Building Deep Learning Models to understand Language Processing in the Human Brain

Master's Thesis in Computational Engineering

submitted
by

Kishore Surendra

born 21.08.1992 in Bangalore

Written at

Lehrstuhl für Mustererkennung (Informatik 5)
Department Informatik
Friedrich-Alexander-Universität Erlangen-Nürnberg.

Advisor: Prof.Andreas Maier, Dr.Patrick Krauss

Started: 20.09.2019

Finished: 20.03.2020

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Die Richtlinien des Lehrstuhls für Studien- und Diplomarbeiten habe ich gelesen und anerkannt, insbesondere die Regelung des Nutzungsrechts.

Erlangen, den 18.Februar 2020

Übersicht

Deep-Learning-Methoden waren in einer Vielzahl von Aufgaben der natürlichen Sprachverarbeitung (NLP) wie der Textklassifizierung, der Textvorhersage und der Textwiedergabe sehr erfolgreich. Die Repräsentationen von Sprache sind nicht gut verstanden. Obwohl sowohl das Gehirn als auch künstliche neuronale netze zur repräsentation von Sprache in der lage sind werden die Beziehungen zwischen diesen verschiedenen Systemen bisher nicht grundlich untersucht. In dieser Arbeit werden verschiedene neuronale Netze für Aufgaben wie Textvorhersage und Textwiedergabe implementiert und getestet. Während die Accuracy dieser Netzwerke wichtig ist, liegt das Hauptaugenmerk darauf, die visuelle Darstellung der Daten auf jeder Ebene der Netzwerke zu untersuchen und zu überprüfen, ob genau definierte Wortklassen entstehen. Diese Wortklassen wären analog zu verschiedenen Arealen im Gehirn, die verschiedene Arten von Informationen speichern. MDS-Diagramme (Multidimensional Scaling) werden zur Visualisierung der Daten verwendet, und der Grad der Klassentrennung in jeder Schicht der neuronalen Netze wird unter Verwendung einer genau definierten Metrik mit dem Namen ”Generalized Discrimination Value”(GDV) berechnet und aufgezeichnet. Sobald die Wortklassen festgelegt sind, wird die Auswirkung der verschiedenen Parameter und Input konfigurationen für das neuronale Netzwerk auf die oben erwähnten Diagramme untersucht.

Abstract

Deep learning methods have been very successful in a variety of Natural Language Processing (NLP) tasks such as text classification, text prediction and text reproduction. However, the representation of language learned by such methods is still opaque. Even though both the brain and deep learning methods are representing language, the relationships among these representations are not thoroughly studied. In this thesis, different neural networks are built and tested for tasks such as text prediction and text reproduction. While the accuracies of these networks are important, the main focus is on using visualization and analysis methods from Neuroscience to study the data at each layer of the networks and checking if well-defined word classes exist. These word classes would be analogous to different regions in the brain which store different types of information. Multidimensional Scaling (MDS) plots are used to visualize data, and the degree of class separations at each layer of the neural networks are calculated and plotted using a well-defined metric called ’Generalized Discrimination Value’(GDV). Once the word classes are established, the impact of varying the different input configurations to the neural network on the above mentioned plots is covered.

Acknowledgement

I wish to express my sincere gratitude to my supervisor Dr.Patrick Krauss of the HNO Klinik for proposing the thesis topic, patiently reviewing my work and constantly guiding me throughout my thesis duration. My experience with Dr.Krauss not only improved my knowledge about Natural Language Processing but also gave me new insights about how corpus linguistics principles can be applied to Deep Learning.

I would also like to thank Dr.Achim Schilling of the HNO Klinik for helping me structure a good presentation and contributing his thoughts about what can be done next to carry the thesis forward.

Finally, I want to sincerely thank my main supervisor Prof.Andreas Maier for providing me with the opportunity to pursue a wonderful Master Thesis with both the Pattern Recognition Lab of FAU and the HNO Klinik. In addition, the technical support staff of both the PR Lab and the HNO Klinik deserve credit for providing me with access to clusters which made my job easier.

Contents

Chapter 1

Introduction

1.1 Human Brain and Language Processing

The human brain is a very complex organ. Despite decades of research, Neuroscience is yet to truly reveal its full functionality.

Text comprehension usually involves delivering signals from the eyes to the visual cortex via the optic nerves. The sensory cortex is used to read in Braille. If we listen to someone else, then we use the auditory cortex located in the temporal lobe. The angular gyrus in the parietal lobe, Wernicke's area (comprising mainly the top rear portion of the temporal lobe), insular cortex, basal ganglia and cerebellum are used to interpret text [Abb16].

These regions work together as a network to process word sequences. Parts of the frontal, parietal and temporal lobes govern what we want to say, while parts of the motor cortex in the frontal lobe enables us to utter the words. To speak sensibly, one must think of words to convey an idea or message, formulate them into a sentence according to grammatical rules and then use lungs, vocal cords and mouth to create sounds.

Today, we can get a much better view of brain function by using imaging techniques, especially magnetic resonance imaging (MRI), a safe procedure that uses magnetic fields to take pictures of the brain.

Scientists are also using functional MRI to build a finer picture of how the brain processes language by designing experiments that compare which areas are active during various tasks. For instance, differences have been observed in brain language regions of dyslexic children compared to those without dyslexia [Abb16].

In the 1960s, it was also discovered that the inferior parietal lobe is important for language

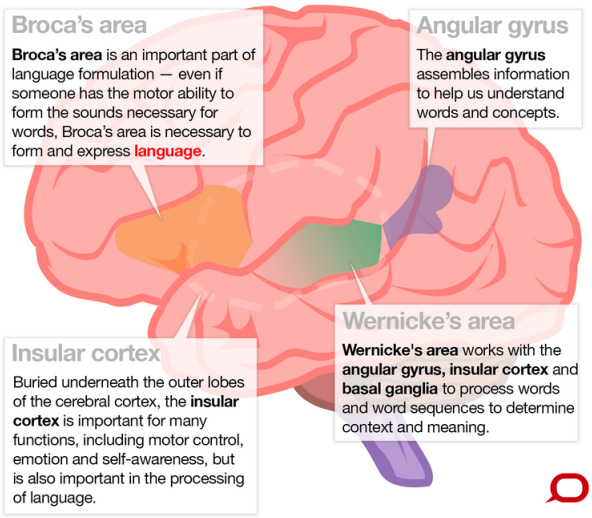


Figure 1.1: Parts of the brain responsible for language processing [Abb16]

processing. Now, we know there is another route through which language travels between Broca's area and Wernicke's area in the inferior parietal lobule. This region of the brain is all about language acquisition, where we collect and consider spoken and written words, understand their meanings, and how they sound and work grammatically [Har19]. Classification of objects using auditory, sensory and visual stimuli also falls under this brain region. Its late maturation might be the reason why children usually do not learn to read and write until after the age of 4.

Towards the end of the 20^{th} century, if a surgeon needed to find out which side of the brain was responsible for language, he would put one side of the brain to sleep using an anesthetic. The doctor would then ask a series of questions, determining the language side from the patient's ability to answer them. This invasive test is known as the Wada test [vE99], named after Juhn Wada, who first described it just after the second world war.

Although the meaning processing section of the brain has been investigated for many years, cognitive neuroscientists are yet to reach a consensus about the functional and the organizational principles of semantic knowledge. A range of neuropsychological patient studies suggest that several cortical areas contribute to semantic processing, but the precise role of each cortex is still not understood well. Cognitive and neuroscientists have suggested that the meanings of all words are equally processed and stored in a central system cortically located in a ”semantic hub” [Tom18].

1.2 Language Processing approaches

Many approaches to understand language processing have been researched extensively. Each approach aims to simplify the brain into a model and apply this model to all the functions of the brain. The popular ones such as the Hebbian model, Cortex Semantic grounding model and Deep Learning approaches are discussed here.

1.2.1 Hebbian Model

Localizationist approaches assume that small cortical regions are capable of performing extremely complex cognitive functions. For example, an area of few centimeter square of the cortex solely takes care of word comprehension, and no other area contributes to this process.

Holistic approaches on the other hand state that the entire cortex is equally involved in all cognitive operations, meaning all cortex areas equally contribute to complex processes such as language processing.

The Hebbian approach contradicts both views [Pul99]. It contrasts the localizationist approach in that, neurons in different cortical areas may be part of the same functional unit. It is different from the holistic approach in that, the representation of an image may involve regions of the cortex entirely different from the regions contributing to the representation of, say odour. The Hebbian model makes three fundamental assumptions:
1. Coactivated neurons become associated.
2. Associations can occur between adjacent or distant neurons, implying that the entire cortex is an associative memory.
3. If neurons become associated, they will develop into a functional unit, namely a cell assembly.

The Hebbian framework implies that different word forms have distinct cortical assemblies, because perception of these entities will activate different but possibly overlapping populations of neurons. If a language is not learned through the vocal and auditory modalities, but through the manual and visual modalities (sign languages), cortical localization of cell assemblies representing meaningful elements should be different. Because gestures are performed with both head and hands and perceived through the eyes, they are related to neuronal activity farther away from the more superior motor cortices and occipital visual cortices [Pul99]. Thus, it must be evident that meaningful gestures included in sign languages involve these extra-perisylvian visual, motor, and association cortices.

Hebbian logic also suggests that words of different vocabulary types contain different cortical

representations. Neurons activated by stimuli pertaining to the meaning of most concrete content words (nouns, verbs and adjectives) are likely to be stored in both hemispheres. For example, the visual perceptions of objects that can be referred to as ”cat” will probably activate equal numbers of left and right hemispheric neurons because a corresponding visual stimulus is likely to be perceived equally in the right and left visual half-fields, and, in most cases, will be at fixation so that half of it is projected to the right visual field (left hemisphere) and the other half to the left visual field (right hemisphere). Assemblies with different degrees of laterality are sketched in Figure 1.2.

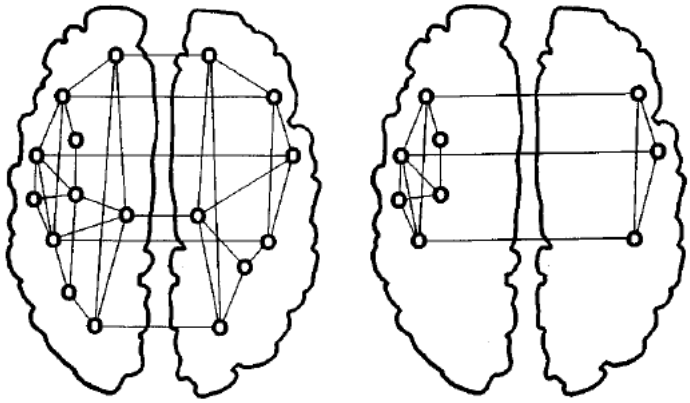


Figure 1.2: Cell assemblies representing content words(left) and cell assemblies representing function words(right) [Pul99]

Action words would refer to movements of body parts, and would thus be used frequently when such actions are being performed. A perisylvian assembly representing the word form would become linked to neurons in the premotor, motor and prefrontal cortices [Pul99]. Since the meaning of Perception words can be best explained using prototypical stimuli, they would consist of a perisylvian assembly along with the posterior cortex neurons. Visual stimuli are involved in most cases and the respective word category may therefore be labelled as vision words. Assemblies representing words of this category would be distributed over perisylvian and visual cortices in parietal, temporal, and/or occipital lobes.

1.2.2 Cortex model of semantic grounding

The Hebbian learning of structure of symbol, action and perception information showed distributed circuits of cell assemblies emerging across various cortices of the network. Category specific topographical distributions were observed, reaching into the motor areas for action related words

and visual areas for visual related words. This model of cortex grounding integrated divergent experimental results about conceptualization, and clubs both the category specific areas and semantic hubs as one emergent process determined by two major factors : Neuroanatomical connectivity structure and correlated neuronal activity during language learning.

This is a neurobiologically constrained model [Tom18] which replicates the fronto-temporal-occipital lobes, to shed light on the background mechanism of semantic processing in action and perception. The architecture contains 15,000 representative neurons and twelve cortical areas in the left hemisphere within which neuronal activity is simulated. These cortical areas represent three levels of processing: Primary, Secondary and higher association which are present in four modality systems: Motor, Articulatory, Auditory and Visual systems. The auditory and articulatory systems (areas highlighted in red and blue in Figure 1.3) are in the perisylvian language cortex and are most relevant for language processing. The motor and visual systems (green and yellow areas in figure 1.3) are outside the perisylvian language cortex and involved in visual object processing and execution of manual actions.

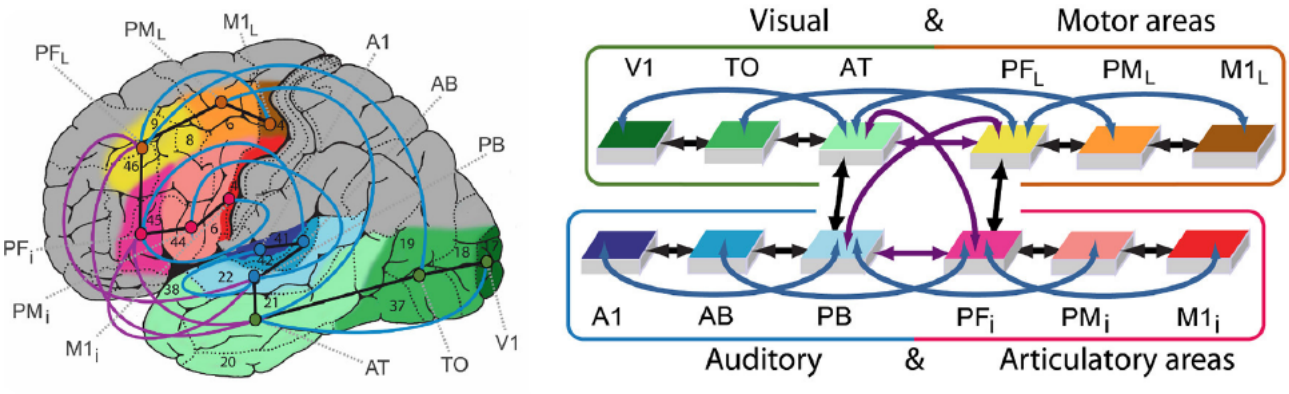


Figure 1.3: Cell assemblies representing content words(left) and cell assemblies representing function words(right) [Tom18]

The Perisylvian cortex comprises an inferior-frontal articulatory-phonological system (red colors), including primary motor cortex ($M1_i$), premotor (PM_i) and inferior-prefrontal (PF_i), and a superior-temporal acoustic-phonological system (areas in blue), including auditory parabelt (PB), auditory belt (AB) and primary auditory cortex (A1). Non perisylvian areas comprise a lateral hand-motor system (yellow to brown), including lateral prefrontal (PF_L), premotor (PM_L), primary motor cortex ($M1_L$), and a visual stream of object processing (green), including anterior-temporal (AT), temporo-occipital (TO), and early visual areas (V1). When learning words in the context of perceived objects or to actions, both perisylvian and non perisylvian systems are involved.

Numbers indicate Brodmann Areas (BAs) and the arrows (black, purple, and blue) represent long distance inter cortical connections.

Based on Hebbian learning principles, word-meaning acquisition was simulated using repeated sensorimotor pattern presentations to the network. Each instance of the network used twelve distinct sets of sensorimotor neural patterns containing six action related and six object related words. Besides perisylvian auditory (A1) and articulatory ($M1_i$) activity, object related words received concordant visual (V1), and action related words received lateral motor area ($M1_L$) activity.

Overall, the model was used to investigate neural mechanisms for word learning in a biologically constrained environment, replicating the features and connectivity of the frontal, temporal and occipital areas to simulate semantic grounding in perception and action.The results of this model can be summarized as follow:

- ”Semantic circuits”, or neuron circuits distributed across primary, secondary and multimodal areas emerged as a result of simulating the word forms in their semantically related objects and actions.
- Distinct, consecutive neuronal and cognitive processes of word comprehension and working memory was observed during re-activation of word related circuits.
- Moderate specificity in multimodal areas, as well as pronounced specificity in the modality preferential areas for word production,object and action recognition were noted.
- All semantic and phonological information converges in the multimodal areas, implying higher neuron densities and prolonged activities in these areas.

1.2.3 Deep Learning approach

Deep learning at first glance, does not seem to be a good candidate for analyzing language processing in the brain. However, deep learning has proven to be amazingly effective at capturing statistical regularities in language. Deep learning methods for natural language processing(NLP) have been very successful in a variety of NLP tasks. However, the representation of language learned by such methods is still opaque.

An approach by Dan Schwartz [Sch19] checks whether a deep learning model is able to predict brain activity from text well, and importantly, whether we can gain any understanding

about the brain activity from the predictions. Electroencephalography (EEG) is a tool that is commonly used to study those integrative processes. In this approach, fine-tuning of a language model and multitask learning are performed to better understand how various language-elicited EEG responses are related to each other. If these EEG responses are better understood, then this understanding is used to better study language processing in people. In this analysis, EEG observations of brain activity are recorded as people read sentences. Several different kinds of deviations from baseline measurements of activity occur as people read text. The most well studied of these is called the N400 response. It is a Negative deflection in the electrical activity (relative to a baseline) that occurs around 400 milliseconds after the onset of a word (thus N400), and it has been associated with semantic effort. If a word is expected in context, for example, "I like gingerbread and jam" versus "I like gingerbread and avacados", then the expected word "jam" will elicit a reduced N400 response compared to the unexpected roller. Six different language associated responses are considered. To predict these six scalar values for each word, a pre-trained bidirectional LSTM is used as an encoder. The forward and backward LSTMs are pre-trained independently on the WikiText-103 dataset to predict the next and previous words respectively from a snippet of text. The model is fine tuned by training the decoder first, keeping the encoder parameters fixed, and then continue training by also modifying the final layer of the LSTM for a few epochs.

A natural question is whether these EEG measures of brain activity can be predicted from the text at all, and whether all of this deep learning machinery actually improves the prediction compared to a simpler model. As our measure of accuracy, the proportion of variance explained is used, i.e. mean squared error on the validation set is normalized by the variance on the validation set and thus number is subtracted from 1:

$$POVE = 1 - \frac{MSE}{Variance}$$

.

The accuracy of using the decoder is compared on top of three different encoders: an encoder which completely bypasses the LSTM (i.e. the output embeddings are the same as the input embeddings to the encoder), an encoder which is a forward-only LSTM, and an encoder which is the full bidirectional LSTM.

All six of the EEG measures could be predicted at above chance levels(0). The full bidirectional encoder was able to better predict the brain activity than the other encoders. This result suggested that the context matters for the prediction of the EEG signals, which means that there is opportunity to learn about the features in the language stream that drive the EEG responses. For

each target EEG signal apart from the N400, it is possible to improve prediction through multitask learning. Multitask learning can help us understand complex relationships between EEG signals.

Another deep learning approach explores the relationship between sentence representations learned by deep learning networks and those encoded by the brain, and checks for any correlation between activations in the hidden layer and those encoded by the brain [Jat19]. Two datasets were used, namely the Magnetoencephalography(MEG) dataset and the Simple Sentence Corpus which is derived from the Wikipedia dataset and the NELL triples. Six embedding models were used to represent sentences, namely Random embedding model, GloVe embedding model, Simple Bi-directional LSTM Language model, Multi-task model, ELMO and BERT. Correlations between brain activity and deep learning model activations for a given sentence are computed using the encoding approach in figure 1.4. 306 channel 500ms MEG signal for a single word was compressed to 306 X 5 by averaging 100ms data into a single column. This MEG brain recording data is then encoded from text representation vector to brain activity using ridge regression.

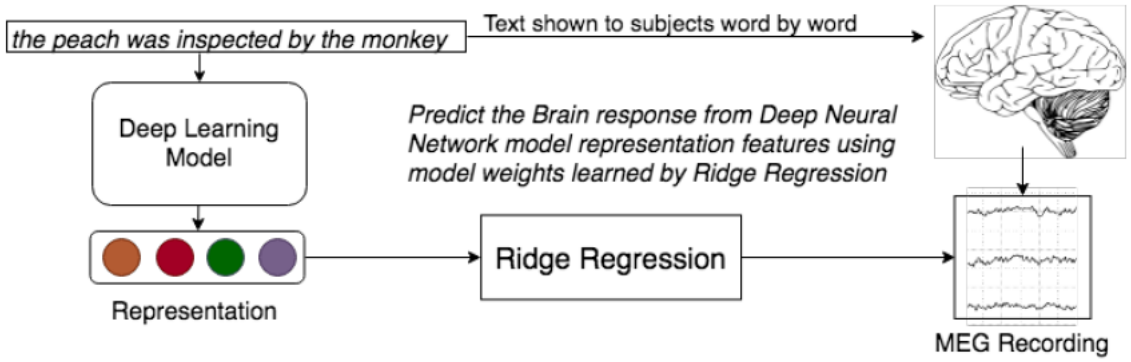


Figure 1.4: Encoding model for MEG data [Jat19]

Both macro-context experiments which aggregate the classification performance of each model's layer on the entire stimuli set, as well as micro-content experiments which evaluate if the models were able to retain information from words in the sentence prior to the word being processed were performed. Additionally, the noun sensitivity, verb sensitivity, determiner sensitivity and adjective sensitivity were computed for all the six embedding models.

Representations learned by BERT were the most effective in predicting brain activity. In particular, most models were able to predict activity in the left temporal region of the brain with high accuracy. This brain region was also known to be responsible for processing syntax and semantics for language understanding.

1.3 Word Classes

Word classes or parts of speech have formed an integral part of any language's grammar since the Greek/Latin tradition. Terms such as 'noun' or 'verb', which are found in almost any recent linguistic theory as well as in any descriptive grammar, are rooted in this tradition [Bis11]. There are four primary word classes, namely nouns, adjectives, verbs and adverbs. Secondary word classes include pronouns, prepositions, conjunctions, determiners, interjections,etc. The universal word classes or Parts of Speech(POS) tags have been summarized with examples in table 1.1.

Previous work by [Pul99] and [Tom18] have extensively dealt with the differences between how action related words and vision related words are perceived.

POS tag	**Description**	**Example**
ADJ	Adjective	small, young, blue, incomprehensible
ADP	Adposition	in, to, during
ADV	Adverb	very, tomorrow, down, where, there
AUX	Auxiliary	is, has (done), will (do), should (do)
CONJ	Conjunction	and, or, but
DET	Determiner	a, an, the
INTJ	Interjection	psst, ouch, bravo, hello
NOUN	Noun	boy, pig, tree, air, elegance
NUM	Numeral	11, 2019, seven, seventy-seven, IV, MXIV
PART	Particle	's, not
PRON	Pronoun	I, you, he, she, myself, themselves, somebody
PUNCT	Punctuation	., (,), ?
SCONJ	Subordinating conjunction	if, while, that
VERB	Verb	walk, walks, running, drink, ate, eating
X	Other	mgkflkjdsh

Table 1.1: Parts of Speech (POS) tags and their examples

From the Hebbian model of language understanding, it has already been discussed that action related words activate neurons in motor, premotor, and prefrontal cortices related to motor programs, while perception or object related words activate neurons in the parietal, temporal, and/or occipital lobes. Examples of words whose meanings are related to the visual modality are concrete, well imaginable nouns such as animal names. The best examples of action words are in the category of action verbs related to movement. Many nouns are vision words and many verbs

are action words. This model however does not consider homonyms into account (A watch/ to watch), and they are assumed to be represented using overlapping cell assemblies. Figure 1.5 makes it clear that different parts of the brain process action and vision words.

Similarly, the semantic grounding model also proved that object-related words seemed to extend more to the visual areas (V1, TO) and less to the motor areas (PML, M1L), while the vice-versa is true for action-related words (Figure 1.3).

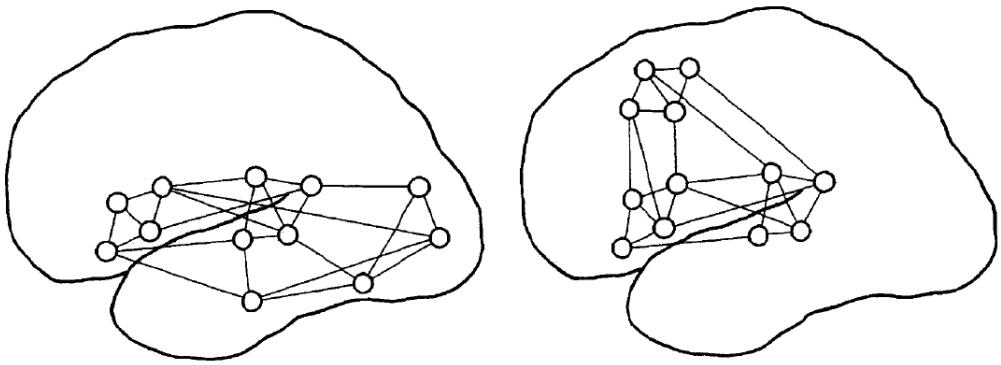


Figure 1.5: Action word assemblies(left) and vision word assemblies(right) [Pul99]

While it has been established that the brain stores motion related words(verbs) and vision related words(nouns) in separate regions, there has not been much research in verifying how other word classes such as adjectives, adverbs, pronouns,etc. are perceived in the brain. One of the main aspects of this thesis is to verify the relation between the POS tags which represent languages, and the word classes in the brain within an unsupervised environment, i.e , without feeding POS tag information to the model in any form. To learn how cognition is implemented in the brain, one must build computational models that can perform cognitive tasks, and test such models with the brain via behavioral experiments [Kri18].

Many deep neural networks will be built for applications such as text reproduction and text prediction, and the network with the best possible word class separations will be selected as the closest deep learning model to the language processing functionality of the brain. So in a nutshell, deep neural networks are built to test known hypotheses in neuroscience.

The thesis aims to answer the following questions:

a. Are the POS tags same as the word classes in the human brain ?

b. How do the internal representations of the word classes look like ?

c. Can other word classes apart from POS tags exist ?

d. How does word class separation change with different layers of the neural network ?

Chapter 2

Background

2.1 Natural Language Processing

Often while performing analysis, lots of data is numerical such as sales numbers, physical measurements and other quantifiable categories. While computers are very good at handling direct numerical information, they need other ways to deal with text data. Text data is highly unstructured and can be in multiple languages. A computer needs specialized processing techniques in order to understand raw text data. Natural Language Processing is a subfield of AI which attempts to use a variety of techniques to create structure out of text data.

Examples of some NLP use cases are:

- Classifying emails as spam vs legitimate
- Sentiment analysis of movie reviews
- Understanding text commands such as ”Hey Google, play this song!”
- Analysing trends from written customer facebook forms.

2.1.1 NLP Pipeline

The overall NLP pipeline is illustrated in figure 2.1.

Sentence Segmentation

The first step involves breaking the text corpus into separate sentences. It may be as simple as splitting the corpus at a punctuation mark and storing each sentence into separate entities. For

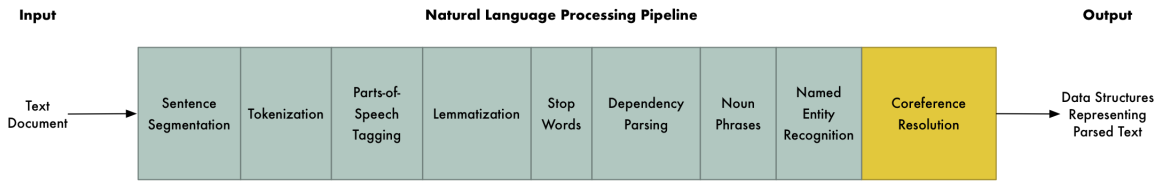


Figure 2.1: NLP pipeline [Gei18]

example, the raw text ”Erlangen is located on the edge of the Middle Franconian Basin and at the floodplain of the Regnitz river.There are roughly 100,000 residents in this city.The castle mountain in particular protects the area of the core city from cold polar air.” is segmented into three sentences : ”Erlangen is located on the edge of the Middle Franconian Basin and at the floodplain of the Regnitz river” , ”There are roughly 100,000 residents in this city ” and ”The castle mountain in particular protects the area of the core city from cold polar air”.

Tokenization

Tokenization is the process of breaking up original text into component pieces called tokens. For example, ”We are moving to Erlangen” is split into individual tokens : ’We’ , ’are’ , ’moving’ , ’to’ , ’Erlangen’. In most languages, words from a sentence can be split separately using space as a delimiter. Everything that helps us understand the meaning of text is derived from tokens and their relationship to one another.

Parts of Speech prediction

Most words are rare, and its common for words that look completely different to have the same meaning. The same words in different order can have different meanings. While it is possible to solve problems by only using raw characters of text, linguistic knowledge can add useful information. This step involves analyzing each token and trying to guess its part of speech - whether it is a noun, a verb, an adjective and so on. Knowing the role of each word in the sentence will figure out what the sentence is talking about.

Each word is fed as an input to a pre-trained Parts of Speech(POS) model. Such a model would have been trained on millions of sentences with each word’s POS already tagged and having it to learn to replicate the behavior. The model analyzes the word of interest as well as its surrounding words to predict its POS tag.

Figure 2.2 shows the result of such a POS tagging. With this information, one can already start to interpret some very basic meaning. For example, we can see that the nouns in the sentence include ”London” and ”capital”, so the sentence is probably talking about London.

Figure 2.2: POS Tagging outcome [Gei18]

Stemming and Lemmatization

When a certain keyword is searched for in a text, it helps if the search returns variations of that word. For example, searching for 'boat' may return 'boats' , 'boating' and 'boater'. Here, 'boat' would be the stem for [boat,boater,boating,boats]. Stemming essentially chops off letters from the end until the stem is reached. While it works for most cases, each language will have many exceptions where a more sophisticated process is required. Porter's algorithm, which contains five phases of word reduction returns one of the most effective stemming results. Some of these results have been summarized in table 2.1.

Word	**Stem**
Caresses	Caress
Ponies	Poni
Feed	Feed
Relational	Relate

Table 2.1: Words and their stems generated from Porter's algorithm

In contrast to stemming, lemmatization looks beyond word reduction and considers the full context and vocabulary of a language in order to apply morphological analysis to words. Lemmatization looks at the surrounding text to determine a given word's parts of speech(POS). Some lemmatization results have been summarized in table 2.2. This operation is more informative than simple stemming, so most of the NLP libraries have scrapped stemmers and are using only lemmatizers.

