Karlsruhe Institute of Technology

Towards End-To-End Information Retrieval: Enabling Question Answering Systems To Answer Open-Domain Questions

Master's Thesis of

Felix Schneider

at the Interactive Systems Lab
Institute for Anthropomatics and Robotics
Karlsruhe Institute of Technology (KIT)

Reviewer:	Prof. Dr. Alex Waibel
Second reviewer:	Prof. Dr. Tamim Asfour
Advisor:	Dr. Jan Niehues

25. April 2018 – 25. October 2018

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text.

Karlsruhe, October 24, 2018

. . . .

(Felix Schneider)

Dr. Crusher *Here's a question you shouldn't be able to answer... Computer: What is the nature of the universe?*
Computer *The universe is a spheroid region 705 meters in diameter.*
Star Trek: The Next Generation—"Remember Me"

Abstract

In a traditional question answering (QA) task, a problem instance consists of a question and a paragraph (called the context) in natural language. A system must read and understand the context to find an answer to the question. Here it is implicitly understood that the provided context contains all the information necessary to answer the question. This task falls within the field of natural language processing (NLP) and is closely related to reading comprehension. Being able to understand and answer questions in natural language opens up myriad possibilities in human-computer interaction.

Providing the system with a relevant piece of context implies that some amount of information about the query is already known in advance. In a real problem setting, the relevant context is often not known, as questions can concern a large variety of topics. When the context is not known, the problem is said to be one of *open-domain* question answering. There are already published models that achieve good performance on the traditional question answering task. This work focuses on methods to find an appropriate context paragraph from a large pool of candidates covering many different topics.

This problem—finding one relevant piece of information in a corpus—is known as information retrieval (IR). It has been widely studied, mostly in the context of search engines. Existing systems typically work by constructing a search index and/or comparing similar words in the question and context candidates. Neural networks may be able to gain and use a deeper understanding of the meaning of the texts in order to make a more informed decision. However, it is not computationally feasible to apply neural network classifiers to a large corpus of possible contexts. Therefore, we explore several alternative models that make do without applying a classifier to every possible question-context pair.

Zusammenfassung

Bei einer traditionellen Question Answering (QA) Aufgabe besteht eine Probleminstanz aus einer Frage und einem Absatz Text in natürlicher Sprache (dem Kontext). Um das Problem zu lösen, muss ein System Frage und Kontext lesen, verstehen und unter Zuhilfenahme der Informationen im Kontext eine Antwort formulieren. Es versteht sich dabei implizit, dass der gegebene Kontext alle Information beinhaltet, die nötig sind, um die Frage zu beantworten. Diese Aufgabenstellung fällt in das Feld des Natural Language Processing (NLP) und ist eng verwandt mit Leseverständnis (Reading Comprehension). Die Fähigkeit, Fragen in natürlicher Sprache zu verstehen und zu beantworten eröffnet zahlreiche Möglichkeiten in der Mensch-Maschine-Interaktion.

Um das System mit relevantem Kontext zu versorgen, muss dieser schon im Vorfeld bekannt sein. In einer realistischen Umgebung ist das oft nicht der Fall, weil mögliche Fragen eine Vielzahl an unterschiedlichen Themen betreffen können. In diesem Fall spricht man von einem Question Answering Problem mit *offener Domäne*. Es gibt bereits publizierte Modelle, die gute Ergebnisse für die traditionelle Question Answering Problemstellung liefern. Diese Arbeit zielt darauf ab, aus einem Korpus von Text über viele unterschiedliche Themen einen relevanten Absatz als Kontext für eine gegebene Frage zu finden.

Dieses Problem (das Finden relevanter Information in einem Korpus) ist bekannt als Information Retrieval (IR). Es wurde bereits ausgiebig untersucht, vor allem im Kontext von Suchmaschinen. Bestehende Systeme funktionieren üblicherweise so, dass sie einen Index erzeugen und/oder ähnliche Wörter in der Suchanfrage und den Kandidaten vergleichen. Neuronale Netze könnten ein tieferes Verständnis der Bedeutung der Texte erarbeiten und verwenden, um eine besser informierte Entscheidung zu treffen. Allerdings ist es nicht zeitlich realistisch, einen neuronalen Klassifikator auf jedes Dokument eines großen Korpus anzuwenden. Aus diesem Grund untersuchen wir mehrere unterschiedliche Modelle, die ohne die Anwendung eines Klassifikators auf jedes mögliche Frage-Kontext-Paar auskommen.

Contents

1 Introduction

Computers have allowed us to answer many questions we couldn't answer or even conceive of without them. They do so for the most part at much higher speed and accuracy than humans. When we want to automate a task, a necessary first step is always to bring the relevant data into a format suitable for the computer. If the computer is unable to process some piece of information because it is not in a format it can understand, it is of no use to any automated process. Therefore, the scope of possible tasks that can be automated is limited by what data a computer can process.

When making data available to a computer, we typically transform it into a structured, well-defined form. When no format exists yet to represent the type of data we need, it is up to a designer to create one. When doing so, the designer must necessarily make assumptions about the data and impose restrictions on it. Not all data is equally well suited to such conversion and creating the format and converting the data takes time and effort. No one format can describe all data. However, humans can use language to communicate almost any concept to one another.

The language humans use to communicate is referred to as *natural language*, as opposed to a *formal language*, such as a programming language, or mathematical formulas. Formal languages are meant to communicate information from within a narrow domain unambiguously and precisely. They follow strict rules and are relatively easy for a computer to comprehend (in fact, many formal languages are constructed for this very purpose). The effort to make computers understand and use information expressed in natural language is an open field of interdisciplinary research known as *natural language processing (NLP).*

At what point can a computer be said to have understood natural language? In a complex system, a judgement based on observation of the sysem's internal state may not be possible. We cannot pause the system, look at the memory it uses, point to a part of it and say: "That part is the system's representation of the concept of, say, *a dog*." The system may use an internal representation of a concept that is incomprehensible to us[1].

In neural networks, the part of the network that transforms an input into the system's internal representation is referred to as the *encoder*, whereas the part that transforms it back into a form we can understand (which can, but does not have to be natural language) is referred to as the *decoder*. While we cannot judge whether the representation produced by the encoder fully encapsulates the meaning of a text, we can judge the output of the decoder. Applied to the question answering task, this

[1] It is likely that if we had the same level of insight into the internal workings of a human brain as we have into a running computer program, we would find the representations of concepts therein equally incomprehensible (for more information, see e. g. Patterson, Nestor, and Rogers [26]).

means that we cannot know for certain whether a system has understood the context paragraph, but we can judge whether or not it correctly answered the question.

This is not unlike education: We cannot know for certain whether a student has understood a lesson, but we can judge whether they answer questions correctly. This highlights the importance of good questions: If we choose poor questions, it may be possible to answer them without actually understanding the text. In section 3.2, we will discuss several published datasets containing different arrangements of questions and contexts.

It could be noted that any problem any human has ever formulated can be posed in the form of a question, resulting in a problem space that is impossibly large, including vastly different examples such as “What is the square root of 4 315?”, “Does this car offer good value for money?”, or “Is it right to eat animals?”. While fiction has many examples of computers that could answer all these questions, real research into question answering must be concerened with the *domain* of inquiry. Question answering problems where questions are expected to remain within a particular topic are called *closed-domain*, whereas problems where questions can have any structure and ask about any topic are called *open-domain* question answering problems.

By that definition, most datasets discussed in section 3.2 pose open-domain question answering problems. However, it could be argued that the topic of each question is in fact constrained significantly: A question must be answerable using only the provided context (remember that in the “traditional” question answering task, a problem instance consists of the question and one paragraph of context). For this work, we raise the bar on what constitutes an open-domain question answering problem by requiring that no context is given, a problem instance consists only of a question and the system has access to a global corpus independent of the questions. We refer to the previous definition as a traditional question answering problem. It still does not fit the definition for closed-domain, as the provided context and therefore the questions can still be from any topic.

1.1 Goals

In many realistic applications, a question will be posed to a question answering system without an appropriate context being known. If the question is posed by a human, they would not even need to ask the question to the system if they knew the context that contains the answer—they would be able to answer it themselves. Therefore, the system should be provided in advance with a large amount of context documents (such as Wikipedia articles) and be capable of finding a context paragraph that is relevant to the query.

There are some proposed approaches to this problem that use a classifier to assign a score to every paragraph for each question. This solution scales poorly with the size of the knowledge corpus and becomes infeasible even when using only a fraction of all Wikipedia articles (which don’t represent a reliable knowledge source by themselves). A solution must be able to answer a question fast enough to be useful.

The contribution of this thesis concerns only the information retrieval aspect of the system—finding the most relevant paragraph from the context corpus. Using that, an existing question answering model is able to answer open-domain questions without a context. The final step of this work will be to create the full pipeline to chain these systems together into a fully featured open-domain question answering system. To summarize:

1. The system must, given only a question in natural language, find a relevant context paragraph from the corpus.
2. With this paragraph and the question, engage a question answering system to answer the question.
3. The system's response time must be fast enough to be useful (ideally less than 1 second per question).
4. The system must not perform many calculations on every possible question-paragraph pair.
5. The system should scale favorably to larger corpus sizes.
6. The system may peform pre-calculations on every paragraph from the corpus, so long as this only needs to be done once.

As the question answering system is not developed as part of this work, the main performance metric of the developed model will be the percentage of questions that *could* be answered using the context paragraph that the model predicts, regardless of whether or not the question answering system actually finds the answer.

1.2 Outline

Chapter 2 gives an introduction into the basic models used in this thesis, in particular neural networks and TF-IDF. In chapter 3, we discuss already published work on this topic, as well as give an overview over the available datasets. In chapter 4 and 5, we describe our data preprocessing and the models used in our experiments, respectively. We show the results of the experiments in chapter 6 and draw our conclusions in chapter 7.

2 Theory

This chapter provides a brief introduction into the underlying methods used in the following experiments. This includes both a primer on artificial neural networks as well as on non-neural information retrieval using the term-weighting scheme TF-IDF (term frequency inverse document frequency).

2.1 Neural Networks

The initial inspiration for neural networks, more precisely called *artificial* neural networks, comes from the function of biological neurons: A neuron has a number of "inputs"—in the biological case these are synapses coming from other neurons connected to it by dendrites—that stimulate the neuron with an electrical potential. The level of activation of the neuron is a non-linear function of the sum of its inputs. This level of activation could be considered the neuron's "output". In biological neurons, the activation takes place as a discrete event called a spike, when the neuron discharges some of its electric potential to other connected cells. However, modelling biological neurons in this way, which involves simulating differential equations over time, is too computationally expensive to be useful for most tasks. It is nevertheless explored as, among other uses, a form of extremely distributed computing. These types of models are called *spiking neural networks* [11].

More commonly, neural networks are abstracted to a computationally simpler form. In this model, a neuron has some number of inputs x, each with a weight w (also called the neuron's parameters) and a non-linear activation function Φ. The output of the neuron is calculated as

$$y = \Phi(w^\top x) \tag{2.1}$$

If we wish to apply a biological interpretation, calculating a scalar output for the neuron could be seen as calculating the rate of activation, rather than calculating the exact time of each activation event, although most models using neural networks are not concerned with biological analogy and no attempt is made to find a biological equivalent to more advanced techniques introduced below, such as gradient descent and the different activation functions.

Initially, popular choices for the activation functions were the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$ and the tanh function, because their output is bounded (between 0 and 1 and -1 and 1, respectively) and they have a relatively simple derivative. A single neuron can be used as a non-linear classifier. The parameters for the neuron are not typically selected by the designer, but rather learned from data. In order to do so, the output of the neuron is compared to a desired output called the label using an error function (also called a loss).

A typical loss function for a single neuron is the square error $E(y, y') = (y - y')^2$. In order to update the parameters of the neuron, the contribution of each parameter to the error is calculated. For the square error, the calculation is as follows:

$$\begin{aligned} \frac{d}{dw_i}(y - y')^2 &= 2(y - y')\frac{d}{dw_i}(y - y') \\ &= 2(y - y')\frac{d}{dw_i}(\Phi(w^\top x) - y') \\ &= 2(y - y')x_i\Phi'(w^\top x) \end{aligned} \tag{2.2}$$

Then the parameters of the neuron are updated in proportion to their contribution to the loss:

$$w'_i = w_i + \eta \frac{d}{dw_i}E(y, y') \tag{2.3}$$

The parameter η is called the *learning rate*. Because the parameters are updated according their partial derivative of the error (called the gradient), this method of learning is called gradient descent. Because it requires labeled data, it is considered supervised learning.

A single neuron only produces a single output. It is common to group neurons into layers with all neurons in a layer sharing the same inputs. However, each neuron still has its own parameters, which are now a matrix W. Additionally, we introduce a new bias parameter b that is added to the weighted sum of inputs. The definition then becomes as follows:

$$y = \Phi(Wx + b) \tag{2.4}$$

When using the same activation functions as before, they are applied element-wise to the vector $Wx + b$. However, we can also use activation functions that are applied to the whole vector, such as the *softmax* function, which takes unscaled inputs and computes a probability distribution.

$$s(x)_i = \frac{\exp(x_i)}{\sum_j \exp(x_j)} \tag{2.5}$$

The softmax function is usually used in the final layer of a network used for classification and its output is interpreted as the probability that the sample belongs to a given class. In classification tasks, it is also common to use the crossentropy loss function:

$$C(y, y') = -\sum_i y'_i \log(y_i) \tag{2.6}$$

In this case, y' is the true probability distribution for a sample. In many cases, a sample can belong to only one class in which case the probability distribution will be zero for all other classes and one for the correct class.

A significant improvement of neural networks and the development that led to their popularity was the invention of the backpropagation algorithm [48]. This allowed using multiple layers of neurons, with the input to each layer being the output of the last. The derivative of the error is calculated with respect to every parameter in the model (all weights and biases) and they are updated accordingly. There have been several improvements to the update rule from simply adding the gradient scaled by a learning rate constant, such as stochastic gradient descent:

$$w_i' = w_i + \eta \frac{d}{dw_i} E(y, y') + \xi \tag{2.7}$$

where ξ is a random variable evaluated in each step. This is an improvement that aims to combat a common problem of neural networks: Local minima. Because the parameters never move against the gradients and the gradients are always directed so as to reduce the error, it is possible for the model to be "stuck" in a configuration where the error can no longer decrease, but a different configuration of parameters exists which would have a lower error. This configuration cannot be reached by gradient descent as defined in 2.3, because to get there, the model would have to take a step that increases the error before it would decrease again. Adding a random term to the update rule allows the model to continue improving in some of these cases.

Another common problem of neural networks, especially networks with many layers, is the problem of vanishing gradients. This problem generally comes about in one of two ways: Either a parameter is so removed (so many layers apart) from the loss that its contribution to it is minute, or a neuron can become "saturated", i. e. it has a sigmoid or tanh activation function and the input to the function is very far from zero in either direction. Both functions asymptotically approach a value in those cases, which means their derivatives become infinitesimal. Both phenomena prevent the model from making significant updates to its parameters, meaning it does not learn.

