way the UK and the EU seek to address “Ireland’s unique geographic situation” in the negotiations could have constitutional and practical implications for Northern Ireland’s status within the UK. Could my right honorable Friend reassure me that he can square that circle, or is it, on the current trajectory of the talks, an impossible objective? <nextspeaker></p>	
<p>Michael Gove That circle can be squared using an exercise of what I believe in the EU is known as variable geometry. The truth of course is that Northern Ireland’s position within the UK is constitutionally secure and unchanged.</p>	
<p>Sir Jeffrey M. Donaldson (Lagan Valley) (DUP) [V] The Minister will be aware that we voted against the withdrawal agreement because of the Northern Ireland protocol, but we welcome the clarity that this statement brings—that Northern Ireland will remain part of the UK customs territory, that there will be no new customs infrastructure, that there will be no tariffs on goods flowing between Great Britain and Northern Ireland and that Northern Ireland businesses will have unfettered access to the Great Britain market. Will the Minister and his team continue to work with us and the business community in Northern Ireland to ensure that these matters are taken forward and that Northern Ireland remains an integral part of the UK single market? <nextspeaker></p>	
Real [59]	Model Generated
<p>Michael Gove Yes, we absolutely will. Our whole approach is about making sure that the protocol, which of course was unwelcome in many quarters in Northern Ireland, is implemented now that it is law, but in a way that goes with the grain of Northern Ireland opinion and reflects the interests of Northern Ireland’s peoples, whom the right hon. Gentleman so eloquently defends.</p>	<p>Michael Gove My honorable Friend is entirely right. Those are the maximum achievable objectives and the withdrawal agreement allows for them to be reached, and in my conversations with the Taoiseach, I have been clear about the Northern Ireland-Ireland aspect of that. We have been clear in our objective to keep Northern Ireland in the single market and the customs union, which is in the economic interests of both sides. We intend to pursue that objective and the Bill will do precisely that.</p>

Table A.3: Comparison between real and model generated responses for a given prompt.

Prompt [60]	
Education 2020-03-02	
<p>The Parliamentary Under-Secretary of State for Education (Vicky Ford) Every child should receive a world-class education, no matter what their needs. That is why we are investing £7.2 billion this coming year to support those with the most complex needs—an increase of £780 million. Local performance varies across the country, so we are reviewing the entire SEND system and working closely with stakeholders and parents. <nextspeaker></p> <p>Fleur Anderson I am glad to hear that the system is being reviewed. Cuts to council funding for special educational needs and disability services mean that children in Wandsworth are waiting too long for a diagnosis and for an education, health and care plan. Then, too many do not receive the support they need that is outlined in that plan. This common experience of parents and children was backed up by a recent Ofsted report that said that Wandsworth’s EHC plans were of poor quality and that there were significant concerns. There is a cost to cuts. Will the Secretary of State ensure that there is significant additional funding for councils in the Budget for special educational needs and disability services? <nextspeaker></p>	
Real [60]	Model Generated
<p>Vicky Ford The London Borough of Wandsworth will receive £47.8 million in high-needs funding in the next year, which is an increase of 8.6%. The performance of local areas in producing education, health and care plans is variable, but some 30 areas do get more than 90% of plans done within the 20-week period which, I note, is a reduction from the 26-week period under the previous Labour Government. Performance does vary across the country. Where it is not good enough, we support and challenge local areas to improve.</p>	<p>Vicky Ford It is incredibly important that young people with special educational needs get the support that they need, and the national funding rate is increasing by £2.5 billion by 2019-20. We have also announced £250 million in flexible capital funding, which local authorities can use to improve their children’s special educational provision.</p>

Table A.4: Comparison between real and model generated responses for a given prompt.