Benha University
Faculty of Engineering at Shoubra
Electrical Engineering Department

GRADUATION PROJECT
HUMANLY INTERACTIVE VR

Team Members

Ehsan Ahmed El-Saeed
Laila Mahmoud Abd El-Rahim
Muhammad Mustafa Abd El-Fattah
Raafat Ahmed Muhammad
Toqa Magdy Fouad
Kirollos Atef Ibrahim
Mina Latif Lotfy
Omar Khaled Abd El-Hafiz
Soad Samir Fawzy

Supervised By:

Dr. Shimaa Ibrahim

ACKNOWLEDGEMENT

We would like to express our deep and sincere gratitude to our supervisor, Dr. Shimaa Ibrahim, for providing invaluable guidance, comments and suggestions throughout this project. It was a great pleasure to study under her guidance and support. We are really grateful that we managed to complete the project within the given time. It could not be completed without the effort and cooperation of our team members. We would also like to thank anyone responded with support and willingness to give us time and offered help.

CONTENTS

ABSTRACT

Presentation skills are always the first impression of your work and could either leave it well received and appreciated or underrated and more prone to being dismissed, hence they need to be as reliable and confident as possible. In our project, we are trying to create a platform that is easily available in which the presenter could practice their presentation skills and receive the necessary feedback to work on their weak points.

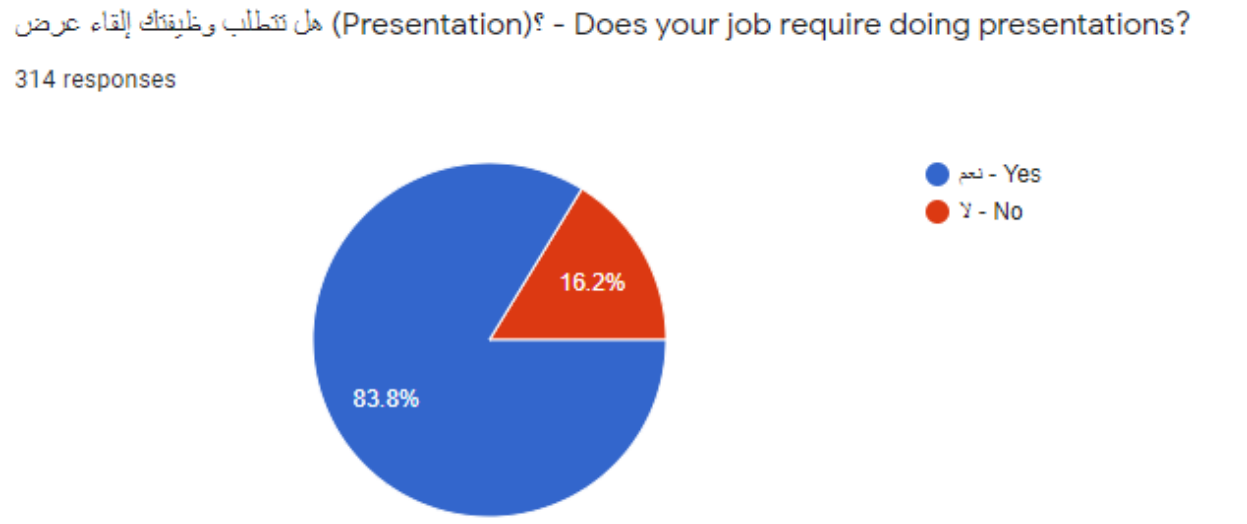
Our project consists of a virtual scene of a hall containing some characters that can react to the user's speech, either by asking questions regarding said speech, reacting to the clarity of speech by making satisfied or dissatisfied facial expressions.

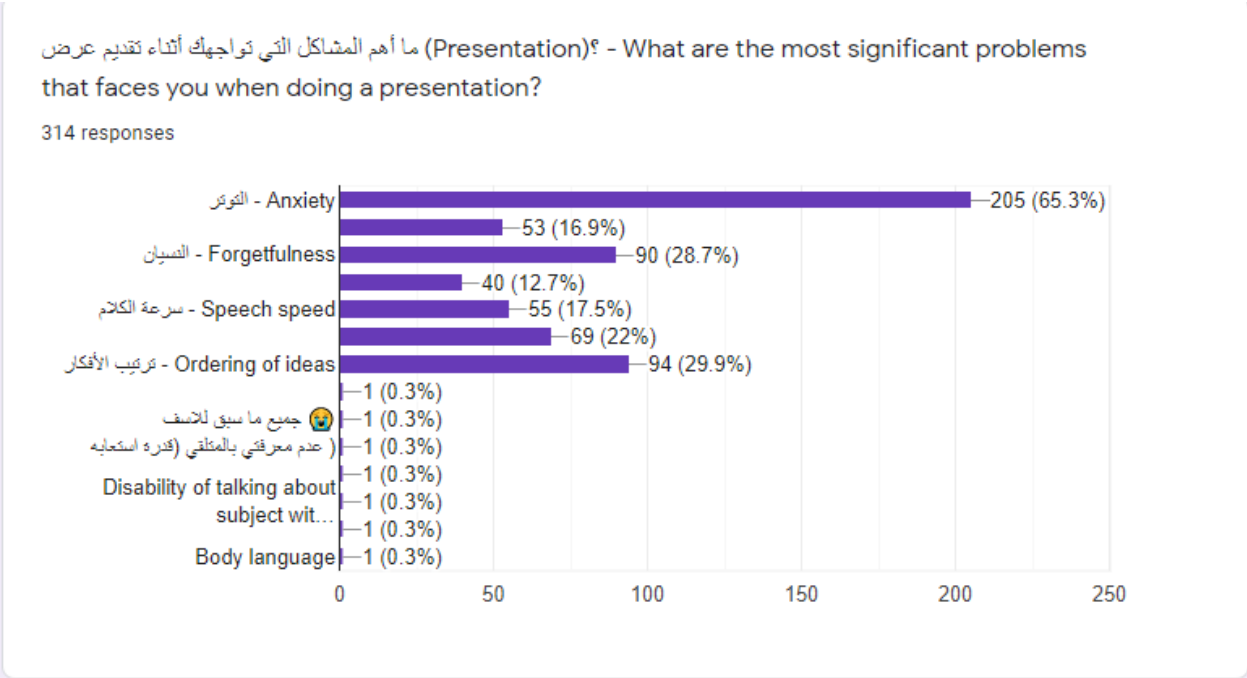
Our objective is mainly achieved by using NLP methods to understand the user's speech and constructing questions that are relevant to the presenter's speech, and animating the characters to either react appropriately to the given speech or by asking relevant questions, hopefully making the user experience most aspects of a real presentation and finding a way to react appropriately to situations.

MOTIVATION

After dealing with different situations and aspects in life, we found that presentation skills were essential to achieving most goals, whether it a new idea, a new job, a graduation project. Many people have exceptional ideas that may not make it through due to bad presentation of the idea. Due to this we decided to do a survey to find out how many people actually needed presentation skills and the difficulties that people face in presentations and talking in public and the results of the survey are as follow.

The Survey Results:





According to the responses we got, we found that 83.8% of people's jobs required doing presentations. We also inquired about the challenges faced and 65.3% had anxiety issues, 29.9% with ordering of ideas, 28.7% dealt with forgetfulness, 22% with distractions, 17.5% with speech speed, 16.9 with not finding a place to practice, 12.7% can't finish at specified time.
Also we found that 91.5% of responders would find it helpful if they were offered a virtual platform to practice their presentation beforehand.
After studying the responses, we found that the best way to get over these issues is by providing something that can help people practice easily and give them the feedback they needed to gain confidence and work on their errors.

CHAPTER 1
INTRODUCTION

Our project is a VR platform that can help in solving the problems that most people face during giving a presentation by allowing them to practice on their own and get the feedback they need to improve their presentation.

The project is mainly a scene consisting of a hall filled with characters that react to the presentation using facial expressions or ask questions relevant to the presentation. The rate of speed of the presenter's speech is also calculated and displayed in the hall to help him find a pace that is clear and informative.

The project consists of mainly five phases, Speech-To-Text, NLP, Question Generation, Text-To-Speech and Graphics.

The presenter starts the application and starts giving the presentation, here the speech-to-text phase starts and doesn't end unless the presentation ends, for speech-to-text phase we use Google's API, where the speech the presenter gives is converted to text in real-time. The words are counted and the speech's rate is calculated over intervals of time, and the feedback of the performance is updated each interval as a test in the hall and also the characters make facial expressions to reflect if they are satisfied with the rate or if it's too slow or too fast, to train the user how to deal with momentary disapproval.

The text we extracted from the speech is passed every interval of time to a model called the UniLM using an API that performs the NLP and Question Generation phases. The model analyzes the text and understands the context and generates relevant questions as an output. The questions generated are then returned from the API to the application.

Every interval of time, a question is chosen from the questions generated from the model and the text-to-speech phase starts where we pass the text to Google's Text-to-Speech API, that returns an audio file of the question.

We choose a character randomly to voice the question and an appropriate voice is chosen to reflect the gender of the character and the character raises its hand to grab the presenter's attention and asks the question. Here the user is trained on being interrupted, and how to return to the same point he left in the presentation.
The graphics phase is in progress in parallel with all other phases as the scene is created in the graphics phase, the feedback of the rate of speech is animated over the facial expressions of the characters so here is was integrated with the speech to text phase. The characters are also animated to move and raise their hands and ask the question so here the graphics phase was integrated with the text-to speech phase.

CHAPTER 2
VIRTUAL REALITY (VR)

2.1 Introduction

Virtual Reality (VR) is the use of computer technology to create a simulated environment. Unlike traditional user interfaces, VR places the user inside an experience. Instead of viewing a screen in front of them, users are immersed and able to interact with 3D worlds.
The idea of VR is quite old as the first headset was created in 1968, But as computing power is more advanced now more than ever many companies are making games and application specifically for VR and it is a profitable industry.

2.2 Why VR?

As VR puts the user in immersive 3D world and incorporates all the user's senses, it seemed to be the most appropriate environment for users to practice their speech in to be as replicative to real life as possible.
The target is to create a room similar to a lecture hall with audience that the user can interact with via his speech and they give a feedback on the user's speech or lecture.

2.3 Headsets

There are many headsets available for VR applications; they range in size, price and capabilities like HTC vive, Oculus Rift, Samsung VR and Google Cardboard. We can separate head into two categories based on their features.

- **Oculus/Samsung/HTC**
  They are expensive in the range of 500$ or higher, they offer more features like Degrees of Freedom that the user is allowed to move in. They are also able to detect the user's position in the room via sensors scattered in the surrounding environment. They also must be connected to a computer for computing and the image is transferred to the user's headset. Fig 2.1 shows a simple room setup for these kind of headsets.

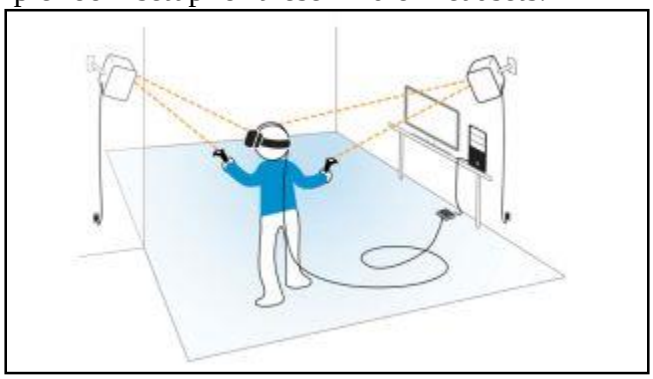


Figure 2.1 Room Setup

- **Google Cardboard**

It is very cheap in the range of 10$ or less. It does not have any of the features discussed above from detecting the user's position. It's only one feature is the simplicity it offers compared to the much more expensive options. It is a holder for a mobile device and blocks light from the environment giving the desired feeling of VR. Fig. 2.2 shows Google Cardboard.

Figure 2.2 Google Cardboard / Alternatives

Considering the simplicity and prices of cardboards, it seemed to be the appropriate option to go with for our application since everyone could get it without paying so much a gadget they might not use much.

2.4 VR Development Issues

Although VR seems a great and fun experience, it also has it issues as the user has to wear the head for a long period, which can cause some discomfort to users. VR issues are eyestrain, nausea and motion sickness since the senses are not used to that kind of new experience.
It is the job of developers to handle those issues and make the application as eye pleasing as possible with no lagging or frame rate fluctuations, which are the main reasons of the discomforts, stated previously. Suitable frame rate is usually 60+ FPS and the quality of the objects should be clear as possible for the most pleasing experience.

CHAPTER 3
TEXT TO SPEECH AND TEXT PROCESSING

3.1 Introduction

3.1.1 Prologue

It has been always the case that new ways to communicate with our computers are appearing everyday every hour. In the past people used to interact with their computers mainly through keyboards and text-based systems Zen came about the revolution of graphical user interfaces and the advent of computer mice, after that the Interactions With our computing devices became dominated with touch interfaces. Now to accommodate people with disabilities and to devise new ways to interact with our devices in scenario in scenarios when Traditional methods cannot be visible there has been a rise in Voice interactions with all sorts of electronic devices with the proliferation of digital assistants such as Apple's Siri and Google Assistant it's only evident that this mode of interaction Is popular with clients.

3.1.2 Why Speech-To-Text

Before popularisation of voice commands there have been developments in inferring meanings from text through Natural Language Processing (NLP) and Natural Language Understanding (NLU) and those were primarily text based and to build upon these technologies, it only made sense to convert voice or speech to take it to its text representation and run the established algorithms to infer what a user wants to make a better service. There have been other methodologies that work on the Voice or the wavy nature of speech, if so to speak, but they haven't been as successful as a text -based methods so converting speech into text then processing the resulting text makes much more sense in this regard.

3.1.3 History of Speech-To-Text

With the ever-increasing power to perform computations, the advances in artificial intelligence, especially in the Machine Learning subfield, and the abundance of speech data, many of which accompanied with its text representation, there has been a huge boon to the capabilities of computers to recognise human speech, and recently they have become as efficient as humans in this regard.

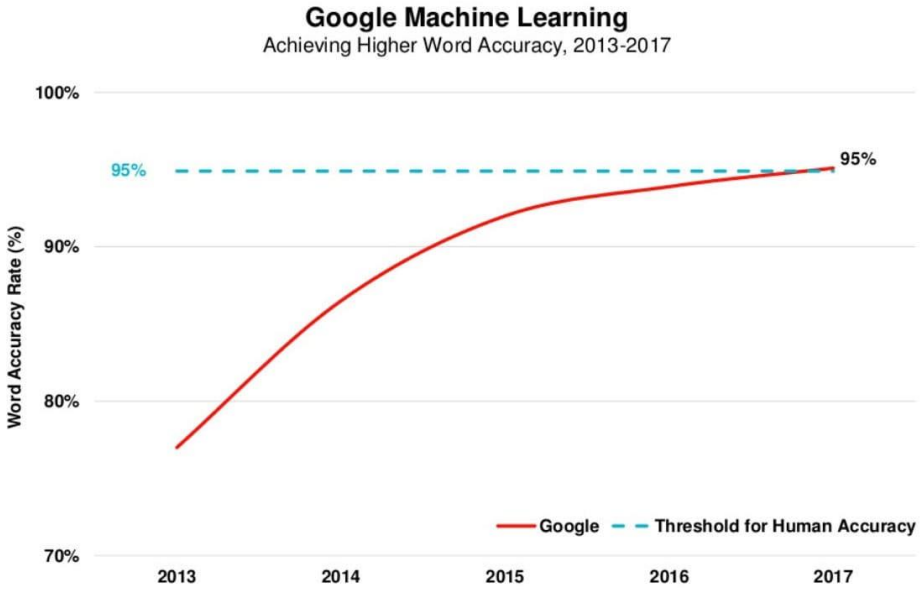


Figure 1 Speech Accuracy vs Years

Milestones

1950s to 1960s

Bell labs first designed Audrey Systems: a speech recognition device focused on recognising numbers. Ten later, in 1960s, IBM came with the Shoebox system; it could understand and respond to 16 words in English.

1970s

Several advancements in this field were made in this decade courtesy of none other than US Department of Defence's (DoD) DARPA, the original progenitors of Internet. They created a system called Speech Understanding Research (SUR); its child system called Harpy at Carnegie-Mellon University was able to understand 1,000 English words, the equivalent of a 3-year-old's vocabulary.

1980s

A great growth in the vocabulary understood by recognition systems from thousands of words to tens of thousands of words. A breakthrough in statistical algorithms known as Hidden Markov Models were of tantamount importance towards the realisation of these advancements. HMMs work by estimating the probability of a sound utterance being a word instead of using fixed sound patterns to map to words directly in a deterministic way.

1990s

Speech Recognition became popular with the masses in the 90s because Personal Computers had faster processors capable of running recognition systems like Dragon Dictate which became widely used. Also, a accompany called BellSouth was pivotal in popularising such systems through the introduction of Voice Portal (VAL) which was a dial-in phone service based on Speech Recognition. This system gave to myriad of phone tree services currently in use today.

2000s

By the time it was 2001, Speech Recognition could attain around 80% in accuracy; however, there was not much development in the field till the arrival of Google Voice Search in the latter end of the decade. Due to the popularity of Google, this service was accessed by millions of people, which gave valuable datasets to be worked on and the needed processing power to serve the users' requests were offloaded to its datacentres, therefore allowing a higher accuracy, thus quality, service.

2010s

In 2011 Apple launched its Siri App, much like Google Voice Search, now known as Google Assistant, it came like an App that is a point that helped in making both popular in the age of handheld devices. Then came Amazon's Alexa, its competitor Google Home, and Microsoft's Cortana. With the rise of Machine Learning techniques and ever-greater datasets, accuracy became higher and higher. In 2016 IBM achieved a word error rate of 6.9%; Microsoft beat it in 2017 and achieved an error rate of 5.9%; however, Google now reigns supreme with its error rate of just 5.5%, which is the rate of human error in recognising speech.

1952	Invention	A team at Bell Labs designs the Audrey, a machine capable of understanding spoken digits.[1]
1962	Demonstration	IBM demonstrates the Shoebox, a machine that can understand up to 16 spoken words in English, at the 1962 Seattle World's Fair.[4]
1971	Invention	IBM invents the Automatic Call Identification system, enabling engineers to talk to and receive spoken answers from a device.[5]

1971–1976	Program	DARPA funds five years of speech recognition research with the goal of ending up with a machine capable of understanding a minimum of 1,000 words. The program led to the creation of the Harpy by Carnegie Mellon, a machine capable of understanding 1,011 words.[1]
Early 1980s	Technique	The hidden Markov model begins to be used in speech recognition systems, allowing machines to more accurately recognize speech by predicting the probability of unknown sounds being words.[1]
Mid 1980s	Invention	IBM begins work on the Tangora, a machine that would be able to recognize 20,000 spoken words by the mid 1980s.[5]
1987	Invention	The invention of the World of Wonder's Julie Doll, a toy children could train to respond to their voice, brings speech recognition technology to the home.[1]
1990	Invention	Dragon launches Dragon Dictate, the first speech recognition product for consumers.[1]
1993	Invention	Speakable items, the first built-in speech recognition and voice enabled control software for Apple computers.
1993	Invention	Sphinx-II, the first large-vocabulary continuous speech recognition system, is invented by Xuedong Huang.[6]
1996	Invention	IBM launches the MedSpeak, the first commercial product capable of recognizing continuous speech.[5]
2002	Application	Microsoft integrates speech recognition into their Office products.[7]
2006	Application	The National Security Agency begins using speech recognition to isolate keywords when analyzing recorded conversations.[8]
2007	Application	Microsoft releases Windows Vista, the first version of Windows to incorporate speech recognition.[9]
2007	Invention	Google introduces GOOG-411, a telephone-based directory service. This will serve as a foundation for the company's future Voice Search product.[10]
2008	Application	Google launches the Voice Search app for the iPhone, bringing speech recognition technology to mobile devices.[11]

2011	Invention	Apple announces Siri, a digital personal assistant. In addition to being able to recognize speech, Siri is able to understand the meaning of what it is told and take appropriate action.[12]
2014	Application	Microsoft announces Cortana, a digital personal assistant similar to Siri.
2014	Invention	Amazon announces the Echo, a voice-controlled speaker. The Echo is powered by Alexa, a digital personal assistant similar to Siri and Cortana. While Siri and Cortana are not the most important features of the devices on which they run, the Echo is dedicated to Alexa.

Table 1 Showing The timeline of advancements in Speech Recognition Systems

3.2 How Speech Recognition Systems Work

The first step in designing any algorithm or processing pipeline, whether it was deterministic or statistical, is the choice of the input features. In the field of Recognition two systems are mainly used Linear Prediction Cepstral Coefficients (LPCC) and Mel Frequency Cepstral Coefficients (MFCC). The other important aspect is the choice of the algorithm itself that will process the input, and we have plenty of choices in this regard: the aforementioned HMMs, Artificial Neural Networks (ANNs) and LSTM Networks. In practice a combination of HMMs and ANNs works well and achieves high accuracy.

3.2.1 MFCC

When a human speaks, the sounds he makes are filtered through the shape of his larynx, teeth and tongue. This leads two iPhone M being represented is a short time scales by a set of Cesptra, a changing in the frequency representation of the voice, Answer isn't much difference between How the human brain perceives neighboring frequencies as the pitch of the sound increases. These features are replicated by MFCC to get the phoneme In a four form suitable for a computer algorithm this exploits the fact that a word just string of phonemes and by getting those we infer a word to a high degree of confidence. MFCC was developed in a set of experiments to understand how humans speak and understand speech.
Steps to Compute MFCC:

- Cut the sound into tiny frames each typically is 20ms to 40ms.
- Compute the Fourier or rather the Short Time Fourier Transform.
- Transform the Spectrum to the Mel-Scale. $Mel(f) = 2595 \log\left(1 + \frac{f}{700}\right)$
- Calculate the Log of the result
- Do another Fourier Transform or a Discrete Cosine Transform.

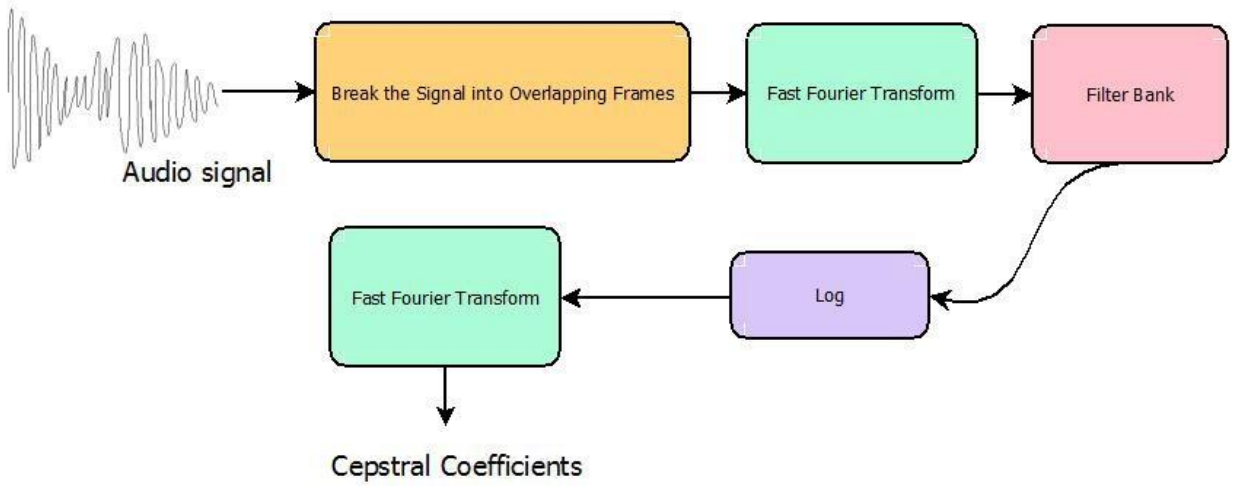


Figure 2 A block diagram for an algorithm to compute MFCC

3.2.2 Hidden Markov Models (HMMs)

Markov Chains

Let us begin with first identifying what is a Markov State and a Markov Chain. In a random process in which the random variable exists in a certain value or a *state* if it changes states and the probability it settles in a certain state is only given by the previous state *irrespective of its history or how it ended up there* is called a Markov State. A Markov Chain is just a series of successive states

$$\mathbb{P}(X_t = j | X_1 = i_1, \ldots, X_{t-1} = i_{t-1}) = \mathbb{P}(X_t = j | X_{t-1} = i_{t-1})$$

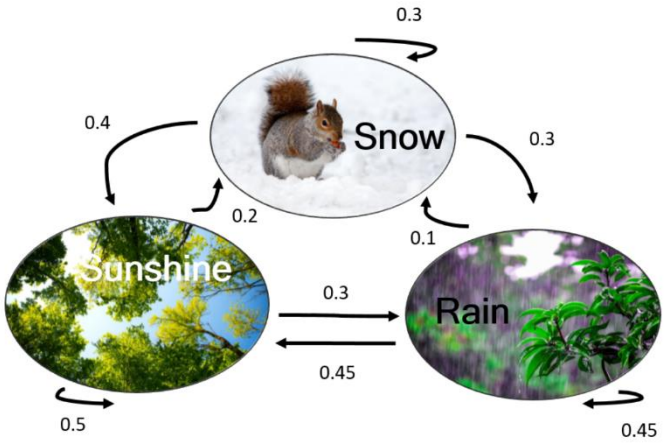


Figure 3 Markov Chain example

Hidden Markov Models

Hidden Markov Models are just random or probabilistic processes which its underlaying Markov Chain is *hidden* from us and we must come up with a way to reconstruct them from the observations we make. There exist many algorithms to reconstruct them one of note is **Viterbi algorithm**.

