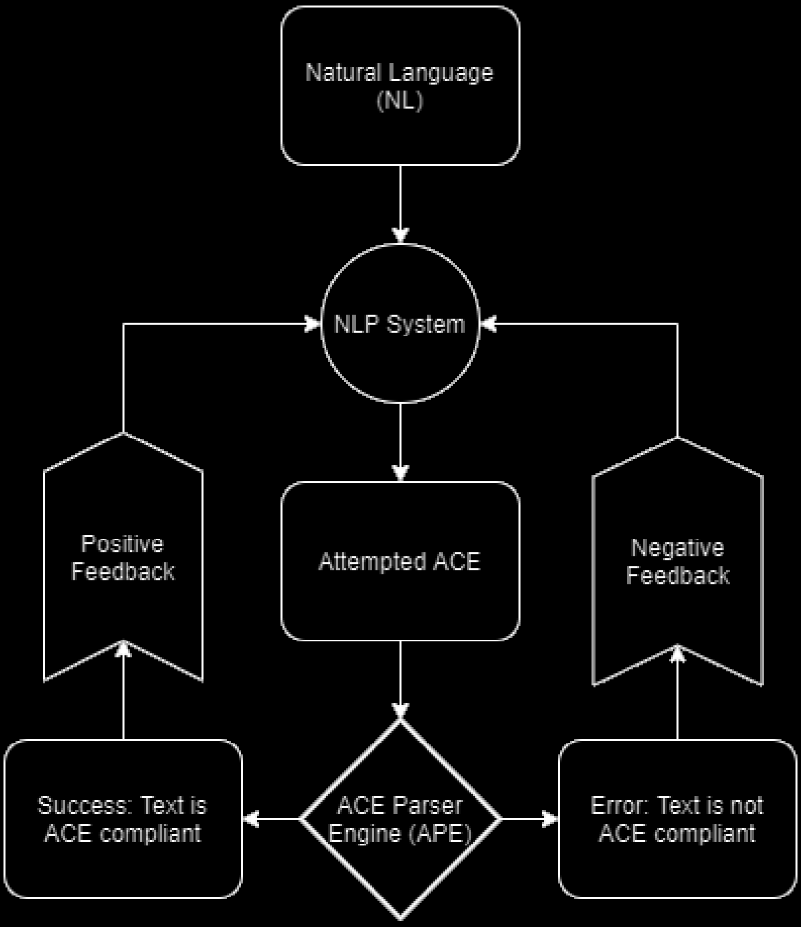
# Preprocessing for ACE

An experiment in mapping from Natural Language to First Order Logic



Max Petra

Layout: typeset by the author using LaTeX.
Cover illustration: Self training phase explained in chapter 3

# Preprocessing for ACE

An experiment in mapping from Natural Language to First Order Logic

Max Petra
11028335

Bachelor thesis
Credits: 18 EC

Bachelor *Kunstmatige Intelligentie*

University of Amsterdam
Faculty of Science
Science Park 904
1098 XH Amsterdam

*Supervisor*
Dr. Giovanni Sileno

Informatics Institute
Faculty of Science
University of Amsterdam
Science Park 907
1098 XG Amsterdam

June, 2021

# Chapter 1

# Abstract

A controlled natural language was long considered to be the holy grail in computer linguistics; a language that could be understood and used by humans, yet was easily parseable into first order logic by relatively lightweight rule-based software. That was until Attempto Controlled English or ACE came along in the mid nineties. Unfortunately ACE suffers from a variation on Esperanto syndrome: Its lack of widespread use prevents its widespread use. Popularity aside though, the limited rule set of ACE prevents it from being much more than an academic exercise, rather than an actual language. Its proximity to natural English, however, presents an interesting possibility in the age of machine learning based natural language processing; Is it possible to build an NLP framework that translates natural language into simple statement sentences that conform to ACE, thus completing the chain from natural language to first order logic? This paper outlines several possible designs for such a system, as well as describing relevant theory and a report on a proof-of-concept experiment based on Keras.

# Chapter 2

# Attempto Controlled English

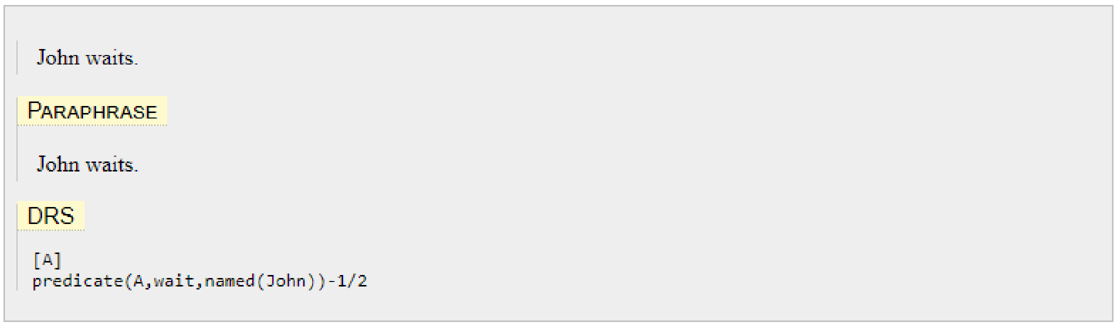
Attempto[1] Controlled English (ACE) is a controlled natural language, or, to be more precise, a controlled subset of English [9]. It is a formally defined language that, using the accompanying ACE Parser Engine (APE), can be unambiguously translated into discourse representation structures (DRS), a syntactic variant of first order logic [2]. Unlike more powerful contemporary NLP systems, the ACE parser is entirely rule-based rather than machine learning based [10]. This is the primary reason that APE[2] is written in Prolog rather than, for instance, Python. These rules are made up of a vocabulary predefined function words (e.g. determiners, conjunctions, query words), some predefined fixed phrases (there is, it is false that, etc.), and content words (nouns, proper names, verbs, adjectives, adverbs). Using a definite, yet expansive ruleset ACE strikes a remarkable balance between having enough depth to describe complex concepts and being limited enough to have direct logical translations. Indeed, it has enough depth to describe something as complex as ACE itself [8].

Despite the remarkable achievement ACE represents, it's still a far cry from natural English. As you can see in figure 2.1, APE is capable of parsing simple sentences without raising any errors; the relatively large ruleset of ACE is a subset of the ruleset of natural English, thus allowing simple English sentences to be ACE compliant without being specifically written with that goal in mind. However, as the second query in figure **??** shows, longer and more natural sentences produce a

[1]Not 'attempt to', but the Latin attempto, meaning "I dare". This slogan was originally conceived in the 1400's and repeated by Norbert E. Fuchs when he started the project, defying the "has been tried, can't be done" sentiment that was prominent in the field of computerlinguistics at the time regarding controlled natural language [2].

[2]Throughout this paper APE and ACE will be referenced seemingly interchangeably; this is because we will be translating natural language to ACE and using the parsing engine to verify that the output indeed conforms to the ACE ruleset. In other words, ACE is the goal, APE is the means by which we verify our having reached that goal.

significant number of errors.



John waits.

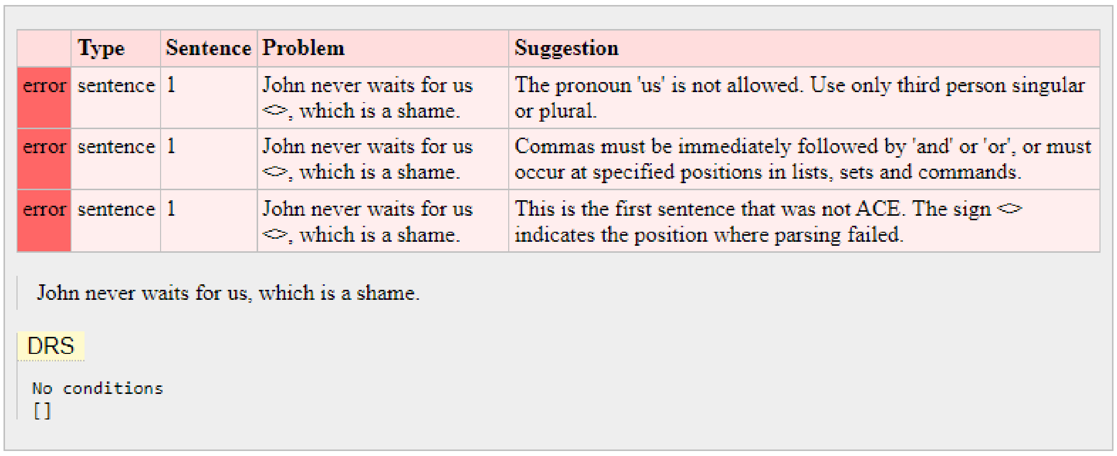
PARAPHRASE

John waits.