Word	**Lemma**
Mice	Mouse
Was	Be
Feed	Feed
Meeting	Meet/Meeting(depends on usage)

Table 2.2: Words and their lemmas

Stop Words

Words like 'a' and 'the' appear so frequently that they don't require tagging as thoroughly as nouns, verbs and modifiers. They are called 'stop words' and are usually filtered from the text to be processed. Stop words are usually identified by just checking a hardcoded list of known stop words. But there is no standard list of stop words that is appropriate for all applications. The list of words to ignore can vary depending on the application.

Dependency Parsing

A dependency parser analyzes the grammatical structure of a sentence, establishing relationships between "head" words and words which modify those heads. Figure 2.2 shows a dependency parse of a short sentence. The arrow from the word 'moving' to the word 'faster' indicates that 'faster' modifies 'moving', and the label 'advmod' assigned to the arrow describes the exact nature of the dependency. Just like how parts of speech were predicted earlier using a machine learning model, dependency parsing also works by feeding words into a machine learning model and outputting a result.

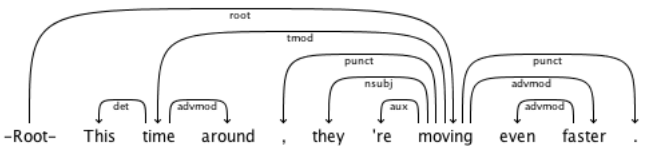


Figure 2.3: Dependency Parsing of a sentence [Gro14]

Named Entity Recognition

Named Entity Recognition(NER) seeks to locate and classify named entity mentions in unstructured text, into predefined categories such as person names, organizations, locations, medical codes, time expressions, quantities,percentages,etc. NER systems are not just performing a simple dictionary lookup. Instead, they are using the context of how a word appears in the sentence and a statistical model to guess which type of POS tag a word represents. A good NER system can spot the difference between "Milan Stankovic" the Serbian singer and the place "Milan" using context clues. An example is illustrated in Figure 2.4.

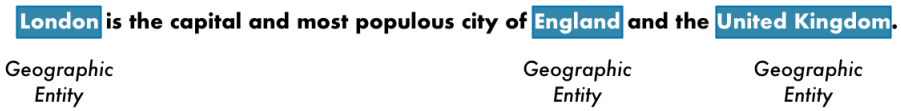


Figure 2.4: Named Entity Recognition(NER) [Gei18]

Coreference Resolution

Until this point, we know the parts of speech for each word, how the words relate to each other and which words are talking about named entities. However, there is still one big problem. Many languages are full of pronouns - words like he, she, and it [Gei18]. These are shortcuts that we use instead of writing out names over and over in each sentence. Humans can keep track of what these words represent based on context. But the NLP model does not know what pronouns mean because it only examines one sentence at a time.

Consider the third sentence in section 2.1 : ”The castle mountain in particular protects the area of the core city from cold polar air”. If we parse this with our NLP pipeline, we will know that the core ”city” is protected from the cold,polar air. But it is a lot more useful to know that ”Erlangen” is the city protected by the castle mountain. Coreference resolution applied to this sentence graphically connects the words ”Erlangen” and ”city”. This is one of the most difficult steps in the NLP pipeline to implement.It is an important step for a lot of higher level NLP tasks that involve natural language understanding such as document summarization, question answering, and information extraction. It is even more difficult than sentence parsing. Recent advances in deep learning have resulted in new approaches that are more accurate, but not perfected yet.

2.1.2 Word similarity

Given pre-trained word embedding vectors, the major use aside from feeding them into a neural network is to compute the similarity between two words using a similarity function over vectors s(m,n) [Gol17]. A common and effective choice for similarity between vectors is the cosine similarity, corresponding to the cosine of the angle between the vectors:

$$s_{cos}(m,n) = \frac{m.n}{\|m\|_2\|n\|_2}$$

When the vectors m and n are of unit-length $\|m\| = \|n\| = 1$, the cosine similarity reduces to a dot-product s(m,n) = m.n = $\sum_i m_i n_i$. Working with dot-products is very convenient computationally, and it is common to normalize the embeddings matrix such that each row has unit length.

2.2 Deep Learning

2.2.1 Perceptron

Perceptron was the fundamental building block of artificial neural network. It could classify linearly separable data. Under this concept, an artificial neuron was made to learn to update its own weights using a simple update rule. The main objective of the perceptron update rule was to find the decision boundary (its parameters a and a_0) such that after every update, the cardinality of the misclassified data samples (M) becomes smaller and smaller. Ultimately, ideally all the samples must lie on the right side of the decision boundary.

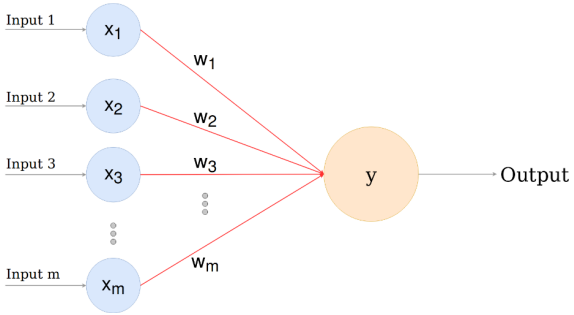


Figure 2.5: Perceptron [Ban]

The decision rule of perceptron is given by:

$$y* = sign(a^T x + a_0)$$

This means, assign a data sample 'x' to a class 'y*' based on the sign (positive or negative) obtained as a result of the operation $a^T x + a_0$.

The objective function which needs to be minimised in this case is given by:

Minimize:

$$D(\boldsymbol{a}, a_0) = -\sum_{x_i \in M} y_i.(\boldsymbol{a}^T \boldsymbol{x} + a_0)$$

In the above equation, the sign of $\boldsymbol{a}^T \boldsymbol{x} + a_0)$ term refers to the side of the decision boundary where the perceptron predicts the data sample x to lie on. The sign of $y_(i)$ refers to the side of the decision boundary where the data sample x actually lies. This implies, if the actual side and predicted side lie on the same side of the decision boundary, $D(\boldsymbol{a}, a_0)$ will be more negative. Hence, in this manner if most of the misclassified samples go to the right side of the decision boundary, the algorithm yields minimum result.

The updated a and a_0 for misclassified samples is calculated as follows:

$$\begin{pmatrix} a_0^{k+1} \\ \boldsymbol{a}^{k+1} \end{pmatrix} = \begin{pmatrix} a_0^{k} \\ \boldsymbol{a}^{k} \end{pmatrix} + \lambda . \begin{pmatrix} y_i \\ y_i . \boldsymbol{x}_i \end{pmatrix}$$

The final decision boundary is now given by:

$$F(x) = (\sum_{i \in C} y_i . \boldsymbol{x}_i)^T . \boldsymbol{x} + \sum_{i \in C} y_i = \sum_{i \in C} y_i . \langle \boldsymbol{x}_i, \boldsymbol{x} \rangle + \sum_{i \in C} y_i$$

The parameter 'a' of the decision boundary is a linear combination of feature vectors. To calculate the parameters a and a_0 , a non-linear discrete optimization problem should be solved. This is achieved using a gradient descent method. The decision boundary after all updates is dependent on initialization conditions. The algorithm ends when $y_i.(a^T x + a_0) > 0$, which means all data samples fall in their appropriate side of the decision boundary.

The decision boundary of each output neuron is linear, so Perceptrons are not capable of learning complex patterns in the data. Only if the training data is linearly separable, the perceptron is capable of converging. This is called Perceptron convergence theorem [Gér17]. The main aim here was to compute a hyperplane (decision boundary) such that the distance of misclassified feature vectors from the decision boundary is minimized.

The major advantage of Multi-layer Perceptron in comparison to the Rosenblatt Perceptron is its ability to classify non-linearly separable data. It consists of several layers of perceptrons. It consists of at least one input layer, one hidden layer and one output layer (Figure 2.5). The hidden layer nodes and the output layer nodes use a non-linear activation function. It is a feed-forward neural network that is trained by backpropagation algorithm. Its structure is the closest to an artificial neural network.

2.2.2 Training Deep Neural Networks

In the forward propagation, we check what the neural network predicts for the first training example with initial weights and bias. We first initialize the weights and bias randomly. Then we calculate the weighted sum of activation and bias. Next we apply activation functions. The most common activation functions are relu, sigmoid and tanh. The forward propagation generates some predictions. Backward propagation improves this learning process even more. Before backpropagation, we decide upon a cost function that will measure how good our neural network performs. This function calculates the difference between the actual and predicted output. In the process of trying of minimize this cost function, we try to adjust the weights and biases. The final

model (neural network) will be built on these optimal sets of weights and biases which minimize the cost function.

Before training a deep neural network it is a good practise to pre-process our data. Make sure that the data is representative. Pre-processing involves data normalization and parameter initialization or sometimes even noise removal. This would ensure that the data that neural network will be later trained on is well scaled, zero mean centred and clean. Batch normalization helps us to scale the inputs of every layer of the neural network where they are used. Dropout regularization helps us avoid overfitting problem.

2.2.3 Convolutional Neural Network

Convolutional layers are the fundamental building blocks of Convolutional Neural Network commonly known as CNN [Gér17]. In fully connected/ dense layers, neurons of one layer are connected to every single pixel in the input image. But with CNNs, each neuron in the second convolutional layer is connected only to the neurons located within a small spatial patch in the first layer. This is how a CNN learns the low level features like edges and corners in an image in the initial layers and later the high level features like overall structure in image in the final layers. CNNs are typically used in the image recognition tasks. Figure 2.6 illustrates a typical CNN for digit recognition.

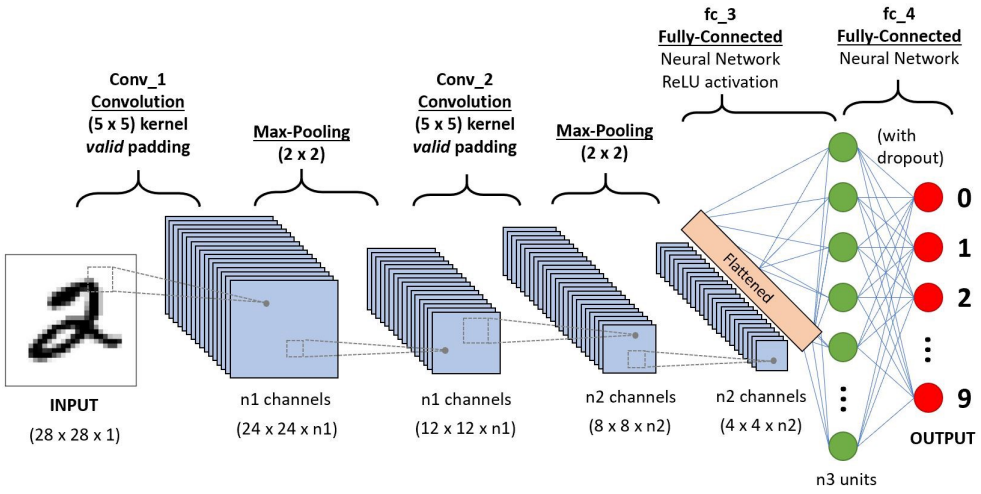


Figure 2.6: CNN for digit recognition [Sah18]

Major steps in the CNN operations are Convolution and Pooling.

Convolution is a technique of applying a filter kernel to the image to encode some local spatial

structures of the image e.g. a filter that is intended to recognize horizontal lines in the centre will be a black square with a white horizontal line in the middle region. The neurons will thus learn only the weights that correspond to the line region and everything else. Similarly, other local structures like vertical lines, edges, corners, blob like structures etc are recognized and taught to the neural network. A layer of neurons using the same filter gives us a feature map that enhances areas in the image that are most similar to the filter [Gér17]. Pooling operates on the feature maps to reduce the spatial size of the representations of local structures ultimately reducing the number of parameters to train the network with lesser computations.

2.2.4 Recurrent Neural Network (RNN)

RNN deals with the short term memory operations. RNN also has input, hidden and output layers similar to any other artificial neural network. They are networks with loops in them, allowing information to persist. A recurrent neural network has multiple copies of the same network, each passing a message to a successor. For a better representation of the RNN, these layers are squashed and stacked one behind the other. Figure 2.7 shows stacked representation of RNN.

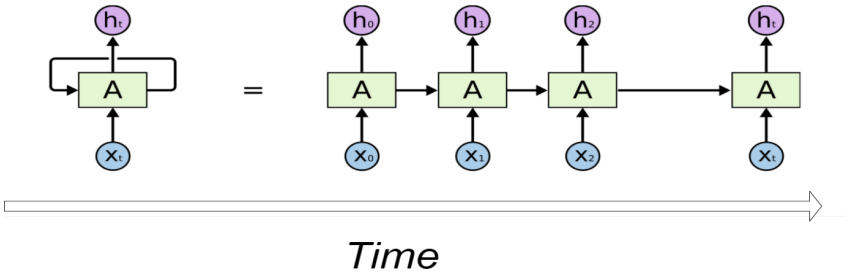


Figure 2.7: Basic RNN architecture [Ola15]

This chain-like nature reveals that RNNs are closely related to sequential data learning. RNNs have a variety of applications: speech recognition, language modeling, translation and image captioning. RNNs can be of several types based on their application: One to Many, Many to One and Many to Many.

Also, RNNs typically face two kinds of problems : Vanishing or Exploding gradient. In the process of training a neural network, the loss propagates backwards from the output to the input layer, propagating the input error gradient. The issues that can arise in deep neural networks from back propagation are vanishing and exploding gradients. As we go back to the lower layers, the gradient often get smaller and smaller. Eventually, the weights never change at deeper layers. As

a result, shallow weights in the network get trained but deeper weights remain untrained. This is referred to as the 'Vanishing Gradient' problem.

The three techniques that can solve Vanishing Gradient problem are:

- Weight Initialization
- Echo State Networks
- Long Short-Term Memory Networks (LSTMs)

Long Short-Term Memory Networks (LSTMs)

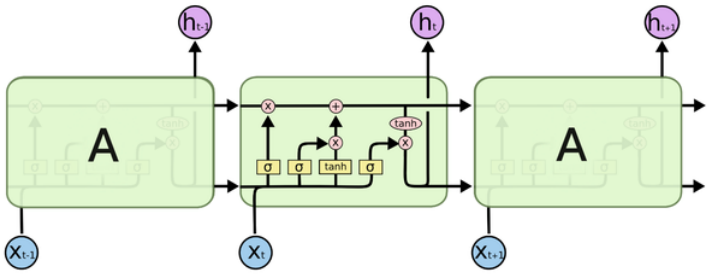


Figure 2.8: LSTM architecture [Ola15]

Figure 2.8 explains the LSTM architecture briefly. The straight line on the top that passes through all the blocks has to go through only two simple point-wise operations: 'X' means some subtraction and '+' means some addition. '**h**' refers to the hidden states while '**X**' refers to the sequential inputs.

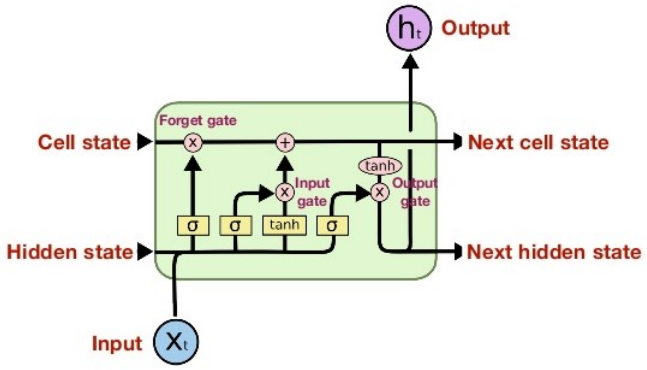


Figure 2.9: Compressed LSTM architecture [Ola15]

All the states contain information in the form of vectors. Valves are gateways that block some information and allow other information to get passed on from previous state to the next state. Cell state is long term memory while next state is short term memory. In figure 2.9, the input contains the old memory. This input vector is multiplied ('X') with either 0 or 1 (forget valve) to either eliminate the information or retain it respectively.

Next, the old memory and new memory are combined by the element wise '+' operator. How much of the new memory is added to the old memory is controlled by another valve, the 'X' below the '+' sign. Between the LSTM blocks, memory gets transferred. The neural network gets inputs from the previous LSTM block. Another memory valve takes the same input as the forget valve. The neural network's output is multiplied with this valve and added to the old memory. Eventually, we need to get the output of this LSTM unit. This step has an output valve that is controlled by the new memory, the previous output and the input. This valve controls how much new memory should go to the next LSTM unit [Yan16].

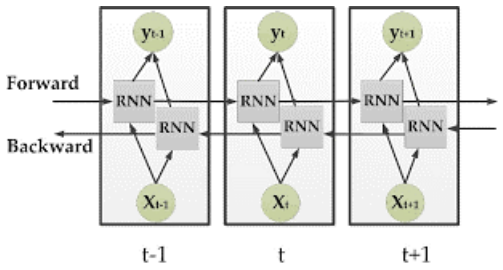


Figure 2.10: Bidirectional LSTM [Fei18]

In Bidirectional Recurrent Neural Networks (BRNN) illustrated in Figure 2.10, there is one output and two hidden layers are connected in opposite directions to this common input. Proposed in 1997 by Schuster and Paliwal [Sch97], BRNN's major objective was to increase the information made available to the neural network by giving it access to both forward states and backward states. With this form of generative deep learning, the output layer can get information from past (backwards) and future (forward) states simultaneously. BRNNs do not require their input data to be fixed. Their future input information is reachable from the current state.

Gated Recurrent Unit (GRU) is a variation of LSTM. It combines the forget and input gates into 'update gate'. It merges the cell state and hidden state into one. There is no output gate in GRU. It is simpler than standard LSTM models, and has become popular. Gated RNN is also another variant of LSTM.

2.2.5 Autoencoders

Autoencoders are artificial neural networks capable of learning efficient representations of input data in an unsupervised fashion,i.e., the training set is not labeled. These efficient representations (or codings) usually have a much lower dimensionality as compared to the input data, making autoencoders useful for dimensionality reduction. Apart from being known as powerful feature detectors [Gér17], they are capable of generating new data that looks very similar to the input training data. Such a model is called 'generative model'. Autoencoders work by simply learning to copy their inputs to their outputs.

The block diagram of a simple autoencoder is shown in Figure 2.11. It consists of four main parts [Bad19]:

1 - Encoder: In which the model learns how to reduce the input dimensions and compress the input data into an encoded representation.

2 - Bottleneck: The layer which contains the compressed representation of the input data. This is the lowest possible dimensions of the input data.

3 - Decoder: The layer in which the model learns how to reconstruct the data from the encoded representation to be as close to the original input as possible.

4 - Reconstruction Loss: This is the method that measures how well the decoder is performing and how close the output is to the original input.

The training then involves using back propagation in order to minimize the network's reconstruction loss.

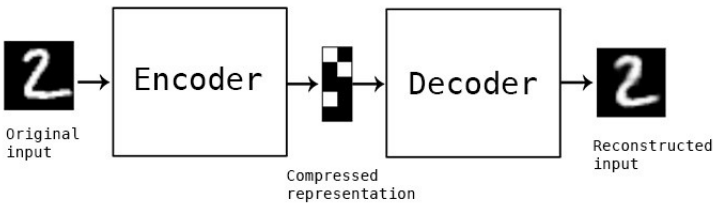


Figure 2.11: Autoencoder for MNIST data [Bad19]

The four types of autoencoders are Vanilla autoencoders, Deep autoencoders, denoising autoencoders and sparse autoencoders.

Vanilla Autoencoders

Essentially, the simplest autoencoder is a 2-layer neural network that satisfies the following conditions:

a. The hidden layer is smaller than the size of the input and output layer.

b. The input layer and output layer are the same size.

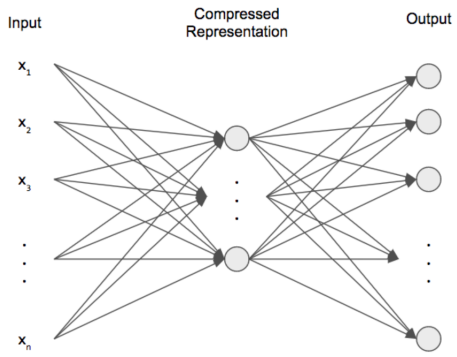


Figure 2.12: Vanilla Autoencoder [Des17]

Figure 2.12 illustrates a simple autoencoder. The hidden layer is a compressed representation, which enables us to compute two sets of biases and weights that encode the input data into compressed form and decode this compressed form back to the input space. The loss function is a euclidean distance loss: $\|x - \bar{x}\|^2$ called the reconstruction error where x is the input and $\bar{x}$ is the reconstruction. Since this error represents how close the reconstructed value is to the original input, the aim is to minimize it. A perfect reconstruction cannot be expected since the number of hidden neurons is less than the number of input neurons, but the goal is to ensure that the parameters give us the best possible reconstruction.

Mathematically, the vanilla autoencoder can be thought of as two separate entities : The encoder and the decoder.

$$\mathbf{z} = \sigma(\mathbf{W}^{(e)}\mathbf{x} + \mathbf{b}^{(e)})$$

$$\hat{\mathbf{x}} = \sigma(\mathbf{W}^{(d)}\mathbf{z} + \mathbf{b}^{(d)})$$

where the superscripts correspond to the encoder and decoder, and the input is $\mathbf{x}$. Hence, our loss function will be the squared Euclidean error which we try to minimize while training the autoencoder:

$$\mathcal{L}(\mathbf{x}, \hat{\mathbf{x}}) = ||\mathbf{x} - \hat{\mathbf{x}}||_2^2$$

Deep Autoencoders

The deep learning principle can be applied and more hidden layers can be used in the autoencoder to reduce and reconstruct the input. The hidden layers have a symmetry where the dimensionality keeps reducing at each layer (the encoder) until it reaches the encoding size, then, we expand back up, symmetrically, to the output size (the decoder) [Des17].

A deep autoencoder comprises of two symmetrical deep-belief networks that typically have four or five shallow layers representing the encoder, and a second set of four or five layers representing the decoder [Nic]. The layers are restricted Boltzmann machines, the building blocks of deep-belief networks. Figure 2.13 presents a basic schema of a deep autoencoder.

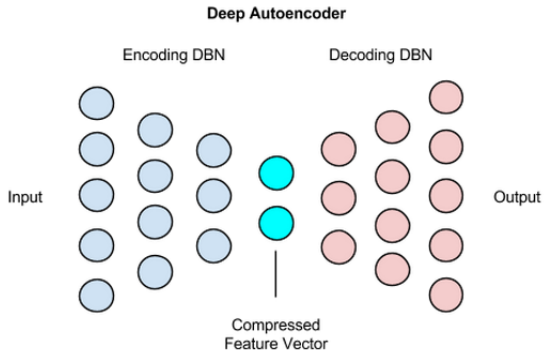


Figure 2.13: Deep Autoencoder [Nic]

Denoising Autoencoder

Instead of using fully-connected layers, convolutional autoencoders use convolutional and pooling layers to reduce the input down to an encoded representation. One application of convolutional autoencoders is denoising. Suppose we have an input image with some noise. These kinds of noisy images are actually quite common in real-world scenarios. For a denoising autoencoder, the model that we use is identical to the convolutional autoencoder. However, our training and testing data are different. For our training data, we add random, Gaussian noise, and our test data is the original, clean image. This trains our denoising autoencoder to produce clean images given noisy images.

Sparse Autoencoders

Sparse autoencoders have more hidden nodes than input nodes. They can still discover important features from the data. The sparsity constraint is introduced on the hidden layer to prevent output

layer from copying the input data directly. The Sparsity penalty $\Omega(h)$ is applied on the hidden layer in addition to the reconstruction error in order to prevent overfitting:

$$L = |x - g(f(x))| + \Omega(h)$$

where g(h) is the decoder output, and h = f(x) is the encoder output.

Sparse autoencoders take the highest activation values in the hidden layer and zero out the rest of the hidden nodes [Goo16]. This prevents the autoencoders from using all of the hidden nodes at the same time and forcing only a reduced number of hidden nodes to be used.

2.3 Word Embeddings

A word embedding is a learned representation for text where words that have similar meanings are represented similarly [Bro17]. This approach to representing words and documents has been considered as one of the key breakthroughs of deep learning on challenging NLP problems.

Individual words are represented as real-valued vectors in a predefined (usually higher dimensional) vector space. Each word is mapped to one vector and the vector values are learned in a way that resembles a neural network, and hence the technique is often lumped into the field of deep learning. A densely distributed representation is used for each word. Usually a real valued vector of tens or hundreds of dimensions is sufficient to represent words. This is in contrast to the thousands or millions of dimensions needed to sparsely represent words in case of one hot encoding. So the main benefit of using word embeddings is lesser computations [Gol17].

Word embedding methods learn a real-valued vector representation for a predefined fixed sized vocabulary from a text corpus. The learning process either involves a neural network model on some task such as document classification, or is an unsupervised process. Three popular techniques that can be used to learn word embeddings from text data are Embedding layers, Word2Vec algorithm and GloVe.

2.3.1 Embedding Layer

An embedding layer is a word embedding that is learned jointly with a neural network model on a specific natural language processing task, such as language modeling or text classification. It requires the text to be cleaned and prepared such that each word is one-hot encoded. The size of the vector space is specified as part of the model, such as 50, 100, or 300 dimensions [Bro17]. The vectors are initialized with small random numbers. The embedding layer is used on the front

end of a neural network and is fit in a supervised way using the backpropagation algorithm. The categorical or one-hot encoded words are mapped to the word vectors. If a Multilayer Perceptron model is used, then the word vectors are concatenated together into one vector before being fed as inputs to the model. If a RNN is used, then each word may be taken as a separate input in a sequence. This approach requires a lot of training data and can be slow, but will learn an embedding which is targeted to the specific text data and the NLP task at hand.

2.3.2 Word2Vec

Word2Vec was developed in 2013 as a response to make the neural network based training of the embeddings more efficient and since then, has become the default standard for developing pre-trained word embeddings. Additionally, the work involved analysis of the learned vectors and the exploration of vector math on the representations of words. For example, that subtracting the ”man-ness” from ”King” and adding ”women-ness” results in the word ”Queen”, capturing the analogy ”king is to queen as man is to woman” [Mik13].

Two different learning models were introduced that can be used as part of the word2vec approach to learn the word embedding: Continuous Bag-of-Words(CBOW) model and Continuous Skip-Gram Model. The CBOW model learns the embedding by predicting the current word based on its context. The continuous skip-gram model learns by predicting the surrounding words given a current word. Figure 2.14 illustrates the difference. Both models are focused on learning about words given their local usage context, where the context is defined by a window of neighboring words. This window is a configurable parameter of the model.

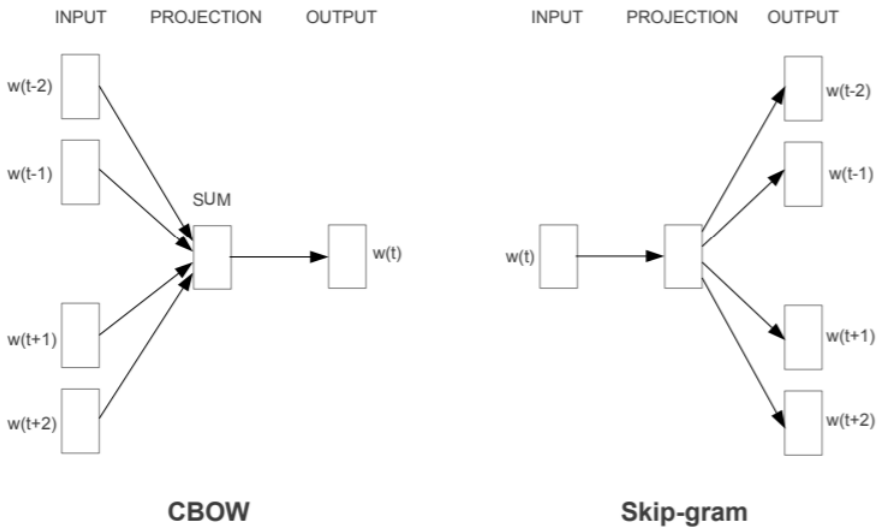


Figure 2.14: Architecture of the Word2Vec CBOW model(left) and the skip-gram model(right) [Mik13]

The key benefit of the approach is that high quality word embeddings can be learned efficiently due to lower space and time complexity, allowing larger embeddings to be learned (higher dimensions) from much larger corpora of text (billions of words).

2.3.3 GloVe

The Global Vectors for Word Representation, or GloVe, algorithm is an extension to the word2vec method for efficiently learning word vectors. Classical vector space model representations of words were developed using matrix factorization techniques such as Latent Semantic Analysis (LSA) that do a good job of using global text statistics but are not as good as the learned methods like Word2Vec at capturing meaning and demonstrating it on tasks like calculating analogies (e.g. the King and Queen example above).

GloVe is an approach to marry both the global statistics of matrix factorization techniques like LSA with the local context-based learning in Word2Vec. Rather than using a window to define local context, GloVe constructs an explicit word-context or word co-occurrence matrix using statistics across the whole text corpus [Pen14]. The result is a learning model that may result in generally better word embeddings.

2.4 Text Prediction

Sequence prediction is a popular data mining/machine learning task, which consists of predicting the next symbol of a sequence of symbols. This task is important as it have many real-life applications such as webpage pre-fetching and product recommendation. Before defining the problem of sequence prediction, it is necessary to first define a sequence. A sequence is an ordered collection of symbols. For example, here are some common types of sequences:

- A sequence of websites visited by a user, ordered by the time of access.
- A sequence of symptoms observed on a patient at a clinic.
- A sequence of products bought by a customer in an online retail store like Amazon.
- A sequence of transactions made by a bank account holder.
- A sequence of characters typed on a cellphone by a user.

The task of sequence prediction consists of predicting the next symbol of a sequence, given the previously symbols [FV16]. For example, if a person has uttered words W1, W2, W3, W4 and W5 in order, one may want to predict what can be the next word that will be uttered by him/her.