HMMs for the Task of Speech Recognition

In Speech Recognition we want to estimate the word **W** given utterances **O** $\hat{W} = \arg\max_W \{P(W|O)\}$

Using Bayes Theorem, we can reformulate the problem as $\hat{W} = \arg\max_W \{P(W|O)\} = \arg\max_W \{P(O|W) \times P(W)\}$ This splits the task into two components **P(O|W)**, which is called the acoustic model i.e. how likely an utterance given a word as an input, and **P(W)**, which is called the language model, how likely a word to occur in a text of the language overall. In most speech recognition systems, the acoustic model is represented by Hidden Markov Models (HMMs), which are a generative model of a linguistic unit of speech (phone, word). Each HMM is a finite state machine with n states whereby each state, besides the first and last, has specific output probabilities and each state transition between states is associated with a transition probability. Each state represents an utterance and the end of a chain represents a word.

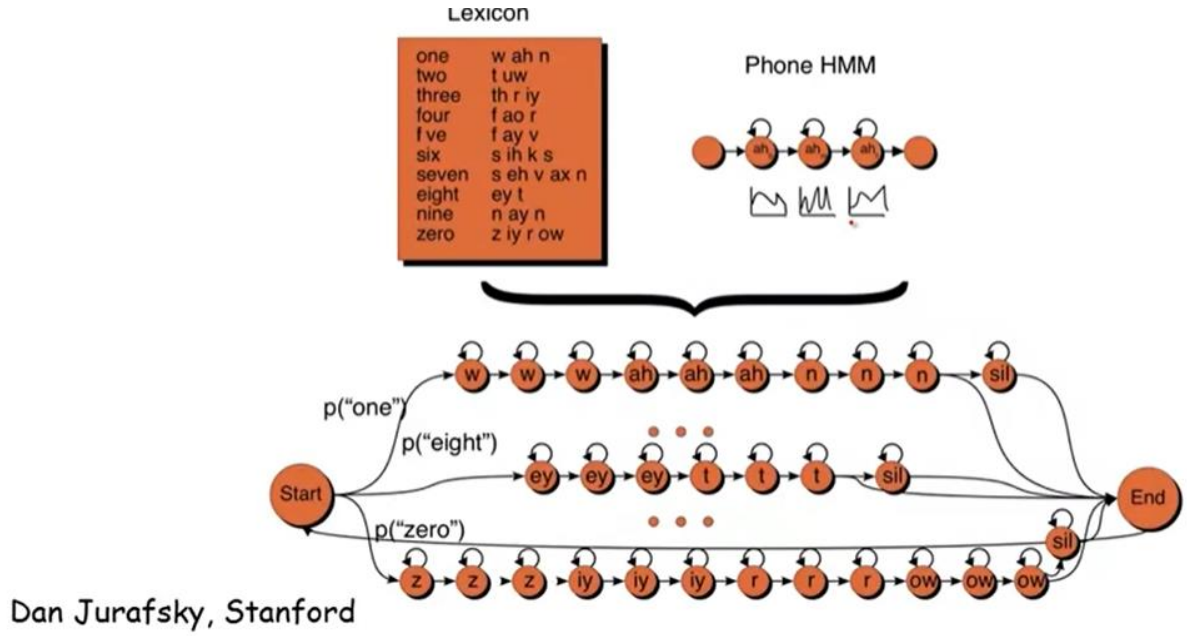


Figure 4 A Markov Model for Speech Recognition

3.2.3 Artificial Neural Networks

It is a class of Machine Learning algorithms in which the biological neural pathways in the brain a mimcked through computational graphs. Each node in the graph is analogues to a neuron, and the connections between the graphs are analogues to synapses in which their thickness is represented by a weight on the connection which gets multiplied by the input. Each node produces a non-linear output from a linear combination of the inputs. $\boldsymbol{y} = f(\boldsymbol{wx} + \boldsymbol{b})$ where y is the output vector, $\boldsymbol{w}$ is the weight vector, $\boldsymbol{x}$ is the input vector and $\boldsymbol{b}$ a bias coeffecient; f is the non-linear activation function. This non-linearity in the output of nodes allows the neural networks to produce highly complex functions that estimates or approximates the outputs to a high degree of accuracy.

The problem with neural networks that prevented itswidespread adoption before recent is that it requires huge resources to train, by training it is meant that a suitable set of weight vectors is calculated for the approximating function, using the most known common algorithms such as Gradient Descent, RMSprob, AdaGrad and Adam. This is done with aid of a *Loss function* ***L***, a function that determines how much is the actual output far away from the desired output. The minmisation of this function is the objective of the

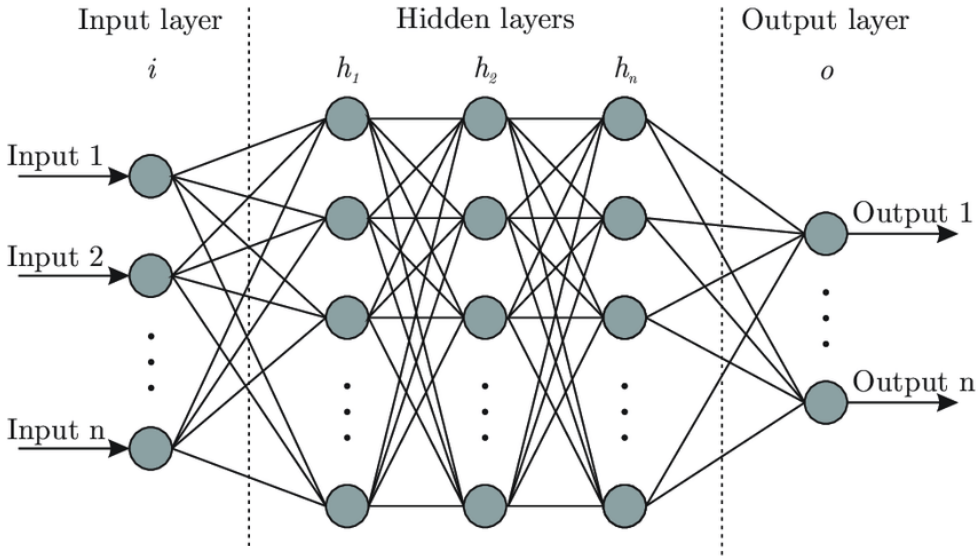


Figure 6 A graphical representation of an Artificial Neural Network

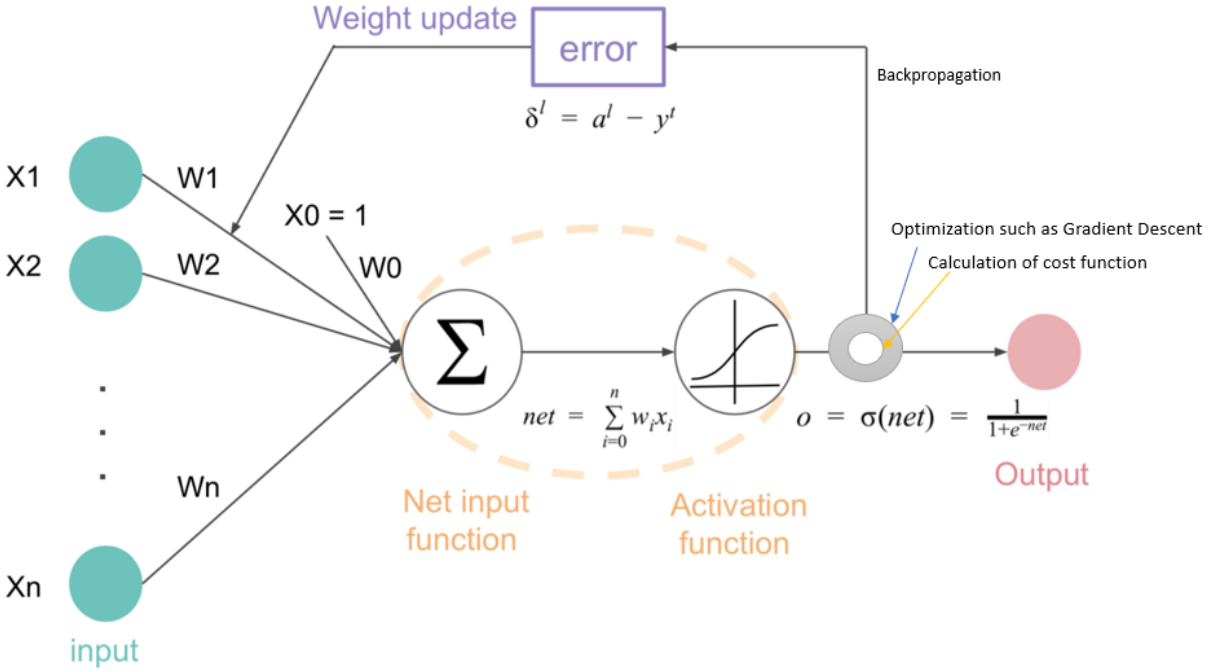


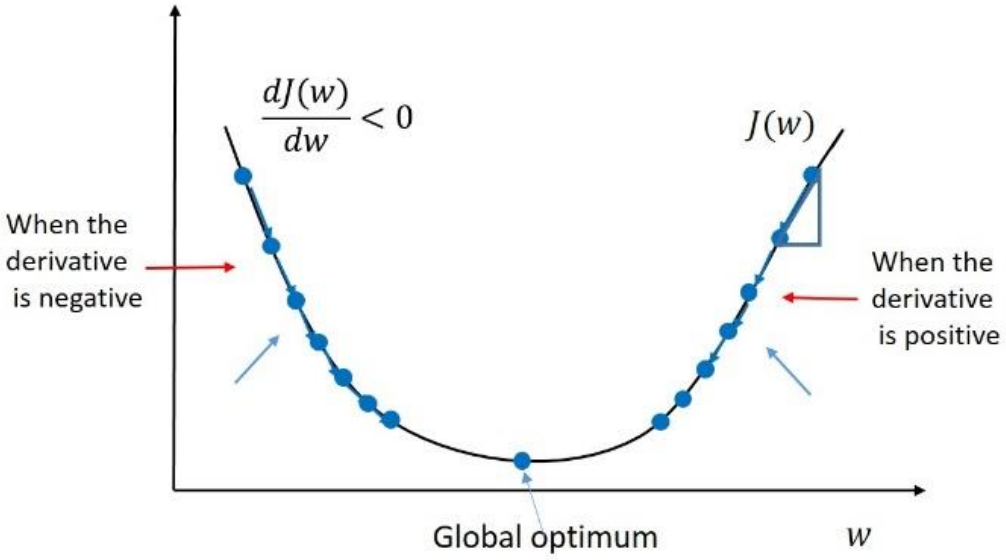
Figure 5 Gradient descent, the father of most of the mentioned algorithms

aforementioned algorithms by tuning weights ***w*** and biases ***b***. All make use of the derivative of the loss function with the respective weight $\frac{\partial L}{\partial w}$.

3.2.4 The Google Speech-To-Text API

Google Speech-To-Text API was used because it offers many of features that will be ease the development of the project. For starters it offers state-of-the-art accuracy as mentioned before and it has very low latency due to the distribution of Google's datacentres worldwide. Most importantly it is an easy-to-use API that integrates well with platforms worked on Unity using C# and python.
Key features

- Speech adaptation
  - It can use domain-specific languages and convert spoken numbers into currencies, dates and addresses using classes.
- Streaming speech recognition
  - It offered streaming the voice from the VR set to its servers to do real-time speech recognition.
- Noise robustness
- Content filtering
  - It can filter profanity if the situation arises.
- It is very affordable for our use
- Well-documented



3.2.5 Processing of the generated text

The text received from the API is tokenised into words then appended to the end of an array tat gets emptied every minute. Using a timer that runs on a separate thread in the code that fires every minute it will calculate the number of words uttered in this minute by the speaker using a variable that contains all the words spoken on or after the last call for timer's handling method till this call; based on it the method will decide whether the speaker was speaking fast or slow or about right.

CHAPTER 4
QUESTION GENERATION

4.1 Introduction

Now we want to make our app more real and enable interaction between humans and characters in the app by making characters ask the user question about the presentation topic and question related to his speech so in this chapter will introduce our question generation system.
Texts with potential educational value are becoming available through the Internet (e.g., Wikipedia, news services). However, using these new texts in classrooms introduces many challenges, one of which is that they usually lack practice exercises and assessments. Here, we address part of this challenge by automating the creation of a specific type of assessment item. Specifically, we focus on automatically generating factual questions. Our goal is to create an automated system that can take as input a text and produce as output questions for assessing a reader's knowledge of the information in the text.

4.2 The dataset (SQuAD)

Large-scale manually annotated passage and question pairs play a crucial role in developing question generation systems. We propose to adapt the recently released Stanford Question Answering Dataset (SQuAD) (Rajpurkar et al., 2016) as the training and development datasets for the question generation task. In SQuAD, the answers are labeled as subsequences in the given sentences by crowed sourcing, and it contains more than 100K questions which makes it feasible to train our neural network models.

4.3 Related work

Automatic question generation from natural language text aims to generate questions taking text as input, which has the potential value of education purpose (Heilman, 2011). As the reverse task of question answering, question generation also has the potential for providing a large scale corpus of question-answer pairs.
Previous works for question generation mainly use rigid heuristic rules to transform a sentence into related questions (Heilman, 2011; Chali and Hasan, 2015). However, these methods heavily rely on human-designed transformation and generation rules, which cannot be easily adopted to other domains. Instead of generating questions from texts, Serban et al. (2016) proposed a neu- ral network method to generate factoid questions from structured data.

RNN Seq2seq Models:

In this will introduce NG++ paper work, in this we conduct a preliminary study on question generation from text with neural networks, which denoted as the Neural Question Generation (NQG) framework, to generate natural language questions from text without pre-defined rules.
The Neural Question Generation framework extends the sequence-to sequence models by enriching the encoder with answer and lexical features to generate answer focused questions. Concretely, the encoder reads not only the input sentence, but also the answer position indicator and lexical features. The answer position feature denotes the answer span in the input sentence, which is essential to generate answer relevant questions.
The lexical features include part-of-speech (POS) and named entity (NER) tags to help produce better sentence encoding. Lastly, the decoder with attention mechanism (Bahdanau et al., 2015) generates an answer specific question of the sentence.

The NQG framework, which consists of a feature-rich encoder and an attention-based decoder. Fig. 4.1 provides an overview of our NQG framework.

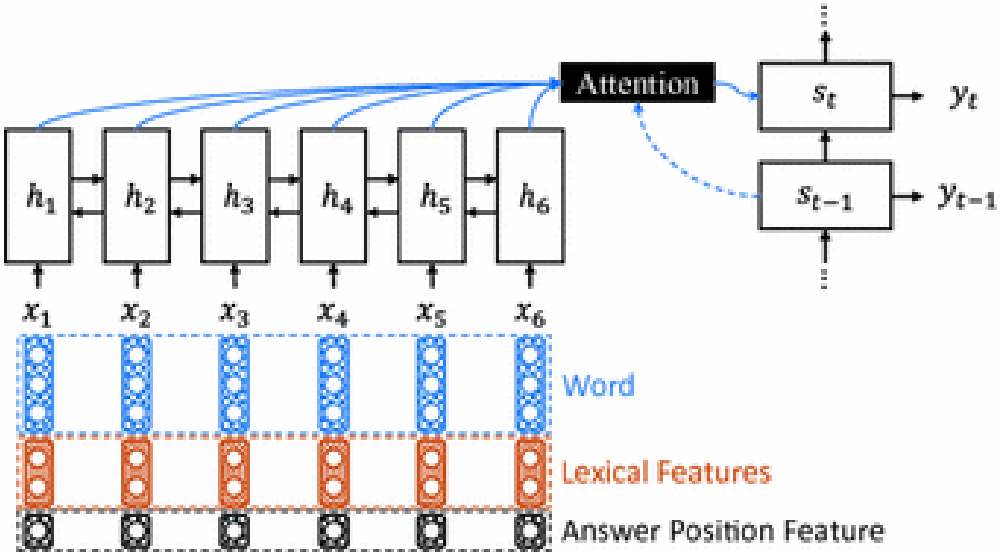


Figure 4.1: Overview of the Neural Question Generation (NQG) framework

Feature-Rich Encoder

We use Gated Recurrent Unit (GRU) (Cho et al., 2014) to build the encoder. To capture more context information, we use bidirectional GRU (BiGRU) to read the inputs in both forward and backward orders. Inspired by Chen and Manning (2014); Nallapati et al. (2016), the BiGRU encoder not only reads the sentence words, but also handcrafted features, to produce a sequence of word-and-feature vectors. We concatenate the word vector, lexical feature embedding vectors and answer position indicator embedding vector as the input of Bi GRU encoder. Concretely, the BiGRU encoder reads the concatenated sentence word vector, lexical features, and answer position feature, $x = (x1; x2; : : : ; xn)$, to produce two sequences of hidden vectors, i.e., the forward sequence $(\sim h1; \sim h2; : : : ; \sim hn)$ and the backward sequence $(h1\sim ; h2\sim ; : : : ; hn\sim)$. Lastly, the output sequence of the encoder is the concatenation of the two sequences, i.e., $hi = [\sim hi; hi\sim]$.

Answer Position Feature

To generate a question with respect to a specific answer in a sentence, we propose using answer position feature to locate the target answer. In this work, the BIO tagging scheme is used to label the position of a target answer. In this scheme, tag B denotes the start of an answer, tag I continues the answer and tag O marks words that do not form part of an answer. The BIO tags of answer position are embedded to real-valued vectors through and fed to the feature rich encoder. With the BIO tagging feature, the answer position is encoded to the hidden vectors and used to generate answer focused questions.

Lexical Features

Besides the sentence words, we also feed other lexical features to the encoder. To encode more linguistic information, we select word case, POS and NER tags as the lexical features. As an intermediate layer of full parsing, POS tag feature is important in many NLP tasks, such as information extraction and dependency parsing (Manning et al., 1999). Considering that SQuAD is constructed using Wikipedia articles, which contain lots of named entities, we add NER feature to help detecting them.

Attention-Based Decoder

We employ an attention-based GRU decoder to decode the sentence and answer information to generate questions. At decoding time step t, the GRU

decoder reads the previous word embedding wt-**1** and context vector ct-**1** to compute the new hidden state st. We use a linear layer with the last backward encoder hidden state h**1**~ to initialize the decoder GRU hidden state. The context vector ct for current time step t is computed through the concatenate attention mechanism (Luong et al., 2015), which matches the current decoder state st with each encoder hidden state hi to get an importance score. The importance scores are then normalized. To get the current context vector by weighted sum:

$$s_t = \text{GRU}(w_{t-1}, c_{t-1}, s_{t-1}) \quad (1)$$

$$s_0 = \tanh(\mathbf{W}_d \overleftarrow{h}_1 + b) \quad (2)$$

$$e_{t,i} = v_a^\top \tanh(\mathbf{W}_a s_{t-1} + \mathbf{U}_a h_i) \quad (3)$$

$$\alpha_{t,i} = \frac{\exp(e_{t,i})}{\sum_{i=1}^{n} \exp(e_{t,i})} \quad (4)$$

$$c_t = \sum_{i=1}^{n} \alpha_{t,i} h_i \quad (5)$$

Then we combine the previous word embedding wt-**1**, the current context vector ct, and the decoder state st to get the readout state rt. The readout state is passed through a max out hidden layer (Goodfellow et al., 2013) to predict the next word with a softmax layer over the decoder vocabulary:

$$r_t = \mathbf{W}_r w_{t-1} + \mathbf{U}_r c_t + \mathbf{V}_r s_t \quad (6)$$

$$m_t = [\max\{r_{t,2j-1}, r_{t,2j}\}]_{j=1,\ldots,d}^\top \quad (7)$$

$$p(y_t | y_1, \ldots, y_{t-1}) = \text{softmax}(\mathbf{W}_o m_t) \quad (8)$$

where rt is a 2d-dimensional vector

Copy Mechanism

To deal with the rare and unknown words problem, Gulcehre et al. (2016) propose using pointing mechanism to copy rare words from source sentence. We apply this pointing method in our NQG system. When decoding word t, the copy switch takes current decoder state st and context vector ct as input and generates the probability p of copying a word from source sentence:

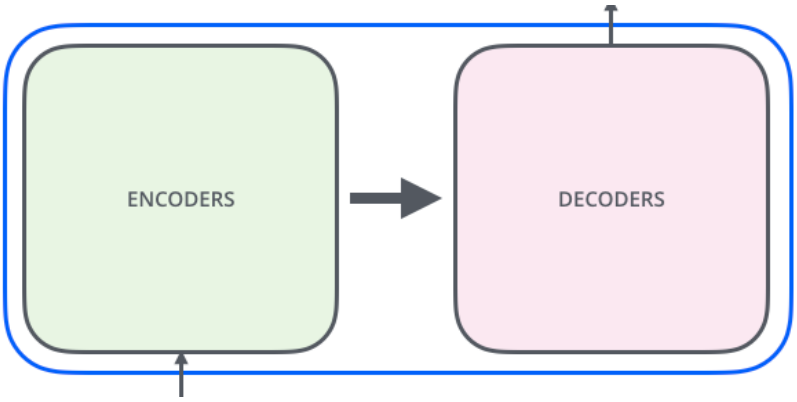
$$p = \sigma(\mathbf{W}st + \mathbf{U}ct + b) \quad (9)$$

where σ is sigmoid function. We reuse the attention probability in equation 4 to decide which word to copy.

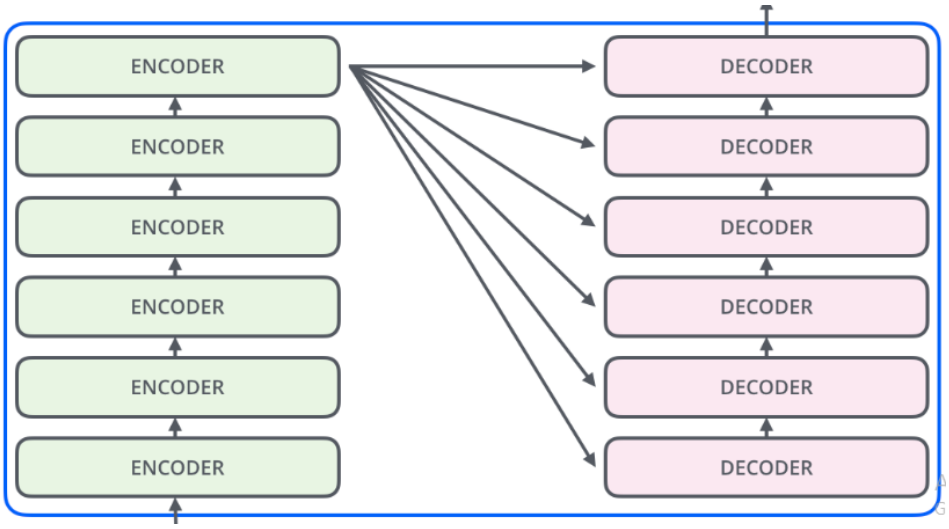
4.4 Attention Is All You Need

In the previous we looked at **Attention** – a ubiquitous method in modern deep learning models. Attention is a concept that helped improve the performance of neural machine translation applications. Now, we will look at **The Transformer** – a model that uses attention to boost the speed with which these models can be trained. The Transformers outperforms the Google Neural Machine Translation model in specific tasks. The biggest benefit, however, comes from how The Transformer lends itself to parallelization. It is in fact Google Cloud's recommendation to use The Transformer as a reference model to use their Cloud TPU offering. So let's try to break the model apart and look at how it functions. The Transformer was proposed in the paper (Attention is All You Need). Let's begin by looking at the model as a single black box.

