Synthetic Dataset Creation for Natural Language Processing in an Email Personal Assistant

Tayyab Hussain

Supervisor: Dr Mohan Sridharan

Abstract

When attempting to train a Neural Network for a specified natural language task it can be difficult to find or develop a dataset. An example of this is in the problem of an e-mail personal assistant that maps natural language commands to specific discrete actions. A Word2Vec approach is optimal as it encodes meaning of words in a vector format, which is easy to process for a neural networks. This approach shows more generality than singular word learning approaches and allows for clever methods of dataset synthesis. A new technique to generate larger datasets using Word2Vec word similarity is used to synthesise artificial sets for the Neural Net to train on and shows very good results in both model accuracy and generality. This algorithm can take a dataset of only 30 samples and extend it to over 1000 samples for better training.

Contents

1 Introduction

1.1 The problem

Personal assistants can be very powerful at helping to automate people's lives. Often times, when using traditional assistants, normal and natural language may lead to assistants misunderstanding the input, and therefore giving an incorrect output. In this case, a very basic email personal assistant will be taken as a test use case for the methods that will be discussed.

The email personal assistant will take an English phrase such as *"What was my last email"* as input. Then from this phrase alone, determine what to do. For this phrase, the body of the latest email sent to the inbox should be returned. The main goal is to create a system that can map a natural language command to a discrete action for the user. There are many factors to consider when looking at this problem. The first of which is how to encode the natural language input. Natural language, by definition is not clean and therefore it is important to create a system that can encode the data in such a way that there is some generality in the solution for different words. An example of this is the phrase - *"Read my last email"*.

This phrase can be expressed in many ways with many different possible words, one of which being *"Show me my latest email message"*.

These two phrases mean the same thing and should output the same action, however they are written with different words. For this reason the *meanings* behind words are very important to encode for this sort of problem.

An exhaustive list of all possible words is not feasible therefore this system must be able to generally deal with words that have the same meaning or at least mean somewhat similar things. Intent and meaning encoding is pivotal for this sort of problem.

After the natural language data has been encoded, processing the data is the next step. The data must be mapped to an action. This is a classification problem. Each phrase can be given an output class and the problem is to map these phrases to one of the given classes.

1.2 Challenges

The first major challenge will be pre processing the data and encoding phrases. Sentences in the English language can be of variable length and contain punctuation. This will have to be accounted for before converting the language into something meaningful for our model. Then finding an embedding that's useful to be able to use it and find mappings is the second major problem. The embedding to be chosen must be able to somehow encode meaning and intent to allow for strong and distinct mappings to the output classes. Multiple words with the same meaning must be accounted for, without using an exhaustive list of data as this is extremely inefficient and infeasible. If meaning is encoded well, our model should be able to somehow map words of similar meaning together and to the same class.

The main challenge for this assistant, however is to find a dataset to train the model. No dataset specifically exists for this problem, therefore one will have to be developed. Manually inputting the data is long and time consuming. User submitted data isn't guaranteed to be balanced or correct and thus will have to be manually pruned. Therefore, an ideal dataset for the assistant is one that is created automatically, is balanced, encodes meaning well, has plenty of samples to train on and requires little to no human pruning or editing.

This is the main problem to be solved.

1.3 Importance of the Solution

Currently, there is no sound method of creating artificial datasets for specific natural language tasks. Specific training data for problems are often unavailable and must be collected by other means. If this dataset synthesis problem is solved, it could allow for many more NLP based specific problems to be solved. An efficient synthetic dataset to train on is incredibly useful for any NLP task that requires a dataset to train on. Its also useful for problems where more sample data is required, or more generality is needed as more data can be generated for the model to train on. Therefore, this could lead to better performance and efficiency on these tasks.

Historically, many language processing tasks lack the data to train on in the first place. This could be used to theoretically create datasets for any possible language task and therefore could lead to more general NLP applications being developed to do a more wide variety of tasks as a whole.

2 Literature Review

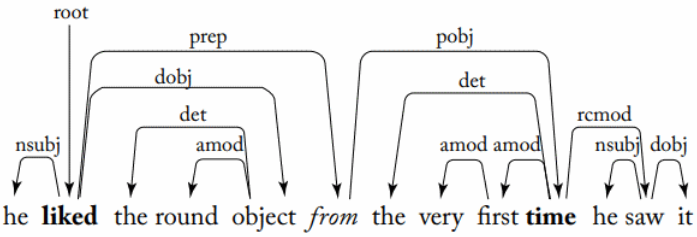
2.1 Breaking Down Natural Language

Attempting to analyze natural language using computers has a very well researched area due to the difficulty but also the widespread applications

of it. Many different techniques have been developed to parse, process, analyse and "understand" language input.

This is no easy task. Natural language doesn't follow rules that are rigid, there are often many exceptions and, depending on context, grammar and spelling can change too. A good example is the sentence *"I ate pizza with friends"*. This sentence can also be expressed as *"friends and I shared Pizza"*.[Gol17] A robust system would be expected to be able to "understand" that these have the same meaning. Language also changes over time, therefore any model that currently works will need to be kept updated.

