Master of Science

Text2Scene: An interface for object-based processing of part-whole relations

Vincent Roy Kühn

Abgabedatum: 19. Februar 2021

Goethe-Universität Frankfurt am Main
Prof. Dr. A. Mehler
Prof. Dr. V. Ramesh

Bitte dieses Formular zusammen mit der Abschlussarbeit abgeben!

Erklärung zur Abschlussarbeit

gemäß § 34, Abs. 16 der Ordnung für den Masterstudiengang Informatik vom 17. Juni 2019

Hiermit erkläre ich Herr / Frau

_____

Die vorliegende Arbeit habe ich selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel verfasst.

Ebenso bestätige ich, dass diese Arbeit nicht, auch nicht auszugsweise, für eine andere Prüfung oder Studienleistung verwendet wurde.

Zudem versichere ich, dass die von mir abgegebenen schriftlichen gebundenen Versionen meiner Masterarbeit mit der auf dem Datenträger abgegebenen elektronischen Version übereinstimmen.

Frankfurt am Main, den

_____

Unterschrift der / des Studierenden

Zusammenfassung

Die folgende Arbeit handelt von einer Text2Scene Anwendung, welche in der *Virtual Reality* (VR) umgesetzt wurde. Das System ermöglicht es den Usern aus einer Beschreibung einer Szene, diese virtuell nachzustellen. Dies bietet eine neue Art der Interaktion mit einem Text, die die visuelle Komponente hervorhebt und somit eine Geschichte auf neue Wege erfahrbar macht.

Dazu kann der User einen fertigen Text entweder vom Server zu laden oder einen eigenen erstellen, der dann automatisch verarbeitet wird. Dabei werden die vorhanden physischen Objekte im Text automatisch erkannt und dem User als 3D-Objekte in der virtuellen Umgebung zur Verfügung gestellt. Diese können dann manuell platziert werden und erzeugen dadurch die Szene, die im Ausgangstext beschrieben wurde. Das Ziel der Textverarbeitung ist eine möglichst genaue Beschreibung der Objekte, damit diese zielgerichtet in der Objektdatenbank gesucht werden können.

Bei der Textverarbeitung wird besonderer Wert auf das Erkennen von Teil-Ganz Beziehungen gelegt. Sodass Objekte, die im Text vorkommen und ein Holonym besitzen, automatisch mit diesem verknüpft werden. Gleichzeitig wird die Teil-Ganz Beziehung aber auch in die andere Richtung genauer betrachtet. Die Textverarbeitung soll ferner dazu in der Lage sein, Objekte genauer zu spezifizieren und an den Kontext des Textes anzupassen. Weiterhin wurde das *Natural Language Processing* (NLP) so ausgebaut, dass der Kontext des Textes erkannt wird und die Objekte entsprechend kategorisiert werden. Die Textverarbeitung wird mithilfe eines Neuronalen Netzes implementiert. Die verwendeten Tools zur Erkennung von Teil-Ganz Beziehungen, Kontext und Spezifikation von Objekten wurden anhand von Texteingaben nach der Genauigkeit der Ausgabe evaluiert.

Zur Nutzung der Textverarbeitung wurde eine virtuelle Szene entwickelt, die das Erstellen von eigenen Szenen aus vorher geladenen beziehungsweise eingegebenen Texten ermöglicht. Dazu kann der Nutzer manuell oder automatisch Objekte laden lassen, die er dann platzieren kann.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

GLUE ..... Genereal Language Understanding Evaluation

GUI ..... Graphical User Interface

NER ..... Named Entity Recognition

POS ..... Part of Speech

SQuAD ..... Stanford Question Answering Datasets

VR ..... Virtual Reality

1. Einleitung

1.1. Aufgabenstellung

Das Ziel dieser Arbeit ist ein System, das einen Text oder eine Beschreibung eines Raumes einliest und mittels Textverarbeitung die enthaltenen Objekte erkennt und die Informationen extrahiert Die 3D-Objekte werden dann dem User zur Verfügung gestellt, damit er sie im virtuellen Raum platzieren kann. Das Hauptaugenmerk bei der Informationsextraktion liegt dabei in der Erkennung des Kontexts und der Teil-Ganz Beziehungen.

1.2. Motivation

Die virtuelle Realität bietet heutzutage vielfältige Möglichkeiten und verändert die User Experience deutlich. In vielen Anwendungen, wie zum Beispiel Videospielen, wird der User noch stärker in die Anwendung beziehungsweise die Szene eingebunden. Die Idee liegt also nahe, dem User weitere Felder, die bisher noch nicht in der VR verfügbar sind, erfahrbar zu machen. Dazu gehört unter anderem *Text-to-Scene* oder auch Text2Scene genannt. Text2Scene ermöglicht es einen Text einzulesen und dann aus dieser Beschreibung eine Szene beziehungsweise einen Raum nachzubilden. Dadurch können zum Beispiel Szenen aus Büchern visualisiert werden und der User erhält eine genaue Vorstellung dessen, was der Autor in seinem Buch beschreiben möchte.

Bisher gibt es verschiedene Applikationen, die Text2Scene umsetzen, zum Beispiel *Wordseye* von Coyne und Sproat (2001). Allerdings beschränken sie sich auf eine Ausführung an einem Bildschirm und damit auf 2D. In dieser Arbeit wird versucht dies zu ändern, indem eine grundlegende Plattform für eine Text2Scene Anwendung in der VR geschaffen wird, die danach weiterentwickelt werden kann.

Damit der erzeugte Raum der Beschreibung ähnelt, muss die Anwendung eine gute Textverarbeitung besitzen. Dazu gehört als erstes das Erfassen von Objekten. Abgesehen davon, wird in dieser Arbeit die Erkennung des Kontexts implementiert. Sie ist wichtig, da Computersysteme nicht automatisch herausfinden können, in welchem Bezug die einzelnen Wörter

zueinanderstehen und ob zum Beispiel mit *Bank* die Parkbank oder die Bankfiliale gemeint ist. Ein Mensch kann das Gesagte automatisch in den richtigen Kontext setzen, aber ein Computer muss die verschiedenen Bedeutungen des gleichen Worts erst erlernen.
Aber nicht nur der Kontext lässt sich aus dem Text extrahieren, sondern auch weitere Einzelheiten wie Positionsangaben und zeitliche Bezüge. In dieser Arbeit wird jedoch der Fokus auf Teil-Ganz Relationen gesetzt.
Wenn beispielsweise in einem Text von einer *Lehne* die Rede ist, soll der Computer erkennen, dass die *Lehne* kein eigenständiges Objekt ist, sondern zu einem übergeordneten Gegenstand gehört, dem Holonym, welches in diesem Fall der Stuhl wäre. Sowohl für die Erkennung des Kontexts als auch für die Extraktion von Teil-Ganz Beziehungen werden Neuronale Netze genutzt, die durch die Tools BERT (Wolf u. a. 2020) und WordNet (Miller 1998) unterstützt werden. Beide Anwendungen sollen grundlegende Informationen erkennen, die dann durch die Neuronalen Netze weiterverarbeitet werden können.

2. Grundlagen

2.1. Textverarbeitung

Ein zentraler Bestandteil dieser Arbeit ist das computerbasierte Extrahieren von Information aus einem Text. Dabei liegt der Fokus einerseits auf der Erkennung von Teil-Ganz Beziehungen, die auch Meronyme (Teile) beziehungsweise Holonyme (Ganzes) genannt werden, je nachdem welche Richtung der Relation betrachtet wird. Andererseits soll der Kontext des Texts bestimmt werden.
Im entwickelten System bezieht sich die Bestimmung der Holonyme auf alle physischen Objekte außer Personen. Ein Beispiel für eine solche Beziehung ist die Relation *backrest* $\rightarrow$ *chair*. Immaterielle Beziehungen, wie *paragraph* $\rightarrow$ *text*, werden dagegen ignoriert und dementsprechend nicht erkannt, da sich diese Arbeit auf die Erstellung einer Szene mithilfe von 3D-Objekten konzentriert.
Gleichzeitig werden aber auch noch weitere lexikalische Relationen in dieser Arbeit verwendet wie Hypernyme. Während Holonyme/Meronyme eine „*part-of*“ Beziehung beschreiben, werden Hypernyme durch eine „*is-a*“ Beziehung dargestellt, welche eine übergeordnete Klasse beschreiben (Miller 1998, vgl.). Beispielsweise ist *furniture* ein Hypernym von *chair*, da der Stuhl ein Möbelstück ist.
Diese lexikalischen Beziehungen sind auch dahingehend für diese Arbeit relevant, weil sie ein zentraler Punkt für die Erkennung von Objekten sind.

2.2. Neuronale Netze

2.2.1. Übersicht zu Neuronalen Netzen

Neuronale Netze bilden einen Teilbereich des Machine Learnings. Es finden sich immer mehr Anwendungsfälle, bei denen sie hilfreich sind, um Probleme zu lösen. Die Klasse der Neuronalen Netze nutzt das menschliche Nervensystem, insbesondere jedoch das Gehirn mit seinen Neuronen, als Vorlage. Dabei gibt es zwei Arten von *Neural Networks*.
Die Erste beschäftigt sich mehr mit der Arbeitsweise des Gehirns und soll die Struktur möglichst genau nachbilden. Sie dienen dazu, das menschliche Gehirn besser zu verstehen und

sind eine Simulation des Gehirns.
Auf der anderen Seite gibt es Neuronale Netze, die nicht zum Verstehen der Arbeitsweise des Gehirns gedacht sind, sondern für eine effiziente Problemlösung optimiert sind (Rey und Wender 2018, vgl.). Es gibt verschieden Typen von Netzen, die sich in drei Hauptgruppen einteilen lassen (Liu u. a. 2015; Medsker und Jain 2001, vgl.):

- *Feed-Forward Networks*
- *Recurrent Neural Networks*
- *Convolution Neural Networks*

Da in dieser Arbeit die klassischen *Feed-Forward Networks* genutzt werden, wird bei der Arbeitsweise und der Beschreibung besonders der Fokus auf diese Netze gelegt.

