HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

PROJECT I REPORT

Topic: Question Answering on SQuAD

Student	Nguyễn Trung Hiếu
Student ID	20204909
Project Class	721018
Lecturer	Assoc. Prof. Dr. Lê Thanh Hương

HÀ NỘI, 2023

Contents

Abstract

Machine Reading Comprehension, a form of Machine Question Answering, is a challenging research field with wide real life applications. Recently, pre-trained contextual embeddings models like Bidirectional Encoder Representations from Transformers (BERT) and variants have attracted much attention for their performance in a wide variety of natural language processing tasks. In this project, we fine-tuned the A Lite BERT (ALBERT) models and implemented a number of additional task-specific layers (e.g. attention layer, highway layer) to improve model performance with respects to the benchmark Stanford Question Answering Dataset (SQuAD 2.0).

Five different models with different layers on top of ALBERT-base model and one model based on ALBERT-large were implemented and tested in comparison to the baseline model (ALBERT-base-v2). Our best-performing single model is ALBERT-base-v2 + LSTM Encoder + Highway + LSTM Decoder + Linear-out, which achieved an F1-score of 81.45 on the dev set. Finally, we considered the answer behaviour of the models in response to the questions.

Keyword reading comprehension, SQuAD, ALBERT.

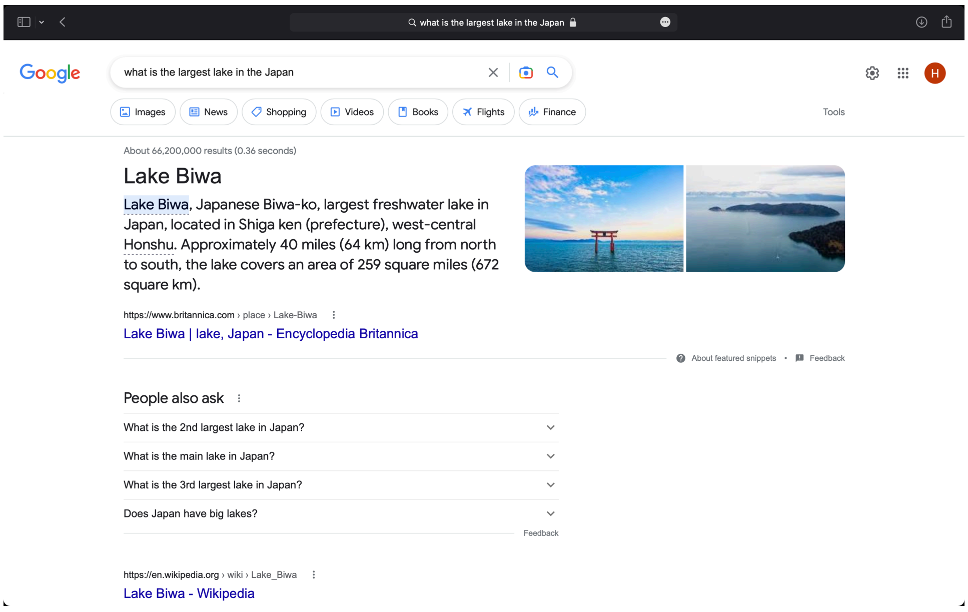
Content

1. Introduction

Machine Reading Comprehension Teaching computer to read the text and understand the meaning of the text has long been a critical research goal in the Natural Language Processing (NLP) history. Recently, with the emergence of large-scale datasets, higher computing power, and the development of deep learning techniques, researchers have made significant progress in many aspects of Machine Reading Comprehension (MRC), leading to a boost to the whole NLP area.

A MRC task typically involves the ability of a machine to understand a given text (called *context*) and provide accurate answers to questions relating to that text. The concept of MRC comes from the human understanding of text, where a common way to test comprehension is to require answering questions about the text. Reading comprehension is thus a natural way to evaluate a computer's language understanding ability.

Machine Reading Comprehension has many potential applications in NLP systems, including search engines and dialogue systems. For example, search engines and chatbots can use the technique to directly return the correct answer to a question by highlighting it in context. MRC can improve the performance of these systems by allowing users to quickly get the right answer to their questions, or to reduce the workload of customer service staff.



what is the largest lake in the Japan

About 66,200,000 results (0.36 seconds)

Lake Biwa

Lake Biwa, Japanese Biwa-ko, largest freshwater lake in Japan, located in Shiga ken (prefecture), west-central Honshu. Approximately 40 miles (64 km) long from north to south, the lake covers an area of 259 square miles (672 square km).

https://www.britannica.com › place › Lake-Biwa
Lake Biwa | lake, Japan - Encyclopedia Britannica

What is the 2nd largest lake in Japan?
What is the main lake in Japan?
What is the 3rd largest lake in Japan?
Does Japan have big lakes?

https://en.wikipedia.org › wiki › Lake_Biwa
Lake Biwa - Wikipedia

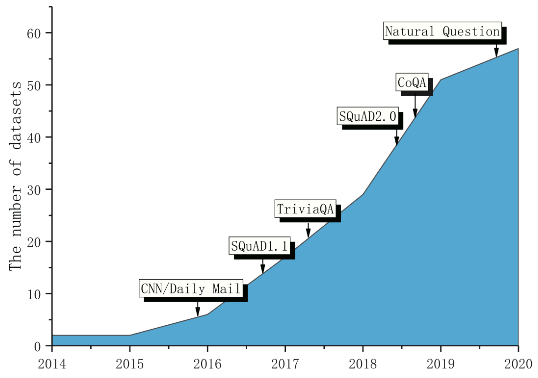
Figure 1.1: An example of MRC – Google can highlight the answer directly in the context

History of MRC Approaches Machine reading comprehension has a long history of development, dating back to the 1970s and 1990s, when Lehnert et al. and Hirschman et al. respectively created question-answering systems. During this period, most MRC systems were rule-based or statistical models. However, due to the lack of high-quality datasets, this field was neglected for some time.

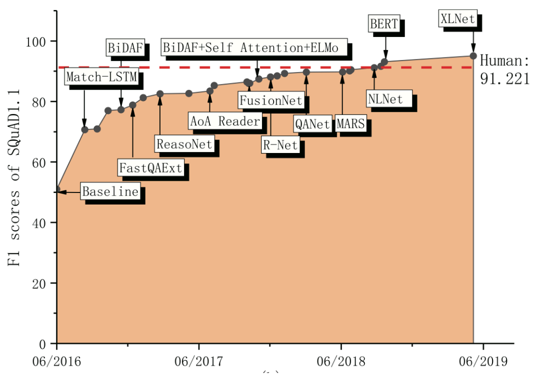
In 2013, a new dataset called MCTest (containing 500 stories and 2000 questions) was created, and researchers began to apply machine learning models to it despite that the original baseline of MCTest is a rule-based model and the number of training samples in the MCTest dataset is not large.

A turning point for the field came in 2015, when Hermann et al. defined a new dataset generation method that provided large-scale supervised reading comprehension datasets. They also developed a class of attention-based deep neural networks that learn to read real documents and answer complex questions with minimal prior knowledge of language structure. Since then, with the emergence of various large-scale supervised datasets and neural network models, the field of machine reading comprehension has entered a period of rapid development.

In recent years, there has been a surge in the creation of MRC benchmark datasets, as shown in Figure 1.2a, with an exponential increase from 2014 to 2020. This increase has inspired the development of numerous neural MRC models, as seen in Figure Figure 1.2b. For example, the development of the SQuAD 1.1 dataset has led to the creation of many state-of-the-art neural network models, including BiDAF, ELMo, BERT, RoBERTa, and XLNet. They have already surpassed human performance in the MRC benchmark datasets.



(a) The cumulative number of MRC datasets from the beginning of 2014 to the end of 2019.



(b) The progress of state-of-the-art models on SQuAD 1.1 since its release.