The two causes must be adressed seperately. The simplest solution would be to simply refrain from using networks with many layers. However, it has been shown in that order to solve complex problems, depth is preferable to width, as using networks with wider layers would require more neurons in total, increasing the computational complexity [38]. One way of adressing the vanishing gradient problem as a result of many layers is the introduction of so-called residual connections, i. e. adding the input of a layer back to its output:

$$y = \Phi(Wx + b) + x \tag{2.8}$$

The second cause can be adressed by using a different activation function. The requirements for an activation function for neural networks are primarily to be a non-linear function, easy to calculate and to have a simple derivative. For this reason, the rectified linear function has recently been popular:

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.9}$$

This function can only saturate in one direction, rather than both like sigmoid and tanh. Nevertheless, it does not completely solve the problem of saturation, as a neuron can “die”, that is, its parameters can come to a point where the output of the rectified linear function is zero for nearly all inputs. This case is best avoided by choosing a lower learning rate. Despite this problem, the rectified linear function has been empirically shown to work well in deep neural networks.

2.1.1 Recurrent Neural Networks

When dealing with sequential information, such as audio (which is sequential in time) or text (which is sequential in that the words follow one another), we must model dependencies between steps. If each step of the sequence were fed into the network individually, they would be considered independent and no information could flow from one step to the next. If they were fed in all at once, any dependency could be modeled, but the whole sequence would have to be known in advance. For very long sequences, this might also be a practical issue—the sequence or the resulting calculation may be be too large for the computer’s memory.

Recurrent neural networks present a compromise between these two options: Input is fed in sequentially (allowing sequences of arbitrary length) and the network has access to some information from the previous step (making each run depend on all previous ones). One such architecture is proposed by Elman [9]:

$$\begin{aligned} h_t &= \Phi_h(W_h x_t + U_h h_{t-1} + b_h) \\ y_t &= \Phi_y(W_y h_t + b_y) \end{aligned} \tag{2.10}$$

Here, the network has access to the hidden state h from the previous step in order to compute the current hidden state and output. The outputs of an RNN are themselves a sequence of the same length as the input sequence, so it is possible to use multiple layers of RNNs, each using the output of the last layer as input. When training the network, the recurrent connections are *unfolded*, that is, the layer is duplicated a number of times equal to the sequence length, into a version with no recurrent connections. The duplicates share parameters and gradients are accumulated through all steps of the network. This is called backpropagation through time.

An unrolled network has a depth equal to the sequence length, which may be dozens or even hundreds of steps. This intensifies the vanishing gradient problem, explained above. Also, because new information is added to the hidden state in each time step, old information (which is important for long-term dependencies) may not be retained. As a result, RNNs are poor learners and are largely incapable of modeling long-term dependencies in the input data. They do however provide a solid framework for handling input sequences of arbitrary length. Consider also this architecture:

$$\begin{aligned} h_t &= \Phi_h(W_h y_{t-1} + U_h h_{t-1} + b_h) \\ y_t &= \Phi_y(W_y h_t + b_y) \end{aligned} \tag{2.11}$$

There is no input to this layer except for the initial hidden state h_0. The output of each step, along with the hidden state, is the input for the next step. This provides a way for the network to generate an output sequence from a fixed-length input, an important ability for tasks like machine translation or speech synthesis. While it suffers from the same problems as the above RNNs, it provides a basic framework for sequence-to-sequence tasks [36]. In these tasks, an input sequence is transformed to a fixed-length representation, then an output sequence is generated from this representation, which may have a different length than the input sequence (such as English text being translated to German text).

In many applications, it is not only important to model the relationship between one step of the input and the previous steps but also in the opposite direction i. e. the following steps. For example, in the phrase "an apple", the word "an" depends on the following word "apple". RNNs can only model backward dependencies. The common solution is to use *bidirectional* RNNs. A second, independent RNN is fed the input in reverse order. The output of the network at step t is the concatenation of networks' states after processing the input at position t (from the front). That means the backward RNN must be calculated for $n - t$ steps, where n is the total sequence length, which must therefore be known in advance. This is the only drawback to the model, otherwise it can be trained with the same methods as one-directional RNNs.

Neural networks have been shown to be able to deal with temporal information without using recurrence, such as by using time-delayed neural networks (TDNNs) [42] or convolutional neural networks (CNNs) [e. g. 10]. These non-recurrent models have the advantage of being able to more effectively make use of parallel computing hardware—in recurrent networks, calculations for the next timestep depend on the result of the current timestep. Both paradigms are the subject of active research today, and there are published models using recurrent networks as well as models using non-recurrent networks (and some that use both!) that achieve state-of-the-art results on a number of different tasks, such as sentence classification (non-recurrent [18], recurrent [19]), speech recognition (non-recurrent [1], recurrent [8]) and of course question answering (non-recurrent [52], recurrent [5]).

2.1.2 Long Short-Term Memory

Long Short-Term Memory (LSTM) cells [14] present a major improvement on the "classic" RNN cells presented above. The major innovation of LSTMs is the introduction of so-called *gates*. A gate is a neuron layer that determines how much of the previous hidden state will be retained and how much the new input will affect the hidden state. They do so dependent on the input and the previous hidden state. For example, consider a network that is processing text. When encountering the sentence "Lucy went to the doctor", the network may decide that the words "to" and "the" should have a lesser impact on the hidden state of the network than the words "Lucy", "went" and "doctor", as they are not as important to capturing the meaning of the sentence.

Similarly, another gate determines how much of the old hidden state should be "forgotten". Consider a network that should output the sentence "He has gone away".

In order to know the correct grammatical form to use for the word “gone”, the network must remember some part of the word “has”, as it determines that the sentence is in the simple perfect. However, after the word “gone” has been output, no more words that depend on the tempus of the verbs follow. The information can be safely forgotten. More precisely, an LSTM can be defined as follows:

$$\begin{aligned}
f_t &= \sigma(W_f[h_{t-1}, x_t] + b_f) \\
i_t &= \sigma(W_i[h_{t-1}, x_t] + b_i) \\
o_t &= \sigma(W_o[h_{t-1}, x_t] + b_o) \\
c_t &= i_t \odot \tanh(W_g[h_{t-1}, x_t] + b_x) + f_t \odot c_{t-1} \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned} \tag{2.12}$$

In this definition, h_t is the output of the network at each timestep, f_t, i_t and o_t are the forget, input and output gates, respectively and c_t is the cell memory state. $\odot$ denotes an element-wise product, $[\cdot]$ denotes concatenation. LSTMs have been shown to be much more resilient to the vanishing gradient problem, as well as being able to model long-term dependencies with much greater reliability than RNNs. This comes at the price of complexity, however: An LSTM has more parameters than a traditional RNN (more than twice as many as the RNN model introduced above) and requires additional calculations for the gates. Nevertheless, LSTMs have been widely used for a variety of tasks, including question answering (see for example Chen et al. [5] and Tay, Tuan, and Hui [37]).

2.1.3 Encoder-Decoder Models

A common neural network architecture is that of the *encoder-decoder model.* The network consists of two parts, the encoder and the decoder. The encoder transforms the input into an internal representation (typically with fewer features than the original input). The decoder uses only this internal representation to perform the task the network is designed for. The encoder is often not task-specific and may be pre-trained on a different task, or using an unsupervised model.

One common way to train a non-task-specific encoder is to build an encoder-decoder model with the task of recreating the input from its hidden representation. This model is called an *autoencoder.* The idea is that the internal representation contains enough information to exactly recreate the input and can therefore effectively replace it. As it has fewer features than the input, a network working on the representation learned by the encoder needs fewer parameters and may train faster (due to reduced computational complexity) or converge in fewer training steps (which also leads to faster training).

Another advantage of this model is that it allows training multiple decoders that use the same encoder and vice-versa. This is one approach to machine translation—for each source language, train an encoder and for each target language, train a decoder, rather than a full network for each source-target pair.

Yet another typical use of the encoder is to calculate a fixed-length representation from an input that is a sequence of arbitrary length. Common methods include a recurrent model like LSTMs, using the last output of the model as the final representation, or *pooling* over all timesteps, such as element-wise max pooling: $y_i = \max_t y_{t,i}$ or summation: $y = \sum_t y_t$.

2.1.3.1 Word Embeddings

Neural networks work on continuous variables, i. e. floating point numbers. However, text input consists of discrete units (words) without an inherent ordering or algebraic operations. One approach to feeding them into the network is to treat each word as its own binary feature and present the network at each step with a *one-hot vector*, i. e. a vector that has zeros in all positions except the one corresponding to the current word's feature. This will mean a number of features equal to the vocabulary size (usually at least 100 000), which is impractical to use throughout the whole network.

It is therefore the job of the first layer to drastically reduce the dimensionality of the input to some managable number of features, in the order of several hundred. Because the input to this layer is one-hot vectors, a simple optimization is to treat this layer not as a matrix multiplication (as is normally the case) but as a simple lookup, indexing the layer's weight matrix. The rows of the weight matrix are referred to as word embeddings. While it is possible to train word embeddings as part of training the rest of the network, the large number of parameters makes it very difficult to train them effectively without overfitting.

A common approach is to use word embeddings that have been pre-trained using an unsupervised training method such as GloVe [27]. This has the advantage that there are sensible word embeddings present even for words that don't occur in the model's training set, which helps the model to generalize. Word embeddings are almost always combined with more complex, task-specific encoders (however, see Shen et al. [35]).

2.1.3.2 Attention

The above methods of arriving at a fixed-length representation lose a lot of information—we throw away all LSTM outputs except the last one or throw away all elements of the sequence except the maximum in each dimension. Neural network performance has been shown to improve massively when using an *attention mechanism* that allows it to make use of all available information (see e. g. Seo et al. [34]). An attention mechanism sits between the encoder and decoder and weighs all outputs of the encoder (rather than a reduced representation), then calculates a weighted sum which is given as output to the decoder. If the decoder produces a sequence, the attention mechanism might even be invoked in every step, to determine what parts of the input are relevant to the current output (consider for example a machine translation task where for each output word, we must decide what word or words of the source language will be translated next).

There exist different variants of attention mechanisms. Later on in the experiments, we will use a variant of the *trilinear* attention based on Seo et al. [34] and Yu et al. [52]. It is based on the trilinear similarity function for vectors a and b:

$$f(a, b) = w^{\top}[a, b, a \odot b] \tag{2.13}$$

where w is a trainable vector.

As we are dealing with a question answering context, we have two inputs: a context paragraph $P \in \mathbb{R}^{n \times d}$ and a question $Q \in \mathbb{R}^{m \times d}$. The paragraph is a sequence of length n, the question of length m and d is the dimensionality of the representation of each word in the sequences. The attention mechanism must find correlations between the paragraph and question and reduce the information of the paragraph to only that which is relevant to the question. We do this by calculating the similarity matrix $S \in \mathbb{R}^{n \times m}$:

$$S_{ij} = f(P_i, Q_j) \tag{2.14}$$

where f is the trilinear similarity function as above and P_i, Q_j are the i-th row of P and the j-th row of Q, respectively. We now calculate a weighted sum of the question vectors, according to how relevant they are to each paragraph vector. This means we will perform n sums (one for each paragraph vector) and require n sets of weights $a_i \in \mathbb{R}^m$ with $\sum_j a_{ij} = 1$ for all i. We calculate the attention output $A \in \mathbb{R}^{n \times d}$ as follows:

$$A_i^C = \text{softmax}(S_i) \in \mathbb{R}^{n \times m} \tag{2.15}$$

$$A = A^C Q \in \mathbb{R}^{n \times d} \tag{2.16}$$

QANet [52] and DCN [50] suggest an extension to this method which calculates a weighted sum of context words for each word in the question. The authors of DCN describe this mechanism as: “One possible interpretation for the operation [...] is the mapping of question encoding into space of document encodings *(sic)*” [50].

$$A_j^Q = \text{softmax}(S_j^{\top}) \in \mathbb{R}^{m \times n} \tag{2.17}$$

$$B = A^C A^Q C \in \mathbb{R}^{n \times d} \tag{2.18}$$

The final attention output is then $[P, A, P \odot A, P \odot B] \in \mathbb{R}^{n \times 4d}$, following Seo et al. [34]. Note that the attention output has reduced dimensionality from the input—the variable question length has been removed. However, the output is still not fixed-length. This is desired for the traditional question answering task, where we want to find the start and end point of the answer span within the context paragraph. For a sentence classification task with a fixed number of classes (like the one discussed for this thesis, which has two classes: accept and reject), the dimensionality must be further reduced. Following Kim et al. [19], we reduce by max-pooling along each column, producing an output vector $a \in \mathbb{R}^{4d}$.

2.2 Information Retrieval

Information retrieval is the task of finding a piece of information in a collection of such pieces that is relevant to a particular query. The related problem of cataloging and indexing information to allow for faster and more accurate retrieval has been a driving and shaping force in the development of computing since its very beginning, even though the term information retrieval has been in use only since 1951 [24]. The development of the internet has been especially motivating to advance the field of information retrieval and has made the collection of large-scale datasets possible.

2.2.1 TF-IDF

Term frequency inverse document frequency (TF-IDF) [32] is a method of encoding a sentence or paragraph into a sparse vector representation suitable for information retrieval. Its central assumption is that if a sentence has a high occurence of a term that is generally rare in the corpus, it is more important to the meaning of the sentence than words which are frequent throughout the whole corpus. This assumption is not entirely true from a semantic perspective (it can make a large difference whether one sees “the man” or “a man”. Even though “the” and “a” are both very common words, they can be defining for the meaning of the sentence in this case), but it largely holds in the context of information retrieval, where we are often looking for specific keywords that define the topic of a document.

The encoding is calculated (as the name implies) as the product of the term frequency and the inverse document frequency for a document d and a term t:

$$\begin{aligned} \text{TF-IDF}(d,t) &= \text{TF}(d,t)\,\text{IDF}(t) \\ \text{TF}(d,t) &= \text{number of occurrences of } t \text{ in } d \\ \text{IDF}(t) &= \log\frac{n_d}{\text{DF}(t)} + 1 \\ \text{DF}(t) &= \text{number of documents that contain } t \end{aligned} \tag{2.19}$$

where n_d is the number of documents. This is only one possible definition of TF-IDF. Common to all definitions is only that a weight is assigned to the *term frequency* (the number of times a term occurs in a document) and the *document frequency* (the number of documents in the corpus that contain a given term). The final encoding is obtained by multiplying the two weighting factors. An alternative definition to the above (which we will use in this work) is as follows:

$$\begin{aligned} \text{TF}'(d,t) &= \log(1 + \text{TF}(d,t)) \\ \text{IDF}(t) &= \log\frac{n_d - \text{DF}(t) + 0.5}{\text{DF}(t) + 0.5} \end{aligned} \tag{2.20}$$

Because a single document will only contain a small fraction of words from the vocabulary, the resulting vector will be very sparsely populated. In order to map a term to a position in the vector, the system must hold a mapping structure in memory, which constrains the system's ability to parallelize the TF-IDF operation, as well as taking up an unnecessarily large amount of memory. It is therefore common to use the *hashing trick*: An index in the vector is assigned to each word by a stateless hashing function. As the function is deterministic, its results do not need to be stored in memory and the operation can be done in parallel.

However, using a hashing function comes with two drawbacks: First, hashing collisions could occur, assigning two different words the same index in the vector. This would either override or sum their respective TF-IDF terms, depending on the implementation. While this may be unproblematic for some words, it is nevertheless best avoided by selecting a large enough feature space (such as 2^{20} or more features) which makes hashing collisions increasingly unlikely. Storing the feature vectors as a sparse matrix still allows for efficient storage in memory.