2.2.2. Aufbau Neuronaler Netze

Ähnlich wie beim biologischen Vorbild besteht ein Neuronales Netz aus verschiedenen Neuronen, die miteinander verknüpft sind und dadurch Daten übertragen. Dabei lassen sich die Neuronen in verschiedene Schichten einteilen. Die erste Schicht ist immer der Inputlayer. Es erhält die Daten von außerhalb und gibt ihre Verarbeitung an die Neuronen des nächsten Layers weiter. Im einfachsten Neuronalen Netz, auch *Perceptron* genannt, wäre dies ein Outputlayer mit einem Neuron (siehe Abbildung 2.2) (Nandy 2018, vgl., Kapitel 1).

$$y(x, w) = f(\sum_{i=1}^{M} \omega_i \phi_i(x))$$

Abbildung 2.1.: Berechnung des Outputs eines Neurons y basierend auf dem Input x und des Gewichts w der Kante (x, y)

Die Daten, die ein Neuron berechnet, hängen von verschiedenen Inputfaktoren ab (siehe Abbildung 2.1). Es erhält den Output der 1 bis M Neuronen aus der vorherigen Schicht. Die Daten beziehungsweise die Kanten zu den vorherigen Neuronen sind gewichtet. Diese werden während des Trainings aktualisiert und dienen dazu, dass nützliche Verbindungen, die die Lösung des Problems begünstigen, höher gewertet werden als Kanten, die überwiegend schlechte Daten liefern. Die Summe der gewichteten Dateneingänge wird dann an eine

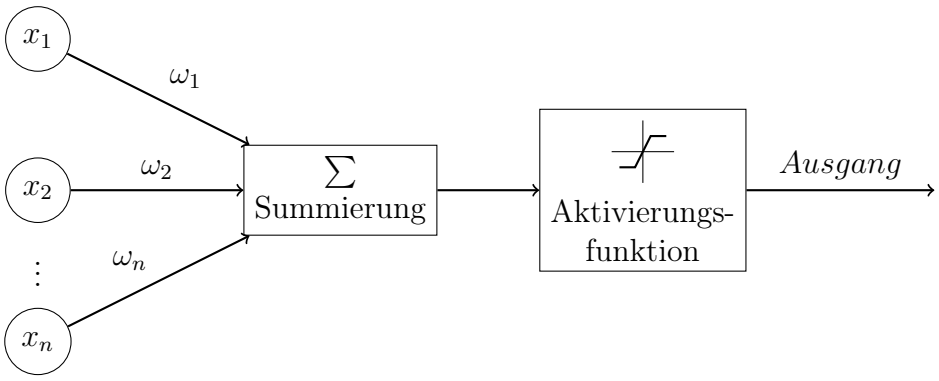

Abbildung 2.2.: Aufbau eines Neurons (Bishop 2006, vgl, eigene Darstellung)

interne Aktivierungsfunktion f übergeben. Sie ist dazu da, die Ausgabe zu generieren und unpassende Eingaben rauszufiltern (Bishop 2006; Nandy 2018, vgl.). Typische Aktivierungsfunktionen sind (Nandy 2018, vgl.):

- sigmoid
- tanh
- ReLu

In dieser Arbeit wird mit der ReLu Aktivierungsfunktion gearbeitet, welche nur positive Werte weiterleitet und negative Eingaben auf 0 setzt ($f = \max(0, x)$). Den Vorteil, den sie gegenüber den anderen Funktionen bietet, ist die Tatsache, dass *Feed-Forward Networks* bessere Ergebnisse mit ReLu erzielen (Glorot, Bordes und Bengio 2011, vgl.), wodurch sie heutzutage weit verbreitet ist (Montúfar u. a. 2014, vgl.).

2.2.3. Feed-Forward Networks

Feed-Forward Networks sind eine Weiterentwicklung des einfachen *Perceptrons* (Unterabschnitt 2.2.2). Der Wert des Outputneurons berechnet sich nach demselben Prinzip wie beim *Perceptron* (Abbildung 2.1). Der Unterschied liegt darin, was trainiert wird. Beim *Feed-Forward Network* wird sowohl das Gewicht des Inputs ω_i als auch die Basisfunktion $\phi_j(x)$, mit dem Wert x des Inputneurons i trainiert (Bishop 2006, vgl.).
Das Netzwerk kann, anders als das *Perceptron*, komplexer aufgebaut werden und enthält neben dem Input- und Outputlayer noch weitere Ebenen dazwischen, die sogenannten Hiddenlayers. Ein Hiddenlayer enthält wiederum 1 bis n Neuronen, die eingehende Kanten von allen Neuronen in den vorherigen Schichten haben können. Die Ausgabe eines Neurons des

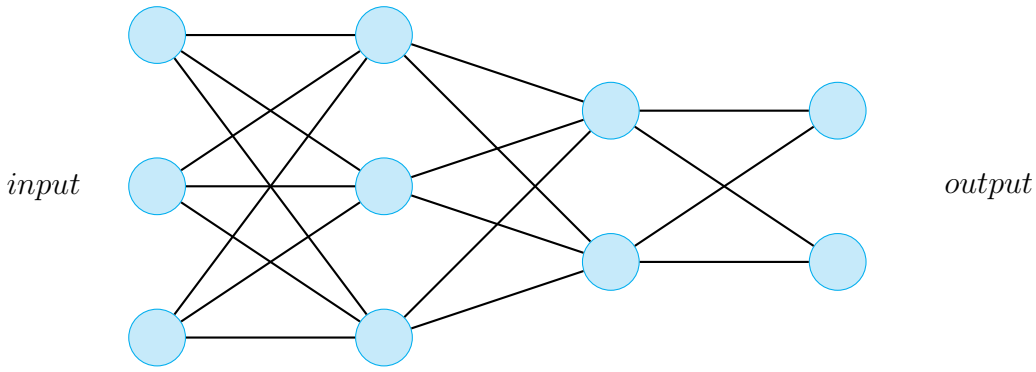

Abbildung 2.3.: Beispiel eines Feed-Forward Networks (Bishop 2006, vgl, eigene Darstellung)

Hiddenlayers ist wiederum die Eingabe für ein weiteres Hiddenlayer oder für das Outputlayer. Die Hiddenlayers müssen nicht zwangsläufig die gleiche Anzahl an Neuronen beinhalten. Sie können auch mehr oder weniger Neuronen als das Input- oder Outputlayer haben (siehe Abbildung 2.3) (Bishop 2006, vgl.).
Zu den Feed-Forward Netzen gehören alle Neuronalen Netze, die keine Kreise in ihrem Aufbau enthalten, sodass keine Schicht wieder Daten an eine vorherige Schicht weitergibt und Rückkopplung somit ausgeschlossen ist. Netze, die Kreise enthalten, gehören zu den *Recurrent Neural Networks* (Bishop 2006, vgl.). Die Eingaben bei einem einzelnen Neuron müssen aber nicht zwangsläufig aus der vorherigen Schicht kommen, da auch Schichten übersprungen werden dürfen oder das Neuron nicht zwangsläufig mit allen Neuronen der vorherigen Schicht verbunden sein muss (Bishop 2006, vgl.).

2.3. Trainieren von Neuronalen Netzen

2.3.1. Übersicht

Grundsätzlich gibt es drei verschiedene Methoden ein Neuronales Netz zu trainieren (Brachman und Dietterich 2009, vgl.):

- Supervised Learning
- Semisupervised Learning
- Unsupervised Learning

Diese unterscheiden sich in der Art, wie viele Daten vorliegen, an denen das Netz trainiert werden kann. Die Stufe, bei der dem Netz am meisten Daten zur Verfügung stehen, ist das

Supervised Learning. Dabei werden dem Neuronalen Netz neben den Inputdaten zusätzlich vordefinierte Outputs, auch Label genannt, mitgegeben. Diese dienen dazu, die Ausgabe des Netzes so anzupassen, dass nach dem Training der Output korrekt erkannt wird. Das Netz bekommt nach jedem Schritt eine direkte Rückmeldung, ob das ausgegebene Ergebnis korrekt war oder nicht (Brachman und Dietterich 2009, vgl.).
Die nächste Stufe ist das *Semisupervised Learning*. Es ist ähnlich wie das *Supervised Learning* aufgebaut. Man trainiert im ersten Schritt wieder mit einem gelabelten Datenset. Es kann aber vorkommen, dass die vorhandenen gelabelten Daten nicht sehr zahlreich sind oder es schwer ist Trainingsdaten für einen breiten Anwendungsfall zu bekommen. Dann kann man *Semisupervised Learning* einsetzen, indem man das trainierte Netz nutzt und den Output von einem größeren ungelabelten Datenset vorhersagen lässt. Nun können beide Datensätze kombiniert werden, um das Neuronale Netz weiter zu trainieren (Brachman und Dietterich 2009, vgl.).
Die letzte Form des Trainings ist das *Unsupervised Learning*, welches nur mit ungelabelten Daten arbeitet, weswegen es für die Eingabe kein vorher definiertes Ergebnis gibt. Diese Art des Lernens wird meistens für Clusteringprobleme verwendet (Brachman und Dietterich 2009, vgl.).
Damit die Ergebnisse, die die Neuronalen Netze ausgeben, nicht nur geraten sind, sondern abhängig von der Eingabe abhängen, muss man das Netz trainieren. Dabei wird versucht den Fehler zwischen einem definierten Label und dem berechneten Wert des Netzes zu verkleinern.

2.3.2. Backpropagation

Ein weit verbreiteter Algorithmus, der auch in dieser Arbeit genutzt wird, ist Backpropagation. Bei jeder Ausgabe eines Neuronalen Netzes wird als Erstes überprüft, ob das Ergebnis mit dem vordefinierten Label übereinstimmt. Wenn dies nicht der Fall ist, muss das Netzwerk angepasst werden, sodass es beim nächsten Mal besser wird (Werbos 1990, vgl.). Dafür wird der Einfluss der Gewichte auf das ausgegebene Ergebnis berechnet. Wenn der Fehler zu groß ist, müssen die Gewichte angepasst werden (Riedmiller und Braun 1993, vgl. S.1). Dazu werden die Ableitungen aller Gewichte w_{ij} ermittelt, wobei w_{ij} der Gewichtung der Kante zwischen den Neuronen i und j entspricht:

$$\frac{\delta E}{\delta w_{ij}} = \frac{\delta E}{\delta s_i} \frac{\delta s_i}{\delta net_i} \frac{\delta net_i}{\delta w_{ij}}$$

Sobald alle partiellen Ableitungen der Gewichte berechnet sind, kann der Fehler durch die Berechnung des Gradient Descents minimiert werden (Riedmiller und Braun 1993, vgl. S.1). Dieser iterative Algorithmus wird in jedem Schritt für das entsprechende Label ausgeführt, sodass die Gewichtungen der Verbindungen zwischen der Neuronen nach und nach besser werden. Dabei ist es wichtig, das Training zu beenden, sobald keine signifikante Verkleinerung des Fehlers sichtbar wird, damit das Neuronale Netz nicht zu sehr auf die Trainingsdaten angepasst ist und *Overfitting* auftritt. Sollte das passieren, kann das Netz auf anderen Datensätzen, wie dem Testdatenset, nicht mehr gut genug funktionieren, sodass diese eventuell nicht richtig zugeordnet werden (Hawkins 2004, vgl.).

3. Vorarbeiten und Stand der Wissenschaft

3.1. WordNet

Bei WordNet handelt es sich um eine lexikalische Datenbank, die von Miller (1998) entwickelt wurde. Sie enthält eine umfassende Sammlung an Adjektiven, Adverbien, Verben und Nomen aus der englischen Sprache. Die Datenbank konzentriert sich vor allem auf die Semantik der Wörter. In der ursprünglichen Form von 1998 enthielt sie fast 80.000 Nomen. In WordNet 2.1 ist sie inzwischen auf über 117.000 Nomen angewachsen (siehe Tabelle 3.1). Neben dem englischsprachigen WordNet sind zahlreiche Wordnets für andere Sprachen verfügbar, die von der Global WordNet Association gesammelt wurden[1].

Tabelle 3.1.: Auflistung der enthaltenen Wörter in WordNet 2.1
Quelle: `https://wordnet.princeton.edu/documentation/21-wnstats7wn`, abgerufen 07.12.2020

POS	einzelne Wörter	Synsets	Total
Nomen	117097	81426	145104
Verben	11488	13650	24890
Adjektive	22141	18877	31302
Adverbien	4601	3644	5720
Total	155327	117597	207016

Da die Semantik im Mittelpunkt steht, richtet sich auch der Aufbau der Datenbank danach und soll eine effiziente Suche ermöglichen. Die einzelnen Datensätze basieren auf sogenannten *Synsets*. Jedes *Synset* entspricht einer Bedeutung eines Wortes und sammelt die Synonyme. Beispielsweise sind zum Wort *chair* insgesamt sechs verschiedene *Synsets* verfügbar. Wobei vier davon das Nomen *chair* (z. B. einerseits als Stuhl, andererseits als Vorsitzender (*chairman*)) betrachten und die anderen beiden das entsprechende Verb. Bei einer Suche nach *chair* wird jedes *Synset* ausgegeben, dass *chair* als Synonym enthält.