There are two steps to perform sequence prediction:

a. First, one must train a sequence prediction model using some previously seen sequences called the training sequences. This process is illustrated in Figure 2.15. For example, one could train a sequence prediction model for product prediction using the sequences of products bought by a user in his previous transactions.

b. The second step is to use a trained sequence prediction model to perform prediction for new sequences (i.e. predict the next symbol of a new sequence), as illustrated in figure 2.16. For example, using a prediction model trained with the sequences of websites visited by several users, one may predict the next website visited by a new user.

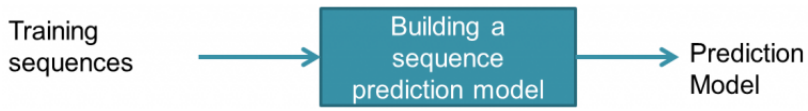


Figure 2.15: Training a prediction model [FV16]

Various sequence prediction models have their own different advantages and limitations, and may perform well on an average, on different types of data. Typically a sequence prediction model

is evaluated in terms of criteria such as prediction accuracy, the memory consumed and the time required to train the model and make predictions.

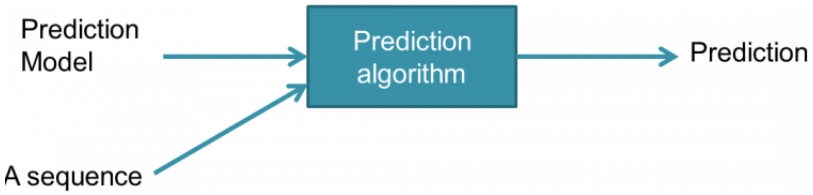


Figure 2.16: Using a trained prediction model [FV16]

2.5 Multidimensional Scaling(MDS)

Multidimensional scaling is a visual representation of distances or dissimilarities between sets of objects. ”Objects” can be colors, word classes, faces, map coordinates or political persuasion. Objects that are more similar (or have shorter distances) are closer together on the graph compared to objects that are less similar (or have longer distances). As well as interpreting dissimilarities as distances on a graph, MDS can also serve as a dimension reduction technique for high-dimensional data [Buj08]. Mathematically, it is possible to implement MDS not only in 2D and 3D spaces, but also in any R dimensional space, where $R = 1, 2, 3, \ldots$. While four dimensional space is impossible to visualize, there are numerous ways of visualization, although they may be imperfect with increasing ’R’ values [Kru78].

The basic steps are as below:

a. Assign a number of points to coordinates in n-dimensional space. N-dimensional space could be 2-dimensional, 3-dimensional, or higher spaces (at least theoretically, since 4-D spaces and above are difficult to model). The orientation of the coordinate axes is arbitrary and in the researcher’s hands.

b. Calculate Euclidean distances for all pairs of points. The Euclidean distance is the ”as the crow flies” straight-line distance between two points x and y in Euclidean space. It is calculated using the Pythagorean theorem ($c^2 = a^2 + b^2$), although it becomes more complicated for n-dimensional space. This results in the similarity matrix.

c. Compare the similarity matrix with the original input matrix by evaluating the stress function. Stress is a goodness-of-fit measure, based on differences between predicted and actual distances. In his original 1964 MDS paper, [Kru78] wrote that fits close to zero are excellent,

while anything over 0.2 should be considered ”poor”. More recent authors suggest evaluating stress based on the quality of the distance matrix and how many objects are in that matrix.

d. Adjust coordinates, if necessary, to minimize stress.

An example of a simple MDS plot has been illustrated in figure 2.17.

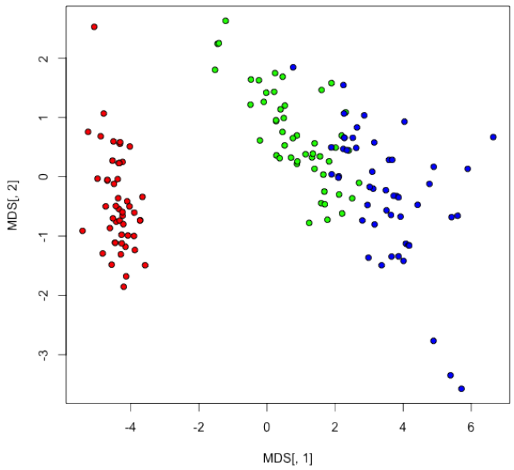


Figure 2.17: MDS plot [Sta]

2.6 Generalized Discriminant Value(GDV)

While the impact and progress of Artificial Intelligence and Machine Learning are evident in various application areas such as medicine, finance, engineering, entertainment and science, they are still facing some problems. Most of the results published in ML based projects are tough to reproduce since these results are dependent heavily on some hyper-parameters and training conditions that are not well documented. Also, the perfect parameters in all such projects are found by trial and error approaches after trying out various networks with slight differences in architecture with respect to each other. We cannot predict how the performance of a Deep Neural network (DNN) will improve/not improve with changes in various hyper-parameters such as number of layers, number of hidden units, batch size and learning rate. Therefore, new analysis tools are required to change the status of a DNN from a ’black box’ and for better theoretical understanding of internal representations at different layers of a DNN.

One way to visualize the internal representations, as mentioned above, is by dimensionality reduction techniques such as t-SNE or Multi-dimensional scaling (MDS) which project the data

in high dimensions into 2 or 3 dimensions for easier evaluation. Each data point can be colour coded according to its corresponding class label. We can then see different data points in different colors corresponding to their classes, and can then look for some well defined clusters or groups to understand class separation. The problem with this approach, however, is that representations drawn from different layers can have different dimensions. For example, the representations from layer 'n' can be 4 dimensional, but those from layer 'n+1' can be 3 dimensional. Computing class separations from them would require different scaling and normalization factors. Another problem is that the low dimensional projections can be manipulated from changing the parameters of the visualization settings.

The Generalized Discrimination Value (GDV) is a solution to these problems which does not project the data and still returns a number representing a compact notion of class separation. It is defined as the difference between the average intra-cluster distance and the average inter-cluster distance, computed on a set of z-scored and labelled vectors in n-dimensional space [Sch18]. For perfectly separated classes, the GDV is -1. For randomly distributed data points, the GDV is 0. A traditional approach to analysing data at hidden layer 'L' would require us to build a separate neural network until this layer, modify the class labels to the output form of this layer, train the network and predict the output for test data. GDV, however can be calculated at every hidden layer of a pre-trained neural network without the need to re-build or re-train.

The equations for GDV are as below:

We consider N points $X_{n=1...N} = (X_{n,1}, X_{n,2}, \ldots . X_{n,D})$, distributed within D-dimensional space. A label l_n assigns each point to one of L distinct classes $C_{l=1,...L}$. In order to become invariant against scaling and translation, each dimension is separately z-scored and, for later convenience, multiplied with 0.5 :

$$s_{n,d} = \frac{1}{2} . \frac{x_{n,d} - \mu_d}{\sigma_d}$$

Here, $\mu_d = \frac{1}{N} . \sum_{n=1}^{N} x_{n,d}$ denotes the mean and $\sigma_d = \sqrt{\frac{1}{N} . \sum_{n=1}^{N} (x_{n,d} - \mu_d)^2}$ is the standard deviation of dimension d. Based on the rescaled data points $s_n = (s_{n,1}, \ldots s_{n,D})$, we calculate the mean intra-class distances:

$$\bar{d}(C_l) = \frac{2}{N_l . (N_l - 1)} . \sum_{i=1}^{N_l - 1} \sum_{j=i+1}^{N_l} d(s_l^i, s_l^j)$$

and the mean inter-class distances:

$$\bar{d}(C_l, C_m) = \frac{1}{N_l.N_m}. \sum_{i=1}^{N_l-1} \sum_{j=i+1}^{N_l} d(s_l^i, s_l^j)$$

Here, N_k is the number of points in class k, and s_i^k is the i^th point of class k. The quantity d(a,b) is the Euclidean distance between a and b. Finally, the GDV value is calculated from the mean intra-class and inter-class distances as follows :

$$GDV = \frac{1}{\sqrt{D}}. \left[\frac{1}{L}. \sum_{l=1}^{L} \bar{d}(C_l) - \frac{2}{L.(L-1)}. \sum_{l=1}^{L-1} \sum_{m=l+1}^{L} d(C_l, C_m) \right]$$

Any layer of a trained neural network would return a GDV value between 0 and -1, as expected (Sometimes a small positive GDV value can occur in case of fully shuffled data). It is more useful to compare GDV values at different layers to actually check if class separation is increasing, rather than individually evaluating decimal numbers. So a plot of GDV vs Layer, also called GDV(L) is required. For classification problems, the GDV curve is expected to decrease overall, to indicate more negative GDV values and hence better separability between classes. Some of the properties of the GDV measure are :

- GDV(L) curve consists of 3 regions and depends on data complexity :

  We find that the curve GDV(L) contains 3 characteristic regimes : An initial regime in which the GDV can increase, a regime of decay where the GDV reduces rapidly, and a final regime where the GDV remains the same or decreases slowly. For the CIFAR10 dataset, the number of layers n_i belonging to the initial regime is 5, the number of layers belonging to the rapid decay regime n_r is also 5. Figure 2.18 depicts a rough plot between the neural network layer and its corresponding GDV value.

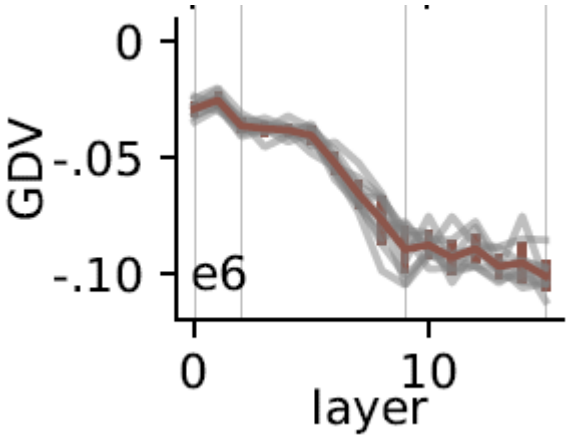


Figure 2.18: GDV vs neural network layers [Sch18]

- GDV(L) is consistent with multi-dimensional scaling analysis:

  The monotonous decrease of GDV is reflected in the MDS plots with better class separability observed. Similarly, an increase in the GDV is also visible as reduced class separation in the MDS plots wherein mixed data points of different classes are observed more frequently. So GDV and MDS plots can be used together to better visualize the data and class separations.

- GDV(L) reveals optimum model hyper-parameters:

  As a thumb rule, the optimal network depth for classification can be equated to the last layer exhibiting rapid GDV decrease. For a given dataset, if the GDV(L) curves are identical for different networks with decreasing layer width, it is better to opt for the model with lesser parameters.

- GDV correlates with test accuracy:

  Test accuracy is the usual mode of performance measurement for most common ML tasks such as classification and regression. The relation between the GDV and the test accuracy has been found to be monotonous. In other words, the test accuracy is higher for lesser (more negative) GDV and vice versa. Here, the GDV refers to that of the final layer of the neural network.

Chapter 3

Methodology

The simplified thesis pipeline has been illustrated in Figure 3.1. The raw data corpus in unstructured format undergoes thorough data cleaning to attain a desirable input format for the neural networks. 80% of the entire preprocessed data is used as the training set to train the neural networks, while the remaining 20% of the preprocessed data is saved as the test set in order to make predictions on it. Various neural networks are built for two main tasks: text reproduction and text prediction; but use the same training and test sets. The final step includes evaluating various network related metrics and plotting useful information using the test set, which can help answer the problem statement at hand. This section covers the preprocessing and neural network steps in detail.

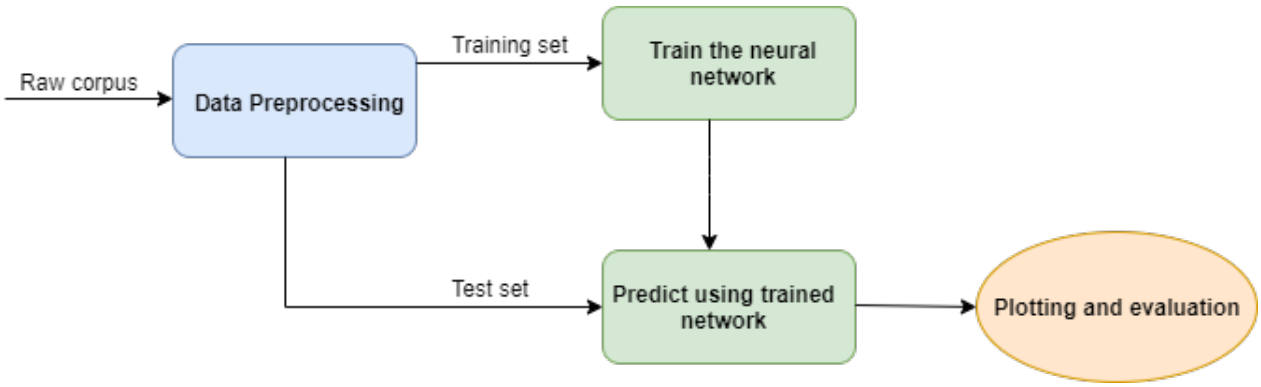
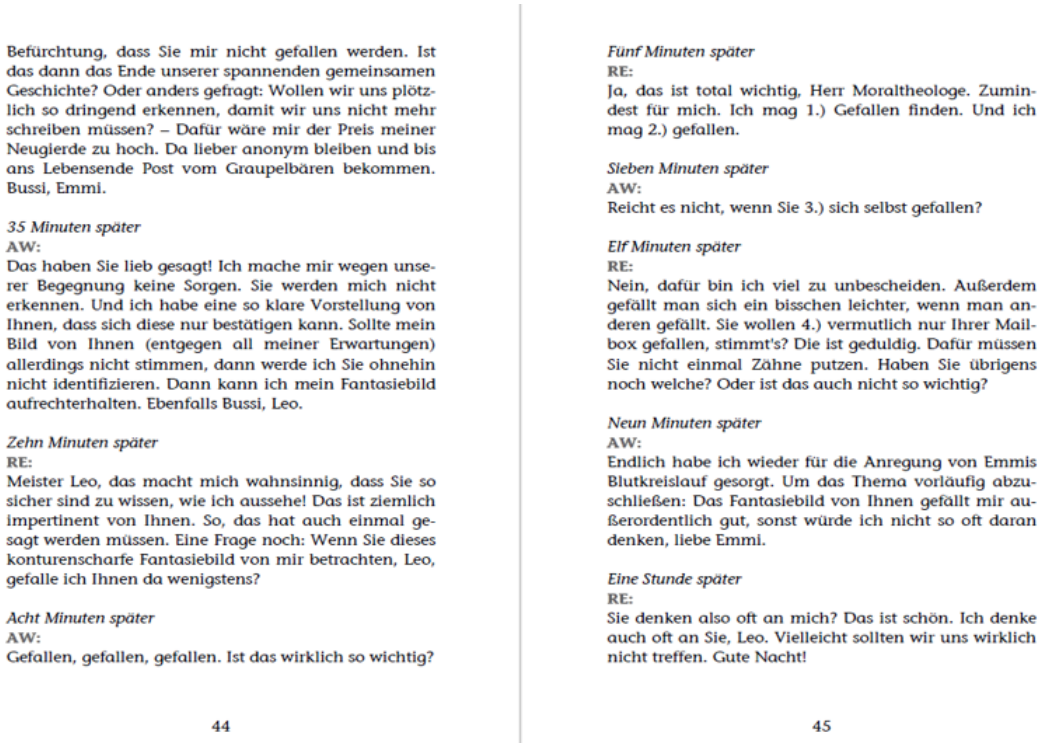


Figure 3.1: Project pipeline

3.1 Data Preprocessing

3.1.1 Data corpus

The PDF version of the famous 273 page German play 'Gut gegen Nordwind' [Gla06] is used as the raw input data. A snapshot of this play is shown in Figure 3.2.



Befürchtung, dass Sie mir nicht gefallen werden. Ist das dann das Ende unserer spannenden gemeinsamen Geschichte? Oder anders gefragt: Wollen wir uns plötzlich so dringend erkennen, damit wir uns nicht mehr schreiben müssen? – Dafür wäre mir der Preis meiner Neugierde zu hoch. Da lieber anonym bleiben und bis ans Lebensende Post vom Graupelbären bekommen. Bussi, Emmi.

35 Minuten später
AW:
Das haben Sie lieb gesagt! Ich mache mir wegen unserer Begegnung keine Sorgen. Sie werden mich nicht erkennen. Und ich habe eine so klare Vorstellung von Ihnen, dass sich diese nur bestätigen kann. Sollte mein Bild von Ihnen (entgegen all meiner Erwartungen) allerdings nicht stimmen, dann werde ich Sie ohnehin nicht identifizieren. Dann kann ich mein Fantasiebild aufrechterhalten. Ebenfalls Bussi, Leo.

Zehn Minuten später
RE:
Meister Leo, das macht mich wahnsinnig, dass Sie so sicher sind zu wissen, wie ich aussehe! Das ist ziemlich impertinent von Ihnen. So, das hat auch einmal gesagt werden müssen. Eine Frage noch: Wenn Sie dieses konturenscharfe Fantasiebild von mir betrachten, Leo, gefalle ich Ihnen da wenigstens?

Acht Minuten später
AW:
Gefallen, gefallen, gefallen. Ist das wirklich so wichtig?

44

Fünf Minuten später
RE:
Ja, das ist total wichtig, Herr Moraltheologe. Zumindest für mich. Ich mag 1.) Gefallen finden. Und ich mag 2.) gefallen.

Sieben Minuten später
AW:
Reicht es nicht, wenn Sie 3.) sich selbst gefallen?

Elf Minuten später
RE:
Nein, dafür bin ich viel zu unbescheiden. Außerdem gefällt man sich ein bisschen leichter, wenn man anderen gefällt. Sie wollen 4.) vermutlich nur Ihrer Mailbox gefallen, stimmt's? Die ist geduldig. Dafür müssen Sie nicht einmal Zähne putzen. Haben Sie übrigens noch welche? Oder ist das auch nicht so wichtig?

Neun Minuten später
AW:
Endlich habe ich wieder für die Anregung von Emmis Blutkreislauf gesorgt. Um das Thema vorläufig abzuschließen: Das Fantasiebild von Ihnen gefällt mir außerordentlich gut, sonst würde ich nicht so oft daran denken, liebe Emmi.

Eine Stunde später
RE:
Sie denken also oft an mich? Das ist schön. Ich denke auch oft an Sie, Leo. Vielleicht sollten wir uns wirklich nicht treffen. Gute Nacht!

45

Figure 3.2: A peek into the play 'Gut gegen Nordwind' [Gla06]

For further analysis, the famous English novel 'Hitchhiker's guide to the galaxy' has also been used as a raw corpus in document format. Its snapshot has been displayed in Figure 3.3.

Far out in the uncharted backwaters of the unfashionable end of the western
spiral arm of the Galaxy lies a small unregarded yellow sun. Orbiting
this at a distance of roughly ninety-two million miles is an utterly insignificant little blue green
planet whose ape-descended life forms are so amazingly
primitive that they still think digital watches are a pretty neat idea.
This planet has - or rather had - a problem, which was this: most of the
people on it were unhappy for pretty much of the time. Many solutions were
suggested for this problem, but most of these were largely concerned with the
movements of small green pieces of paper, which is odd because on the whole
it wasn't the small green pieces of paper that were unhappy.
And so the problem remained; lots of the people were mean, and most of
them were miserable, even the ones with digital watches.
Many were increasingly of the opinion that they'd all made a big mistake
in coming down from the trees in the first place. And some said that even the
trees had been a bad move, and that no one should ever have left the oceans.

Figure 3.3: A peek into the novel 'Hitchhikers Guide to the Galaxy' [Ada17]

3.2 Data Cleaning

Removing stop words and unwanted characters is an important step in NLP to weed out data which brings no value to the deep learning task at hand. It is customary to remove punctuation and other special characters from regular novels before analysis. A play on the other hand would pose more challenges due to additional unwanted information such as extra spaces, words without vocabulary(uttered in cases of anger, disgust or happiness), behind the scenes information which is not a part of the speakers' conversations,etc.

Regular expressions are used to both replace certain characters with other relevant characters, and to completely remove other characters. Table 3.1 shows some of the operations performed on certain kinds of characters.

Character/Word	**Operation**
Repititive words:RE:,AW:,Eine,Zwei,..,Stunden,Sekunden, Stunden...,später,Am nächsten,Kein Betreff,Betreff	Remove completely
Punctuation and other characters: .,',„;?,%,&,',',!	Remove completely
Numbers: 18,1,500,...	Replace with 'nummer'
Extra whitespaces	Replace with single space
E-mail	Replace with 'email'

Table 3.1: Words and their replacements using regular expressions

The resulting word sequences from chapters 1 to 7 are considered as the training set, while those belonging to chapter 8 and 9 are saved as the test set. All words are converted to lower

case to maintain uniformity, so that two same words with a different case are not considered as separate words by the NLP algorithm.

The English corpus is a novel, so the only unwanted characters are punctuations and other special characters. They are removed using simple regular expressions.

3.2.1 Cleaned Dataset

After applying all the data cleaning rules mentioned above, the processed data looks similar to Figure 3.4.

```
eiß zum beispiel noch immer nicht ob sie professor sind google kennt sie jedenfalls nicht oder versteht es sie gut zu verstecken und ist es um ihren humor sc
teil über die email als transportmittel von emotionen deshalb neige ich ein wenig zum fachsimpeln ich werde mich aber künftig zurückhalten das verspreche ich
nen und verzichten sie auf den smiley alles liebe ich find es echt angenehm mit ihnen dern emmi rothner liebe emmi rothner danke für ihre humortipps sie werd
mir welche schuhgröße ich habe alles liebe emmi macht echt spaß mit ihnen nummer sie schreiben wie nummer aber sie sind um die nummer sagen wir nummer woran
essor zu unterhalten und dabei interessant zu finden wie jung oder alt er sieschätzt noch was zu emmi heißt man nun emma und schreibt man jünger als man ist
von ihnen vor mir und das habe ich ihnen in übertriebener sucht ich wollte ihnen wirklich nicht zu nahe treten liebe grüße leo leike lieber professor ich mag
achtsgruß psycho wo haben sie sen anscheinend kränkt man sie zu tode wenn man frohe weihnachten und ein gutes neues jahr sagt gut ich verspreche ihnen ich we
t dieser grundsatzfrage muss ich sie alleine lasen ich habe leider einen termin firmunterricht tanzschule nagelstudio teekränzchen suchen sie es sich ruhig a
n herein feststellt wahrscheinlich werden wir uns nimals sehen dann hat sie natürlich völlig recht und ich teile ihre ansicht ich halte das für sehr sehr klu
r jetzt mehr hyperventiliert als geschrieben stimmts ich muss gar nicht wissen wie sie aussehen wenn sie mir solche antworten geben emmi ich habe sie ohnehin
```

Figure 3.4: Preprocessed German data after cleaning

The total number of words in the entire processed German and English corpora are 40460 and 47470 respectively. Some important statistics such as the number of words and the number of unique words(vocabulary) in the training and test sets of both the corpora are tabulated in Table 3.2.

The processed English data looks similar to Figure 3.5.

```
far out in the uncharted backwaters of the unfashionable end of the western spiral arm of the galaxy lies a small unregarded yellow sun orbiting this at a di
and that no one should ever have left the oceans and then one thursday nearly two thousand years after one man had been nailed to a tree for saying how great
ble book ever to come out of the great publishing houses of ursa minor of which no earthman had ever heard either not only is it a wholly remarkable book it
sday the story of its extraordinary consequences and the story of how these consequences are inextricably intertwined with this remarkable book begins very s
which he always used to tell his friends was a lot more interesting than they probably thought it was too most of his friends worked in advertising on wednes
ffee yawn the word bulldozer wandered through his mind for a moment in search of something to connect with the bulldozer outside the kitchen window was quite
water it would sort itself out he d decided no one wanted a bypass the council didn t have a leg to stand on it would sort itself out god what a terrible han
n for little fur hats he was by no means a great warrior in fact he was a nervous worried man today he was particularly nervous and worried because something
```

Figure 3.5: Preprocessed English data after cleaning

	Number of words	**Vocabulary size**
Train Set (German))	32368	5233
Test Set (German)	8092	1906
Train Set (English)	37976	5252
Test Set (English)	9494	2100

Table 3.2: Training and Test set details

3.3 Text Reproduction

One of the tasks for which neural networks are built is Text reproduction. This task simulates how a person(usually a child) learns language by repeating what he/she hears. Given a sequence of words of fixed length, a neural network is built which tries to replicate or reproduce the same sequence of words at the output.

As discussed, 80% of the words belonging to the training set are converted to their respective 384 dimensional embedding vectors (in-built spaCy dimensions) as a sequence. The neural network designed must try to reproduce the same input as the output. Therefore, the dimensions of the input and output sequence must have the same overall shape.

3.3.1 Embeddings Model

The neural network designed using only the word embeddings at the input is shown in Figure 3.6. The Bidirectional LSTM layers deal with sequential information, while the final linear layer contains the same number of neurons as the embedding dimension of 384 to maintain similar dimensions between the input and output sequences. The input sequence is of shape (32368,384), same as the output sequence.

3.3.2 Embeddings with POS Model

The network architecture for this model is shown in Figure 3.7. It closely resembles the previous architecture (Figure 3.6), except for some minute differences. The embeddings of each word in the input sequence are concatenated with their corresponding Parts of Speech(POS) tags. Each word in the corpus can belong to any of the 13 POS tags.

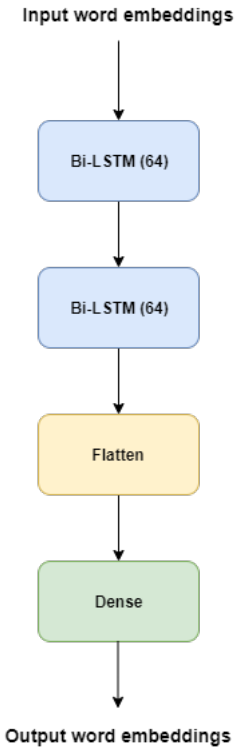


Figure 3.6: Embeddings Model for text reproduction

POS Tag	**Categorical vector**
'NUM'	[0,1,0,0,0,0,0,0,0,0,0,0,0,0]
'VERB'	[0,0,1,0,0,0,0,0,0,0,0,0,0,0]
'ADJ'	[0,0,0,1,0,0,0,0,0,0,0,0,0,0]
'X'	[0,0,0,0,1,0,0,0,0,0,0,0,0,0]
'PART'	[0,0,0,0,0,1,0,0,0,0,0,0,0,0]
'NOUN'	[0,0,0,0,0,0,1,0,0,0,0,0,0,0]
'SCONJ'	[0,0,0,0,0,0,0,1,0,0,0,0,0,0]
'ADP'	[0,0,0,0,0,0,0,0,1,0,0,0,0,0]
'DET'	[0,0,0,0,0,0,0,0,0,1,0,0,0,0]
'PRON'	[0,0,0,0,0,0,0,0,0,0,1,0,0,0]
'CONJ'	[0,0,0,0,0,0,0,0,0,0,0,1,0,0]
'AUX'	[0,0,0,0,0,0,0,0,0,0,0,0,1,0]
'ADV'	[0,0,0,0,0,0,0,0,0,0,0,0,0,1]

Table 3.3: POS tags and their respective vectors

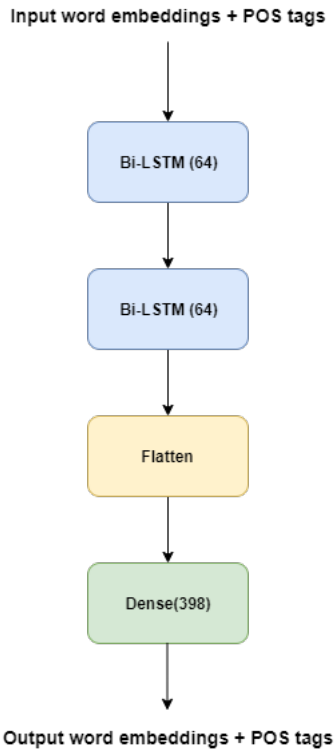


Figure 3.7: Embeddings with POS Model for text reproduction

The 13 POS tags are one hot encoded,i.e, they are converted to 14 dimensional vectors such that only one of the dimensions will be active or '1' for a particular POS tag. Table 3.3 shows the vectors generated for each POS tag. The 384 default dimensions for each word, concatenated with its corresponding 14 dimensional POS tag vector leads to 398 dimensions per word at the input. Therefore, the shape of the input sequence becomes (32368,398). This is also reflected in the 398 neurons at the final linear layer of the network. The expected shape of the neural network output is now (32368,398).

3.3.3 Embeddings with Character Matrix Model

The main motto of building this model is to combine syntax (character matrices) and semantics (word embeddings) in the input and feed this to the network, expecting it to learn language in a more inclusive fashion.

This model is a 2-input 2-output autoencoder. The two inputs are the previously used 384 dimensional word embeddings, as well as the character matrices for the corresponding words. Figure 3.8 illustrates the detailed network. The output of one Dense layer gives us the embeddings and the output of the other Dense layer returns the character matrices. Ideally, the character matrix and embedding input must match their corresponding outputs per word. The dimensions of the character matrix input and embeddings input are (32368, 31, 30,1) and (32368, 1, 384) respectively. The 'Embeddings Model' refers to the previously built model in 3.3.1. The outputs of this model and the flatten layer are concatenated and fed to the dense layers.