One potential starting point for understanding language and breaking it down to its constituent parts using dependency trees. This is when a sentence is split into subjects, verbs, determiner...etc. Finding the relationships between the order of these words can help to find the root and the subject of sentences, as shown in the diagram below: [KMN09]



Order matters when looking at language. Take the sentence *"Live to work"* and the sentence *"Work to Live"*. These two sentences use the exact same words however put in a different order. This drastically affects the meaning of the sentence. This means that whichever method of encoding is chosen, it must, at least somewhat, have a way of distinguishing the order of words.

2.2 Neural Networks

Currently, the most popular method for dealing with natural language is through the use of neural networks and deep learning. [Gol17] Multi-Layered Feed Forward Perceptrons have shown fantastic results in sentiment analysis and even in basic question and answering tasks. [ZZHC15]. Zhou et al were able to demonstrate fantastic parsers of the English language however these were trained on an enormous dataset of words. the English Penn Treebank consists of 1,203,648 word-level tokens in 49,191 sentence-level tokens. [AB]. These giant datasets of the English language are very common and provide fantastic sets for training general purpose language parsers however for specific NLP tasks, large amounts of specific data is required. This is much harder to find.

The actual architecture of the Neural Network is extremely important to model behaviour. Standard recurrent Neural Networks have been the norm for natural language processing however recently, the transformer model has shown incredibly strong results for analysing, parsing and responding to natural language data. This is due to the attention layers. [VSP+17] These "Attention layers" are extremely good at semantically breaking down language structure due to their architecture and vastly outperform traditional RNNs. Using transformers to encode data could be an efficient potential embedding for the Email personal assistant problem since it efficiently tokenises any given word. One potential downside of this is that pretrained transformer models are very large and not created for specific tasks therefore may not actually be the most efficient potential way of encoding the data for a NN to learn from.

3 Methodology

3.1 Formalising the Problem

The goal is to create an Email personal assistant. This assistant will take a natural language string as input and the model must process this to give a number representing class as output. Each class will represent a discrete action for the agent to take, The three starting classes are to return the body of the latest email, to return the subject of the latest email and to return the attachments of the latest email. These three are a good basis for testing the model and can easily be expanded as shown later.

The problem then, is to create a neural network that can deal with this input and learn to

give the correct output. This then means that a dataset will be required to train the model on. No such dataset exists for those three specific classes, therefore one must be created. This is the key problem.

3.2 Language Encoding

3.2.1 Word2Vec

Word2Vec is a technique that converts all words in a corpus into a corresponding vector. This vector aims to encode *meaning*. This seems like a difficult task however vectors are very powerful. Word2Vec takes a large corpus of words and uses the data available in the corpus to classify them and split them as efficiently as possible in a large number of dimensions (300 in this case). [MCCD13]

Encoding data in this way has many uses and benefits. The first benefit is that vectors are values that can be added and subtracted. This means that some rudimentary "word arithmetic" can be done, such as (King - Man) + Woman gives Queen as the output. More importantly to us, it means that words that are closer together in the vector space are more *similar*. This is a key feature that is going to be used and is the crux of the solution.

Vectors are also a perfect data structure for use in a Neural Network and therefore reduces the amount of data changing and pre-processing that needs to be done and can almost directly be put as input into our neural networks.

A large corpus is required to to train the initial word2Vec model however. Ideally, this corpus should reflect natural language enough to produce accurate vectors but also be robust enough to have a large vocabulary. This can be a challenge as large corpuses of data can have file sizes that are too large for most hardware. Thankfully, there are many "pre-trained" word2Vec models that can be downloaded. The one chosen in this case was a model that was trained on Google News. This was because it had a comparatively low file size and was still very accurate as results show. Larger models that more accurately represent the task could theoretically produce even better results.

3.2.2 Pre-Processing and Training Data

As this is a classification problem, the data must be formulated such that it contains a training phrase as the input and a label as the output that represents the action class. Samples will be placed in a .csv file with an input column and a label column. This file is read and the data is split and parsed into a series of arrays. The data still contains stop words however which aren't contained within our Word2Vec model thus are filtered out. Finally our Word2Vec model converts these strings into a tensor of vectors. Here they are padded with zeroes to a maximum of 20 words. Since each sample contains a sentence of 20 words and a word of 300 vector dimensions our tensor has shape:

shape = (sampleCount, 20, 300)

Our labels are sliced but must be processed too. Since the intended output is a confidence value for every class the intended shape for all of the labels should be:

labels shape = (number of samples, number of classes)

The actual labels are purely class numbers in our CSV. This means each label must be changed into an array of 0s, with a 1 in the corresponding class. The reason for this is that it is difficult to calculate meaningful loss from simple numbered categories. For example, if the model predicted the output as class 1 when the output should have been class 3, the loss would be greater than if the correct output should have been class 2. In reality their loss should be based on the confidence values of the model for each class. For this categorical crossentropy is used. Categorical crossentropy is given by:

$$\text{Loss} = -\sum_{i=1}^{\text{output}} y_i \cdot \log \hat{y}_i$$

This loss compares every confidence value with the correct output (zeroes in incorrect classes and 1's in the correct classes). This also allows us to visually asses the performance of the model by looking at each output manually, to see how confident the model's predictions were.

3.3 Setting up the Neural Network

Once the dataset has been formatted and pre-processed correctly, it is the job of the neural network to process the data and attempt to learn to give predictions. To be able to learn efficiently, the NN was initialised with an input layer that took data of shape = (20, 300). This is then flattened to allow for feature extraction and processing. Our output is of shape = (No. of Classes) therefore the

input must be flattened to match the outputs 1D nature.