Figure 1.2: The number of MRC datasets created in recent years and the F1 scores of state-of-the-art models on SQuAD 1.1

The Project Our project focuses on the Stanford Question Answering Dataset (SQuAD 2.0), which approximates real-world reading comprehension circumstances. Models that perform well on SQuAD can be considered as a reliable benchmark for solving reading comprehension and question-answering problems. Although many teams have already made rapid progress on this dataset, most researchers are still striving to develop better models that can (1) adapt well to questions without answers and (2) outperform human oracles ($F1 : 89.452$).

Moreover, in recent years, pre-trained models have been instrumental in improving the performance of many NLP tasks, even those with limited training data. Academic research has demonstrated that large networks are crucial for achieving state-of-the-art results. As a result, it has become commonplace to pre-train large models and then distill them down to smaller ones for use in real-world applications.

Thus, in the project, we developed a model for the SQuAD 2.0 dataset by building upon the ALBERT models. We replaced the linear output layer with either an encoder-decoder architecture or a highway network, which led to successful implementation of models that performed well on the QA task. We also experimented with larger ALBERT models (ALBERT-large) and achieved better results. Our best single model achieved an EM score of 78.11 and an $F1$ score of 81.45 on the dev set.

Lastly, we also explored how the models give the answer to the questions. What makes SQuAD 2.0 more representative of real-world situations compared to its predecessor SQuAD 1.1 is the addition of unanswerable questions, i.e. cases in which what is answered is not in the paragraph. Also, we examined what the answers generally look like in cases of 5W1H questions, "What", "Who", "When", "Where", "Why" and "How", whose answers are considered basic in gathering information. These behavioural analyses can help us gain a deeper understanding on the mechanism of the models and their component layers.

Motivations The motivation of this project is multi-fold:

- Reading comprehension have become an important application of machine learning. We want to explore some general ideas about MRC, gaining more helpful perspectives on this field.
- We wish to examine pre-trained contextual embeddings models in depth, specifically, the BERT family.
- In addition to traditional measures, we want to examine the answer behaviour of the models to get a field-centric view of reading comprehension success.

Contributions The contributions are summarised below:

- We explored the nature of BERT-based approaches and additional task-related layers.

- We applied these approaches with a widely used benchmark dataset to find out more about their behaviour.
- We discussed some behavioural characteristic of recommender systems, besides conventional metrics such as exact match and $F1$.

Outline The outline is as follows. In Chapter 2, we give an outline of the MRC problem, briefly review the history of MRC approaches and give a theory about the BERT models and additional layers, such as encoder layers and highway layers. Chapter 3 then details the implementations of the models used in the project. Next, Chapter 4 focuses on the experiments and hyperparameter tuning that we conducted on the models. The quantitative and behaviour analyses are afterward dealt with in Chapter 5. Finally, in Chapter 6, we give a comprehensive conclusion and present possible future works.

2. Literature Review

2.1 Problem Formulation

2.1.1 Definition

The Machine Reading Comprehension research includes a variety of specific tasks, such as textual MRC and multi-modal MRC. As most machine reading comprehension tasks take the form of question answering, however, the textual QA-based MRC is considered a typical example of a MRC task. A common definition of MRC task is given below,

> **Definition 2.1 — MRC task.** Typical MRC task could be formulated as a supervised learning problem. Given a collection of textual training examples $\{(p_i, q_i, a_i)\}_{i=1}^n$, where p is a passage of text, and q is a question regarding the text p. The goal is to learn a predictor f which takes a passage of text p and a corresponding question q as inputs and gives the answer a as output, which could be formulated as the following formula:
>
> $$a = f(p, q) \tag{2.1}$$
>
> and it is necessary that a majority of native speakers would agree that the question q does regarding that text p, and the answer a is a correct one which does not contain information irrelevant to that question.

2.1.2 Classifcations

In many research papers, MRC tasks are categorised into the following four types:

- **Cloze styles:** There are some placeholders in the question that need to be filled with suitable words or phrases according to the context content.
- **Multiple-choice:** In a multiple-choice task, the MRC system needs to select a correct answer from a set of candidate answers according to the provided context.
- **Span prediction:** In a span prediction task, the answer is a span of text in the context. We need to select the correct beginning and end of the answer text from the context.
- **Free-form answer:** This kind of tasks allows the answer to be any free-text forms, that is, the answer is not restricted to a single word or a span in the passage.

The current classification method for MRC tasks arguably has some limitations. Some tasks can be classified into multiple types above, leading to ambiguity. To iillustrate, a task may have questions in cloze styles but the answers are multiple-choice. Additionally, new MRC tasks have emerged, such as multi-modal MRC, which require models to understand the semantics behind text and visual images, but ignored by the current method.

This traditional method has thus been challenged, for example, Zeng et al. proposes that MRC tasks be characterised using four criteria [1]:

- **Type of corpus:** Texual, or multimodal (including sound, image).
- **Type of questions:** Natural text, placeholders or synthesis (a list of answers, often shorts, just a combination of words and may not follow grammatical rules).
- **Type of answers:** Natural (a natural word, phrase, sentence or image) or multi-choices (serie of candidate answers).
- **Sources of answers:** Spans or free-form (the answer may be any phrase, word, or even image and does not have to be in the context).

Zeng et al. also collected 57 different MRC tasks and divides them into the four attributes [1]. The result show the textual tasks still accounts for nearly 90%, showing that multi-modal tasks still have much room for expansion in the future. In terms of question types, the natural form is the most common at around 70%. In terms of answer types, the natural type ($\approx$ 53%) slightly outweighs the multiple choices type. Finally, in terms of answer source, 29.82% of the answers are of spans type, and the rest are of free-form.

2.1.3 Span detection problem

In the 'classical' classification, SQuAD 2.0 is a span detection problem. The SQuAD 2.0 dataset, the main dataset of this project, has the textual corpus, natural text questions, and answers that are natural text and confined in a span within the context (in case there is answer); making it a very typical example of MRC dataset.

Context	The Normans (Norman: Nourmands; French: Normands; Latin: Normanni) were the people who in the *10th and 11th centuries* gave their name to Normandy, a region in France. They were descended from Norse ("Norman" comes from "Norseman") raiders and pirates from Denmark, Iceland and Norway who, under their leader Rollo, agreed to swear fealty to King Charles III of West Francia. [...]
Question	When were the Normans in Normandy?
Answer	10th and 11th centuries

Figure 2.1: An example question in SQuAD 2.0

The Stanford Question Answering Dataset (SQuAD) is a set of reading comprehension tasks. It consists of questions asked by collaborators about Wikipedia articles, where the answer to each question is a segment of text, or span, from the corresponding passage. In some cases, the question may not have an answer within the passage [2].

SQuAD2.0 is a collection of 100,000 questions from SQuAD1.1 along with more than 50,000 unanswerable questions intentionally crafted by crowdworkers to appear like answerable ones. The challenge for systems in SQuAD2.0 is not only to provide answers but also to recognize when there is no support for an answer in the paragraph, and to refrain from answering in such cases.

Context	There are infinitely many primes, as demonstrated by Euclid around 300 BC. There is no known simple formula that separates prime numbers from composite numbers. However, the distribution of primes, that is to say, the statistical behaviour of primes in the large, can be modelled. The first result in that direction is the prime number theorem, proven at the end of the 19th century, which says that the probability that a given, randomly chosen number n is prime is inversely proportional to its number of digits, or to the logarithm of n.
Question	When was the random number theorem proven?
Answer	

Figure 2.2: An example unanswerable question in SQuAD 2.0

During the dataset creation, the unanswerable questions must be formed adhering to the following goals. First, they should appear related to the topic of the context. Secondly, there should be some span in the context whose type match the type of answer that the question ask for (plausible answers). For instance, if the question asks, "What school did Harry Potter attend", then some school, correct or not, should appear in the context. This makes the dataset more closely resemble real-life situations as it forces the algorithms to not only select the span in the text that seems the most relevant to the question but also to check whether the text entails the answer [2].