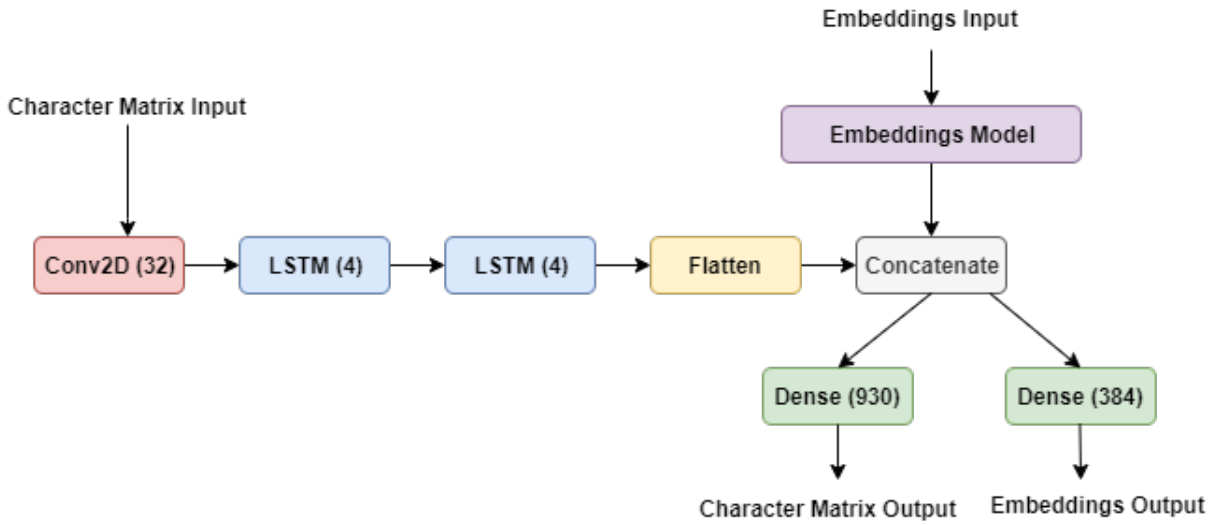


Figure 3.8: Embeddings with Character Matrix Model

Character Matrices are 31×30 in dimension, where '30' refers to the number of unique alphabets used in the language of the input (German), and '31' refers to the length of the longest word in the corpus (frostschutzmittelverkaufsstände). Therefore, each row refers to the position of an alphabet in a word, and each column refers to the alphabet itself. Figure 3.9 illustrates a simple character matrix for the word 'bad'. The dimensions of the character matrix dictate the number of neurons in the final dense layer (930).

a	b	c	d
0	1	0	0
1	0	0	0
0	0	0	1

Figure 3.9: Character Matrix for the word 'bad'

3.4 Text Prediction : 1 word

Another important task for which neural networks are built is Text prediction. This task simulates how a person can predict the remaining part of a sentence or phrase, given the initial part. Given a sequence of words of fixed length, a neural network is built to predict

the next word of the same sequence.

As discussed, 80% of the words belonging to the training set are converted to their respective 384 dimensional embedding vectors as a sequence. Here, we aim to predict the tenth word, given a prior sequence of nine words.

3.4.1 Embeddings Model

The neural network designed using only the word embeddings at the input is shown in Figure 3.10. The Bidirectional LSTM layers deal with sequential information, while the final linear layer contains the same number of neurons as the embedding dimension of 384 to achieve the expected output dimensions. The input sequence for the German corpus is of shape (32358,9,384), which means that there are 32358 rows of 9 words, where each word has 384 dimensions as per spaCy library's inbuilt dimensionality for German words. The dimension of the output will be (32358,384), since we predict only 1 word of 384 dimensions.

Similarly, the input sequence for the English corpus is of shape (37966,9,384). It turns out that 384 is the default spaCy library's dimensionality for English words too. The dimension of the output will be (37966,384).

This model will be used as a benchmark for most of the tasks in the thesis such as unigram to 9-gram predictions, Particles case study and collocations vs non collocations analysis.

3.4.2 Embeddings with POS Model

Similar to section 3.3.2, the embeddings of each word in the input sequence are concatenated with the one hot encoded vectors of their corresponding Parts of Speech(POS) tags. 13 POS tags have been considered. Therefore, the dimensions of each word will increase by 14, i.e, 398. The input dimension now is (32358,9,398) and the output dimension is (32358,398).

3.4.3 Embeddings with Character Matrix Model

Similar to section 3.3.3, the intent of building a model with both embeddings and character matrices is to combine syntax and semantics to ensure that the model is comprehensive. Fig-

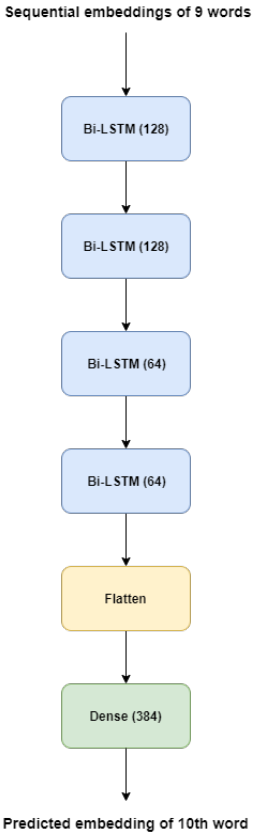


Figure 3.10: Embeddings model for single word text prediction

ure 3.11 illustrates the detailed network. It is similar to Figure 3.8, except for the dimensions of the input/output as well as the last dense layer. The dimensions of the character matrix and embeddings input are (32358,31,30,9) and (32358,9,384) respectively. The output dimensions of the character matrix and embeddings are (32358,31,30) and (32358,384) respectively.

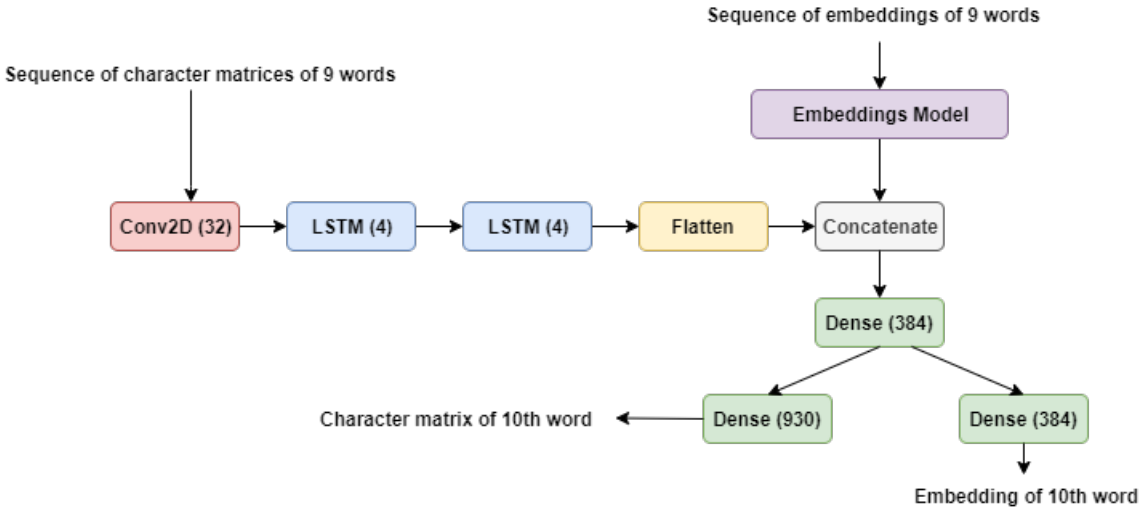


Figure 3.11: Embeddings with Character Matrix model for single word text prediction

3.5 Text Prediction : 2 Words

Similar to section 3.4.1, the embeddings of 9 words are used as inputs. The only difference is that we aim to predict both the tenth **and eleventh** words. The input sequence will now take the shape (32357,9,384), while the output dimension will be (32357,2,384) which means that there are 32357 word pairs at the output where each word is represented by a default spaCy 384 dimensional vector.

The simple neural network built for this task is shown in Figure 3.12. It is used as the word pair prediction model for both the German play and the English novel.

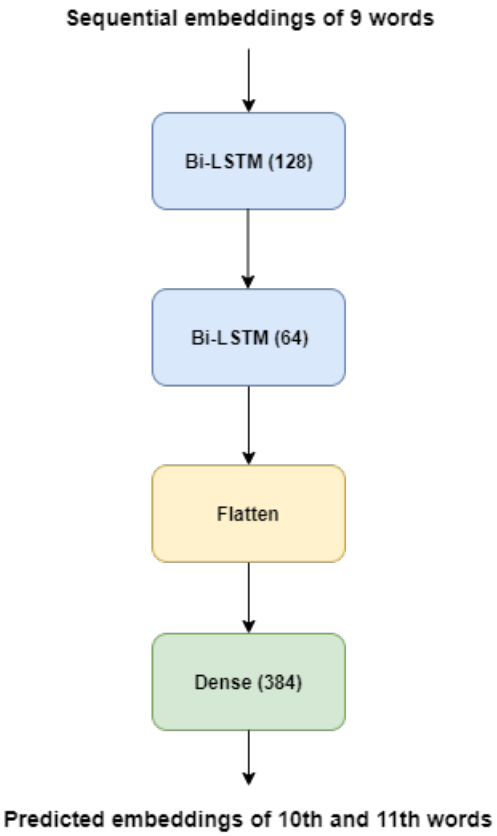


Figure 3.12: Embeddings model for word pair text prediction

3.6 Collocations vs Non Collocations

Apart from POS tags, it is interesting to look out for other word classes. According to Corpus linguistics, collocations are word groups which co-occur more frequently than expected by chance. In this thesis, we test whether collocations and non collocations are represented in different parts of space by the neural network. Both the bigram and trigram models are tested, i.e, we plot the MDS for 2 word and 3 word collocations. This task is performed for both the German play and English novel.

Consider the bigram model. The first step is to identify which word groups can be collocations in the text. For this, a 2 dimensional bigram matrix of dimensions (N_{unique}, N_{unique}) is built, where N_{unique} refers to the number of unique words in the entire corpus. Each entry in the matrix will contain a number corresponding to the number of times the bigram representing this particular row and column has appreared in the text. For example, a number 'n_{ij}' in row 'i' and column 'j' means that the bigram of the i^{th} word and j^{th} word

together occurs 'n_{ij}' times in the text. The sum of all the entries in the bigram matrix will be equal to the total number of words in the corpus. Figure 3.13 illustrates how such a bigram matrix would look like.

Once the bigram matrix is built, the number of occurrences of a bigram is plotted against the frequency of occurrences to understand the distribution. The vast majority of bigrams are expected to occur only once or twice. Therefore, a logarithmic plot is preferred due to unevenly high frequencies for small occurrences.

Next, the threshold of occurrences for a bigram to qualify as a collocation must be established. Collocations lie in between very frequently occurring bigrams and sparsely occurring bigrams. Therefore, it makes sense to consider a threshold in the middle.

	1	**2**	.	.	.				N_{unique}
1	0	1							
2						8			
.								9	
.				5					
					2	0			
	0	0	0	3			0	0	
		5			4	0			
N_{unique}									0

Figure 3.13: Bigram matrix

Chapter 4

Results and Discussion

4.1 German Play

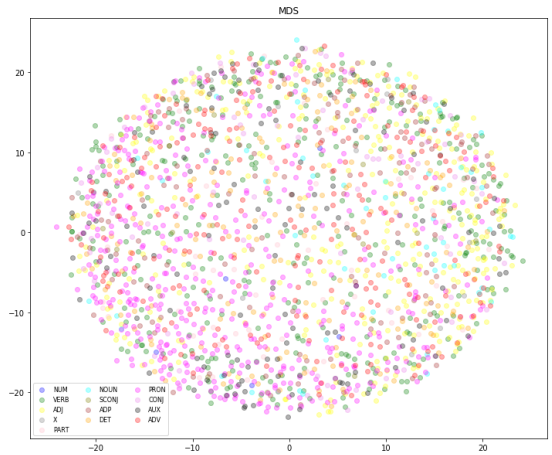
4.1.1 Text Reproduction Results: One word

Embeddings Model

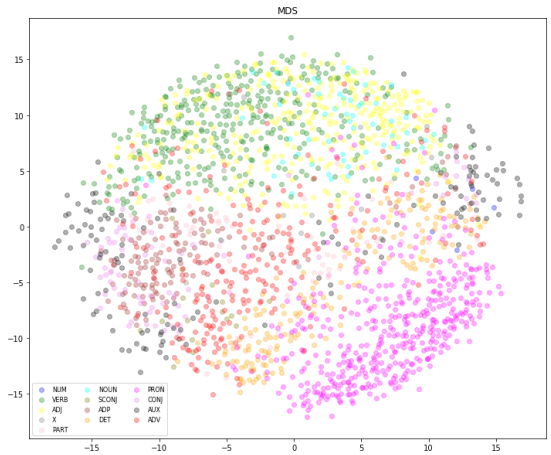
The model trained in section 3.3.1 is tested on the remaining 20% of the input corpus which is considered as the test set. The dimension of the test input data is therefore (8082,384). The predicted embeddings are compared with their corresponding original embeddings of the test set, and useful metrics such as the word similarity and test accuracy are tabulated.

MDS plots at each layer of the neural network (Figure 3.6) are visualized to check if the POS tags can be considered as word classes. Since the POS tag or label name for each data point to be plotted is known beforehand, different colors are selected for different POS tags in order to visualize their separation in the MDS plots.

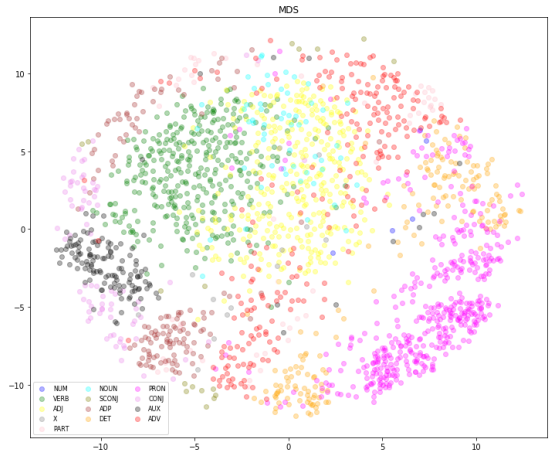
Table 4.1 illustrates the four MDS plots from layer 1 until layer 4. While the layer 1 MDS shows random distribution of data points, we can see almost clear class separations in layers 3 and 4. This is also validated from the general decline in the GDV curve against the layers (Figure 4.1).



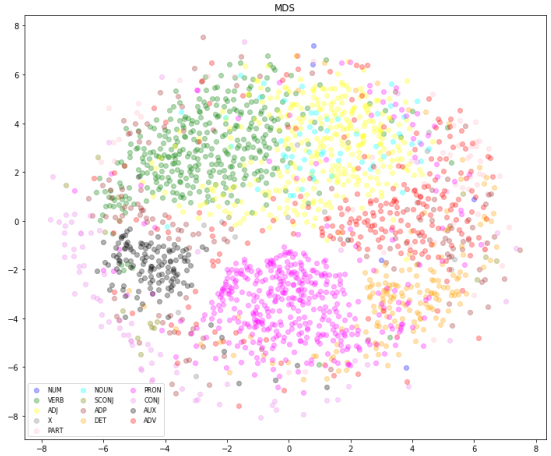
(a) Layer 1 MDS plot



(b) Layer 2 MDS plot



(c) Layer 3 MDS plot



(d) Layer 4 MDS plot

Table 4.1: MDS plots for Text Reproduction with Embeddings model

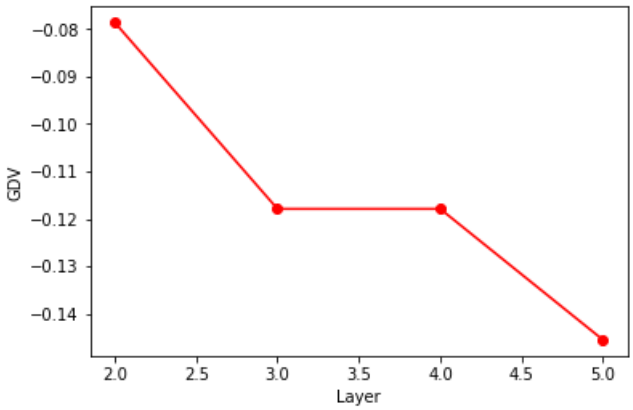


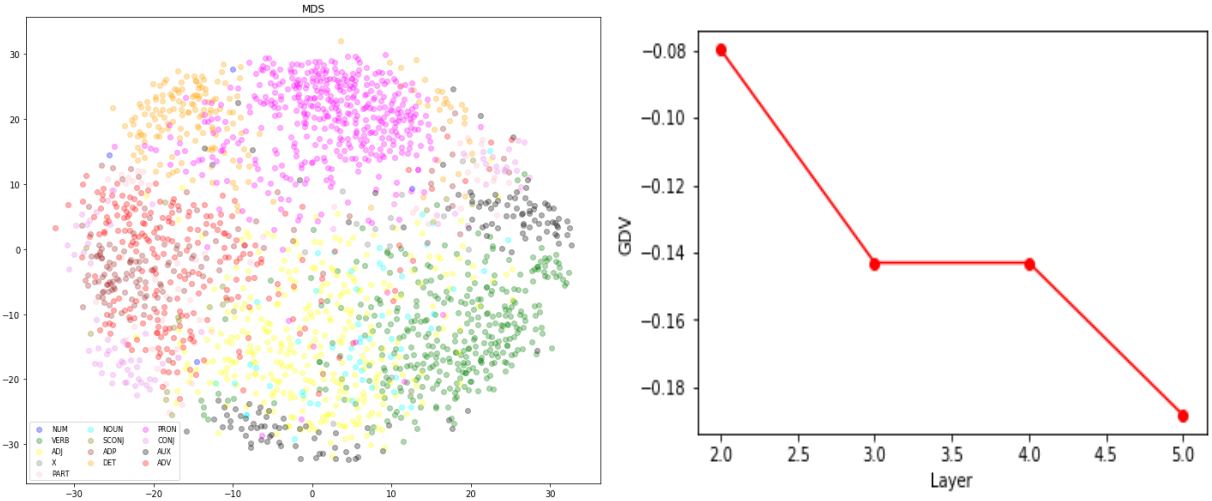
Figure 4.1: GDV plot for Text Reproduction - Embeddings Model

The class separations in the MDS final layer plot suggest that POS tags are good fits to be the word classes being sought.

Embeddings+POS Model

The model trained in section 3.3.2 is tested on the remaining 20% of the input corpus which is considered as the test set. The dimension of the test input data is therefore (8082,398). The predicted 398 dimensional embeddings are compared with their corresponding original embeddings which were concatenated with the POS vectors. MDS plots are visualized at each layer output and the GDV is plotted against layers. Figure 4.2 illustrates the last layer MDS plot and the overall GDV plot generated from the network.

This is a supervised learning approach where the POS information is added to each word, thereby making it easier for the network to separate the 13 unique POS tags. The intent was just to check if adding the POS tag information with each word would lead to significant improvement in the metrics. The GDV improves compared to the previous case, achieving more negative values. But this method will not be considered in the future, since unsupervised ways to find word classes is the focus in the thesis.



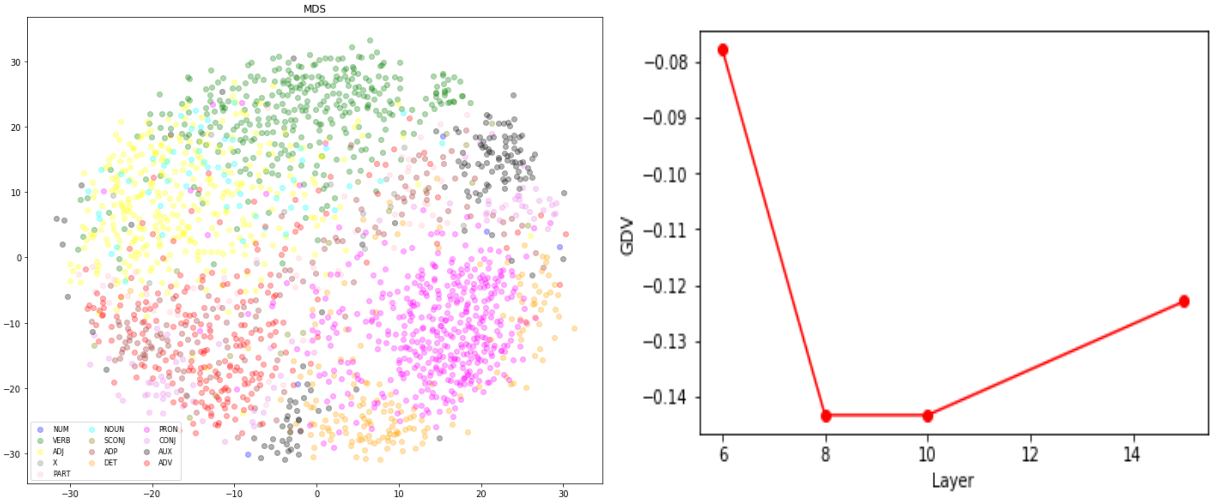
(a) Final layer MDS plot

(b) GDV plot

Figure 4.2: Layer 4 MDS plot (a) and GDV plot (b) for the Text Reproduction Embeddings + POS model

Embeddings with Character Matrix Model

The model trained in section 3.3.3 is tested on the remaining 20% of the input corpus which is considered as the test set. The dimensions of the test set character matrix input and embeddings input are (8082,31,30,1) and (8082,1,384) respectively. The predicted embedding outputs of 384 dimensions are compared with their corresponding original embeddings which were passed as one of the inputs apart from the Character Matrix input. MDS plots are visualized at each layer output and the GDV is plotted against layers. Figure 4.3 illustrates the last layer MDS plot and the overall GDV plot obtained from the network.



(a) Final layer MDS plot

(b) GDV plot

Figure 4.3: Layer 4 MDS plot (a) and GDV plot (b) for the Text Reproduction Embeddings with Character Matrix model

While adding the Character Matrices was done to combine syntax and semantics, this model performs worse than the other models in terms of class separation. The GDV curve drops down at the third layer and rises up again at the final layer. The conclusion is that only embeddings are sufficient to build a good text reproduction network with increasing class separation against layers of the network.

Useful Statistics

Table 4.2 states the Test accuracy and word similarity obtained for all the 3 explained models for the task of text reproduction.

Cosine similarity measure is used to compute the word similarity between the predicted words and their corresponding words at the input. Similarly, test accuracy is measured by calculating the euclidean distance between each predicted word and the entire test set corpus such that the predicted word is compared to the word with the least distance from it. Since we have discarded the Embeddings+POS model, the Embeddings with Character Matrix model performs the best.

Model	**Test Accuracy**	**Word Similarity**
Embeddings Model	89.718%	99.048%
Embeddings + POS Model	93.858%	99.79%
Embeddings with Character Matrix Model	91.634%	99.296%

Table 4.2: Statistics for Text Reproduction Models

4.1.2 Text Prediction Results

Embeddings Model

The model trained in section 3.4.1 is tested on the remaining 20% of the input corpus which is considered as the test set. The dimension of the test input data is therefore (8072,9,384). The predicted embeddings are compared with their corresponding original embeddings of the test set, and useful metrics such as the word similarity and test accuracy are tabulated.

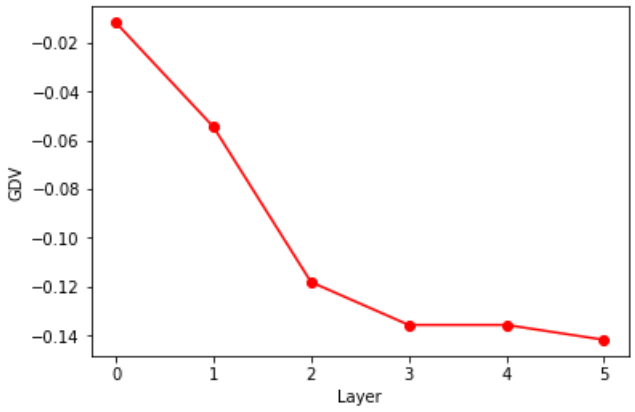
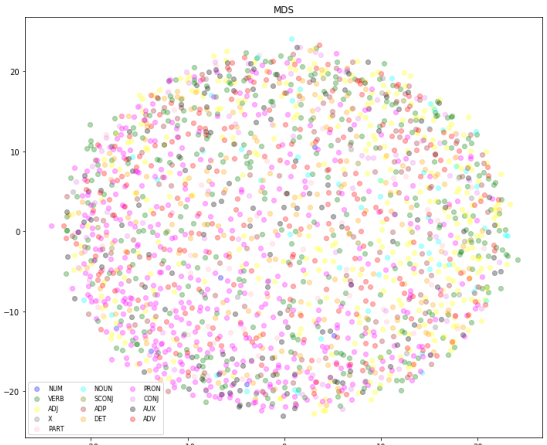
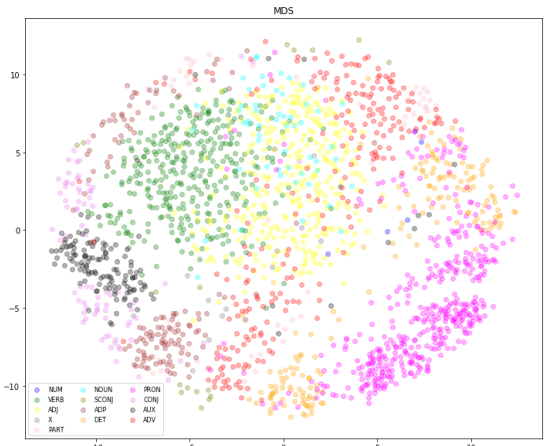


Figure 4.4: GDV plot for Text Prediction - Embeddings Model

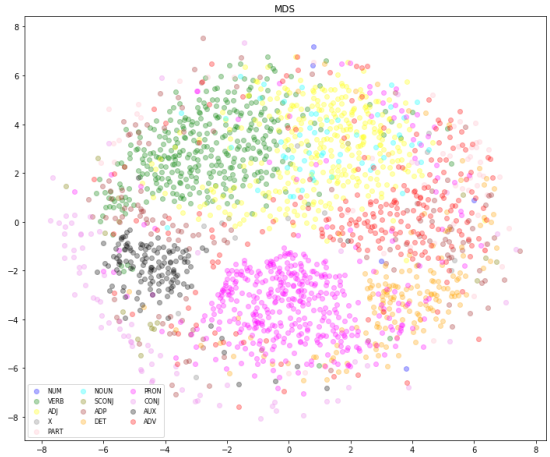


(a) Layer 1 MDS plot

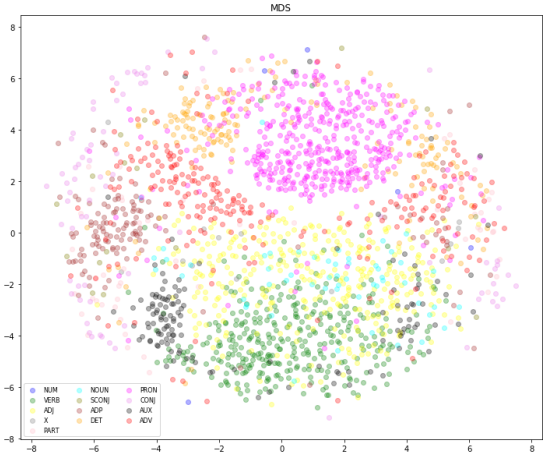
(b) Layer 2 MDS plot



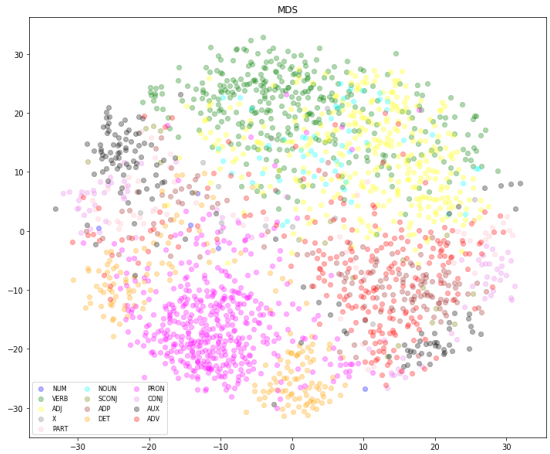
(c) Layer 3 MDS plot



(d) Layer 4 MDS plot



(e) Layer 5 MDS plot



(f) Layer 6 MDS plot

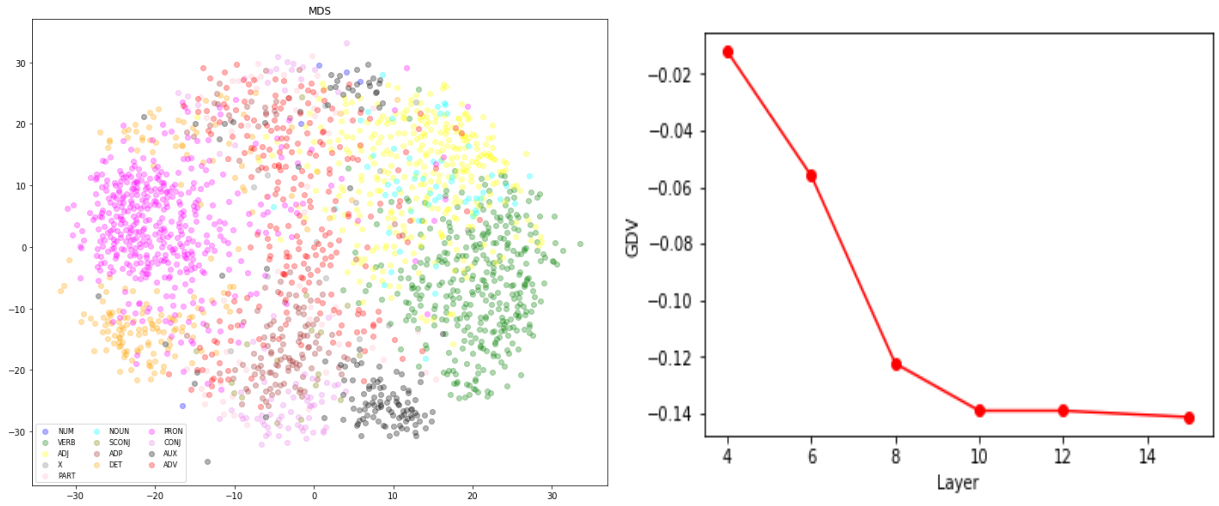
Table 4.3: MDS plots for Text Prediction with Embeddings model

Since the tenth word is predicted, given the previous nine words, the POS tags from the tenth word onwards in the test set are considered and color coded such that different POS tags or labels are visualized with different colors. MDS plots at each layer of the neural network are visualized to check if the POS tags can be considered as word classes. Table 4.2 illustrates the six MDS plots from the output of layer 1 until layer 6. While the layer 1 MDS shows random distribution of data points, we can see almost clear class separations already from layer 2 onwards. This is also validated from the general decline in the GDV curve against the layers (Figure 4.4).