[1] `http://globalwordnet.org/resources/wordnets-in-the-world/`, abgerufen 07.12.2020

Der noch wesentlich interessantere und wichtigere Aspekt an WordNet, der auch zum Erfolg dieser Anwendung geführt hat, ist die Verknüpfung der einzelnen *Synsets*. Dabei gibt es verschiedene Arten von Beziehungen zwischen den *Synsets* (Miller 1995, vgl. Tabelle 1):

- Hyponymie/Hypernymie
- Meronymie/Holonymie
- Antonymie (Gegenteile)
- Troponymie
- Semantische Implikation

Troponymie tritt nur im Zusammenhang mit Verben auf und beschreibt die Art und Weise von ihnen. Die Beziehung ist dabei ähnlich wie die Hyponymie bei Nomen, da die verknüpften Verben eine Unterkategorie des ursprünglichen Verbs darstellen (Bsp. *speak* → *whisper*) (Miller 1995, vgl.). Genau wie die Troponymie kommt die Semantische Implikation bei Verben vor und gibt an, welche anderen Verben daraus folgen. Ein Beispiel dafür ist das Verb *walk*. Wird es benutzt, kann daraus das Verb *step* gefolgert werden, da man sich beim Gehen bewegt und zwangsläufig Schritte laufen muss. WordNet stellt diese Beziehungen durch Referenzen auf die entsprechenden *Synsets* dar (Miller 1995, vgl.).

3.2. BERT

BERT (Devlin u. a. 2019) ist eine Abkürzung für *Bidirectional Encoder Representations from Transformers*. BERT bietet, im Vergleich zu klassischen Ansätzen in der Textinteraktion, ein neues Modell zum Lösen bestimmter *NLP* Aufgaben. Dies können Probleme auf einer Satzbeziehungsweise Absatzebene sein, wie das Fortführen eines Texts oder die Zusammenfassung eines Eingabetexts sein. Aber nicht nur Probleme auf Satzebene können gelöst werden, sondern auch auf Tokenbasis (Wordebene). Dazu gehört die *Named Entity Recognition* (NER) und vor allem die Beantwortung von Fragen (Devlin u. a. 2019, vgl. S.1).

Wie bereits ausgeführt, ist BERT aus einem *Bidirectional Transformer* aufgebaut, der im Gegensatz zu *Unidirectional Transformern* auf den Kontext links und rechts des Tokens zugreift, welche erstmals von Vaswani u. a. (2017) entwickelt wurden. Bevor BERT veröffentlicht wurde, wurden vor allem *Left-to-Right Transformer* verwendet, die nur auf den Text links vom Token zugreifen. Dabei ist es offensichtlich, dass Informationen rechts vom Wort nicht betrachtet werden und bei der Auswertung durch den Transformer verloren gehen. Der Kontext

wird dadurch nicht so gut wie bei einem bidirektionalen Modell erkannt.
Der große Vorteil bei BERT liegt darin, dass man als Endnutzer das Neuronale Netz nicht von Grund auf trainieren muss, sondern auf einem bereits trainierten Modell aufbauen kann. Der Kerngedanke im Aufbau des Frameworks besteht darin, dass es ein *Pretraining* und ein *Fine-Tuning* gibt. Mittels *Fine-Tuning* kann BERT innerhalb von ein paar Stunden fertig trainiert werden und erzielt dabei gute Ergebnisse (Devlin u. a. 2019, vgl. S.5). Das *Pretraining* basiert auf *Unsupervised Learning*. Im Gegensatz dazu wird beim *Fine-Tuning* gelabelter Input verwendet und fällt damit in die Kategorie des *Supervised Learnings*. Das Modell wird dabei immer mit den gleichen Parametern des *Pretrainings* geladen, wobei es egal ist, ob man gerade im *Pretraining* oder *Fine-Tuning* ist. Danach werden weitere eventuell vorhandene Parameter des *Fine-Tunings* geladen. Das Modell von BERT besteht aus zwölf *Encodern*, die jeweils 768 Hiddenlayers enthalten. Zusätzlich gibt es noch ein umfassenderes Modell, dass 24 *Encoder* mit 1024 Hiddenlayers besitzt.

Damit das Training von BERT gute Ergebnisse erzielt, wurde das *Masked Language Model* verwendet, welches die Arbeit von Taylor (1953) als Grundlage nutzt (Devlin u. a. 2019, vgl.). Dabei werden verschiedene Token im Text maskiert. Diese werden vorher zufällig gewählt. Die Maskierung ist sehr wichtig beim Arbeiten mit *bidirektionalen Transformern*, da diese mächtiger sind als normale *Left-to-Right Transformer* und dadurch auch die Standardlernmechanismen aushebeln können. Die Autoren erklären es folgendermaßen: „bidirectional conditioning would allow each word to indirectly ‚see itself', and the model could trivially predict the target word in a multi-layered context" (Devlin u. a. 2019, S.4). Es werden 15% der Token maskiert, wobei das auf drei verschiedene Arten passiert:

- Ersetze das Wort durch [MASK] (80% der Fälle)
- Ersetze das Wort durch ein zufälliges anderes Wort (10% der Fälle)
- Ersetze das Wort nicht (10% der Fälle)

Das Modell soll dann wiederum das maskierte Wort herausfinden.
Zur Beantwortung von Fragen haben die Autoren BERT dahingehend trainiert, dass es in der Lage ist, die Beziehung zwischen Sätzen zu verstehen. Als Training nutzen sie die sogenannte *Next Sentence Prediction*. Dabei werden zwei Sätze angegeben und das Modell muss nun herausfinden, ob der zweite Satz auf den ersten folgen kann oder nicht (Devlin u. a. 2019, vgl.). Im Training sind die eine Hälfte der Sätze zufällig aufeinander folgend und die andere zueinander passend.

3.2.1. Benchmarks für BERT

BERT wurde den Benchmarks GLUE und SQuAD unterzogen und schnitt bei diesen überdurchschnittlich gut ab.

3.2.1.1. GLUE Benchmark

Der erste Test den BERT bestehen soll, ist der *General Language Understanding Evaluation* Benchmark (GLUE) (Wang u. a. 2019). Dieser basiert auf dem Fakt, dass Menschen die Fähigkeit haben Sprache schnell zu verstehen und dabei das Gesagte richtig einordnen können (Wang u. a. 2019, vgl.). Dies fällt Computern schwer, da sie auf solche Aufgaben erst trainiert werden müssen. Der Benchmark testet die Programme darauf, wie gut sie mit der Sprache umgehen können. Dies beinhaltet einerseits Erkennung von Gefühlen, andererseits müssen Folgerungsbeziehungen extrahiert werden.
Dafür wurde ein Classifier im *Fine-Tuning* von BERT angehängt, der jeweils für die Aufgabe in GLUE trainiert wurde. Bei allen Aufgaben übertraf BERT vergleichbare Systeme wie zum Beispiel OpenAI GPT von Radford u. a. (2018).

3.2.1.2. SQuAD Benchmark

Die beiden weiteren Benchmarks, die vorgestellt werden, testen die Beantwortung von Fragen. Sie sind Teil des *Stanford Question Answering Datasets* (Rajpurkar, Jia und Liang 2018; Rajpurkar, Zhang u. a. 2016) . Wobei sie jeweils eine Frage und einen zugehörigen Text, der die Antwort beinhaltet, enthalten. Der Unterschied zwischen den beiden Benchmarks liegt in der Länge der enthaltenen Antworten. Das Programm muss dann die Stelle der Antwort im Text herausfinden. Dafür wurde wieder ein *Fine-Tuning* von BERT durchgeführt. Ähnlich zum *General Language Understanding* erzielt BERT ebenfalls sehr gute Ergebnisse, die besser als die der Konkurrenz sind.

3.2.2. DistilBERT

Gerade das große Modell von BERT enthält sehr viele Parameter und der Trend geht eher zu noch größeren Modellen (siehe auch Abbildung 3.1) (Sanh u. a. 2020, vgl.). Zwar lassen sich durch größere Modelle bessere Ergebnisse erzielen, allerdings wird gleichzeitig auch mehr Rechenleistung benötigt. DistilBERT versucht dem entgegenzuwirken, indem es das Wissen von BERT „destilliert“. Dabei wird aus einem größeren Modell, dem Lehrer, ein kleineres, kompakteres Neuronales Netz erzeugt, der Student (Hinton, Vinyals und Dean 2015; Buciluǎ, Caruana und Niculescu-Mizil 2006, vgl.). Dies ist möglich, da beim Training des Lehrers nicht

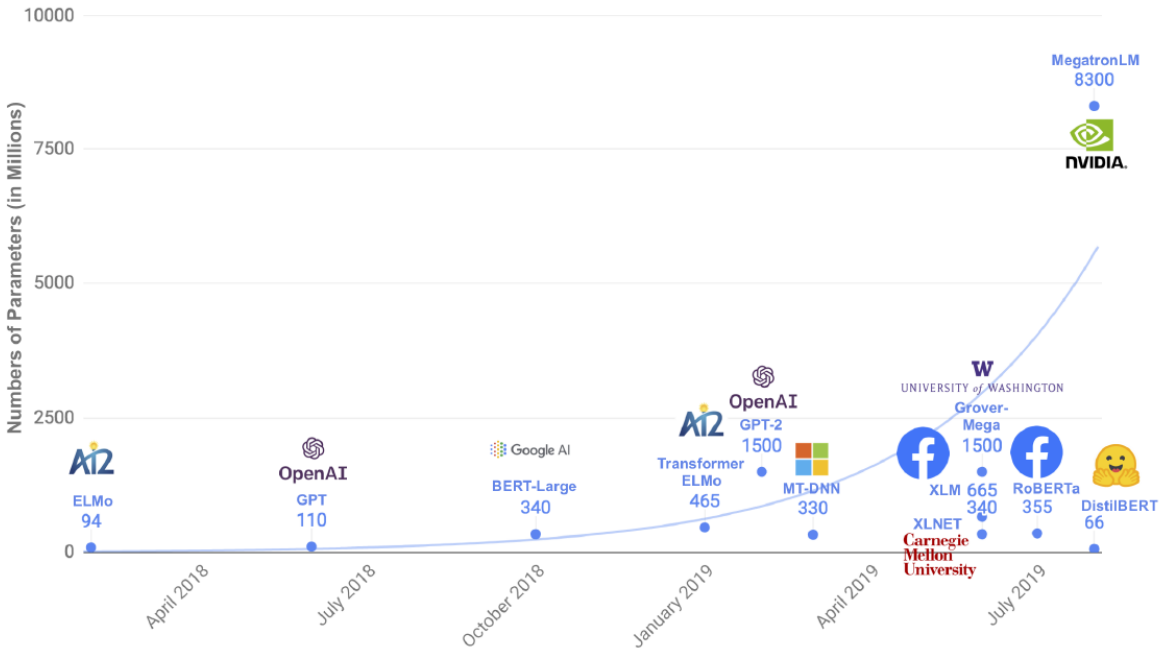

Abbildung 3.1.: Anzahl an Parametern von verschiedenen Modellen über die Zeit betrachtet (Sanh u. a. 2020, vgl. Fig.1)

jedes Merkmal gleich gut erkannt wird. Mithilfe des Studenten bereinigt man das Modell um diese Merkmale. DistilBERT ist im Grunde gleich aufgebaut wie BERT, aber der Unterschied liegt in der Anzahl der *Hiddenlayers*. Diese wurden im Vergleich zu BERT um die Hälfte reduziert.