Since its introduction, the SQuAD 2.0 dataset has become a benchmark dataset for span detection problem. It is commonly used in extractive reading comprehension, where a context is provided so that the model can refer to it and make predictions about where the answer is inside the context. Extractive reading comprehension is increasingly dominated by transformers, replacing recurrent neural networks. Transformers' training parallelisation allows training on larger datasets, leading to comprehensive pretrained systems which can then be fine-tuned for specific tasks through transfer learning.

2.2 Transformers

2.2.1 Background

Until 2017, the state-of-the-art approach for NLP problem was using recurrent neural network (RNN)-based or concurrent neural network (CNN)-based models that includes an encoder and a decoder, and an attention layer (a mechanism that determines the importance of words to other words in a sentence or which words are more likely to come together) that connects the encoder and the decoder.

In 2017, Vaswani et al. introduced a novel neural network architecture, Transformer [3]. Transformer is an encoder-decoder neural network characterised by the use of an attention and the non-use of RNN or CNN. While both the encoder-decoder architecture and the attention mechanism are themselves not new ideas, the novelty of the Transformer is that we can simply use attention to solve tasks that involve sequences and we do not need recurrent connections, which is an advantage, given that recurrent connections can hinder the parallelisation of the training process.

2.2.2 Model Architecture

The Transformer has an encoder-decoder structure. The encoder maps an input sequence of symbol representations $(x_1, \dots, x_n)$ to a sequence of continuous representations $\mathbf{z} = (z_1, \dots, z_n)$. Given $\mathbf{z}$, the decoder then generates an output sequence $(y_1, \dots, y_n)$ of symbols one element at a time. At each step the model is auto-regressive (cite), consuming the previously generated symbols as additional input when generating the next.

The Transformer follows this overall architecture using stacked self-attention and pointwise, fully connected layers for both the encoder and decoder, shown in the left and right halves of Figure 2.3, respectively.

Encoder and Decoder Stacks The encoder is composed of a stack of $N = 6$ identical layers. Each layer has two sub-layers. The first is a multi-head self-attention mechanism, and the second is a simple, position- wise fully connected feed-forward network. A residual connection is employed around each of the two sub-layers, followed by layer normalization. That is, the output of each sub-layer is LayerNorm$(x + \text{Sublayer}(x))$, where Sublayer(x) is the function implemented by the sub-layer itself. To facilitate these residual connections, all sub-layers in the model, as well as the embedding layers, produce outputs of dimension $d_{model} = 512$.

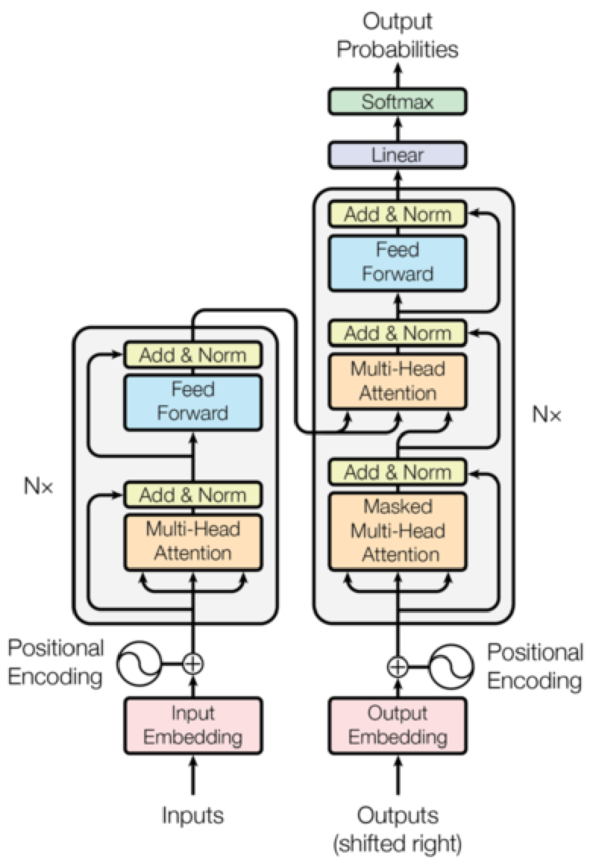


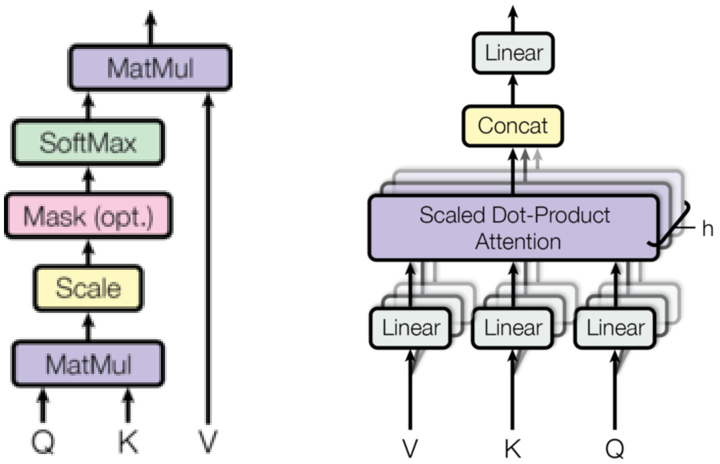
Figure 2.3: The architecture of Transformer

The decoder is also composed of a stack of $N = 6$ identical layers. In addition to the two sub-layers in each encoder layer, the decoder inserts a third sub-layer, which performs multi-head attention over the output of the encoder stack. Similar to the encoder, residual connections were employed around each of the sub-layers, followed by layer normalization. The self-attention sub-layer in the decoder stack is also modified to prevent positions from attending to subsequent positions. This masking, combined with fact that the output embeddings are offset by one position, ensures that the predictions for position i can depend only on the known outputs at positions less than i.

Attention An attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key.

The particular attention used is termed "Scaled Dot-Product Attention". The input consists of queries and keys of dimension d_k , and values of dimension d_v. The model computes the dot products of the query with all keys, divides each by $\sqrt{d_k}$, and applies a softmax function to obtain the weights on the values. We simultaneously compute the attention function on a matrix Q, set of queries, K keys, and V, values. The matrix of outputs is then:

$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V \tag{2.2}$$



(a) The attention (b) Multi-headed attention

Figure 2.4: Attentions

Instead of performing a single attention function with d_{model} -dimensional keys, values and queries, it is beneficial to linearly project the queries, keys and values h times with different, learned linear projections to d_k, d_k and d_v dimensions, respectively. On each of these projected versions of queries, keys and values, the attention function is then performed in parallel, yielding d_v-dimensional output values. These are concatenated and once again projected, resulting in the final values, as depicted in Figure 2.4b.

$$\begin{aligned} \mathrm{MultiHead}(Q, K, V) &= \mathrm{Concat}\left(\mathrm{head}_1, \ldots, \mathrm{head}_\mathrm{h}\right) W^O \\ \text{where head } &= \mathrm{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right) \end{aligned} \tag{2.3}$$

Where the projections are parameter matrices $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$.

The original authors employ $h = 8$ parallel attention layers, or heads. For each of these, they use $d_k = d_v = d_{\text{model}}/h = 64$. Due to the reduced dimension of each head, the total computational cost is similar to that of single-head attention with full dimensionality.

This kind of attention, termed "Multi-head Attention", allows the model to jointly attend to information from different representation subspaces at different positions. With a single attention head, averaging inhibits this.