DRS

```
[A]
predicate(A,wait,named(John))-1/2
```

Figure 2.1: APE output for the sentence "John waits.", as per the online APE client. [18]



| | Type | Sentence | Problem | Suggestion |
|---|---|---|---|---|
| error | sentence | 1 | John never waits for us <>, which is a shame. | The pronoun 'us' is not allowed. Use only third person singular or plural. |
| error | sentence | 1 | John never waits for us <>, which is a shame. | Commas must be immediately followed by 'and' or 'or', or must occur at specified positions in lists, sets and commands. |
| error | sentence | 1 | John never waits for us <>, which is a shame. | This is the first sentence that was not ACE. The sign <> indicates the position where parsing failed. |

John never waits for us, which is a shame.

DRS

```
No conditions
[]
```

Figure 2.2: APE output for the sentence "John never waits for us, which is a shame.", as per the online APE client. [18]

Figure 2.3 shows a short story from Aesop, translated for children [1]. Despite its simple structure, it's beyond the parsing capabilities of APE. In figure 2.4 the fable has been deconstructed to simple, single statement sentences that can all be parsed by APE without error or warning, while maintaining the story's key events and objects. Note that a great deal of information and nuance is lost in this deconstruction to make the text ACE compliant; this is not a flaw in the functionality or robustness of ACE specifically, but a reality of representing even relatively simple natural language in logical terms. Nuance by its nature simply cannot[3] be adequately represented in logic.

[3]There is a certain irony to saying this cannot be done in a paper working with ACE, some-

Going back to the two texts, one can get a far better idea of the structure of ACE and its ruleset. More importantly, we get a clear picture of the differences between ACE and natural English, and how best to bridge them. The task or bridging these two is the focus of this thesis and can be described in two general ways: a paraphrasing or text simplification task, or a translation task (from natural English to a simplified version of English with a limited subset of its vocabulary and grammar). There are systems available for either approach. We'll return to this question in section 3.1.

One final observation to make about ACE is to further explore its relative simplicity. It's structurally far closer to simple languages such as the one used in Randall Munroe's “Thing Explainer”, a subset of English only using the 1000 most commonly used English words. It's no coincidence then that an analysis of the toy language and the book Munroe wrote entirely in it was written by a long time contributor to the ACE project, Tobias Kuhn [14]. Munroe's book goes into detail in explaining rather complex mechanical systems using only basic descriptors, and does so rather effectively (though not particularly efficiently). The reason this minimized language does so well is that while the vocabulary is severely limited, the grammar isn't. Using enough commonly used descriptors and the full might of complex grammar Munroe is able to describe increasingly complex objects and concepts using an equally increasing complex combination of descriptors. Indeed, by comparison the only thing holding ACE back is the obligation by design to be directly translatable to logic. Despite this it still manages to be a powerful language tool, as shown in 'ACE can be described by itself' [8].

thing that was regarded with similar skepticism for many years until it was achieved. Here's to being proven wrong!

```
A cock was once strutting up and down the farmyard
among the hens when suddenly he espied something
shinning amid the straw. "Ho! ho!" quoth he,
"that's for me," and soon rooted it out from beneath
the straw.  What did it turn out to be but a Pearl
that by some chance had been lost in the yard?
"You may be a treasure," quoth Master Cock,
"to men that prize you, but for me I would rather
have a single barley-corn than a peck of pearls."
```

Figure 2.3: The Cock and the Pearl by Aesop

```
There is a cock.
There is a farmyard.
The cock walks in the farmyard.
There is more than one hen.
The cock walks among the hen.
There is some straw.
There is a pearl.
The pearl is in the straw.
The cock espies the pearl.
The cock says "Ho! ho!".
The cock says "that's for me".
The cock roots out the pearl.
The pearl is lost in the farmyard.
The cock says "You may be a treasure, to men that prize you,
but for me I would rather have a single barley-corn than
a peck of pearls.".
```

Figure 2.4: The Cock and the Pearl by Aesop translated to ACE

# Chapter 3

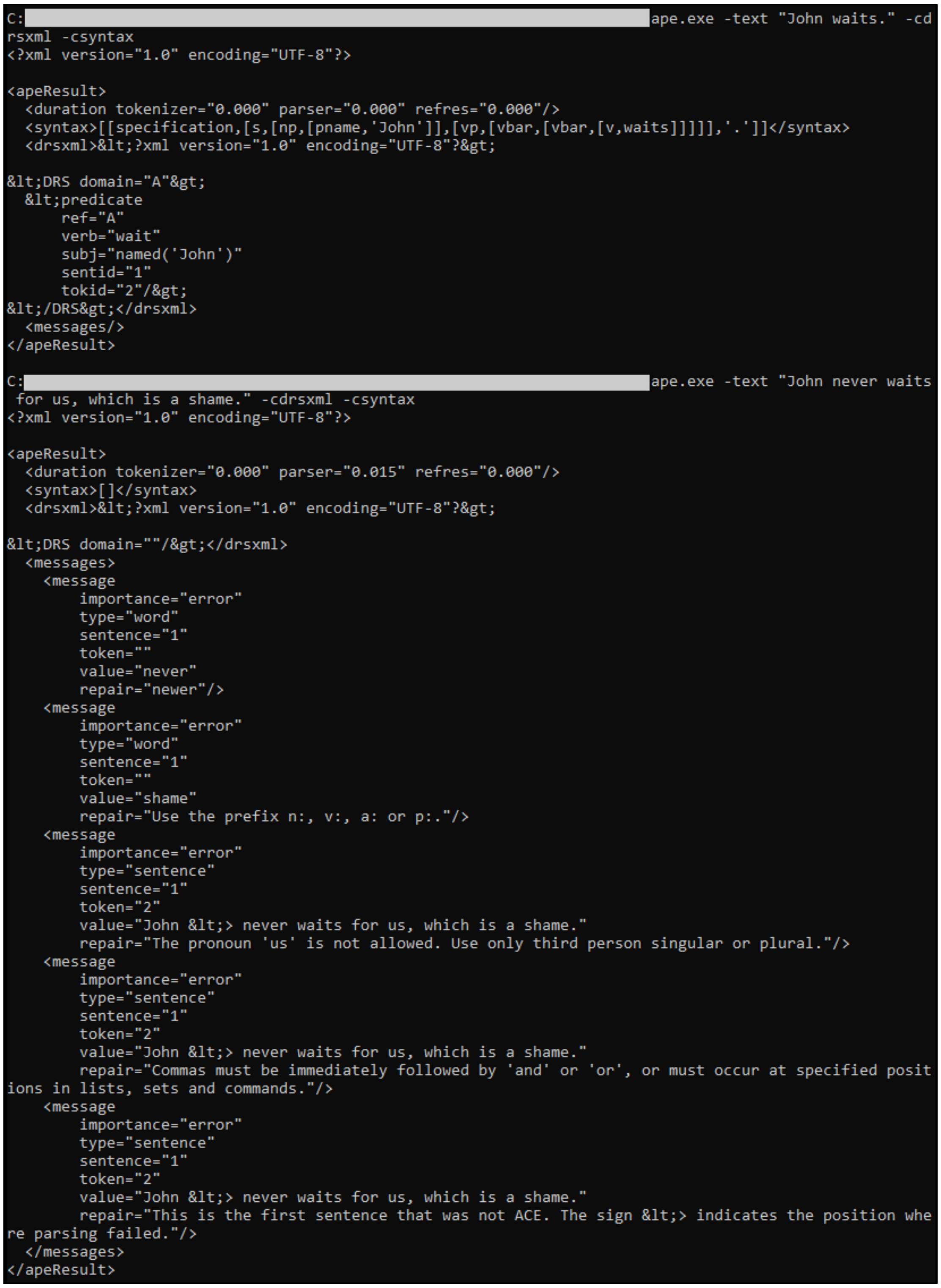
# Method

As the training set has to be manually created it's impractical to create enough data to sufficiently train the system (and likely impossible within the limited time frame of this project). Fortunately the open availability of APE allows for automated querying, in turn allowing for automated training. The methodology behind this will be explained below.

## 3.1 Proposed system architecture

Figure 3.2 shows the initial training phase to be completed using the manually created dataset. This proceeds in a similar manner to most training phases of general purpose neural networks; feeding in as much data as is available, tuning where needed and verifying the final outputs. From here on out, though, we are able to greatly improve accuracy; a must given the limited dataset. At this stage, it would most likely be necessary to continually input new data rather than reusing it in order to prevent over-fitting.