Secondly, when using a traditional lookup to assign words to indices in the vector, there are as many entries in the vector as there are words in the vocabulary. Each word is guaranteed to occur in at least one document, otherwise it would not be in the vocabulary. Therefore, dividing by $\mathrm{DF}(t)$ is safe, is it can never be zero. When using hashing, it is possible that no words map to a particular index in the vector, meaning the document frequency would be zero. We therefore add a normalization in 2.20 to avoid dividing by zero. A similar normalization can be applied to the definition in 2.19:

$$\mathrm{IDF}(t) = \log \frac{n_d + 1}{\mathrm{DF}(t) + 1} + 1 \tag{2.21}$$

In this case, the normalization is equivalent to adding an additional document to the corpus which has all features.

In order to rank documents according to a query, the query is transformed into a feature vector by the same algorithm, then it is compared to all documents in the corpus using the inner product or dot product. If the vector is normalized, this is also called the cosine similarity, as the resulting value is the cosine of the angle between the two vectors.

$$\begin{aligned} \mathrm{sim}(a, b) &= a^\top b \\ \mathrm{best}(q) &= \underset{d}{\mathrm{argmax}}\, \mathrm{sim}(\text{TF-IDF}(q), \text{TF-IDF}(d)) \end{aligned} \tag{2.22}$$

Because all entries in the TF-IDF vector are positive, the inner product is also positive. If they are also normalized, the cosine similarity is bounded by $[0, 1]$. In either case, an inner product of 0 indicates that the two documents have no word in common and a high number indicates a strong similarity between the documents. For performance reasons, the paragraph encodings are calculated ahead of time, as they are independent of the query. The retrieval complexity scales with the number of paragraphs in the corpus, but due to efficient sparse matrix storage and multiplication algorithms, remains acceptable well into the millions of paragraphs.

3 Related Work

In this chapter, we will review previous research done on the problems of question answering as well as information retrieval. We will also take inventory of the available datasets for question answering.

3.1 Question Answering

The earliest approach to answering questions in natural language involved applying grammatical analysis to the question, then retrieving the answer by a set of pre-defined rules. This approach built on the successful application of rule-based transformation to questions used in ELIZA [46]. ELIZA was not built to actually answer questions, but instead turn them around on the asker—for example, given the question "Who are you?", the program may answer "Why are you interested in whether I am or not?" This requires a syntactic analysis of the question that later programs built on, such as Baseball [12], one of the first question answering systems. It is a closed-domain system, answering questions on baseball statistics.

Rule-based systems continued to be dominant at least until the end of the 1990s (see for example Voorhees [40]) and some are still developed today [31]. However, designing a rule-based system requires enumerating every possible variation of question or proposing a formal grammar that all questions must conform to. Similarly, to extract information from natural language text, it must also be in a specific format. These systems cannot deal with the variety present in natural language and cannot be said to comprehend text. Nevertheless, analysis methods developed for rule-based systems, such as part-of-speech taggers or named entity recognizers continue to be in use to enrich the input features [5].

Rule-based systems often have strong requirements for the format of the question (for example, that it must start with an interrogative) whereas the development of question answering in general strives towards removing as many of these artificial requirements as possible and allow asking questions in completely free-form natural language.

While current datasets mostly consist of such challenges (i. e. with freeform questions), some datasets use cloze-style questions, an intermediate type between artificial and freeform question: A cloze-style question is a statement with one or more of the relevant words blanked out. The "answer" is the word or phrase that correctly fills in the blank.

Cloze tests have been used in education (see for example Bachman [2]) to test comprehension when learning a foreign language. It is an intermediary between a fixed-format question and a freeform question, because the sentences that form the basis for the cloze are freeform, but a cloze-style sentence provides more contextual clues to the answer than a question does, as well as being more similar to information the system might find in context documents (which also describe facts using statements, rather than questions).

3.1.1 State of the Art

The most successful question answering systems today are based on neural networks, achieving near-human performance or in some cases even outperforming humans[1]. Neural networks have been shown to be able to deal with the inconsistent structure of natural language as well as recognize semantic subtleties, such as sentence entailment [49]. They are therefore a promising (and widely researched) approach to question answering.

3.1.1.1 Traditional Question Answering

In the traditional question answering task, some context information is provided for each question, constraining the information retrieval portion of the task to only finding the span of text within the context that represents the answer. A large number of approaches to this task have been published, several of which we will discuss here because of their relevance for this work:

QANet At the time of writing, QANet [52] is the best-performing published model for the SQuAD dataset (see chapter 4). Its main innovation is that it refrains from using recurrent neural networks like LSTMs, opting instead for an approach based on convolutional neural networks. The authors report improved evaluation and training speed as a result. It is mainly based on the BiDAF (Bi-Directional Attention Flow, Seo et al. [34]) network and DCN (Dynamic Coattention Networks, Xiong, Zhong, and Socher [50]), for their advances in attention mechanisms and classifier architecture. QANet is a typical encoder-decoder network, encoding the context and question through a number of convolutional operations, combining the representations through an attention mechanism and using a classifier on the result to predict the beginning and end of the answer span within the context.

Reinforced Mnemonic Reader The Reinforced Mnemonic Reader [16] is another high-performing model for the SQuAD dataset. In addition to encoding question and context vectors with recurrent neural networks (specificially, bidirectional LSTMs), the mnemonic reader makes use of a novel Semantic Fusion Unit (SFU), which fuses several vectors together using a gating process similar to that used in LSTMs. The model can be trained both in the usual way, minimising the crossentropy error of the predicted distribution of start and end pointers, and by using a reinforcement learning technique, maximising the model's expected reward, which is based on the answer's F1 score (a measure of the "overlap" between the true and predicted answer).

[1] On the SQuAD leaderboard (`https://rajpurkar.github.io/SQuAD-explorer/`), QANet outperforms humans on the exact match metric.

Answer Re-Ranking Wang et al. [43] propose a method of re-evaluating answer candidates from a question answering system. The question answering system (they use R^3 from [44]) returns the top 10 answers by probability and the re-ranker uses criteria such as coverage of the context to select from them the top answer. In compound, the system achieves a better accuracy than the question answering system would have on its own.

3.1.1.2 Open-Domain Question Answering

In open-doman question answering, the context is either not directly provided along with the question, or it is, but is too broad or too much to be used by the question answering system directly. Such systems must use an information retrieval (IR) component to narrow the available context down to a managable amount that can be fed into the (mostly neural network based) question answering system. Because this task is the focus of this work, we must measure our performance against the models from this section. However, due to different datasets being used or different amounts of context being retrieved, exact numbers are often difficult to compare.

DrQA Short for Document Reader Question Answering, DrQA [5] exemplarizes a typical system for this problem: From a corpus of Wikipedia articles, the top five articles are retrieved with TF-IDF [32]. Those articles are split into paragraphs and all fed into the question answering system, which aggreagates answer candidates, finally selecting the most likely answer from among all answer candidates from all paragraphs from all articles. Because they also test mainly on the SQuAD dataset, their work is an interesting point of comparison for the models proposed in chapter 5.

Classifiers Models such as R^3 [44], relation-networks (RN) ranker [15] and supervised semantic indexing (SSI) [3] all follow the same basic principle: a classifier is applied to all question-context pairs to determine the most likely context. While this is a powerful method (see chapter 6), this method scales very poorly to large context collections. R^3 and RN operate on QUASAR-T (see section 3.2) and similar datasets, ranking the passages provided by these datasets. SSI constructs a testing set of 10 000 or 100 000 passages by taking all correct results and filling the rest up with random context paragraphs. Both approaches require prior knowledge about the problem instance.

Wikipedia QA Another model to answer questions using Wikipedia as a knowledge source is proposed by Ryu, Jang, and Kim [31]. It is a rule-based system that does not use any machine learning techniques. Instead, it extracts structured information from wikipedia articles using pre-defined regular expressions and exploiting the structure explicitly embedded in Wikipedia. While they are able to answer over 85% of questions, the results are practically impossible to compare to other models, because their model is hand-tailored to the Korean-language Wikipedia, and their test set consists of only 600 (unpublished) questions which they collected themselves based on criteria which they do not report.

Name	Type	#Queries	Context Length
SQuAD	substring	107k	one paragraph
QUASAR-S	cloze	37k	50 documents or paragraphs
QUASAR-T	freeform	54k	50 long or short passages
TriviaQA	freeform	95k	6 documents
MS-MARCO	freeform	182k	10 paragraphs
RACE	multiple-choice	100k	one paragraph
WikiMovies	list	220k	several synthetic sentences
WikiHop	substring	51k	up to 64 paragraphs
CNN/Daily Mail	cloze	1.4M	800 words on average
NarrativeQA	freeform	47k	synopsis or full book/film script
MCTest	multiple-choice	2k	200 words on average
WikiQA	substring	3k	one paragraph
TREC-8	substring	200	shared 528k article corpus

Table 3.1: Overview of various question answering datasets.

3.2 Datasets

In the following section, we will review various published question answering, information retrieval and reading comprehension datasets. In particular, we will differentiate the datasets with regard to the broadness of the question domain or domains, whether questions are automatically generated or posed by humans, what context is given for each question, both to the system that answers the question as well as to the humans or systems which designed the question. Other distinguishing features are the grammatical variety of the questions, how distributed the required information is (whether the answer requires reasoning over multiple pieces of evidence) and whether answers are expected as free text, as a substring of the context, multiple-choice or cloze-style fill-in-the-blank.

3.2.1 Stanford Question Answering Dataset (SQuAD)

The Stanford Question Answering Dataset (SQuAD) [29] is one of the more widely studied question answering datasets. At the time of this writing, the leaderboard[2] lists over 70 submissions for version 1.1 of the dataset. The best submissions have exceeded human performance on the exact match metric. It could therefore be argued that the challenge is “solved” and further study of this dataset is unlikely to yield more interesting results. It remains of great interest to this work, however, precisely because so many high-performing question answering systems exist on this task. In

[2] `https://rajpurkar.github.io/SQuAD-explorer/`

Context: One of the most famous people born in Warsaw was Maria Skłodowska-Curie, who achieved international recognition for her research on radioactivity and was the first female recipient of the Nobel Prize. Famous musicians include Władysław Szpilman and Frédéric Chopin. Though Chopin was born in the village of Żelazowa Wola, about 60 km (37 mi) from Warsaw, he moved to the city with his family when he was **seven months old**. Casimir Pulaski, a Polish general and hero of the American Revolutionary War, was born here in 1745.
Question: How old was Chopin when he moved to Warsaw with his family?
Answer: seven months old

Figure 3.1: A sample problem instance from SQuAD.

chapter 4, we will transform this dataset into an open-domain challenge, while making use of the already published results.

In order to create the dataset, the authors selected 536 articles from the English Wikipedia from the top 10 000 articles by PageRank [25][3]. After filtering tables, images and paragraphs under 500 characters, the remainder is 23 215 paragraphs. Using Amazon Mechanical Turk crowdsourcing, up to five questions were posed by human annotators for each paragraphs, resulting in 107 785 questions in total.

Figure 3.1 shows a problem instance from SQuAD: One paragraph of context is given in addition to the question, the answer is a substring from the text. In an additional step, the authors collected up to four alternative answers for each question, all of which are considered correct answers. Up to five questions share each context paragraph. The paragraphs may share topics (as they may come from the same article) but between them cover a wide variety of topics. Nevertheless, SQuAD is not an open-domain question answering dataset. The authors point out that the goal of open-domain question answering is "to answer a question from a large collection of documents", with this dataset not requiring that but rather requiring *answer extraction*, i. e. finding the correct answer span in a single document.

In June 2018, version 2.0 of the dataset was released [28], which extends the existing dataset with a new class of question-context pair: A problem instance where the question can *not* be answered using the paragraph provided, often because of a subtle semantic variation. While this introduces a necessary step of distinguishing whether the paragraph is valid, pushing the problem closer to a true open-domain one, this new feature is not useful for this work, which will continue to work with version 1.1 of the dataset as a result.

3.2.2 QUASAR

QUestion Answering by Search And Reading (QUASAR) [6] introduces two different datasets with two different tasks: QUASAR-S and QUASAR-T.

[3] Using a list compiled by *project Nayuki*: `https://www.nayuki.io/page/computing-wikipedias-internal-pageranks`

Question: javascript—javascript not to be confused with java is a dynamic weakly-typed language used for **XXXXX** as well as server-side scripting.
Answer: client-side
Context excerpt:
JavaScript is not weakly typed, it is strong typed.
JavaScript is a **Client Side** Scripting Language.
JavaScript was the **original** **client-side** web scripting language.

Figure 3.2: Sample problem instance from QUASAR-S.

Question: 7-Eleven stores were temporarily converted into Kwik E-marts to promote the release of what movie?
Answer: the simpsons movie
Context excerpt:
In July 2007, 7-Eleven redesigned some stores to look like Kwik-E-Marts in select cities to promote **The Simpsons Movie**.
Tie-in promotions were made with several companies, including 7-Eleven, which transformed selected stores into Kwik-E-Marts.
"7-Eleven Becomes Kwik-E-Mart for '**Simpsons Movie**' Promotion".

Figure 3.3: Sample problem instance from QUASAR-T.

QUASAR-S (S for Stackoverflow) is a cloze-style closed-domain question answering task consisting of 37 000 statements. The tasks have been automatically generated from the programming help section of `stackoverflow.com`. Each statement is the definition of a programming-related question tag (such as `java` or `.net`), as given on the site. The tag itself is replaced by a placeholder. Two variations of the dataset exist, with differernt forms of context: One version has 50 long parts (the full answer plus the full text of the question) or 200 short parts (one sentence) from the 50 top-rated question threads related to the question's tag. Figure 3.2 shows a problem instance from QUASAR-S.

QUASAR-T (T for trivia) consists of almost 54 000 human-written trivia questions on a large variety of topics. As a knowledge base for answering these questions, it uses `ClueWeb09`, a collection of about one billion web pages. However, the questions were written and collected independently from this corpus and come from different sources on the web. From the `ClueWeb09` websites, 50 long (2048 characters) or 200 short (200 characters) context pseudodocuments are selected. Because of the nature of the context retrieval, it is not guaranteed that the answer string is contained in any context document.

Both *QUASAR* datasets provide more context than SQuAD, but the documents are expected to be highly redundant, with the right answer being contained in more than one context document.

Question: The Dodecanese Campaign of WWII that was an attempt by the Allied forces to capture islands in the Aegean Sea was the inspiration for which acclaimed 1961 commando film?
Answer: The Guns of Navarone
Context excerpt:
The Dodecanese Campaign of World War II was an attempt by Allied forces to capture the Italian-held Dodecanese islands in the Aegean Sea following the surrender of Italy in September 1943, and use them as bases against the German-controlled Balkans. The failed campaign, and in particular the Battle of Leros, inspired the 1957 novel **The Guns of Navarone** and the successful 1961 movie of the same name.

Figure 3.4: Sample problem instance from TriviaQA.