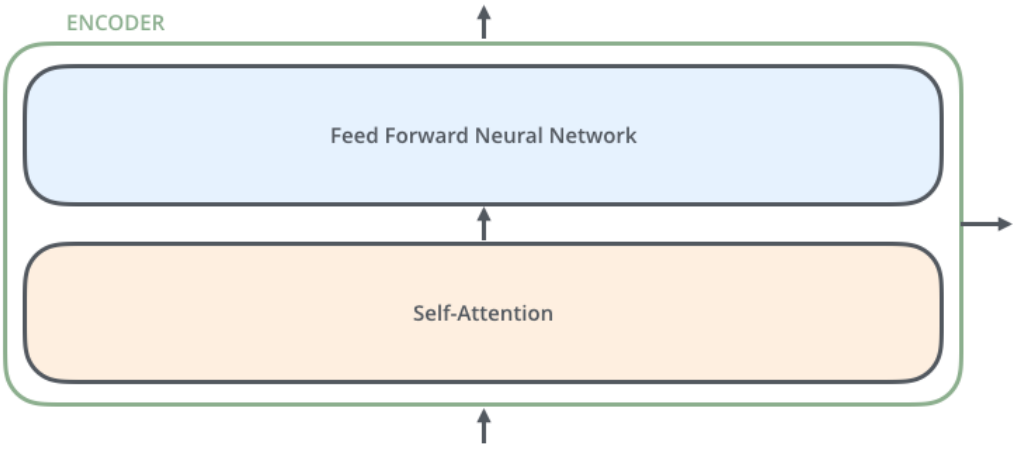
Popping open that Optimus Prime goodness, we see an encoding component, a decoding component, and connections between them.



The encoding component is a stack of encoders (the paper stacks six of them on top of each other – there's nothing magical about the number six, one can definitely experiment with other arrangements). The decoding component is a stack of decoders of the same number.

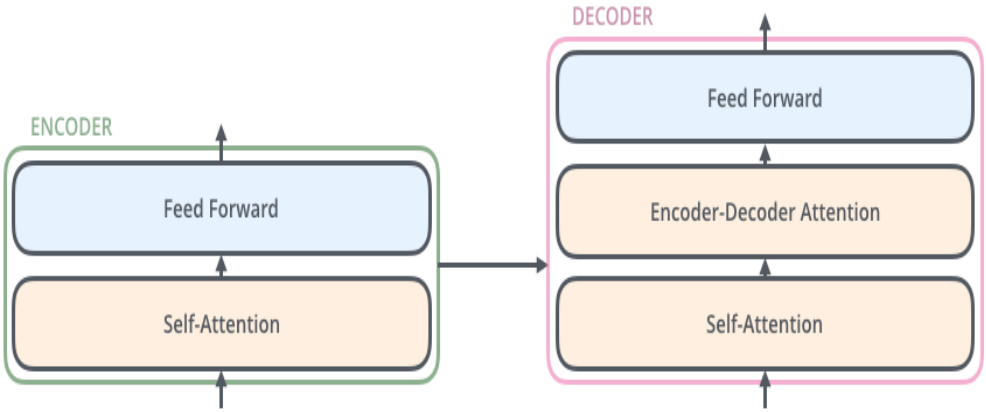


The encoders are all identical in structure (yet they do not share weights). Each one is broken down into two sub-layers:



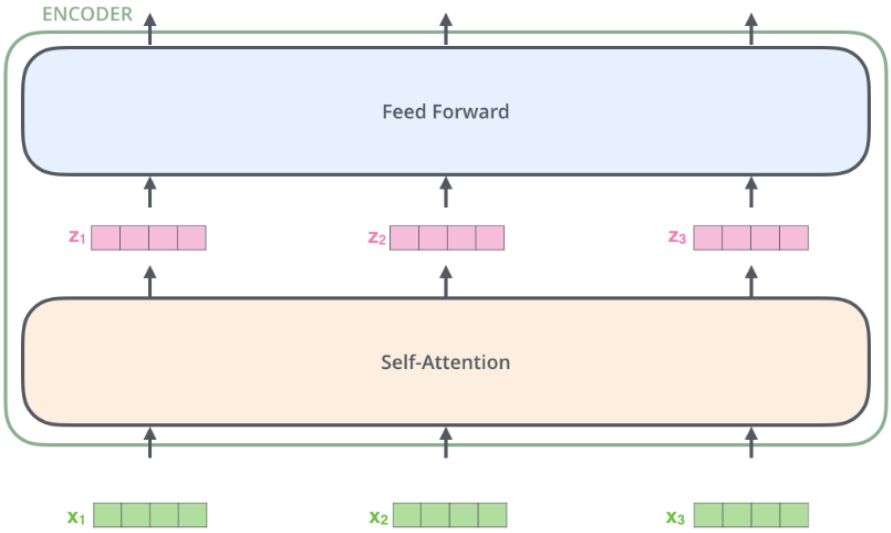
The encoder's inputs first flow through a self-attention layer – a layer that helps the encoder look at other words in the input sentence as it encodes a specific word.

The outputs of the self-attention layer are fed to a feed-forward neural network. The exact same feed-forward network is independently applied to each position.
The decoder has both those layers, but between them is an attention layer that helps the decoder focus on relevant parts of the input sentence (similar what attention does in seq2seq models).



Now that we've seen the major components of the model, let's start to look at the various vectors/tensors and how they flow between these components to turn the input of a trained model into an output.
As is the case in NLP applications in general, we begin by turning each input word into a vector using an embedding algorithm.

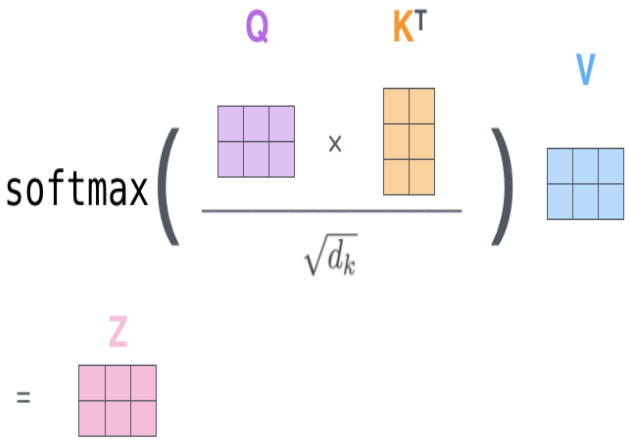
The embedding only happens in the bottom-most encoder. The abstraction that is common to all the encoders is that they receive a list of vectors each of the size 512 – In the bottom encoder that would be the word embedding, but in other encoders, it would be the output of the encoder that's directly below. The size of this list is hyper parameter we can set – basically it would be the length of the longest sentence in our training dataset. After embedding the words in our input sequence, each of them flows through each of the two layers of the encoder.



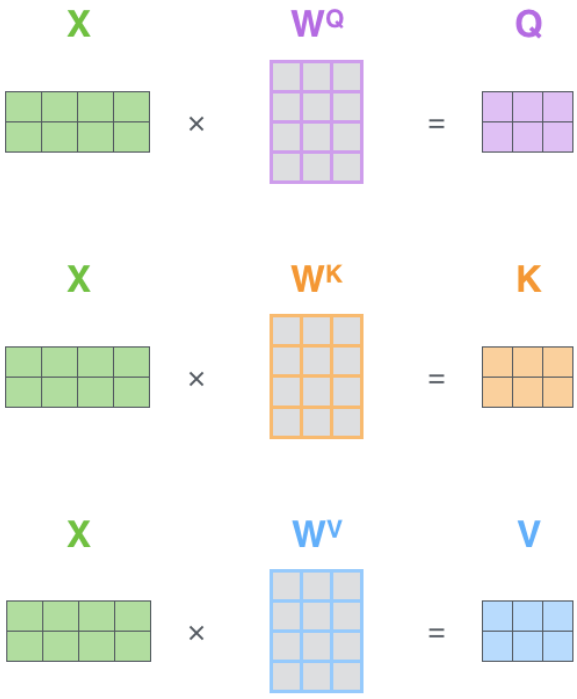
Here we begin to see one key property of the Transformer, which is that the word in each position flows through its own path in the encoder. There are dependencies between these paths in the self-attention layer. The feed-forward layer does not have those dependencies, however, and thus the various paths can be executed in parallel while flowing through the feed-forward layer.

4.4.1 Matrix Calculation of Self-Attention

The first step is to calculate the Query, Key, and Value matrices. We do that by packing our embedding into a matrix X, and multiplying it by the weight matrices we've trained (WQ, WK, WV).
Finally, since we're dealing with matrices, we can condense steps two through six in one formula to calculate the outputs of the self-attention layer.



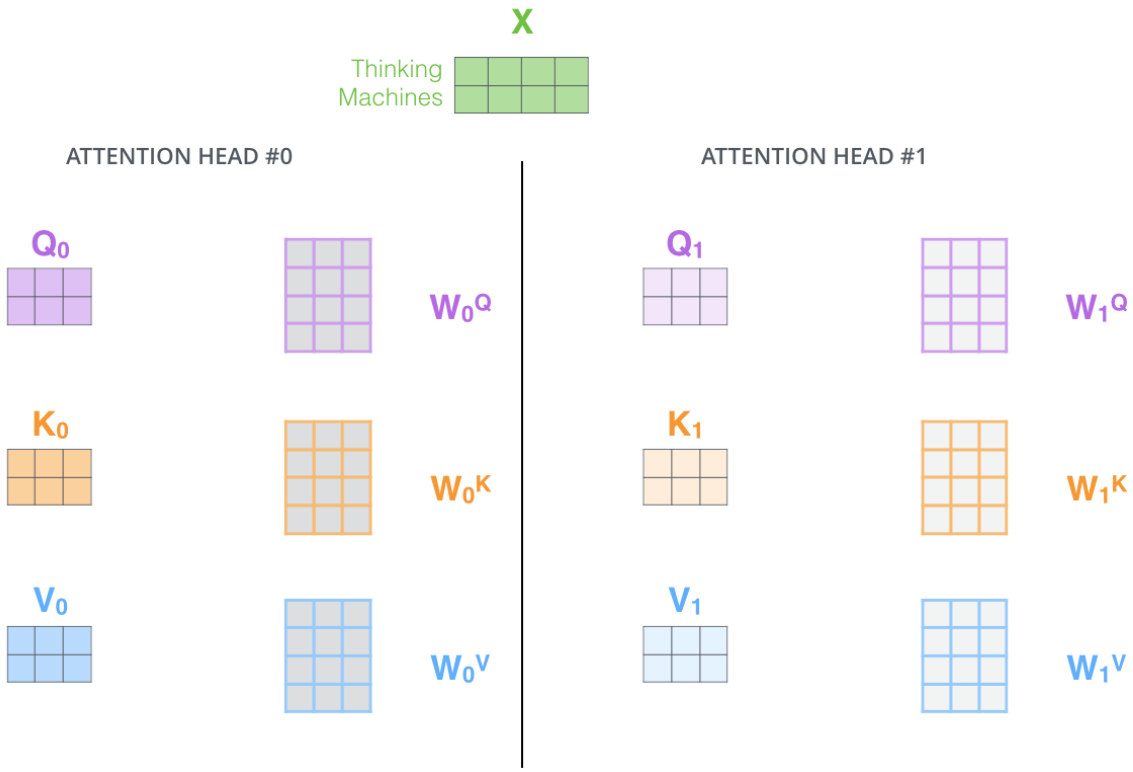
self-attention calculation in matrix form



Every row in the X matrix corresponds to a word in the input sentence. We again see the difference in size of the embedding vector (512, or 4 boxes in the figure), and the q/k/v vectors (64, or 3 boxes in the figure)

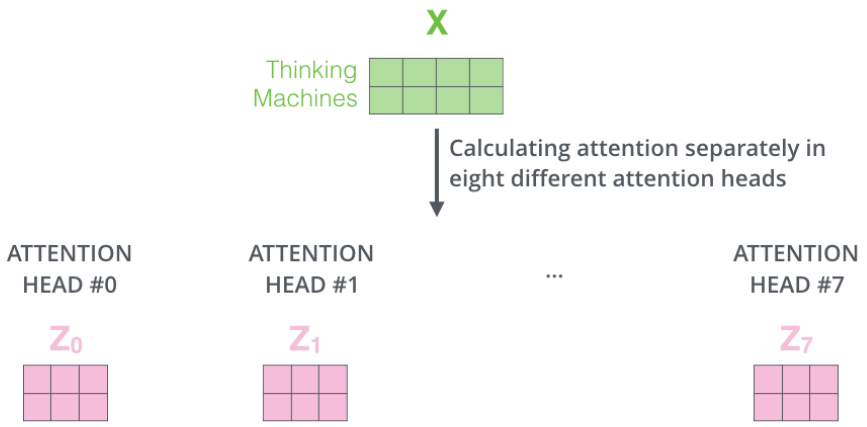
The paper further refined the self-attention layer by adding a mechanism called “multi-headed” attention. This improves the performance of the attention layer in two ways:

1. It expands the model’s ability to focus on different positions. Yes, in the example above, z1 contains a little bit of every other encoding, but it could be dominated by the actual word itself.
2. It gives the attention layer multiple “representation subspaces”. As we’ll see next, with multi-headed attention we have not only one, but multiple sets of Query/Key/Value weight matrices (the Transformer uses eight attention heads, so we end up with eight sets for each encoder/decoder). Each of these sets is randomly initialized. Then, after training, each set is used to project the input embedding (or vectors from lower encoders/decoders) into a different representation subspace.



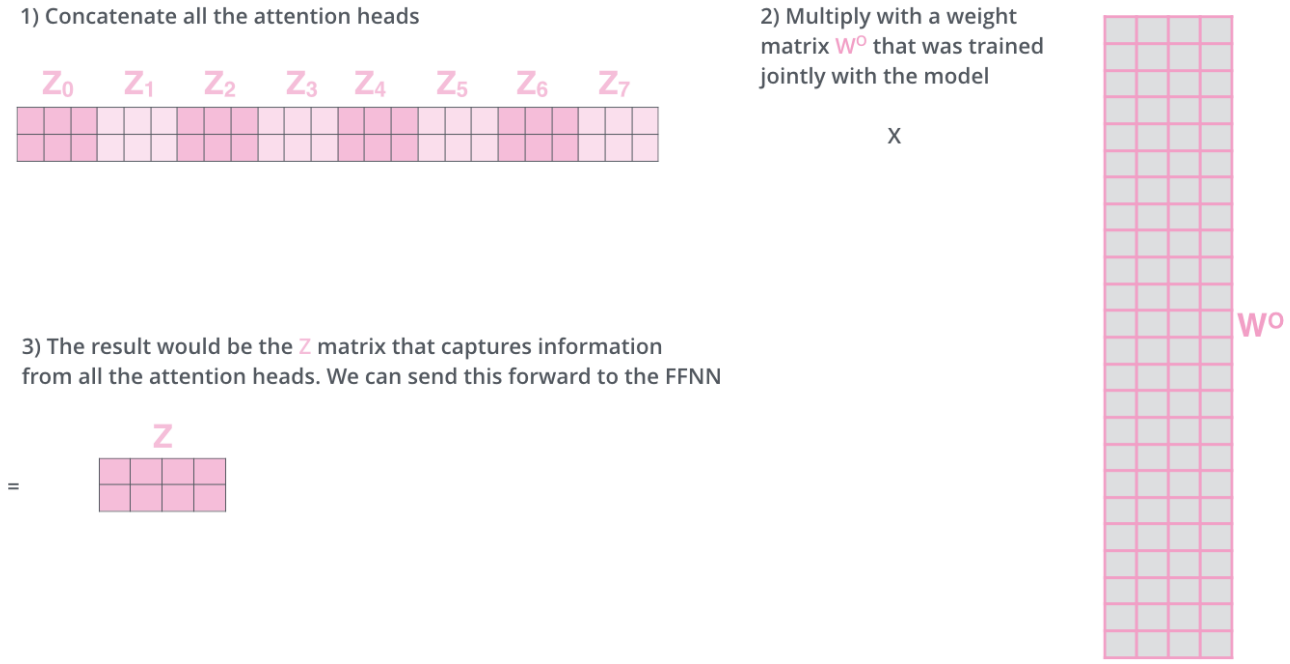
With multi-headed attention, we maintain separate Q/K/V weight matrices for each head resulting in different Q/K/V matrices. As we did before, we multiply X by the WQ/WK/WV matrices to produce Q/K/V matrices.

If we do the same self-attention calculation we outlined above, just eight different times with different weight matrices, we end up with eight different Z matrices

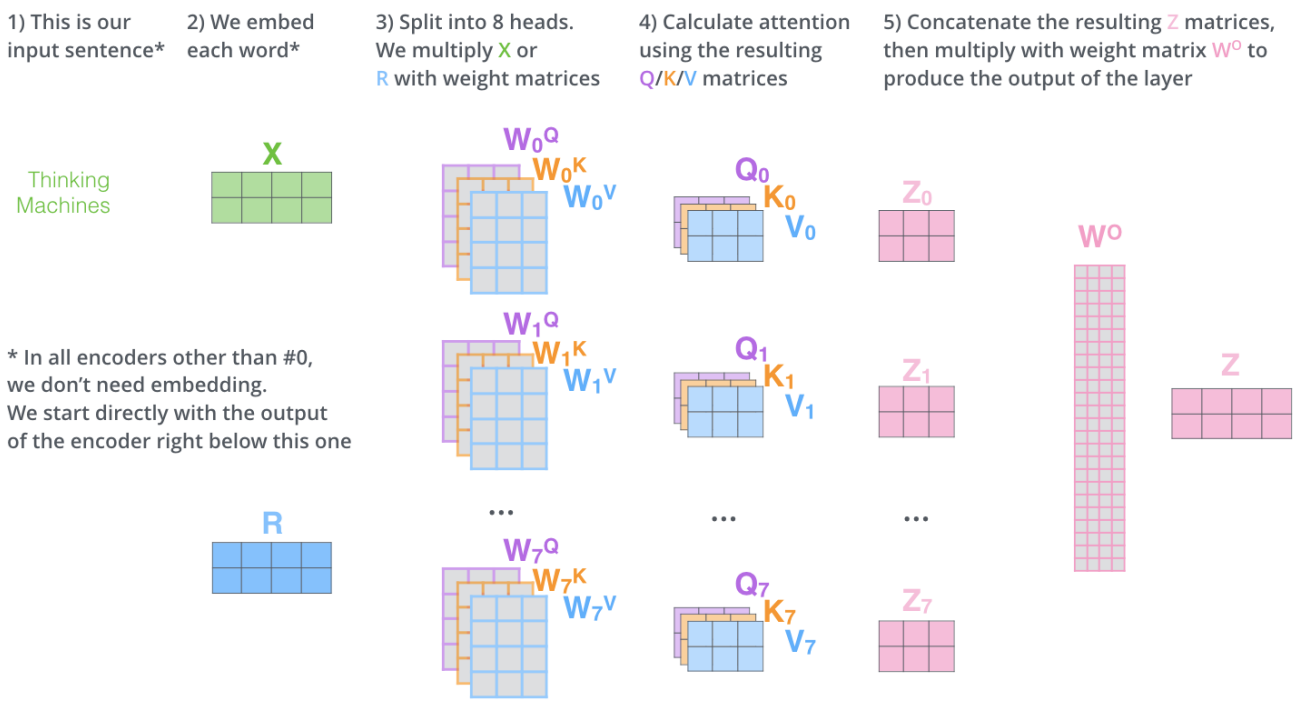


This leaves us with a bit of a challenge. The feed-forward layer is not expecting eight matrices – it's expecting a single matrix (a vector for each word). So we need a way to condense these eight down into a single matrix.

So, we concatenate the matrices then multiple them by an additional weights matrix WO.



That's pretty much all there is to multi-headed self-attention. It's quite a handful of matrices, put them all in one visual so we can look at them in one place



4.4.2 Why Self-Attention

In this section we compare various aspects of self-attention layers to the recurrent and convolutional layers commonly used for mapping one variable-length sequence of symbol representations (x1; :::; x**n**) to another sequence of equal length (z1; :::; z**n**), with x**i**; z**i** 2 **Rd**, such as a hidden layer in a typical sequence transduction encoder or decoder. Motivating our use of self-attention, we consider three desiderata. One is the total computational complexity per layer. Another is the amount of computation that can be parallelized, as measured by the minimum number of sequential operations required. The third is the path length between long-range dependencies in the network. Learning long-range dependencies is a key challenge in many sequence transduction tasks. One key factor affecting the ability to learn such dependencies is the length of the paths forward and backward signals have to traverse in the network. The shorter these paths between any combination of positions in the input and output sequences, the easier it is to learn long-range dependencies. Hence we also compare the maximum path length between any two input and output positions in networks composed of the different layer types.

As noted in Table 1, a self-attention layer connects all positions with a constant number of sequentially executed operations, whereas a recurrent layer requires O(n) sequential operations. In terms of computational complexity, self-attention layers are faster than recurrent layers when the sequence length n is smaller than the representation dimensionality d, which is most often the case with sentence representations used by state-of-the-art models in machine translations, such as word-piece and byte-pair representations. To improve computational performance for tasks involving very long sequences, self-attention could be restricted to considering only a neighborhood of size r in 6 the input sequence centered around the respective output position. This would increase the maximum path length to O(n=r). We plan to investigate this approach further in future work.

A single convolutional layer with kernel width k < n does not connect all pairs of input and output positions. Doing so requires a stack of O(n=k) convolutional layers in the case of contiguous kernels, or O(log**k**(n)) in the case of dilated convolutions [18], increasing the length of the longest paths between any two positions in the network. Convolutional layers are generally more expensive than recurrent layers, by a factor of k. Separable convolutions, however, decrease the complexity considerably, to O(k · n · d + n · d2). Even with k = n, however, the complexity of a separable convolution is equal to

the combination of a self-attention layer and a point-wise feed-forward layer, the approach we take in our model.
As side benefit, self-attention could yield more interpretable models. We inspect attention distributions from our models and present and discuss examples in the appendix. Not only do individual attention heads clearly learn to perform different tasks, many appear to exhibit behavior related to the syntactic and semantic structure of the sentences.

Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

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	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

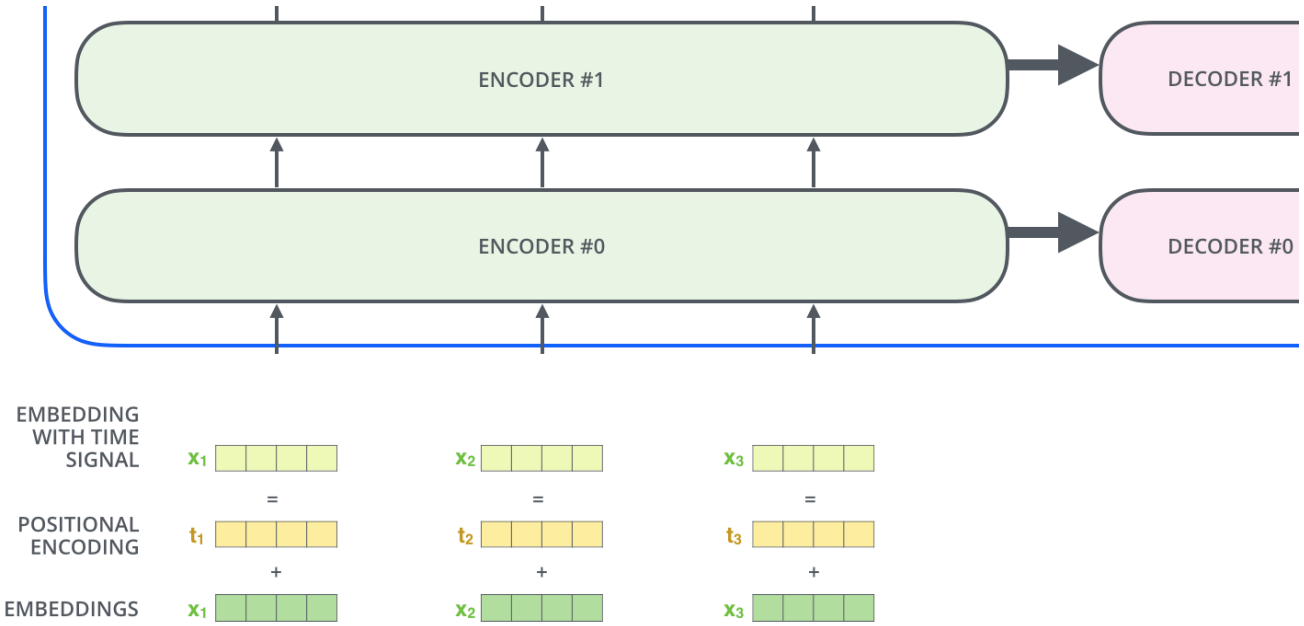
4.4.3 Representing The Order of The Sequence Using Positional Encoding

One thing that's missing from the model as we have described it so far is a way to account for the order of the words in the input sequence. To address this, the transformer adds a vector to each input embedding. These vectors follow a specific pattern that the model learns, which helps it determine the position of each word, or the distance between different words in the sequence. The intuition here is that adding these values to the embedding provides meaningful distances between the embedding vectors once they're projected into Q/K/V vectors and during dot-product attention.

In this work, we use sine and cosine functions of different frequencies:

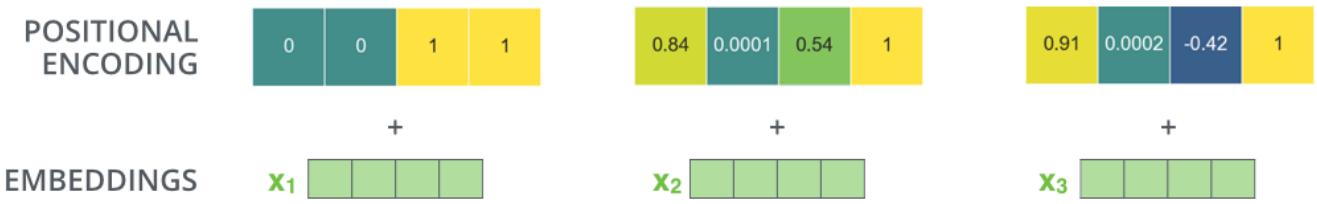
$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

where *pos* is the position and *i* is the dimension. That is, each dimension of the positional encoding corresponds to a sinusoid. The wavelengths form a geometric progression from 2π to $10000 \cdot 2\pi$. We chose this function because we hypothesized it would allow the model to easily learn to attend by relative positions, since for any fixed offset *k*, *PE*pos+k can be represented as a linear function of *PE* pos. We also experimented with using learned positional embedding instead, and found that the two versions produced nearly identical results (. We chose the sinusoidal version because it may allow the model to extrapolate to sequence lengths longer than the ones encountered during training



To give the model a sense of the order of the words, we add positional encoding vectors -- the values of which follow a specific pattern.

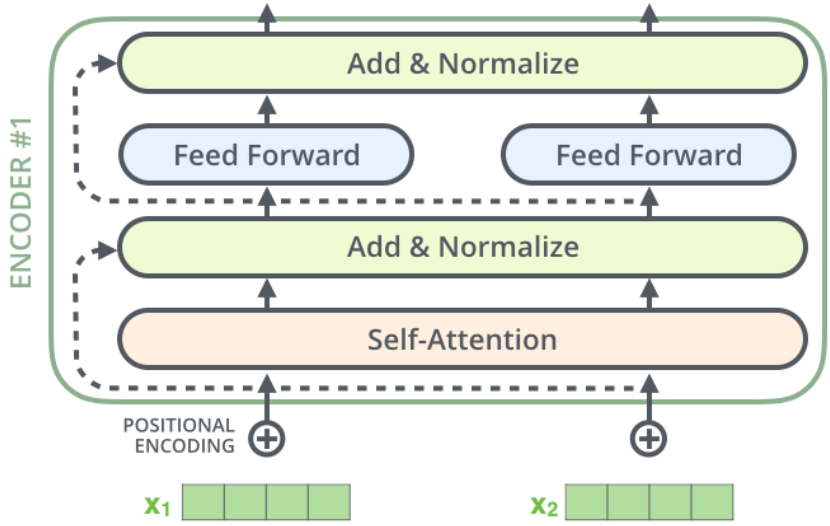
If we assumed the embedding has a dimensionality of 4, the actual positional encodings would look like this:



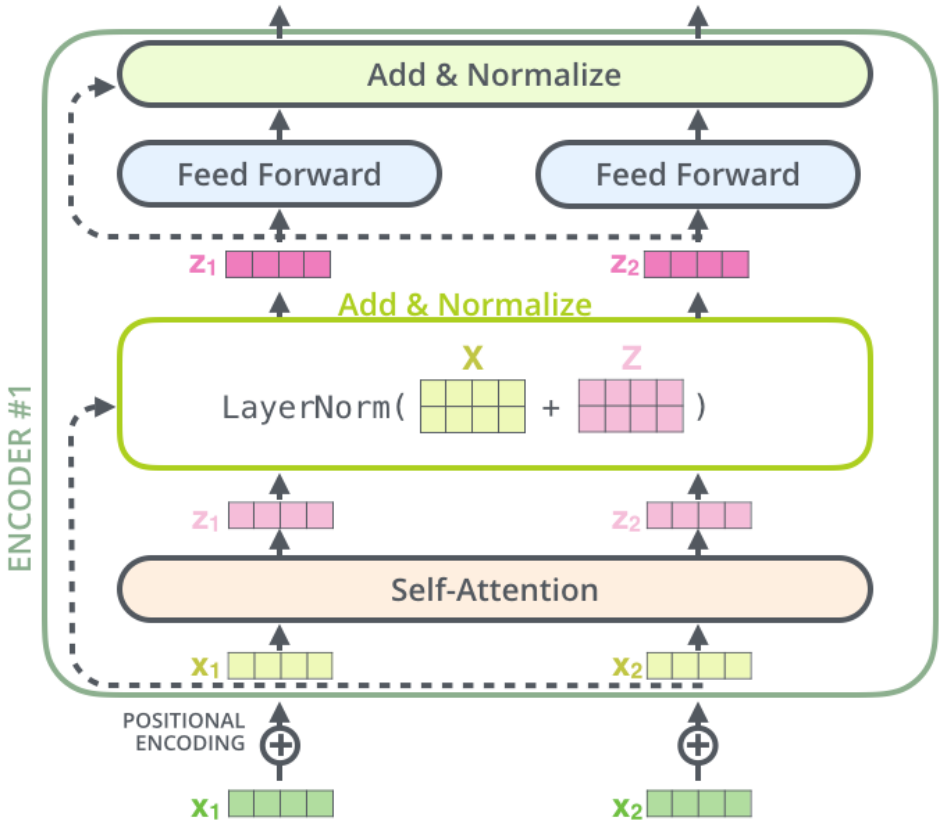
A real example of positional encoding with a toy embedding size of 4

4.4.4 The Residuals

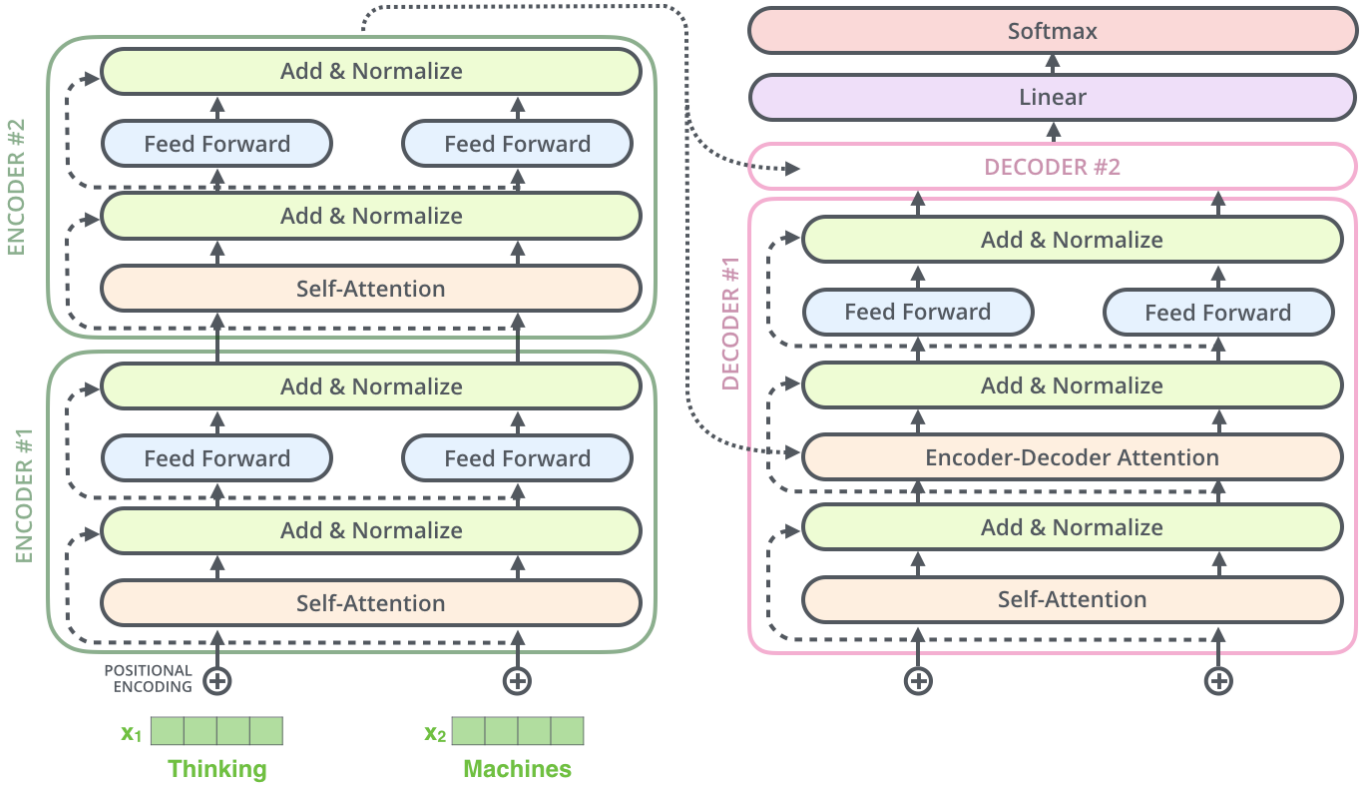
One detail in the architecture of the encoder that we need to mention before moving on, is that each sub-layer (self-attention, ffnn) in each encoder has a residual connection around it, and is followed by a layer-normalization step.



If we're to visualize the vectors and the layer-norm operation associated with self-attention, it would look like this:



This goes for the sub-layers of the decoder as well. If we're to think of a Transformer of 2 stacked encoders and decoders, it would look something like this:



4.4.5 The Decoder Side

Now that we've covered most of the concepts on the encoder side, we basically know how the components of decoders work as well. But let's take a look at how they work together.

The encoder starts by processing the input sequence. The output of the top encoder is then transformed into a set of attention vectors K and V. These are to be used by each decoder in its "encoder-decoder attention" layer which helps the decoder focus on appropriate places in the input sequence: The following steps repeat the process until a special symbol is reached indicating the transformer decoder has completed its output. The output of each step is fed to the bottom decoder in the next time step, and the decoders bubble up their decoding results just like the encoders did. And just like we did with the encoder inputs, we embed and add positional encoding to those decoder inputs to indicate the position of each word.

The self-attention layers in the decoder operate in a slightly different way than the one in the encoder: In the decoder, the self-attention layer is only allowed to attend to earlier positions in the output sequence. This is done by masking future positions (setting them to -inf) before the softmax step in the self-attention calculation.

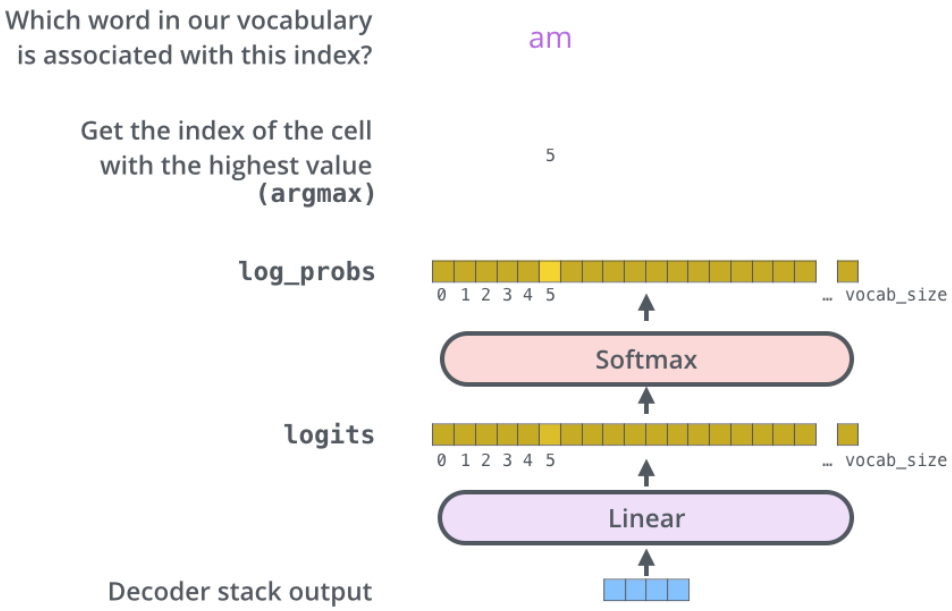
The "Encoder-Decoder Attention" layer works just like multiheaded self-attention, except it creates its Queries matrix from the layer below it, and takes the Keys and Values matrix from the output of the encoder stack.

4.4.6 The Final Linear and Softmax Layer

The decoder stack outputs a vector of floats. How do we turn that into a word? That's the job of the final Linear layer which is followed by a Softmax Layer.

The Linear layer is a simple fully connected neural network that projects the vector produced by the stack of decoders, into a much, much larger vector called a logits vector.

The softmax layer then turns those scores into probabilities (all positive, all add up to 1.0). The cell with the highest probability is chosen, and the word associated with it is produced as the output for this time step.



This figure starts from the bottom with the vector produced as the output of the decoder stack. It is then turned into an output word.

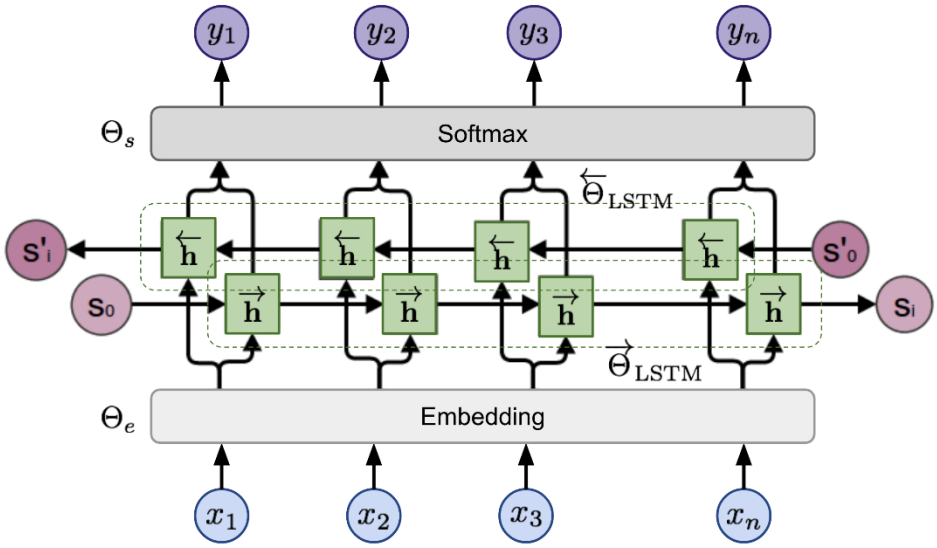
4.5 Transfer Learning in NLP

The year 2018 has been an inflection point for machine learning models handling text (or more accurately, Natural Language Processing or NLP for short). Our conceptual understanding of how best to represent words and sentences in a way that best captures underlying meanings and relationships is rapidly evolving. Moreover, the NLP community has been putting forward incredibly powerful components that you can freely download and use in your own models and pipelines

4.5.1 ELMo

ELMo stands for Embedding from Language Model, as the name suggests in this models the deeply contextualized word embedding are created from the Language Models (LM).

ELMo uses bidirectional language model (biLM) which is pre-trained on a large text corpus, to learn both words (e.g., syntax and semantics) and linguistic context (i.e., to model polysemy). BiLM capture context-dependent aspects of word meaning.



ELMo is applied on semantic-intensive and syntax-intensive tasks respectively using representations in different layers of biLM

- For a semantic-intensive task, the top layer is better than the first layer.
- And for a syntax-intensive task, the first layer is better than top layers.

4.5.2 OpenAI GPT-2

The OpenAI GPT-2 is the successor of the GPT model. GPT-2 is a large transformer-based language model, with generative pre-training of a language model on a diverse corpus of unlabeled text, followed by discriminative fine-tuning on each specific task.

GPT has two major differences from ELMo:

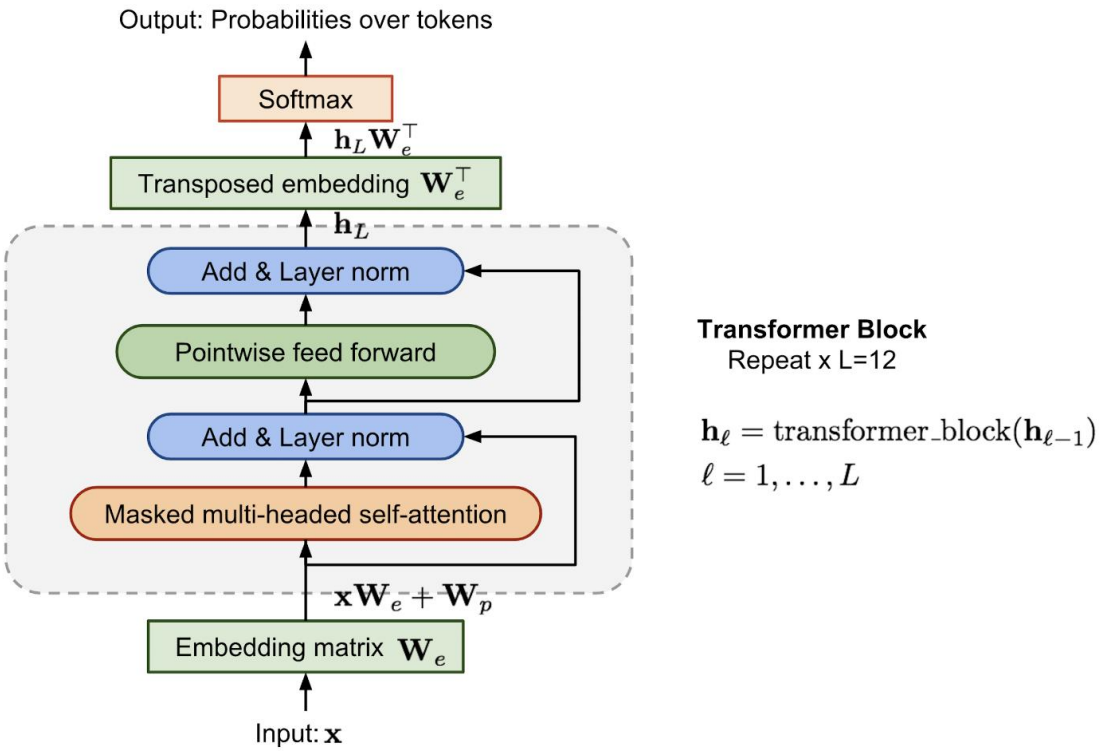
1. The model architecture: ELMo uses the concatenation of forward and backward LSTMs, but GPT uses multi-layer transformers decoder**.**

2. Contextualized embedding: ELMo uses unsupervised Feature-based approach, while GPT fine-tunes the same base model for all end tasks.

Transformer Decoder as Language Model

Unlike original transformer architecture, the transformer decoder model discards the encoder part, so there is only one single input sentence rather than two separate source and target sequences.

Transformer block contains a masked multi-headed self-attention followed by *pointwise feed-forward* layer and normalization layers in between. The final output produces a distribution over target tokens after softmax.

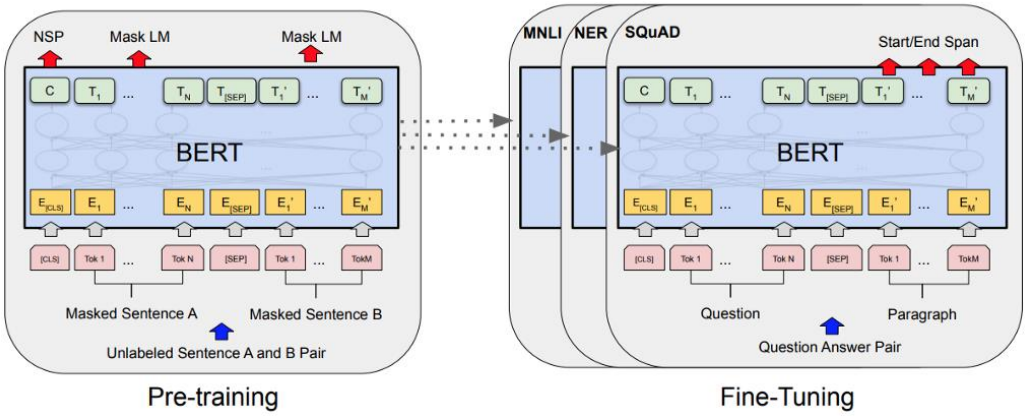


4.6 BERT

BERT stands for **B**idirectional **E**ncoder **R**epresentations from **T**ransformers, as the name suggests this model is based on bidirectional representations from the unlabeled text by jointly conditioning on both left and right context in all layers. As a result, BERT is one of the most breakthroughs ideas in the last few years.

Compared to GPT, the largest difference and improvement of BERT is to make training bi-directional. The paper claim that:

"bidirectional nature of our model is the single most important new contribution"



Pre-Training BERT
Pre-training BERT uses two unsupervised tasks, that are Masked LM and Next Sentence Prediction (NSP) to train.

Task 1: **Masked Language Model** (MLM)
Learning the context around a word rather than learning just after the word makes it able to better capture its meaning, both syntactically and semantically.
The training data generator chooses 15% of the token positions at random for prediction. If the i^{th} token is chosen, we replace the i^{th} token with

1. The [MASK] token 80% of the time
2. A random token 10% of the time
3. The unchanged i^{th} token 10% of the time

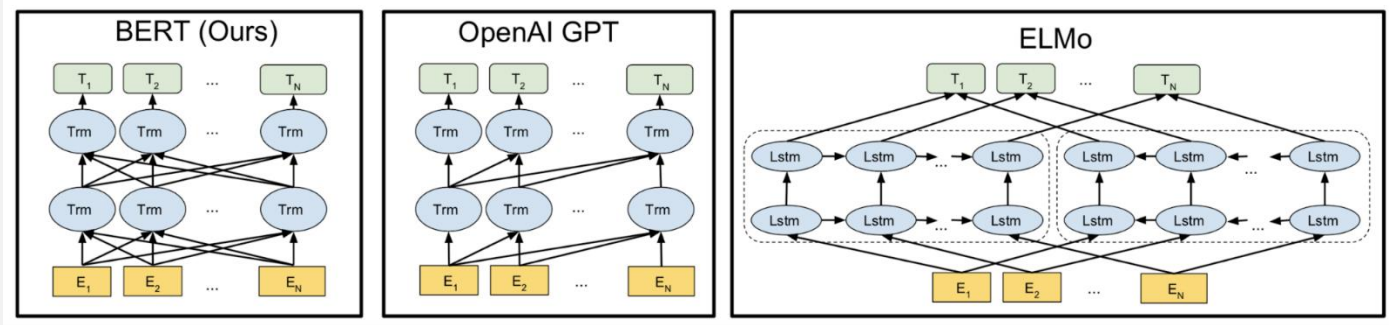
Ti will be used to predict the original token with cross-entropy loss

Task 2: **Next Sentence Prediction** (NSP)
Many important downstream tasks such as Question Answering (QA) are based on the relationship between two sentences, which is not directly captured by language modeling.
BERT performs state-of-the-art results in many NLP task such as:

- Multi-Genre Natural Language Inference (MNLI)
- Quora Question Pairs (QQP)
- Question Natural Language Inference (QNLI)
- The Stanford Sentiment Treebank (SST-2)
- The Corpus of Linguistic Acceptability (CoLA)
- The Semantic Textual Similarity Benchmark (STS-B)
- Microsoft Research Paraphrase Corpus (MRPC)
- Recognizing Textual Entailment (RTE) etc.