Figure 3.3 shows how including the automated APE interface allows for feedback on the NLP system. In figure 3.1 you can see the APE system running on a local Windows machine within the command prompt. Note that beyond either saying whether a text is or isn't ACE compliant, it returns a detailed report of what exactly has gone wrong. This allows a framework to either check if a text is correct or not (purely binary), return the number of errors, or perhaps even give a different 'severity' score to different error types to further distinguish between more and less severe mistakes. This allows the entire framework to be essentially self-training, and could run largely unsupervised indefinitely. The longer it runs and the larger the input corpus, the more accurate and precise the system becomes. Depending on performance and results, this system might even be able to continue training by retraining itself on the same dataset over and over, rather



```
C:[redacted]ape.exe -text "John waits." -cd
rsxml -csyntax
<?xml version="1.0" encoding="UTF-8"?>

<apeResult>
  <duration tokenizer="0.000" parser="0.000" refres="0.000"/>
  <syntax>[[specification,[s,[np,[pname,'John']],[vp,[vbar,[vbar,[v,waits]]]]],'.']]</syntax>
  <drsxml>&lt;?xml version="1.0" encoding="UTF-8"?&gt;

&lt;DRS domain="A"&gt;
  &lt;predicate
      ref="A"
      verb="wait"
      subj="named('John')"
      sentid="1"
      tokid="2"/&gt;
&lt;/DRS&gt;</drsxml>
  <messages/>
</apeResult>

C:[redacted]ape.exe -text "John never waits
 for us, which is a shame." -cdrsxml -csyntax
<?xml version="1.0" encoding="UTF-8"?>

<apeResult>
  <duration tokenizer="0.000" parser="0.015" refres="0.000"/>
  <syntax>[]</syntax>
  <drsxml>&lt;?xml version="1.0" encoding="UTF-8"?&gt;

&lt;DRS domain=""/&gt;</drsxml>
  <messages>
    <message
        importance="error"
        type="word"
        sentence="1"
        token=""
        value="never"
        repair="newer"/>
    <message
        importance="error"
        type="word"
        sentence="1"
        token=""
        value="shame"
        repair="Use the prefix n:, v:, a: or p:."/>
    <message
        importance="error"
        type="sentence"
        sentence="1"
        token="2"
        value="John &lt;> never waits for us, which is a shame."
        repair="The pronoun 'us' is not allowed. Use only third person singular or plural."/>
    <message
        importance="error"
        type="sentence"
        sentence="1"
        token="2"
        value="John &lt;> never waits for us, which is a shame."
        repair="Commas must be immediately followed by 'and' or 'or', or must occur at specified posit
ions in lists, sets and commands."/>
    <message
        importance="error"
        type="sentence"
        sentence="1"
        token="2"
        value="John &lt;> never waits for us, which is a shame."
        repair="This is the first sentence that was not ACE. The sign &lt;> indicates the position whe
re parsing failed."/>
  </messages>
</apeResult>
```

Figure 3.1: APE output for the sentences "John waits." and "John never waits for us, which is a shame." as per the code available on github. [3]

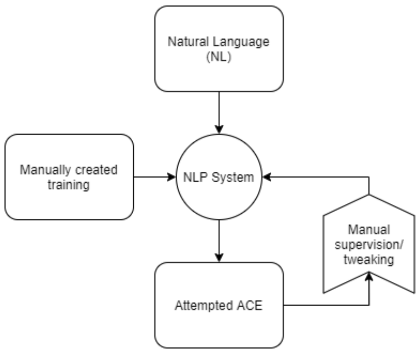


Figure 3.2: Initial manual training

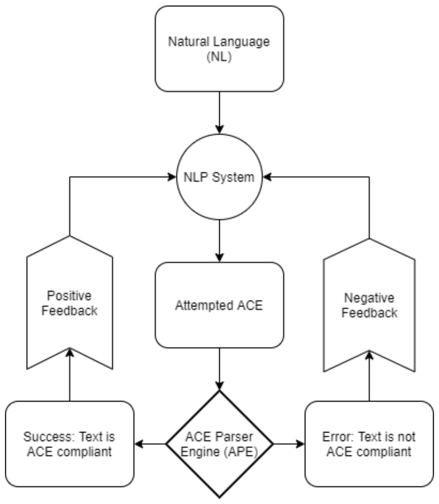


Figure 3.3: Primary self-training

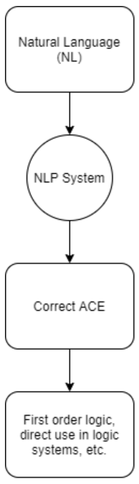


Figure 3.4: Completed system

than requiring more data. Note how, in theory, this system could be run in this way without any manual training preceding it and still eventually produce a viable system thanks to the feedback it receives from APE. This brute force method would, however, require a great deal of time and computing power before chancing upon a translation that conforms to ACE even once, let alone enough times to reinforce that behavior.

Figure 3.4 shows what a completed 'product' based on a fully trained system will look like: A fully trained system that takes a simple natural language input and translates it into ACE compliant sentences. These sentences can then be decomposed into DRS as needed, or easily converted into some other logical system by a simple rule-based converter. Note that this is a theoretical system, and in practice one would adopt a system like this that can't be modified or corrected. No translation system is ever 'done' unless it is 100% accurate, which besides being an impractical goal for mathematical reasons is impossible in a system that works with language, something that is by definition open to interpretation.

## 3.2 Framework Selection

Having properly defined the task to be completed, selecting the correct framework to complete said task becomes more feasible. Fortunately anyone looking for a sentence transformation model will find themselves spoiled for choice; From pre-trained models to blank slates, freshly published to tried-and-true, and of course, from ready made 'just-add-dataset' implementations to interdependent houses of python-package-cards. As a result, several frameworks of various types were tested for ease of implementation, fitness for the task, and overall performance.

### 3.2.1 Machine translation frameworks

One of the most promising models for this purpose is the as of yet still in beta Generative Pre-trained Transformer 3 (GPT-3) [4]. One of its key attributes is that it first trained on a large corpus of general purpose text before being fine tuned to a specific task. Given the need to manually create a training set for ACE[1], the ability to have a well functioning system with a limited training set is an extremely valuable one in this case. On November 9th 2020 access was requested to the closed GPT-3 Beta [19]. Unfortunately, as of writing this paper no reply to this request has been given.

Given the limited time frame of this project, other options were immediately investigated. A more promising choice than the obvious alternative to GPT-3 (which would be GPT-2) is the Bidirectional Encoder Representations from Transformers (BERT) [7]. Similarly to GPT-3 BERT goes through so-called pre-training on a massive corpus of unlabeled text before being trained on a smaller specialized corpus. What sets BERT apart from the competition, however, is it's bidirectional training paradigm. Whereas most frameworks' encoders read text either right-to-left or, more commonly, left-to-right, BERT reads all input simultaneously[2]. This allows BERT to be far more context-sensitive than other frameworks of its caliber, a suspected advantage in this particular use case. This is because, as shown in figures 2.3 and 2.4, translating to ACE requires an awareness of meaning spread out over several sentences. The lines spoken by the cock are an example of this.

One option that was briefly considered after researching BERT is BART, a denoising autoencoder with architecture based upon both BERT and the GPT

[1]This is a deceptively important point which will be further explained later in this section.

[2]As such, calling BERT bidirectional is somewhat misleading; it would be more accurate to say BERT has no direction at all.

design paradigms [15]. It achieves this by using BERT's bidirectional encoder and GPT's auto-regressive decoder, linked together with an abstraction layer. This allows for seemingly arbitrary noise transformations as the inputs and decoder outputs need not be aligned. This was thought to be of particular use in ACE translation, as ACE's limited sentence complexity requires introducing significant amounts of 'noise' to construct several sentences from a single source sentence. Unfortunately the steep learning curve prevented us setting up the framework and achieving any results in the allotted time, so we moved on to more manageable solutions.

Going back to BERT, many implementations of and frameworks for it have been introduced since its initial release in 2018, but arguably one of the most popular is the OpenNMT variant [22]. Open-Source toolkit for Neural Machine Translation (OpenNMT) is a general purpose framework that has actually been around slightly longer than BERT itself [13]. Nonetheless, its younger BERT-based variant presents a robust yet flexible machine translation interface. If a translation framework is to be used for a practical application of ACE preprocessing, it would most likely prove one of the more favorable candidates.