The Transformer uses multi-head attention in three different ways:

- In “encoder-decoder attention” layers, the queries come from the previous decoder layer, and the memory keys and values come from the output of the encoder. This lets every position in the decoder attend over all positions in the input sequence, mimicking the encoder-decoder attention mechanisms in sequence-to-sequence models.
- The encoder contains self-attention layers. In a self-attention layer all of the keys, values and queries come from the same place, in this case, the output of the previous layer in the encoder. Each position in the encoder can attend to all positions in the previous layer of the encoder.
- Similarly, self-attention layers in the decoder allow each position in the decoder to attend to all positions in the decoder up to and including that position, preventing leftward information flow in the decoder to preserve the auto-regressive property. We implement this inside of scaled dot-product attention by masking out (setting to $-\infty$) all values in the input of the softmax which correspond to illegal connections.

Position-wise Feed-Forward Networks In addition to attention sub-layers, each of the layers in the encoders and decoders contains a fully connected feed-forward network, which is applied to each position separately and identically. This consists of two linear transformations with a ReLU activation in between.

$$\mathrm{FFN}(x) = \max\left(0, xW_1 + b_1\right) W_2 + b_2 \tag{2.4}$$

While the linear transformations are the same across different positions, they use different parameters from layer to layer. The dimensionality of input and output is $d_{\text{model}} = 512$, and the inner-layer has dimensionality $d_{ff} = 2048$.

Embeddings and Softmax Learned embeddings are used to convert the input tokens and output tokens to vectors of dimension d_{model}. The usual learned linear transformation and softmax function are also used to convert the decoder output to predicted next-token probabilities. In the model, the same weight matrix is shared between the two embedding layers and the pre-softmax linear transformation. In the embedding layers, those weights are multiplied by $\sqrt{d_{\text{model}}}$.

Positional Encoding Since the model contains no recurrence and no convolution, in order for the model to utilise the order of the sequence, some information about the relative or absolute position of the tokens in the sequence can be injected by adding “positional encodings” to the input embeddings at the bottoms of the encoder and decoder stacks. The positional encodings have the same dimension d_{model} as the embeddings, so that they can be summed.

Sine and cosine functions of different frequencies are used:

$$\begin{aligned} PE_{(pos,2i)} &= \sin\left(pos/10000^{2i/d_{\text{model}}}\right) \\ PE_{(pos,2i+1)} &= \cos\left(pos/10000^{2i/d_{\text{model}}}\right) \end{aligned} \tag{2.5}$$

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 $10000 \cdot 2\pi$. This function hypothetically allows 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}.

The sinusoidal version is chosen because it may allow the model to extrapolate to sequence lengths longer than the ones encountered during training.

2.2.3 Why Self-Attention

The use of pure self-attention mechanisms brings several benefits for transformer compared to RNN-based model. Transformer can process the tokens parallelly. Its encoders belong to a kind of feedforward neural nets, which have many encoding layers, each layer processes the tokens as a whole. On the other hand, in the LSTM (a RNN variant) architecture, the tokens must be processed sequentially. Moreover, a transformer can process a sequence from both sides without having to stack another LSTM layer as in a typical bidirectional LSTM layer. This means a reduced computational complexity for transformers, leading to a faster training process in comparison with RNN-based architectures.

Moreover, transformer addresses the vanishing gradient problem. Even though LSTMs and GRUs have memory mechanisms to regulate the flow of information when processing sequences to achieve a "long term memory", if a sequence is long enough, they still have a hard time carrying information from earlier time steps to later ones due to the vanishing gradient problem. Gradients are values used to update the weights of a neural network and thus, learn. The vanishing gradient problem occurs when the gradient shrinks as it back-propagates through time. If a gradient value becomes extremely small, it doesn't contribute much to learning and therefore the information is loss.

Transformer can then be trained faster while performing better than recurrent neural networks, which has been confirmed through many experiments. Transformer forms a significant basis for BERT (Bidirectional Encoder Representations from Transformers), which created a breakthrough in the NLP area in 2018.

2.3 BERT and ALBERT

2.3.1 Background

Language model is a probability distribution over strings of text, that is, how likely is a given string (observation) or which words are more likely to come after another word in a given "language". For example, in English, if

$$\begin{aligned}
P_1 &= P(\text{“A quick brown dog”}) \\
P_2 &= P(\text{“A brown quick dog”}) \\
P_3 &= P(\text{“Dog a brown quick”}) \\
P_4 &= P(\text{“Chien un brown quick”})
\end{aligned} \tag{2.6}$$

then it is likely that

$$P_1 > P_2 > P_3 > P_4 \tag{2.7}$$

It has been demonstrated that language model pre-training is effective for improving many natural language processing tasks, including sentence-level tasks such as natural language inference and paraphrasing as well as token-level tasks such as named entity recognition and question answering. Pre-trained language representations can then be applied to downstream tasks in two ways, feature-based and fine-tuning. The feature-based approach uses task-specific architectures that include the pre-trained representations as additional features. The fine-tuning approach, on the other hand, introduces minimal task-specific parameters and is trained on the downstream tasks by fine-tuning all pre-trained parameters. The two approaches share the same objective function during pre-training, where they use unidirectional language models to learn general language representations.

Before the introduction of BERT in 2018, existing approaches limit the power of the pre-trained representations, especially for the fine-tuning approaches. The restriction is that language models are unidirectional, limiting the choice of architectures that can be used during pre-training. For example, in OpenAI GPT, the authors use a left-to-right architecture, where every token only depends on previous tokens in the self-attention layers of the Transformer. This can be harmful when applying fine-tuning based approaches to token-level tasks such as question answering, where it is important to integrate context from both directions.

Unlike its predecessors, BERT is designed to pre-train deep bidirectional represnetations from text by jointly conditioning on both left and right context in all layers. The pre-trained BERT can then be fin-tuned with as little as one additional output layer to create state-of-the-art models for a wide range of sentence-level and token-level tasks, without significant task-specific architecture modifications.

2.3.2 Model Architecture

BERT's model is a multi-layer bidirectional Transformer encoder (based on the original implementation described in [3]. The same architecture can be used uniformly accross different task and there is minimal difference between the pre-trained architecture and the final downstream architecture.

Following the original authors' notation, we call **L** the number of Transformer blocks, the hidden size as **H**, the number of self-attention heads as **A**, and the vocabulary embedding size as **E**. Also, the feed-forward/fiter size is set at $4\mathbf{H}$. The authors implemented two models, BERT base ($\mathbf{L} = 12, \mathbf{H} = 768, \mathbf{A} = 12, \mathbf{E} = 768,$ total parameters $= 110M$), and BERT large ($\mathbf{L} = 24, \mathbf{H} = 1024, \mathbf{A} = 16, \mathbf{E} = 1024,$ total para. $= 340M$).

ALBERT reduces the model size of BERT ($18\times$ fewer parameters) and can be trained $1.7\times$ faster while not suffering from a tradeoff in performance. ALBERT is able to attain the results it does with the smaller model architecture with these parameter-reduction techniques:

Factorized Embedding Parameterization To ensure the size of the hidden layers and the embedding dimensions are different, Alberta deconstructs the embedding matrix into 2 pieces. This allows it to essentially increase the size of the hidden layer without truly modifying the actual embedding dimension. After decomposing the embedding matrix, Alberta adds a linear layer/fully connected layer onto the embedding matrices after the embedding phase is done, and this maps/ensures the dimension of the embedding dimension is the same correct. You can read more about this here.

Cross-layer parameter sharing Recall that BERT and Alberta have 12 encoder blocks each. In Alberta, these encoder blocks share all parameters. This reduces the parameter size 12-fold and also increases the regularization of the model (regularization is a technique when calibrating ML models that are used to prevent overfitting/underfitting).

Alberta removes dropout layers A dropout layer is a technique where neurons that are randomly selected are ignored during training. This means that they are no longer being trained and are essentially useless temporarily.

The following ALBERT models were implemented (the number of attention heads (**A** is set to be **H**$/64$).