DistilBERT wurde ebenfalls mit den Benchmarks GLUE und SQuAD getestet. Dabei kam heraus, dass DistilBERT nur in geringem Maße schlechter ist als BERT, obwohl es um 40% weniger Parameter hat und damit deutlich kleiner als BERT ist (Sanh u. a. 2020, vgl S.3).

3.3. Word2Vec

Word2Vec ist ein Machine Learning Modell, das unter anderem 2013 von Mikolov u. a. (2013) veröffentlicht wurde. Das Ziel des Algorithmus ist die Klassifizierung von Worten, sodass auch semantische Beziehungen dargestellt werden können. Dabei werden Worten, die als Vektoren dargestellt werden, lineare Beziehungen hinzugefügt. Beispielsweise haben „Paris“ → „Frankreich“ die gleiche Art von Beziehung wie „Berlin“ → „Deutschland“ (Mikolov u. a. 2013, vgl.). Das heißt, dass Worte, die von der Bedeutung ähnlich sind, ebenfalls ähnliche Vektordarstellungen besitzen.
Der Vorteil an der Darstellung durch Vektoren, liegt darin, dass sich durch Addieren und Subtrahieren genau diese Beziehungen abbilden lassen. *vector(„Paris“) - vector(„Frankreich “) + vector(„Deutschland“)* ergibt dann als Ergebnis den Vektor des Worts *Berlin*. Das genau der Vektor *Berlin* als Ergebnis ausgegeben wird, ist aber nur der Idealzustand. In der Praxis wird das Modell einen recht ähnlichen Vektor ausgeben, wobei man dann mit Hilfe der Kosinusdistanz den nächsten passenden Vektor aus dem Datenset erhält. Sie ist wie folgt definiert:

$$Kosinus\ Distanz: \quad \cos(\alpha) = \frac{\vec{x} \cdot \vec{y}}{|\vec{x}| \cdot |\vec{y}|}$$

3.3.1. Aufbau

Es werden zwei verschiedene Modelle konstruiert und gegeneinander getestet. Die erste Variante nennt sich *Continous Bag of Words* und bestimmt aus dem Kontext das fehlende Wort. „The probabilistic feedforward neural network language model has been proposed in (Bengio u. a. 2003)“ (Mikolov u. a. 2013, S.3). Es besteht aus den vier verschiedenen Schichten *Input-*, *Projection-*, *Hidden-*, und *Outputlayer*. Dieses Neuronale Netz wird als Grundlage genutzt, wobei zwei zentrale Anpassungen getätigt wurden. Erstens wurde das *Hiddenlayer* weggelassen und zweitens wurde das *Projectionlayer* angepasst.

Die Eingabe im *Inputlayer* wurde als *One-Hot-Encoding* umgesetzt, wobei die Größe des Encodings auf der Größe der Vokabelmenge beruht. Als Grundlage dafür werden jeweils die Sätze links und rechts des fehlenden Worts genommen. Die gewählte *Windows Size*, also der Ausschnitt des Texts, beeinflusst dabei die Genauigkeit des Modells. Je mehr Kontext vorhanden ist, desto besser ist die Vorhersage. Allerdings ist zu berücksichtigen, dass gleichzeitig auch die Komplexität steigt und damit die Berechnungen deutlich kostenintensiver werden (Mikolov u. a. 2013, vgl.).

Das zweite Modell nennt sich *Continous Skip-gram Model* und betrachtet den umgekehrten Fall vom *Continous Bag of Words*-Modell, die Eingabe ist ein Wort, aus dem dann die Nachbarwörter bestimmt werden. Das Neuronale Netz ist gleich aufgebaut wie das vorherige und hier zeigt sich ebenfalls der Zusammenhang zwischen *Windows Size* und Komplexität beziehungsweise Berechnungskosten (Mikolov u. a. 2013, vgl.).

3.3.2. GloVe

Ein weiteres Modell, das semantische Aspekte durch Vektoren darstellen kann, ist GloVe (Pennington, Socher und Manning 2014). Das Modell basiert auf den Wahrscheinlichkeiten von Koexistenzen verschiedener Wörter. Ein Beispiel, das von den Autoren genannt wird, sind die Wörter *ice* und *steam* (Pennington, Socher und Manning 2014, vgl.). Dabei werden die Wahrscheinlichkeiten zur Koexistenz zwischen jeweils einem dieser Wörter und einem weiteren Wort miteinander verglichen. Je nachdem wie sich die Wahrscheinlichkeiten verhalten, kann man sagen, ob diese Wörter in einem gemeinsamen Kontext vorkommen oder nicht. Dies lässt sich mit Hilfe der gewichteten linearen Regression als mathematisches Modell darstellen, dass dann mit *Unsupervised Learning* trainiert werden kann.

Das Modell wurde anhand von zwei Benchmarks getestet und mit anderen Systemen verglichen. Dafür wurden verschiedene Modelle mit unterschiedlichen Tokendatensätzen unter anderem von Wikipedia trainiert, die mindestens eine Milliarde Token enthalten.
Der erste Benchmark beschäftigt sich mit Wortanalogien, die durch GloVe gelöst werden müssen. Die Fragestellung lautet: „*d* verhält sich zu *c*, wie *b* zu *a*“. *a*, *b* und *c* sind gegeben und *d* muss um das richtige Wort ergänzt werden. Im verwendeten Benchmark von (Sang und Meulder 2003) gibt es die Unterteilung in semantische und syntaktische Aufgaben. Beim semantischen Teil werden meistens Eigennamen überprüft, beim Syntaktischen hauptsächlich Verben und Adjektive. GloVe war in den Benchmarks insgesamt immer besser als die anderen Word2Vec-Tools, mit denen es verglichen wurde. Mit dem zweiten Benchmark wird die *Word-similarity* überprüft. Hierbei geht es um die Erkennung von Eigennamen, welche unter anderem in Zeitungsartikeln veröffentlicht wurden (Pennington, Socher und Manning 2014, vgl.). Auch in diesem Test schneidet GloVe besser ab, obwohl die Dimension der genutzten Vektoren kleiner ist und nur bei 50 liegt. Dies zeigt, dass auch mit einer kleineren Dimension eine gute Unterscheidung der Wörter möglich ist, was die Modelle etwas kleiner macht, da weniger In- und Outputknoten bei den Modellen verwendet werden können. Außerdem kann das Training im Vergleich zu anderen Word2Vec Tools verkürzt werden und man erhält dadurch schneller bessere und genauere Modelle (Pennington, Socher und

Manning 2014, vgl.).

3.4. PartNet

PartNet wird von den Autoren als „large-scale dataset of 3D objects annotated with fine-grained, instance-level, and hierarchical 3D part information“ beschrieben (Mo u. a. 2019, vgl. S.1). PartNet schafft eine Grundlage, mit der man 3D-Objekte, die aus der ShapeNet-Datenbank entstammen (Chang u. a. 2015), annotieren kann. Das Datenset, dass manuell erstellt wurde, beinhaltet 24 Kategorien, wobei insgesamt 573.585 Objekte annotiert wurden. Die Kategorien sind 24 Objekte wie beispielsweise ein Laptop oder eine Spülmaschine (Mo u. a. 2019, vgl. Fig. 2). Diese wurden dann hierarchisch annotiert, sodass mit jeder neuen Ebene das vorherige Objekt in feinere Teilobjekte aufgeteilt wurde. Für diese Aufteilung haben sie sechs verschiedene Kriterien aufgestellt, an denen sich orientiert wurde:

- well-defined
- consistent
- compact
- hierarchical
- atomic
- complete

Für die hierarchische Datenstruktur wird ein *And-Or-Tree* genutzt, welcher mittels einer webbasierten GUI von den Annotatoren abgefragt wird. Daraufhin wird der Baum erstellt, beziehungsweise aktualisiert. Dabei nutzen sie Fragen, wie „What parts does it have?“ (Mo u. a. 2019, vgl. Fig. 4) auf die der Annotator antworten muss. Zur Frage im Beispiel würde aus der Antwort ein *AND-Knoten* erstellt werden. Es wird solange weitergefragt, bis es keine kleinere Aufteilung mehr gibt. Der Baum würde also ein Blatt anfügen.
Mit Hilfe dieses Datensets, werden dann vier verschiedene aktuelle *Semantic-Segmentation* Algorithmen getestet und ein Benchmark ausgeführt, welcher gute Ergebnisse liefert (Mo u. a. 2019, vlg.).

4. Theoretische Vorarbeiten

4.1. Pipeline

Zur Verarbeitung des Texts müssen nacheinander verschiedene Schritte durchgegangen werden, wodurch sukzessive mehr Informationen extrahiert werden. Zur Pipeline gehören die folgenden Schritte (siehe auch Abbildung 4.2):

1. Tokenizer
2. Part of Speech Tagging
3. Objekterkennung
4. Verarbeitung durch BERT (Wolf u. a. 2020)
5. Bestimmung des Kontexts
6. Holonymiebestimmung

Die grundlegenden Textverarbeitungen, wie Tokenizer und Part of Speech Tagging, werden durch den TextImager von Hemati, Uslu und Mehler (2016) durchgeführt. Die beiden verwendeten Tagger sind aus dem CoreNLP Projekt (Qi u. a. 2018). Sie dienen dazu, dass jedes Wort mit dem entsprechenden lexikalischen Typ versehen wird. Dies ist relevant, da Objekte immer Nomen sind, und in den folgenden Schritten nur noch diese betrachtet werden. Sobald alle Wörter eingeordnet wurden, müssen als nächstes die Objekte herausgefiltert werden. Dafür wird WordNet (Miller 1998) zu Hilfe genommen.

Nachdem alle Wörter klassifiziert wurden, werden die Nomen genauer betrachtet und es wird überprüft, ob das aktuelle Nomen ein Objekt darstellt. Da die Datenbank von WordNet für jedes *Synset* weitere Metadaten enthält, werden diese Daten ausgelesen, wodurch dann schlussendlich die Objekte herausgefiltert werden können. Dabei werden vor allem die Hypernyme des betrachteten Worts angeschaut. WordNet stellt die Hypernym und Meronym Strukturen als Baum dar, wodurch sich die Frage, ob ein Wort ein Objekt beschreibt, einfach lösen lässt. Jedes *Synset* in WordNet, welches ein Objekt beschreibt, enthält irgendwo auf

1 sense of bathtub

Sense 1
bathtub, bathing tub, bath, tub -- (a relatively large open container that you fill with water and use to wash the body)
=> vessel -- (an object used as a container (especially for liquids))
=> container -- (any object that can be used to hold things (especially a large metal boxlike object of standardized dimensions that can be loaded from one form of transport to another))
=> instrumentality, instrumentation -- (an artifact (or system of artifacts) that is instrumental in accomplishing some end)
=> artifact, artefact -- (a man-made object taken as a whole)
=> whole, unit -- (an assemblage of parts that is regarded as a single entity; "how big is that part compared to the whole?"; "the team is a unit")
=> object, physical object -- (a tangible and visible entity; an entity that can cast a shadow; "it was full of rackets, balls and other objects")
=> physical entity -- (an entity that has physical existence)
=> entity -- (that which is perceived or known or inferred to have its own distinct existence (living or nonliving))

Abbildung 4.1.: Ein Beispiel für den Hypernymbaum in WordNet (Miller 1998). Das Synset bathtub enthält eine ganze Reihe an Hypernymen, die bis zum Synset entity reichen

dem Weg zur Wurzel des Baums das *Synset* „*object, physical object*“ als Knoten (siehe Abbildung 4.1). Wenn dieses *Synset* vorkommt, dann wird das Wort als Objekt markiert.