The NN is then a collection of Dense Layers that reduce in units over time to a final result. The initial hidden layer has 2000 units. This gets reduced at each hidden layer 15. Dropout layers were added as these have been shown to improve generalisation in the model. This technique turns off random units in the network to prevent overfitting of data.

3.4 Training Data and the Synthetic Dataset

3.4.1 Starting dataset

It is important that the dataset is balanced such that no class is favoured and a wide variety of examples are given. To begin with, 6 samples were manually inputted with correct training labels. These samples were manual chosen as a good starting place for the data. The starting email assistant has three possible classes representing possible actions. These are the following
1: Return the body of the latest email
2: Return the subject of the latest email
3: Return the attachments of the latest email
Since there are 6 samples for each of these classes, there are 18 total starting samples. These aren't anywhere near enough to produce a good generalisation on the data as the results show.

3.4.2 The Dataset Expansion Trick

To be able to generally classify natural natural language a large dataset of examples that correctly shows the features of the data must be available. The challenge, then, is to extract this from only the starting 18 samples.

Word2Vec offers a neat way to do this. As discussed, words that are similar are closer in vector distance. Therefore, the main feature for the neural network to learn is the approximate vector space positions of solutions for each class. This is useful for the solution. To expand the starting dataset, synthetic samples must be created by replacing words in every sample with other "close" similar words. This can be done by searching for the words with the lowest cosine distance in the Word2Vec vector space to essentially find the most similar words according to Word2Vec.

This replaces every non stop-word token with 10 new similar words to create 10 new samples from just the one word alone. as the following diagrams shows:

```
model.most_similar("email")

[('e_mail', 0.8479690551757812),
 ('emails', 0.6680365204811096),
 ('E_mail', 0.6678352355957031),
 ('Email', 0.6487499475479126),
 ('roycimagala@hotmail.com', 0.6414469480514526),
 ('cclark###nd@netscape.net', 0.6410639882087708),
 ('email_info@unify.com', 0.6385294795036316),
 ('Stephen_Gallien', 0.6362109184265137),
 ('Website_www.citizen_news.org', 0.6310882568359375),
 ('Cathryn_Khalil', 0.6270006895065308)]
```

Fig. 1: Caption

Fig. 2: expanding the word "email"

One thing to note is that depending on the pretrained Word2Vec model that is used, there may be some garbage data. In this case, since it was trained on google news, there are some real email addresses given as the most similar words. This at first, may seem like a problem however this behaviour is exactly what is being sought after. The words themselves do not matter at all. They are just converted into their Word2Vec vector positions before being input into the network. All that actually matters in this case is the positions of each word. These "garbage" words help the network to generalise the vector positions that are being looked for as the feature for each class. It may be that the network generalises a range of vectors within the vector space in specific dimensions as the main separator, these "garbage" words only help to enhance that range of positions.

3.4.3 The Algorithm

Here is the algorithm for the dataset expansion:

Algorithm 1 Expanding the Starting Dataset

Ensure: *clean* ← 2d array of all training inputs with stopwords removed
Ensure: *model* ← Word2Vec pretrained model
  clean ← Clean Data
  dataset ← *EmptyArray*
  temp ← *EmptyArray*
  for Each i in clean **do**
    for Each j in i **do**
      for k from 0 TO 10 **do**
        word ← model.mostsimilar(j)[k][0]
        temp ← i
        temp[*i.indexOf*(*j*)] ← word
        dataset.append(*temp*)
      end for
    end for
  end for

This algorithm uses temporary arrays to store versions of an input that use alternate wording then adds that to the dataset. creating an expanded dataset. The idea behind this is to return a large number of possible variations within an accurate vector range for the model to learn from. This algorithm has the drawback that it has a time complexity of $O(n^3)$. This means that, it would take an extremely long time to execute this expansion on an already large database. Therefore it is recommended to be used on smaller sets or on high-end hardware.

3.5 New Hyperparameters

With this technique two new hyperparameters are born. These are known as the expansion ceiling and the expansion floor. When expanding a word into its most similar words. The upper limit of the words being checked and added is known as the expansion ceiling. This is equal to the number of new samples created from a single word if the expansion floor is 0. Tuning this number has consequences for the learning process and impacts performance. In theory, increasing this expansion ceiling will allow for much bigger synthetic datasets as more samples will be created. However, this isn't without drawbacks. Increasing the amount of words also increases the cosine distance threshold of each word too. Effectively, this increases the range of vectors in the vector space for that word. This will mean the neural network will look at words that are less similar for the class predictions. This means that increasing the expansion ceiling, decreases the accurate separability of the model. Decreasing the expansion ceiling will decrease the size of the synthetic dataset but will more accurately reinforce the vector position ranges that the neural network should be looking at for class predictions. The optimal value for the expansion ceiling is very problem and task dependant. Certain tasks will favour a higher ceiling such as those in which generality is favoured or those in which the largest dataset possible is requires. However, for very specific tasks, or tasks in which very specific features need to be reinforced, a smaller expansion ceiling will be favoured.