Another way to approach machine translation is by utilizing a general-purpose deep learning framework, such as Keras. Keras is presented as a user-friendly interface for the commonly used TensorFlow2. To quote the about page [12]: *"Being able to go from idea to result as fast as possible is key to doing good research."*. Its use of TensorFlow2 allows for excellent performance even when compared to other frameworks, despite usability and performance usually being a zero sum game.

Unfortunately, all of these frameworks have an Achilles's heel which was eluded to earlier; they are translation frameworks first and foremost. As such, they require pairs of sentences for training, in our case one natural English, one ACE. As there is no readily available corpus of natural English texts with ACE counterparts, this would require creating one manually. Even with the aforementioned benefits of using APE to automatically validate outputs and give feedback to a framework (not to mention both input and output are technically the same language), for that chain to be effective even the most advanced translation frameworks in the world would need a rather sizeable corpus to start the self-training process off. After all, the feedback it gives is a simple score and doesn't offer up a correction which could be added to the training set.

### 3.2.2 Paraphrasing frameworks

This is where we transition from translation frameworks to paraphrasing frameworks. Though there are several functional differences between the two styles, two of them are of particular interest to this use case. Generally speaking, paraphrasing frameworks aren't trained by equivalence pairs. Instead they transform all corpus sentences and the input into vector representations and select the corpus sentence with the closest vector. This is useful not only because it eliminates traditional training, but also because the APE source code includes an expansive list of ACE compliant sentences in `tests/acetexts.pl` for referencing [3]. By using this list as a corpus, we are essentially providing a framework with a dictionary of possible ACE sentences. Given the limited possible sentence structures of ACE (at least compared to the natural languages translation frameworks are usually applied to) this should result in a significant performance and accuracy advantage over the alternative.

Some of the most recent and work in this area was done by Hannes Westermann et al., on the Computer-Assisted Efficient Semantic Annotation & Ranking (CAESAR) interface [25]. In a similar vein to this project, CAESAR aims to automate the annotation of legal documents for the purposes of creating datasets for other machine learning systems. These lateral annotations are chosen based on meaning captured by sentence embeddings. The choice of sentence embedding vectors over the also widely used bag-of-words representation is particularly advantageous in the ACE conversion use case; as natural English and corresponding ACE are semantically similar, but very different syntactically. On top of that, a bag-of-words vector representation has no means by which to connect semantically similar words, a key factor in many ACE conversions (for example, verb tense). What truly sets this system apart, however, is its use of semantic sentence embeddings and efficient vector similarity search to enable lateral annotation. Unfortunately the public release of CAESAR will not made available in time to be used for this project[3].

One final point to be made regarding the framework to be used is that of object definition. Referring back to figure 2.3 and 2.4, you will note that every object needs to be defined before it is used in a sentence (The cock, the farmyard, etc.). This is of particular concern to the translation problem, as all mentioned models (and most models in general) work one sentence at a time. As such, it is likely that a fully trained system will produce superfluous information by re-defining every object before using it again in the next sentence.

## 3.3 Dataset

The final decision to be made is the dataset with which the system will be trained, validated, and evaluated. In theory the designed system is extremely robust to the point of being able to process nearly any kind of natural language input.

[3]As of the writing of this paper, it has not yet been made available. In a private correspondence with one of the authors it was made clear a publicly available version is in the works, and will be posted on Github.

| | |
|---|---|
| A man likes a dog. It barks. | A man likes a dog. It barks. |
| "" | A man likes a dog that barks. |
| "" | A man likes a dog that is barking. |
| "" | A man is liking a dog that barks. |
| "" | A man is liking a dog. The dog barks. |
| "" | A man likes a dog. The dog is barking. |

Figure 3.5: Example of a dataset created by translating a single ACE text to several natural English variants.

Unfortunately the limitations of ACE (and, as previously discussed, all logical derivatives of natural language) combined with the short time in which this project is to be completed force the use of a specific kind of dataset. In this case, one with unambiguous underlying meaning, a relatively simple grammatical structure and few 'rare' words; Simply put, input as close to ACE as can be found in natural English [4]. As eluded to earlier, Aesop's fables translated for children are an excellent candidate. The corpus fits all mentioned requirements, is entirely public domain, and requires minimal formatting for use as input.

However, translating from natural English to ACE is a tedious process; as such, translating a sufficiently large dataset to even start off the initial self-training phase is impractical. Another option is to work in reverse, translating an existing list of ACE sentences into richer natural English sentences. With the aforementioned `tests/acetexts.pl` file, we essentially have a corpus of 3779 ACE compliant sentences which can be translated into natural English far more easily than the other way around. Since translating a sentence to ACE is in essence a disambiguation/denoising operation, translating from ACE to English is as simple as adding 'noise'. This noise can take the form of different verb tenses, stringing ACE sentences together into more natural sentences, or adding superfluous information/nuance through words not in the ACE dataset. By doing this, we can essentially create several natural English counterparts for a single ACE compliant sentence, thus creating a dataset at many times the speed and ease with which we would translate natural English sentences into ACE one by one. An example of this is shown in figure 3.5.

[4]Given the amount of training that can be done simply using the self-training method discussed earlier, there's no reason to worry about making things 'too easy'. If this system can work on a dataset, it can be made to work on any dataset when given enough time and computing power.

# Chapter 4

# Implementation

Unfortunately, the timeframe of this project won't allow a full realization of the architectures sketched in figures 3.3 and 3.4. As such, a smaller project of a more limited scale will be completed as a proof of concept, proving that the initial training phase (figure 3.2) is theoretically possible. As the preferred option (CAESAR) is not yet available, we will be simulating a 'worst case scenario'; machine translation using Keras and a dataset measuring in the dozens. Naturally, this will not result in anything resembling a correct output, but may show a statistically significant improvement over the course of several iterations.

Keras was chosen for this project not so much because it was the most likely to produce a favorable result[1], but for its ease of implementation and flexibility in experimentation. This is largely thanks to it being a completely open-source interface for the TensorFlow library. As such, it is a well documented framework with multitudes of tutorials available. One such tutorial [16] was used as a guide for creating the included iPython notebook, which will detail the experiment in its final form with sufficient documentation and flexibility built in for other experiments.

## 4.1 Machine translation model

After some simple preprocessing in the form of Start of Sentence (SoS) and End of Sentence (EoS) tokens, the framework tokenizes the input and output, representing sentences of words as lists of integers. The sentences are also padded to a certain length; for the input, this length is the longest input line, and for the output this length is the longest output line. This is because the model training algorithm (Long short-term memory, or LSTM) expects fixed-length inputs. As a result,

[1]As explained earlier, it is unlikely that a standard translation system of any caliber will be able to produce a favorable result with the given data.

both the input and output sets will have at least one sentence with no added padding. This is done automatically, based on provided inputs.

Once this is complete, the framework creates word embeddings; these are different from tokenizations. Where tokenizations are integers signifying words with no real semantic binding (i.e. 'this is the $n$th word I've encountered'), embeddings are vectors of integers that do represent semantic meaning[2]. In both the tutorial and the final product GloVe word embeddings are used[20].

After all this preprocessing is complete, the machine translation model itself needs to be created. The input layers are fairly standard-issue, processing the padded and tokenized inputs in separate cells before combining them in a hidden LSTM layer. The final layer of this model will need to be dense to create accurate predictions. As such, one-hot encoded vectors[3] will be needed for the softmax activation function at this layer. The next step is to define the encoder and decoders. The encoder takes our natural English input and translates it to the hidden state of the LSTM. The decoder is somewhat more complex, taking two inputs: the hidden state and the original input sentence with an added `<sos>` tag at the beginning. The entire structure is detailed in figure 4.1. It is at this point that data is input and the model is trained.

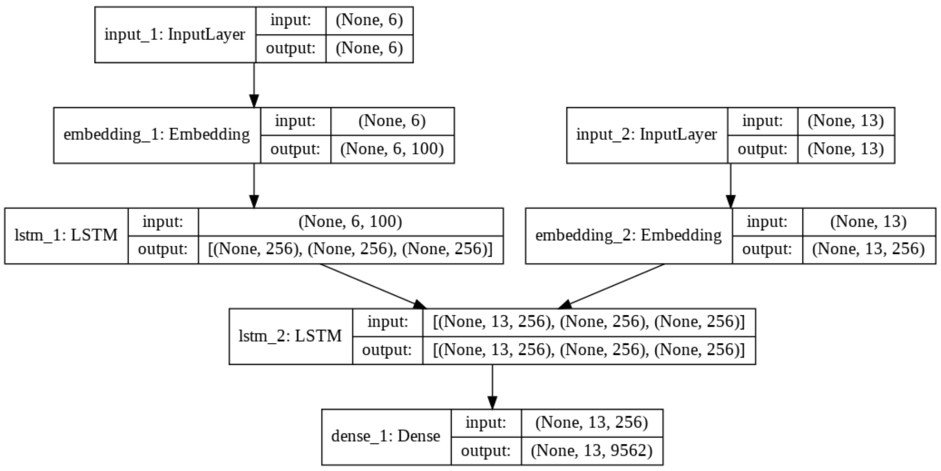


Figure 4.1: Original model layout, following [16]

[2]That is to say, semantic meaning as far as contemporary NLP models are concerned. Stating that one can capture true 'meaning' with a simple vector is problematic at best.