3.2.3 TriviaQA

TriviaQA by Joshi et al. [17] is similar to QUASAR-T in that it consists of human-written trivia questions on a wide number of topics gathered from the web. It consists of 95 000 question-answer pairs, each is annotated with on average six different context documents, making for a total of 650 000 question-answer-context triplets. While the questions were written independently of any context, the dataset does include two set of context documents for each question. The first set is based on a crawl of top Bing web seach results for the question, excluding keywords like “trivia”, “question” or “answer”. The second set consists of Wikipedia articles of any entities that were automatically recognized in the question.

As a result, TriviaQA provides a very large amount of context for each question. The authors also provide an unfiltered version (where context documents that do not contain the answer string have not been filtered out), which contains 110 495 question-answer pairs and 740 000 context documents. Even the filtered version has a considerably larger context for each question than SQuAD. The average context length for TriviaQA is 2 895 words, making many approaches that may work on SQuAD infeasible for this dataset.

3.2.4 MS-MARCO

The Microsoft MARCO (MAchine Reading COmprehension) dataset consists (in its most recent version at time of writing) of 182 000 web queries and answers. The questions are real queries given to the Bing search engine by users, collected and anonimized by Microsoft. The answers are written by crowdworkers, who were encouraged to write full-text answers. As context, they were given the top search results from the given query. These same extracts are also included in the dataset as context documents. Additionally, each question is annotated with the URLs of the web pages that the context excerpts were extracted from as well as a classification tag such as ENTITY or NUMERIC, describing what kind of answer is expected. This tag is not intended for training, but for analysis of the models (e g whether the models struggle with one category of question more than others).

Query: will I qualify for osap if I'm new in canada
Context (excerpts):
[...] you must be a: 1. Canadian citizen; 2. Permanent resident or 3. Protected person/convention refugee [...]
You will not be eligible for a Canada-Ontario Integrated Student Loan [...]
Answer: No. You won't qualify.

Figure 3.5: Sample problem instance from MS-MARCO

Context excerpt: . . . Many people optimistically thought industry awards for better equipment would stimulate the production of quieter appliances. It was even suggested that noise from building sites could be alleviated. . .
Question: What was the author's attitude towards the industry awards for quieter? *(sic)*
Answers: A: suspicious, B: positive, C: enthusiastic, D: indifferent

Figure 3.6: Sample problem instance from RACE.

3.2.5 RACE

The Large-scale ReAding Comprehension dataset from Examinations (RACE) [22] consists of 28 000 english passages and 100 000 questions written by english instructors for chinese middle and high school english classes. As the examination questions target an age range of 12 to 18-year-olds, there is a significant difference in difficulty between the questions. Therefore, the authors propose a split of the dataset into RACE-M, consisting of the questions for middle schoolers (12-15 years old) and RACE-H, with the questions for high schoolers (15-18 years). The questions have four multiple-choice answers each.

What sets this document apart from other QA datasets is a higher degree of reasoning required, as well as many of the questions requiring multi-sentence reasoning. As a result, at the time of writing, published systems are not close to surpassing human performance.

3.2.6 Others

Other datasets not studied in detail here include WikiMovies [7], WikiHop [47], CNN/DailyMail [13], NarrativeQA [21], MCTest [30], WikiQA [51] and TREC-8-QA [41]. Some basic statistics about them can be seen in table 3.1 and a reasoning for why they are not suitable for this work can be found in section 4.1.

4 Data

In the previous chapter, we examined a number of datasets for open- and closed-domain question answering. However, not all of them are suitable for training the kind of information retrieval model we are looking to study in this thesis. In this chapter, we will discuss the selection of datasets for training and cross-evaluation as well as the steps taken to prepare the data for this task.

The goal of our task is, given a question, to retrieve one paragraph that is sufficient to answer the question from a large corpus of paragraphs common to all questions. No large-scale dataset exists for this specific purpose, but one could be easily constructed by using one of the existing question answering datasets and dicarding the context they provide. The difficulty with this approach is judging whether a retrieved paragraph is correct. For datasets where the answer is a substring of the context, a simple heuristic exists: If the retrieved paragraph contains the answer string, it is judged correct. This approach is still noisy, as a string identical to the answer string might occur, but is used in a completely different context. For example, if the question asks in what year an event occured, the paragraph might mention the correct year, but talk about a different event. In that case, the paragraph should actually not be considered correct.

Another possibility, which can also be used for tasks which require freeform answers, is to train a model using distant supervision: If a question answering system can, given the question and retrieved paragraph, answer the question correctly, then the paragraph must have been sufficient. This approach is also noisy, as there is currently no question answering system that can always find the correct answer, even given the correct paragraph.

Finally, if our new dataset contains all of the original dataset's contexts, we can apply the strictest metric: A retrieved paragraph is considered correct if and only if it is the exact same one that was originally given as context for the question in the original dataset.

The exact method or methods we can use to train an information retrieval system depends heavily on the nature of the dataset. Therefore, we carefully consider all available datasets with regard to their usability for this task.

4.1 Dataset Selection

The datasets in the bottom section of table 3.1 are all unsuitable for this task after only a brief examination: WikiHop and WikiMovies consist of synthetic questions, generated automatically from structured knowledge bases, they are not the kind of realistic task we are interested in. CNN/Daily Mail is a cloze-style task. In these tasks, the appropriate context section is likely to be very similar to the query, as the query is just a statement with the relevant entity blanked out. They are also not a realistic application—humans do not ask for information in cloze-style queries. NarrativeQA could be an interesting information retrieval task, but the information required to answer the questions in that dataset is highly distributed and requires advanced reasoning and interpretation of the text. It could be an interesting target for future work, but for this thesis we will focus on a simpler task. MCTest, WikiQA and TREC-8 are quite simply too small to train a neural network model and therefore unsuitable.

RACE RACE could be a promising dataset to study information retrieval on; General knowledge questions from middle and high school education should be possible to answer using a readily available corpus (such as Wikipedia) and the dataset provides a label of what the "correct" paragraph is—each question is provided with one paragraph of context, if the system retrieves that same paragraph, it is the correct one. However, the questions in RACE do not come from general education, but rather language education. As a result, many questions (such as the example in figure 3.6) ask questions relating to the specific wording and expressions in the given paragraphs, without referencing any topic or entity that the paragraph talks about. This is of course a direct result of the fact that the questions were written for the context paragraphs, not independently of them. The question "What was the author's attitude towards the industry awards for quieter machinery?" both requires one specific paragraph to answer it (no other paragraph talking about quieter machinery, but written by someone else would be good enough) and gives insufficient information to find this paragraph (it does not mention who the author is).

Furthermore, we do not have access to the corpus from which the context paragraphs for RACE have been taken (if one such corpus exists). Constructing one from all paragraphs included in RACE gives 20 000 paragraphs—large, but not comparable to retrieving information from Wikipedia (the English Wikipedia has at time of writing 5 720 000 articles). Supplementing these paragraphs with ones not originally from the dataset would require finding a source with text about similar topics, written in a similar style.

Each of these problems individually would have been possible to work around, but put together they lead us to conclude that RACE is not a suitable dataset for this information retrieval task.

QUASAR and TriviaQA The two trivia datasets, QUASAR-T and TriviaQA are similar in most regards and can be evaluated together: The main problem they both have is that there is no label for what paragraph is sufficient to answer the question. We could test whether the answer string is contained in the paragraph, but as both datasets allow freeform answers, it is possible for a paragraph to have all the information needed but not contain the exact answer string. The distant supervision approach described above has its own drawbacks, such as the binary feedback—either the question answering system answers the question correctly, or it does not. The questions themselves are ideally suited for our task—they are written independently of the context, and must therefor explicitly state their domain . For example, the question “In golf, what is an ‘eagle’?” must mention golf (the question domain), because it is not provided alongside a context paragraph about golf. Without explicitly introducing the domain, it would not be clear what kind of “eagle” the question is referring to. This is important for information retrieval, as it provides us with keywords that make it possible to find a matching paragraph.

Training the model on QUASAR or TriviaQA may be challenging, but they are excellent candidates for evaluation of the system and judging how well it generalizes to unseen data.

SQuAD Being the most widely studied of these datasets, there are a number of question answering systems that are trained on SQuAD amd tailored to its specific task (one paragraph of context, the answer is a substring of the context). As we are looking to combine our information retrieval system with a state-of-the-art question answering system, this is an advantage. The biggest advantage of SQuAD, however, is that the authors provide detailed information on how they narrowed down a full dump of Wikipedia to the about 23 000 paragraphs they used. As the source corpus for these paragraphs is known, we can construct the information retrieval task by taking a larger subset of Wikipedia than the authors of SQuAD did and use the paragraphs provided in the dataset as labels for which ones are correct. In cases where multiple paragraphs could answer the question, we can take advantage of the fact that SQuAD requires that answers are substrings of the context and consider a paragraph correct if it contains the correct answer.

SQuAD does have the disadvantage that the questions were not written indepently of the context. Contrary to RACE, however, they ask for facts expressed in the paragraphs (which could be equally answered from other paragraphs), rather than about sentiment or wording of the particular paragraph. Some questions are still worded in a way that gives too little information about the domain to make a retrieval possible. Nevertheless, SQuAD remains the best option and so we used it as the basis for the construction of an information retrieval dataset.

4.2 Dataset Preparation

The goal of this section is to create a dataset which encompasses all the paragraphs used in SQuAD, as well as other, similar paragraphs, both on the same topics as well as others. To this end, we will try to recreate the pool of paragraphs that the authors of SQuAD drew from. The new dataset will then consist of the same questions as SQuAD, each labeled with the index of the paragraph that the question was originally written for. This gives us the option of using two different means of evaluation: Considering a paragraph correct only when it is the exact same one given in SQuAD (exact match) or accepting a paragraph if it contains the answer string for the question (answer match).

We first take inventory of the dataset: It consists of 107 785 questions on 23 215 paragraphs from 536 articles. The training and development sets are available to the

Article: Chinese Characters
September 26, 2018: In Old Chinese including Classical Chinese, most words [...]
SQuAD: In Old Chinese, (e.g. Classical Chinese) most words [...]
Revision March 9, 2016: In Old Chinese *(e.g. Classical Chinese)* most words [...]
Revision June 18, 2016: In Old Chinese (~~e.g.~~ *and hence in* Classical Chinese) most words [...]
⇒ The article was taken between March 9, 2016 and June 18, 2016.

Figure 4.1: Determining the date of the paragraphs from SQuAD.

public, the testing set remains with the authors and a model can be tested against it on request. Seeing as we will make changes to the dataset, we will work only on the training and development sets, totaling 98 169 question from 20 963 paragraphs from 490 articles.

The authors have provided detailed information about how the dataset was compiled: They started with a full dump of Wikipedia, selected 10 000 articles based on their PageRank score [25], then selected the 536 articles from this pool uniformly at random. All paragraphs shorter than 500 characters were discarded, as well as Wikipedia content that isn't plain text (such as tables, figures, images, links etc.).

For our dataset, we want to use the full pool of 10 000 articles that the 536 from SQuAD were drawn from. The list of articles is available at *Project Nayuki*[1], which is based on a Wikipedia dump from 2014. However, the paragraphs from SQuAD are not taken from that same dump. Neither were they from the most recent version of Wikipedia at the time of research. It is important that the dataset contains the paragraphs exactly as they appear in SQuAD, so that it is still guaranteed that all answers are substrings of their respective paragraphs. In order to make sure all paragraphs from SQuAD were in the dataset, we determined the date of their Wikipedia dump (which they do not report) to around April 2016 based on the date of changes using WikiBlame[2]. Figure 4.1 illustrates this process.

The April 2016 Wikipedia dump is still available on Archive.org[3]. The full dump was reduced to 9 108 articles based on the PageRank list (due to title mismatches between the list from 2014 and the more recent dump, not all 10 000 articles could be retrieved). Wikiextractor[4] was used to remove tables, markup, figures and other non-plain-text features from the articles. In total, the dataset contains 1 060 558 paragraphs. Contrary to SQuAD, we did not filter out paragraphs shorter than 500 characters. It is up to the information retrieval system to determine which paragraphs are relevant. Of the 98 169 questions from the SQuAD train and development set, 92 680 could be matched with their original paragraph, which were split according to their original sets, 82 749 for the training set and 9 931 for the testing set.

This dataset now provides a new and more difficult challenge compared to the original SQuAD, as it is not practical to feed all paragraphs into a question answering system for each question, making an information retrieval component necessary.

[1] `https://www.nayuki.io/page/computing-wikipedias-internal-pageranks`

[2] `http://wikipedia.ramselehof.de/wikiblame.php`

[3] `https://archive.org/details/enwiki-20160407`

[4] `https://github.com/attardi/wikiextractor`

4.3 Full Wikipedia

The reduced dataset is much easier to work with and allows for more rapid iteration when developing and testing models. However, the comparable publications [5, 44] use all of Wikipedia as a basis for answering questions. For this reason, we also construct a dataset from the same Wikipedia dump that is not restricted to the top 10 000 articles, but instead uses all articles.

Using the above Wikipedia dump and filtering out lists and disambiguation pages, we arrive at 4 403 413 articles. To keep the number of paragraphs down, we discard paragraphs shorter than 100 characters, as they are unlikely to contain meaningful information. Afterwards, 24 949 529 paragraphs remain. We store them in an sqlite database for easy indexing and retrieval without having to keep them all in memory.

5 Methods

The information retrieval system must find the most relevant paragraph from one million paragraphs in the dataset. We explore two general approaches to this task: First, a method that does not require a neural network classifier to process pairs of questions and paragraph but instead only calculates an encoding for each question and each paragraph seperately, then compares them using cosine similarity in order to find the most similar one. Second, a method that does use a classifier, but uses another information retrieval system to make a preliminary selection of 100 paragraphs which is then refined to one top result using a classifier.

5.1 Models and Experimental Setup

Previously published models such as [44] and [15] train a classifier that given a question and a paragraph decides whether the paragraph contains an answer to the question. That means that for each question, all paragraphs must be passed through this classifier. The selected paragraph is then the one with the highest confidence. Even with a very efficient encoder, running one million paragraphs through a classifier is too slow to be a practical system. One must also note that these one million paragraph represent less than one percent of Wikipedia, rendering a “realistic” information retrieval task using this method even more daunting. The systems above accomplish this by either using the provided context documents of QUASAR-T or by using traditional information retrieval techniques to retrieve in the order of 200 paragraphs, then run the question answering system on all of them.

Instead, we will attempt to train an encoder of paragraphs and questions such that the most relevant paragraph will also be the closest one to the question by cosine similarity. For a nonlinear encoder function f that produces normalized results, the problem of finding the best paragraph can be descibed as follows:

$$p_{best} = \operatorname*{argmax}_{p \in P} f(q)^\top f(p) \tag{5.1}$$

where P is the set of paragraphs and q is the question. Because the encoder can be evaluated independently of the question, the paragraph encodings can be calculated in advance. During evaluation, the above equation is relatively easy to evaluate, requiring only a matrix multiplication and argmax. This promises to be much faster than running a neural network classifier on a large number of paragraphs.

Note that the question and paragraphs share the same encoder. We also explore the possibility of applying a nonlinear transformation t to the question encoding only, attempting to predict the encoding of the paragraph from that of the question:

$$p_{best} = \underset{p \in P}{\operatorname{argmax}}\, t(f(q))^\top f(p) \tag{5.2}$$

During training, the model is presented with positive and negative examples, i. e. question-paragraph pairs where the paragraph either contains an answer to the question (positive) or does not (negative). Instead of choosing the best paragraph, as above, we simply calculate $f(q)^\top f(p)$ and derive the loss from the resulting scalar which indicates a similarity score between the question and paragraph.