The expansion floor is the number of words, starting from the most similar word, to be omitted from the word expansion. For example in

the above "email" example, the four closest words were variations in spelling of the word "email". Also the cosine distances of the vectors for all of these words are very small. If these words were in the expansion, that may specify and reinforce those vector positions more so than intended. Increasing the expansion floor will decrease the number of samples created from a single word. However, this could be beneficial as the vector ranges are wider and less repetitive and therefore would, in theory, allow for more model generality. If the expansion floor is low or 0, then all of the closest words will be included and therefore more samples will be created that help to emphasise the vectors for them. This is best when the task requires the largest dataset possible or requires the vectors to be very specific.

The total samples created per word is given by:
Samples per Word $= Ceiling - Floor + 1$

The "+ 1" is due to the inclusion of the original sample. These hyperparameters will be tuned and different values will be tested to see which gives the best performance after training.

Another interesting side effect of increasing the expansion floor is that it reduces overfitting. Since the most similar words will have a very small cosine distance, its likely that there will be many samples that are the same. It is very easy for the network to be skewed in favour of these results and overfit the data. Increasing the expansion floor increases the width of range of the vectors being looked and creates samples of more variety.

3.6 Creating a Test Set

There are certain features that are important for this model to learn to allow it to classify correctly, The first, and perhaps the easiest to learn, are key words and phrases. The model must recognise when certain key words are used, it means that the output must be for a specific corresponding class. These are easy to train for as any key words that required for the AI to classify properly should be put in the training set. The test set should reflect this by having standard examples of phrases the model *should* get correct. Secondly, perhaps less importantly for this particular problem, is the structure of the language. The order of language matters greatly. However, in this particular problem, since the sentences are small, and the stopwords are being removed, its not as relevant. Finally, an ideal model can perform well on unseen data. A good test of the dataset expansion trick will be to have words that aren't in the original dataset but are in the expanded one. and then also have words that aren't in either to really test the generality of learning using this dataset expansion technique. Below a subset of the testing data is shown. Each class has 15 samples that test all of the above.

input	label
return the latest email	1
return the latest message	1
return the last email	1
return the last message	1
return the latest email body	1
what is the latest email	1
what is the latest message	1
what is the last email	1
what is the last message	1
what is the latest email body	1
I need the latest email	1
I need the latest message	1
I need the last message	1
I need the latest email body	1
I need the latest email	1

Fig. 3: Class 1's Test Set

Three different sentence types are used for each class to test the models ability to deal with different sentence structures. for classes 2 and 3: Of each 5, 3 words that are in the training set. 2 are not. Class 1 is a control set and the model is expected to get close to 100% as they are in the training sets. A good performance for this dataset is then around 70% or greater. With any score greater than 77% being excellent as it is able to handle outliers in language and unseen language to achieve this.

4 Results

4.1 The Validation Set

When deciding the split ratio of training to validation sets, many values were tried. Below are the accuracy results of various training-validation splits tested on the validation set:

```
nnmodel.evaluate(x=x_valid, y=y_valid)
18/18 [==============================] - 0s 8ms/step - loss: 0.0477 - accuracy: 0.9893
[0.047665733844041824, 0.9892857074737549]
```

```
nnmodel.evaluate(x=x_valid, y=y_valid)
14/14 [==============================] - 0s 8ms/step - loss: 0.0348 - accuracy: 0.9933
[0.034827541559934616, 0.9933035969734192]
```

```
nnmodel.evaluate(x=x_valid, y=y_valid)
11/11 [==============================] - 0s 7ms/step - loss: 0.0353 - accuracy: 1.0000
[0.03534436970949173, 1.0]
```

Fig. 4: Training-Validation Split - 50:50, 60:40 and 70:30 respectively

The model performs extremely well on the various validation sets. This is to be expected as the initial dataset is expanded into a dataset with many similar inputs. Therefore the data is still present in those similar examples therefore it is easy to achieve good results. Figure 4 shows that even with half of the data taken away, the model is still able to achieve an accuracy of 98.9%. Any ratio greater than 70:30 leads to an accuracy of 100% most of the time. This means that the validation set doesn't really allow us to get a feel for model performance and there will have to be a custom made test set to test the model.

4.2 The Test Set

Using the test set discussed earlier, more meaningful results can be found. This test set consists of words both in the training set and the expanded set, and also words in the initial set, as well as words in neither. This really tests the limits of the dataset expansion using Word2Vec and gives insight into the vector ranges and separability of the model. This set was used as a benchmark for all the hyperparameters to be tested against. Below is the full test set:

input	label
return the latest email	1
return the latest message	1
return the last email	1
return the last message	1
return the latest email body	1
what is the latest email	1
what is the latest message	1
what is the last email	1
what is the last message	1
what is the latest email body	1
I need the latest email	1
I need the latest message	1
I need the last message	1
I need the latest email body	1
I need the latest email	1
return the subject of latest email	2
return the subject of last email	2
return the topic of last email	2
what is the the latest email concerning	2
what is the title of the last email	2
I need the subject of latest email	2
I need the subject of last email	2
I need the topic of last email	2
I need the topic the latest email is concerning	2
I need the title of the last email	2
Give me the subject of latest email	2
Give me the subject of last email	2
Give me the topic of last email	2
Give me the concern of the latest email	2
Give me the title of the last email	2
I need the Attachments of the latest email	3
I need the files attached	3
I need the last files	3
I need the latest links	3
I need the latest documents	3
Give me the attachments of the latest email	3
Give me the files attached	3
Give me the last files	3
Give me the latest links	3
Give me the latest documents	3
what are the attachments of the latest email	3
what are the files attached	3
what are the last files	3
what are the latest links	3
what are the latest documents	3

Fig. 5: The full test set

This set has been carefully chosen to have very balanced data. Each class has exactly 15 samples and the examples are chosen according to the initial dataset. Certain words were chosen as they are similar enough to be used in everyday language however are not in the dataset. Therefore the model will have to use the information gained from dataset expansion to infer the class of these examples. Any result above 70% will be considered good and any result above 77% is considered excellent.