Die verbleibenden Wörter, die als Objekte klassifiziert wurden, werden nun durch den ObjectTagger weiterverarbeitet (siehe Abschnitt 5.5). Er ist eine modifizierte Version von DistilBERT (Sanh u. a. 2020; Wolf u. a. 2020) und wird in dieser Arbeit dazu verwendet, Objekte genauer zu klassifizieren. Wenn im Text nur von einem Tisch gesprochen wird, ist dies ungenau und es könnte je nach beschriebener Szene entweder ein *Esstisch* oder *Stehtisch* sein. DistilBERT wird dazu genutzt, um das passende Präfix wie „Ess-“ oder „Steh“ zum Wort hinzuzufügen, damit die Unterscheidungen zwischen Objekten noch präziser werden. BERT bietet die Möglichkeit, Stellen im Text zu maskieren beziehungsweise zu markieren. Da für die Anwendung Stellen, bei denen von Objekten die Rede ist, wichtig sind, wird jeweils eine Maskierung vor das Wort gesetzt, das ein Objekt beschreibt. Dieser modifizierte Text wird dann durch den ObjectTagger bearbeitet und die Maskierungen werden durch passende Präfixe ersetzt.

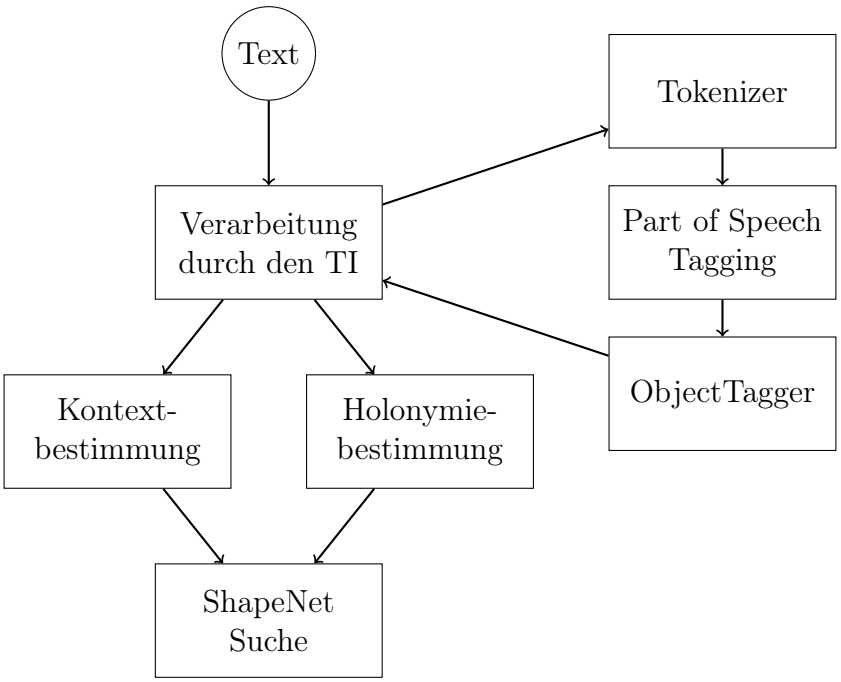

Abbildung 4.2.: Übersicht über den Ablauf und den Zusammenhang der einzelnen Systeme

Nachdem der Text durch den ObjectTagger verarbeitet wurde, kann man nun als nächstes die Teil-Ganz Beziehungen bestimmen. Dazu wird ein *Feed-Forward Network* trainiert, das diese Beziehungen erkennen soll und das passende Holonym ausgibt.
Nach dem gleichen Prinzip wird der Kontext bestimmt. Auch hier wird ein Neuronales Netz verwendet, welches aber nicht nur das Wort erfasst, sondern noch einen gewissen Abschnitt rechts und links davon.

5. Technische Umsetzung

Die Anwendung ist in das VR-System des Lehrstuhls, den VAnnotatoR, integriert worden. Dieser bietet noch weitere Funktionen, wie zum Beispiel einen Annotator für Texte, der durch Controllergesten gesteuert werden kann (Spiekermann, Abrami und Mehler 2018; Kühn, Abrami und Mehler 2020; Mehler u. a. 2018; Abrami, Henlein u. a. 2020b; Abrami, Henlein u. a. 2020a). Neben diesen für den User sichtbaren Anwendungen, bietet der VAnnotatoR aber auch viele Hintergrundsysteme an, die es ermöglichen, die unterschiedlichen Anwendungen miteinander zu verknüpfen.

5.1. Texteingabe und Texterstellung

5.1.1. Laden eines Texts

Die einfachste Möglichkeit, einen Text zur Verarbeitung auszuwählen, ist das Laden einer bereits vorhandenen Datei. Dies ist über den Document Tab des SceneBuilders von Abrami, Henlein u. a. (2020a, vgl.) möglich (siehe Abbildung 5.1). Solange keine Datei geladen wurde, wird eine Liste von Dateien angezeigt, die man durchblättern und so die gewünschte Datei auswählen kann. Die angewählte Datei wird daraufhin vom *ResourceManager*[1] in die virtuelle Szene geladen. Daraufhin kann der User entweder das Text2Scene Interface starten (siehe Abschnitt 5.3) und dort weiterarbeiten oder direkt die Objekte laden mit dem erweiterten und weiter automatisierten Text2Scene Interface (siehe Abschnitt 5.2).

5.1.2. Erstellen von Texten

Sollte der Text, den der User bearbeiten möchte, nicht als vorgefertigte Datei vorhanden sein, muss er ihn manuell in der virtuellen Umgebung erstellen. Dazu kann er den integrierten TextObjectBuilder verwenden (siehe Abbildung 5.2). Er lässt sich durch Anklicken des Plus Symbols im SceneBuilder öffnen, sofern keine Datei geladen wurde. Daraufhin erscheint er vor dem User, der dann eine Textdatei erstellen kann. Der Text wird beim Klicken auf das Plus Symbol erstellt und an den *ResourceManager* verschickt, sodass die Datei dann

[1] `https://resources.hucompute.org/`

nach demselben Prinzip wie in Unterabschnitt 5.1.1 geladen werden kann. Dabei wird sie unter dem eingegebenen Titel abgespeichert. Wenn das Fenster nicht mehr benötigt wird, kann es durch Klicken auf *Close* wieder geschlossen werden. Der letzte Button (↻) ermöglicht es, die beiden Eingabefelder zurückzusetzen. Sobald eine Datei erstellt wurde, kann der User mit der Verarbeitung des Texts im Text2Scene Interface (siehe Abschnitt 5.3) fortfahren.

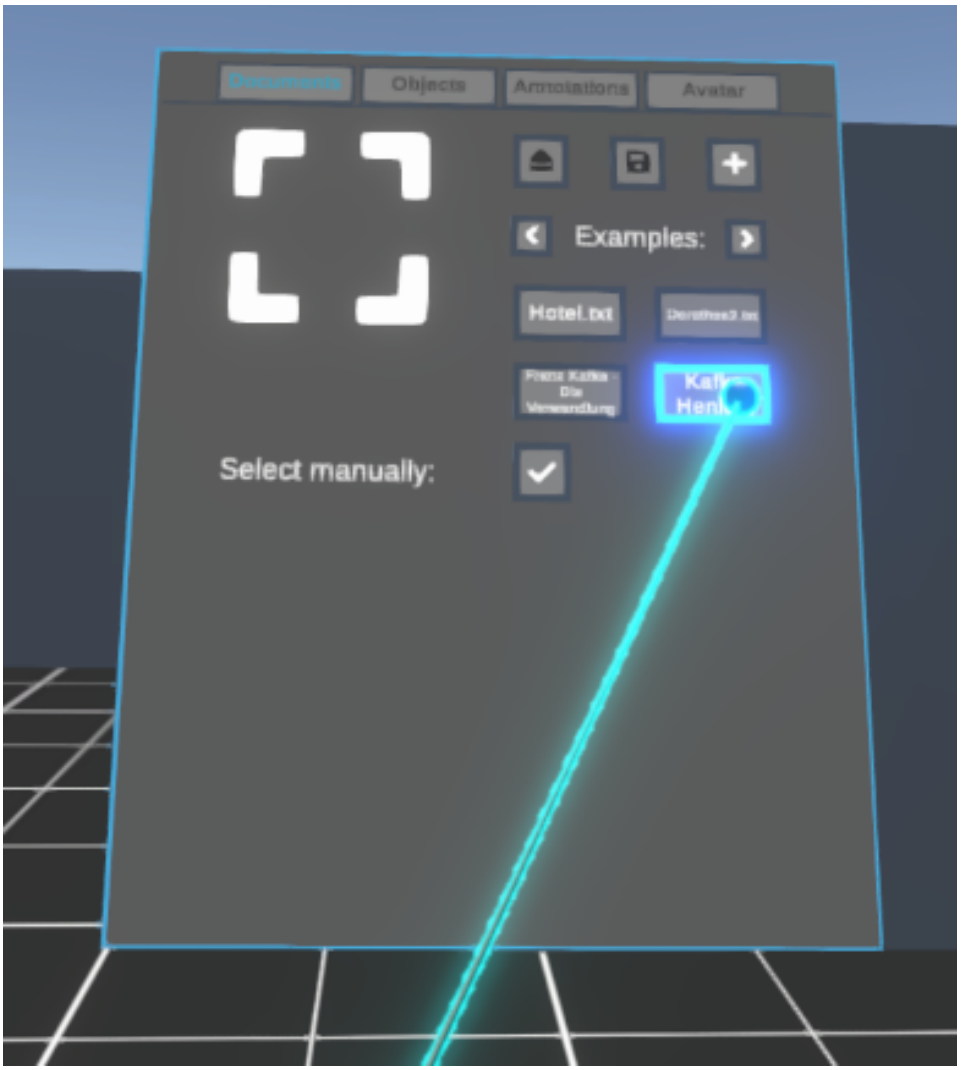

Abbildung 5.1.: Abgebildet ist das Interface des SceneBuilders (Abrami, Henlein u. a. 2020a), mit dem der User Texte laden kann

Abbildung 5.2.: Das Layout des TextObjectBuilders orientiert sich am Design des Vannotators (Abrami, Mehler und Spiekermann 2019; Mehler u. a. 2018; Spiekermann, Abrami und Mehler 2018)

5.2. Automatisierte Verarbeitung durch das Text2Scene Interface

Die erste Möglichkeit einen Text zu verarbeiten, wird aktiv, falls der User im SceneBuilder das Häkchen beim Unterpunkt *Select manually* weggeklickt hat (siehe Abbildung 5.1). Sobald ein Dokument geladen wird, startet die Verarbeitung des Texts automatisch durch die Pipeline. Der Text wird an den TextImager geschickt und durch die Neuronalen Netze verarbeitet. Die Rückgabe wird in einer modifizierten Version der *IsoSpatialEntity* Datenstruktur gespeichert (ISO 2014; ISO 2020), die die Ergebnisse des ObjectTaggers (Abschnitt 5.5), das Holonym (Abschnitt 5.6) und den Kontext (Abschnitt 5.7) enthält. Die *IsoSpatialEntity* bildet eine Grundlage zur Darstellung von 3D-Objekten (Pustejovsky, Moszkowicz und Verhagen 2011).