In addition to the cosine similarity, we also explore other ranking schemes which do not rely on classifiers: Square cosine difference $(f(q)^\top f(p))^2$, to constrain the cosine difference to the $[0, 1]$ range and to encourage unrelated paragraphs and questions to receive an orthogonal encoding. We also considered weighted dot product $f(q)^\top W f(p)$, where W is a learned matrix as described in Bai et al. [3]. When W is the identity matrix, this method is equivalent to cosine similarity. In all cases, the output from the encoder is expected to be a normalized vector.

5.1.1 Loss function

The initial loss function considered was mean square error, with the target being 1 for correct paragraphs and 0 for incorrect ones. However, the margin ranking loss [3] also proved effective and we evaluate models on both. It is defined as follows:

$$\sum_{i=1}^{k} \max(0, 1 - r(q, p_{pos}) + r(q, p^i_{pos})) \tag{5.3}$$

With p_{pos} being a positive example paragraph, p^i_{neg} for $i \in \{1, \ldots k\}$ a set of k negative examples for the same question and $r(\cdot, \cdot)$ being the ranking function used. he main advantage of this loss function is that is always considers several examples together without biasing the classifier. At the same time, the system can be presented with data more representative of the true distribution (when predicting, there are hundreds of thousands of negative examples to one positive one). The classification improves if one uses several negative examples for each positive one. The margin rank function avoids biasing the classifier by considering the ranking for the positive example multiple times and by ensuring that a set of considered examples must always include a positive one. At the same time, it is a margin loss, i. e. it rewards not only predicting the correct result with a higher probability than the incorrect one, but also attempts to maximise the margin by which the correct result wins.

5.1.2 Transformations

As a rule, the encoder is shared between the question and paragraph, though we experiment with training two encoders as well. The goal is to produce a question encoding that is similar to that of a paragraph which contains the requested information. If we share an encoder between questions and paragraphs, it is possible that the encoder will only encode information contained in the sentence into the encoding, or otherwise produce encodings such that question and paragraph encodings are quite different as a rule.

We therefore investigate the effect of applying a transformation to the question encoding only, in an attempt to predict the correct paragraph encoding. It is not necessary to exactly predict a paragraph encoding from only the question encoding (after all, a paragraph may contain other information not required to answer the question, which would be contained in its encoding but is impossible to predict form the question encoding alone). The goal is only to produce an encoding that is closer to the encoding of the correct paragraph by applying an additional transformation. We evaluate two transformation models: a simple and a more advanced one.

The simple model consists of a ReLU (rectified linear unit) layer with 500 dimensions, followed by a linear projection to the same dimensionality as the paragraph encoding (which was 300 in most experiments).

$$\begin{aligned} h &= \mathrm{ReLU}(W_1 q + b_1) \\ q' &= W_2 h + b_2 \end{aligned}$$

For the advanced transformation, we attempt to learn a vector containing information independent of both questions and paragraphs, but useful for interpreting either. We call this vector the knowledge vector. The knowledge vector should represent information about how a question and associated paragraph might look. In order to also have information about what kind of questions might belong to a paragraph, we attempt to predict the question encoding from the paragraph as well as vice versa. Using the knowledge vector as a common input, we apply the simple transformation to both the paragraph encoding and the question encoding, attempting to predict the other. The regular loss term for the network is the cosine difference between the predicted question encoding and the true paragraph encoding. For the advanced transformation, we introduce a second loss term: The cosine difference between the predicted paragraph encoding and the true question encoding.

$$\begin{aligned} h_q &= \mathrm{ReLU}(W_1[q;k] + b_1) & \qquad h_p &= \mathrm{ReLU}(W_3[p;k] + b_3) \\ q' &= W_2 h_q + b_2 & \qquad p' &= W_4 h_p + b_4 \end{aligned}$$

As with the simple transformation, the hidden layer size for the classifier is 500, the knowledge vector has 1000 dimensions.

5.1.3 Word Embeddings

Most experiments were performed using pre-trained 300-dimensional GloVe embeddings [27] trained on a 840 billion token common web crawl corpus. In some models, we also try training an embedding based on byte-pair encoding [33], obtained by the implementation of Github user Rico Sennrich[1]. The primary advantage of the subword encodings is a drastically reduced vocabulary size (~360 000 used out of 2.2M total for GloVe vs. ~41 000 for bpe). However, despite the lower vocabulary size, the amount of data in the training set may not be sufficient to achieve better results than using the pre-trained embeddings.

5.2 IR Baseline

In order to have a baseline model to compare our models against, we use traditional IR methods to retrieve paragraphs. Specifically, we use elasticsearch[2] and TF-IDF (see section 2.2.1). Both have been used as baselines before [5, 44] and are well-established information retrieval methods. We evaluate both methods on the testing set (derived from the development set of SQuAD, see chapter 4), and our paragraph corpus.

For elasticsearch, we construct an index containing all paragraphs as documents, then query it using the question as a simple query string. The “snowball” stemming analyzer is used for both questions and paragraphs. We evaluate retrieval recall on both the exact match (retrieved paragraph is equal to the label) and answer match (retrieved paragraph contains a correct answer) metrics when retrieving between 1 and 1000 paragraphs.

For TF-IDF, the document frequencies are trained on the paragraphs, then all paragraphs and all questions are encoded. Following Chen et al. [5], we use bigram counts and the `murmurhash` hashing scheme [45] with 2^{24} features. For prediction, the paragraph with the highest dot product similarity to the question is chosen.

5.3 Random Baseline

To ascertain that models that don’t outperform TF-IDF are at least adapting to the task at all, we also establish a baseline by randomly selecting a paragraph as best match for each question. Outperforming this baseline is the lowest hurdle that a model must pass to prove it has learned anything about this task. As for the information retrieval baselines, we retrieve between 1 and 1000 paragraphs. In addition to revealing which models have not adapted to the task, the random baseline can provide information on the dataset itself: The odds of finding the exact paragraph specified in a question’s label are approximately $\frac{1}{1\,000\,000}$. If we observe a higher accuracy for the answer match metric, this would indicate that some questions have answers that occur in a large number of paragraphs.

[1] `https://github.com/rsennrich/subword-nmt`

[2] `https://www.elastic.co/`

5.4 Encoders

In each model, the same encoder function is trained to encode both paragraphs and questions. The goal is that the encoder produces a similar result for questions and paragraphs that are concerned with the same subject matter, allowing the system to quickly select the most relevant paragraph.

5.4.1 SWEM

The most basic encoder is a Simple Word Embedding-based Model (SWEM) [35]. In particular, $swem_{hier}$ proved effective. It works by average pooling word embeddings within a sliding window along the time dimension, then max pooling the results. Or more formally: Let v_i be the word embedding at position i in the input sentence. It is itself a vector, so let $v_{i,j}$ be the j-th position of that vector. Then the encoding is calculated as follows:

$$\begin{aligned} \text{avg}_{i,j} &= \frac{1}{w} \sum_{k=i}^{i+w} v_{k,j} \\ \text{swem}_j &= \max_i \text{avg}_{i,j} \end{aligned} \tag{5.4}$$

Where w is the sliding window size, which was 5 for all experiments. While this is a parameterless model, it can be used to train word embeddings. However, models working with pre-trained GLoVe embeddings achieved better results.

While the authors reported impressive performance for sentence classification, it is unclear how well this model will perform without a classifier (and therefore practically no learnable parameters). For this reason, we also test a version of SWEM with an additional linear projection layer on top. The projection does not change the dimensionality of the output, it is simply there to perform an operation on the sentence embedding.

5.4.2 LSTM

We apply a single-layer bidirectional LSTM (long-short-term-memory, [14]) network to the word embeddings, using the final output as the sentence embedding. LSTMs have been widely explored for use as encoders in question answering, for example in Tay, Tuan, and Hui [37] and Chen et al. [5] and have shown good performance even without further layers. We experiment with many different configurations on training data and hyperparameters.

Because the encoder must be able to encode paragraphs and questions independently, we cannot make use of an attention mechanism, which would require knowing both a question and paragraph encoding. We therefore opt for the simpler method of using the final output as the encoding.

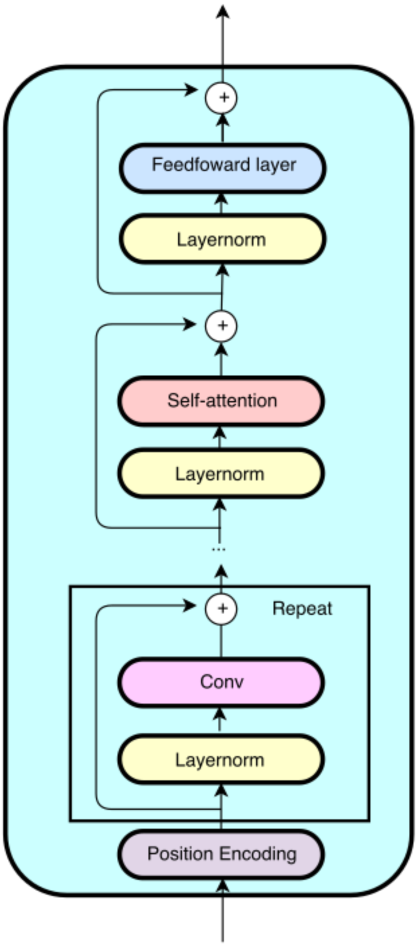

Figure 5.1: Schematic showing the QANet encoder. From Yu et al. [52].

The models were implemented in Tensorflow[3] using both the generic implementation (`LSTMBlockCell`) and the CUDNN versions which are optimized for Nvidia GPUs. At the time of implementation, there were concerns in the machine learning community about RNN performance in Tensorflow, so we also tested a comparison implementation in PyTorch[4]. The information on Tensorflow RNN performance appeared to be outdated, as Tensorflow outperformed PyTorch in both execution speed and memory efficiency, particularly when using the CUDNN-specific implementations of LSTM provided by Tensorflow (which were not available through PyTorch).

5.4.2.1 QANet

As a question answering system, QANet must encode relevant information from both the question and context into an internal representation. Normally, QANet would go on to run a classifier to predict a substring of the paragraph that answers the

[3] `https://www.tensorflow.org`
[4] `https://pytorch.org/`

question. We use only the encoder portion of QANet, which is a series of convolutional operations and self-attention (see figure 5.1). As QANet encodes an input sequence into another sequence of equal length, we must also experiment with aggregation methods to reduce the dimensionality of the output and their effect on performance.

Additionally, we test the encoder portion of QANet, as pre-trained on its original question answering task.

5.5 Classifier

As noted above, running a classifier on the full set of paragraphs for each question is not feasible for a useful question answering system. However, a classifier could still be run on a subset of paragraphs from the full set in order to refine the selection. Other models [e. g. 44] use a somewhat similar approach, using an information retrieval system to narrow the search space of paragraphs to 100-200 paragraphs, then running the question answering system on all of them and finally aggregating the most likely answer spans from across all paragraphs. Our system differs in that the question answering system is only run on one paragraph, as predicted by the classifier. Using this method, we are able to achieve state-of-the-art results on the SQuAD dataset.

The advantage of a classifier is that in addition to having the encoding of both the paragraph and the question, it is able to perform more advanced computations on top of both, extracting more meaningful information. In our model the best performance was achieved when the first such computation was an attention mechanism, described in section 2.1.3.2. In additon to both variants of the attention mechanism described there, we compare the performance of this model against one with no attention mechanism at all. We evaluate all the encoders of the classifier-free model, as well as a simple sliding window average (like SWEM but without the final max-pooling step). Omitting the max pooling allows the model to make use of an attention mechanism.

The classifier follows Htut, Bowman, and Cho [15]: One ReLU (size 500) layer and one linear layer (size 1) to produce an unscaled output value for each paragraph. Since one example consists of one correct and several incorrect paragraphs, we treat this as a classification task, taking the softmax over all paragraph outputs which yields a probability distribution over all considered paragraphs. We use the crossentropy loss function for all experiments.

6 Results

In this chapter, we report the performance of the models described in chapter 5, starting with an evaluation of the dataset itself.

6.1 Evaluating the Dataset

We evaluate models by two metrics: The exact match metric (EM) considers a retrieved paragraph correct if and only if it matches the original context paragraph from the labeled dataset. The answer match (AM) metric also considers paragraphs that contain one or more of the answer strings as correct. There is only one correct paragraph for the exact match metric, but there may be several for the answer match metric. In order to quantify just how easy or difficult it is to find a paragraph to satisfy the answer match metric, for each question, we test how many paragraphs contain a correct answer.

Because this is a computationally expensive task that requires processing all pairs of questions and paragraphs, we perform this test on a reduced dataset of only 50 000 paragraphs. These were selected by including the correct paragraph for each question (1979 paragraphs for 9931 questions), then filling the remainder with random paragraphs from the 1 million set.

Figure 6.1 shows that about 45% of questions have only a single paragraph that contains an answer. For almost 70% of paragraphs, the answer can be found in only 10 or fewer paragraphs. It is also visible that there is some number of questions for which almost any paragraph contains a correct answer. This is due to dataset noise in SQuAD: For example, one question (“Why did the Shah of Iran give an interview?”) erroneously has “.” (a single period) listed as one of its correct answers. A period is found in 49 430 out of 50 000 paragraphs, meaning they are all recognized as correct paragraphs. Questions whose answer is a year number are also prone to having the answer appear in many unrelated paragraphs.

We conclude that the answer match metric is most likely appropriate for most questions, but has some issues with noisiness. For this reason, we cannot use it as the sole measure of performance of the information retrieval system. We will therefore report all results in both this metric as well as the more difficult exact match metric. For the cross-evaluation on QUASAR-T, we have no other option than to use answer match, because the dataset is not labelled appropriately to use exact match.