[3]A one-hot vector is a group of binary bits where all but one are negative, (or in this case, 0) and one bit is positive (or in this case, 1). Using such vectors is useful in machine learning because it circumvents the problem of the network giving a higher importance to higher numbers.

This model is correctly structured for training, but not for predicting. To make predictions, the model needs to be restructured to function not only on direct word translation but also bigrams (i.e. predicting which word follows a given word). This new layout is shown in figure 4.2. Now the system is ready to make predictions.

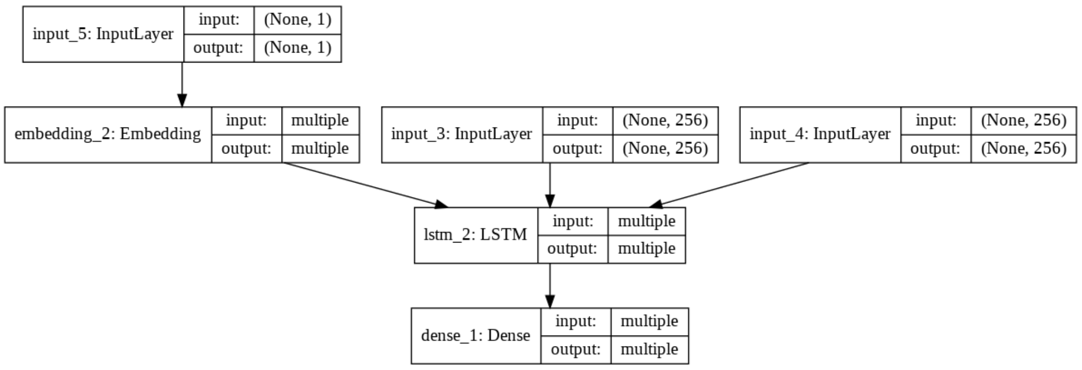


Figure 4.2: Final model layout, following [16]

## 4.2 Training method

To measure the level (and existence) of improvement, separate models are trained on subsets of the dataset; a model trained on the first two sentences, the first three, the first four, etc. While the ordering of the dataset is deterministic (i.e. it always picks the first two sentences, then the first three, and so on) the division of that subset into training and validation sets is random. As such, re-running this experiment can result in differing results than the ones that will be presented here, especially in the models trained using a small number of sentences. Repeated experimentation, however, has shown that while the results can fluctuate somewhat from one experiment to the next, the impact on the eventual conclusion is nonexistent. After generating a model, it is used to translate several test sentences.

Running this system over a dataset of $n$ English/ACE pairs results in $n - 1$ translations[4] in order from least to most data used. In theory, the last of these translations should be the most similar to a correct one. However, this isn't the entire set that each consecutive model is trained with. As ACE uses a subset of the rules of natural English, a direct reproduction of any natural English sentence is already relatively 'close' to ACE compliant. As such, a significant level of

[4]The first step in the iteration uses two pairs so that the set can be split into training and validation sets. This step is technically depreciated by the the pretraining step that will be explained later, but was left in place to keep the code flexible for further experimentation.

robustness can be added to the model by adding natural English to natural English pairs, containing equal text.

The pretraining database used for the final experiment, `pretrain20000.txt`, is constructed from the 3768 lines of ACE compliant sentences taken from the aforementioned `acetexts.pl` (with all formatting removed) combined with lines from an English to French translation (with a few duplicates removed) set to total 20.000 lines[5]. This translation set was taken from the Tatoeba Project [24], which is also used in the aforementioned Keras tutorial [16]. These lines are appended to the start of every iteration as both input and output.

| | Natural English | Attempto Controlled English |
|---|---|---|
| 1 | A man walked | There is a man. The man walks. |
| 2 | A man is liking a dog. The dog barks. | A man likes a dog. It barks. |
| 3 | All men wait. | Every man waits. |
| 4 | No man is eating and drinking, nor waiting. | It is false that a man eats and that the man drinks, or that the man waits. |
| 5 | A card is valid. | A card is valid. |

Figure 4.3: Natural English - ACE testing pairs

## 4.3 Evaluation

Evaluation of the resulting models was performed using cosine similarity between vector representations of the translated sentences and the manually created ACE translation of the test sentence. Different vectorization schemes were used for the data to eliminate any bias the Keras encoder might have; As can be seen in the included notebook, the final epoch's accuracy is consistently and significantly higher with each concurrent iteration. As will become apparent in the results, however, this increase in accuracy with each iteration is not indicative of actual results. As such, several vector encoding schemes were used: Doc2Vec, Sentence-BERT, InferSent, and the Universal Sentence Encoder. The implementations of these vectorization schemes were taken from a tutorial [11]. Finally, the vectorizations of each iteration's translated output was compared to the vectorization of the model output via cosine similarity.

[5]Due to computing limitations the code used for the final experiment only uses the first 5000 lines of this dataset. Theoretically it could perform much better if the entire dataset were used, or this dataset were larger.

### 4.3.1 Doc2Vec

Doc2Vec is a so-called 'bag-of-words' encoding model based on Word2Vec. Word2Vec functions by assigning a value to each word that is representative of its meaning [17]. At the time of its creation it was revolutionary for being one of the first practically applicable models using said value to not only differentiate between words, but quantify semantic similarity between words. It does this with two separate algorithms: Continuous Bag-of-Words (CBOW) Model, which predicts a word from its context, and the Skip-Gram model, which predicts context from a given word[23]. Doc2Vec uses the CBOW model with one critical addition; a document id[6]. This additional id acts as a representation of the document's meaning as a whole, and is included in model training.

### 4.3.2 SentenceBERT

As the name implies, SentenceBERT (also known as SBERT) is an application of the pretrained BERT network discussed earlier that derives semantic-encoding sentence embeddings. Unlike Doc2Vec it does not take the entire input or single sentence as context for deriving semantic meaning, only siamese and triplet network structures[21]. This is its largest advantage over Doc2Vec; using the vast pretrained BERT model will make the semantic equivalences far more meaningful than the Doc2Vec set, which will only be trained with the input sentence, correct output sentence, and all generated output sentences.

### 4.3.3 InferSent

InferSent is a sentence embedding system created by a Facebook AI research team in 2017 [6]. It is trained with the Stanford Natural Language Inference (SNLI) dataset, which consist of 570k human-generated English sentence pairs that have been manually labeled as entailment, contradiction, or neutral. It's this labeling of inference that makes any system trained on it capable of learning sentence representations that capture it.

### 4.3.4 Universal Sentence Encoder

The Universal Sentence Encoder is another system that attempts to capture semantic similarity in it's vectorizations [5]. It's trained on a variety of sentence to paragraph length texts with the express goal of creating a very flexible system.

[6]Also called the paragraph id or sentence id. In this case, sentence id would be the most accurate name as only one sentence will be input at a time.

Furthermore, it's designed specifically for measuring semantic textual similarity, making it well-suited for the purpose of testing a translation.

## 4.4 Results

As expected given the translation results, none of the text vectorizations show any improvement in cosine similarity beyond a reasonable margin of error. However, the graphs do all dip at similar points. While the results themselves may not be encouraging, this does prove that a 'bad' translation registers as such for all four vector encoding systems, and thus that they are all valid measures of fitness.

Notable is that the Doc2Vec method appears to have the largest variance, which could be attributed to its highly meaning-influenced encoding scheme. However, other experiments[7] did not show such great variance and as such it can be assumed to be an anomaly. InferSent, on the other hand, shows relatively little variance from one translation to the other. In fact, if there were no other data and the variance were slightly lower one might be tempted to conclude there was a measurable increase in accuracy.

Finally we compare the average scores of all encoding schemes to evaluate the schemes themselves. As predicted, InferSent shows by far the lowest amount of noise, so little in fact it insignificant improvement over the iterations seem almost significant. However, the lack of any visible (let alone statistically significant) improvement proves that this is an anomaly of InferSent, vindicating the decision to use several vectorization schemes.