Für jedes *IsoSpatialEntity* Objekt wird ein Suchbefehl, bestehend aus den drei Wörtern der einzelnen Tools und dem eigentlichen Wort, an die ShapeNet-Datenbank (Chang u. a. 2015)

geschickt. Dies passiert in zwei Schritten, da die API aufgeteilt ist und zwei verschiedene Bereiche von ShapeNet angesprochen werden. Die erste Abfrage an die API besteht aus den Suchwörtern und durch sie wird ShapeNet nach potenziell passenden Objekten durchsucht. Es wird eine JSON-Datei mit den Metadaten der gefundenen Objekte zurückgegeben, wobei die Anwendung automatisch das erste, folglich das beste, auf die Anfrage passende Objekt auswählt. In einer zweiten Abfrage an ShapeNet erhält das System das entsprechende 3D-Objekt und kann es daraufhin in die Szene des VANNOTATOR laden. Die für jedes Wort erhaltenen 3D-Objekte werden in einer Linie vor dem User platziert (siehe Abbildung 5.4), sodass er nun mit ihnen arbeiten kann. Der User hat die Möglichkeit ein bestimmtes Objekt auszuwählen (Abbildung 5.5), indem er darauf zeigt. Sobald es ausgewählt ist, kann er das Objekt im Raum bewegen (Abbildung 5.6), wodurch nach und nach die Szene aus dem Text nachgestellt werden kann (siehe Abbildung 5.7).

Neben der vollständig automatischen Verarbeitung des Texts, hat der User die Option, das Laden der 3D-Objekte hinauszuzögern und manuell über einen Button zu starten. Dies kann Sinn ergeben, wenn er manuell Wörter als Objekt annotiert. Die Funktion ist im *Annotation Window* des VANNOTATORS enthalten (siehe Abbildung 5.3) (Abrami, Mehler und Spiekermann 2019, vgl.). Es lässt sich im Annotation Tab des SCENEBUILDERS öffnen. Die Textverarbeitung startet in diesem Fall auch beim Laden der Datei, allerdings werden die Objekte erst platziert, wenn der User den Button „Objekte platzieren“ anklickt. Dadurch werden die Anfragen an ShapeNet gestartet und die geladenen 3D-Objekte werden vor dem User aufgebaut.

Abbildung 5.3.: Das Annotation Window ermöglicht die manuelle Annotation des Texts (Abrami, Henlein u. a. 2020a)

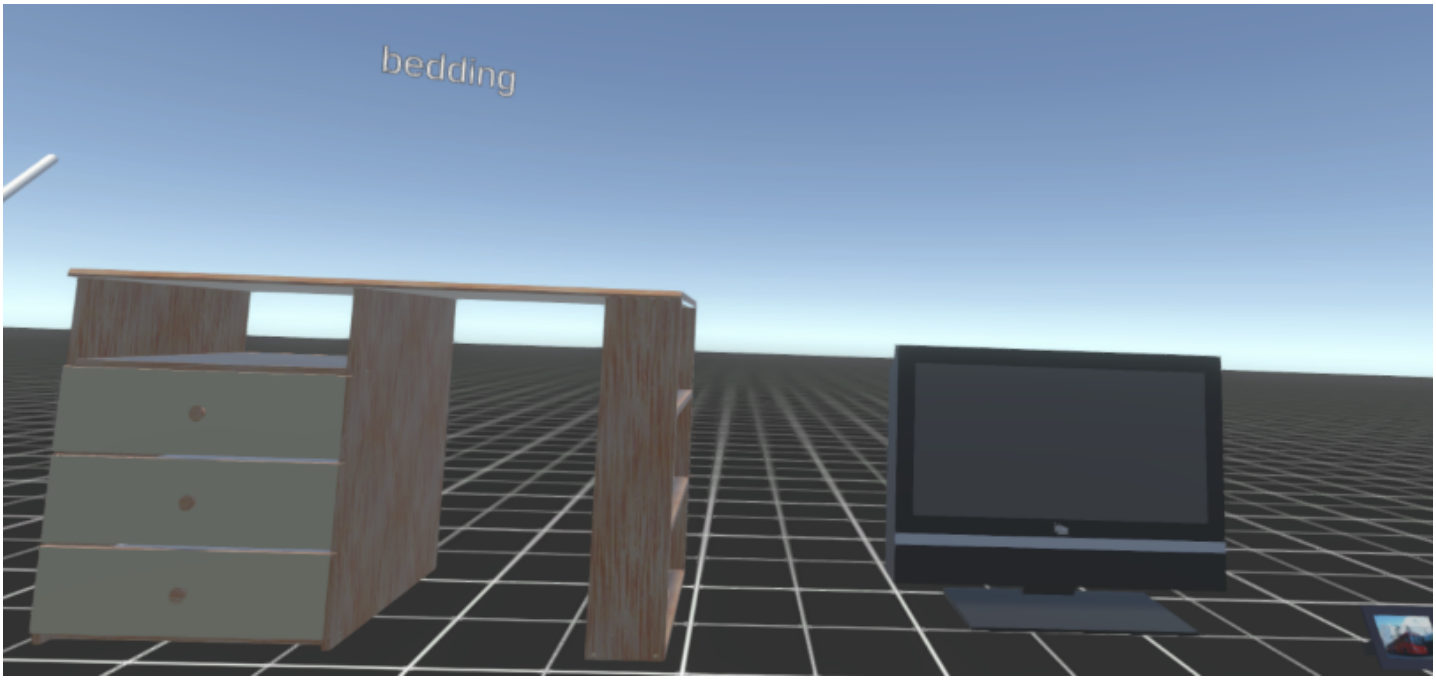

Abbildung 5.4.: Platzierung der geladenen Objekte vor dem User

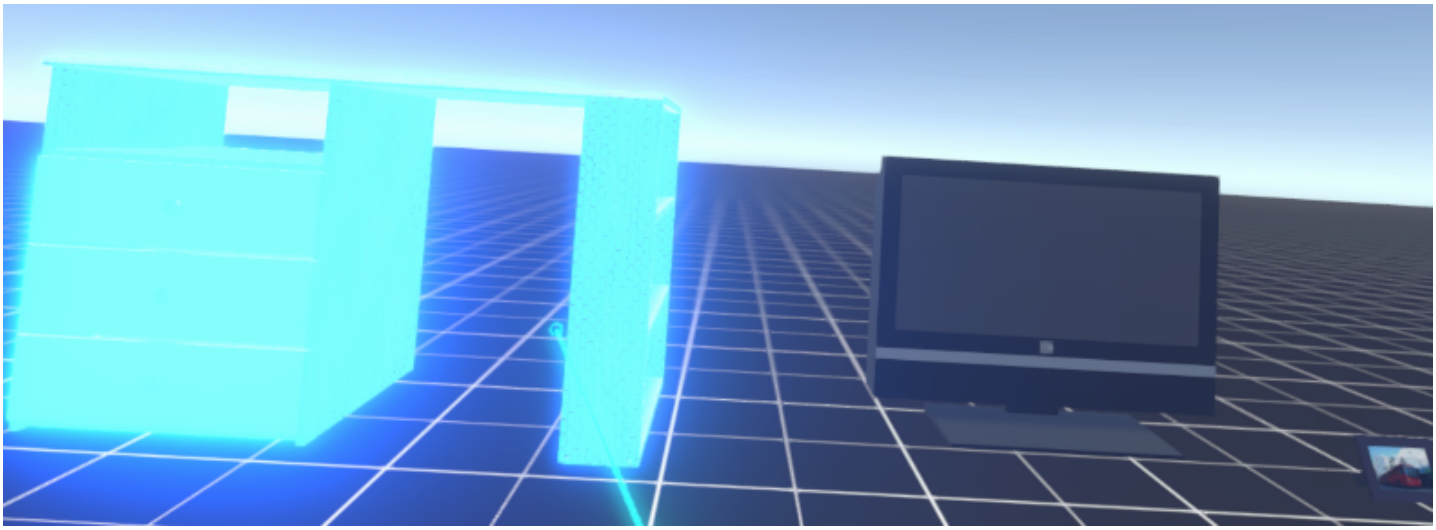

Abbildung 5.5.: Beim Auswählen des Objekts wird dieses farbig hervorgehoben und ermöglicht das Platzieren im Raum (Abrami, Henlein u. a. 2020a, vgl.)