Embeddings+POS Model

The model trained in section 3.4.2 is tested on the remaining 20% of the input corpus which is considered as the test set. (8072,9,398). The predicted 398 dimensional embeddings are compared with their corresponding original embeddings which were concatenated with the POS vectors. MDS plots are visualized at each layer output and the GDV is plotted against layers. Figure 4.5 illustrates the last layer MDS plot and the overall GDV plot from the network.

As stated before, this method was performed only to validate whether providing the network with a hint of the POS tag with each word would improve the results. Just like in the Text reproduction case, this model led to better GDV and MDS plots. The aim towards studying word classes in an unsupervised environment led to this model not being considered for any future task.



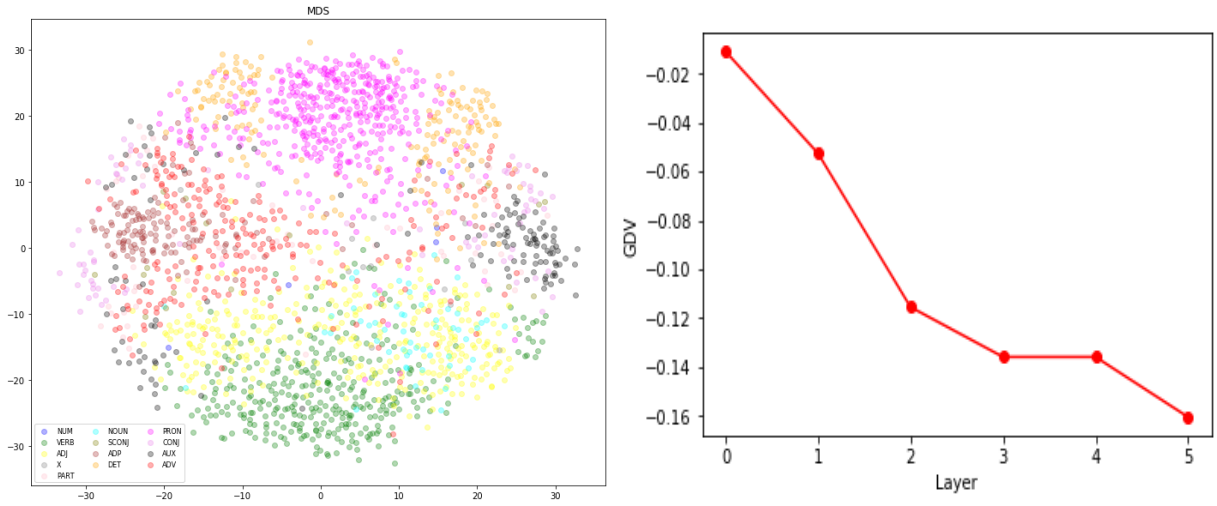
(a) Final layer MDS plot

(b) GDV plot

Figure 4.5: Layer 6 MDS plot (a) and GDV plot (b) for the Text Prediction Embeddings with POS model

Embeddings with Character Matrix Model

The model trained in section 3.4.3 is tested on the remaining 20% of the input corpus which is considered as the test set. The dimensions of the test set character matrix input and embeddings input are (8072,31,30,9) and (8072,9,384) respectively. The predicted embedding outputs of 384 dimensions are compared with their corresponding original embeddings which were passed as one of the inputs apart from the Character Matrix input. MDS plots are visualized at each layer output and the GDV is plotted against layers. Figure 4.6 illustrates the last layer MDS plot and the overall GDV plot obtained from the network.



(a) Final layer MDS plot

(b) GDV plot

Figure 4.6: Layer 6 MDS plot (a) and GDV plot (b) for the Text Prediction Embeddings with Character Matrix model

Unlike in the text reproduction case, the GDV drops well per layer. But it does not outperform other models even after combining syntax(character matrices) and semantics(word embeddings). Therefore, even for the text prediction case, embeddings are sufficient to build a good text reproduction network with increasing class separation against layers of the network.

Useful Statistics

Table 4.4 states the Test accuracy and word similarity obtained for all the 3 explained models for the task of single word text prediction.

Cosine similarity measure is used to compute the word similarity between the predicted words and their corresponding words at the input. Similarly, test accuracy is measured by calculating the euclidean distance between each predicted word and the entire test set corpus such that the predicted word is compared to the word with the least distance from it. Since we have discarded the Embeddings+POS model, the Embeddings-only model performs the best.

One common remarkable result observed for almost all the above models is that the neural network organizes its internal representations according to the word classes of the input words (for text reproduction) or the predicted words(for text prediction) despite no

Model	Test Accuracy	Word Similarity
Embeddings Model	42.675%	73.558%
Embeddings + POS Model	42.984%	74.276%
Embeddings with Character Matrix Model	41.537%	72%

Table 4.4: Statistics for Text Prediction Models

information regarding the POS tags passed into the network in case of the Embeddings model and the Embeddings+CM model for both the tasks. This lays the foundation for word class detection in an unsupervised environment.

4.1.3 Case Study : 'Particles'

Now that POS tags have been established as good word classes for both the text reproduction and text prediction cases, the attention is diverted to studying a specific POS tag, namely the 'Particle' or 'PART' illustrated in the MDS plots. The text prediction embeddings-only model of section 3.4.1 is considered for this case study.

Unlike nouns, verbs and adjectives, particles have no solid definition. The Oxford guide to English [Eas94] does not consider the particle as a separate POS tag, but considers adverbs used with verbs as particles. Other sources [McA05] define particles as words which no not fit under other word classes. They can be further classified into adverbial, infinitival, negative and imperative particles. Different languages have their own definitions and categories for particles.

In this thesis, the internal representations of words tagged as particles at the final layer of the neural network are plotted, reverse engineered and researched to check for any subclasses. The intent is to check if there are fixed subclasses which have not been covered in literature.

Figure 4.7 shows the final layer MDS plot of only the words tagged as particles by the spaCy German library. Each word is number coded for easier analysis.

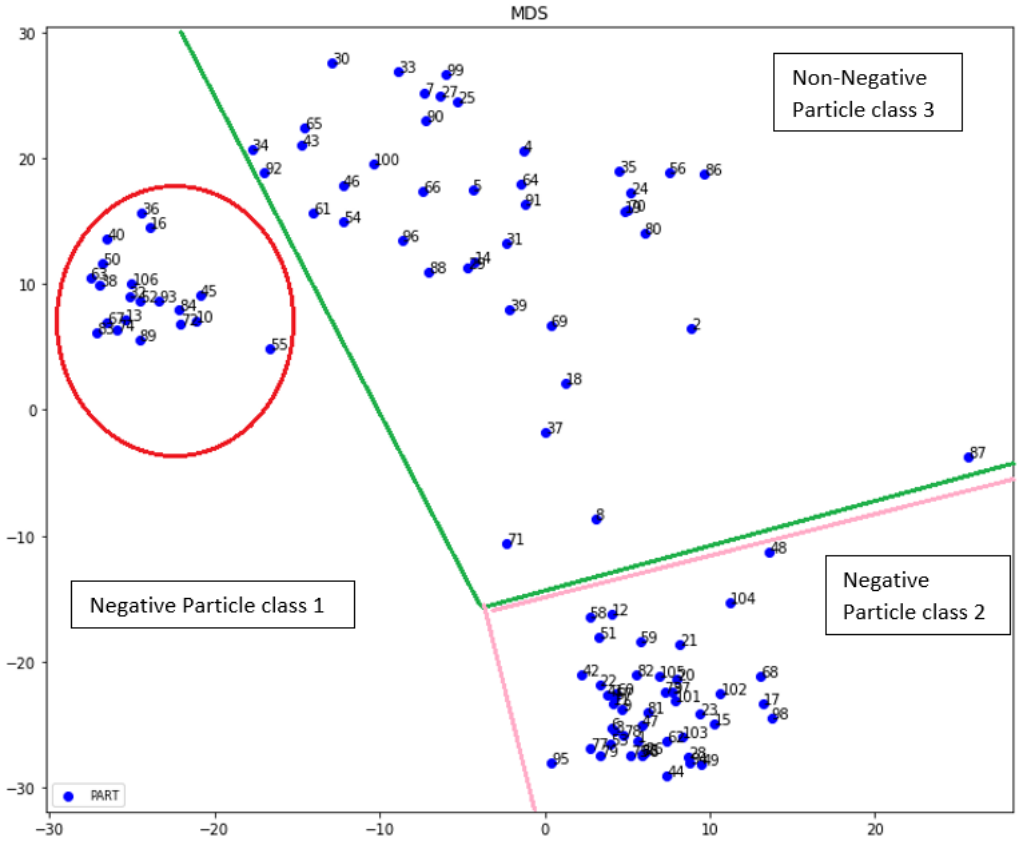


Figure 4.7: Final Layer MDS plot of Particles in the German play

After analyzing the words from each data point, 3 main sub-classes were identified :

- Negative Paricle Class 1: Mainly one word : 'nicht'.
- Negative Paricle Class 2: Mainly one word : 'nicht'.
- Non Negative Particle class 3 : All other particles such as 'ab','vor','zu','bitte',etc..

Some of the particles and the phrases in which they are used in the German corpus are tabulated in Table 4.5. The particle is depicted in Italics, and the number in brackets refers to the location of that particle's data point in the MDS plot of Figure 4.7.

Subclass	Phrases (Data point number in plot)
Negative Particle class 1	– ihr *nicht* mehr (50) – hatte *nicht* einmal (67) – können *nicht* mein (93)
Negative Particle class 2	– leike *nicht* antastbar (12) – ihr *nicht* biete (44) – esgeben *nicht* brechen (59)
Non-Negative Particle class 3	– kein *licht* an (2) – toleranz *zu* übergehen (35) – arbeit *ab* dort (100)

Table 4.5: Particle phrases belonging to 3 subclasses

4.1.4 One word Prediction with different Input configurations

The Embeddings model of Text Prediction task was discussed in section 4.1.2 where the embeddings of nine words were passed as the input to the neural network, and the tenth word is predicted.

In this section, we analyze the impact of passing different inputs from 1 word up to 8 words as the input, on the MDS and GDV plots. This is possible by making the preceding embeddings zeroes. For example, if we pass 3 words as the input, the first 6 words are represented by 384 dimensional zero vectors and the remaining 3 positions are filled by 384 dimensional embeddings of the 3 words. The predicted word for this scenario will be word 4.

Table 4.6 illustrates the last layer MDS plot and GDV plot for each type of input provided. The GDV and MDS plots for all inputs are observed to be very similar. Since the GDV plots show a general expected decreasing trend, only the final layer GDVs are considered for further analysis.

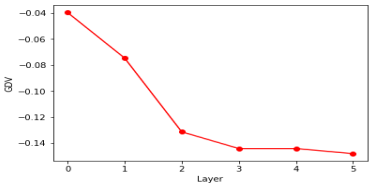
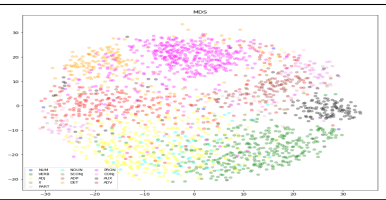
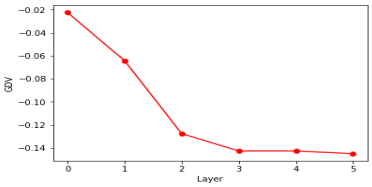
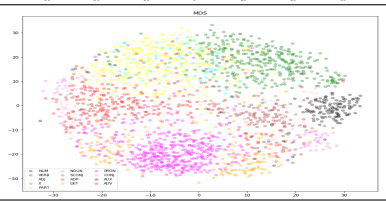
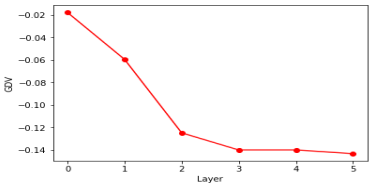
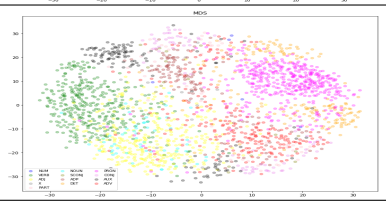
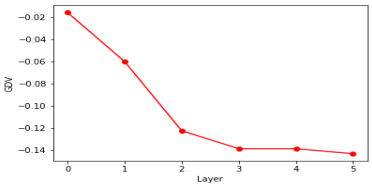
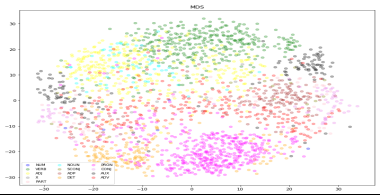
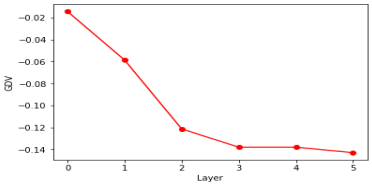
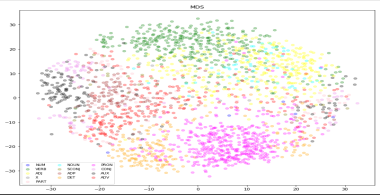
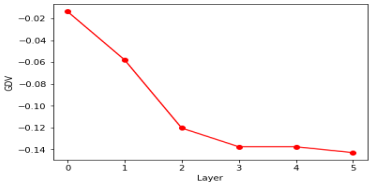
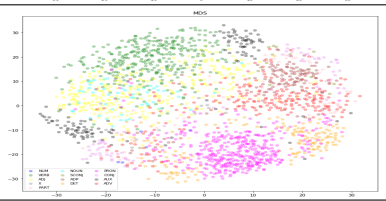
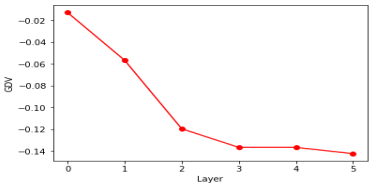
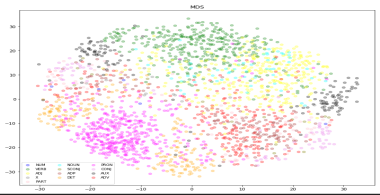
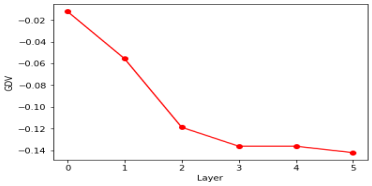
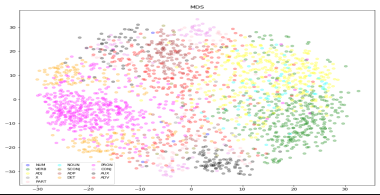
Input	GDV plot	Last layer MDS plot	Final GDV
[0,0,0,0,0,0,0,0,0,w1]			-0.14815
[0,0,0,0,0,0,0,0,w1,w2]			-0.14501
[0,0,0,0,0,0,0,w1,w2,w3]			-0.14316
[0,0,0,0,0,0,w1,w2,w3,w4]			-0.14311
[0,0,0,0,0,w1,w2,w3,w4,w5]			-0.1423
[0,0,0,0,w1,w2,w3,w4,w5,w6]			-0.14303
[0,0,0,w1,w2,w3,w4,w5,w6,w7]			-0.14245
[0,0,w1,w2,w3,w4,w5,w6,w7,w8]			-0.1421

Table 4.6: 1 word text prediction plots for different inputs

Figure 4.8 illustrates the plot between the number of words (or non zero embeddings) in the 9 word input vs the final layer GDV. Surprisingly, the trend shows that lesser words or more zero embeddings in the input leads to better GDV. This means that having lesser information at the input side leads to better class separation at the output of each neural network layer.

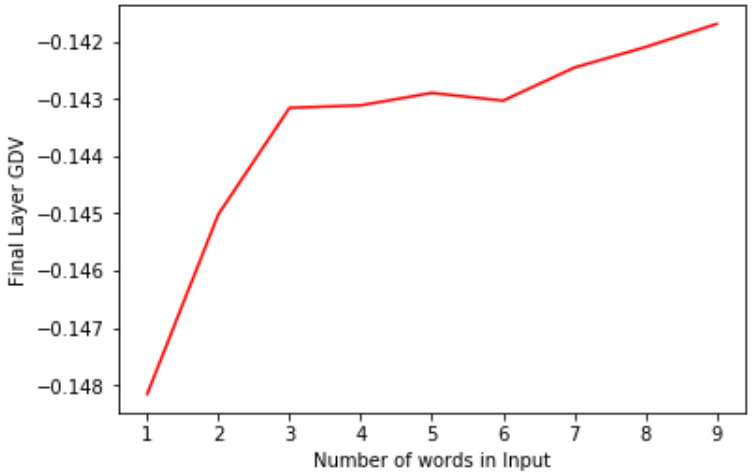


Figure 4.8: Number of words in the input vs Final Layer GDV curve

4.1.5 Text Prediction : 2 words

Sections 4.1.2 until 4.1.4 covered prediction of the tenth word, given the first nine words as the input. The results so far have established that POS tags are indeed the word classes.

Now, the aim is to check if POS tag pairs can also be visualized as word classes. In other words, the trained neural network introduced in section 3.5 is tested on the remaining 20% of the data. Instead of predicting one word, the network predicts the tenth **and** eleventh words, given the first nine words at the input. The dimension of the test input data is (8071,9,384).

13 unique POS tags were used as class labels in the 1 word prediction case. In order to test for POS tag pairs, there would be $13 \times 13 = 169$ POS tag pair combinations. It is almost impossible to assign 169 unique colors for each of the pairs and visualize the messy plots. Therefore, only the top 10 frequently occurring POS tag pairs are plotted and visualized. A good way to visualize the number of occurrences of each POS tag pair is using a 13×13 heatmap, as illustrated in Figure 4.9.

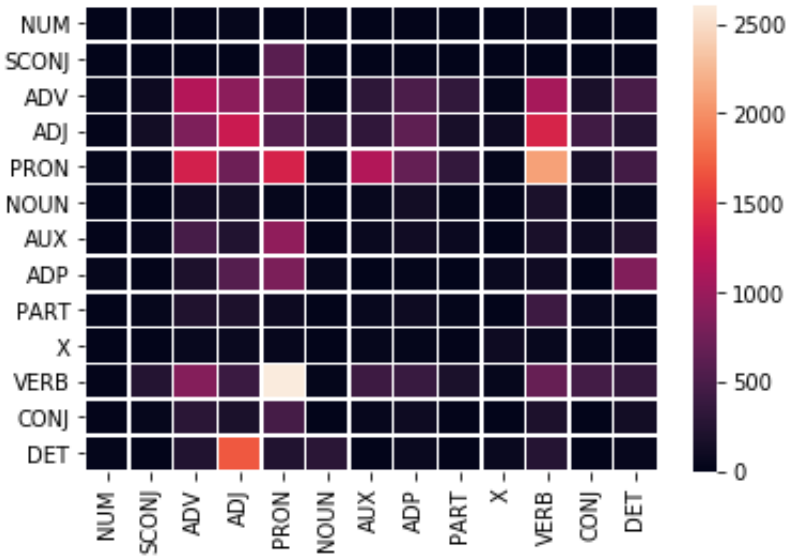


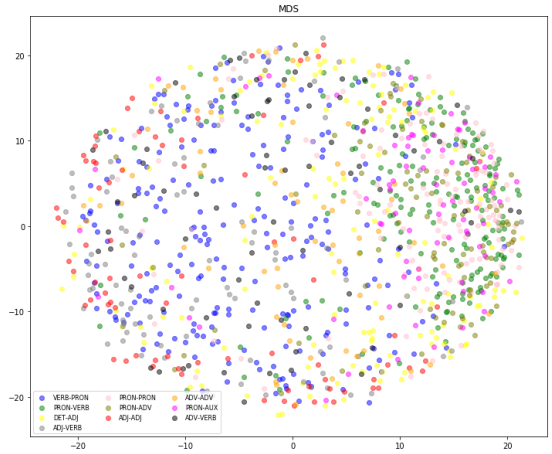
Figure 4.9: Heatmap of POS tag pair occurrences

The rows refer to the first POS tag in the pair and the columns refer to the second POS tag. Lighter colors in the heatmap imply higher frequency of occurrence. For example, the pair of VERB-PRON is depicted by a whitish color. Based on this, the top 10 frequent POS tag pairs are tabulated in table 4.7.

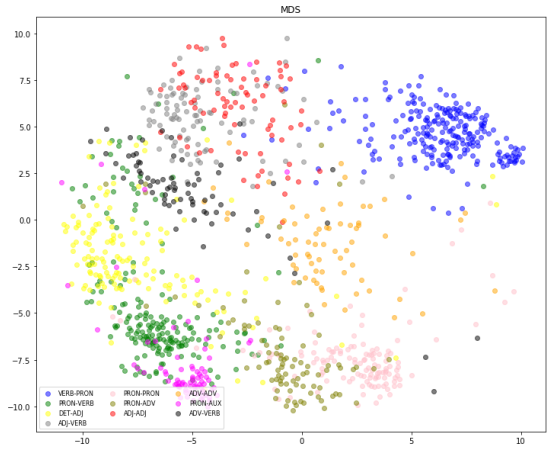
Top 10 POS tag pairs	**Number of occurrences**
VERB - PRONOUN	2601
PRONOUN - VERB	2112
DETERMINER - ADJECTIVE	1694
ADJECTIVE - VERB	1384
PRONOUN - PRONOUN	1381
PRONOUN - ADJECTIVE	1353
ADJECTIVE - ADJECTIVE	1293
ADVERB - ADVERB	1149
PRONOUN - AUXILIARY	1140
ADVERB - VERB	1073

Table 4.7: Top 10 POS tag pairs and their frequency of occurrence

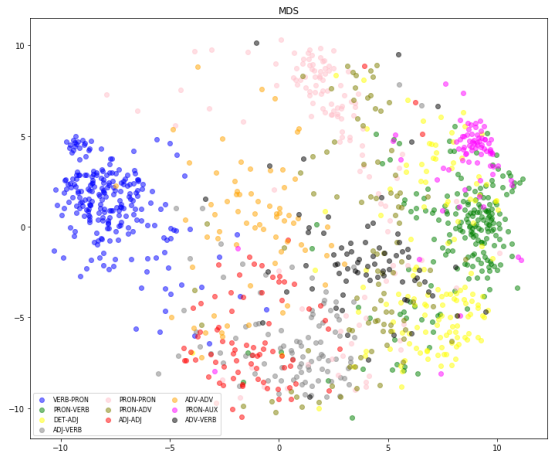
Table 4.8 illustrates the layer-wise MDS plots. We observe a good class separation for all 10 POS tag pairs, indicating that POS pairs can also be considered as word classes. The GDV plot in Figure 4.10 validates the results with a standard decline.



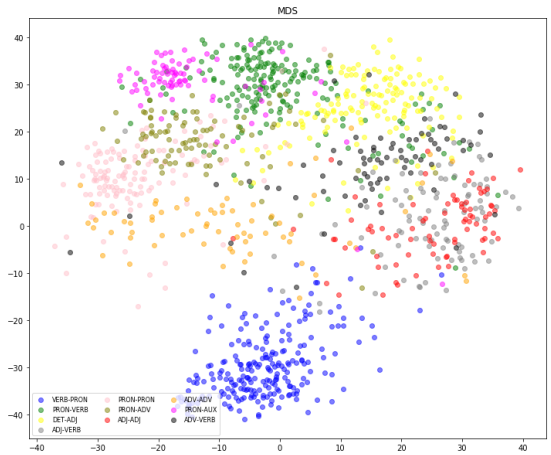
(a) Layer 1 MDS plot



(b) Layer 2 MDS plot



(c) Layer 3 MDS plot



(d) Layer 4 MDS plot

Table 4.8: MDS plots for 2 words Text Prediction model

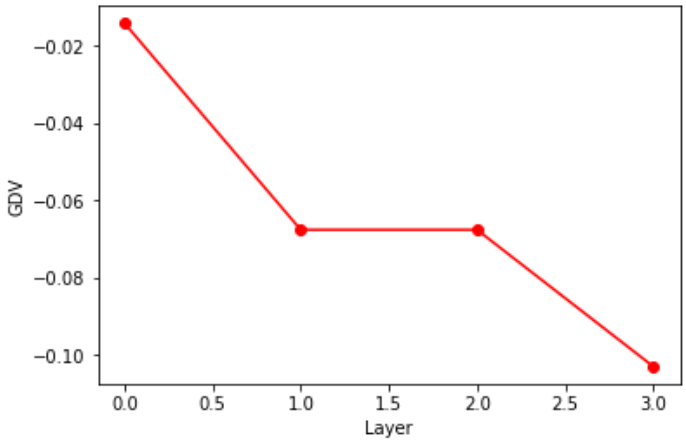


Figure 4.10: GDV plot for 2 words Text Prediction model

4.1.6 Two words Prediction with different Input configurations

The Embeddings model of the 2 words Text Prediction task was discussed in the previous section where the embeddings of nine words were passed as the input to the neural network, while the tenth and eleventh words are predicted. In this section, we analyze the impact of passing different inputs from 1 word up to 8 words as the input, on the MDS and GDV plots. This is possible by making the preceding embeddings zeroes. The inputs are prepared in the same way as mentioned in section 4.1.4. Table 4.9 illustrates the last layer MDS plot and GDV plot for each type of input provided. Since the GDV plots show a general decreasing trend, only the final layer GDVs are considered for further analysis.

Figure 4.11 illustrates the plot between the number of words (or non zero embeddings) in the 9 word input vs the final layer GDV. Surprisingly, the trend shows that lesser words or more zero embeddings in the input leads to better GDV. The results are in line with section 4.1.4 for the 1 word prediction case.

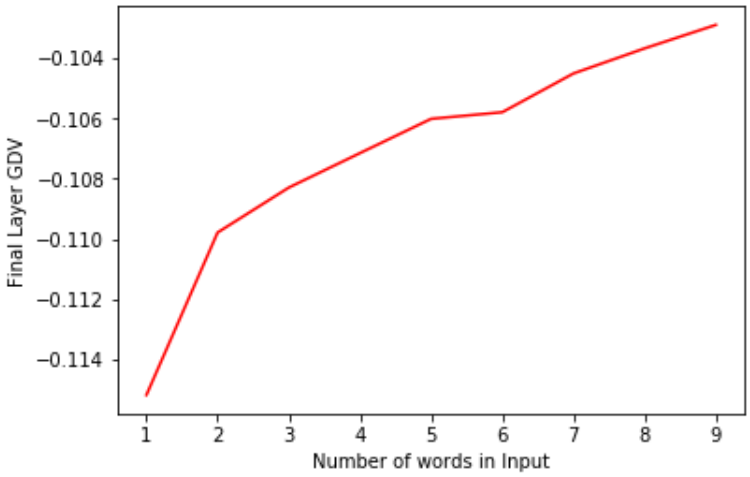


Figure 4.11: Number of words in the input vs Final Layer GDV curve

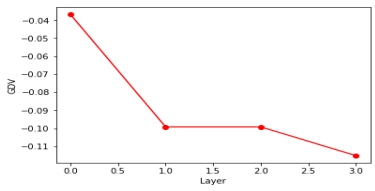
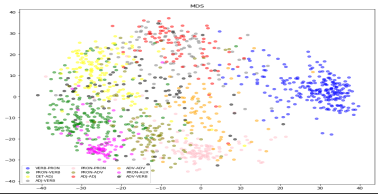
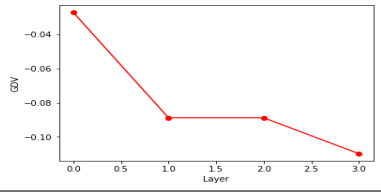
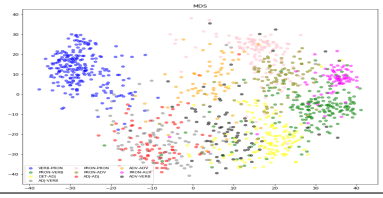
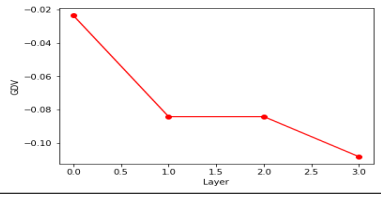
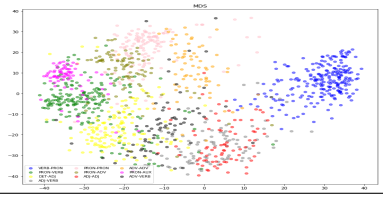
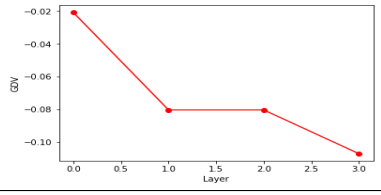
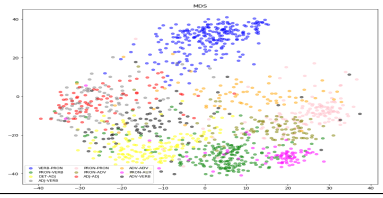
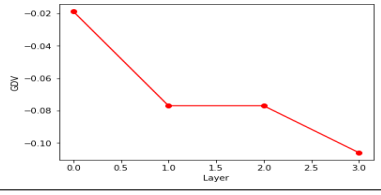
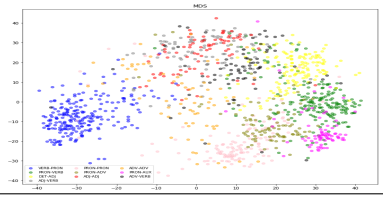
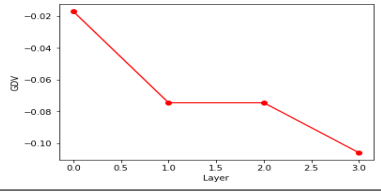
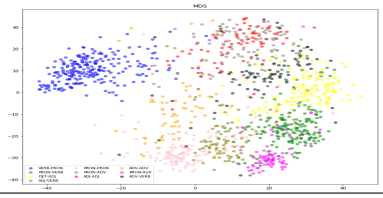
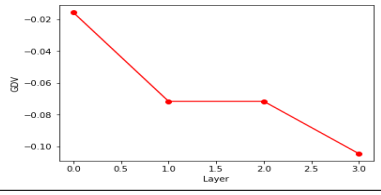
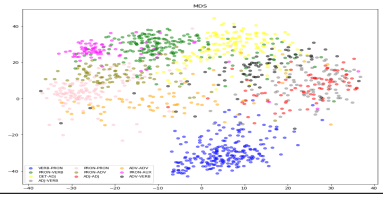
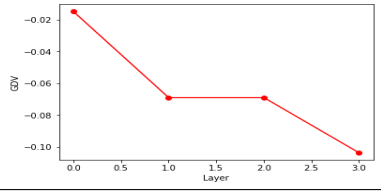
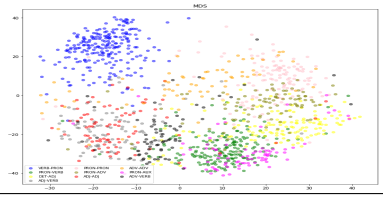
Input	GDV plot	Last layer MDS plot	Final GDV
[0,0,0,0,0,0,0,0,w1]			-0.11519
[0,0,0,0,0,0,0,w1,w2]			-0.10979
[0,0,0,0,0,0,w1,w2,w3]			-0.10829
[0,0,0,0,0,w1,w2,w3,w4]			-0.10715
[0,0,0,0,w1,w2,w3,w4,w5]			-0.10601
[0,0,0,w1,w2,w3,w4,w5,w6]			-0.10579
[0,0,w1,w2,w3,w4,w5,w6,w7]			-0.1045
[0,w1,w2,w3,w4,w5,w6,w7,w8]			-0.10367

Table 4.9: 2 words text prediction plots for different inputs

4.1.7 Collocations vs Non Collocations

In this section, we test whether collocations and non collocations are represented in different parts of space by the neural network. Both the bigram and trigram models are tested, i.e, we plot the MDS for 2 word and 3 word collocations. Here, we only consider the 2 word or bigram case for the German play.