[7]Earlier trials of the same experiment, to be precise. They have not been included in this paper because the results were inaccurate due to coding errors, but this is not believed to be of influence on the anomalous variance of Doc2Vec.

| | Sentence 1 | Sentence 2 | Sentence 3 | Sentence 4 | Sentence 5 |
|---|---|---|---|---|---|
| 0 | a car a car | is a a | they all here. | no one one a car | she's a man. |
| 1 | what a book. | he has a a lot. | all all all | what no one no gun. | this a man. |
| 2 | she's a lot. | he's a car a car | they all | no is a one | |
| 3 | a a car | my a car a car | they all all | no one a one | this a |
| 4 | a car a lot. | a a car | they all it. | he is a lot. | this a |
| 5 | a car a lot. | a one a car | they all here. | no one one it? | he's a |
| 6 | a car a car | a car a car car car a car car | they all here. | no one one one dog. | this a |
| 7 | what a lot. | what a car a car | they all here. | what is is it? | he's a |
| 8 | a a car a friend. | a a car a car a curse. | all all here. | he is no to | this a man. |
| 9 | what a lot. | a one a car is a lot. | all all here. | what is no one | this a |
| 10 | that's a lot. | is a car | are they here. | no is no one | this a |
| 11 | he's a car | a a car a car | they all here. | what is is no dog. | he's a pen. |
| 12 | that's a book. | i'm has a dog. | they all go. | he is my dog. | what a joke. |
| 13 | he a car | i'm a car a dog. | they all here. | he is at it? | what a twin. |
| 14 | what a a book. | he's a a book. | they all here. | it's too my dog. | is a pen. |
| 15 | he's a a lot. | i'm a one on no dog. | they all now. | he is no joke. | this a pen. |
| 16 | what a doctor. | i'm a one on | they all here. | i'm on on my lot. | is a man. |
| 17 | a car a lot. | he's a my lot. | they all go. | he's my my lot. | what a what |
| 18 | he's a car | i is a lot. | they all won. | i no one on don't don't don't me. | what a man. |
| 19 | a one car a man. | no a a gun. | they all go. | he is a job. | this a man. |
| 20 | what a lot. | he is a lot. | they all go. | he is my dog. | this is a man. |
| 21 | what a book. | tom a one it's a lot. | they all it. | tom is no no job. | what a book. |
| 22 | what a doctor. | he is a dog. | they all here. | he is my my job. | is a book. |
| 23 | what a a dog. | tom a a twin. | they all go. | he is no to it? | is a twin. |
| 24 | he's a man. | i'm a a man. | they all go. | he has no dog. | this a man. |
| 25 | what a car | he is a dog. | they all go. | he is my job. | what a man. |
| 26 | a car a pen. | he a car a lot. | they all here. | he is my job. | is a man. |
| 27 | what a pen. | he a a book. | they all it. | he is my my job. | what a pen. |
| 28 | a car a car | he a a dog. | they all go. | he is my my dog. | is a man. |
| 29 | what a a gun. | i'm a one car. | they all here. | no one no door. | is a pen. |
| 30 | what a a man. | i'm my a job. | they all away. | he is my job. | is a man. |
| 31 | he's a a dog. | tom a car a car | they all here. | no one my dog. | what a a man. |
| 32 | he's a lot. | he is a dog. | they all here. | he is no to he to | this a man. |
| 33 | he's a a job. | no a a joke. | they all go. | don't all all it? | this a pen. |
| 34 | that's a a dog. | i'm a my car | they all go. | it's no my dog. | this a twin. |
| 35 | a car a dog. | a one on a job. | they all here. | is it my dog. | is a joke. |
| 36 | what a a lot. | a one one my car | they all here. | it is my my job. | this is a man. |
| 37 | what a a book. | no a one one my job. | look all go. | no one is us. | this a poet. |
| 38 | that's a gun. | a one is a lot. | they all here. | it on my dog. | is a a twin. |
| 39 | it's a man. | what a job. | they all go. | it is my job. | this is a man. |
| 40 | what a a dog. | a one a dog. | they all here. | he is to now. | is a man. |
| 41 | he's a lot. | what a a man. | they all up. | he is my dog. | is a man. |
| 42 | a one one dog. | a one a lot. | they all go. | all on it? | a a dog. |
| 43 | a one car | a one a job. | they all now. | he one my job. | this a man. |

Figure 4.4: Translation output per iteration, as per the notebook. Recall that iteration 0 only uses the pre-training.