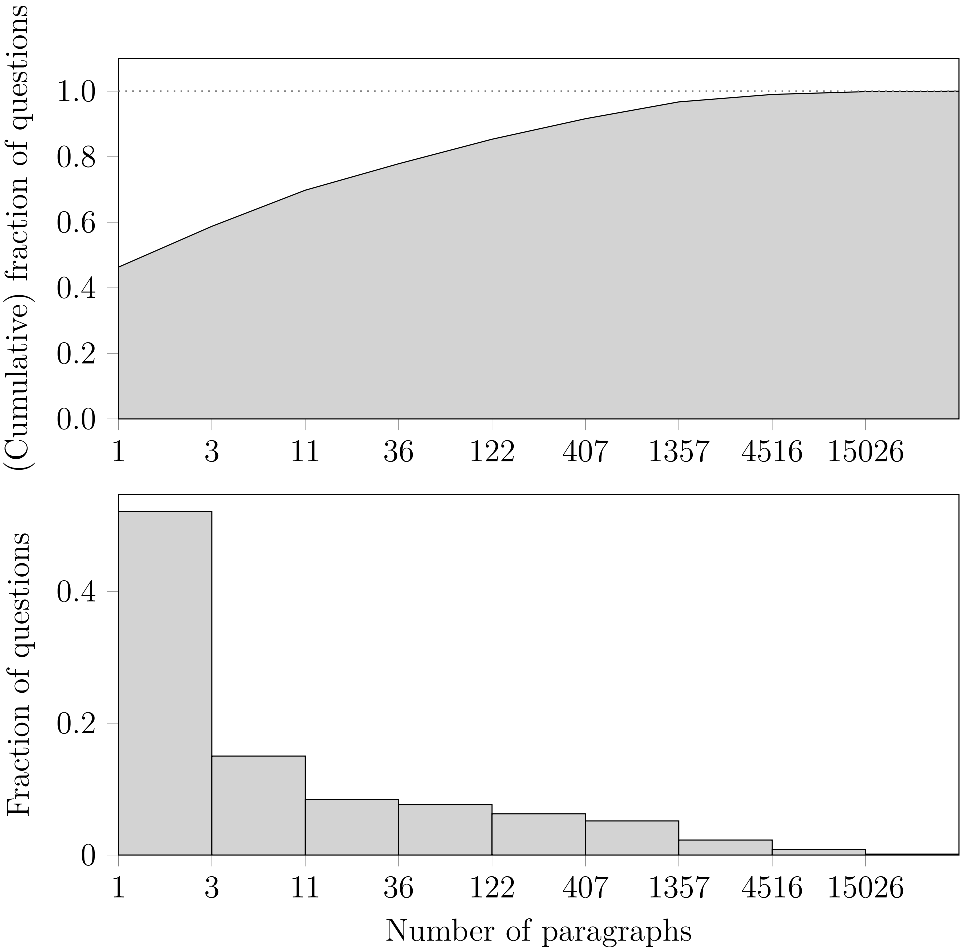

Figure 6.1: Number of “correct” paragraphs per question visualized.

6.2 Baselines

We test two information retrieval baselines: TF-IDF and elasticsearch. Both are tested on their retrieval performance for a different retrieval volumes in both exact match and answer match. TF-IDF performs better than elasticsearch, providing a very strong baseline, as well as satisfying the demand for a fast evaluation even on a large paragraph set, making it feasible as a component in a hypothetical real-time question answering system.

We additionally evaluate a random selection as an additional baseline. We do not expect this method to outperform either of the information retrieval baselines, it is here to provide an absolute worst-case performance indicator. Any system that is suitable for this problem at all must outperform at least the random baseline. Additionally, the random baseline is another way to measure dataset noise, such as questions whose answer appears in many paragraphs.

In table 6.1 we report the performance for each metric and model when retrieving 1, 10, 100 or 500 paragraphs for the 9 931 questions of our testing set. As our TF-IDF implementation follows Chen et al. [5], our results are consistent with theirs: TF-IDF

Encoder	**Exact Match**				**Answer Match**			
	top-1	10	100	500	top-1	10	100	500
TF-IDF	**39.9**	**63.5**	**78.4**	**82.9**	**47.4**	**75.2**	**89.9**	**93.7**
elasticsearch	9.7	24.7	42.9	58.3	12.1	31.3	54.8	71.3
random	0.0	0.0	0.0	0.1	0.3	2.2	6.8	19.7

Table 6.1: Baseline information retrieval results

outperforms elasticsearch by a large margin, and when selecting 100 paragraphs, the TF-IDF model makes a very good pre-selection which we can later refine. As these results are gathered on the reduced dataset of 10 000 articles, they are somewhat better than those reported in Chen et al. [5], however, see section 6.6 for an application of TF-IDF to the full Wikipedia dataset. At 47.4% answer match score for the top result, TF-IDF provides a very strong baseline, outperforming all the non-classifier models presented in the following sections, and the 89.9% score on the top 100 results will provide a good basis for the classifier model, the results of which we present in section 6.5.

The random baseline results are interesting: The exact match scores are consistent with the expected value of retrieving one paragraph out of one million (expected performance for 500 is 0.05%, which would be rounded up to 0.1%), but in the answer match metric we see that there are a certain number of questions that are “easy” to answer, typically because the answer is a common phrase occuring in many (unrelated) paragraphs. Seeing this result is precisely why we performed the experiment with the random baseline, as otherwise an answer match score of 2.2% on 10 paragraphs might be considered a positive result for a model, when in fact this is merely equal to the performance of random selection.

6.3 Individual Training and Mean Square Error

For the first set of experiments, we train an encoder on question-paragraph pairs. The cosine difference of the question and paragraph encoding is the prediction of the network (a result close to 1.0 indicating a high confidence that the paragraph can answer the question). The training signal is 1.0 for the correct paragraph and 0.0 otherwise. The loss is mean square error, the models were trained in minibatches of 32 examples and optimized using the Adam Optimizer [20] with a learning rate of 0.001.

Besides testing two different encoders, the main revelation of this set of experiments was the importance of the ratio of positive and negative training examples. At first, we chose a 1:1 ratio (i. e. each question appears twice in the training set—once with the correct paragraph and once with a random incorrect one). The lower section of table 6.2 shows the performance of models trained with a ratio of 1:31 (31 incorrect results to one positive one). Using this unbalanced distribution, we more closely model the true data distribution of the testing set (which has hundreds of thousands of incorrect paragragraphs to one correct one). However, this training scheme makes

Encoder	**Exact Match**			**Answer Match**		
	top-1	10	100	top-1	10	100
TF-IDF	**39.9**	**63.5**	**78.4**	**47.4**	**75.2**	**89.9**
random	0.0	0.0	0.0	0.3	2.2	6.8
LSTM	0.0	0.0	0.0	0.5	2.7	11.0
SWEM-transform				0.4	3.4	14.1
LSTM	0.4	2.6	9.5	3.1	14.1	35.1
SWEM-transform	0.7	3.7	14.0	4.3	18.7	43.2

Table 6.2: Mean square error training results

no effort to avoid biasing the classifier towards rejecting a paragraph (because that tends to be the correct choice more often).

We arrived at the ratio of 1:31 after first trying less unbalanced ratios like 1:5, which yielded continuously better results as the ratio increased and extremely unbalanced ratios like 1:100, which yielded no improvement whatsoever. It became clear that it was important that each minibatch contains at least one positive example—otherwise the encoders will converge on producing zero for all outputs. Therefore a ratio of 1 to (batchsize −1) is the most extreme ratio that still has one positive example in each minibatch. This is an expected value, not a guarantee, as the training set is shuffled before training.

The two strongest models, which are reported here, were the single-layer bidirectional LSTM and SWEM with an additional linear layer on top (we call this model SWEM-transform). Additonal LSTM layers were not found to yield improvement. With all encoders, the encodings for all paragraphs were pre-calculated during testing. Retrieving the paragraph consists only of encoding the question and a matrix multiplication of the question encoding with all pararaph encodings.

Another notable result is the discrepancy between the exact match and the answer match metric. Both trained models have a much larger difference (both relative and absolute) between the two metrics than TF-IDF. This may indicate that the improvement we observe after introducing more negative training results affect primarily the “easy” questions, i. e. ones where the answer appears in a large number of paragraphs.

6.4 Training with Margin Ranking Loss

After achieving positive results when training with several negative examples to each positive one, the adoption of the margin ranking loss seemed like the next logical step. A single training example now consists of a question, the correct paragraph and some number of false ones, similar to the improved setup above. The difference here is that there is guaranteed to be a correct example in each example and that the loss function takes into account the results from all paragraphs that are paired with the same question. Initially, a ratio of 1 to 5 of correct to incorrect paragraphs was chosen. These examples were further processed in minibatches of size 5 (meaning 5 questions and 30 paragraphs were processed in each minibatch).

Encoder	Exact Match			Answer Match		
	top-1	10	100	top-1	10	100
TF-IDF	**39.9**	**63.5**	**78.4**	**47.4**	**75.2**	**89.9**
random	0.0	0.0	0.0	0.3	2.2	6.8
SWEM-transform	1.7	6.0	14.5	3.5	13.3	32.1
LSTM	0.6	1.9	6.1	2.7	12.0	31.5
QANet	0.2	0.9	4.8	1.9	10.4	28.5

Table 6.3: Best margin ranking loss results

The reported performances for each model are for the best parameter configuration for each respective model. All models adapted to the task, i. e. they exceeded the performance of the random baseline, but none outperformed TF-IDF or even the same model trained with the mean square error. The model performance varies greatly depending on the exact hyperparameters used, so we conducted experiments testing verious different hyperparameter configurations.

6.4.1 Hyperparameter Searching

There are a number of potentially relevant hyperparameters for each model that might affect its performance. The examined parameters include different similarity functions (squared cosine similarity and weighted dot product), adding a non-linear transformation to the encoder for the question only (attempting to predict the paragraph encoding from the question encoding) and training byte-pair encoded word embeddings instead of using pre-trained GloVe embeddings. Additionally, some model-specific hyperparameters were tested.

6.4.1.1 LSTM

LSTM is the base system for many of these experiments, as it is reasonably simple and relatively fast to train (the LSTM models presented here usually take about one hour to train on an Nvidia Tesla K80). First, we establish what the basic model should look like by determining whether to use the hidden state as sentence encoding, the cell state or a concatenation of both.

Encoder	Exact Match			Answer Match		
	top-1	10	100	top-1	10	100
LSTM-c	0.2	1.1	4.4	1.9	10.9	28.7
LSTM-h	**0.6**	**1.9**	**6.1**	**2.7**	**12.0**	**31.5**
LSTM-both	0.1	0.3	2.3	1.6	8.7	25.8

Table 6.4: LSTM output type comparison

Using the hidden state proved best, so it will be used for all future experiments. However, the margins are fairly small, so it is possible that the difference are due to random chance resulting from the shuffled samples in the training set. Using the hidden state is reasonable, however, as it makes use of all the LSTM parameters. Curiously, using both states does not yield an improvement over either single state.

Another hyperparameter to test is the number of units. TF-IDF has a considerably larger number of features (2^{24} as opposed to 300) compared to the previous LSTM models, so it is possible that its greater accuracy comes from this increased representation capacity. We compare three different sizes of LSTM networks in order to reveal a correlation between evaluation accuracy and encoding size.

Encoder	**Exact Match**			**Answer Match**		
	top-1	10	100	top-1	10	100
LSTM-160	0.3	1.0	4.9	2.1	10.4	29.8
LSTM-300	**0.6**	**1.9**	**6.1**	**2.7**	**12.0**	**31.5**
LSTM-600	0.3	1.4	6.0	2.5	11.8	30.8

Table 6.5: LSTM encoding size comparison

From the results in table 6.5, a clear benefit to a larger encoding cannot be established. The differences in performance for these three sizes can be the result of random variation resulting from the shuffled training set and random initialization.

6.4.1.2 QANet

Using a series of mostly convolutional operations, QANet produces a representation of the input that still has a sequence length dimension. In order to obtain a fixed-length representation, we must compare different methods of accumulating the results into a single vector. We compare taking the maximum and the mean along the sequence dimension, as well as using SWEM on the QANet output.

Encoder	**Exact Match**			**Answer Match**		
	top-1	10	100	top-1	10	100
QANet-max	**0.1**	**0.5**	**2.1**	**1.8**	**9.2**	**25.6**
QANet-mean	0.1	0.8	3.2	1.6	9.1	26.6
QANet-swem	0.0	0.0	0.2	1.0	6.6	19.9

Table 6.6: QANet aggregation methods

Similar to the LSTM above, we investigate the effect of encoding size. By default, QANet usses a fairly small encoding size of 96, following Yu et al. [52]. This is sufficient, as the original purpose of QANet is a classification task making use of an attention mechanism. QANet preserves the original sequence length with an encoding size of 96 per sequence step, effectively increasing the number of features drastically.

As this model does not use an attention mechanism, the encoding size is effectively much smaller than that of LSTM. Increasing this paramater may allow the model to differentiate encodings more effectively.

Encoder	**Exact Match**			**Answer Match**		
	top-1	10	100	top-1	10	100
QANet-96	0.1	0.8	3.2	1.6	9.1	26.6
QANet-192	**0.2**	**0.9**	**4.8**	**1.9**	**10.4**	**28.5**
QANet-288	0.0	0.2	1.5	1.5	8.6	24.3

Table 6.7: QANet encoding size comparison

6.4.1.3 Transformations

By applying a transformation to the question encoding only, we hope to predict the matching paragraph encoding more accurately. We compare models with no transformation, the simple (one ReLU and one linear layer) and the advanced transformation.

Encoder	**Exact Match**			**Answer Match**		
	top-1	10	100	top-1	10	100
LSTM-None	**0.6**	**1.9**	**6.1**	**2.7**	**12.0**	**31.5**
LSTM-Simple	0.0	0.1	1.0	1.5	7.6	23.1
LSTM-Advanced	0.0	0.0	0.1	1.0	5.9	19.0

Table 6.8: Effect of transformations

Not only do the transformations not help, but they actively degrade the performance of the model. Consequently, the other tests do not use the transformations.

6.5 Classifier

The results of only training and comparing an encoding for questions and paragraphs without the use of a classifier were unsatisfactory. At the same time, using a classifier on the same task as the above models is infeasible due to the high number of paragraphs. Therefore, as the final model, we retrieve a sample of paragraphs for each question using TF-IDF and then apply a classifier on this drastically reduced subset of paragraphs. Using a classifier further opens up the possibility of using an attention mechanism, making better use of the input encodings. The use of a classifier dramatically improved retrieval accuracy and the models were able to exceed the performance of single-result TF-IDF.

For testing, the model is presented with the top 100 paragraphs as retrieved by TF-IDF. If the correct paragraph is not part of the top results of TF-IDF, it is added to the testing set as well, replacing the lowest rated one. The model then ranks the paragraphs independently and returns the highest rated one. We report a single and a compound score: The single score is simply the model's performance on this testing set. The compound score takes into account the accuracy of TF-IDF and counts only positive results on samples where TF-IDF also returned a correct result. Effectively, the compound score is bounded by the score of TF-IDF (78% exact match or 90%

Encoder	**Exact Match**		**Answer Match**	
	single	compound	single	compound
SWEM	9.2	8.6	15.3	14.9
Pooling	38.2	35.8	44.8	43.1
LSTM	**68.3**	**63.2**	**73.6**	**69.6**
QANet	23.2	22.1	29.6	28.9

Table 6.9: Best classifier results

answer match for 100 results). Samples where TF-IDF did not return a correct result are counted as an automatic failure.

The results are much better than before, not only exceeding the performance of TF-IDF on one result (which was 47.4% answer match), but also promising a solid pre-selection for a question answering system based on the results.

One of the main advantages of the classifier model is the ability to make use of an attention mechanism. In addition to the trilinear attention, we evaluate DCN attention, as well as no attention, to quantify just how much the attention mechanism adds to the model's performance. We examine the effect of an attention mechanism primarily on the LSTM encoder, but also on the simple word embedding based models, Pooling and SWEM, which differ only in that Pooling uses trilinear attention whereas SWEM uses a max operation to obtain a fixed-length encoding.

Encoder	**Exact Match**		**Answer Match**	
	single	compound	single	compound
LSTM-trilinear	**58.3**	**54.7**	**65.6**	**63.0**
LSTM-DCN	56.7	53.5	64.1	61.7
LSTM-no-attention	0.9	0.8	4.7	4.6
Pooling	29.9	28.2	36.1	34.8
SWEM	9.2	8.6	15.3	14.9

Table 6.10: Comparing attention mechanisms
All models were trained on a ratio of 10:1

From table 6.10, it is clear that the attention mechanism is crucially important to the task, with the attention-free model barely exceeding random performance. On the tested ratio of 10:1 negative to positive results, trilinear attention performed best, so it is the default for the following experiments. In addition, the next experiment shows little to no performance improvement of the DNC attention model when trained with higher ratios, whereas trilinear improves significantly.