4.6.1 Comparison between BERT, GPT-2 and ELMo

The comparisons between the model architectures are shown visually below. Note that in addition to the architecture differences, BERT and OpenAI GPT are finetuning approaches, while ELMo is a feature-based approach.



Comparison of BERT, OpenAI GPT and ELMo, (Image source: BERT original paper)

- BERT and GPT are transformer-based architecture while ELMo is Bi-LSTM Language model.
- BERT is purely Bi-directional, GPT is unidirectional and ELMo is semi-bidirectional.
- GPT is trained on the BooksCorpus (800M words)**;** BERT is trained on the BooksCorpus (800M words) and Wikipedia (2,500M words).
- GPT uses a sentence separator ([SEP]) and classifier token ([CLS]) which are only introduced at fine-tuning time; BERT learns [SEP], [CLS] and sentence A/B embeddings during pre-training.

- GPT was trained for 1M steps with a batch size of 32,000 words; BERT was trained for 1M steps with a batch size of 128,000 words.
- GPT used the same learning rate of 5e-5 for all fine-tuning experiments; BERT chooses a task-specific fine-tuning learning rate which performs the best on the development set.

	Base model	pre-training	Downstream tasks	Downstream model	Fine-tuning
CoVe	seq2seq NMT model	supervised	feature-based	task-specific	/
ELMo	two-layer biLSTM	unsupervised	feature-based	task-specific	/
CVT	two-layer biLSTM	semi-supervised	model-based	task-specific / task-agnostic	/
ULMFiT	AWD-LSTM	unsupervised	model-based	task-agnostic	all layers; with various training tricks
GPT	Transformer decoder	unsupervised	model-based	task-agnostic	pre-trained layers + top task layer(s)
BERT	Transformer encoder	unsupervised	model-based	task-agnostic	pre-trained layers + top task layer(s)
GPT-2	Transformer decoder	unsupervised	model-based	task-agnostic	pre-trained layers + top task layer(s)

Summary of various models, Figure: From Lil'Log by Lilian.

4.7 Unified Language Model Pre-training for Natural Language Understanding and Generation

We propose a model called UNIfied pre-trained Language Model (UNILM). Fine-tune model that can do both NLU and NLG Pretraining with 3 LM tasks (unidirectional,bidirectional ,sequence-to-sequence prediction)

- LM pre-training enables SOTA imaging in a variety of NLP tasks (substantially advanced)
- Pre-trained LMs learn contextualized text representations by predicting words using context around words and use large amounts of text data.
- Pre-trained LMs can be fine-tuned for downstream tasks Various prediction tasks and training objectives have been used depending on the type of pre-training LMs.
- ELMo uses two unidirectional LMs. Because it learns left-to-right and right-to-left in case of GPT, it is left-to-right.
- BERT is bidirectional LM

	ELMo	GPT	BERT	UNILM
Left-to-Right LM	√	√		√
Right-to-Left LM	√			√
Bidirectional LM			√	√
Sequence-to-Sequence LM				√

Table 1: Comparison between language model (LM) pre-training objectives.

Although BERT is a very good model, it is difficult to apply to NLG task due to its characteristics.In this study, we propose a UNIfied pre-trained Language Model (UNILM) and apply the model to both NLU and NLG tasks.
UNILM is a multi-layer transformer network and pre-trains and learns three types of unsupervised language modeling objectives at the same time.

Backbone Network	LM Objectives of Unified Pre-training	What Unified LM Learns	Example Downstream Tasks
Transformer with shared parameters for all LM objectives	Bidirectional LM	Bidirectional encoding	GLUE benchmark Extractive question answering
	Unidirectional LM	Unidirectional decoding	Long text generation
	Sequence-to-Sequence LM	Unidirectional decoding conditioned on bidirectional encoding	Abstractive summarization Question generation Generative question answering

Table 2: The unified LM is jointly pre-trained by multiple language modeling objectives, sharing the same parameters. We fine-tune and evaluate the pre-trained unified LM on various datasets, including both language understanding and generation tasks.

- We have specifically designed some cloze tasks (filling in the blanks) and the context we see there is:
    - unidirectional LM
        - left-to-right unidirectional LM
            - context becomes all words on the left
        - right-to-left unidirectional LM
            - Conversely, all words on the right
    - bidirectional LM
        - context is all words around a word that includes both left and right directions
    - sequence-to-sequence LM
        - The context is the information of the encoder and all words before the word to be predicted in the target sequence.
- Similar to BERT, pre-trained UNILM can be fine-tuned (with additional task-specific layers if necessary), but unlike BERT whose main NLU task is, UNILM is designed to use different self-attention masks to combine contexts of different types of LMs. And both NLG tasks
- The proposed UNILM has three advantages
    - The unified pre-training procedure allows a single Transformer LM to share model parameters and architecture for various types of LMs (alleviating the need of separately training and hosting multiple LMs)
    - Learning different LM objectives that capture context differently prevents overfitting that can occur in any sing LM task, so this parameter sharing makes learned text representations more general.
    - UNILM uses sequence-to-sequence LM, which is a natural choice for NLG (such as abstractive summarization and question generation)
    - According to the experimental results, the proposed model using the bidirectional encoder is comparable to the BERT in GLUE, and also gives good results in two extractive QA tasks (both NLU and NLG are good).

4.7.1 Unified Language Model Pre-training

- Given input sequence x=x1⋯xnx=x1⋯xnFor, UNILM obtains contextualized vector representation for each token.
- In the pre-training phase, the shared Transformer network unidirectional LM, bidirectional LM, and sequence-to-sequence LMis learned as LM objectives.
- To this end, different masks were introduced for self-attention (use masking to control how much context the token should attend)
- After pre-training, you can use fine-tuning with task-specific data for downstream tasks.

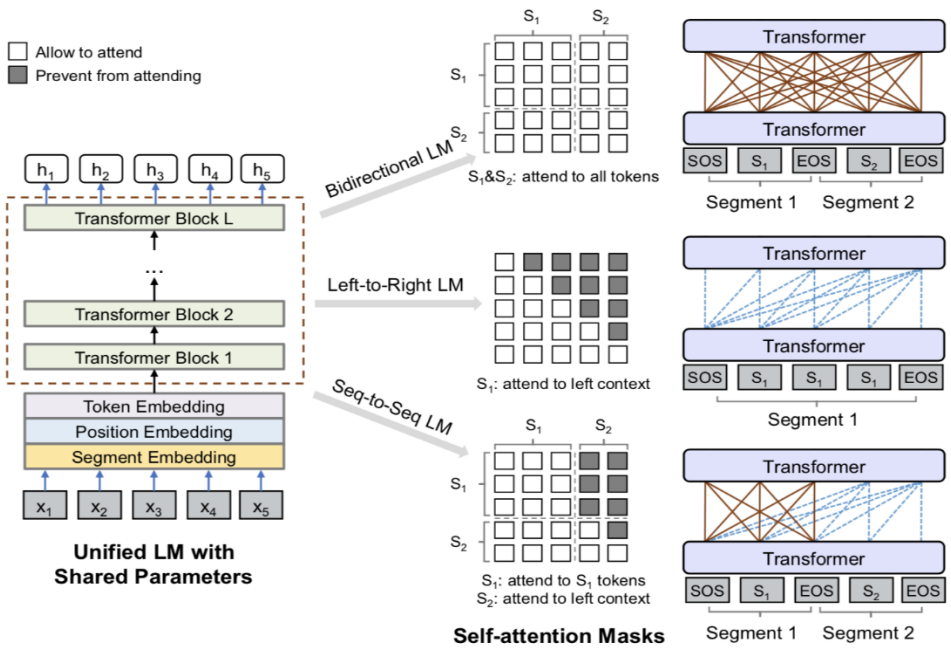


Figure 1: Overview of unified LM pre-training. The model parameters are shared across the LM objectives (i.e., bidirectional LM, unidirectional LM, and sequence-to-sequence LM). We use different self-attention masks to control the access to context for each word token. The right-to-left LM is similar to the left-to-right one, which is omitted in the figure for brevity.

4.7.2 Input Representation

- Special token added
  - [SOS]: start-of-sequence
  - [EOS]: end-of-sequence
- input representation follows BERT format
- Tokenized with WordPiece
- Segment varies depending on the type of LM (see Figure 1)

Backbone Network: Multi-Layer Transformer: Using transformer that we explained before

4.7.3 Pre-training Objectives

- The parameters of UNILM are learned to minimize the cross-entropy loss computed using the predicted tokens and the original tokens
- LM type
  - Unidirectional LM:
    - use both left-to-right and right-to-left LM objectives
    - For instance, to predict the masked token of "x1x2x1x2 [MASK] x4x4", only tokens x1,x2x1,x2 and itself can be used. This is done by using a triangular matrix for the self-attention mask MM
  - LM Bidirectional:
    - the self-attention mask MM is a zero matrix, so that every token is allowed to attend across all positions in the input sequence.
  - Sequence-to-Sequence LM:
    - the tokens in the first (source) segment can attend to each other from both directions within the segment, while the tokens of the second (target) segment can only attend to the leftward context in the target segment and itself, as well as all the tokens in the source segment

        - "[SOS] t1t2t1t2 [They] t3t4t5t3t4t5 [EOS]" into the model. While both t1 and t2 have access to the first four tokens, including [SOS] and [EOS], t4 can only attend to the first six tokens
        - In the case of sequence-to-sequence LM, it can be considered as learning bidirectional encoder and unidirectional decoder
- Next Sentence Prediction:
    - NSP is applied to Bidirectional LM

4.7.4 Pre-training Setup

- Per one training batch, 1/3 is bidirectional LM objective, 1/3 is seq2seq LM objective, and the other 1/3 is unidirectional LM objective (left-to-right, right-to-left)
- The structure of the model
    - gelu activation
    - 24-layer transformer (340M params)
        - with 1,024 hidden size
        - 16 attention heads
    - weight matrix of the softmax classifier is tied wtih token embeddings
    - BISRTLARGISBISRTLARGISInitialized with the weight of
- Corpus uses English Wikipedia and BookCorpus
- Vocab size: 28,996
- Maximum lengths of input seq: 512
- Masking Prob: 15%
    - 80%: [MASK]
    - 10%: random token
    - 10%: original token
- When masking, 80% is masked with one token and the remaining 20% is masked with bigram or trigram.
- Optimizer:
    - Adam: b1=0.9,b2=0.999b1=0.9,b2=0.999
    - lr: 3e-5
    - warm up: first 40,000 steps (and linear decay)
    - weight decay: 0.01
- Dropout rate: 0.1
- Batch size: 330 (~~Peculiar~~)
- pre-training procedure runs: 770,000 steps
- time: 7 hours for 10,000 steps
- GPUs: 8 NVidia Telsa V100 32GB

4.7.5 Fine-tuning on Downstream NLU and NLG Tasks

- For NLU task, fine-tuning like BERT
    - [SOS] vector for tokens hL1h1LAttach a randomly initialized softmax classifier to
- Similar to seq2seq task for NLG tasks
    - Notation
        - S1: source sequence
        - S2: target sequence
    - Pack into one
        - "[SOS] S1 [them], S2 [them]"
    - fine-tuning method:

- Learn to match the tokens in the target sequence after randomly masking them at a specific rate (masking some percentage of tokens in the target sequence at random, and learning to recover the masked words.)
- The training objective is to maximize the likelihood of masked tokens given context
- Masking is also good for [EOS], which is also used as a means to end generation, because the model can learn when to end the generation process (It is worth noting that [EOS], which marks the end of the target sequence, can also be masked during fine-tuning, thus when this happens, the model learns when to emit [EOS] to terminate the generation process of the target sequence.)
- Use content selector to select salient phrases
- 3 beam size

4.7.6 Question Generation

	BLEU-4	MTR	RG-L
CorefNQG [11]	15.16	19.12	-
SemQG [50]	18.37	22.65	46.68
UNILM	**22.12**	**25.06**	**51.07**
MP-GSN [51]	16.38	20.25	44.48
SemQG [50]	20.76	24.20	48.91
UNILM	**23.75**	**25.61**	**52.04**

Table 8: Question generation results on SQuAD. MTR is short for METEOR, and RG for ROUGE. Results in the groups use different data splits.

- Generating a question when passage and answer are given
- Solved as seq2seq problem
  - 1st seg: input passage + answer
  - 2nd seg: generated question
- SQuAD 1.1 dataset is used as evaluation set
- As in the previous study, the original training set is divided into training and test sets, and the original dev set is left as it is.
- hyper params:
  - epoch: 10
  - batch size: 32
  - mask prob: 0.7
  - lr: 2e-5
  - label smoothing: 0.1

4.7.7 Generated Questions Improve QA

	EM	F1
UNILM QA Model (Section 3.2)	80.5	83.4
+ UNILM Generated Questions	**84.7**	**87.6**

Table 9: Question generation based on UNILM improves question answering results on the SQuAD development set.

- Creating a question with the question generation model (data augmentation) and learning again improves the performance of the existing question answering model.

CHAPTER 5
INTERFACING MODEL AND TEXT TO SPEECH

5.1 Cloud computing

In cloud computing, what you might be used to thinking of as software and hardware products, become *services*. These services provide access to the underlying resources. The list of available Google Cloud services is long, and it keeps growing. When you develop your website or application on Cloud, you mix and match these services into combinations that provide the infrastructure you need, and then add your code to enable the scenarios you want to build.

5.2 Google Cloud

Google Cloud Platform is a provider of computing resources for deploying and operating applications on the web. Its specialty is providing a place for individuals and enterprises to build and run software, and it uses the web to connect to the users of that software.

5.2.1 Google Cloud resources

Google Cloud consists of a set of physical assets, such as computers and hard disk drives, and virtual resources, such as virtual machines (VMs).
This distribution of resources provides several benefits, including redundancy in case of failure and reduced latency by locating resources closer to clients. This distribution also introduces some rules about how resources can be used together.

5.3 App Engine VS Compute Engine

Google App Engine and Google Compute Engine are both services businesses can use for deploying applications once they've been coded. Google App Engine is a platform-as-a-service solution designed to make app deployment as easy possible. In contrast, Google Compute Engine is an infrastructure-as-a-service tool that provides a highly configurable and flexible platform for application deployment. Both options are most popular with small businesses, but Google App Engine is more popular with larger businesses, likely due to its automation features.

5.3.1 Features

Google App Engine and *Google Compute Engine* both provide a platform for application deployment, but they also have some standout features that are important to consider.
Google App Engine provides a host of automation features that make it easy for businesses to focus on app development, instead of configuring the deployment. As applications deployed on *Google App Engine* see more or less use, the platform will automatically adjust the number of instances without input from a developer. *Google App Engine* also provides a software development kit to help users optimize applications for the platform.

Google Compute Engine allows for a high level of customization so users can set up their deployment however they want. Businesses with a skilled development team can create as many or as few virtual machines as they want, while customizing them for the needs of their applications. *Google Compute Engine* is also generally more affordable compared to *Google App Engine*, which can make it more appealing to businesses on a smaller budget.

5.3.2 Limitations

Google App Engine and *Google Compute Engine* both help businesses deploy their applications, but they also have a few limitations that are important to consider.
Google App Engine provides a high level of automation and is simple to use, but isn't as customizable as *Google Compute Engine*. Businesses with specific needs for their application may prefer the customizability of *Google Compute Engine*. Additionally, while *Google App Engine*'s software development kit is great for applications that are developed with *Google App Engine*, it can be difficult to take advantage of it if for applications that were developed before *Google App Engine* was selected.
Google Compute Engine is highly customizable, but it isn't as automated or easy to use. Businesses using *Google Compute Engine* will have to manually adjust the volume of their virtual machines as application traffic grows or shrinks. *Google Compute Engine* needs a development team to work with it, unlike *Google App Engine*, which can be managed with less effort.

5.3.3 Pricing

Google App Engine pricing depends on the type of instance, but starts as low as $0.05 per hour per instance.
Google Compute Engine offers pay as you go pricing starting as low as $0.006543 per hour.
When we tried to use *Google App Engine* we faced a problem that there is no GPU therefore we cannot install NVidia driver and cannot use CUDA, so we used *Google Compute Engine*.

5.3.4 Building on Compute Engine:

1. Use virtual machines (VMs), called *instances*, to build your application, much like you would if you had your own hardware infrastructure. You can choose from a variety of instance types to customize your configuration to meet your needs and your budget.
2. Choose which global regions and zones to deploy your resources in, giving you control over where your data is stored and used.
3. Choose which operating systems, development stacks, languages, frameworks, services, and other software technologies you prefer.
4. Create instances from public or private images.
5. Use Google Cloud storage technologies or any third-party technologies you prefer.
6. Use Google Cloud Marketplace to quickly deploy pre-configured software packages. For example, you can deploy a LAMP or MEAN stack with just a few clicks.
7. Create instance groups to more easily manage multiple instances together.
8. Use autoscaling with an instance group to automatically add and remove capacity.
9. Attach and detach disks as needed.
10. Use SSH to connect directly to your instances.

5.3.5 Deploying UniLM Model on cloud and returning random questions

a. Create new VM instance called *_Question generation.*
b. Choose a region with few number of people.
c. Setup machine configuration
    i. General purpose: for CPU, n1-standard-4 (4 VCPU, 15 GB memory).
    ii. GPU Type: NVidia Tesla T4.
    iii. Boot disk: Ubuntu 16.04 LTS.
    iv. Disk: 30GB.
d. Install gcc (Required for CUDA, PyTorch, Apex)
   When completed, you must change to the gcc you want to work with by default. Type in your terminal: `sudo update-alternatives --config gcc`
   To verify if it worked. Type in your terminal: `gcc -v`
e. Get NVidia driver 410.
f. Get CUDA toolkit 10.1.
g. Modify path var because system can't see CUDA installed yet.
h. Download Pytorch for Python 3.6 and CUDA toolkit 10.1 to be able to install NVidia/Apex.
i. Clone the code: `git clone https://github.com/chentinghao/download_google_drive.git`
   Download code from Google drive
   `python3 download_gdrive.py GoogleFileID FileName`
j. Test the server using Postman.

5.4 FLASK

Flask is some code already written for us in Python that makes it easy to get up and running with a simple web application that has a lot of features that can be useful as we go about building web applications, including managing HTTP requests and rendering templates applications.

```
@app.route('/output', methods=['POST'])
Def output():
      #the request should be array of strings
      input_list = request.get_json('text')['text']
      return main(input list)
```

In original code input and output are passed from and into files, the applied modification is that input and output both deal with API.

5.5 API

API stands for Application Programming Interface. An **API** is a software intermediary that allows two applications to talk to each other. In other words, an **API** is the messenger that delivers your request to the provider that you're requesting it from and then delivers the response back to you
In our project we use Google API to send text paragraph to server and get questions back to be asked to presenter by one of the audience.
The input is array of questions and we need to choose one question to ask the presenter, therefore we used `Random.Range()` function to choose a random question.

- Each sentence takes about 20 seconds, so every minute 3 sentences are available for the server to return 3 questions.
- Every minute one of the audience requests to ask a question and the request takes 30 seconds, so the difference between the questions is about two minutes.

5.6 Text to Speech:

Text-to-Speech (TTS) is the ability of a computer to produce spoken words by converting text to voice. In other words, Text-to-Speech software is a speech synthesizer that vocalizes text in real time in a natural way. Text-to-Speech technology can be used in various areas:

Telecommunications: Text to speech can be implemented in IVR systems to create an efficient self-service solution that improves customer satisfaction by informing and guiding callers while reducing costs. TTS can also be used in automated outbound call systems in order to provide information to customers without the need of an agent.

Mobile: From navigators to mobile phones and from e-readers to tablets Text-to-Speech technology is used to vocalize contents in order to provide an easier and more natural interaction. It is also used in mobile applications for various development platforms.

Education: Text-to-Speech technology can be used in language teaching applications to vocalize any word in order to improve the pronunciation capabilities of the users.

Accessibility: Text to speech technology can read text out loud and enables the use of computers and mobile devices for the disabled and for people with special needs. For example, menus of ATMs are vocalized with Text to Speech technology in order to provide enhanced customer experience especially for the disabled. Web site and newspaper.

5.6.1 Comparison among text to speech service providers:

The most popular APIs are Google cloud Text to Speech, IBM Watson Text to Speech and amazon Text to Speech and the following is an abstract information about every one and its benefits.

1. Google cloud Text to Speech API: Convert text into natural-sounding speech using an API powered by Google's AI technologies.

Google cloud Text to Speech features:

1. High fidelity speech: Deploy Google's groundbreaking technologies to generate speech with humanlike intonation. Built based on DeepMind's speech synthesis expertise, the API delivers voices that are near human quality.
2. Widest voice selection: Choose from a set of 220+ voices across 40+ languages and variants. Pick the voice that works best for your user and application.
3. Accelerated innovation: Combine with the best of Google's technologies in Translation and Speech-to-Text to unlock use cases like multilingual audio content and voice bots.

Voice and language selection	Choose from an extensive selection of 220+ voices across 40+ languages and variants, with more to come soon.
WaveNet voices	Take advantage of 90+ WaveNet voices built based on DeepMind's groundbreaking research to generate speech that significantly closes the gap with human performance.
Text and SSML support	Customize your speech with SSML tags that allow you to add pauses, numbers, date and time formatting, and other pronunciation instructions.
Pitch tuning	Personalize the pitch of your selected voice, up to 20 semitones more or less than the default.
Speaking rate tuning	Adjust your speaking rate to be 4x faster or slower than the normal rate.
Volume gain control	Increase the volume of the output by up to 16db or decrease the volume up to -96db.
Integrated REST and gRPC APIs	Easily integrate with any application or device that can send a REST or gRPC request including phones, PCs, tablets, and IoT devices (e.g., cars, TVs, speakers).

Audio format flexibility	Choose from a number of audio formats including mp3, Linear16, and Ogg Opus.
Audio profiles	Optimize for the type of speaker from which your speech is intended to play, such as headphones or phone lines.

Pricing
Text-to-Speech is priced per 1 million characters of text processed after the free tier.
If you pay in a currency other than USD, the prices listed in your currency on Google Cloud SKUs apply.

2. IBM Watson Text to Speech API:
With Watson Text-to-Speech, you can generate human-like audio from written text. Improve the customer experience and engagement by interacting with users in multiple languages and tones. Increase content accessibility for users with different abilities, provide audio options to avoid distracted driving, or automate customer service interactions to increase efficiencies.
Watson Text to Speech features:

- Enable systems to "speak".
- Develop interactive products for education, automate call center interactions, communicate directions hands-free, build engaging toys for children and more.
- Customize pronunciation
- Deliver a seamless voice interaction that caters to your audience with control over every word.
- Go across languages and voices.
- Convert in English, French, German, Italian, Japanese, Spanish and Brazilian Portuguese. Detects different dialects, such as U.S. and UK English and Castilian, Latin American, and North American Spanish.