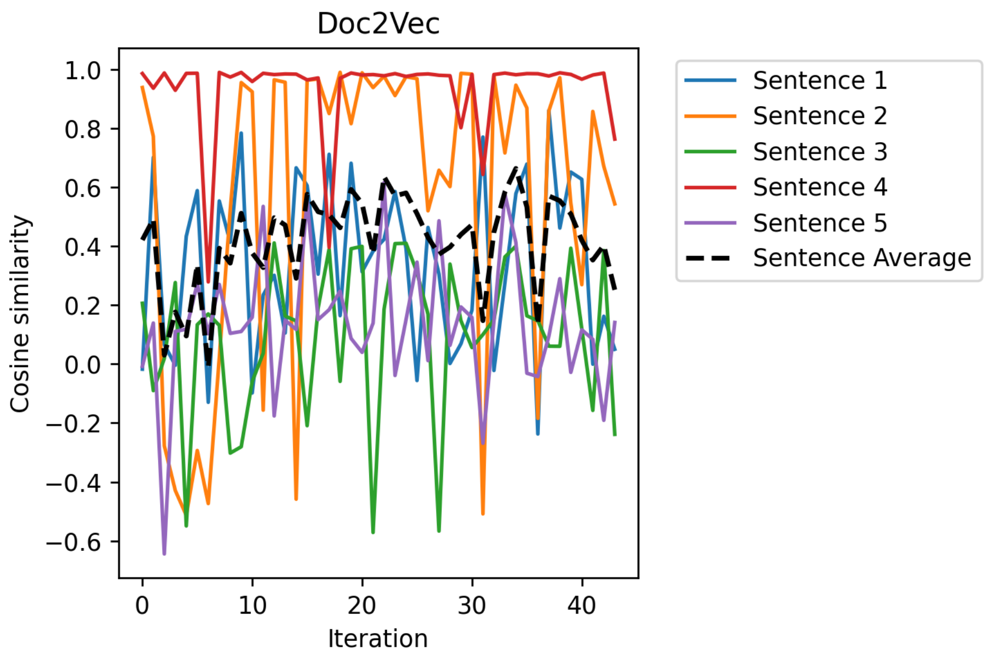


Figure 4.5: Doc2Vec results

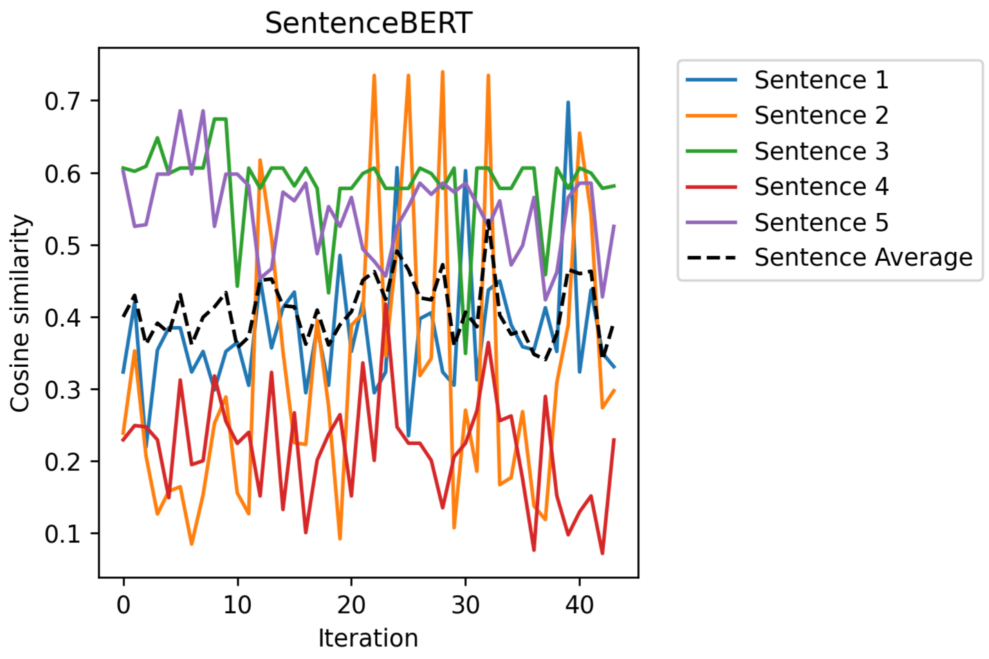


Figure 4.6: SentenceBERT results

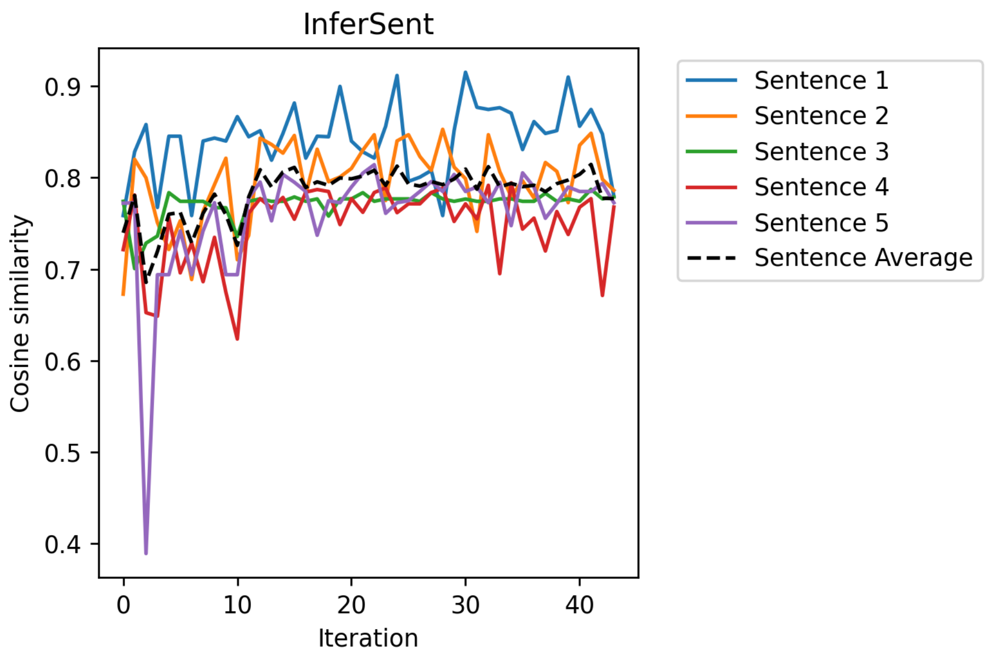


Figure 4.7: InferSent results

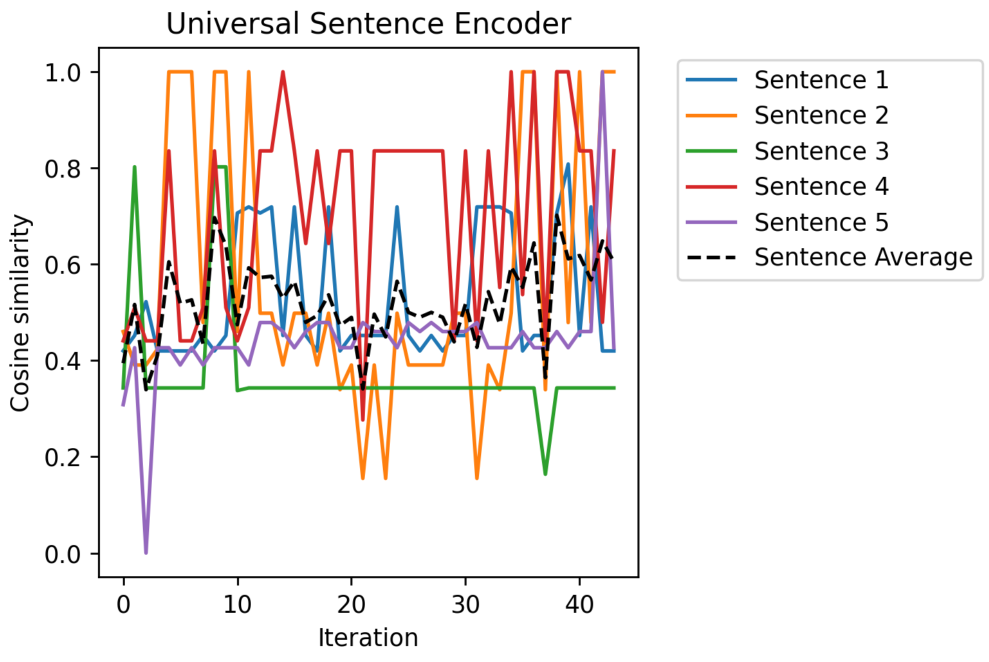


Figure 4.8: Universal Sentence Encoder results

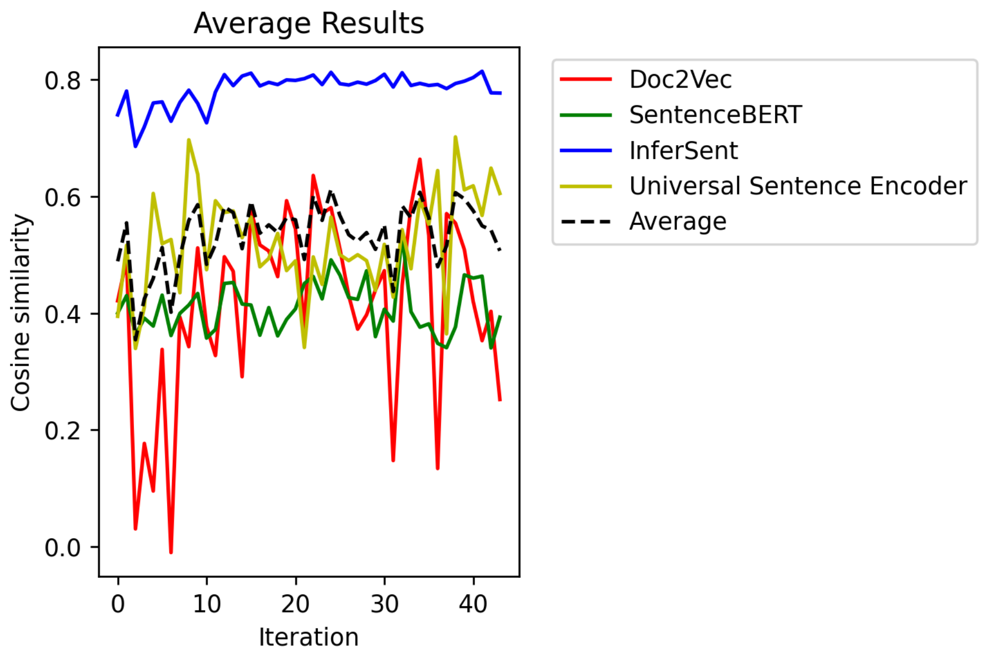


Figure 4.9: Comparisons of average scores per vectorization scheme

# Chapter 5

# Discussion & Conclusion

## 5.1 On the implemented method

It should be noted that the verbatim code from the tutorial[1] from the tutorial taken as basis for this work does allow a user to actually input a test sentence, nor does it randomly generate a sentence to be translated. Instead, it randomly selects an input sentence from the encoded training set, and translates it using the model. That is how the results appear flawless in most runs of the system. After being modified to re-encode any given string/test input, it no longer functions nearly as well. Furthermore, even with a large dataset with identical inputs and outputs, such a system was unable to produce a direct copy of the input sentence. This is relevant because, as discussed, simply copying the natural English input given is already half of the way towards a correct ACE compliant and semantically equivalent translation.

All of this means that, in hindsight, the chosen tutorial may not have been the best source of a frame upon which to construct our system. With this in mind, the included iPython Notebook contains a great deal of documentation to make swapping out said system with another one (while retaining the iterative testing and analysis framework) as simple as possible for any future research.

## 5.2 Continued work

With further feedback training (and perhaps a greater training set to start with) a framework like this has the potential to become the de facto automated reasoning

[1]Not precisely verbatim; the code neglects to define a certain variable. Fortunately a comment below the post provides the proper definition, should anyone wish to reproduce the system as presented on the site.