4.3 Optimal Hyperparameters

4.4 Initial training set

The initial training set is vitally important for general performance on the model. When using an initial training set with 6 samples per class, and the data encoding the main sort of answers the model is likely to see, the results were promising. The model was trained on the expanded dataset with a learning rate of 0.00001 and a batch size of 128 (see Batch Size Section), through 100 epochs of training. The following shows the loss-accuracy graph of the training:

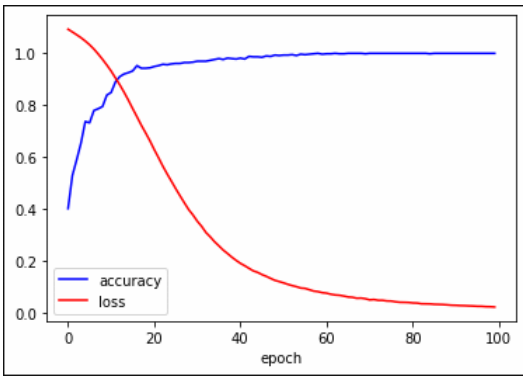


Fig. 6: Loss-Accuracy graph of small training set

The loss seems to plateau towards the end but still seems to be declining thus more training could potentially marginally increase the performance on the test set. This initial data was able to yield 73% on the test set which is surprising given there were only 18 samples in the initial training. The expansion of this data helped to reach a very decent result on the test set.

A Second training set was created as a test to observe the effect of the initial data on the expansion and model generalisation. Instead of 6 samples per class, the new initial training set has 10 samples per class, so 30 samples in total. This set contains a much wider array of wording and vocabulary.

The same hyperparameter were used in training and below is the loss-Accuracy graph of the model from training on the expanded larger dataset:

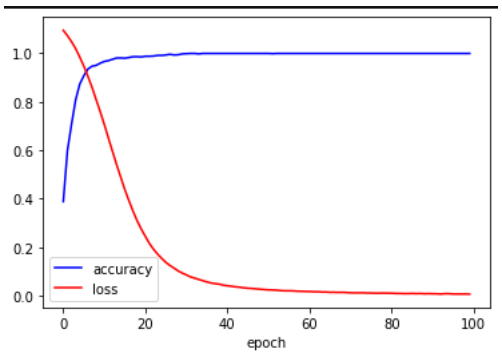


Fig. 7: Accuracy-Loss graph of training on the bigger dataset

The accuracy of this graph is much smoother and plateaus much faster. Interestingly, the new dataset has an accuracy of only 64% on the test set.This is significantly worse than the smaller initial dataset. This could perhaps be due to overfitting, or due to the hyper parameters not matching the needs of the bigger training data, Since the data looked as if it was overfitting quickly, a smaller learning rate and more epochs of training were tried to see if it was possible for the model to achieve the same results as the smaller one. A learning rate of 0.000001 through 200 epochs of training on a batch size of 64 was attempted and below is the graph:

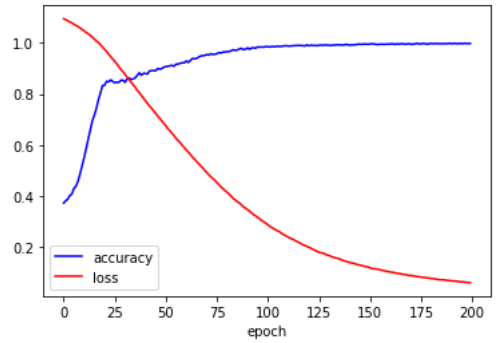


Fig. 8: Accuracy-Loss Graph of the lower learning rate training

This looks as if it was able to generalise much better as the loss only starts to plateau towards the end just as the first result did. The accuracy on the test set was equally good, with an accuracy of 73.3%.

4.4.1 Batch Size

Tuning hyperparameters in the model made a very large difference to the accuracy of it. Therefore optimal hyperparameters are incredibly important. The first of which is batch size. Increasing the batch size allows for the better use of parallelism of the processor being used. This leads to "faster" convergence however, higher batch sizes often lead to poor generalisation. [SKL17]. Smaller batch sizes improve performance and generalisation however, do require a large training time. To find the optimal batch size, the model was trained using batch sizes of 2,4,8,16,32,64 and 128 and their accuracy was measured against the test set. The following results were with a learning rate of 0.00001, an expansion ceiling of 10, an expansion floor of 0 after 50 epochs of training on the synthetic dataset:

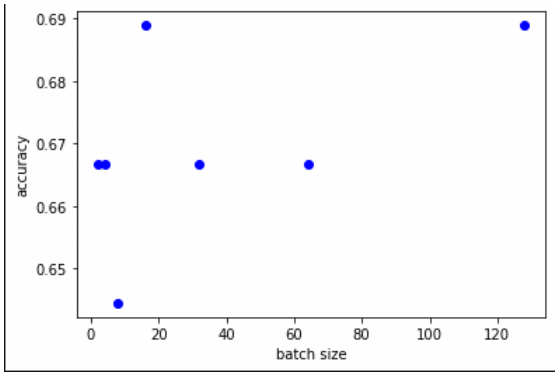


Fig. 9: Batch Size Accuracies

Interestingly, the changes to the test set accuracy with different patch sizes are negligible and all within the 67% - 69% region. Therefore, its optimal to keep the batch size as high as possible while maintaining a good test accuracy, in this case, that would be 128. at a batch size of 256 the test set accuracy begins to start falling thus, 128 is optimal.

4.4.2 Learning Rate

Tuning the learning rate can be important to finding optimal solutions. If the learning rate is too high, local optima can be missed when attempting to optimise the weights in the model. A higher learning rate leads to "quicker" training and convergence (if it converges at all) but may not give the best solutions. A smaller learning rate can lead to much better convergence [Rud16]. Below is a graph showing accuracy on the test set when using different learning rates:

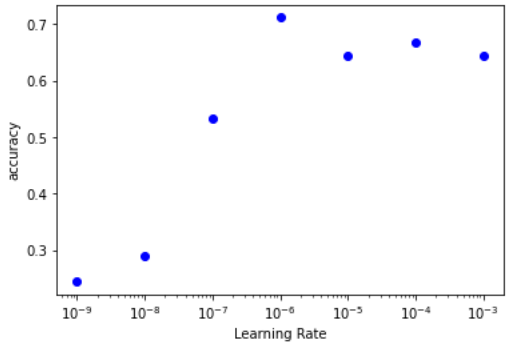


Fig. 10: Learning rates

The results show that with a high learning rate, the performance is fairly average, and as it gets smaller the performance improves up until its peak. After this the accuracy dramatically decreases. This is likely due to the fact that the learning rate is so small that very many more epochs are required to get the best results. Using purely the pre-dip data, if the trend continued and there were enough epochs to train, performance would likely improve further. To test this a learning rate value of 10^{-8} was used with a batch size of 128 and 500 epochs of training. This scored an accuracy of 68.8%.

4.4.3 Expansion Ceiling

This is a hyperparameter for the dataset expansion. The expansion ceiling effectively determines how wide the word similarities will be for the dataset expansion. The higher the ceiling, the greater the range will be of word vectors that the model is looking for to help classify. Ceilings of both 6 and 10 were tested, both with floors of 0. The model had a batch size of 128 through 100 epochs of training using a learning rate of 0.00001. With an expansion of Ceiling 6 an initial dataset of 30 samples was expanded to be 672 samples large. The accuracy on the test set was 71.1%. With a Ceiling of 10, the 30 samples were expanded to 1120. The other hyperparameters remained the same and the resulting accuracy was 66.7%.

4.4.4 Expansion Floor

The expansion floor is also a hyperparameter specifically for the dataset expansion. The floor determines which of the most similar words to omit from the expansion. Since naturally, for certain problems, the most similar words are important, this parameter will change depending on the intention of the data. If the problem requires more dynamic and general models then increasing the expansion floor could yield better results. With a ceiling of 10 a floor of 0 and a floor of 4, the model was trained. A floor of 4 expanded 30 samples to 672. The accuracy on the test set was 75.6%. The accuracy of a floor of 0 was 66.7%.

4.5 Test Set Expansion

A possibly interesting interaction would be to test how well the model would perform on a test set that had also been expanded. The test set was expanded from 45 samples to 1600 samples. While the model achieved a 73.3% on the test set before expansion. The expanded test set yielded an accuracy of 72%. This is a negligibly lower percentage and this is expected.

4.6 Interesting Examples

There are many interesting specific results from both the training data and the testing data. The first of which is shown below:

```
['What', 'last', 'email', 'concern']
[[2.2413826e-04 9.9971431e-01 6.1456121e-05]]
Class Number:  2
Email Output:
'Your Amazon.co.uk order of "Indeme A5 Notebook/Notepad..." has been dispatched'
```

Fig. 11: The word "Concern"

The word *Concern* doesn't appear in either the training set nor the expanded training set yet the model was very confident in the correct class being output (Class 2 - The email subject). This is due to its position in the vector space being very close to some other words in the training set such as the word *Concerning*. This is shows that the dataset expansion is having the desired effect on the model.