Pricing plan	Features	Price	Details
Lite	10,000 Characters per Month	Free	The Lite plan gets you started with 10,000 characters per month at no cost. When you upgrade to a paid plan, you will get access to Customization capabilities.
standard	Standard Characters	$0.02 USD /THOUSAND CHAR	You will be charged per thousand characters
premium	Usage and Training Data is Private + Stored in an Isolated Single Tenant Environment High Availability and Service Level Uptime Guarantee IBM Cloud Service Endpoints HIPAA - Washington DC Only Custom Voice (Beta)		To purchase a premium plan, visit: https://ibm.biz/contact-wdc-premium

3. Amazon Text to Speech API:

Amazon Polly is a service that turns text into lifelike speech, allowing you to create applications that talk, and build entirely new categories of speech-enabled products. Polly's Text-to-Speech (TTS) service uses advanced deep learning technologies to synthesize natural sounding human speech. With dozens of lifelike voices across a broad set of languages, you can build speech-enabled applications that work in many different countries.
In addition to Standard TTS voices, Amazon Polly offers Neural Text-to-Speech (NTTS) voices that deliver advanced improvements in speech quality through a new machine learning approach. Polly's Neural TTS technology also supports two speaking styles that allow you to better match the delivery style of the speaker to the application: a Newscaster reading style that is tailored to news narration use cases, and a Conversational speaking style that is ideal for two-way communication like telephony applications. Finally, Amazon Polly Brand Voice can create a custom voice for your organization. This is a custom engagement where you will work with the Amazon Polly team to build an NTTS voice for the exclusive use of your organization.
Amazon Text to Speech features:

- Natural sounding voices: Amazon Polly provides dozens of languages and a wide selection of natural-sounding male and female voices. Amazon Polly's fluid pronunciation of text enables you to deliver high-quality voice output for a global audience.
- Store & redistribute speech: Amazon Polly allows for unlimited replays of generated speech without any additional fees. You can create speech files in standard formats like MP3 and OGG, and serve them from the cloud or locally with apps or devices for offline playback.
- Real-time streaming: Delivering lifelike voices and conversational user experiences requires consistently fast response times. When you send text to Amazon Polly's API, it returns the audio to your application as a stream so you can play the voices immediately.
- Customize & control speech output: Modify Amazon Polly voices to best suit your needs – Amazon Polly supports lexicons and SSML tags which enable you to control aspects of speech, such as pronunciation, volume, pitch, speed rate, etc.
- Low cost: Amazon Polly's pay-as-you-go pricing, low cost per character converted, and unlimited replays make it a cost-effective way to voice your applications.

According to this comparison we choose Google cloud text to speech API.

5.6.2 Why do we need Text to Speech in our project?

The last stage the application made *HTTP* request and sent an array of sentences to the server, server ran the model and generated questions, sent these questions back to the application.
Following, one of the audience needs to ask the presenter one of those generated questions to interact with him and help him to overcome the problem of being anxious from sudden questions leading to bad performance. This step is made using Text to Speech (take input text and return an audio file).
The configuration of Text to Speech gives us the ability to control how the audio looks like. So when the talking character is a woman, voice is configured as female's voice and when the character is a man, voice is configured as male's voice. Voice could also be configured as neutral sound or unspecified.
Interaction with voices give the simulation more reality. Imagine that questions just appear as text in the scene. It is boring and not real though the aim of VR is to simulate the real world. This is why we chose to interact with voices not just a text.

5.6.3 How to use Google Text to Speech API

This quickstart introduces you to Text-to-Speech. In this quickstart, you set up your Google Cloud Platform project and authorization and then make a request for Text-to-Speech to create audio from text.

1. In the Cloud Console, on the project selector page, select or create a Cloud project.
   Note: If you don't plan to keep the resources that you create in this procedure, create a project instead of selecting an existing project. After you finish these steps, you can delete the project, removing all resources associated with the project.

2. Make sure that billing is enabled for your Google Cloud project.
3. Enable the Cloud Text-to-Speech API.
4. Set up authentication:
    a. In the Cloud Console, go to the Create service account key page.
    b. From the Service account list, select New service account.
    c. In the Service account name field, enter a name.
    d. Don't select a value from the Role list. No role is required to access this service.
    e. Click Create. A note appears, warning that this service account has no role.
    f. Click Create without role. A JSON file that contains your key downloads to your computer.
5. Set the environment variable GOOGLE_APPLICATION_CREDENTIALS to the path of the JSON file that contains your service account key. This variable only applies to your current shell session, so if you open a new session, set the variable again.
6. Open your visual studio project and open tools menu then choose *NuGet Package Manager*, Then Manage NuGet Packages for solution.
7. Search for *Google.Cloud.TextToSpeech* and choose **version 1.1.0**.
8. Make sure of version number because other versions conflict with Speech to Text previously installed.
9. After installation is done you will find a folder named "packages" beside the Asset folder. open this folder and select all *.dll* files and copy it.
10. Create a new folder in the Assets folder and name it "Packages" and past the *.dll* files in it.
11. If you found an error that the `grpc_csharp_ext.dll not found`, go to packages folder in the Asset and rename the file *grpc_csharp_extx64.dll* to *grpc_csharp_ext.dll*.
12. when you try to run Text to Speech service on android platform, error in the GRPC files appears and should do this:
    a. Go to *project_name\Packages\Grpc.Core.1.22.1\native\android\armeabi-v7a*
    b. Copy the file named *libgrpc_csharp_ext.so*.
    c. Create the following path: *project_name \Assets\Plugins\Android\libs\armeabi-v7a*
    d. Paste the file in this path.

Fifth step is suitable when you need to run the application on a PC, but we need to run the application on mobile platform not PC and this requires another method to setup the credentials:
We need to make the credential setup embedded with the code to make it available on all platforms, so we change the fifth step with this step:
This is a C# code to be used with unity (our graphics interface)
Use a variable to contain the credentials and pass it to a GRPC channel

```
var credentialTextToSpeech = GoogleCredential.FromJson("Content of
Json file")
            .CreateScoped(TextToSpeechClient.DefaultScopes);
var channelTextToSpeech = new Grpc.Core.Channel(
            TextToSpeechClient.DefaultEndpoint.ToString(),
            credentialTextToSpeech.ToChannelCredentials());
```

Then we add this channel to the client instance of text to speech class

```
TextToSpeechClient client =
TextToSpeechClient.Create(channelTextToSpeech);
```

We should save the returned mp3 file in a path which be seen in the android files so we will save it using this function which determine the application data path

```
String questionPath =
Path.Combine(Application.persistentDataPath,"question.mp3");
```

Now we are able to make a function that convert text to speech with C# as following:

```
void textToSpeech(string text)
    {
        var credentialTextToSpeech =GoogleCredential.FromJson("Json
content file")
            .CreateScoped(TextToSpeechClient.DefaultScopes);
        var channelTextToSpeech = new Grpc.Core.Channel(
            TextToSpeechClient.DefaultEndpoint.ToString(),
            credentialTextToSpeech.ToChannelCredentials());
        questionPath = Path.Combine(Application.persistentDataPath,
"question.mp3");
        TextToSpeechClient client =
TextToSpeechClient.Create(channelTextToSpeech);
        // Set the text input to be synthesized.
        SynthesisInput input = new SynthesisInput{
            Text = text
        };
        // Build the voice request, select the language code ("en-
US"),
        // and the SSML voice gender ("neutral").
        VoiceSelectionParams voice = new VoiceSelectionParams{
            LanguageCode = "en-US",
            SsmlGender = SsmlVoiceGender.Neutral
        };
        // Select the type of audio file you want returned.
        AudioConfig config = new AudioConfig{
            AudioEncoding = AudioEncoding.Mp3
        };
        // Perform the Text-to-Speech request, passing the text input
        // with the selected voice parameters and audio file type
        var response = client.SynthesizeSpeech(new
SynthesizeSpeechRequest{
            Input = input,
            Voice = voice,
            AudioConfig = config
        });
        // Write the binary AudioContent of the response to an MP3
file.
        using (Stream output = File.Create(questionPath)){
            response.AudioContent.WriteTo(output);
            Debug.Log("Audio content written to file 'question.mp3'");
        }
        //call talking funtion here to ensure that the file is saved
    }
```

At the end of this chapter we build a function which take a text input and output an audio file and save it at the application data path.
when the characters need to ask they will read these files and use an audio source object from unity and interact with the presenter.

CHAPTER 6
GRAPHICS

6.1 Introduction

This chapter focuses on the graphics part of the project. The idea is to build a scene similar to a simple lecture hall to give the user the feeling of authenticity when using the application.
To bring this idea described in Fig. 6.1 to life we have three important tasks the first is to model the scene on some 3D modeling application and then export that scene to a game engine. The second part is to use that game engine to add lighting and further enhancement to the static scene. The third and final part is to add the characters to that scene.
Now that we know the target, we have some problems. What 3D modeling software to use? What game engine is best suitable for our needs and our time plan? Should we design the characters or get some predesigned characters? This chapter mainly answers these questions and gives a walkthrough the development process.

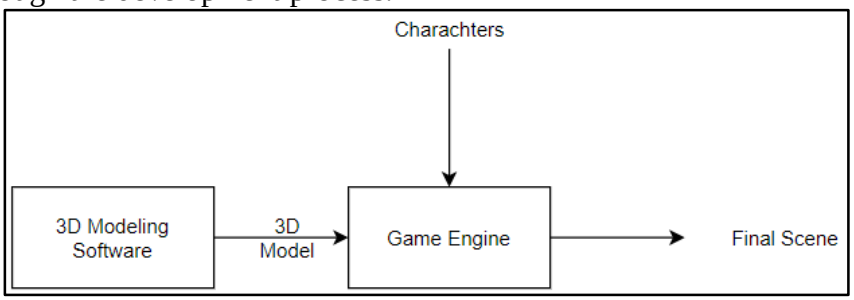


Figure 6.1 Graphics Phase

6.1.1 Scene Modeling

Designers use 3D modeling programs to make 3D animations, models, games and images with ease. It is used for variety of reasons from architectural design to game design.
3D modeling programs are available to use like 3Ds Max, Maya, Blender and we have to choose one them for our project. We had to decide which one based on ease of use and how long would it take to have it ready and how good is the software actually is.
Our best bet was 3Ds max. Its offers simple interface for both beginners and professionals with some easy to follow tutorials and documentation and we had some previous experience using it so it seemed to be a great choice for our project.

6.1.2 Game Engine

A game engine is a software-development environment designed for people to build video games.
We have to use a game engine to combine the scene - after further enhancing it with lighting, materials and textures - with the characters and their animation and later on to build our application for the android platform. There are many good engines available like Unity or Unreal Game Engine.

Unity has more available resources to learn from and game models and packages to experiment with compared to Unreal Game Engine and the second issue is now resolved.

6.1.3 Characters and Animations

After the static scene is built, it is time to add the characters to the scene but that is easier said than done. Character design is hard and tedious process and very time consuming, as we have to use a software to first design the shape, looks of the character, rig it and later export it to Unity. Rigging in Fig. 6.2 means giving the character bones or a skeleton and that is necessary for bringing the characters to life via animations.

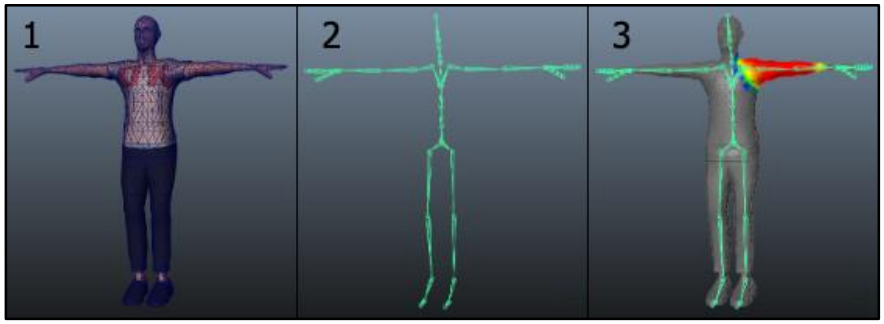


Figure 6.2 Character Design and Rigging

An easy approach to this problem is use free rigged characters that are available on the internet and use them directly rather than designing them from scratch. Mixamo by Adobe characters and Unity's asset store characters are the best options available. Now that we all questions are answered this is the modified plan in Fig. 6.3.

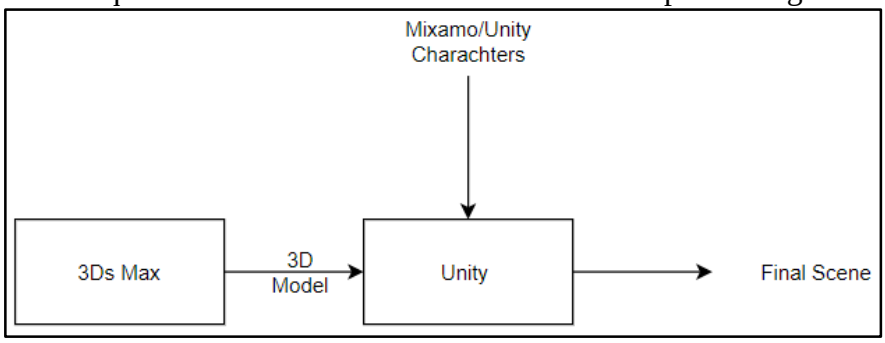


Figure 6.3 Graphics Phase (Modified)

6.2 3Ds Max

A computer graphics program for creating 3D models, animations, and digital images. It is one of the most popular programs in the computer graphics industry and is well known for having a robust tool-set for 3D artists.

We have chosen this program because it is easier to use and it is one of the best programs in design so that it is easy to convert it to FBX to use in Unity.

6.2.1 Interface

Like any other program, it has a workspace to design the scene and many tools to create it also to modify on the scene to make it as we want as shown in Fig 6.4.

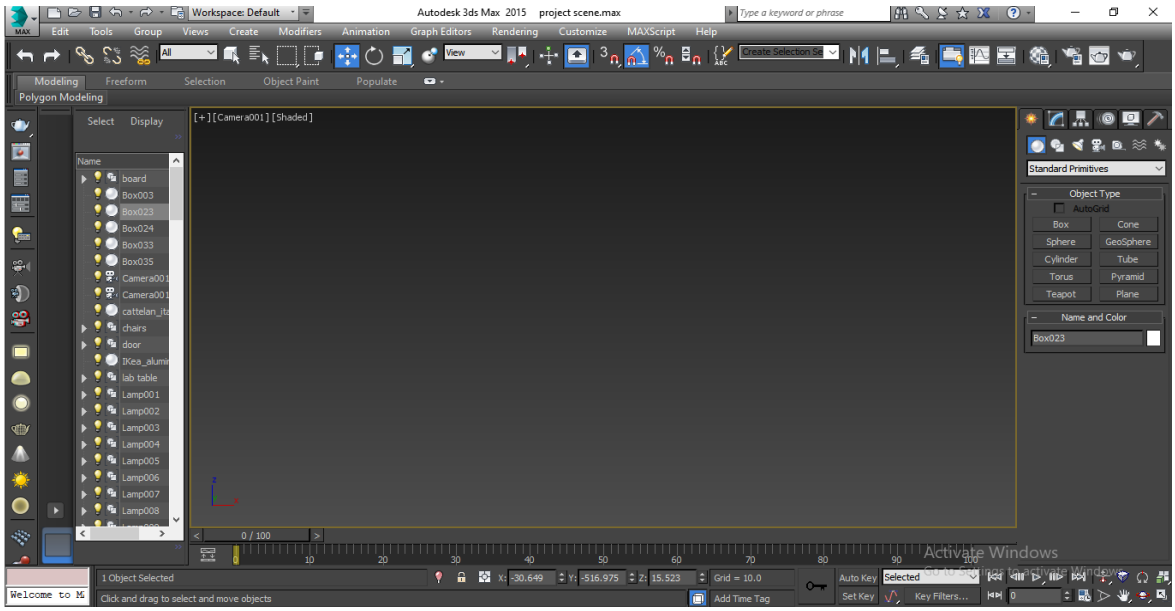


Figure 6.4 3Ds Max Interface

6.2.2 Designing

The first stage is to design the scene in a comfortable and realistic way. We use some tools to design it like (boxes, cubes, cylinders) and many other shapes to design chairs, stairs, board, laptop, projector, clock and also microphone, combine them all together to form the scene. Fig 6.5 shows how we combine those objects to create our scene.

Figure 6.5 Scene

6.2.3 Lighting and Materials

We had to make the scene more realistic by adding some lighting and materials to each component in it. V-Ray add-on is used to add lighting easily.

- **V-Ray**
  V-Ray for 3ds max offers a number of lights essential for a good render. Whether you are preparing an interior or exterior scene, you can find the appropriate lighting options in the V-Ray toolbar or in the Asset Editor.
- **Material**
  We just use an ArchShaders file appending to the program just drag the material and drop on whatever you want.

The combination of light and material is shown in Fig, 6.6.

Figure 6.6 Materials and Lights

6.2.4 Rendering

The process of generating a photo realistic image from 3D model and the last stage in our process is to create a scene. There are two types of render in 3ds max

1. Progressive: which generate an image in short time but the resulting image is noisy and not clear.
2. Bucket: which generate a real image without any noises but take too much time to generate it

Here we render the scene using bucket type, using V-Ray render to produce image with lighting and materials.

6.3 Unity

Unity is a 3D/2D game engine and powerful cross-platform IDE for developers. As a game engine, Unity is able to provide many of the most important built-in features that make a game work. That means things like physics, 3D rendering, and collision detection. From a developer's perspective, this means that there is no need to reinvent the wheel. Rather than starting a new project by creating, a new physics engine from scratch calculating every movement of each material, or the way light should bounce off different surfaces.

Unity is an IDE shown in Fig. 6.7. IDE stands for "integrated development environment," which describes an interface that gives you access to all the tools you need for development in one place. The Unity software has a visual editor that allows creators to simply drag and drop elements into scenes and then manipulate their properties.

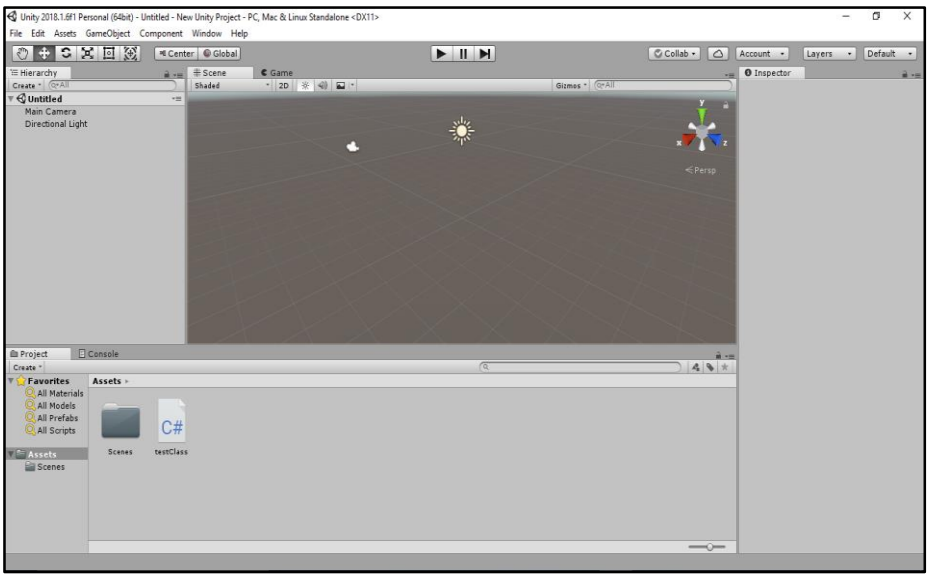


Figure 6.7 Unity IDE

6.3.1 Importing Scene

3Ds Max exports the scene in FBX format, is used in Unity to add some characters, and animate them in some way that characters interact with the presenter to start our project.
We encountered some problems such that the lighting and materials disappeared because 3Ds does not actually export those objects in the FBX format. That meant we have added lighting and materials to the scene once again.

6.3.2 Lighting and Materials

In Unity, we have many tools that help us to add lighting and materials. We use the same ArchShaders files to add materials by a simple drag-and-drop onto any object we want. As for the lights, we use Unity lights like point light, spot light and directional light shown in Fig 6.8.

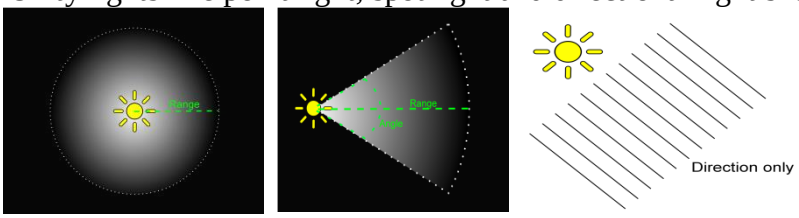


Figure 6.8 Unity Lights

Adding materials and lights make the scene more real, comfortable, it simulates real life, and all of that is required to make the presenter believe it is a real scene.
After all is said and done, in Fig 6.9 we see the scene after we fixed it after importing it.

Figure 6.9 Unity Scene

6.4 Characters

Now that the scene is finally finished in Unity, we now turn our focus to the characters and their facial expressions as that one main aspect of the application and a main indicator of how the user is performing.

6.4.1 Importing Chars. in Unity

As we are using characters from the Unity Assets and Mixamo, we have to import them to the project. This a simply task as we can get those files as .FBX (Filmbox) extension files and that is easily integrated with unity as soon as we import it we get a prefab of our character ready to use which is shown in Fig. 6.10.

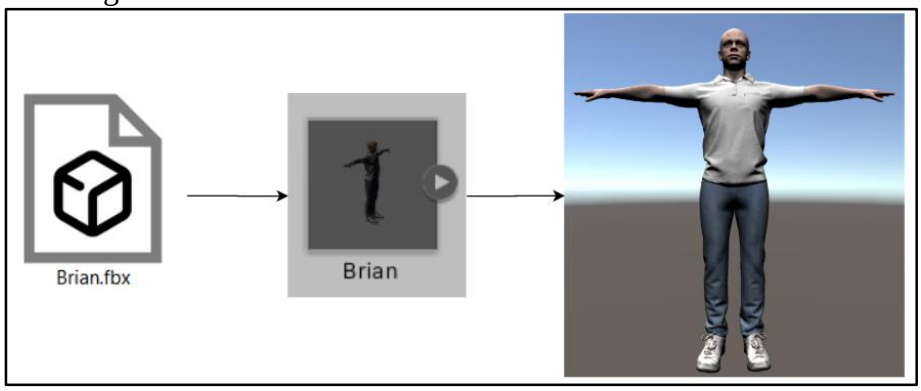


Figure 6.10 Importing Character

6.4.2 Facial Expressions

To control the facial expressions of a character we have to control the muscles of the face and the basic rigging does not offer that option, to overcome this problem there are two options, the first

is to enhance the rig to control every bit of the face, but that is a very hard and time consuming process. The second choice is to use a special option that Unity offers which is Blend shapes. Every character with a Blend shape object attached to it can be manipulated in many more ways compared to the normal rig specially the facial expressions and that can be done easily via sliders as shown in Figure 6.11.
That ease of usage comes at the cost of more memory usage, which we shall discuss more in detail later in optimization in this chapter, but for now, we can simply use a few characters with the Blendshape settings and fill the remainder of the scene with normal rigged Mixamo characters

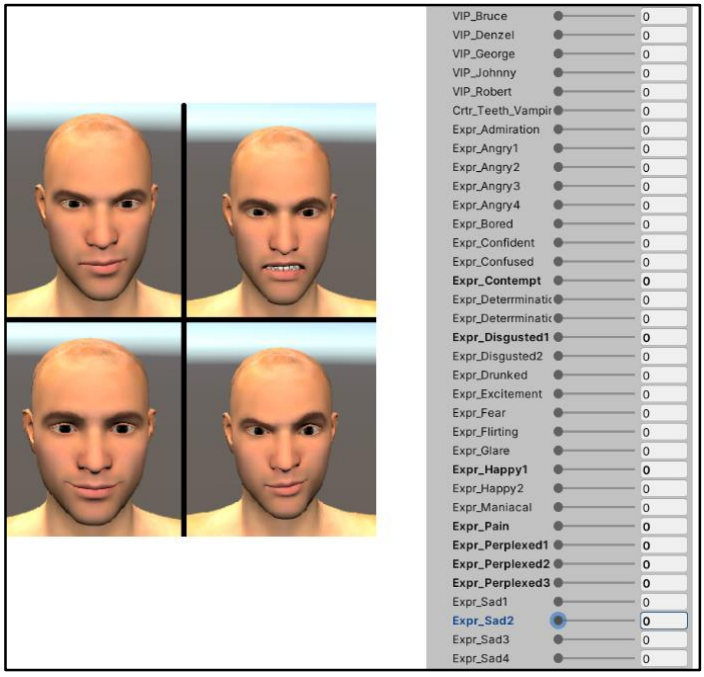


Figure 6.11 Blendshape Settings

Fig 6.12 shows the scene after we added the characters from both Unity assets and Mixamo.