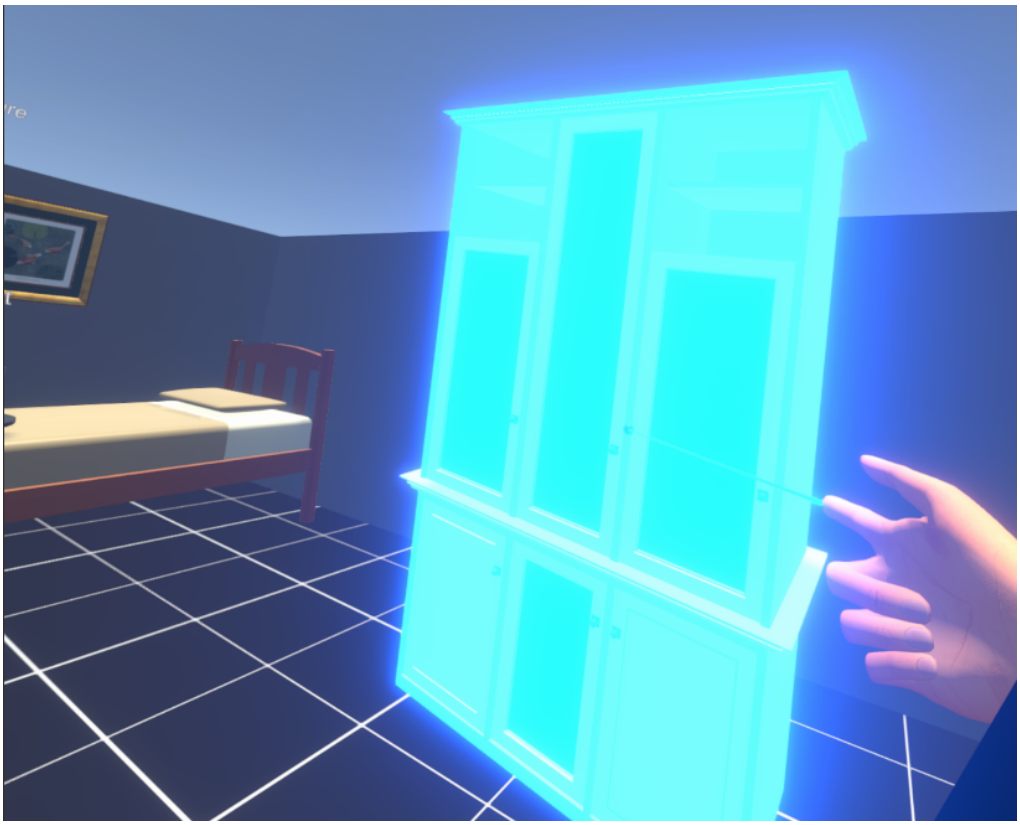

Abbildung 5.6.: Ein Objekt wird im Raum platziert

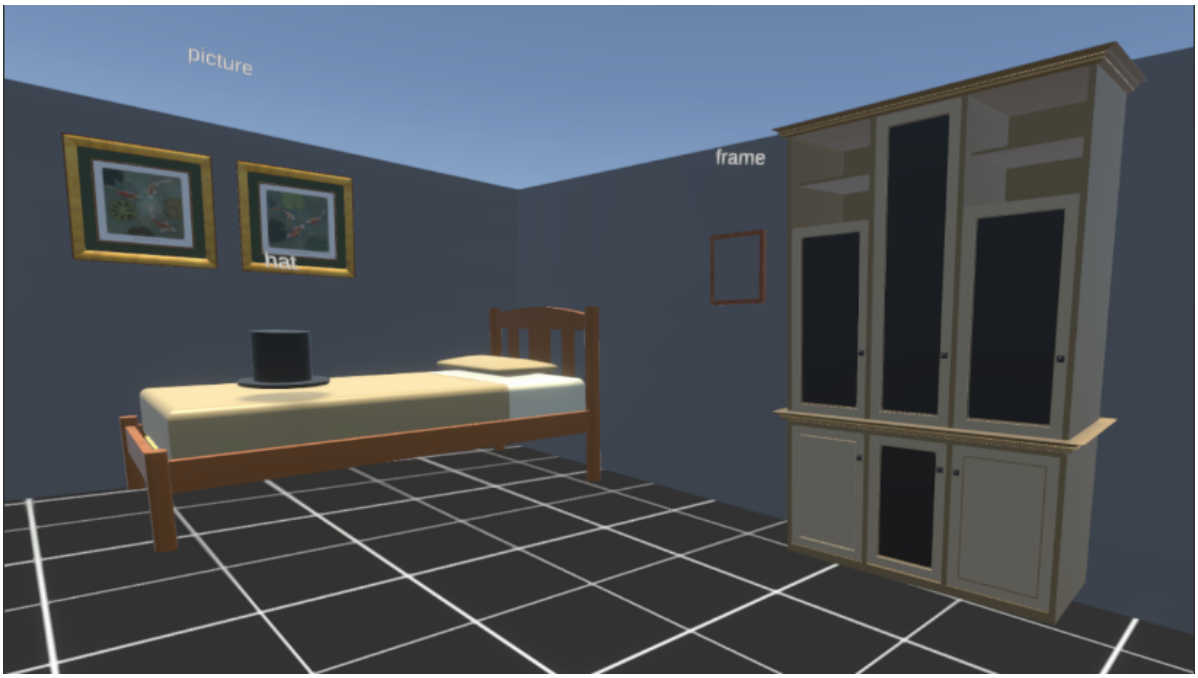

Abbildung 5.7.: Hier sind einige Objekte zu sehen, die in einer Beispielszene platziert wurden

5.3. Text2Scene Interface

Das Text2Scene Interface (siehe Abbildung 5.10 und Abbildung 5.8) ist in den DataBrowser (Spiekermann, Abrami und Mehler 2018, vgl.) integriert und muss erst mit Hilfe der Smartwatch des VR-Users (siehe Abbildung 5.9) gestartet werden. Dies ist in zwei Schritten möglich:

1. Auswählen des DataBrowsers in der Smartwatch
2. DataBrowser in die virtuelle Szene ziehen, wodurch er geöffnet wird

Der DataBrowser ist der zentrale Objekt Browser, mit dem man unter anderem auf der Festplatte nach Dateien suchen kann (Kett u. a. 2018, vgl.). Für die Text2Scene Interaktion spielt der ShapeNet Browser eine entscheidende Rolle. Wird er gestartet, öffnet sich links automatisch das Text2Scene Interface, sobald die Word2Vec Datenbank geladen wurde. Falls der User eine Datei bereits geladen hat, wird diese direkt an die Pipeline geschickt (siehe Abschnitt 4.1) und das verarbeitete Ergebnis ausgegeben. Ähnlich zur Automatischen Verarbeitung wird die Rückgabe in der *IsoSpatialEntity* Datenstruktur gespeichert und bietet die Grundlage für die nachfolgende Bearbeitung durch den User.

Der Text wird im Text2Scene Interface angezeigt (siehe Abbildung 5.10), während parallel im ShapeNet Interface die Suchergebnisse für das in Rot ausgewählte Wort angezeigt werden. Der User kann nun ein Objekt aus dem ShapeNet Interface auswählen, welches aus seiner Sicht am besten in die Szene passt. Das ausgewählte Objekt wird nicht direkt als 3D-Objekt in die virtuelle Umgebung geladen, sondern als erstes wird eine *Virtual Resource* erstellt (Abrami, Henlein u. a. 2020a, vgl.). Die *Virtual Resource* ist die Schnittstelle zwischen dem 3D-Objekt und dem geöffneten Dokument. Um die *Virtual Resource* in ein Objekt zu konvertieren, muss sie in den Slot des sogenannten Object Tabs des SceneBuilders (Abrami, Henlein u. a. 2020a, vgl.) (siehe auch Abbildung 5.1) geschoben werden. Sobald sie im Slot erkannt wird, lädt dieser das 3D-Objekt, welches der User nun im Raum platzieren kann. Dabei kann sowohl die Position, die Größe als auch die Drehung des Objekts verändert werden.

Hat man ein Objekt platziert, kann man zum nächsten Objekt übergehen. Dazu gibt es im Text2Scene Interface über dem Textfeld Vorwärts- und Rückwärtspfeile, mit denen man durch die Objekte im Satz navigieren kann. Wird ein neues Objekt im Text ausgewählt, wird wieder parallel im ShapeNet Interface nach 3D-Objekten gesucht. Alternativ kann man die Objekte, die im aktuell betrachteten Satz vorhanden sind, in die Szene laden. Dafür wurde der Button „Objekte platzieren“ entworfen. Dabei werden identisch zur Automatisierten

Verarbeitung die ShapeNet-Objekte in einer Reihe vor dem User platziert, sodass er sie platzieren kann. Er kann für jeden Satz genau einmal die Objekte laden. Um zwischen den Sätzen zu navigieren, gibt es an der rechten Seite des Textfelds Vorwärts- und Rückwärtspfeile, die beim Anklicken den vorherigen beziehungsweise den nächsten Satz im Textfeld anzeigen. Falls der User das *Annotation Window* geöffnet hat, kann er durch Anklicken der Satzindizes, die sich links neben dem Satz befinden, den passenden Satz im Text2Scene Interface laden (siehe Abbildung 5.3).

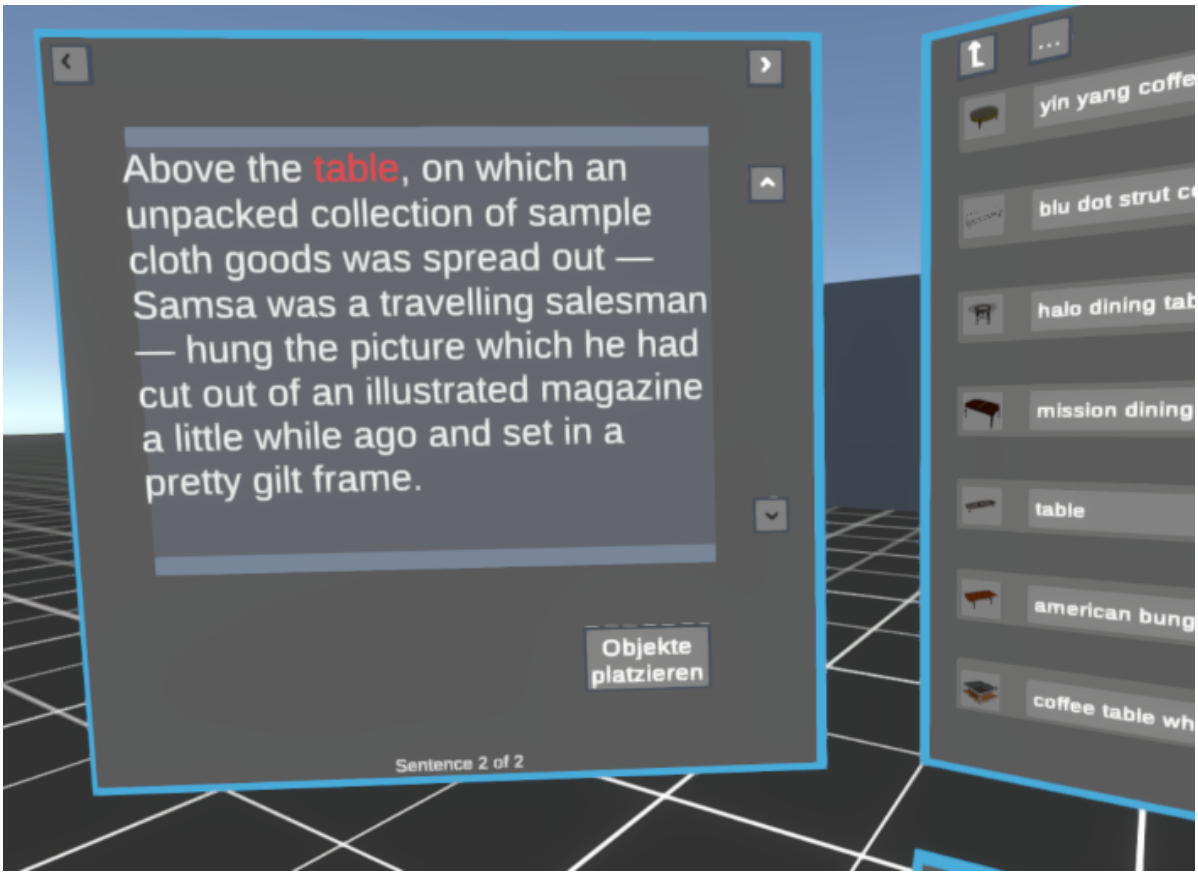

Abbildung 5.8.: Der DataBrowser mit dem geöffneten Text2Scene Interface (Abrami, Henlein u. a. 2020a)

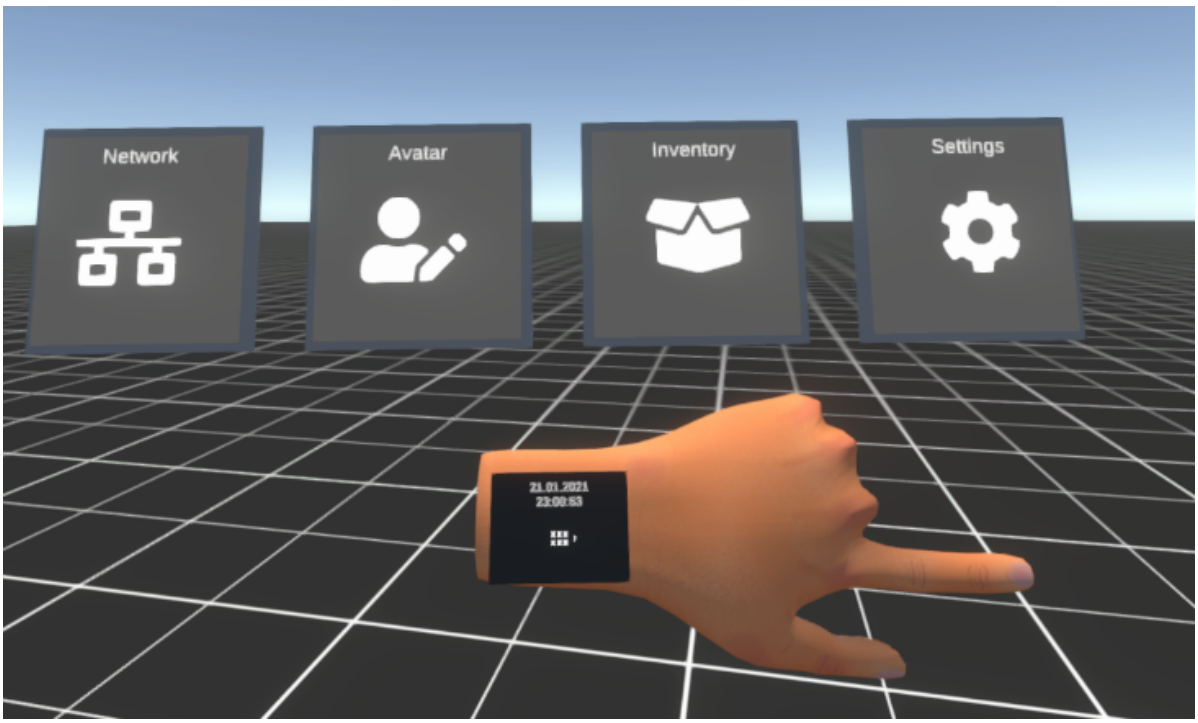

Abbildung 5.9.: Smartwatch des Users (Spiekermann, Abrami und Mehler 2018, vgl.)