- base: **L** $= 12$, **H** $= 768$, **E** $= 128$, total parameters $= 12M$
- large: **L** $= 24$, **H** $= 1024$, **E** $= 128$, total parameters $= 18M$
- xlarge: **L** $= 24$, **H** $= 2048$, **E** $= 128$, total parameters $= 60M$
- xxlarge: **L** $= 12$, **H** $= 4096$, **E** $= 128$, total parameters $= 235M$

2.3.3 Input/Output Representation

The input can be a representation of a single sentence or a pair of sentences (e.g. (Question, Answer)) in one token sentence. For a given token, its input representation is constructed by summing the corresponding token, segment, and position embeddings. A visualisation of this construction can be seen in [...].

- The WordPiece embeddings with a 30000 token vocabulary was used and '##' is used as separator. For example, *playing* is separated into *play##ing*.
- Positional embeddings are used with the maximum sentence length set at 512 tokens.
- The first token of every sequence is a special classification token (`[CLS]`). The final hidden state corresponding to this token is used as the aggregate sequence representation for classification tasks.
- Sentence pairs are packed together into a single sequence. The sentences are differentiated in two ways. First, they are separated with a special token (`[SEP]`). Second, a segment embedding is added to every token indicating whether it belongs to sentence A or sentence B.

2.3.4 Pre-training

We pre-train BERT using two unsupervised tasks, Masked Language Modelling and Next Sentence Prediction.

Masked Language Modelling (MLM) To train a deep bidirectional model, we simply mask some percentage of the input tokens at random and then predict those masked tokens. The final hidden vectors corresponding to the mask tokens are fed into and output softmax over the vocabulary. BERT authors masked 15% of all WordPiece tokens in each sequence a random. A disadvantage is that it leads to a mismatch between pre-training and fine-tuning as the `[MASK]` token does not appear during fine-tuning. To mitigate this, we do not always replace "masked" words with the actual `[MASK]`, rather, the data generator chooses 15% of the token positions randomly for prediction and do the following:

For example, in the sentence *My dog is beautiful*, the masked word is *beautiful*. The masked word is then

- replaced with the `[MASK]` token 80% of the time,
- replaced with a random token, e.g. *ugly* 10% of the time,
- unchanged 10% of the time,

Next Sentence Prediction Many important downstream tasks such as Question Answering (QA) and Natural Language Inference are based on understanding the relationship between two sentences, which is not directly captured by language modeling. In order to train a model that understands sentence relationships, we pre-train for a binarized next sentence prediction task that can be trivially generated from any monolingual corpus. Specifically, when choosing the sentences A and B for each pre-training example, 50% of the time B is the actual next sentence that follows A (labeled as `IsNext`), and 50% of the time it is a random sentence from the corpus (labeled as `NotNext`).

For example,

- Input: `[CLS]` a man work##s as a `[MASK]` in a shop `[SEP]` he is `[MASK]` `[SEP]`
  Output: `IsNext`
- Input: `[CLS]` a man work##s as a `[MASK]` in a shop `[SEP]` she like##s to eat hamburger and `[MASK]` `[SEP]`
  Output: `NotNext`

The `NotNext` sentences are then randomly chosen. The final model achieves an accuracy of around 98% in this task.

2.3.5 Fine-tuning for Question Answering

BERT can be fine-tuned quite straightforwardly since the self-attention mechanism in the Transformer allows BERT to model many downstream tasks, whether they involve single text or text pairs—by swapping out the appropriate inputs and outputs.

For applications involving text pairs, BERT uses the self-attention mechanism to unify these two stages, as encoding a concatenated text pair with self-attention effectively includes bidirectional cross attention between two sentences. For each task, we simply plug in the task-specific inputs and outputs into BERT and fine-tune all the parameters end-to-end. At the input, sentence A and sentence B from pre-training are analogous to question-passage pairs in question answering. At the output, the token representations are fed into an output layer for token-level tasks, such as sequence tagging or question answering.

In the Question Answering task, the input question and passage are represented as a single packed sequence, with the question using the A embedding and the passage using the B embedding. We only introduce a start vector $S \in \mathbb{R}^H$ and an end vector $E \in \mathbb{R}^H$ during fine-tuning. For each word i, let $T_i \in \mathbb{R}^H$ be its final hidden vector. Then the probability of that word being the start of the answer span is computed as a dot product between T_i and S followed by a softmax over all of the words in the paragraph:

$$P_i = \frac{e^{S \cdot T_i}}{\sum_j e^{S \cdot T_j}} \tag{2.8}$$

The analogous formula is used for the end of the answer span. The score of a candidate span from position i to position j is defined as $S \cdot T_i + E \cdot T_j$, and the maximum scoring span where $j \geq i$ is used as a prediction. The training objective is the sum of the log-likelihoods of the correct start and end positions.

In case of tasks involving questions having no answers, these questions are treated as having an answer span starting and ending at token 0, i.e. `[CLS]`. The probability space for the start and end answer span positions is extended to include the position of the `[CLS]` token. Let $C \in \mathbb{R}^H$ be the final vector of the token `[CLS]`. Then for prediction, the score of the no-answer span is computed $s_{\text{null}} = S \cdot C + E \cdot C$. The score of the best non-null span is computed $s\hat{i}, j = \max_{j \geq i} S \cdot T_i + E \cdot T_j$. A non-null answer is predicted when $s_{\hat{i},j} > s_{\text{null}} + \tau$, where the threshold τ is selected on the dev set to maximize F1 or other wanted metrics.

2.4 Designing Modules on Top of ALBERT

Language modelling has been shown to be effective in solving many natural language processing tasks. BERT and ALBERT have proven to be a conceptually simple but empirically powerful one. In this project, we decide to use ALBERT due to its leaner architecture and no performance compromises. According to paper [4], the pre-trained BERT representations can be fine-tuned to perform better in specific tasks. We may then try to improve the performance of ALBERT through additional architectures on top of ALBERT. Therefore, here we are going to build our own task-specific layers on top of pre-trained ALBERT.

In this project, we simply used ALBERT model with a linear output layer ALBERT-SQuAD-out as the baseline model. As for the task-specific layers, we have a subsequent encoder-decoder architecture as post-processing to improve ALBERT model's performance specifically on SQuAD 2.0. For the main encoder-decoder architecture, RNN-based bi-directional long short-term memory layer (BiLSTM) and gated recurrent units (GRU) were adopted as the encoder and decoder, which are commonly used in sequence to sequence translation task. As for the multi-layer state transitions, a highway network was used to adaptively copy or transform representations. More attention layers have also been added in transitions; an detailed explanation on how the attention layers turned out is offered in Chapter 4. And for the final output layer, we compared the original ALBERT-SQuAD-out and BiDAF-out. The details are explained in the Approach section below.

Although in this project, we strived to apply a variety of layers on top of the ALBERT model, it turns out that most of the layers do not outperform the model performance of ALBERT-base-v2 + ALBERT-SQuAD-out. Previous works exploring adding various layers over ALBERT or BERT have experienced similar setbacks. For example, a work explored adding Highway layer, BiDAF, CNN, transformers, etc. over BERT, but the only model that ends up outperforming BERT + Linear is BERT + Highway. Most of the model fails to improve the performance of the baseline model because a lot of these layers force the model to learn non-existing correlations and thus add noises to the final predictions. It is then within expectation that additional layers don't necessarily improve the ALBERT-base-v2 + ALBERT-SQuAD-out performance. More explanation to why adding more layers doesn't necessarily improve model performance is included in the Experiments and Analysis sections below.

3. Approach

3.1 Baseline Model

Baseline ALBERT-base-v2 + Linear-out uses the fine-tuned ALBERT (albert-base-v2) with a Linear-out layer on top.

Oracle Human performance is commonly used to assess models performance. The oracle used is the human performance given in the SQuAD dataset: $EM = 86.83, F1 = 89.45$.

3.2 Modules on Top of ALBERT

In this section, we discuss modules we implemented as the output architecture. We gained these insights from existing networks and implemented them in our own way on top of the pre-trained ALBERT-base model.