Figure 6.12 Scene with Characters

6.5 Animation

Animating of characters is the technique of figures being manipulated to appear as if moving. In our project, we use animation to make the characters as real and as interactive as we can on a virtual platform. The character can be manipulated to reach the reaction we need to achieve, for example, raising its hand, imitating talking and many more animations that can be applied.
As we discussed in part 5.1.3, characters should be rigged so we can manipulate the whole character and make it move we can make the animation we want, but this is very extensive and time consuming, so there are various standard animations that can be downloaded and imported in Unity and works on some characters, here we used the Mixamo Animations.

6.5.1 How animations work on characters

First characters should be rigged, and in our project, we have already rigged characters from Mixamo and from Blendershapes, so we start by downloading the characters and importing them to Unity.
For animations, we use something called the Animator. To start the animator, we create a folder in the Assets folder called Animation, right click and go to Create -> “Animator Controller”. The process is shown in Fig 6.13.

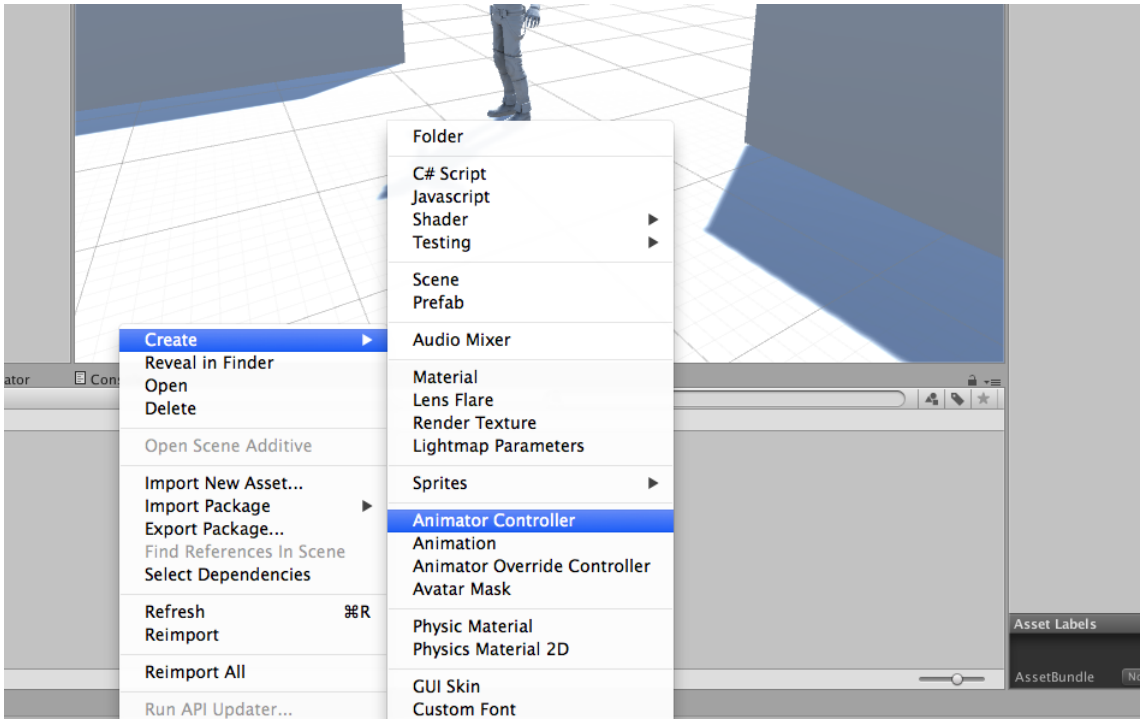


Figure 6.13 Animator Controller

We name the character controller, preferably the character's name and drag our new Animator Controller on the character's model as shown in Fig 6.14.

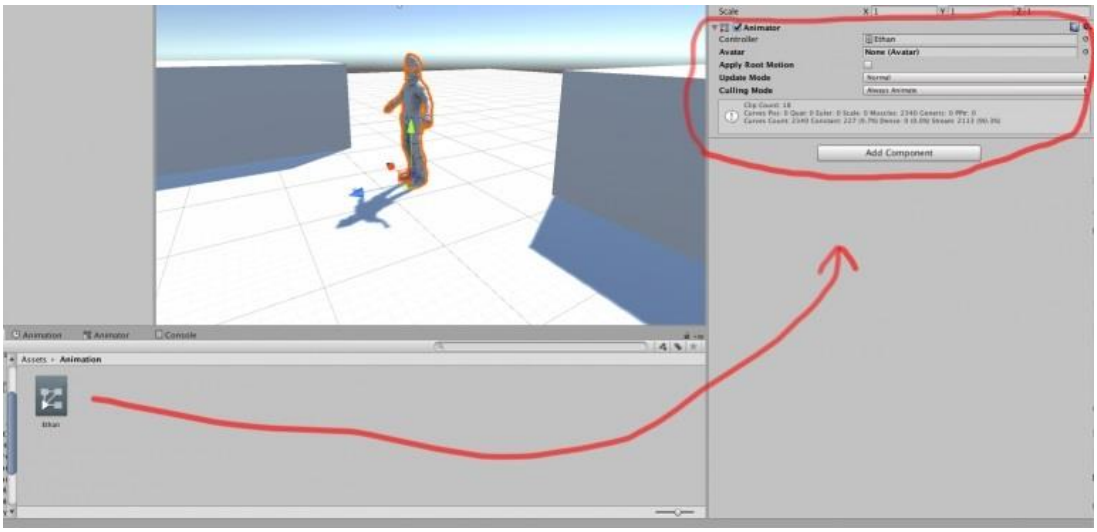


Figure 6.14 Animator on Character

The Animator Controller allows you to assign multiple animations to a model. For humanoid characters, such as ours, it requires something called an "Avatar" seen in Fig 6.15.

The Avatar system is how Unity identifies that a particular animated model is humanoid in layout, and which parts of the model correspond to the legs, arms, head and body.
Because of the similarity in bone structure between different humanoid characters, it is possible to map animations from one humanoid character to another, allowing **retargeting** and **inverse** kinematics (IK).

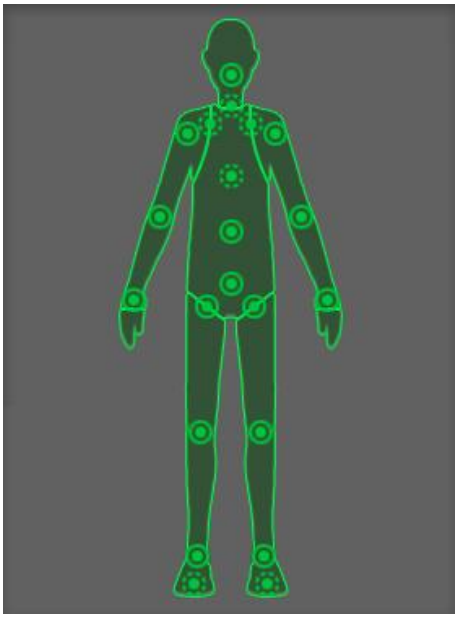


Figure 6.15 Avatar

The characters we have in our project, already have their avatars.
In the character's model, Fig 6.16 shows some of the settings in the Animator component.

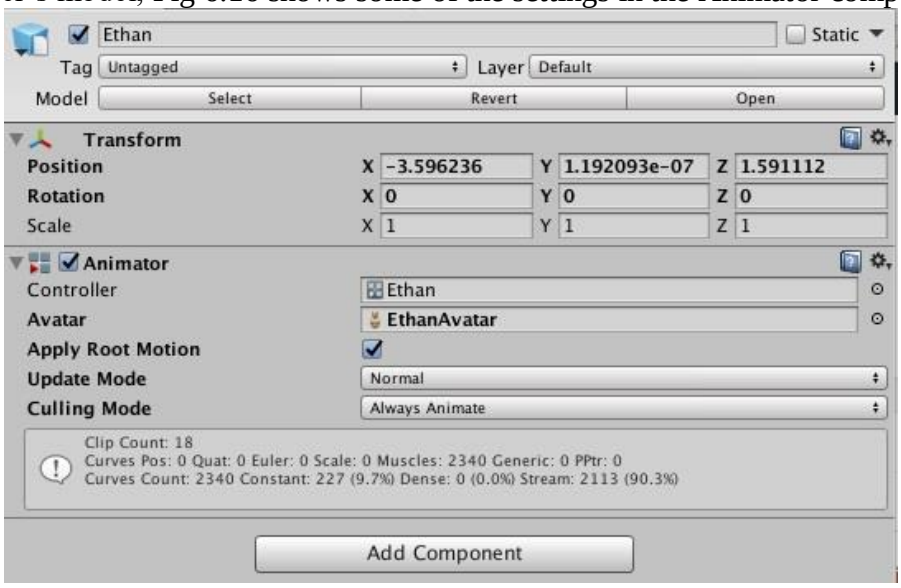


Figure 6.16 Animator Settings

The "Apply Root Motion" Boolean determines if Unity will use the movement in the animation or a script. For example, say I have a script that makes my character move forward. I have an animation attached of the character running in place. Since the script is moving the character not the animation then I would want Root Motion set to false.

With “Update Mode” you can determine how you want your animations to play. Such as with physics, with a fixed speed, or just normal. Culling mode allows you to determine if you want to keep animating even off screen.
The character can move from one animation to another and change its state using State Machine Diagram an example is shown in Fig 6.17.
The basic idea is that a character is engaged in some particular kind of action at any given time. The actions available will depend on the type of gameplay but typical actions include things like idling, walking, running, jumping, etc. These actions are referred to as **states**, in the sense that the character is in a “state” where it is walking, idling or whatever. In general, the character will have restrictions on the next state it can go to rather than being able to switch immediately from any state to any other. For example, a running jump can only be taken when the character is already running and not when it is at a standstill, so it should never switch straight from the idle state to the running jump state. The options for the next state that a character can enter from its current state are referred to as state transitions. Taken together, the set of states, the set of transitions and the variable to remember the current state form a **state machine**.

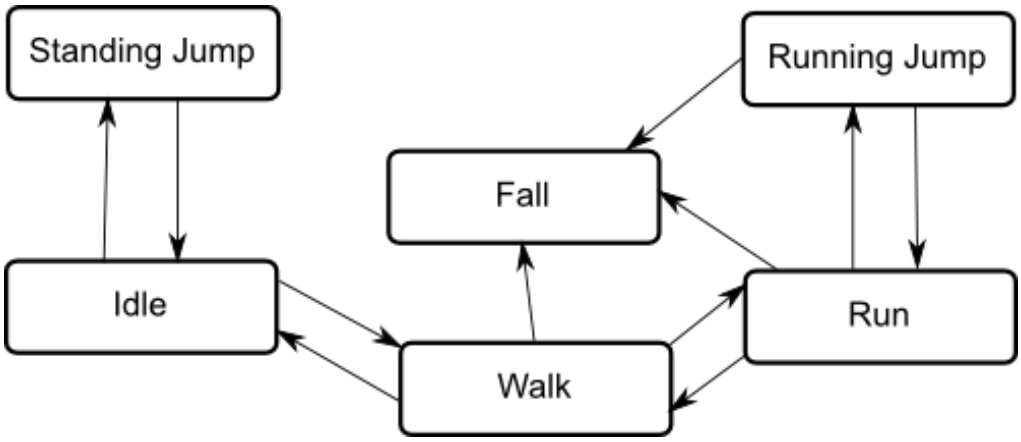


Figure 6.17 State Machine Diagram

To make our state machine we drag our states into the animator tab shown in Fig 6.18, the orange one is the first one you selected in your Project tab. That is known a “Default State”. We can also change this default state by right clicking the desires state and choosing “Set as Layer Default State”

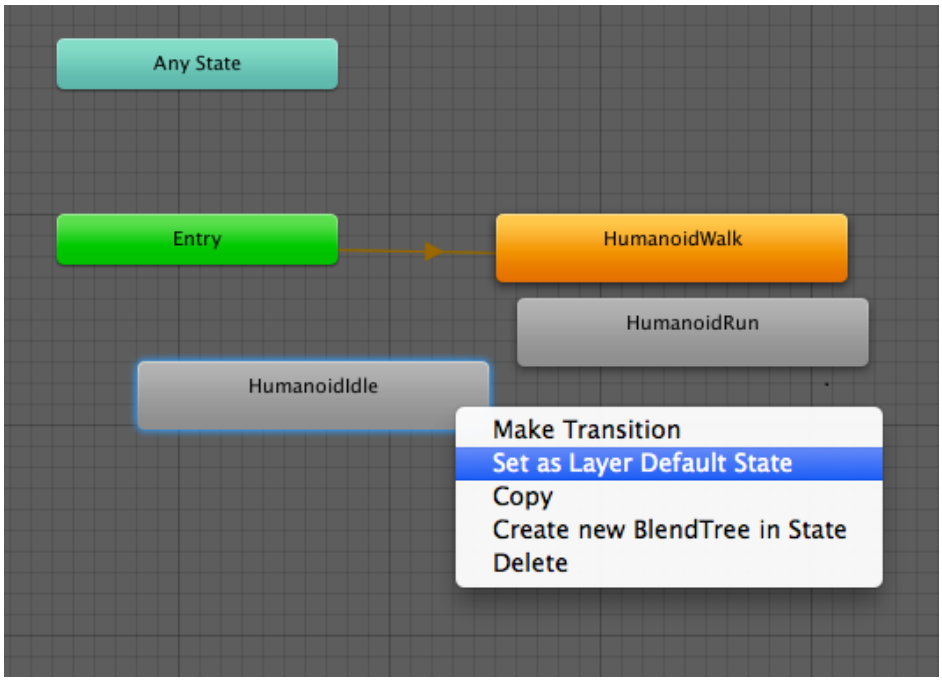


Figure 6.18 Animator State Diagram

The character is by default in that state and if we want to make a transition to another state, right click on the Default State and select "Make transition". Then click on the desired state.
So now, if we play our animation we will find that the character is idle for some time and then starts the next state for example "Walking".

We have our model animating but it lacks control. It idles and runs without us doing anything. The way we add control is through Parameters. Parameters are variables that are defined within the animator that scripts can access and assign values to; this is how a script can interact with the Animator. There are three types: Int (or Integer), Float, Boolean, and Trigger. All of them are self-explanatory except for Trigger. A trigger is like a Boolean but as soon as it is set to true it immediately goes back to false. Parameters can be assigned values from a script using functions in the Animator class: SetFloat, SetInteger, SetBool, SetTrigger and ResetTrigger.
By using parameters, we can check if a Boolean flag is true or false and transit to another state according to the value of flag.

6.5.2 Animation Layers

Unity uses Animation Layers for managing complex state machines for different body parts. An example of this is if you have a lower-body layer for walking-jumping, and an upper-body layer for throwing objects / shooting.
You can manage animation layers from the Layers Widget shown in Fig 6.19 in the top-left corner of the Animator Controller

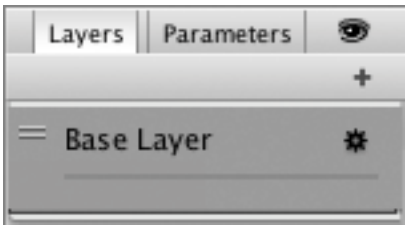


Figure 6.19 Layers Tab

Clicking the gear wheel on the right of the window shows you the settings as seen in Fig 6.20 for this layer.

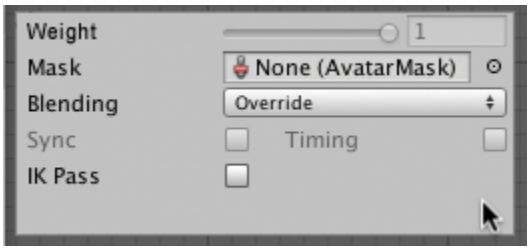


Figure 6.20 Layer Settings

On each layer, you can specify the mask (the part of the animated model on which the animation would be applied), and the Blending type. Override means information from other layers will be ignored, while Additive means that the animation will be added on top of previous layers.
You can add a new layer by pressing the + above the widget.
The Mask property is there to specify the mask used on this layer. For example, if you wanted to play a throwing animation on just the upper body of your model, while having your character also able to walk, run or stand still at the same time, you would use a mask on the layer which plays the throwing animation where the upper body sections are defined an example is shown Fig 6.21.

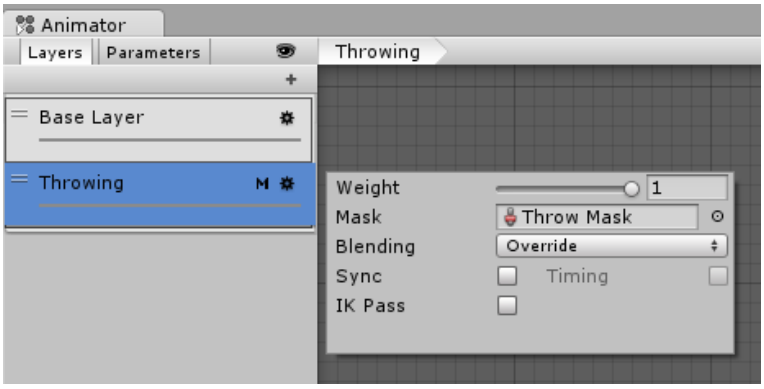


Figure 6.21 Controlling Layers

An ‘M’ symbol is visible in the Layers sidebar to indicate the layer has a mask applied.

6.5.3 Animations used in this Project

We have two layers in the project, the Base Layer and the Face Layer. The **Base layer** shown in Fig 6.22 is responsible for the body movements of the characters it consists of three states: Idle Sitting, Raise Hand, and Talk.

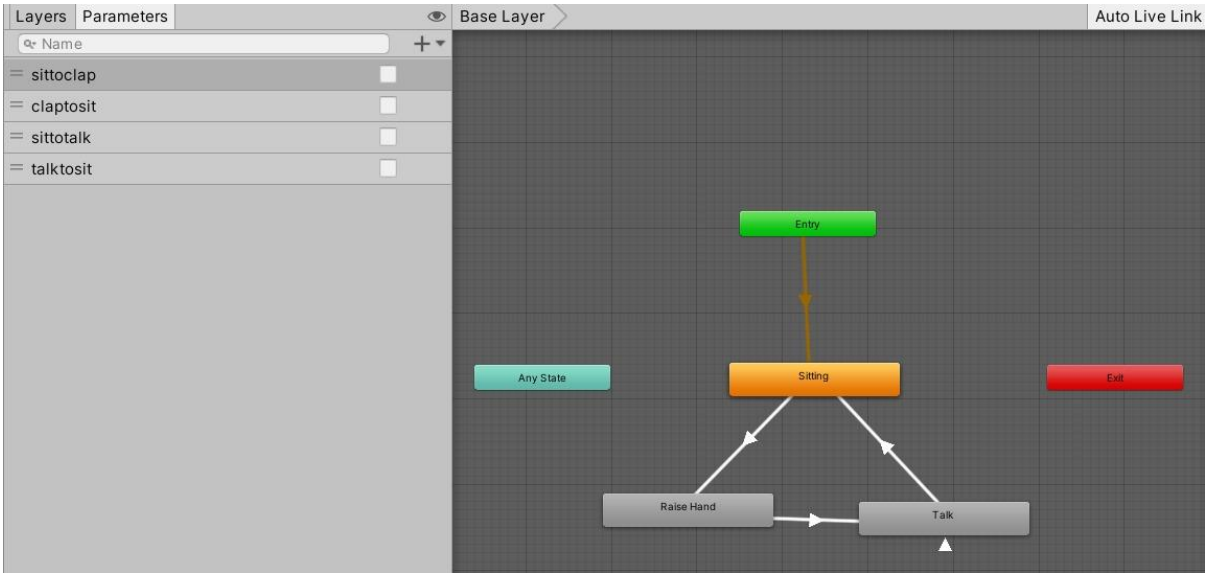


Figure 6.22 Base Layer

- **Idle Sitting**
  Idle sitting is the animation that works for most of the application, where characters are basically sitting in an idle position, the Mixamo animation makes them appear as it they are breathing, and this animation works on all characters unless it is invoked by any other flag that causes it to enter into another state of the state machine.

- **Raise Hand**
  This animation in Fig. 6.23 is invoked if the sit-to-talk flag, which is a Boolean parameter, here the character raises its hand to ask a question.

Figure 6.23 Raise Hand

- **Talk**
  This animation in Fig 6.24 is invoked after the Raise Hand animation is exited, here the character starts imitating talking and asks the question we got from the model.

Figure 6.24 Talking

The **Face layer** shown in Fig 5.25 is responsible for the Facial Expressions of the characters it consists of three states: Neutral, Confused, Sad, Happy, and Bored.

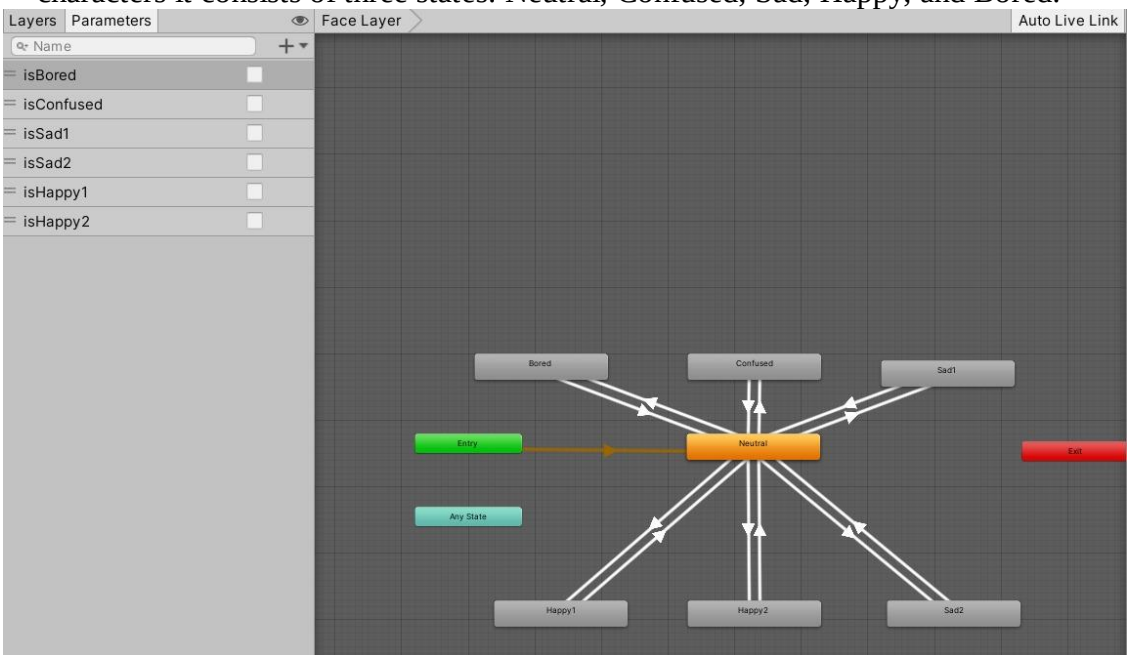


Figure 6.25 Face Layer

We transit from these states according to the speech rate, if it is a good rate the is-Happy Boolean is set to true and the character makes a happy face in Fig 6.26 to reflect satisfaction.

Figure 6.26 Happy Faces

But if the speech rate is too slow or too fast the is-Sad Boolean is set to true and the facial expressions reflect not being satisfied as seen in Fig. 6.27.

Figure 6.27 Unsatisfied Faces

6.6 Optimization

The hall scene is finally complete and filled with characters but it is not optimized at all and it is even laggy with low frame late even when run on PC, which means it will only run slower on android devices especially older models.

The reason for this laggy performance because we are making a lot of calls to the GPU card to draw every single object in our scene every single frame, which is consuming and we can reduce those calls greatly with some techniques, which we will discuss in the part of the chapter.

6.6.1 Draw Calls

We have to first analyze what are draw calls and how many of them are too bad or more than what the GPU can handle, to draw a GameObject on the screen, the engine has to issue a draw call to the graphics API (such as OpenGL or Direct3D). Draw calls are often resource-intensive, with the graphics API doing significant work for every draw call, causing performance overhead on the CPU side. This is mostly caused by the state changes done between the draw calls (such as switching to a different Material), which causes resource-intensive validation and translation steps in the graphics driver.