The 2 dimensional bigram matrix is of dimensions (6117,6117) is built, where 6117 refers to the number of unique words in the entire German play corpus. The sum of all the entries in the bigram matrix will be equal to the total number of words in the corpus, i.e, 40460.

Figure 4.12 shows the number of occurrences of a bigram plotted against the natural log frequency of occurrences to understand the distribution. As expected, more bigrams have few occurrences.

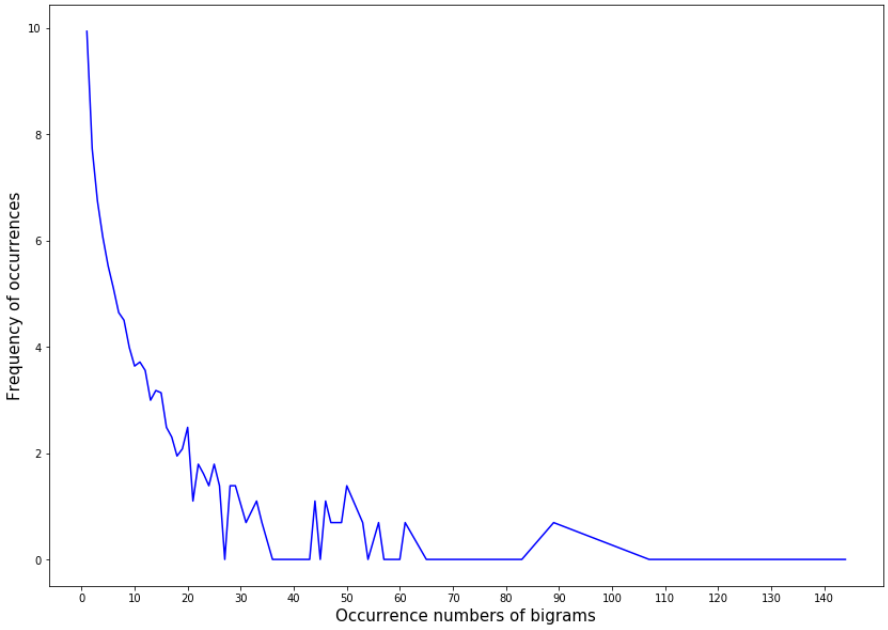


Figure 4.12: Occurrences of bigrams vs Log Frequency

Since collocations lie in between very frequently occurring bigrams and sparsely occurring bigrams, it makes sense to consider a threshold between 10 and 20 occurrences. Experimentally, a threshold of 13 led to the best class separations for this corpus. Therefore, the bigrams which occur less than 13 times are considered as non collocations, and the rest are grouped as collocations.

The traditional single word text prediction neural network from section 3.4.1 where nine embeddings are passed into the input is used here. However, instead of 13 class labels corresponding to the POS tags, we now use 2 class labels : One for collocations (visualized in orange) and the other for non collocations (visualized in blue). Table 4.10 illustrates the MDS plots from all six layers of the neural network. We observe some level of class separation between collocations and non collocations, though not at the same degree as the POS tags case. The majority of collocation pairs are grouped at the bottom right part of the last layer MDS plot. But a lot of collocation data points are also spread across the plot uniformly. The GDV plot in Figure 4.13 shows an overall decline, except for the hiccup at the fourth layer.

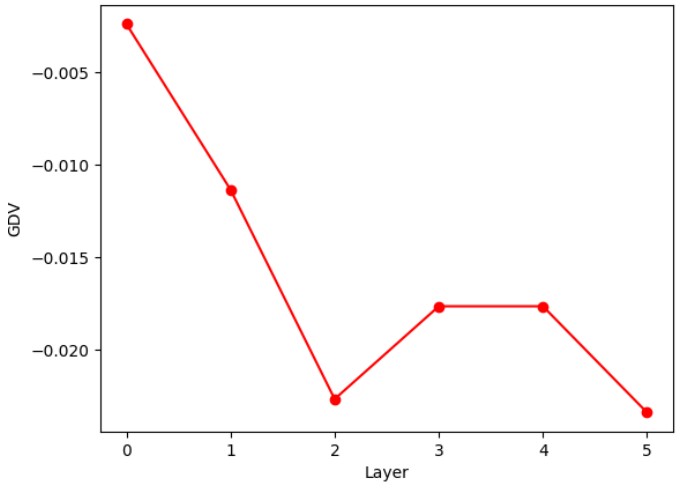
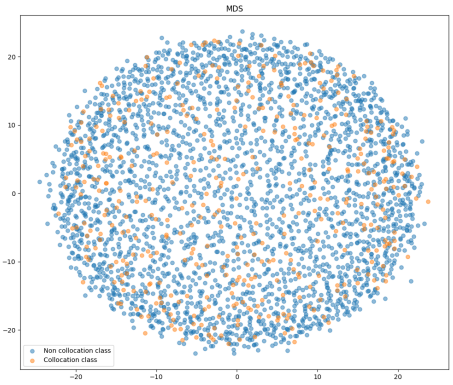
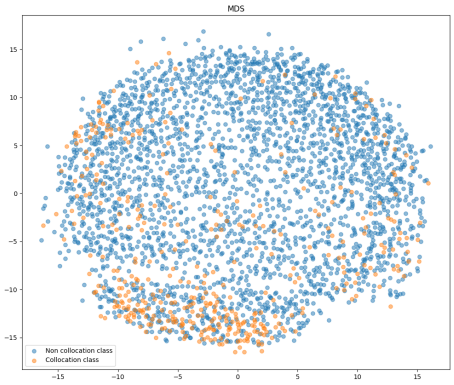


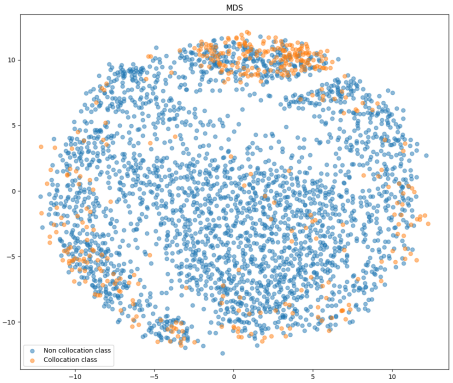
Figure 4.13: GDV plot for Collocations vs Non Collocations task: German corpus



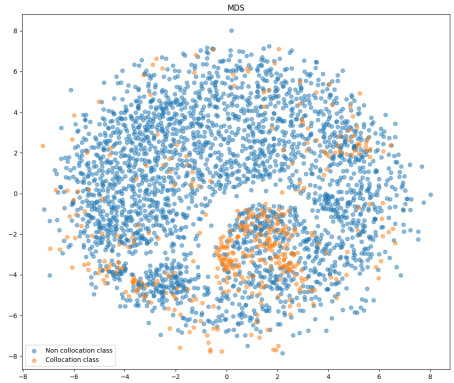
(a) Layer 1 MDS plot



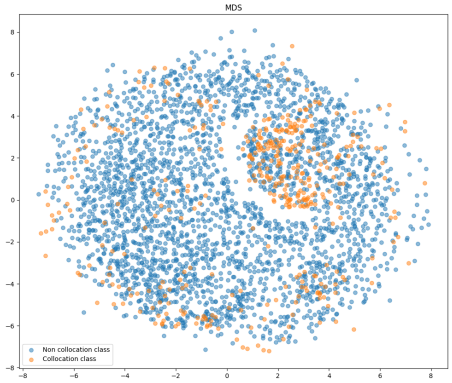
(b) Layer 2 MDS plot



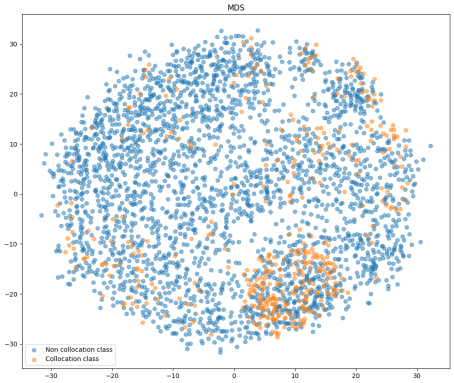
(c) Layer 3 MDS plot



(d) Layer 4 MDS plot



(e) Layer 5 MDS plot



(f) Layer 6 MDS plot

Table 4.10: MDS plots for Collocations vs Non Collocations task: German corpus

4.1.8 Collocations vs Non Collocations with different Input configurations

In the previous section, the single word text prediction model was used to visualize collocations and non collocations. In this section, we analyze the impact of passing different inputs from 1 word up to 8 words as the input, on the MDS and GDV plots.This is possible by making the preceding embeddings zeroes. The inputs are prepared in the same way as mentioned in section 4.1.4. Table 4.11 illustrates the last layer MDS plot and overall GDV plot for each type of input provided. Since the GDV plots show a general decreasing trend, only the final layer GDVs are considered for further analysis.Figure 4.14 illustrates the plot between the number of words (or non zero embeddings) in the 9 word input vs the final layer GDV. Unlike the results in sections 4.1.4 and 4.1.6, the GDV decreases after a threshold of 2 words, with an increase in the number of non zero embeddings in the input. It means that more word information in the input leads to better class separation for the collocations vs non collocations task.

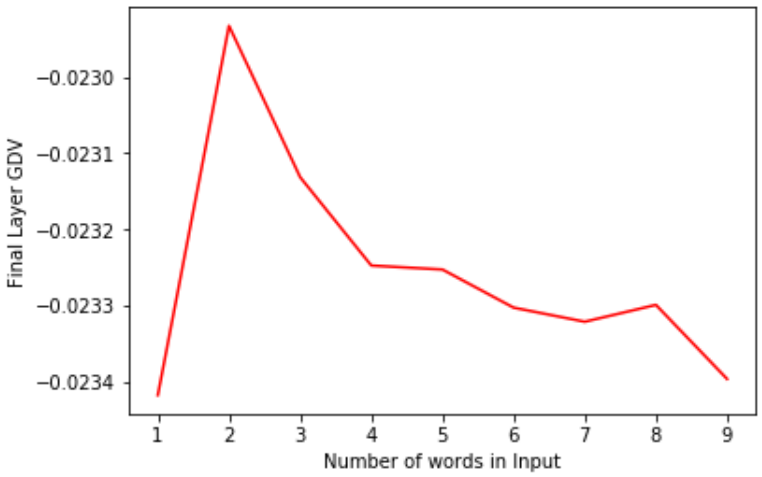


Figure 4.14: Number of words in the input vs Final Layer GDV curve

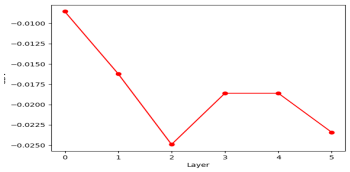
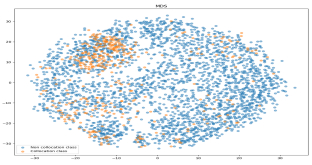
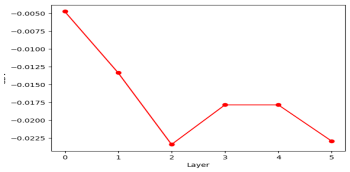
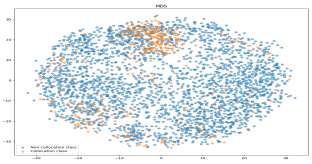
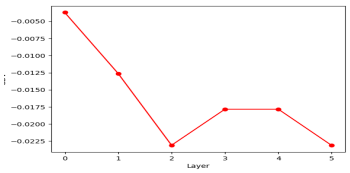
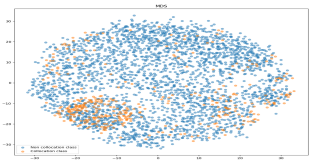
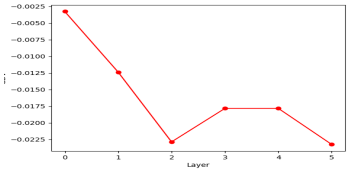
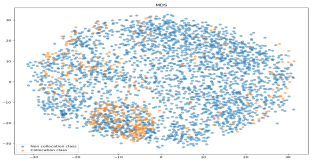
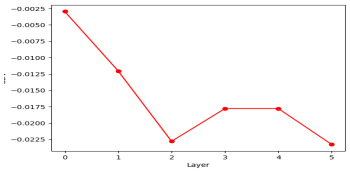
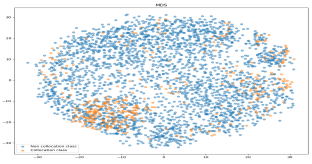
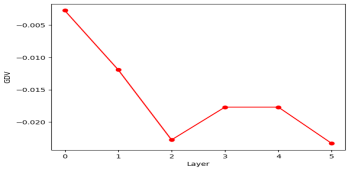
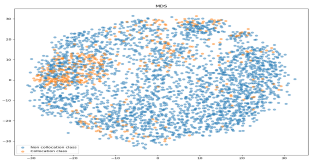
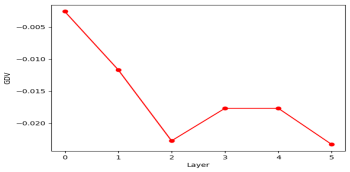
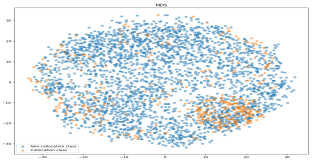
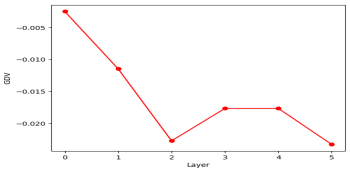
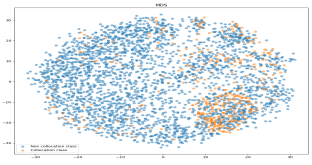
Input	GDV plot	Last layer MDS plot	Final GDV
[0,0,0,0,0,0,0,0,w1]			-0.02342
[0,0,0,0,0,0,0,w1,w2]			-0.02293
[0,0,0,0,0,0,w1,w2,w3]			-0.02313
[0,0,0,0,0,w1,w2,w3,w4]			-0.02325
[0,0,0,0,w1,w2,w3,w4,w5]			-0.02325
[0,0,0,w1,w2,w3,w4,w5,w6]			-0.0233
[0,0,w1,w2,w3,w4,w5,w6,w7]			-0.02332
[0,w1,w2,w3,w4,w5,w6,w7,w8]			-0.0233

Table 4.11: Text prediction plots for Collocations vs Non Collocations case

4.2 English Novel

Most of the tasks carried out for the German play will be repeated for the English novel to see if similar results are reproduced in a different language. Instead of re-building 3 models each for text reproduction and text prediction, only the text prediction embeddings-only model will be used for all the tasks.

4.2.1 Text Prediction Results: One word

The model trained in section 3.4.1 is tested on the remaining 20% of the input English corpus, which is considered as the test set. The dimension of the test input data is therefore (9484,9,384). The predicted embeddings are compared with their corresponding original embeddings of the test set, and useful metrics such as the word similarity and test accuracy are tabulated.

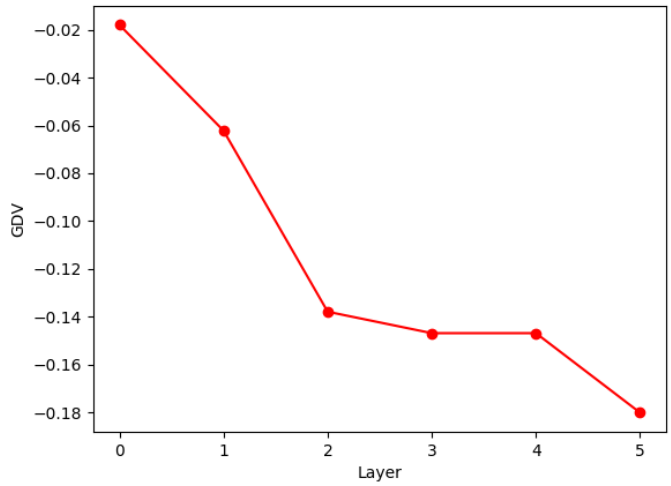
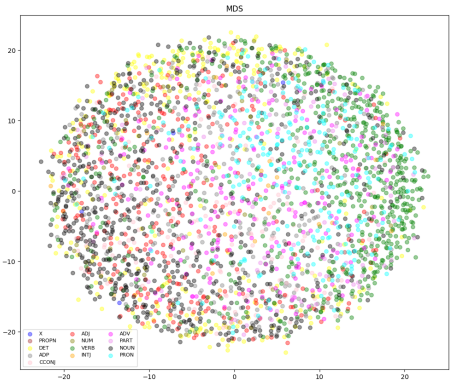
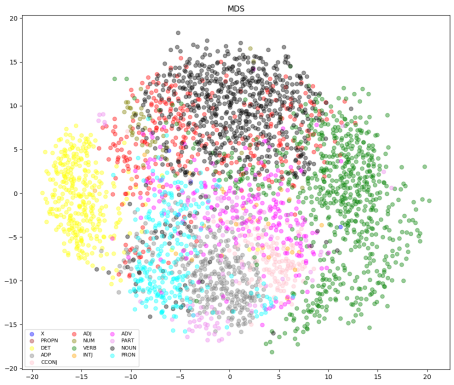


Figure 4.15: Final Layer MDS plot of Particles in the English novel

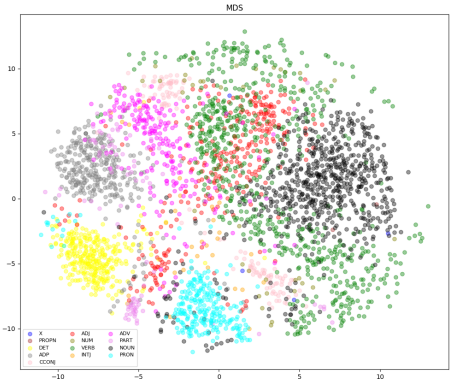
MDS plots at each layer of the neural network are visualized to check if the POS tags can be considered as word classes. Table 4.12 illustrates the six MDS plots from the output of layer 1 until layer 6. While the layer 1 MDS shows random distribution of data points, we can see almost clear class separations already from layer 2 onwards. This is also validated from the general decline in the GDV curve against the layers (Figure 4.15).



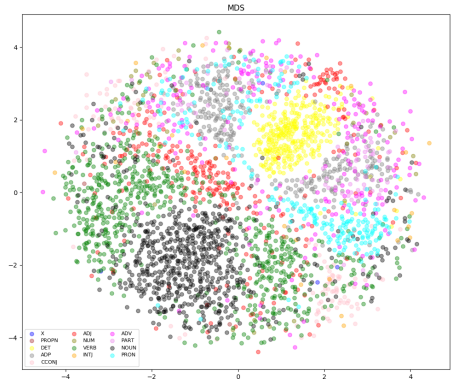
(a) Layer 1 MDS plot



(b) Layer 2 MDS plot



(c) Layer 3 MDS plot



(d) Layer 4 MDS plot

(e) Layer 5 MDS plot

(f) Layer 6 MDS plot

Table 4.12: MDS plots for Text Prediction (English corpus)

4.2.2 Case Study : 'Particles'

Now that POS tags have again been established as good word classes for the English text prediction case, the attention is diverted to studying a specific POS tag, namely the 'Particle' or 'PART' illustrated in the MDS plots.

As discussed in section 4.1.3, the internal representations of words tagged as particles at the final layer of the neural network are plotted, reverse engineered and researched to check for any subclasses. The intent is to check if there are fixed subclasses which have not been covered in literature.

Figure 4.16 shows the final layer MDS plot of only the words tagged as particles by the spaCy English library. Each word is number coded for easier analysis.

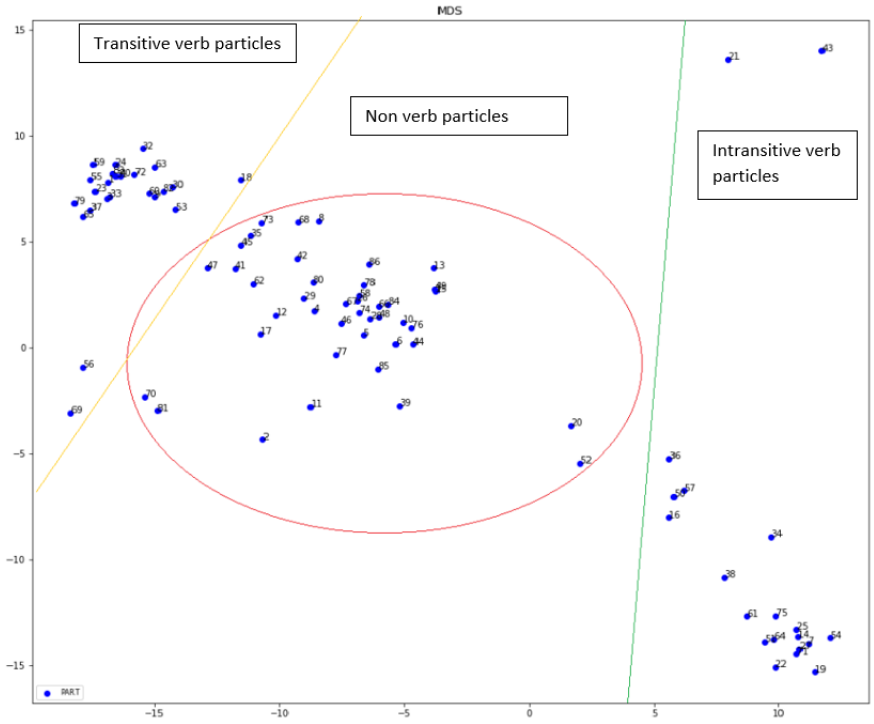


Figure 4.16: Final Layer MDS plot of Particles in the English novel

After analyzing the words from each data point, 3 main subclasses were identified :

- Transitive Verb Paricle Class 1: Words like 'give', 'invite', 'kick',etc..
- Intransitive Verb particle Class 2: Words like 'strike', 'run', 'stray', etc..
- Non Verb particle Class 3: Words like 'attention' , 'the', 'now', etc..

Since classes 1 and 2 contain verb particles, many words are common between them. However, they vary in the context of usage of the words. Class 1 contains verb particles in sentences with both a subject and an object, while Class 2 contains particles used in sentences with only a subject but no object. Class 3 is an amalgamation of all other non verb particles.

Some of the particles and the phrases in which they are used in the English corpus are tabulated in Table 4.13. The particle is depicted in Italics, and the number in brackets refers to the location of that particle's data point in the MDS plot of Figure 4.16.

Subclass	**Phrases (Data point number in plot)**
Transitive Verb Particle class 1	– to *create* an (23) – to *invite* you (32) – to *give* it (59)
Intransitive Verb Particle class 2	– to *say* bellowed (19) – to *come* up (25) – to *clear* away (38)
Non-Verb Particle class 3	– aside *the* ineffectual (2) – off *now* what (11) – down *into* a (58)

Table 4.13: Particle phrases belonging to 3 subclasses

4.2.3 One word Prediction with different Input configurations

As covered in section 4.1.4, we analyze the impact of passing different inputs from 1 word up to 8 words, on the MDS and GDV plots. This is possible by zeroing the preceding embeddings.

Table 4.14 illustrates the last layer MDS plot and GDV plot for each type of input provided. The GDV and MDS plots for all inputs are observed to be very similar. Since the GDV

plots show a general expected decreasing trend, only the final layer GDVs are considered for further analysis.

Figure 4.17 illustrates the plot between the number of words (or non zero embeddings) in the 9 word input vs the final layer GDV. Like the plot in section 4.1.4, the overall trend here shows an increasing GDV curve. However, it is not strictly increasing due to the decrease between 2 and 4 non zero embeddings at the input. The least GDV is observed for 1 word and 4 word input cases respectively.

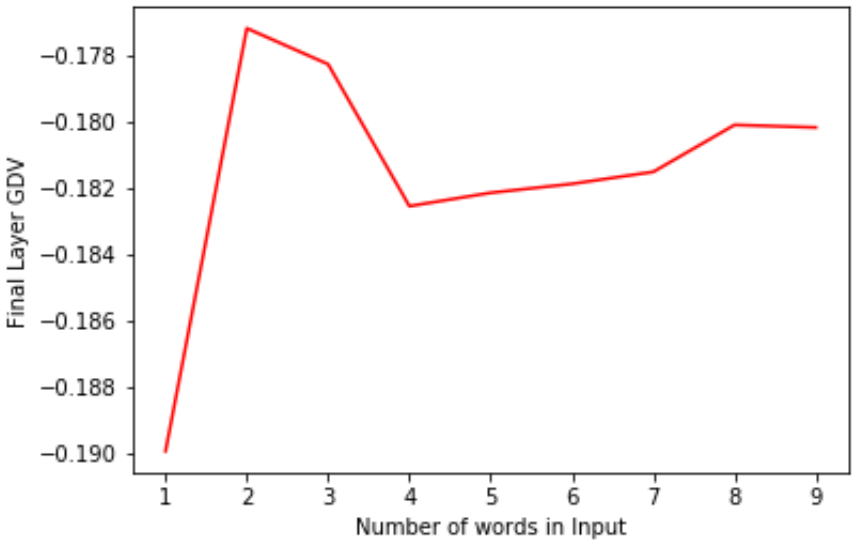


Figure 4.17: Number of words in the input vs Final Layer GDV curve for 1 word prediction case (English)

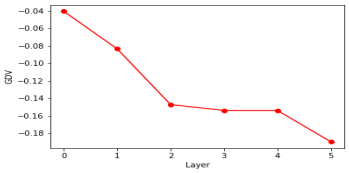
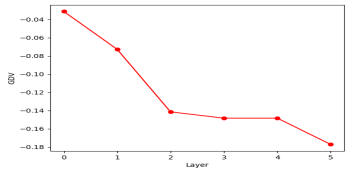
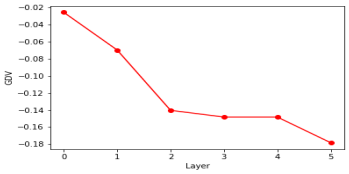
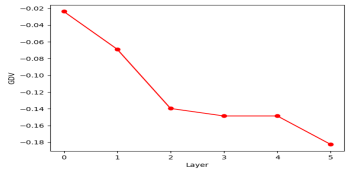
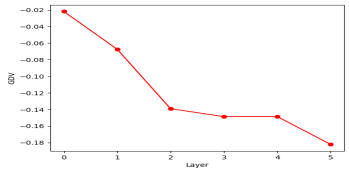
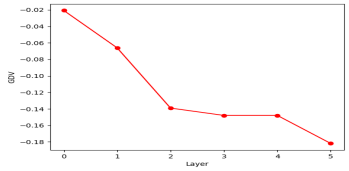
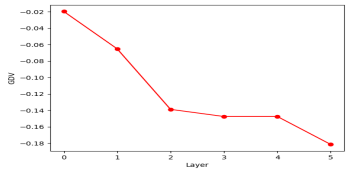
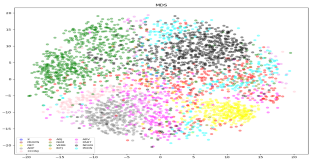
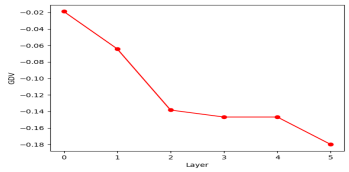
Input	GDV plot	Last layer MDS plot	Final GDV
[0,0,0,0,0,0,0,0,w1]			-0.18995
[0,0,0,0,0,0,0,w1,w2]			-0.1772
[0,0,0,0,0,0,w1,w2,w3]			-0.17829
[0,0,0,0,0,w1,w2,w3,w4]			-0.18256
[0,0,0,0,w1,w2,w3,w4,w5]			-0.18216
[0,0,0,w1,w2,w3,w4,w5,w6]			-0.18189
[0,0,w1,w2,w3,w4,w5,w6,w7]			-0.18153
[0,w1,w2,w3,w4,w5,w6,w7,w8]			-0.18012

Table 4.14: 1 word text prediction plots for different inputs

4.2.4 Text Prediction : 2 words

Similar to section 4.1.5, we now check if POS tag pairs can also be visualized as word classes. In other words, the trained neural network introduced in section 3.5 is tested on the remaining 20% of the data. Instead of predicting one word, the network predicts the tenth and eleventh words, given the first nine words at the input. The dimension of the test input data is (9483,9,384).