Another interesting example is with the word *pics*. The word doesn't appear in the initial dataset however, does appear in the expanded dataset. This allows the model to classify sentences like these correctly:

```
['Show', 'latest', 'pics']
[[0.38536903 0.08511478 0.5295162 ]]
Class Number:  3
Email Output:
```

Fig. 12: The word "pics"

Finally, another interesting thing to note is the way the way an input is worded can drastically change the way the model classifies it as shown by the following examples:

```
sentence = "return the topic"
['return', 'topic']
[[0.9146363  0.08253296 0.00283068]]
Class Number:  1
```

Fig. 13: Incorrect classification

Compared to the following example:

```
sentence = "return the topic of the last email"
['return', 'topic', 'last', 'email']
[[0.04590047 0.93181175 0.02228781]]
Class Number:  2
Email Output:
'Your Amazon.co.uk order of "Indeme A5 Notebook/Notepad..." has been dispatched'
```

Fig. 14: Correct classification

5 Evaluation

5.1 Attacking The Problem

The problem of datasets for NLP isn't a simple one to solve. Natural language often bends rules, and isn't fixed or easy for a computer to be able to effectively process. The dataset expansion trick is one potential method to help improve performance in specific NLP tasks. The idea of synthetic datasets isn't one that is heavily explored like other areas of AI. Attacking problems like this can be difficult. The approach taken here aims to use the features of Word2Vec and the ability of Neural Networks to seperate, generalise and spot patterns. Other potential ways of attacking problems like these are by using more complex models than neural networks. Better, attention focused ways of expanding sentences and samples and Generally harnessing the power of hardware much more efficiently than what was done here. many new language models have even shown the power of *general* approaches to specific problems such as Open AI's GPT-3 [BMR+20].

5.2 Importance of the Initial Training Set

The initial training set lays the groundwork for the basis of the dataset expansion, and therefore the model. The models accuracy is directly linked with how good the training set is to a specific problem. In this case, the training set wasn't very large but was very balanced which allowed the model to be able to successfully classify samples from a difficult unseen test set. The importance of the training set is further highlighted by where the model failed. Figure 13 shows the model failing on a relatively easy sentence. If the example was better encoded in the initial dataset, it would have been better encoded in the expanded one too, and therefore the model would have correctly classified it. Not only does the initial data have to be balanced, the word choice is incredibly important too. Each word adds another vector position for the neural network to use to help it classify. Eventually the idea is that the neural network will have a range of vectors to separate solutions. The words that are chosen will expand the networks vector ranges, or reinforce one of the already established vector ranges. The best training set is one that can encapsulate most word choices for the given class, as well as the many ways of wording the problem. This is then expanded and the network can do the rest of the learning. The results showed some interesting examples where words that were not in the training set were correctly classified. *This is the point of using Word2Vec.* This is evidence of the dataset expansion allowing for the vector ranges to expand to similar-ish words, and therefore achieving the intended result. The model may not have perfect performance, especially on the difficult test set, but does what is intended and the expansion of the initial dataset has shown to be a promising technique.

5.3 Hyperparameters

5.3.1 The Learning Rate

Figure 10 shows how a learning rate of 10^{-6} *seems* to give the best result however this is not the whole story. The learning rate is not simple to tune and requires other hyper parameters to be tuned alongside it to find the optimal results. As the learning rate decreases, training becomes "slower". It takes more training time to achieve the best results and find the minima. In this case, the results seem to get worse as the learning rate decreased after this peak, however, it may be that these values are better. More training is just required. This could be from training with a lower batch size or through more epochs of training. Unfortunately, hardware and time constraints make it difficult to test all the possible combinations that allow for the optimal learning rate to be find - if it is possible at all. The rationale behind keeping the learning rate at 10^{-6} is that it seemed any other values changes were negligible anyway and that since that value showed the most promising results within the training time, it was fine to be chosen as an optimal result. More complex models and better hardware would almost definitely improve the results even more.

5.3.2 The Expansion Parameters

These parameters are interesting as they aren't traditional hyperparameters that one would expect to see, but do have a very large impact on the solution. Perhaps more interestingly, these are extremely problem specific. When testing, the Ceiling of 10 performed worse than the ceiling of 6. There could be many reasons for this. by only accepting the 6 most similar word to any given word in the dataset, the model is being much more closed. This enforces much smaller boundaries to separate on as the distance between the words in the samples are lower overall. This could help the model's separability and improve results that way. Frustratingly, the expansion ceiling is capped at 10 due to Word2Vec's limitations and for higher values a custom cosine distance algorithm would have to be written. The Floor also had some un-

expected results. The Floor determines which of the most similar words to omit from the expanded database. A value of 4 outperformed a value of 0. This could be because the very similar words skew the dataset too much. if the Floor is 0, the dataset could be overrun with examples that all relate to the same word and the vector range could be very small. By having a floor value, the vector range is allowed to be increased as the similar examples do not skew the dataset.

5.4 The Neural Network

The neural network used for this example problem is relatively simple. It uses dense layers with dropout to classify the input into one of three classes. In an ideal world, a much more complex model could be used to better fit the data and problem. For instance, LSTM as well as Transformer based approaches have been shown to work incredibly effectively on specific natural language tasks. [HS97] [VSP+17] While 2000 units in the biggest hidden layer may be enough for this simple trinary classification problem, it may not be enough for anything meaningful, especially when looking at natural language. Without time or hardware constraints, the power of Dataset expansion could be shown on a larger scale with a more involved and in depth problem. This, network, however, *was* enough to be able top successfully classify a large portion of a difficult dataset and this, in a way, shows that the dataset expansion can be used for some generalising. Its limits however, are unknown.

5.5 The Test Set

The Test Set was very carefully chosen to be deliberately tricky by testing the model on wording it had never seen before. This is because the dataset expansion technique purely focuses on what words are used and not the order of them. This is why an accuracy of 77% on the dataset would be considered excellent. 20% of the dataset was deliberately formatted with different sentence structure than is given in the dataset for a corresponding class. This is shown in figure 14. Another 10% of the test set is made up of words that cant be found in the training set.