Another factor we found to be important for the model's performance was the ratio of correct to incorrect training examples, as well as whether the training examples were random, or the top results of TF-IDF. We found that all models improve when trained on a higher ratio of incorrect examples to correct ones and further improve by using TF-IDF-selected training examples.

Encoder	**Ratio**	**Exact Match**		**Answer Match**	
		single	compound	single	compound
LSTM	5	56.3	53.0	62.9	60.5
LSTM	10	58.3	54.7	65.6	63.0
LSTM	t10	55.1	51.7	62.4	59.8
LSTM	20	57.9	54.4	64.6	62.1
LSTM	50	59.2	55.5	66.3	63.6
LSTM	100	60.2	56.6	67.2	64.5
LSTM	**t100**	**68.3**	**63.2**	**73.6**	**69.6**
LSTM-DCN	10	56.6	53.3	64.1	61.7
LSTM-DCN	20	56.7	53.5	64.1	61.7
Pooling	10	29.9	28.2	36.1	34.8
Pooling	t100	38.2	35.8	44.8	43.1

Table 6.11: Comparing attention mechanisms
When the ratio is marked with t, the model was trained on TF-IDF results.

The total performance of the model is a compound of the performance of TF-IDF and its own. If we retrieve fewer results with TF-IDF, the task for the neural model becomes easier (choosing the correct paragraph from 10 is easier than from 100), but the upper bound for the performance is lower. We therefore investigate the relationship between TF-IDF performance and the neural network performance on different retrieval sizes, attempting to find the one that maximises the compound score.

# Results	**5**		**10**		**20**	
Encoder	single	compound	single	compound	single	compound
TF-IDF		68.5		75.3		80.9
SWEM	43.4	33.8	32.5	27.5	25.5	23.1
Pooling	65.0	51.4	58.7	50.5	53.5	48.6
LSTM	**84.0**	**62.1**	**80.8**	**65.3**	**78.2**	**67.8**
QANet	54.2	44.3	46.0	40.2	39.6	36.6

# Results	**50**		**100**		**200**	
Encoder	single	compound	single	compound	single	compound
TF-IDF		86.4		89.9		92.0
SWEM	18.7	17.7	15.3	14.9	13.0	12.7
Pooling	48.1	45.4	44.8	43.1	42.4	41.0
LSTM	**75.3**	**69.0**	**73.6**	**69.6**	**72.5**	**69.4**
QANet	33.7	32.3	29.6	28.9	26.7	26.2

Table 6.12: Different retrieval size results (answer match only)

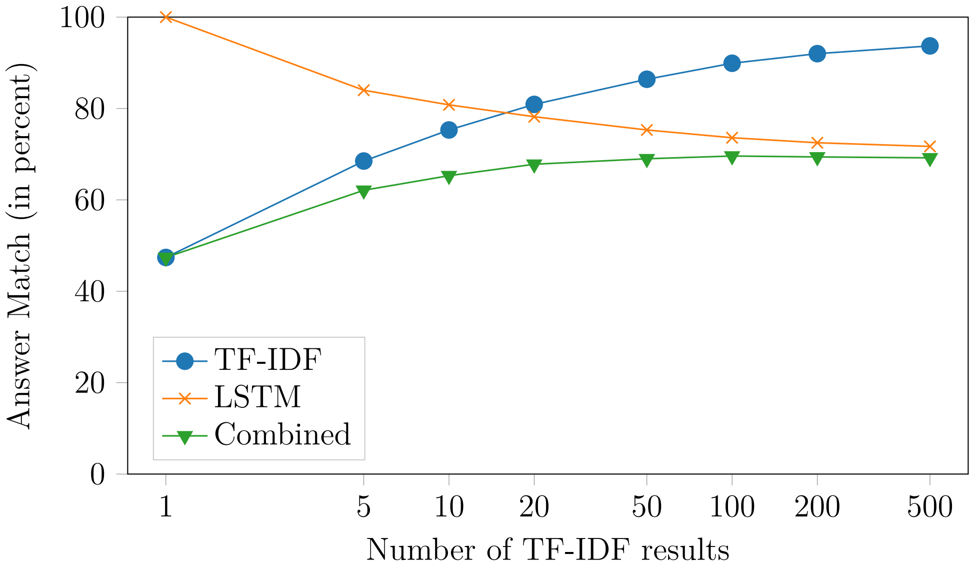

Figure 6.2: Answer match accuracy for different retrieval sizes

After testing six different retrieval sizes, we have determined the optimal number of results to be 100, with the compound model achieving an answer match score of 69.6%

6.6 Full Stack

For the final set of experiments, we combine the TF-IDF pre-selection, the LSTM classifier selection and QANet question answering into one open-domain question answering system. This system is now capable of answering open-domain questions as defined in section 3.1.1.2. Given only a question, the TF-IDF system retrieves the top 100 paragraphs from the database, then passes them to the classifier, which selects one paragraph to use as context for QANet. The components for this model are trained independently—for the classifier, we use the best-performing model from above, for QANet, we use the implementation of Github user NLPLearn[1], pre-trained on SQuAD. On the regular SQuAD dataset, QANet achieves a performance of 70.4% exact match and 79.6% F1, though we expect the performance on our dataset to be significantly worse, as the task is more difficult.

Each step of the process imposes an upper bound on the performance of the next, so we report both the cumulative performance at each step, as well as estimating the performance of each step individually (obtained by simply dividing the performances after and before the step). For TF-IDF and the LSTM classifier, we use the exact match and answer match metrics from above. For QANet, we use the metrics used for evaluating SQuAD: Exact match is true only if the predicted answer exactly matches the label, F1 calculates the precision and recall of words in the answer and

[1] https://github.com/NLPLearn/QANet

calculates the F1 score from these, so it can take into account partial answers or answers containing additional words.

Step	**Answer Match**	**Exact Match**	**F1**	**Relative (approx.)**
TF-IDF	89.9	78.2		
LSTM	65.4	56.2		72.0
QANet		44.6	52.6	79.7
Logistic Regresstion[29]		40.0	51.0	

Table 6.13: Full stack results on SQuAD using a part of Wikipedia

Even though we use our reduced Wikipedia corpus as a knowledge source rather than using the provided contexts from SQuAD, we exceed the performance of the logistic regression baseline from the original SQuAD paper, which does use the provided context.

In order to be able to compare our results to other literature, we also conduct the experiment using all of Wikipedia as a knowledge source. This makes the task considerably harder both for TF-IDF as well as for our classifier, as we are able to retrieve much more highly topical candidates for a question, making the decision of which one answers the question more difficult.

Step	**Answer Match**	**Exact Match**	**F1**	**Relative (approx.)**
TF-IDF	79.6	58.9		
LSTM	50.0	37.0		62.5
QANet		34.3	41.5	86.5

Table 6.14: Full stack results on SQuAD using all of Wikipedia

Finally, we compare our results to other literature, in particular Chen et al. [5] and Wang et al. [44]. In order to have a more complete comparison, we also evaluate on QUASAR-T [6], TriviaQA [17], WikiMovies [7], WebQuestions[2] and CuratedTREC[3]. On SQuAD and TriviaQA, the tests were performed on the development set, as the testing sets are not publically available. On all other datasets, we report the performance on the testing set.

We only compare to YodaQA and DrQA-MTL indirectly, as they make use of additional data—YodaQA uses an additional knowledge base and DrQA-MTL is trained on multiple datasets. While our model performs best on SQuAD, DrQA and R^3 show better generalization to other datasets. On the evaluation machine, which is a CPU-only system using 10 1.4GHz CPUs, each question took about 1.6s to answer. This figure could likely be improved significantly by using a GPU and grouping questions together into minibatches, but it is still acceptable.

[2] `https://nlp.stanford.edu/software/sempre/`

[3] `https://github.com/brmson/dataset-factoid-curated/tree/master/trec`

Model	SQuAD		WikiMovies		WebQuestions		CuratedTREC	
	EM	F1	EM	F1	EM	F1	EM	F1
DrQA[5]	28.4	—	34.3	—	**19.5**	—	25.7	—
R^3[44]	29.1	37.5	**38.8**	**39.9**	17.1	**24.6**	**28.4**	**34.3**
Ours	**34.3**	**41.5**	6.4	10.1	10.4	13.1	10.2	16.4
YodaQA[4]	—	—	—	—	39.8	—	31.3	—
DrQA-MTL[5]	29.8	—	36.5	—	20.7	—	25.4	—

Table 6.15: Comparison on different datasets and different models
All datasets use the same model trained on SQuAD.

We consider the two trivia datasets, QUASAR-T and TriviaQA particularly interesting benchmarks because they consist of questions written by humans both independent of a context and with the intention to be challenging questions. Surprisingly, even though the datasets are superficially similar, the results vary drastically on the two datasets.

Step	No contexts			Short Contexts		
	Answer Match	Exact Match	F1	Answer Match	Exact Match	F1
TF-IDF/Contexts	60.7	n/a		67.1	n/a	
LSTM	24.4	n/a		27.3	n/a	
QANet		10.4	14.6		11.5	18.3
R^3					34.2	40.9

Table 6.16: Full stack results on QUASAR-T

QUASAR-T poses a number of difficult problems for the model: It contains some cloze-style questions ("The female marine catfish hatches eggs in her ______"), which the model has not been trained to handle. Even the provided contexts do not always contain the exact answer string, as QUASAR-T expects answers in free-form. The biggest bottleneck in this setting is the LSTM classifier which is particularly struggling on this task.

Step	TriviaQA			TriviaQA verified		
	Answer Match	Exact Match	F1	Answer Match	Exact Match	F1
TF-IDF	82.7	n/a		94.3	n/a	
LSTM	42.9	n/a		57.9	n/a	
QANet		26.1	33.8		26.4	35.4

Table 6.17: Full stack results on TriviaQA

TriviaQA on the other hand is much more lenient in what it allows for each answer, providing synonyms and multiple alternatives, so that TF-IDF can usually find relevant contexts. Nevertheless, the fact that the questions are worded indepently of the context and therefore different from SQuAD shows in the LSTM classifier performance, which is worse here than on SQuAD. The model still generalizes well to TriviaQA, showing its ability to answer realistic questions in an open-domain setting. To the best of the author's knowledge, no comparable results have been published for TriviaQA, so our system will stand as a baseline.

6.6.1 Retrieval Evaluation

In addition to the automatic results above, we would like to show examples where our classifier model has used its understanding of the question to produce better results than TF-IDF, as well as examples where it failed. In the below examples, we show the top result retrieved by TF-IDF and the result (picked from the top 100 TF-IDF results) that was chosen by the classifier, highlighting the answer given in bold.

Question: How did Alan Turing die?

TF-IDF

The film starts in October 1952 after Alan Turing (Ed Stoppard) has been convicted. He is talking to his psychiatrist, Dr. Franz Greenbaum (Henry Goodman). Dr. Greenbaum and Alan continue to discuss; Alan informs Dr. Greenbaum that he cannot talk about his war time activities. Dr. Greenbaum informs him that he can talk about anything he wants. Sir Dermot Turing, nephew of Alan, is shown and he goes on to explain how life was for John Turing (Alan's brother) and Alan Turing during their childhood. David Leavitt appears next and talks about Turing's school time activities. David further explains that Turing was good at mathematics and athletics. His favourite sport was running.

LSTM

Lobban spoke of his regret over the treatment of cryptographer Alan Turing in October 2012. Turing, who committed **suicide** after being convicted of homosexuality, was described by Lobban as a "national asset" and said that more people like Turing were needed to face contemporary information security threats.

This is a typical example for how the classifier improves upon the TF-IDF algorithm: The first paragraph frequently contains both "Turing" and "Alan Turing", both comparatively rare phrases and therefore rated highly by TF-IDF. As a result, the top paragraph (with the highest similarity according to TF-IDF) is one containing these phrases multiple times but with otherwise no bearing on the question. The classifier, which is able to extract more semantic information from the question is able to select a more relevant paragraph, even though the second paragraph does not contain the word "die" or similar words.

Nevertheless, the system is still sensitive to the wording of the question, as illustrated in the following example: The tokenizer splits "sun's" into two tokens and TF-IDF only takes into account bigrams, so it does not correlate "sun" and "core" and instead searches for them independently. While TF-IDF does retrieve at least some paragraphs about the sun for this query, the classifier looks for paragraphs that contain temperatures and thus ends up with a paragraph that is not about the sun at all. Changing the wording of the question causes the retrieved paragraphs to become more relevant, though it still takes the classifier refinement to retrieve the correct paragraph.

Question: What is the temperature at the sun's core?
Context
The annual range of monthly mean temperatures—that is to say, the difference between the warmest and coldest months—at Andagoya is mere 0.7 °C (approximately 1.3 °F) as April, the warmest month, has an average temperature of 27.7 °C (81.9 °F) while November, the "coldest" month, averages 27.0 °C (80.6 °F); the average annual temperature is **27.4 °C** (81.3 °F). This near complete uniformity of temperature is caused by the fact that Andagoya, at a latitude of 5°5' North, lies extremely close to the thermal equator, [...]
Question What is the core temperature of the sun?
Context
Stars actually have a temperature gradient, going from their central core up to the atmosphere. The "core temperature" of the sun—the temperature at the centre of the sun where nuclear reactions take place—is estimated to be **15 000 000 K**.

In the final example, TF-IDF once again selected a paragraph with the (uncommon) phrase "Eiffel Tower" occuring multiple times. The classifier was at least able to choose a paragraph that includes height information, but has as little information as TF-IDF does about what Eiffel Tower is being referred to.

Question: How high is the Eiffel Tower?
TF-IDF
A semantic difficulty may arise when considering reference in representationalism. If a person says "I see the Eiffel Tower" at a time when they are indeed looking at the Eiffel Tower, to what does the term "Eiffel Tower" refer? The direct realist might say that in the representational account people do not really see the tower but rather 'see' the representation. However, this is a distortion of the meaning of the word see which the representationalist does not imply. [...]
LSTM
The fair celebrated that country and the South Seas. Exhibitions included "New Zealand's Eiffel Tower", a **40 metre** high wooden structure based on the Eiffel Tower built by the Austral Otis Elevator Company and used to display their products. The tower was estimated to cost £1200 and included an elevator that travelled about 30 m. A smaller Eiffel Tower, without an elevator, was situated in an adjacent garden area, near the internal courtyard of the exhibition.

7 Conclusion

7.1 Review

The goal of this work was to create a neural information retrieval system that could allow a traditional question answering system to work in an open-domain setting. Given only a question and a large text corpus, the system would find a single paragraph that is the most relevant to the question, creating a problem instance suitable for the question answering system. We outline the problem in section 3.1, describe our approaches in chapter 5 and show our results in chapter 6.

The first tested model refrained from using a neural classifier, instead only calculating an encoding for each paragraph and each question, then determining the most similar paragraphs by cosine similarity. Sections 6.3 and 6.4 show that the performance of this model did not exceed even that of the baseline. We speculate on the reason for this below. We did, however, identify a key aspect that led to a significant improvement in performance: Training the model with more incorrect than correct training examples. In section 6.4.1, we tested a large number of hyperparameter options in order to identify other key performance factors.