13 unique POS tags used as class labels in the 1 word prediction case would lead to $13 \times 13 = 169$ POS tag pair combinations. TO avoid messy, overlapping MDS plots containing 169 classes, only the top 10 frequently occurring POS tag pairs are plotted and visualized. A good way to visualize the number of occurrences of each POS tag pair is using a 13×13 heatmap, as illustrated in Figure 4.18. Note that the 'PUNCT' POS tag is ignored since it contains irrelevant words which do not contribute to the performance of the network.

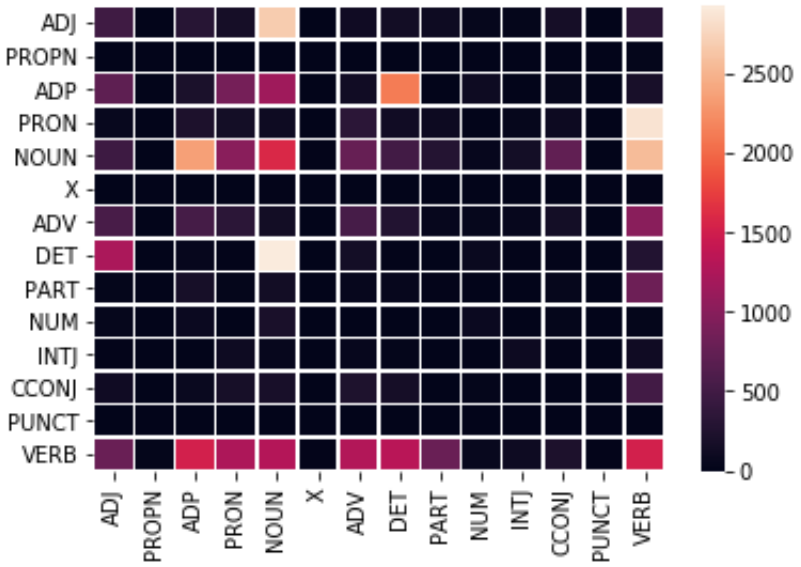


Figure 4.18: Heatmap of POS tag pair occurrences (English)

The rows refer to the first POS tag in the pair and the columns refer to the second POS tag. Lighter colors in the heatmap imply higher frequency of occurrence. For example, the pair of DET-NOUN is depicted by a whitish color. Based on this, the top 10 frequent POS tag pairs are tabulated in table 4.15.

Top 10 POS tag pairs	Number of occurrences
DETERMINER - NOUN	2922
PRONOUN - VERB	2860
ADJECTIVE - NOUN	2676
NOUN - VERB	2558
NOUN - ADPOSITION	2360
ADPOSITION - DETERMINER	2123
NOUN - NOUN	1592
VERB - VERB	1529
VERB - ADPOSITION	1523
VERB - DETERMINER	1328

Table 4.15: Top 10 POS tag pairs and their frequency of occurrence (English)

Table 4.16 illustrates the layer-wise MDS plots. We observe a good class separation for all 10 POS tag pairs, indicating that POS pairs can also be considered as word classes. The GDV plot in Figure 4.19 validates the results with a standard decline. These results are in line with the German play results for 2 word prediction.

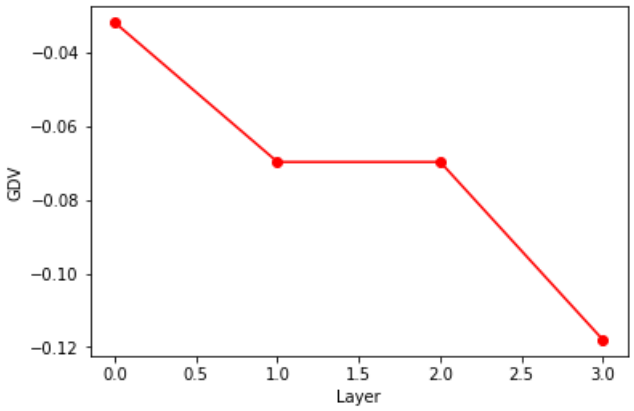
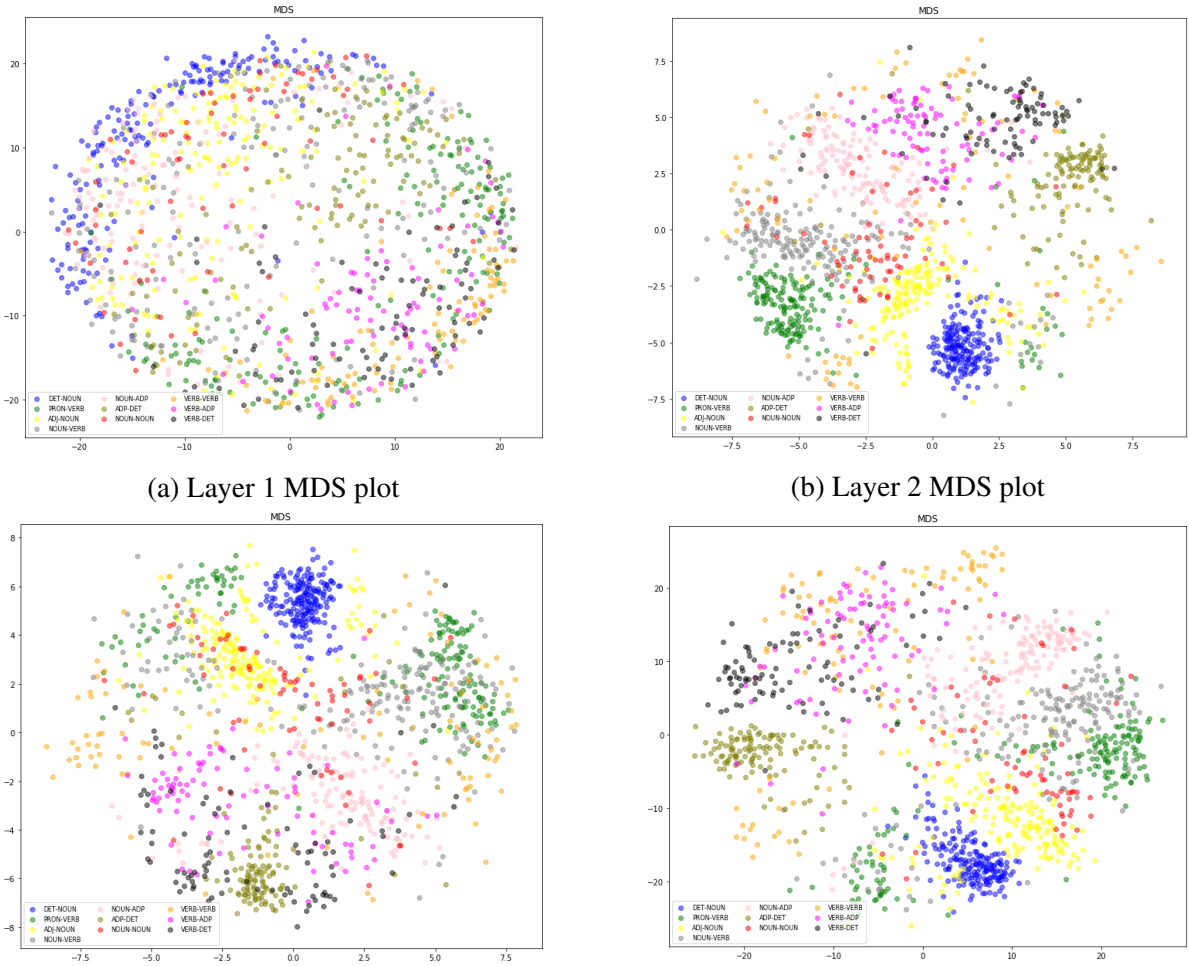


Figure 4.19: GDV plot for 2 words Text Prediction model (English)



(a) Layer 1 MDS plot

(b) Layer 2 MDS plot

(c) Layer 3 MDS plot

(d) Layer 4 MDS plot

Table 4.16: MDS plots for 2 words Text Prediction model (English)

4.2.5 Two words Prediction with different Input configurations

The Embeddings model of the 2 words Text Prediction task was discussed in the previous section where the embeddings of nine words were passed as the input to the neural network, while the tenth and eleventh words are predicted. Just like in section 4.1.6, we analyze the impact of passing different inputs from 1 word up to 8 words as the input, on the MDS and GDV plots. This is possible by making the preceding embeddings zeroes. Table 4.16 illustrates the last layer MDS plot and GDV plot for each type of input provided. Since the GDV plots show a general decreasing trend, only the final layer GDVs are considered for further analysis.

Figure 4.20 illustrates the plot between the number of words (or non zero embeddings) in

the 9 word input vs the final layer GDV. Surprisingly, the trend shows that lesser words or more zero embeddings in the input leads to better GDV. The results are in line with section 4.2.3 for the 1 word prediction case.

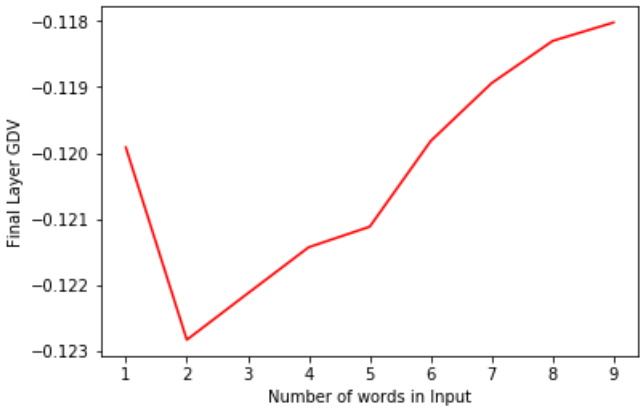


Figure 4.20: Number of words in the input vs Final layer GDV curve (English)

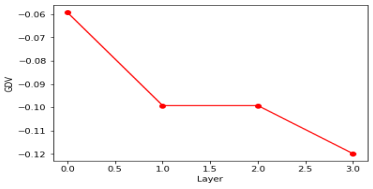
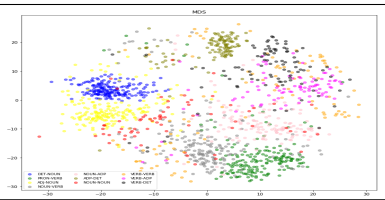
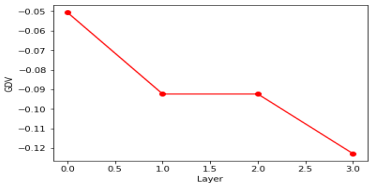
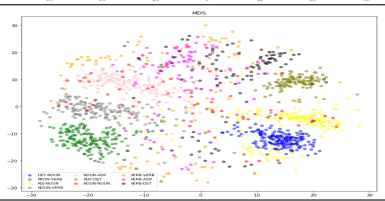
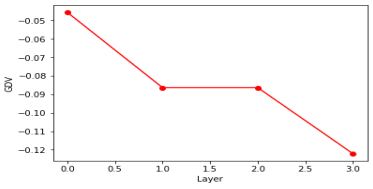
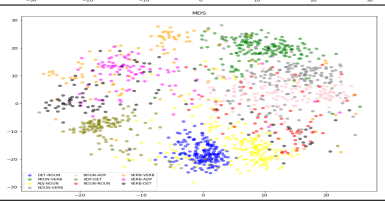
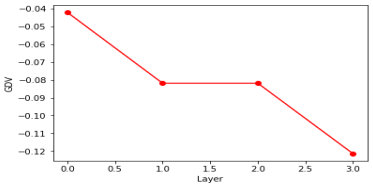
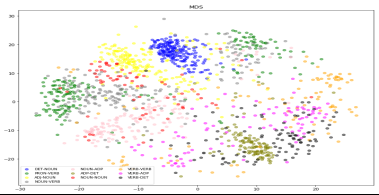
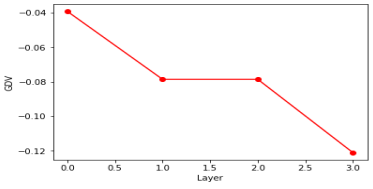
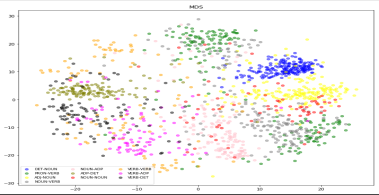
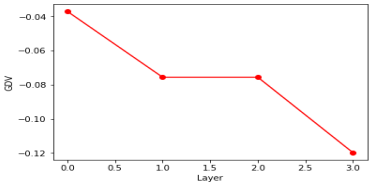
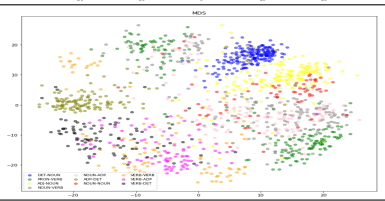
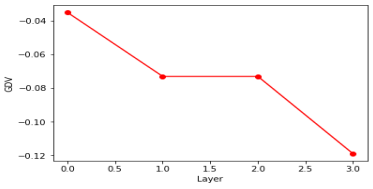
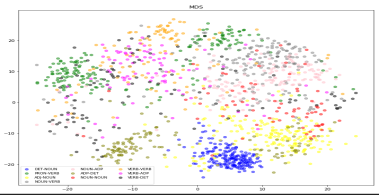
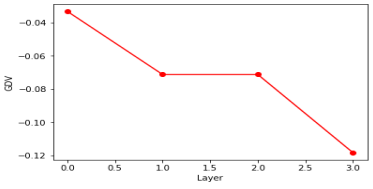
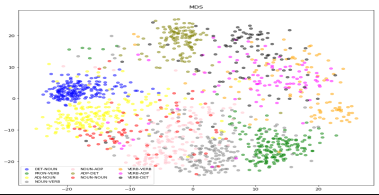
Input	GDV plot	Last layer MDS plot	Final GDV
[0,0,0,0,0,0,0,0,0,w1]			-0.11991
[0,0,0,0,0,0,0,0,w1,w2]			-0.12283
[0,0,0,0,0,0,0,w1,w2,w3]			-0.12213
[0,0,0,0,0,0,w1,w2,w3,w4]			-0.12143
[0,0,0,0,0,w1,w2,w3,w4,w5]			-0.12112
[0,0,0,0,w1,w2,w3,w4,w5,w6]			-0.11982
[0,0,0,w1,w2,w3,w4,w5,w6,w7]			-0.11894
[0,0,w1,w2,w3,w4,w5,w6,w7,w8]			-0.1183

Table 4.17: 2 words text prediction plots for different inputs (English)

4.2.6 Collocations vs Non Collocations

In this section, we test whether collocations and non collocations are represented in different parts of space by the neural network, similar to section 4.1.7. Both the bigram and trigram models are tested, i.e, we plot the MDS for 2 word and 3 word collocations. Here, we only consider the 2 word or bigram case for the English novel.

The 2 dimensional bigram matrix is of dimensions (5946,5946) is built, where 5946 refers to the number of unique words in the entire English corpus. The sum of all the entries in the bigram matrix will be equal to the total number of words in the corpus, i.e, 47470. Figure 4.21 shows the number of occurrences of a bigram plotted against the natural log frequency of occurrences to understand the distribution. As expected, more bigrams have few occurrences.

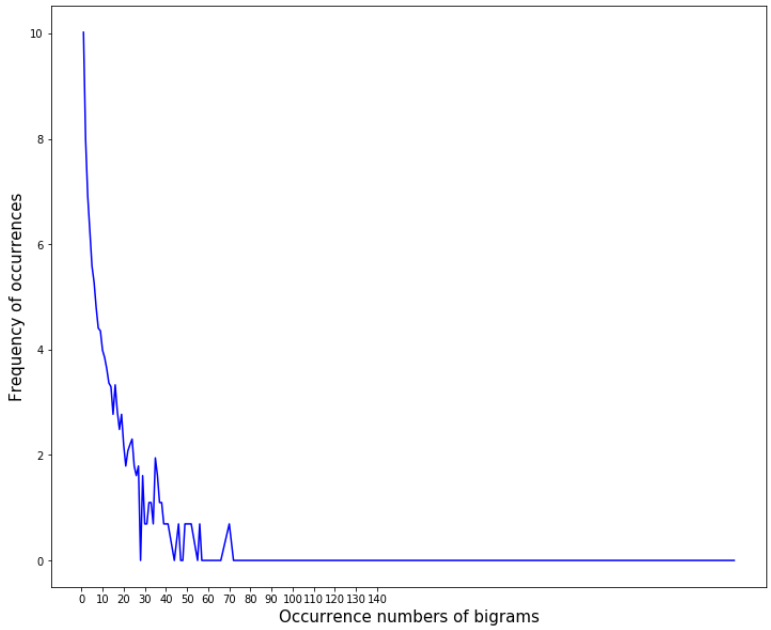


Figure 4.21: Occurrences of bigrams vs Log Frequency (English)

Since collocations lie in between very frequently occurring bigrams and sparsely occurring bigrams, it makes sense to consider a threshold between 5 and 25 occurrences. Experimentally, a threshold of 7 led to the best class separations for this corpus. Therefore, the bigrams which occur less than 7 times are considered as non collocations, and the rest are grouped as collocations.

The traditional single word text prediction neural network where nine embeddings are passed into the input is used here. However, instead of 13 class labels corresponding to the

POS tags, we now use 2 class labels : One for collocations (visualized in orange) and the other for non collocations (visualized in blue). Table 4.18 illustrates the MDS plots from all six layers of the neural network. We observe some level of class separation between collocations and non collocations, though not at the same degree as the POS tags case. The majority of collocation pairs are grouped at the bottom right part of the last layer MDS plot. But a lot of collocation data points are also spread across the plot uniformly. The GDV plot in Figure 4.22 shows an overall decline, except for the small hiccup at the fourth layer.

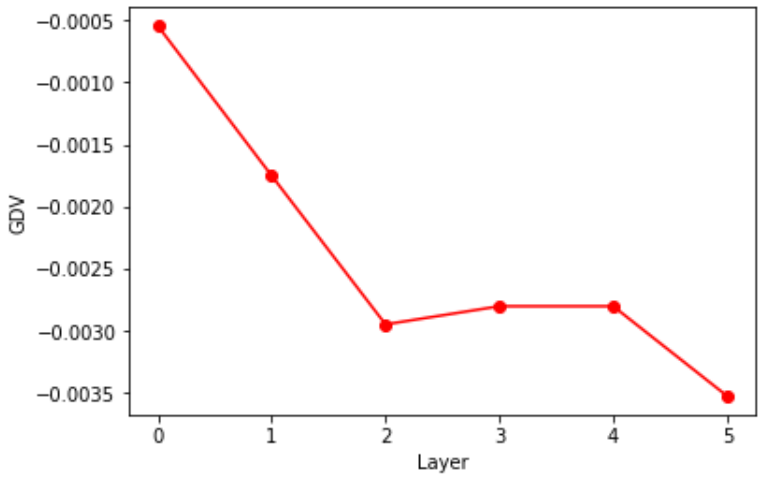
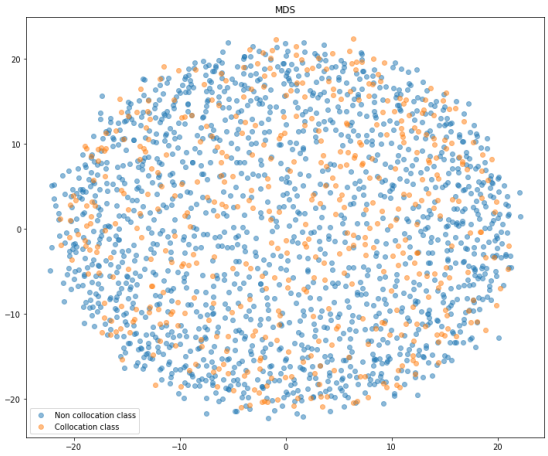
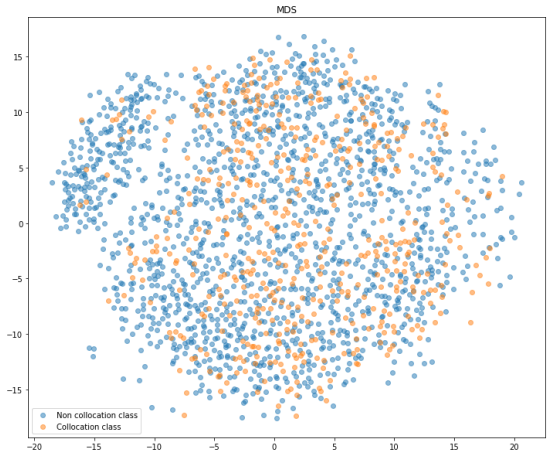


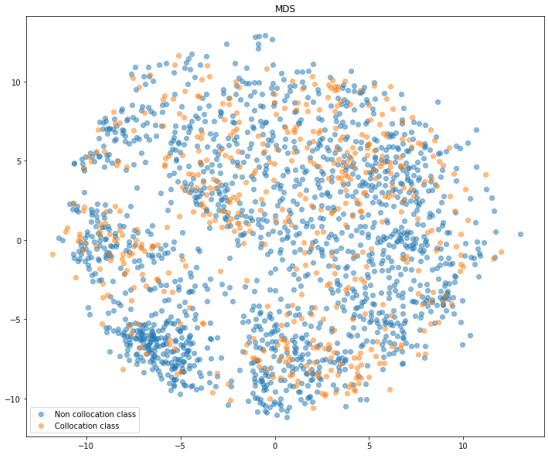
Figure 4.22: GDV plot for Collocations vs Non Collocations task: English corpus



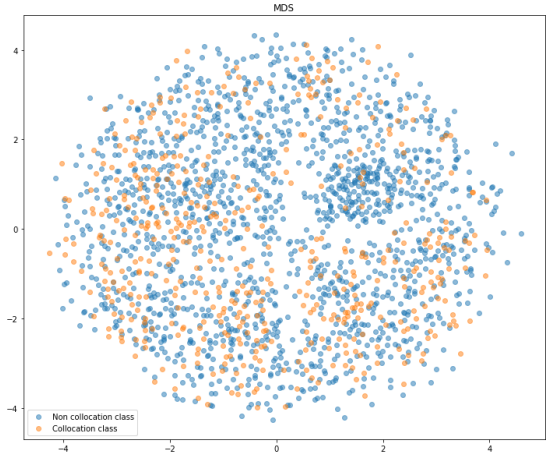
(a) Layer 1 MDS plot



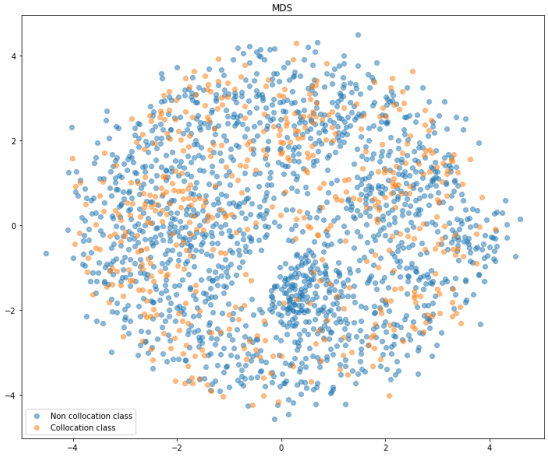
(b) Layer 2 MDS plot



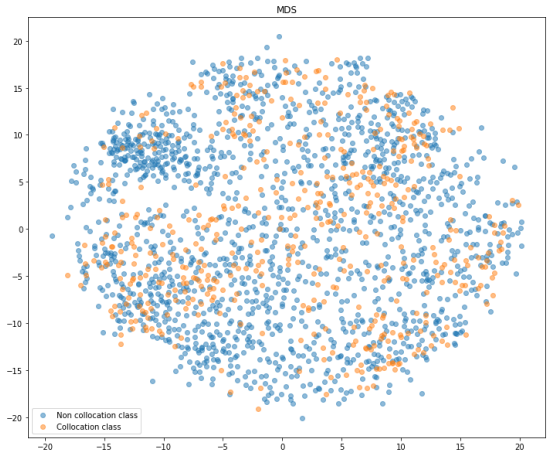
(c) Layer 3 MDS plot



(d) Layer 4 MDS plot



(e) Layer 5 MDS plot



(f) Layer 6 MDS plot

Table 4.18: MDS plots for Collocations vs Non Collocations task: English corpus

4.2.7 Collocations vs Non Collocations with different Input configurations

In the previous section, the single word text prediction model was used to visualize collocations and non collocations. In this section, we analyze the impact of passing different inputs from 1 word up to 8 words as the input, on the MDS and GDV plots.This is possible by making the preceding embeddings zeroes. The inputs are prepared in the same way as mentioned in section 4.2.1. Table 4.19 illustrates the last layer MDS plot and overall GDV plot for each type of input provided. Since the GDV plots show a general decreasing trend, only the final layer GDVs are considered for further analysis.Figure 4.23 illustrates the plot between the number of words (or non zero embeddings) in the 9 word input vs the final layer GDV. Unlike the previous results, the GDV keeps increasing and decreasing with different number of words at the input, leading to no definite pattern. The least GDV or highest class separation is observed for 2 words and 4 words at the input respectively.

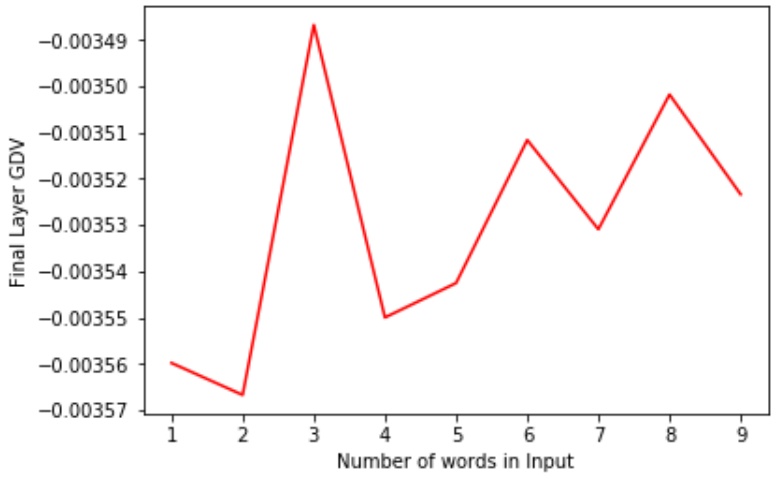


Figure 4.23: Number of words in the input vs Final Layer GDV curve

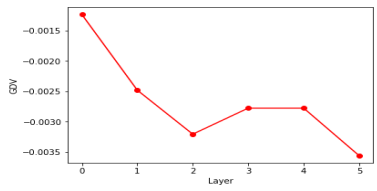
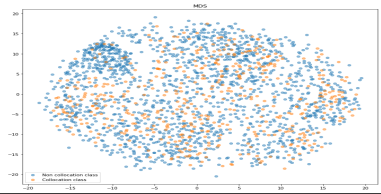
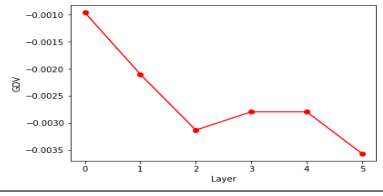
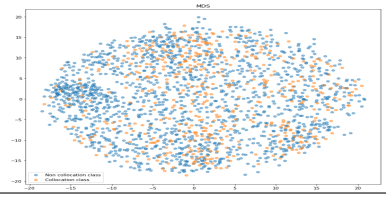
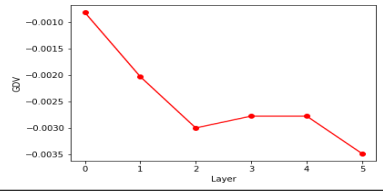
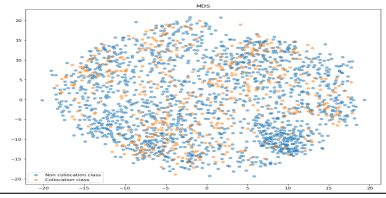
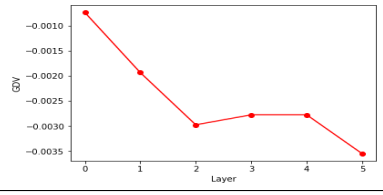
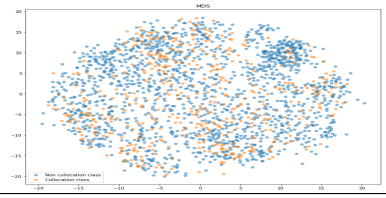
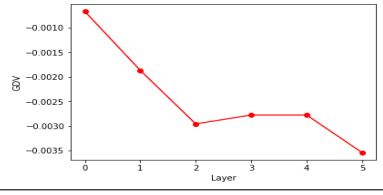
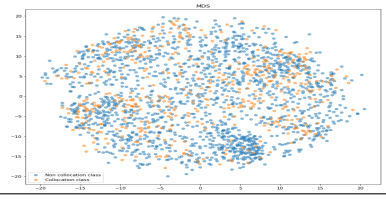
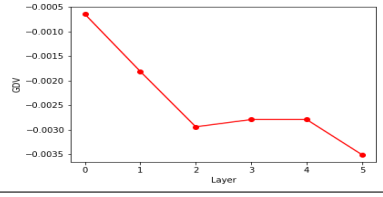
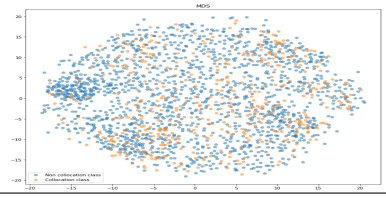
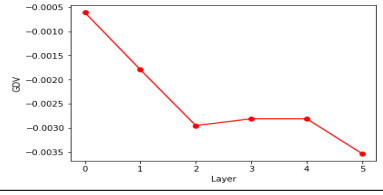
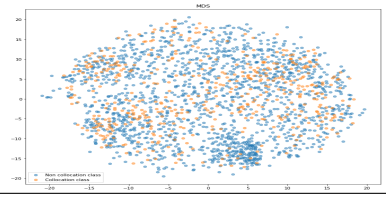
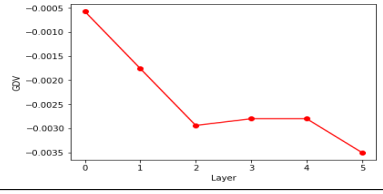
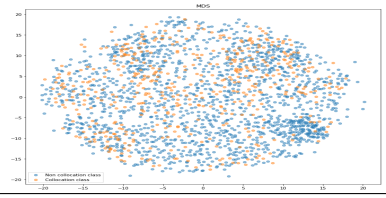
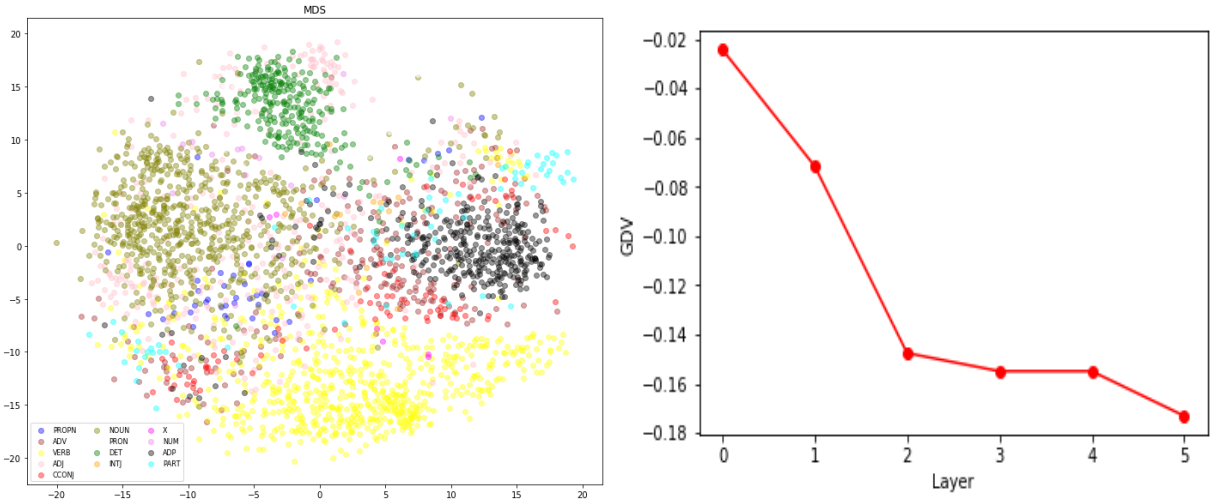
Input	GDV plot	Last layer MDS plot	Final GDV
[0,0,0,0,0,0,0,0,w1]			-0.00356
[0,0,0,0,0,0,0,w1,w2]			-0.00357
[0,0,0,0,0,0,w1,w2,w3]			-0.003487
[0,0,0,0,0,w1,w2,w3,w4]			-0.00355
[0,0,0,0,w1,w2,w3,w4,w5]			-0.003542
[0,0,0,w1,w2,w3,w4,w5,w6]			-0.00351
[0,0,w1,w2,w3,w4,w5,w6,w7]			-0.00353
[0,w1,w2,w3,w4,w5,w6,w7,w8]			-0.0035

Table 4.19: Text prediction plots with different input configurations for Collocations vs Non Collocations (English)

4.2.8 Testing existing model on new corpus

So far, 80% of the corpus data was used as training data to train the model and the remaining 20% of the same corpus was used as the test set upon which predictions were made. Now we use the model in section 3.4.1 which was trained on the 'Hitchhiker's guide to the Galaxy' English corpus. But the test set is from the popular novel 'Harry Potter and the Philosopher's Stone' [Row15]. This was done to verify if the model trained on one English corpus can produce similar results in another corpus.