Now that we now what is a draw call, it is obvious that we have to reduce their number as much as possible to get a smoother feeling to our application with more than 50-60 fps. On computer devices we can get that frame rate with as many as 1000-2000 draw calls made from Unity, but that is not the case for mobile devices as we are limited to about 120-160 draw calls and that is only available on the newer models of mobile devices which we are targeting currently.

The lecture hall now uses about 2200 draw calls, we have to reduce that to be in the range of 120-160 draw calls, and we are going to discuss how to achieve that in the following bits of this section.

6.6.2 Draw Calls Reduction Techniques

Fig. 6.28 shows the current state of the application as seen from stats window in Unity. We can see we have 2216 draw calls and the objects drawn consist of 6.6 million vertices. We have to reduce both the draw calls and number of vertices. The FPS is currently in the range 60-80, but on android, it does not reach 15.

Figure 6.28 Scene Stats

Now we begin to explore how to make our application suitable for mobile devices via some simple methods that can reduce the draw calls greatly without affecting the quality too much.

- **Camera**
  The camera controls the quality of the drawn images so we can reduce that quality a bit since the mobile devices are usually so small so it won't affect the quality that much. Fig 6.29 shows the camera settings and how we can choose a different rendering path.

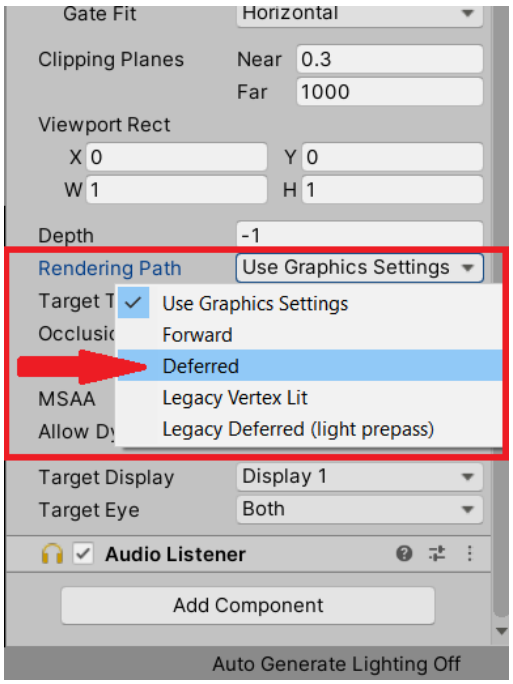


Figure 6.29 Camera Rendering Path

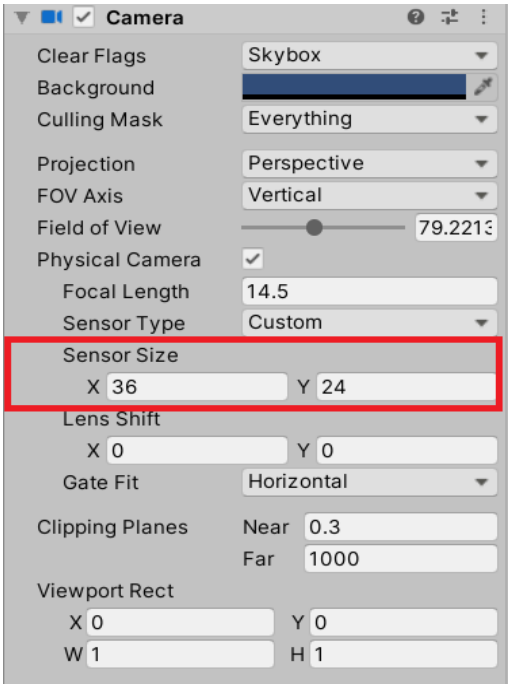


Figure 6.30 Camera Sensor Size

The camera angle controls how much we see so if we can reduce the angle and cover some object this means we can remove those objects from our draw calls. We can modify this setting in the camera settings shown in Fig. 6.30.

Fig. 6.31 shows the new stats we achieved after modifying the previous settings and we are already off to a great start.

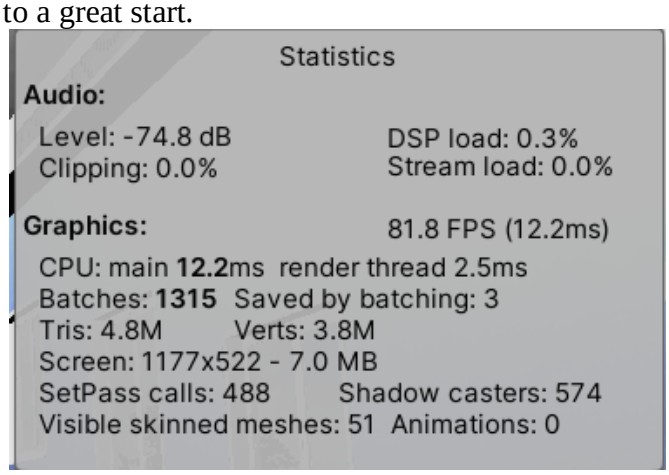


Figure 6.31 Modified Stats

- **Static Objects**

  Each object is responsible for a draw call but the scene is static meaning that no object in it changes its position as time passes. We can take advantage of that by marking the scene objects as static – except characters – in the object settings shown in Fig 6.32 to minimize draw calls. If an object is marked as static it's properties get calculated and drawn only once as we always know it's position.

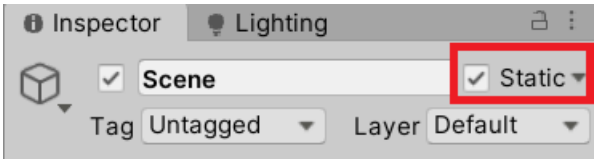


Figure 6.32 Static Object

- **Mesh Combining**
  As we mentioned before every object get a draw call but some objects are identical to each other like the chairs, walls and others. We can take advantage of that and combine those objects or combine their meshes to have them be drawn as a single object at run time.

The idea discussed above can be achieved in two ways. The first is to design the object as a single item but that is not always possible and hard to achieve. The second method is easier and achieved through a simple script shown in Fig. 6.33.

```
void combineMeshesByCode()
{
    MeshFilter[] meshFilters = GetComponentsInChildren<MeshFilter>();
    CombineInstance[] combine = new CombineInstance[meshFilters.Length];

    int i = 0;

    Matrix4x4 myTransform = transform.worldToLocalMatrix;
    while (i < meshFilters.Length)
    {
        combine[i].mesh = meshFilters[i].sharedMesh;
        combine[i].transform = myTransform * meshFilters[i].transform.localToWorldMatrix;
        meshFilters[i].gameObject.SetActive(false);

        i++;
    }
    MeshFilter mFilter = this.GetComponent<MeshFilter>();
    mFilter.mesh = new Mesh();
    mFilter.mesh.CombineMeshes(combine, true);
    mFilter.mesh.RecalculateBounds();
    mFilter.mesh.RecalculateNormals();
    mFilter.mesh.Optimize();
    this.transform.gameObject.SetActive(true);

}
```

Figure 6.33 Mesh Combining Script

The main advantage of this script is that if we 1000 identical objects they would cause 1000 draw calls but if we combine them they would all require only a single draw call.

- **Mesh Simplification**
  Surface mesh simplification is the process of reducing the number of faces used in a surface mesh while keeping the overall shape, volume and boundaries preserved as much as possible. Although it does not decrease draw calls count but it does have two features it simplifies the amount of work done in a draw call since the mesh is simpler than it was. The second and more important feature is that Unity cannot combine meshes with more than 64,000 Vertices so sometimes mesh simplification is a necessary step before combining meshes. Fig. 6.34 shows a package called “Mesh Simplify” and the settings as it reduces a mesh and the percentage of vertices we want to keep of the original mesh.

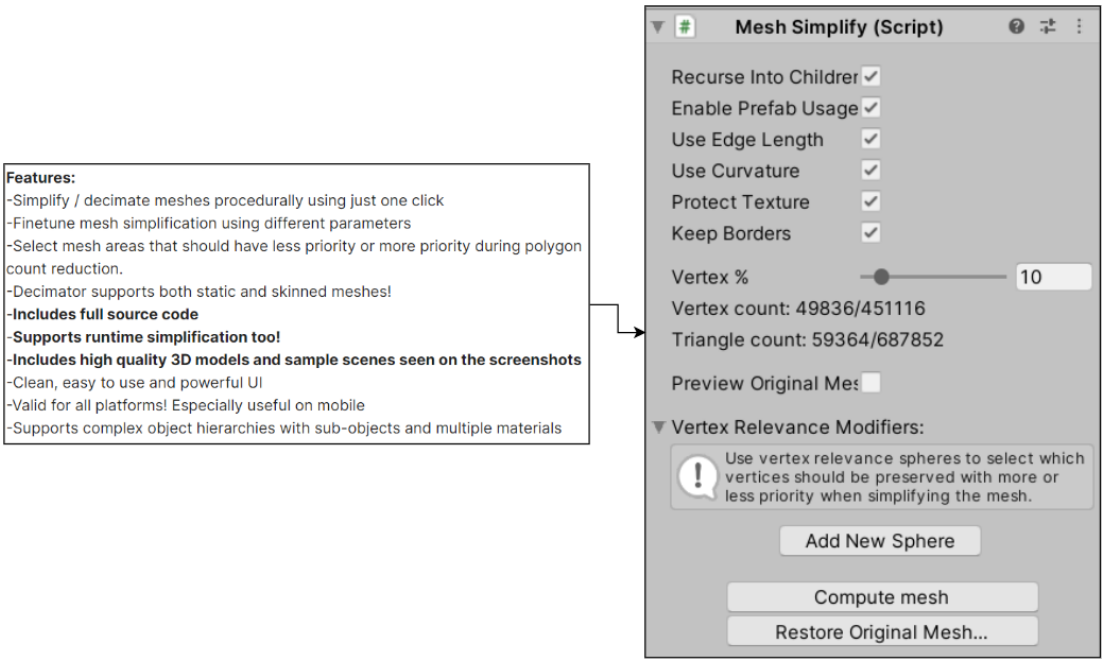


Figure 6.34 Mesh Simplify Features

- **Materials**
  Similar to static objects we can allow GPU to make an instance of that material and use it one time to draw all the objects in the scene that use this material all in a single swoop. This is done by enabling GPU instancing in materials settings shown in figure 6.35.

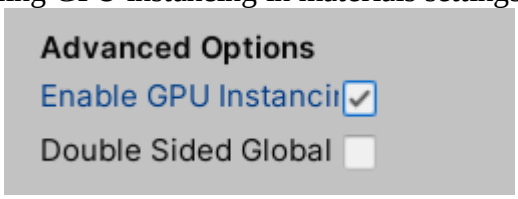


Figure 6.35 GPU Instancing

Another enhancement that we can do is choose a suitable shader type. A shader is a type of computer program originally used for shading in 3D scenes. Depending on the target platform, we can change the shader to a less memory consuming option, in our case we can choose Mobile/Diffuse shader. Different options for shaders are shown in figure 6.36.

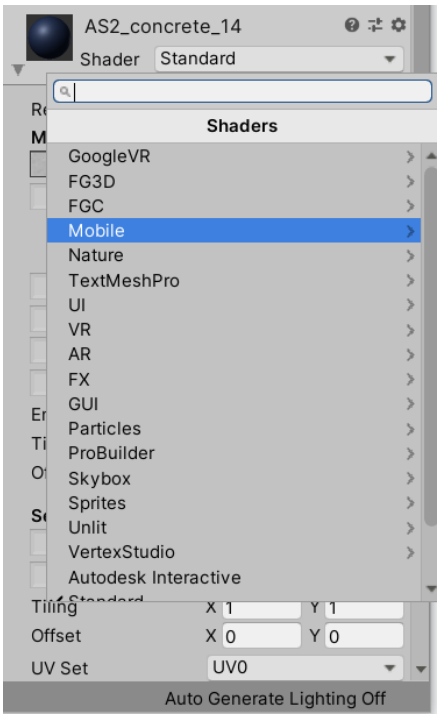


Figure 6.36 Shader Types

- **Lighting**
  The last thing we can enhance is the light settings. Light directions and shadows are calculated during runtime and that adds an overhead to the draw calls but our scene is mostly static so we always know where is the light hitting the objects and because of that we can set the light mode to Baked which means the light is calculated only once as we always know it's settings. We can also turn of the shadows as they add an overhead to the calculations as well. Fig 6.37 shows the modified light settings we are using.

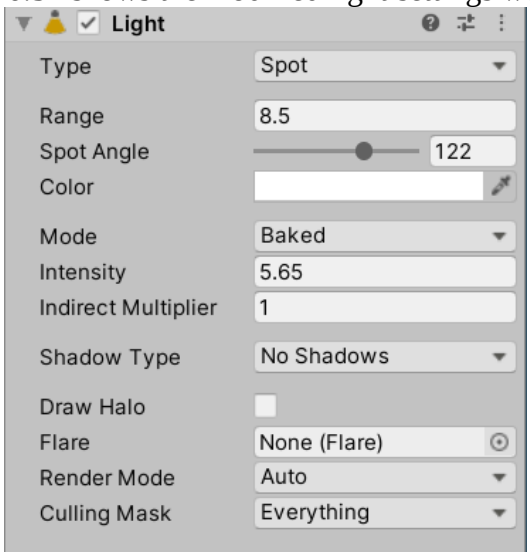


Figure 6.37 Light Settings

6.6.3 Conclusion

Now it is time to reflect back on what we discussed in optimization and how it affected the performance of our application. First, we need to understand that it is a tradeoff between quality and performance and we need to figure the perfect balance to reach best performance with best quality. Fig 6.38 shows a comparison between the numbers we started with as shown previously in Fig 6.28. These results are achieved by combining all of the above techniques.

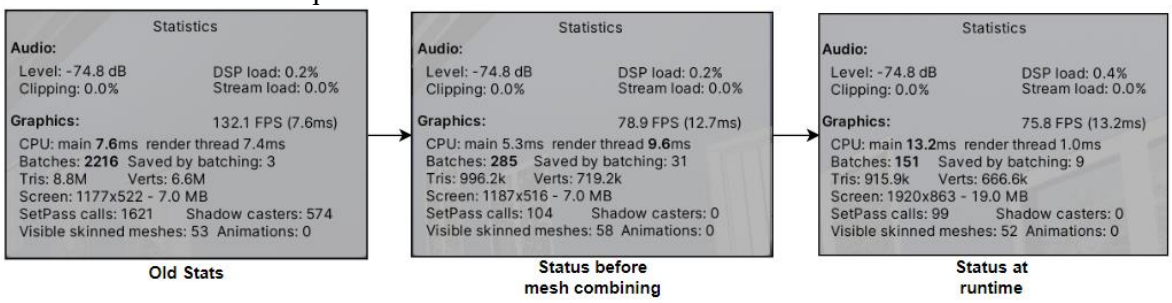


Figure 6.38 Modified Stats

6.7 Final Presentation

The scene is now all finished and functional but we need to polish it for a better user experience. The first thing we notice that we launch directly into the hall scene with no start screen. We can easily fix this by adding a simply starting screen or room in our case since we are in VR and add a start button and some simple options. The second thing we are going to do is add tree and a sky to our lecture scene to give more of real life feeling.

6.7.1 Starting Room

Fig 6.39 shows a simple hollow cube with our buttons and instructions all laid out on the walls of sides of that cube.

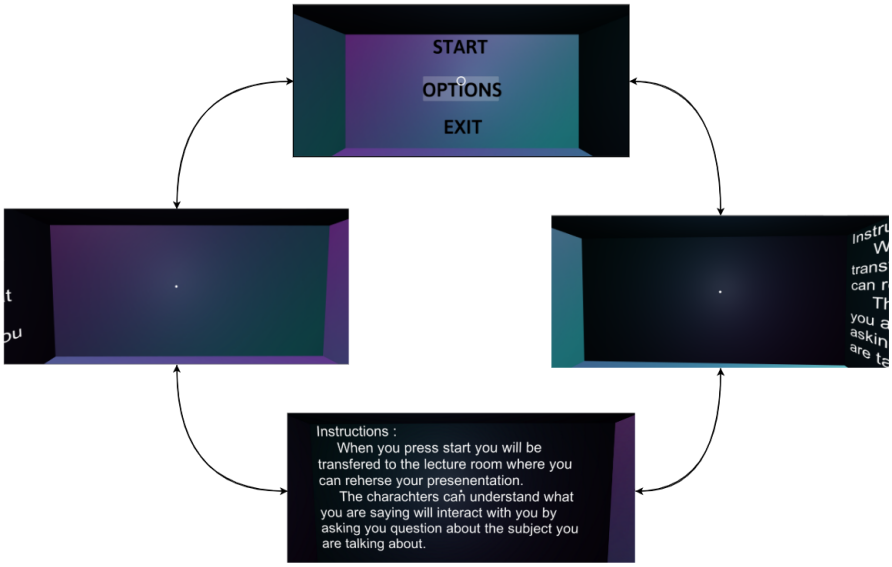


Figure 6.39 360° Rotation of Starting Room

As for the options menu, Fig 6.40 shows another 360 view of the starting menu after we choose the click the menu button.

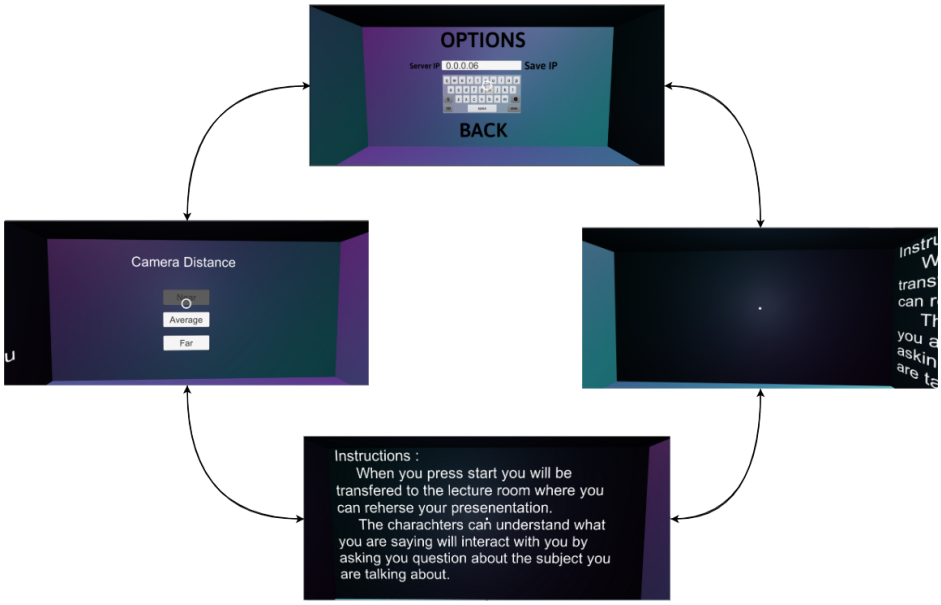


Figure 6.40 360° Rotation of Options Screen

The server IP is currently static so we want to change it during run time to be able to receive question from the UNILM model.
The camera distance refers to the distance between the distance between the camera and the first line of audience and the user is able to change based on their liking and level of comfort.

6.7.2 Main Scene Polishing

To make the user make more at a real life scene we added some simple items to make it more realistic below we explore those items:

- Tree and a Skybox
- Laptop image
- Textbox for feedback on speech's speed

Fig. 6.41 shows the final scene image during runtime.

Fig 6.41 Final Scene

Fig 6.42 shows the application on an android device.

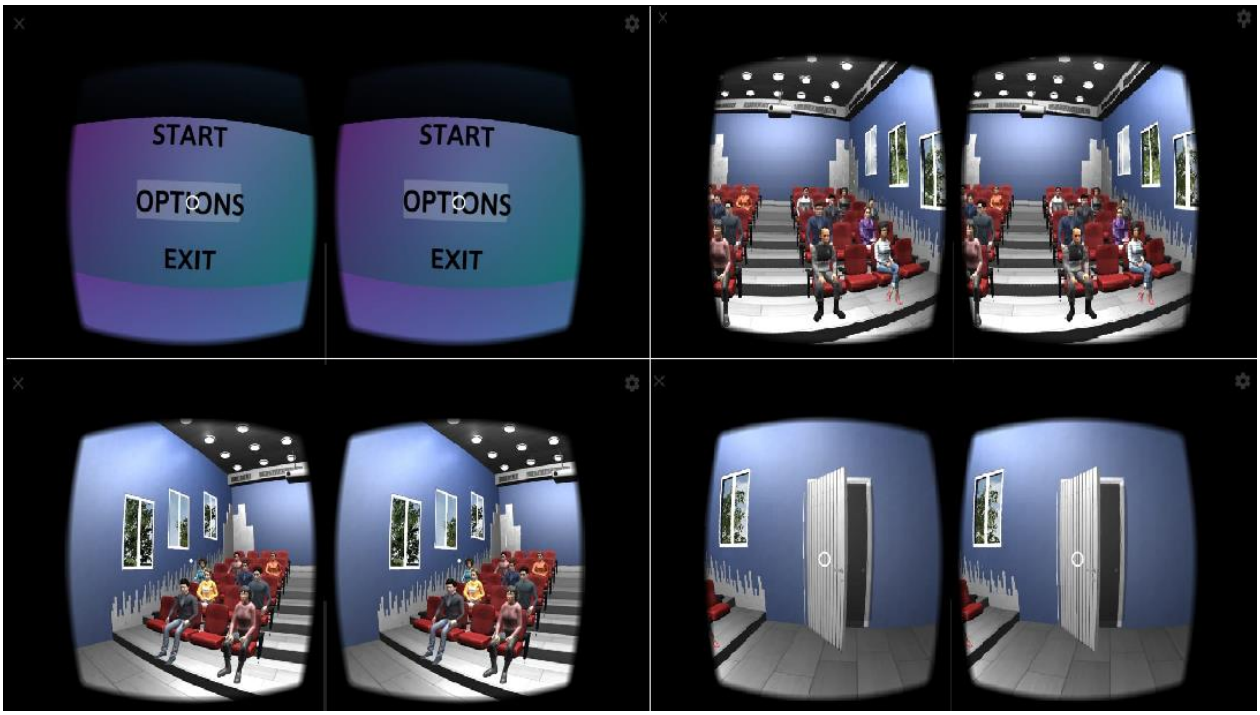


Figure 6.42 Mobile Application

REFERENCES

Websites

- https://cloud.google.com/speech-to-text
- https://cloud.google.com/speech-to-text/docs/basics
- https://www.youtube.com/watch?v=mNSQ-prhgsw
- https://medium.com/@jonathan_hui/speech-recognition-gmm-hmm-8bb5eff8b196
- https://en.wikipedia.org/wiki/Hidden_Markov_model#Applications
- https://en.wikipedia.org/wiki/Hidden_Markov_model#:~:text=Hidden%20Markov%20Model%20(HMM)%20is,to%20learn%20about%20by%20observing%20.
- https://en.wikipedia.org/wiki/Mel-frequency_cepstrum
- https://medium.com/prathena/the-dummys-guide-to-mfcc-aceab2450fd
- https://www.quora.com/What-are-the-best-algorithms-for-speech-recognition
- https://sonix.ai/history-of-speech-recognition#:~:text=1950s%20and%2060s,to%2016%20words%20in%20English.
- https://en.wikipedia.org/wiki/Timeline_of_speech_and_voice_recognition
- https://www.ibm.com/eg-en/cloud/watson-text-to-speech
- https://cloud.google.com/text-to-speech/docs
- https://aws.amazon.com/polly

Papers

- Neural Question Generation from Text: A Preliminary Study Qingyu Zhouy∗ Nan Yangz Furu Weiz Chuanqi Tan] Hangbo Baoy Ming Zhouz yHarbin Institute of Technology, Harbin, China 2017
- Attention Is All You Need 2017
- BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding Jacob Devlin Ming-Wei Chang Kenton Lee Kristina Toutanova
- Google AI Language (2018)
- Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 2227–2237, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.
- Unified Language Model Pre-training for Natural Language Understanding and Generation Li Dong∗ Nan Yang∗ Wenhui Wang∗ Furu Wei∗y Xiaodong Liu Yu Wang Jianfeng Gao Ming Zhou Hsiao-Wuen Hon 2019