3.2.1 Embedding Layers

The embedding that we use comes directly from the ALBERT models, and is labelled with the ALBERT model name in the following literature (e.g. ALBERT-base if the embedding is generated using ALBERT-base).

3.2.2 Encoder and Decoder Blocks

We explored several encoders and decoders discussed in other NLP works and here is a summary.

Bidirectional Long Short-Term Memory (LSTM) Encoder/Decoder Long Short-Term Memory (LSTM) is an RNN architecture aimed to solve vanishing gradients problem. On each time step t, we have a hidden state $h^{(t)}$ and a cell state $c^{(t)}$. The cell can store long-term information, and the LSTM can erase, write and read information from the cell controlled by three gates (forget gate $f^{(t)}$, input gate $i^{(t)}$ and output gate $o^{(t)}$). On each time step, we can update our hidden state and cell state as following:

$$
\begin{aligned}
\widetilde{\mathbf{c}}^{(\mathbf{t})} &= \tanh\left(\mathbf{W}_{\mathbf{c}}\mathbf{h}^{\mathbf{t}-\mathbf{1}} + \mathbf{U}_{\mathbf{c}}\mathbf{x}^{\mathbf{t}} + \mathbf{b}_{\mathbf{c}}\right) \\
\mathbf{c}^{(\mathbf{t})} &= \mathbf{f}^{(\mathbf{t})} \circ \mathbf{c}^{(\mathbf{t}-\mathbf{1})} + \mathbf{i}^{(\mathbf{t})} \circ \widetilde{\mathbf{c}}^{(\mathbf{t})} \\
\mathbf{h}^{(\mathbf{t})} &= \mathbf{o}^{(\mathbf{t})} \circ \tanh \mathbf{c}^{(\mathbf{t})}
\end{aligned}
\tag{3.1}
$$

By adding an RNN LSTM encode/decoder on top of the BERT model, we can integrate temporal dependencies between time-steps of the output tokenized sequence better.

Gated Recurrent Units (GRU) Encoder/Decoder Gated Recurrent Units [4] is a simpler alternative to the LSTM. And on each time step t, we have input $x^{(t)}$ and hidden state $h^{(t)}$ (no cell state), and we used two gates (update gate $u^{(t)}$ and reset gate $r^{(t)}$) to control the states.

3.2.3 Self-Attention

For some NLP tasks, people propose to use attention mechanism on top of the CNN or LSTM model to introduce an extra source of information to guide the extraction of sentence embedding. Here we also implemented a self-attention layer discussed, and it allows each position of the output token to attend to all positions up to and including that position. In our implementation, we used the simplest basic dot-product attention. This can help for better interpreting the inference between different positions in the output sequence.

3.2.4 Highway Network

Highway network is a novel architecture that enables the optimization of networks with virtually arbitrary depth. By applying a gating mechanism, a neural network can have paths along which information can flow across several layers without attenuation.

Given an input $x \in \mathbb{R}^H$, a one-layer highway network computes:

$$
\begin{aligned}
\mathbf{x}_{\text{gate}} &= \sigma\left(\mathbf{W}_{\text{gate}}\mathbf{x} + \mathbf{b}_{\text{gate}}\right) \in \mathbb{R}^H \\
\mathbf{x}_{\text{proj}} &= \operatorname{Re} LU\left(\mathbf{W}_{\text{proj}}\mathbf{x} + \mathbf{b}_{\text{proj}}\right) \in \mathbb{R}^H \\
\mathbf{x}_{\text{highway}} &= \mathbf{x}_{\text{gate}} \odot \mathbf{x}_{\text{proj}} + \left(1 - \mathbf{x}_{\text{gate}}\right)
\end{aligned}
\tag{3.2}
$$

$\mathbf{W}_{\text{proj}}$, $\mathbf{W}_{\text{gate}}$, $\mathbf{b}_{\text{gate}}$ and $\mathbf{b}_{\text{proj}}$ are learnable parameters. We choose a highway network to train multi-layer state transitions in our recurrent neural networks. Instead of traditional neural layers, it can allow the network to adaptively copy or transform representations. So it can help to refine the tokenized sequence.

3.2.5 Output Layers

For the output layer, we tried the original ALBERT linear output layer and the QA output layer from Bi-Directional Attention Flow (BiDAF-Out).

ALBERT Output Layer (Linear-out) A simple linear output layer that converts the dimension of the output sequence from (batch_size, seq_len, hiddenstate) to (batch_size, seq_len, 2). And we split it to get the start and end logits. Finally, we compute the cross-entropy loss with the start and end position vectors.

QA output layer from Bi-Directional Attention Flow (BiDAF-out) To replace the linear output layer in default BERT model, we add a QA output layer adapted from BiDAF model. First we apply a LSTM to the ALBERT tokenized output sequence $o_1, o_2, \ldots, o_N \in \mathbb{R}^H$ and get a model output sequence $(m_1, m_2, \ldots, m_N \in \mathbb{R}^{2H})$.

Then we apply a bidirectional LSTM to the model output sequence, producing a vector m_i' for each m_i :

$$\begin{gathered}\mathbf{m}'_{\mathbf{i},\mathbf{fwd}} = \text{LSTM}\left(\mathbf{m}'_{\mathbf{i-1}}, \mathbf{m}_\mathbf{i}\right) \in \mathbb{R}^H, \mathbf{m}'_{\mathbf{i},\mathbf{rev}} = \text{LSTM}\left(\mathbf{m}'_{\mathbf{i+1}}, \mathbf{m}_\mathbf{i}\right) \in \mathbb{R}^H \\ \mathbf{m}'_\mathbf{i} = \left[\mathbf{m}'_{\mathbf{i},\mathbf{fwd}}; \mathbf{m}'_{\mathbf{i},\mathbf{rev}}\right] \in \mathbb{R}^{2H}\end{gathered} \tag{3.3}$$

Now, let $O \in \mathbb{R}^{H\times N}$ be the matrix with columns $o_1, o_2, \ldots, o_N \in \mathbb{R}^H$. And let $M, M' \in \mathbb{R}^{2H\times N}$ be matrices with columns $m_1, m_2, \ldots, m_N$ and $m_1', m_2', \ldots, m_N'$. Then we calculate the start logits and end logits as following:

$$\mathbf{S}_{\text{logits}} = \mathbf{W}_{\text{start}} [O; M], \mathbf{E}_{\text{logits}} = \mathbf{W}_{\text{end}} [O; M']$$

Finally we compute the cross entropy loss with the start and end position vectors.

3.3 Proposed Models

The idea is to add an encoder-decoder architecture on top of the PyTorch implementation of ALBERT baseline. This idea may come from the computer vision area. For multi-view synthesis task, we always use a general auto-encoder to generate the sketch of other views and an additional auto-encoder for texture level reconstruction.

Here, we propose a branch of models based on combinations of different modules. The architecture of our models can be found in Figure 3.1.

We tried different combinations among these modules to get better performance than BERT baseline model. The details of representative models are presented below.

ALBERT-base-v2 + Highway + Linear-out An extra highway layer is added before outputting.

ALBERT-base-v2 + GRU Encoder + Highway + GRU Decoder + Linear-out Adds the encoder-decoder architectures which are bridged using a highway network.

ALBERT-base-v2 + LSTM Encoder + Highway + LSTM Decoder + Linear-out Similar to the previous one, but the GRU encoder-decoder is replaced with LSTM.

ALBERT-base-v2 + LMTM Encoder + Attention + LSTM Decoder + BiDAF-out This model uses LSTM encoder-decoder and attention.

ALBERT-base-v2 + GRU Encoder + Attention + Self-Att + GRU Decoder + BiDAF-out This model features GRU encoder-decoder, attention, self-attention and BIDAF output.