We evaluate the retrieval accuracy of our second model in section 6.5, identifying the attention mechanism as a second major improvement. Introducing the attention mechanism allowed the model to select an appropriate paragraph from a pre-selection made using TF-IDF reliably. Refining the pre-selection this way, our model improved on the performance of TF-IDF alone, making it a promising candidate for open-domain question answering, even though it makes use of a non-machine-learning component (TF-IDF), making it not "end-to-end" (i. e. fully differentiable and therefore trainable with backpropagation).

Finally, we applied our model within the proposed open-domain question answering pipeline in section 6.6, achieving state-of-the-art performance on the SQuAD dataset. We also applied our model to several other question answering datasets in the same open-domain context (that is, dicarding the datasets' provided contexts). On these datasets, we did not match the state of the art in performance.

Despite that, the model fulfills our original goal—answering open-domain questions using QANet as a traditional question answering system, fulfilling our requirements for good evaluation speed, scaling to a large corpus and question answering performance.

7.2 Discussion

The encoder-only model was our first attempt at a contribution to the open-domain question answering problem. This approach promised to be fast—at least capable of handling millions of candidate paragraphs with acceptable performance. It also represented a completely novel approach that had not been attempted in this way in published literature. All variations of this paradigm failed to live up to the expectations of accuracy (or, more specifically, recall). Having tested a variety of hyperparameters, we can rule out some causes for this lack of performance but we can ultimately not say with certainty why it did not perform (after all, if we could, we would have improved the model accordingly). The two most telling data points for trying to find the cause are the performance improvements after training with an uneven ratio of correct to incorrect training examples and after adding an attention mechanism in the classifier model.

The first of those is easy to understand: Training with more incorrect examples than correct ones allows us to make use of more of our data (the model sees more paragraphs than it would if it trained with fewer incorrect examples) and more accurately resembles the true data distribution in the testing set. It is easy to see why both of these factors would improve a machine learning system. While training with the classifier, we also determined that an additional improvement can be gained by training with samples retrieved by TF-IDF rather than random ones, again mimicking the data during testing.

The improvement yielded by the attention mechanism is likely due to a combination of two factors: Without it, the model can only compare the final encoding of the complete sentence, whereas with the attention mechanism, it can perform a pair-wise comparison of each word in the sentence and question. However, contrary to TF-IDF it does not need to rely on an exact match of the question and context words. It can recognize similar words and through the bidirectional LSTM also make use of contextual features (such as recognizing acronyms or synonyms). The sentence embedding cannot encode the specifics of each word due to its restriced dimensionality. Effectively, using the attention mechanism greatly increases the dimensionality of the sentence embedding, without requiring that the system learns many new features.

With the sentence encoding, we hoped to arrive at an encoding that captures the semantic information contained in the sentence and similarly the semantic information requested in the question. The best paragraph for the question would then be the one where the encodings have the greatest similarity. Our model did not achieve this. Perhaps the encoder was too simple and unable to capture relevant information. On the other hand, even the QANet encoder, which is intended to capture information sufficient for answering the question, did not perform better at this task. This points towards the problem being the training methodology: The encoders were trained without a connected question answering system that may have provided additional feedback about the quality of the encoding. Training a question answering model at the same time as the encoders may have yielded additional distant supervision data which may have led to an improvement in performance. Finally, it is possible that the encoding size, be it 300 or 1000 (the largest we tested), was simply insufficient to capture enough information to differentiate similar and dissimilar paragraphs.

With the classifier model, we investigated a more reliable, but less innovative approach. It is admittedly similar to previous work done in the field, and the differences are subtle at times (R^3 samples the top paragraph randomly based on the softmax output of the classifier, DrQA uses only TF-IDF, [23] uses no attention mechanism in the selector, [15] retrieves 5 paragraphs and computes answers on all of them, etc.), so it is no surprise that its performance does not greatly exceed that of the other models on SQuAD. If there had been more time, we could have properly trained the model on the other datasets and likely brought the performance on them closer in line with that on SQuAD.

Achieving state-of-the-art performance on the SQuAD dataset is in part because the full pipeline model benefitted from the already excellent performance of the pre-trained QANet model. That was one of the goals of this work: To allow the use of a question answering model trained on the abundant traditional question answering data and apply it to an open-domain context. Our model makes no specific assumptions about QANet in particuar—when a better question answering model is developed, we can substitute it instead, probably to the benefit the total system performance.

7.3 Future Work

It is the belief of the author that the the encoder-only system has not yet been brought to its true potential. Lin et al. [23], published in July 2018 (after this work has started) uses a somewhat similar approach to retrieve a paragraph from a smaller set. The authors, like those of [3, 15, 44], trained their retrieval model together with a question answering model (most adopt the terminology of ranker and reader for the retrieval and question answering part, respectively). We believe that the key to improving the model is not necessarily the structure of the encoder, but how the model is trained. One possible approach is to train it using a question answering system as distant supervision. Reinforcement learning, as used by R^3, is also a possible avenue of research (as the evaluation of a prediction is always binary—the selected paragraph either contains the answer or it does not).

The classifier model has shown good performance on the SQuAD dataset. When improving it, care must be taken not to add to its complexity so much that it fails one of the requirements that were set for it—being faster than running the question answering system on all the pre-selected paragraphs. Bi-LSTMs showed the best performance, even exceeding that of the much more complex QANet encoder. But it is hard to believe that they could not be improved. Possible approaches might be adding a self-attention mechanism [39] or adding traditional NLP features such as named entity recognition, or part-of-speech tags [5].

Additionally, simply training the classifier on the other datasets should improve retrieval accuracy, rather then using the model pre-trained on SQuAD. Unfortunately, there was no more time to fully realize such a system.

A major improvement to the system as a whole would be the ability to attend to multiple paragraphs at once, cross-referencing multiple sources to disambiguate a query. For example, to TF-IDF (and the classifier model), “Eiffel Tower” refers just as much to the building in Las Vegas as it does to the one in Paris. By taking into account multiple paragraphs, the system would be able to determine that the one in Paris is the one more likely to be referred to.

Such an improvement could also lead to the use of the model as a dialog system, using previous statements and questions as additional information for clarification, which leads to a more natural communication with the computer.

Overall, our results show that open-domain question answering is a rapidly advancing area of research. The ability of machine learning systems to comprehend and use natural language is improving continuously. As a result of this work, the advances of traditional question answering systems can be made available to the open domain problem, pushing the boundaries of what questions computers can answer.

Bibliography

[1] Ossama Abdel-Hamid et al. “Applying convolutional neural networks concepts to hybrid NN-HMM model for speech recognition”. In: *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*. IEEE. 2012, pp. 4277–4280.

[2] Lyle F Bachman. “The trait structure of cloze test scores”. In: *Tesol Quarterly* 16.1 (1982), pp. 61–70.

[3] Bing Bai et al. “Learning to rank with (a lot of) word features”. In: *Information Retrieval* 13.3 (June 2010), pp. 291–314. ISSN: 1573-7659. DOI: 10.1007/s10791-009-9117-9.

[4] Petr Baudiš and Jan Šedivỳ. “Modeling of the question answering task in the yodaqa system”. In: *International Conference of the Cross-Language Evaluation Forum for European Languages*. Springer. 2015, pp. 222–228.

[5] Danqi Chen et al. “Reading Wikipedia to Answer Open-Domain Questions”. In: *CoRR* abs/1704.00051 (2017).

[6] Bhuwan Dhingra, Kathryn Mazaitis, and William W. Cohen. “Quasar: Datasets for Question Answering by Search and Reading”. In: *CoRR* abs/1707.03904 (2017).

[7] Jesse Dodge et al. “Evaluating Prerequisite Qualities for Learning End-to-End Dialog Systems”. In: *CoRR* abs/1511.06931 (2015).

[8] Jun Du. “The USTC-iFlytek system for CHiME-4 challenge”. In: ().

[9] Jeffrey L Elman. “Finding structure in time”. In: *Cognitive science* 14.2 (1990), pp. 179–211.

[10] Kunihiko Fukushima and Sei Miyake. “Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition”. In: *Competition and cooperation in neural nets*. Springer, 1982, pp. 267–285.

[11] Samanwoy Ghosh-Dastidar and Hojjat Adeli. “Spiking neural networks”. In: *International journal of neural systems* 19.04 (2009), pp. 295–308.

[12] Bert F Green Jr et al. “Baseball: an automatic question-answerer”. In: *Papers presented at the May 9-11, 1961, western joint IRE-AIEE-ACM computer conference*. ACM. 1961, pp. 219–224.

[13] Karl Moritz Hermann et al. “Teaching Machines to Read and Comprehend”. In: *Advances in Neural Information Processing Systems 28*. Ed. by C. Cortes et al. Curran Associates, Inc., 2015, pp. 1693–1701.

[14] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[15] Phu Mon Htut, Samuel R. Bowman, and Kyunghyun Cho. “Training a Ranking Function for Open-Domain Question Answering”. In: *CoRR* abs/1804.04264 (2018).

[16] Minghao Hu, Yuxing Peng, and Xipeng Qiu. “Mnemonic Reader for Machine Comprehension”. In: *CoRR* abs/1705.02798 (2017).

[17] Mandar Joshi et al. “TriviaQA: A Large Scale Distantly Supervised Challenge Dataset for Reading Comprehension”. In: *CoRR* abs/1705.03551 (2017).

[18] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. “A Convolutional Neural Network for Modelling Sentences”. In: *CoRR* abs/1404.2188 (2014).

[19] Seonhoon Kim et al. “Semantic Sentence Matching with Densely-connected Recurrent and Co-attentive Information”. In: *CoRR* abs/1805.11360 (2018).

[20] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *CoRR* abs/1412.6980 (2014).

[21] Tomáš Kočiský et al. “The NarrativeQA Reading Comprehension Challenge”. In: *CoRR* abs/1712.07040 (2017).

[22] Guokun Lai et al. “RACE: Large-scale ReAding Comprehension Dataset From Examinations”. In: *CoRR* abs/1704.04683 (2017).

[23] Yankai Lin et al. “Denoising distantly supervised open-domain question answering”. In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vol. 1. 2018, pp. 1736–1745.

[24] Calvin N Mooers. *Making information retrieval pay*. 55. Zator Company, 1951.

[25] Lawrence Page et al. *The PageRank Citation Ranking: Bringing Order to the Web*. Technical Report 1999-66. Previous number = SIDL-WP-1999-0120. Stanford InfoLab, Nov. 1999.

[26] Karalyn Patterson, Peter J Nestor, and Timothy T Rogers. “Where do you know what you know? The representation of semantic knowledge in the human brain”. In: *Nature Reviews Neuroscience* 8.12 (2007), p. 976.

[27] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. “GloVe: Global Vectors for Word Representation”. In: *Empirical Methods in Natural Language Processing (EMNLP)*. 2014, pp. 1532–1543.

[28] Pranav Rajpurkar, Robin Jia, and Percy Liang. “Know What You Don’t Know: Unanswerable Questions for SQuAD”. In: *arXiv preprint arXiv:1806.03822* (2018).

[29] Pranav Rajpurkar et al. “SQuAD: 100, 000+ Questions for Machine Comprehension of Text”. In: *CoRR* abs/1606.05250 (2016).

[30] Matthew Richardson, Christopher JC Burges, and Erin Renshaw. “Mctest: A challenge dataset for the open-domain machine comprehension of text”. In: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. 2013, pp. 193–203.

[31] Pum-Mo Ryu, Myung-Gil Jang, and Hyun-Ki Kim. “Open domain question answering using Wikipedia-based knowledge model”. In: *Information Processing & Management* 50.5 (2014), pp. 683–692.

[32] Gerard Salton and J Michael. “McGill. 1983”. In: *Introduction to modern information retrieval* (1983).

[33] Rico Sennrich, Barry Haddow, and Alexandra Birch. “Neural Machine Translation of Rare Words with Subword Units”. In: *CoRR* abs/1508.07909 (2015).

[34] Min Joon Seo et al. “Bidirectional Attention Flow for Machine Comprehension”. In: *CoRR* abs/1611.01603 (2016).

[35] Dinghan Shen et al. “Baseline Needs More Love: On Simple Word-Embedding-Based Models and Associated Pooling Mechanisms”. In: *CoRR* abs/1805.09843 (2018).

[36] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. “Sequence to Sequence Learning with Neural Networks”. In: *Advances in Neural Information Processing Systems 27*. Ed. by Z. Ghahramani et al. Curran Associates, Inc., 2014, pp. 3104–3112.

[37] Yi Tay, Luu Anh Tuan, and Siu Cheung Hui. “Multi-range Reasoning for Machine Comprehension”. In: *CoRR* abs/1803.09074 (2018).

[38] Matus Telgarsky. “Benefits of depth in neural networks”. In: *CoRR* abs/1602.04485 (2016).

[39] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 5998–6008.

[40] Ellen M Voorhees et al. “The TREC-8 Question Answering Track Report.” In: *Trec*. Vol. 99. 1999, pp. 77–82.

[41] Ellen M Voorhees and Dawn M Tice. “Building a question answering test collection”. In: *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*. ACM. 2000, pp. 200–207.

[42] Alexander Waibel et al. “Phoneme recognition using time-delay neural networks”. In: *Readings in speech recognition*. Elsevier, 1990, pp. 393–404.

[43] Shuohang Wang et al. “Evidence Aggregation for Answer Re-Ranking in Open-Domain Question Answering”. In: *CoRR* abs/1711.05116 (2017).

[44] Shuohang Wang et al. “R^3: Reinforced Reader-Ranker for Open-Domain Question Answering”. In: *CoRR* abs/1709.00023 (2017).

[45] Kilian Weinberger et al. “Feature Hashing for Large Scale Multitask Learning”. In: *Proceedings of the 26th Annual International Conference on Machine Learning*. ICML ’09. Montreal, Quebec, Canada: ACM, 2009, pp. 1113–1120. ISBN: 978-1-60558-516-1. DOI: 10.1145/1553374.1553516.

[46] Joseph Weizenbaum. “ELIZA—a computer program for the study of natural language communication between man and machine”. In: *Communications of the ACM* 9.1 (1966), pp. 36–45.

[47] Johannes Welbl, Pontus Stenetorp, and Sebastian Riedel. “Constructing Datasets for Multi-hop Reading Comprehension Across Documents”. In: *CoRR* abs/1710.06481 (2017).

[48] Paul John Werbos. “Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences”. PhD thesis. Harvard University, 1974.

[49] Adina Williams, Nikita Nangia, and Samuel Bowman. “A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference”. In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, 2018, pp. 1112–1122.

[50] Caiming Xiong, Victor Zhong, and Richard Socher. “Dynamic Coattention Networks For Question Answering”. In: *CoRR* abs/1611.01604 (2016).

[51] Yi Yang, Wen-tau Yih, and Christopher Meek. “Wikiqa: A challenge dataset for open-domain question answering”. In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. 2015, pp. 2013–2018.

[52] Adams Wei Yu et al. “QANet: Combining Local Convolution with Global Self-Attention for Reading Comprehension”. In: *CoRR* abs/1804.09541 (2018).

List of Figures

List of Tables