(a) Final layer MDS plot

(b) GDV plot

Figure 4.24: Layer 6 MDS plot (a) and GDV plot (b) for the Text Prediction model implemented on the new 'Harry Potter' corpus

The plots shown in Figure 4.24 are very similar to the previously obtained ones. The final layer MDS shows a good separation between POS tags, further validating that they are good candidates for word classes. The drop in the GDV curve implies increased class separation with layers of the neural network. These results can be extrapolated to other input configurations for the same model, as well as the word pair prediction task. Therefore, the model did not overfit on a single corpus.

Chapter 5

Conclusion and Outlook

Based on the various plots and tables obtained in the results, the following conclusions can be made:

1. **POS tags are the word classes for any text generation or prediction model of any language**: Even though the language source of the German corpus(play) and the English corpus(novel) were different, the end results obtained in sections 4.1.1, 4.1.2 and 4.2.1 in the form of MDS and GDV plots indicate good class separations between each word class or POS tag. These results can therefore be extrapolated to other languages too.

Future work can be done to implement this model in other languages to observe if similar separation can be observed for word classes.

2. **Results are analogous to the human brain** : Similar to how action related words and vision related words are stored in separate areas of the brain, verbs and nouns are also well separated as shown by the MDS plots. In addition, such separations were achieved for other word classes such as adjectives, pronouns, adverbs, etc.

3. **POS tag sequences can be considered as word classes** : The results obtained for single POS tags can be extended to sequences of POS tags such as bigrams, trigrams, and so on. This is evident from the MDS and GDV plots in sections 4.1.5 and 4.2.4.

4. **The text prediction embeddings model is the closest to the human brain** : The text prediction model was preferred over the text reproduction model due to an ideal falling GDV curve with the layers of the neural network. Among the text prediction models, the embeddings model of section 3.4.1 is the closest to the human brain due to the better class separation and higher test accuracy than the embeddings with character matrix model.

This further proves the idea validated in [Sta17] that the neural network performing text prediction of the next word/sequence of words is a special case of the successor representation, and the idea in [Cla13] that the brain is a prediction machine.

5. **Collocations and non collocations cannot be considered as word classes** : While the MDS plots in section 4.1.7 indicate some kind of word class formation, many data points of either class are present in the other class's space. Uneven number of data points for each class (higher number of non collocations than collocations) also make it difficult to assess class separation. In case of the English corpus in section 4.2.6, there are clearly no visible classes.

Corpus Linguistics has a more sophisticated definition for collocations. The rough threshold based selection of collocations and non collocations may not have worked well. Also, the neural networks built may be too small to capture such fine details. More work on better ways to select collocations can be done.

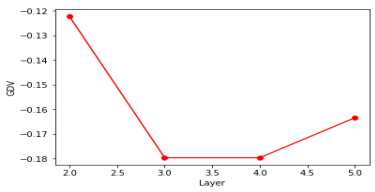
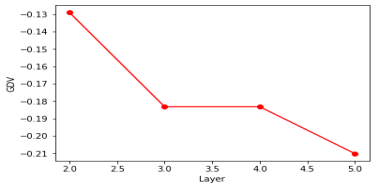
6. For most text prediction cases, the final layer GDV increases with the number of words in the 9-word input. In other words, lesser words at the input side lead to better class separations at the output, as evident from the final layer GDV plots in sections 4.1.4, 4.1.6 and 4.2.5 . In case of collocations vs non collocations, the final layer GDV plots in sections 4.1.8 and 4.2.7 increase and decrease irregularly, leading to no meaningful conclusion.

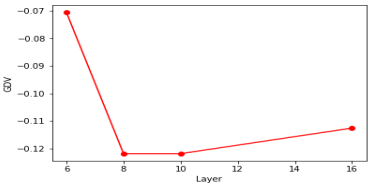
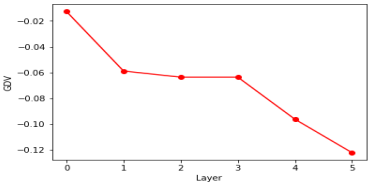
This contra-intuitive finding that less input leads to better class separation must be further investigated.

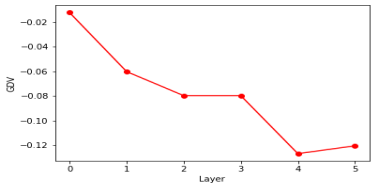
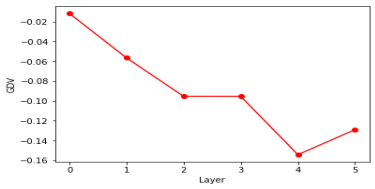
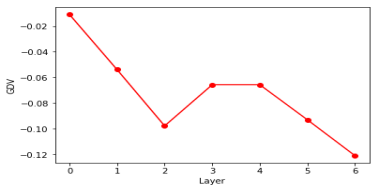
Appendix A

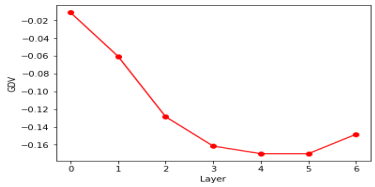
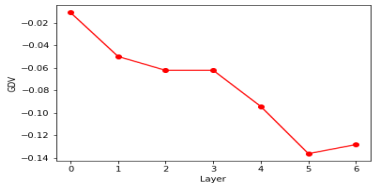
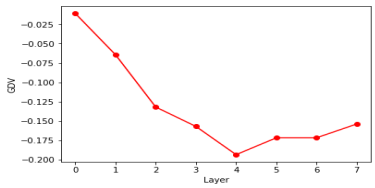
All Model results

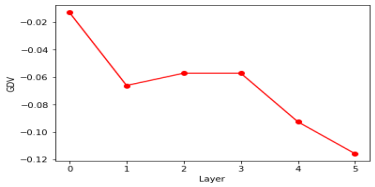
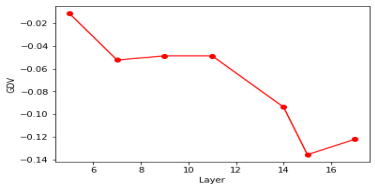
Table A.1 illustrates the Network Architecture and GDV plots of all models which were built for two primary tasks of text reproduction and text prediction and were not considered for further analysis due to better performance of the selected models.

Model	Task	Network Architecture	GDV Plot
Embeddings Model	Text Reproduction	Number of layers : 4 Word embeddings input → LSTM(64) → LSTM(128) → Flatten → Dense(384) → Word embeddings output	
Embeddings with concatenated POS tags Model	Text Reproduction	Number of layers : 4 Word embeddings input → LSTM(64) → LSTM(128) → Flatten → Dense(384) → Word embeddings output	

Embeddings with concatenated character matrices Model	Text Reproduction	Submodel 1 : Character Matrix input → Conv2D (2,2) → LSTM(4) → LSTM(4) → Flatten → Submodel 1 output Submodel 2: Word embeddings input → Bi-LSTM(64) → Bi-LSTM(128) → Flatten → Submodel 2 output Submodel 3: Concat(Submodel 1 output, Submodel 2 output) → Dense(384) → Submodel 3 output Submodel 4: Submodel 3 output → Dense(384) → Word embeddings output Submodel 5: Submodel 3 output → Dense(930) → Character matrix output	
Embeddings Model	Text Prediction	Number of layers : 6 Word embeddings input → Bi-LSTM(128) → Bi-LSTM(128) → Bi-LSTM(64) → Flatten → Dense(768) → Dense(384) → Word embeddings output	

Embeddings Model	Text Prediction	Number of layers : 6 Word embeddings input → Bi-LSTM(128) → Bi-LSTM(128) → Bi-LSTM(64) → Flatten → Dense(384) → Dense(384) → Word embeddings output	
Embeddings Model	Text Prediction	Number of layers : 6 Word embeddings input → Bi-LSTM(128) → Bi-LSTM(128) → Bi-LSTM(64) → Flatten → Dense(192) → Dense(384) → Word embeddings output	
Embeddings Model	Text Prediction	Number of layers : 7 Word embeddings input → Bi-LSTM(128) → Bi-LSTM(128) → Bi-LSTM(64) → Bi-LSTM(64) → Flatten → Dense(768) → Dense(384) → Word embeddings output	

Embeddings Model	Text Prediction	Number of layers : 7 Word embeddings input → Bi-LSTM(128) → Bi-LSTM(128) → Bi-LSTM(64) → Bi-LSTM(64) → Bi-LSTM(32) → Flatten → Dense(384) → Word embeddings output	
Embeddings Model	Text Prediction	Number of layers : 7 Word embeddings input → Bi-LSTM(128) → Bi-LSTM(128) → Bi-LSTM(64) → Flatten → Dense(384) → Dense(384) → Dense(384) → Word embeddings output	
Embeddings Model	Text Prediction	Number of layers : 8 Word embeddings input → Bi-LSTM(128) → Bi-LSTM(128) → Bi-LSTM(64) → Bi-LSTM(64) → Bi-LSTM(32) → Bi-LSTM(32) → Flatten → Dense(384) → Word embeddings output	

Embeddings concatenated with POS tags Model	Text Prediction	Number of layers : 6 Word embeddings input → Bi-LSTM(128) → Bi-LSTM(128) → Bi-LSTM(64) → Flatten → Dense(796) → Dense(398) → Word embeddings output	
Embeddings with concatenated character matrices Model	Text Prediction	Submodel 1 : Character Matrix input → Conv2D (2,2) → LSTM(4) → LSTM(4) → Flatten → Submodel 1 output Submodel 2: Word embeddings input → Bi-LSTM(128) → Bi-LSTM(128) → Bi-LSTM(64) → Flatten → Submodel 2 output Submodel 3: Concat(Submodel 1 output, Submodel 2 output) → Dense(384) → Submodel 3 output Submodel 4: Submodel 3 output → Dense(384) → Dense(384) → Word embeddings output Submodel 5: Submodel 3 output → Dense(930) → Character matrix output	

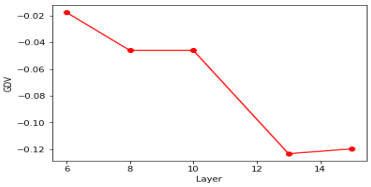
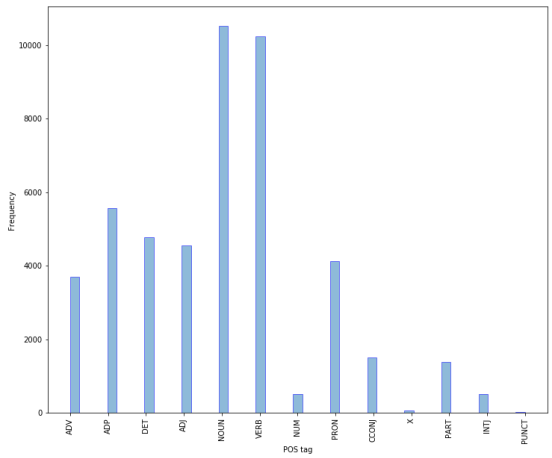
Embeddings with concatenated character matrices Model	Text Prediction	Submodel 1 : Character Matrix input → Conv2D (2,2) → LSTM(4) → LSTM(4) → Flatten → Submodel 1 output Submodel 2: Word embeddings input → Bi-LSTM(128) → Bi-LSTM(64) → Flatten → Submodel 2 output Submodel 3: Concat(Submodel 1 output, Submodel 2 output) → Dense(384) → Submodel 3 output Submodel 4: Submodel 3 output → Dense(384) → Dense(384) → Word embeddings output Submodel 5: Submodel 3 output → Dense(930) → Character matrix output	

Table A.1: Other models and their GDV plots

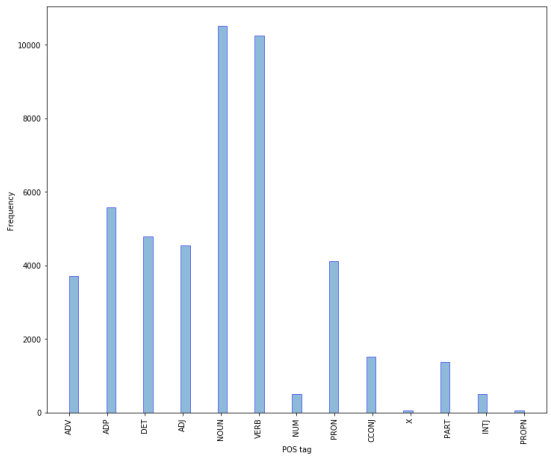
Appendix B

Bar Graphs

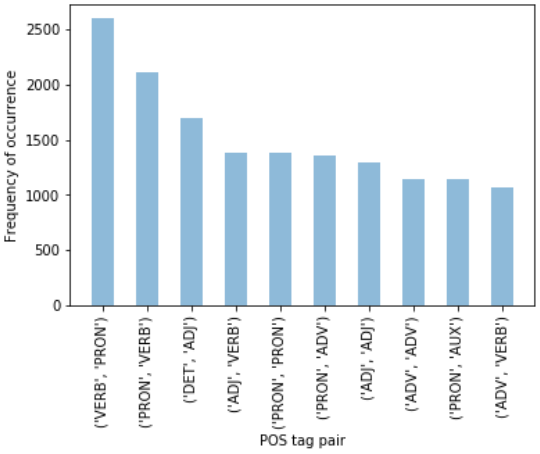
Table B.1 illustrates how frequent each POS tag (pair) is in the German and English corpora.



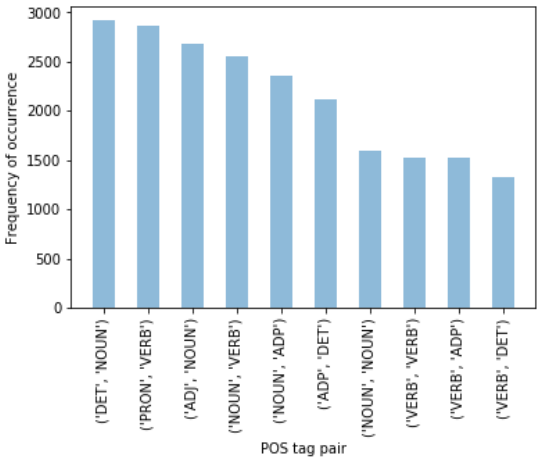
(a) Single POS distribution (German)



(b) Single POS distribution (English)



(c) POS pair distribution (German)



(d) POS pair distribution (English)

Table B.1: POS distributions in German and English corpora

Appendix C

Collocations vs Non Collocations for Trigram Model

So far, the bigram model was considered to evaluate collocations vs non collocations in both the English and German corpora. Here, we consider 3 word collocations or the Trigram model for the German corpus.

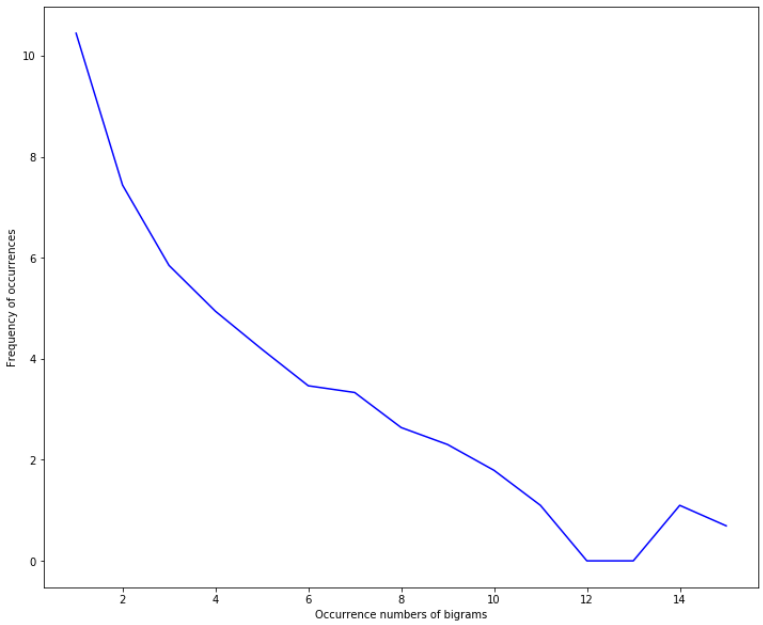


Figure C.1: Occurrence of trigrams vs Log Frequency

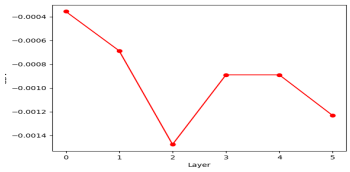
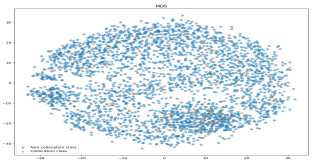
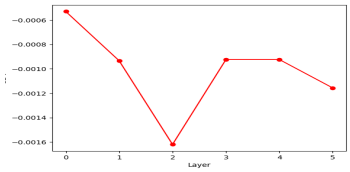
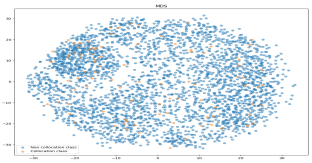
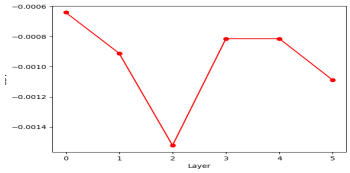
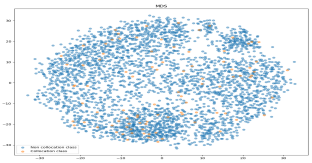
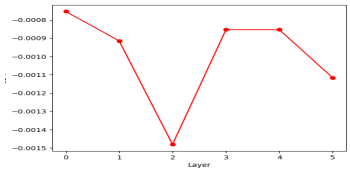
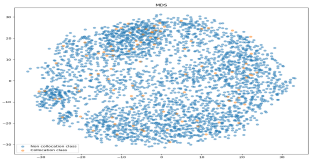
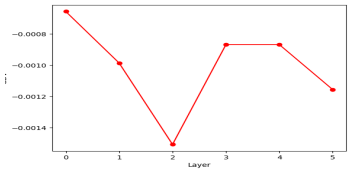
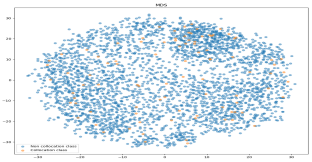
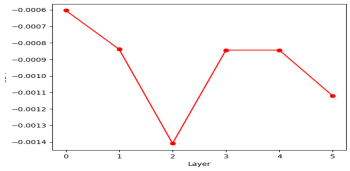
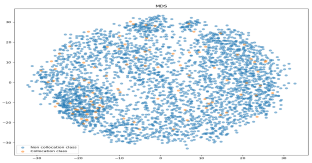
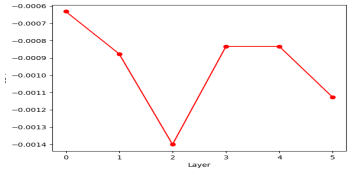
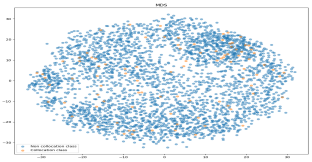
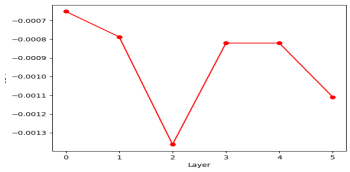
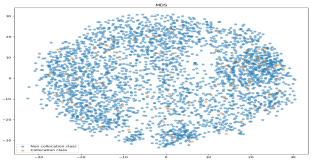
Input	GDV plot	Last layer MDS plot	Final GDV
[0,0,0,0,0,0,0,0,w1]			-0.00123
[0,0,0,0,0,0,0,w1,w2]			-0.00116
[0,0,0,0,0,0,w1,w2,w3]			-0.00109
[0,0,0,0,0,w1,w2,w3,w4]			-0.00112
[0,0,0,0,w1,w2,w3,w4,w5]			-0.00115
[0,0,0,w1,w2,w3,w4,w5,w6]			-0.00112
[0,0,w1,w2,w3,w4,w5,w6,w7]			-0.00113
[0,w1,w2,w3,w4,w5,w6,w7,w8]			-0.00111

Table C.1: Text prediction plots with different input configurations for Collocations vs Non Collocations Trigram Model

Figure C.1 shows the log frequency of occurrences of trigrams. A good threshold which led to the best separation of classes was '5' occurrences. Therefore, any trigrams which occur 5 or more times were considered as collocations, and the remaining as non collocations.

The MDS plots in Table C.1 indicate that very few trigram collocations exist overall in the entire corpus (few orange points). Except between layers 3 and 4 of the neural network, the GDV trend in overall decreasing.

Figure C.2 indicates that there is no specific trend observed for the final layer GDV against different input configurations. The least GDV is observed for 1 word at the input and 5 words at the input respectively.

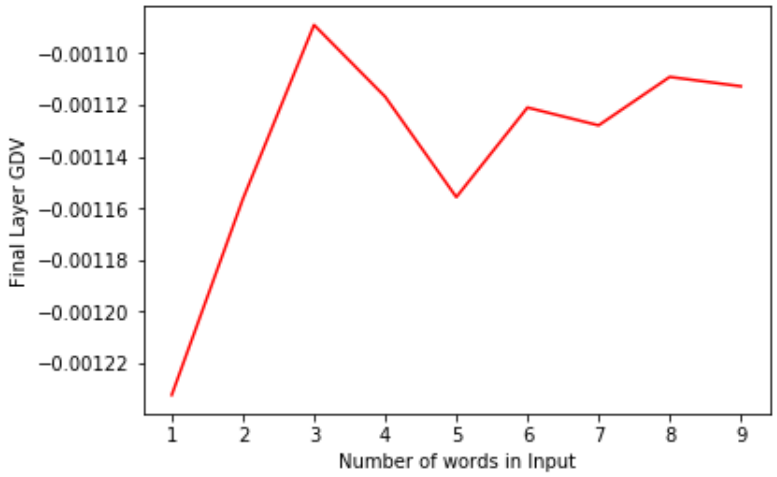


Figure C.2: Number of words in the input vs Final Layer GDV curve

List of Figures

List of Tables

Bibliography

[Abb16] David Abbott. What brain regions control our language?and how do we know this? *The Conversation*, September 2016.

[Ada17] Douglas Adams. *The Hitchhiker's Guide to the Galaxy Omnibus: A Trilogy in Five Parts*, volume 6. Pan Macmillan, 2017.

[Bad19] Will Badr. Auto-encoder: What is it? and what is it used for? (part 1). *Towards Data Science*, 2019.

[Ban] Arunava Bannerjee. The perceptron.

[Bis11] Walter Bisang. Word classes. *OXFORD HANDBOOKS IN*, page 281, 2011.

[Bro17] Jason Brownlee. What are word embeddings for text? *URL: https://machinelearningmastery. com/what-are-wordembeddings*, 2017.

[Buj08] Andreas Buja, Deborah F Swayne, Michael L Littman, Nathaniel Dean, Heike Hofmann, and Lisha Chen. Data visualization with multidimensional scaling. *Journal of computational and graphical statistics*, 17(2):444–472, 2008.

[Cla13] Andy Clark. Whatever next? predictive brains, situated agents, and the future of cognitive science. *Behavioral and brain sciences*, 36(3):181–204, 2013.

[Des17] Mohit Deshpande. All about autoencoders, 2017.

[Eas94] John Eastwood. *Oxford guide to English grammar*. Oxford University Press, 1994.

[Fei18] Hongxiao Fei and Fengyun Tan. Bidirectional grid long short-term memory (bigridlstm): A method to address context-sensitivity and vanishing gradient. *Algorithms*, 11(11):172, 2018.

[FV16] Philippe Fournier-Viger. An introduction to sequence prediction. *The Data Mining Blog*, 2016.

[Gei18] Adam Geitgey. Natural language processing is fun! *Medium*, 2018.

[Gér17] Aurélien Géron. *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems*. Ö'Reilly Media, Inc.", 2017.

[Gla06] Daniel Glattauer. *Gut gegen Nordwind: Roman*. Paul Zsolnay Verlag, 2006.

[Gol17] Yoav Goldberg. Neural network methods for natural language processing. *Synthesis Lectures on Human Language Technologies*, 10(1):92–92, 2017.

[Goo16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[Gro14] The Stanford Natural Language Processing Group. Neural network dependency parser, 2014.

[Har19] Erin Harte. How your brain processes language. *Brain World*, March 2019.

[Jat19] Sharmistha Jat, Hao Tang, Partha Talukdar, and Tom Mitchell. Relating simple sentence representations in deep neural networks and the brain. *arXiv preprint arXiv:1906.11861*, 2019.

[Kri18] Nikolaus Kriegeskorte and Pamela K Douglas. Cognitive computational neuroscience. *Nature neuroscience*, 21(9):1148–1160, 2018.

[Kru78] Joseph B Kruskal and Myron Wish. *Multidimensional scaling*, volume 11. Sage, 1978.

[McA05] Thomas Burns McArthur, Tom McArthur, and Roshan McArthur. *Concise Oxford companion to the English language*. Oxford University Press, USA, 2005.

[Mik13] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[Nic] Chris Nicholson. Deep autoencoders.

[Ola15] Christopher Olah. Understanding lstm networks. 2015.

[Pen14] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[Pul99] Friedemann Pulvermüller. Words in the brain's language. *Behavioral and brain sciences*, 22(2):253–279, 1999.

[Row15] Joanne K Rowling. *Harry Potter and the philosopher's stone*, volume 1. Bloomsbury Publishing, 2015.

[Sah18] Sumit Saha. A comprehensive guide to convolutional neural networks–the eli5 way, 2018.

[Sch97] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

[Sch18] Achim Schilling, Claus Metzner, Jonas Rietsch, Richard Gerum, Holger Schulze, and Patrick Krauss. How deep is deep enough?–quantifying class separability in the hidden layers of deep neural networks. *arXiv preprint arXiv:1811.01753*, 2018.

[Sch19] Dan Schwartz and Tom Mitchell. Understanding language-elicited eeg data by predicting it from a fine-tuned language model. *arXiv preprint arXiv:1904.01548*, 2019.

[Sta] Stanford. Multidimensional scaling.

[Sta17] Kimberly L Stachenfeld, Matthew M Botvinick, and Samuel J Gershman. The hippocampus as a predictive map. *Nature neuroscience*, 20(11):1643, 2017.

[Tom18] Rosario Tomasello, Max Garagnani, Thomas Wennekers, and Friedemann Pulvermuller. A neurobiologically constrained cortex model of semantic grounding with spiking neurons and brain-like connectivity. *Frontiers in computational neuroscience*, 12:88, 2018.

[vE99] Boas W van Emde. Juhn a. wada and the sodium amytal test in the first (and last?) 50 years. *Journal of the History of the Neurosciences*, 8(3):286, 1999.

[Yan16] Shi Yan. Understanding lstm and its diagrams. *Software engineer & wantrepreneur. Interested in computer graphics, bitcoin and deep learning*, 13(03), 2016.