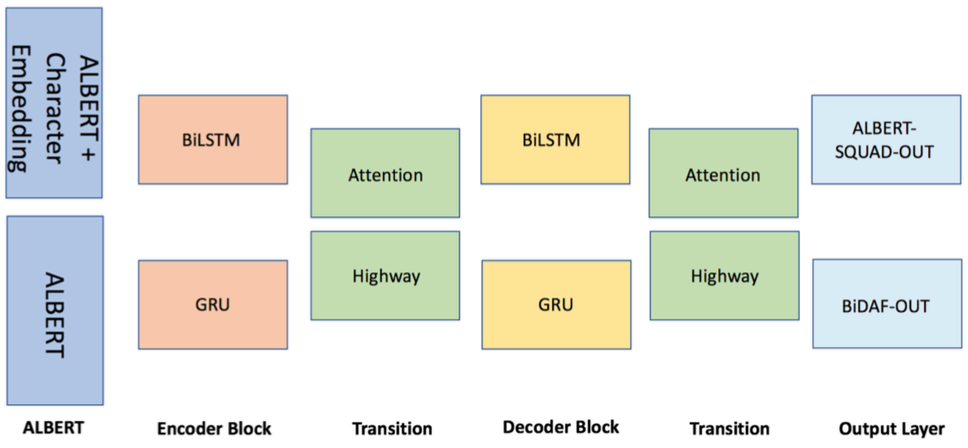


Figure 3.1: Schema of Model Architecture

3.4 Already Implemented Models

We used two implemented models found in the HuggingFace library:

ALBERT-base-v2 + Linear-out We implemented four models having the same architecture but with different initialisations (seed = 128, 256, 384, 512) (baseline model's seed is 128.)

ALBERT-large-v2 + Linear-out Seed is set at 128.

3.5 Implementation Details

Library We mainly utilise the HuggingFace library throughout this project for its excellent support for language models such as ALBERT and close integration with PyTorch.

Models that are already implemented in HuggingFace (ALBERT-base-v2 and ALBERT-large-v2) are accessible using

```
ALBERTForQuestionAnswering.from_pretrained('model_name')
```

where `'model_name'` may be either `'albert-base-v2'` or `'albert-large-v2'`.

Otherwise, the customised models and their extra custom layers can found in `models.py` and `layers.py`, respectively.

For training and testing, the file `run_squad.py` is used.

Debugging and Training We first run `run_squad.py` on a small sampling of training set and testing set to see if there are problems with the models, then we train the models over the whole data.

The ALBERT-base-v2-based models typically take 3.5 hours for 3 epochs and the ALBERT-large-v2 model takes around 7 hours for 2 epochs on a P100 GPU given in Kaggle environments. Due to the lack of resources, we decide not to explore the bigger models (ALBERT-xlarge-v2 and ALBERT-xxlarge-v2).

4. Experiments

4.1 Data

We used Stanford Question Answering Dataset (SQuAD 2.0) to train and evaluate our models. Samples in this dataset include (question, answer, context paragraph) tuples. The paragraphs are from Wikipedia. The questions and answers were crowdsourced using Amazon Mechanical Turk. And we have around 150k questions in total, and roughly half of the questions are not answerable (this is new in SQuAD 2.0). However, if a question is answerable, the answer is guaranteed to be a continuous span in the context paragraph.

4.2 Evaluation method

We apply two metrics to measure the performance of our model: Exact Match (EM) score and $F1$ score.

Exact Match A binary measure (true/false) of whether the system output exactly matches the ground truth answer exactly. This is a fairly strict metric.

Ground Truth	Prediction	EM
Albert Einstein	Einstein	0
Albert Einstein	Albert Einstein	1

Table 4.1: EM example

F1 A less strict metric, and it is the harmonic mean of precision and recall.

$$F1 = \frac{2 \times \text{ precision } \times \text{ recall}}{\text{precision+recall}} \tag{4.1}$$

For example, if the ground truth is Albert Einstein, and the model predicts Einstein, then the precision is 100% and the recall is 50%. So $F1 = 66.67\%$.

The system would have 100% precision if its answer is a subset of the ground truth answer and 50% recall if it only includes one out of the two words in the ground truth output. When a question has no answer, both the $F1$ and EM score are 1 if the model predicts no-answer, and 0 otherwise.

4.3 Experimental details

4.3.1 Models involving ALBERT-base-v2

- Learning rate: 3×10^{-5}
- Epochs: 3
- Max sequence length: 384
- Document stride: 128
- Dropout rate: 0.2
- Batch size: 8.

4.3.2 Models involving ALBERT-large-v2

- Learning rate: 2×10^{-5}
- Epochs: 2
- Max sequence length: 384
- Document stride: 128
- Dropout rate: 0.2
- Batch size: 8.

5. Analysis

5.1 Quantitative Analysis

No	Model	EM	F1
1	ALBERT-base-v2 + Linear-out (Seed 128)	77.90	81.29
2	ALBERT-base-v2 + Linear-out (Seed 256)	77.25	80.61
3	ALBERT-base-v2 + Linear-out (Seed 384)	76.47	79.87
4	ALBERT-base-v2 + Linear-out (Seed 512)	75.99	79.48
5	ALBERT-base-v2 + Highway + Linear-out	77.79	81.25
6	ALBERT-base-v2 + GRU Encoder + Highway + GRU Decoder + Linear-out	50.07	50.07
7	ALBERT-base-v2 + LSTM Encoder + Highway + LSTM Decoder + Linear-out	**78.11**	**81.45**
8	ALBERT-base-v2 + LSTM Encoder + Attention + LSTM Decoder + BiDAF-out	77.13	80.58
9	ALBERT-base-v2 + GRU Encoder + Att + Self-Att + GRU Decoder + BiDAF-out	74.50	77.77
10	ALBERT-large-v2 + Linear-out	0.53	4.06

Table 5.1: Single model performance on the dev set

From the results, we can see that adding more layers on top of ALBERT-base-v2 model (with char embedding) did not bring about significant improvement to the performance of the model: the only model that offered minor improvement according to F1 score was the ALBERT-base-v2 + LSTM Encoder + Highway + LSTM Decoder + Linear-out model. This lack of improvement in performance is contrary to the expectation that adding more layers such as attention should improve the model performance.

Very noticeable are the anomalies found in the results for the models 6 and 10. We have repeated the experiments for these models but the results stay largely the same, about which we cannot give a clear explanation at this time. Also, the same ALBERT-base-v2 model gives different results with different initialisations. The differences are in our view quite noticeable ($EM = 1.91$, $F1 = 1.81$)

5.1.1 Comparing Models

Adding more complicated layers on top of ALBERT also doesn't make the model performance significantly better than the baseline ALBERT- base-v2 + ALBERT-SQuAD-out. An explanation is that since our ALBERT models take the context and question together as input and the transformer in ALBERT has multi-head attention, the token embeddings generated by ALBERT itself already incorporate better attention between context and question.

Recurrent units LSTM may help improve performance on top of BERT as shown in the model 7. RNN unit may help to integrate temporal dependencies between time-steps of the output tokenized sequence better, thus refine the output sequence for following operations.

BiDAF output layer It cannot outperform the ALBERT output layer. Even adding an additional encoder, the output layer of BiDAF still cannot beat the baseline (Model 8, 9). This may because, for BiDAF model, we actually pass the query-to-context and context-to-query attention to the output layer (modeling layer + an additional output layer). But in our ALBERT model, although the tokenized sequence we pass to the output includes both the query and the context information, we do not necessarily interfere with their relations further before passing to the output.

Highway network It may be effective when added between the encoder and decoder (Model 7), it is promising to serve as a multi-layer state transition.

5.2 Behavioural Analysis

5.2.1 Selected Examples

- Question: What is the original meaning of the word Norman?
- Context: The English name "Normans" comes from the French words Normans/Normanz, plural of Normant, modern French normand, which is itself borrowed from Old Low Franconian Nortmann "Northman" or directly from Old Norse Norðmaðr, Latinized variously as Nortmannus, Normannus, or Nordmannus (recorded in Medieval Latin, 9th century) to mean "Norseman, Viking".
- Human Answers: Norseman, Viking.
- Model Answer: plural of Normant.
- The error: The model provides a wrong answer.
- Potential causes: The models can understand that the question is asking for the meaning of the word, but it does not quite capture that the meaning has to be original.