One whole class was given as a control to ensure the model was performing correctly. Class 1 was simple, and the model was expected to get class 1 all correct. In all the examples with full training, this was always achieved and every class 1 example was always correctly classified.

45 Samples may not be much, but each had to be manually inputted and selected purposefully. Perhaps a larger dataset could be generated and could better test the limits of a model that is trained on an expanded dataset. When expanding the test set, no new data is expected to be added, but word structure will change slightly, hence the prediction that if a model is tested on an expanded test set, it is expected to have an accuracy that is slightly worse than the accuracy on the initial dataset.

5.6 The Expansion Algorithm and pretrained models

The expansion algorithm is rudimentary in its application however acts as a template for further ideas, and even in its current form, performs excellently to allow a model to work and classify well given barely any starting data at all, by harnessing the power of Word2Vec models. The Word2Vec Model plays a *huge* role in the performance of the model as all of the vectorisation and tokenisation is done through the pretrained Word2Vec Model. There are many good pretrained Word2Vec Models that can be downloaded however unfortunately, the one used was chosen due to hardware constraints. Given better hardware, much stronger models could be used as a basis for the dataset expansion for much better results.

The algorithm also has poor time efficiency, being $O(n^3)$. This lead to very slow data processing, especially when expanding datasets to over 1000 samples large. The cosine distance algorithm used to find the most similar word in the Word"Vec model is also very slow, leading to most of the slowdowns. A custom *most similar* algorithm would be trivial to write, and would come with the benefit of allowing for expansion ceiling's of above 10.

6 Conclusion

The idea of dataset expansion has been shown to work well as a distinct method to create synthetic data for a model to train on for specific natural language related problems. The expansion algorithm is able to take a small dataset of only 30 samples and extend them to over 1000 samples to train on and give good result. It can definitely be improved and extended for better efficiency and performance, especially with more complex networks and better pretrained Word2Vec models. 2 new hyperparameters have been generated to affect the samples based on word similarity. Overall, in the space of an email personal assistant the results were good, and while there is room for improvement, the algorithm has potential to improve the performance of specific NLP tasks.

References

[AB] Colin Warner Ann Bies, Justin Mott. English news text treebank: Penn treebank revised.

[BMR+20] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *CoRR*, abs/2005.14165, 2020.

[Gol17] Yoav Goldberg. Neural network methods for natural language processing. *Synthesis Lectures on Human Language Technologies*, 10(1):1–309, 2017.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, November 1997.

[KMN09] Sandra Kübler, Ryan McDonald, and Joakim Nivre. Dependency parsing. *Synthesis Lectures on Human Language Technologies*, 2(1):1–127, January 2009.

[MCCD13] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[Rud16] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

[SKL17] Samuel L. Smith, Pieter-Jan Kindermans, and Quoc V. Le. Don't decay the learning rate, increase the batch size. *CoRR*, abs/1711.00489, 2017.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[ZZHC15] Hao Zhou, Yue Zhang, Shujian Huang, and Jiajun Chen. A neural probabilistic structured-prediction model for transition-based dependency parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1213–1222, Beijing, China, July 2015. Association for Computational Linguistics.

Appendices

Link to github: https://github.com/Tayyab-H/Synthetic-Dataset-Creation-for-NLP
Intructions are in the README.md

```
nnmodel = keras.Sequential()
nnmodel.add(keras.layers.InputLayer(input_shape = (20, 300)))
nnmodel.add(keras.layers.Flatten())
nnmodel.add(keras.layers.Dropout(0.1))
nnmodel.add(keras.layers.Dense(units = 2000, activation = "relu"))
nnmodel.add(keras.layers.Dropout(0.1))
nnmodel.add(keras.layers.Dense(units = 1600, activation = "relu"))
nnmodel.add(keras.layers.Dense(units = 100, activation = "relu"))
nnmodel.add(keras.layers.Dense(units = 100, activation = "relu"))
nnmodel.add(keras.layers.Dense(units = 20, activation = "relu"))
nnmodel.add(keras.layers.Dense(units = num_classes, activation = "softmax"))
```

Fig. 15: NN Structure

```python
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
model = KeyedVectors.load_word2vec_format("GoogleNews-vectors-negative300.bin", binary = True)
```

```python
colnames = ["input","label"]
ds = pd.read_csv('training_data.csv', usecols = colnames)
with open("stoplist.txt", "r") as F:
    stoplist = F.read()
    clean = []
for i in range(0, len(ds["input"])):
    clean.append([word for word in ds["input"][i].split() if word not in stoplist])

dataset = []
temp = []
label = []
for i in clean:
    for j in i:
        for k in range(0,10):
            word = model.most_similar(j)[k][0]
            temp = i.copy()
            temp[i.index(j)] = word
            dataset.append(temp)
            label.append(ds['label'][clean.index(i)])
```

```python
sentence = ""
sentences = []
for i in dataset:
    for j in i:
        if i.index(j) == 0:
            sentence = sentence + j
        else:
            sentence = sentence + " " + j
    sentences.append(sentence)
    sentence = ""

df = pd.DataFrame({
    'input': sentences,
    'label': label
})
df.to_csv('synthetic_training_data.csv',index = False)
```

Fig. 16: Dataset Expansion Algorithm in Practice