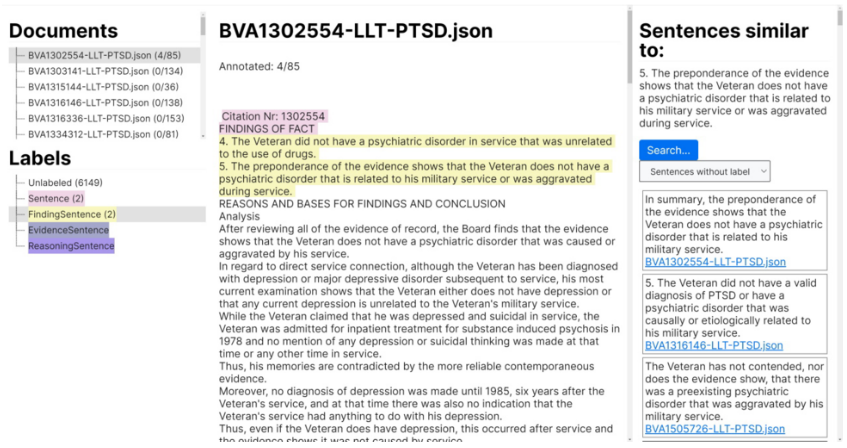
system. Only further research will be able to determine whether it is practically possible, not just theoretically possible. As mentioned earlier, with the current architecture it would either require hundreds of man hours to create a sufficiently large dataset, or the creation of a system that can automatically create natural English-ACE pairs, which would also need to be trained.

From a theoretical point of view, further research is needed to fully prove paraphrasing systems like CAESAR are more suited to this type of task than translation systems. Even if the aforementioned dataset creation problem could be addressed, paraphrasing systems are by definition based on semantic equivalence, rather than syntactic equivalence[2]. This makes them not only easier to train with a smaller dataset as semantic information is more dense, but also makes them more suited to translate to a subset of English.

## 5.3 Potential Applications

Though creating a direct bridge from natural English to first order logic has thusfar been a largely academic exercise, there are several practical applications of such a system once completed. One of them became apparent while studying CAESAR, namely automatic text processing. Specifically, taking a step from deriving meaning from legal texts by comparing them to semantically equal but syntactically different texts to translating them to ACE to derive a true meaning. Granted, both the translation system and ACE itself have a long way to come before either can handle such a complex subject, but if past developments are any indication the potential of ACE is limitless. One can imagine an intern using an interface not unlike the one currently used by CAESAR (see figure 5.1) to reduce a thousand pages of legal documents down to a few pages of 'ACE+' compliant sentences. Such an approach could, in the long run, prove more effective and more feasible than any attempt to translate natural language directly into logic.

[2]Or, to be more accurate, they create semantic equivalence by learning from syntactic equivalence.



**Documents**

- BVA1302554-LLT-PTSD.json (4/85)
- BVA1303141-LLT-PTSD.json (0/134)
- BVA1315144-LLT-PTSD.json (0/36)
- BVA1316146-LLT-PTSD.json (0/138)
- BVA1316336-LLT-PTSD.json (0/153)
- BVA1334312-LLT-PTSD.json (0/81)

**Labels**

- Unlabeled (6149)
- Sentence (2)
- FindingSentence (2)
- EvidenceSentence
- ReasoningSentence

**BVA1302554-LLT-PTSD.json**

Annotated: 4/85

Citation Nr: 1302554
FINDINGS OF FACT
4. The Veteran did not have a psychiatric disorder in service that was unrelated to the use of drugs.
5. The preponderance of the evidence shows that the Veteran does not have a psychiatric disorder that is related to his military service or was aggravated during service.
REASONS AND BASES FOR FINDINGS AND CONCLUSION
Analysis
After reviewing all of the evidence of record, the Board finds that the evidence shows that the Veteran does not have a psychiatric disorder that was caused or aggravated by his service.
In regard to direct service connection, although the Veteran has been diagnosed with depression or major depressive disorder subsequent to service, his most current examination shows that the Veteran either does not have depression or that any current depression is unrelated to the Veteran's military service.
While the Veteran claimed that he was depressed and suicidal in service, the Veteran was admitted for inpatient treatment for substance induced psychosis in 1978 and no mention of any depression or suicidal thinking was made at that time or any other time in service.
Thus, his memories are contradicted by the more reliable contemporaneous evidence.
Moreover, no diagnosis of depression was made until 1985, six years after the Veteran's service, and at that time there was also no indication that the Veteran's service had anything to do with his depression.
Thus, even if the Veteran does have depression, this occurred after service and the evidence shows it was not caused by service.

**Sentences similar to:**

5. The preponderance of the evidence shows that the Veteran does not have a psychiatric disorder that is related to his military service or was aggravated during service.

In summary, the preponderance of the evidence shows that the Veteran does not have a psychiatric disorder that is related to his military service.
BVA1302554-LLT-PTSD.json

5. The Veteran did not have a valid diagnosis of PTSD or have a psychiatric disorder that was causally or etiologically related to his military service.
BVA1316146-LLT-PTSD.json

The Veteran has not contended, nor does the evidence show, that there was a preexisting psychiatric disorder that was aggravated by his military service.
BVA1505726-LLT-PTSD.json

Figure 5.1: Experimental CAESAR interface, image taken from paper [25].

# Bibliography

[1] *Aesop's Fables, Public Domain*. URL: `http://www.gutenberg.org/ebooks/28`.

[2] *Attempto Project Website*. URL: `http://attempto.ifi.uzh.ch/site/`.

[3] *Attempto/APE Github*. URL: `https://github.com/Attempto/APE`.

[4] Tom B. Brown et al. *Language Models are Few-Shot Learners*. 2020. arXiv: `2005.14165 [cs.CL]`.

[5] Daniel Cer et al. *Universal Sentence Encoder*. 2018. arXiv: `1803.11175 [cs.CL]`.

[6] Alexis Conneau et al. *Supervised Learning of Universal Sentence Representations from Natural Language Inference Data*. 2018. arXiv: `1705.02364 [cs.CL]`.

[7] Jacob Devlin et al. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[8] Norbert E Fuchs et al. "ACE can be described by itself". In: (2009).

[9] Norbert E. Fuchs, Kaarel Kaljurand, and Tobias Kuhn. "Attempto Controlled English for Knowledge Representation". In: *Reasoning Web: 4th International Summer School 2008, Venice, Italy, September 7-11, 2008, Tutorial Lectures*. Ed. by Cristina Baroglio et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 104–124. ISBN: 978-3-540-85658-0. DOI: `10.1007/978-3-540-85658-0_3`. URL: `https://doi.org/10.1007/978-3-540-85658-0_3`.

[10] Norbert E. Fuchs and Tobias Kuhn. *ACE 6.7 in a Nutshell*. URL: `http://attempto.ifi.uzh.ch/site/docs/ace/6.7/ace_nutshell.html`.

[11] Purva Huilgol. *4 Sentence Embedding Techniques using Python*. URL: `https://www.analyticsvidhya.com/blog/2020/08/top-4-sentence-embedding-techniques-using-python/`.

[12] *Keras homepage*. URL: `https://keras.io`.

[13] Guillaume Klein et al. *OpenNMT: Open-Source Toolkit for Neural Machine Translation.* 2017. arXiv: 1701.02810 [cs.CL].

[14] Tobias Kuhn. "The Controlled Natural Language of Randall Munroe's Thing Explainer". In: *CoRR* abs/1605.02457 (2016). arXiv: 1605.02457. URL: http://arxiv.org/abs/1605.02457.

[15] Mike Lewis et al. "Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension". In: *arXiv preprint arXiv:1910.13461* (2019).

[16] Usman Malik. *Python for NLP: Neural Machine Translation with Seq2Seq in Keras.* URL: https://stackabuse.com/python-for-nlp-neural-machine-translation-with-seq2seq-in-keras/.

[17] Tomas Mikolov et al. *Efficient Estimation of Word Representations in Vector Space.* 2013. arXiv: 1301.3781 [cs.CL].

[18] *Online APE Client.* URL: http://attempto.ifi.uzh.ch/ape/.

[19] *OpenAI API.* URL: https://beta.openai.com/.

[20] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. *GloVe: Global Vectors for Word Representation.* URL: https://nlp.stanford.edu/projects/glove/.

[21] Nils Reimers and Iryna Gurevych. *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks.* 2019. arXiv: 1908.10084 [cs.CL].

[22] Shakeel Ahmad Sheikh. *OpenNMT-py-BERT GitHub.* URL: https://github.com/shakeel608/OpenNMT-py-with-BERT.

[23] Gidi Shperber. *A gentle introduction to Doc2Vec.* 2017. URL: https://medium.com/wisio/a-gentle-introduction-to-doc2vec-db3e8c0cce5e.

[24] *Tab-delimited Bilingual Sentence Pairs, Tatoeba Project.* URL: http://www.manythings.org/anki/.

[25] Hannes Westermann et al. "Sentence Embeddings and High-Speed Similarity Search for Fast Computer Assisted Annotation of Legal Documents". In: *Legal Knowledge and Information Systems* 334 (2020), pp. 164–173. DOI: 10.3233/FAIA200860. URL: http://ebooks.iospress.nl/volumearticle/56173.