- Question: What is the seldom used force unit equal to one thousand newtons?
- Context: The pound-force has a metric counterpart, less commonly used than the newton: the kilogram-force (kgf) (sometimes kilopond), is the force exerted by standard gravity on one kilogram of mass. The kilogram-force leads to an alternate, but rarely used unit of mass: the metric slug (sometimes mug or hyl) is that mass that accelerates at 1 m·s–2 when subjected to a force of 1 kgf. The kilogram-force is not a part of the modern SI system, and is generally deprecated; however it still sees use for some pur-

poses as expressing aircraft weight, jet thrust, bicycle spoke tension, torque wrench settings and engine output torque. Other arcane units of force include the sthène, which is equivalent to 1000 N, and the kip, which is equivalent to 1000 lbf.

- Human Answers: sthène.
- Model Answer:
- The error: The model fails to predict an answer when there is one.
- Potential causes: The fact that 'newton' is not directly within the text may hinder the performance of the model, despite the theoretical ability of language model to connect the two concepts '1000 N' with '1000 Newton'.
- Question: When was the first measurement of the value of the Newton Universal Gravitation Constant?
- Context: In this equation, a dimensional constant is used to describe the relative strength of gravity. This constant has come to be known as Newton's Universal Gravitation Constant, though its value was unknown in Newton's lifetime. Not until 1798 was Henry Cavendish able to make the first measurement of using a torsion balance; this was widely reported in the press as a measurement of the mass of the Earth since knowing could allow one to solve for the Earth's mass given the above equation. Newton, however, realized that since all celestial bodies followed the same laws of motion, his law of gravity had to be universal. Succinctly stated, Newton's Law of Gravitation states that the force on a spherical object of mass due to the gravitational pull of mass is
- Human Answers: sthène.
- Model Answer:
- The error: The model fails to predict an answer when there is one.
- Potential causes: The fact that 'newton' is not directly within the text may hinder the performance of the model, despite the theoretical ability of language model to connect the two concepts '1000 N' with '1000 Newton'.
- Question: What car is licensed by the FSO Car Factory and built in Egypt?
- Context: The FSO Car Factory was established in 1951. A number of vehicles have been assembled there over the decades, including the Warszawa, Syrena, Fiat 125p (under license from Fiat, later renamed FSO 125p when the license expired) and the Polonez. The last two models listed were also sent abroad and assembled in a number of other countries, including Egypt and Colombia.
- Human Answers: Polonez/125p
- Model Answer:

- The error: The model fails to predict an answer when there is one.
- Potential causes: In the context paragraph, the keywords in the question (FSOO, Egypt license) are far away from each other. Thus, the likelihood of the answer is more sparsely distributed among the words in the three sentence, making the probability of "no answer" even higher than the actual answers.

5.2.2 Yes- and No-Answer

No	Yes-EM	Yes-F1	No-EM	No-F1
1	73.77	80.56	82.02	82.02
2	74.38	81.10	80.12	80.12
3	73.04	79.84	79.90	79.90
4	78.66	85.65	73.34	73.34
5	72.66	79.57	**82.93**	**82.93**
6	—	—	—	—
7	**74.41**	**81.08**	81.82	81.82
8	73.48	80.40	80.76	80.76
9	69.47	76.01	79.51	79.51
10	—	—	—	—

Table 5.2: Single model performance on types of questions

In this section, we analyze the F1 and EM score on both questions that have answers and questions that have no answer. As shown in Table 5.2, for each model, we present their Has Answer F1, EM, and No Answer F1, EM.

Let we note that the models 7, 8, 9 were initialised with the same seed as the first one, so we will make comparisons on these models. It appears that while model 7 gives the best EM and F1 overall and on the Yes-Answers, model 5 is better than it in the case of No-Answers.

Generally, models that cannot beat BERT baselines have poor performance on questions that have no answer (model 8 and model 9). These models also have the most complex architecture, suggesting that excessive information hinders the ability to detect the No-Answer questions. For the models that cannot beat the baselines, they do not increase the performance on questions with answers while sacrificing No-Answer question's performance.

The case of model 5, ALBERT-base-v2 + Highway + Linear-out, suggests a slight improvement in robustness that the highway layer gives for the model in dealing with No-Answer questions. However, the metrics between Yes-Answer and No-Answer questions are more balanced in the baseline model than model 5.

5.2.3 Question Types

The models generally give short questions to 5W questions.

For exampke, the answers for when-questions usally do not have the leading preposition 'in' although the human answers frequently do.

- Question: When did Edward return?
- Context: When finally Edward the Confessor returned from his father's refuge in 1041, at the invitation of his half-brother Harthacnut, he brought with him a Norman-educated mind. He also brought many Norman counsellors and fighters, some of whom established an English cavalry force. This concept never really took root, but it is a typical example of the attitudes of Edward. He appointed Robert of Jumièges archbishop of Canterbury and made Ralph the Timid earl of Hereford. He invited his brother-in-law Eustace II, Count of Boulogne to his court in 1051, an event which resulted in the greatest of early conflicts between Saxon and Norman and ultimately resulted in the exile of Earl Godwin of Wessex.
- Human Answers: in 1041
- Model Answer: 1041
- The Analysis: The model captures the time itself but not the preposition before it.

The same cases can be found in where-questions.

- Question: Where did Harold II die?
- Context: In 1066, Duke William II of Normandy conquered England killing King Harold II at the Battle of Hastings. The invading Normans and their descendants replaced the Anglo-Saxons as the ruling class of England. The nobility of England were part of a single Normans culture and many had lands on both sides of the channel. Early Norman kings of England, as Dukes of Normandy, owed homage to the King of France for their land on the continent. They considered England to be their most important holding (it brought with it the title of King—an important status symbol).
- Human Answers: Battle of Hastings, the Battle of Hastings, at the Battle of Hastings
- Model Answer: Battle of Hastings
- The Analysis: The model captures the place itself but not the preposition before it.

This suggests a strong robustness in capturing what is referred to (entities, times, etc.) in the question. The models, here, language models, are then implied to be able to perform well in problems such as entity tagging yet require few changes in architecture. This is one of the advantages of Transformer-based language models.

6. Conclusion

The main goal of the project is to search for models that make predictions about answers as accurately as possible. In this paper, we designed several task-specific architectures on top of the ALBERT model according to insights gained from other networks. We have implemented 5 combinations of layers (encoder and decoder layer, self attention layer, highway network, and output layer) on top of ALBERT-base-v2; beyond that, we have also implemented ALBERT-large + ALBERT-SQuAD-out.

We can further improve the performance of our model by fine-tuning the parameters in each layer. A limitation is that we had to truncate the sequences because we could not fit them into the GPU otherwise. Based on the error analysis, having longer sequences could have helped improve the performance of our models; with longer sequences, the sentences that used to be split into two sequences could be fit into one without losing the context, and our models can generate more accurate answers that way. We can also consider adding some extra layers inside of ALBERT to further improve its performance.

Appendix

Bibliography

[1] C. Zeng, S. Li, Q. Li, J. Hu, and J. Hu, "A survey on machine reading comprehension: Tasks, evaluation metrics and benchmark datasets," 2020. [Online]. Available: https://arxiv.org/abs/2006.11880

[2] P. Rajpurkar, R. Jia, and P. Liang, "Know what you don't know: Unanswerable questions for squad," 2018. [Online]. Available: https://arxiv.org/abs/1806.03822

[3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017. [Online]. Available: https://arxiv.org/abs/1706.03762

[4] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2018. [Online]. Available: https://arxiv.org/abs/1810.04805

List of Figures

List of Tables