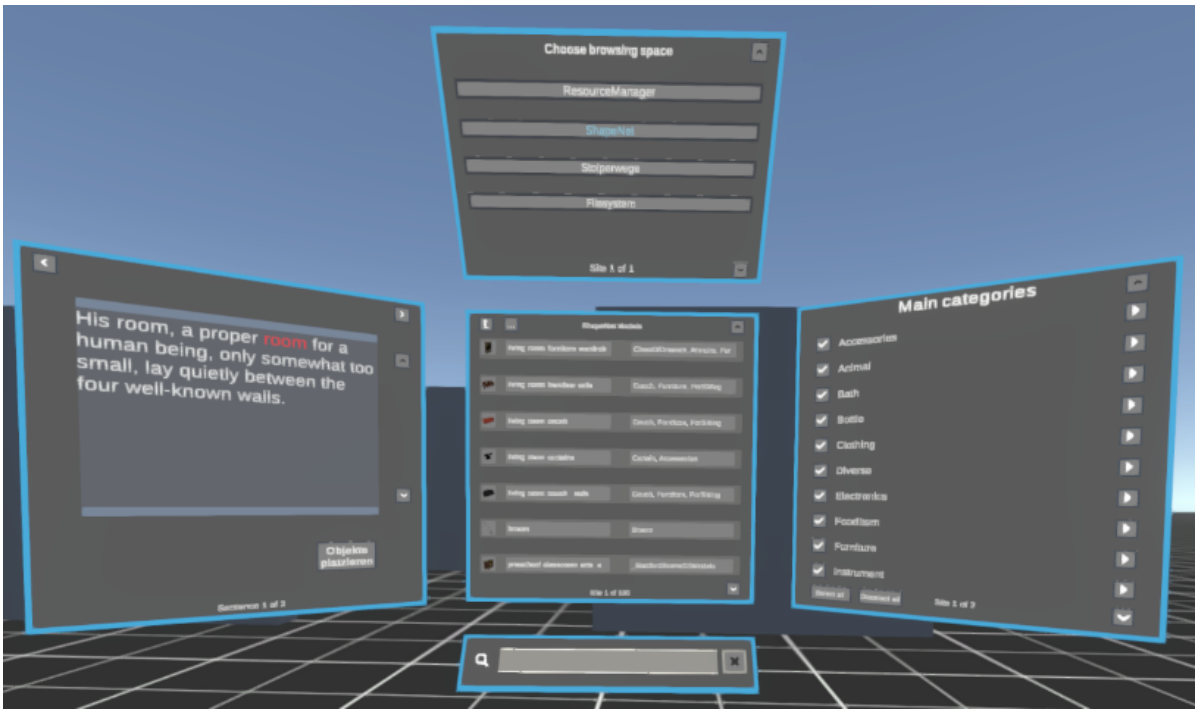

Abbildung 5.10.: Übersicht über den gesamten DataBrowser mit dem Text2Scene, dem ShapeNet Interface, sowie weiteren Einstellungsmenüs (Abrami, Henlein u. a. 2020a)

5.4. Informationsextraktion

Wie bereits in Abschnitt 4.1 erwähnt, ist die grundlegende Pipeline so aufgebaut, dass ein Text eingelesen und dann an den TextImager geschickt wird. Die Anfrage, die per GET-Befehl an die REST-Schnittstelle des TextImagers gestellt wird (Hemati, Uslu und Mehler 2016), muss die Parameter *document*, *pipeline* und *language* enthalten. Der Text wird an den Parameter *document* übergeben, während die Auswahl an Tools, die der TextImager zur Verarbeitung nutzen soll, zum Parameter *pipeline* gehört. Der Parameter *language* benötigt als Eingabe die ISO-Abkürzung der Sprache des Texts (ISO 2002). Aktuell werden von der REST-Schnittstelle Englisch (en), Deutsch (de) und Latein (la) unterstützt [2]. Falls die Sprache des Texts nicht bekannt ist, gibt es einen weiteren GET-Befehl, der den Text als Parameter übergeben bekommt und als Antwort die Sprache ausgibt. Dies ist zum aktuellen Zeitpunkt für die Anwendung aus dieser Arbeit aber nicht relevant, da die anderen Anwendungen nur mit englischsprachigen Texten arbeiten können. Trotzdem wurde bereits die Anfrage in der Applikation eingebaut, sodass man in Zukunft die Sprache dynamisch setzen kann, sofern die restlichen Tools ebenfalls weitere Sprachen unterstützen.

Die Tools, die aktuell verwendet werden, sind einerseits der *CoreRefPosTagger* von Qi u. a. (2018) und andererseits der selbst entwickelte *ObjectTagger*, der in Abschnitt 5.5 vorgestellt wird. Der TextImager bietet zwar eine Reihe an POS-Taggern an, allerdings wurde dem User nicht die Möglichkeit gegeben sich ein Tool auszusuchen. Dies hat den Hintergrund, dass die Applikation größtenteils automatisch ablaufen soll, sobald ein Text eingegeben oder geladen wurde. Daher wurde die Option zum Auswählen der verschiedenen Tagger, insbesondere des POS-Taggers weggelassen. Für Weiterentwicklungen kann der POS-Tagger allerdings in der Konfiguration in Unity vor dem Ausführen des Programms abgeändert werden.

5.5. ObjectTagger

Zur Bestimmung von Objekten, wurde eine Anwendung entwickelt, die als Erweiterung in den TextImager eingebunden werden kann und verschiedene Informationen aus einem Text extrahiert. Dies umfasst einerseits, ob ein Wort ein Objekt beschreibt und andererseits gibt es eine genauere Beschreibung dieses Objekts an.

[2] siehe dazu: `https://textimager.hucompute.org/rest/doku/#/`, abgerufen 06.02.2021

Die Extraktion, ob ein Wort ein Objekt beschreibt, ist schwierig, da es keine lexikalischen Klassifikationen dafür gibt. Andere Probleme lassen sich aus grammatikalischen Regeln ableiten, da ein Satz immer ein Subjekt und ein Prädikat enthalten muss. Da aber ein Objekt nur aus der Bedeutung des Wortes hervorgeht, gibt es keine Regeln, die man aufstellen kann und anhand dieser dann überprüft wird, ob das Wort ein Objekt beschreibt oder nicht. Die einzige Beschränkung ist die Klassifizierung als Nomen. Daher wird im ersten Schritt des ObjectTaggers die Wortart jedes Wortes abgefragt, wobei dann in den folgenden Schritten die potenziellen Gegenstände nur Nomen sind. Sobald der Text mit einem POS-Tagger klassifiziert wurde, wird die Liste an Wörtern, die eventuell Gegenstände beschreiben, nach Objekten gefiltert. Dabei wurde das englische WordNet zu Hilfe genommen (Miller 1998). Es bietet die Möglichkeit, die Hypernym-Beziehungen für das Wort abzufragen. Diese Beziehungen sind wie ein Baum in WordNet aufgebaut, wobei dieser bis zur Wurzel durchgegangen werden kann. Eine Kategorie, die genutzt wird, ist *object, physical object*. Diese wird in WordNet folgendermaßen definiert: „a tangible and visible entity; an entity that can cast a shadow;“ (Miller 1998, vgl.). Sobald diese Kategorie im Baum auf dem Weg zur Wurzel gefunden wird, wird das Wort als Objekt markiert. Alle anderen Nomen werden verworfen und nicht weiter betrachtet.

Beim Testen des ObjectTaggers zeigte sich allerdings, dass wesentlich mehr Wörter markiert werden, als ursprünglich vorgesehen war. Oft handelte es sich um Wörter wie „sister“ oder „father“. Diese werden ebenfalls markiert, weil sie auch in die Gruppe der Objekte fallen. Da sich diese Arbeit jedoch auf nicht lebende 3D-Objekte konzentriert, wurde die Abfrage der WordNet-Datenbank so verfeinert, dass diese Wörter ignoriert werden. Dadurch, dass WordNet eine weitere Kategorie *living object* bei lebenden Objekten hinzufügt, kann man die Abfrage dahingehend modifizieren, dass nur Objekte, die nicht unter diese Kategorie fallen, markiert werden.

Der zweite Teil des ObjectTaggers ermöglicht es, Objekte genauer zu bestimmen. Wenn im Text von einem *Tisch* die Rede ist, weiß das Programm nicht, um was für eine Art des Tisches es sich handelt. Dadurch würden die Ergebnisse, die ShapeNet ausgibt, sehr unkonkret werden. Je nach Kontext kann ein Esstisch oder ein Kaffeetisch gemeint sein. Beide Arten sind in der Shapenet-Datenbank verfügbar, allerdings unterscheiden sich die Modelle stark voneinander. Wenn die Suchwörter spezifischer sind, kann ShapeNet die entsprechenden Modelle höher ranken und als erste Suchergebnisse vorschlagen.

Die Umsetzung wurde mit Hilfe von BERT beziehungsweise DistilBERT vorgenommen. BERT bietet die Möglichkeit Maskierungen im Text mit Token zu füllen. Diese sind abhängig vom Kontext und können gezielt im Text gesetzt werden. In dieser Arbeit wurde so vorgegangen, dass vor jedes Objekt ein „[MASK]“-Token gesetzt wurde. Da DistilBERT eine Beschränkung der Eingabe auf 512 Token aufweist, müssen lange Texte aufgeteilt werden und BERT ersetzt dann die Masken in den einzelnen Textabschnitten. Die Rückgabe ist eine Liste an Maskierungen, wobei diese noch gefiltert werden müssen. Teilweise ersetzt BERT die Masken mit '#' oder nur einem Buchstaben. Da dies keine sinnvollen Präfixe sind, die das Objekt genauer spezifizieren, werden die Masken leer gelassen und es wird keine Maskierung angegeben.

Damit der ObjectTagger genauere Ergebnisse liefert, wurde das verwendete Modell von DistilBERT weiter trainiert, um bessere Präfixe für die Objekte zu finden. Der Vorteil bei BERT liegt darin, dass das Feintuning sehr schnell abgeschlossen ist (Wolf u. a. 2020, vgl.), da man nicht mehr das ganze Modell von Grund auf trainieren, sondern nur auf sein Problem anpassen muss.

Als Trainingsgrundlage wurde die englische Version von Kafkas „Die Verwandlung“ genommen. Der Text wurde dahingehend modifiziert, dass die Wörter, die Objekte darstellen, ein Wort als Präfix erhalten. Dieses Präfix wurde manuell annotiert und ist ein Wort, dass in den Kontext passt und das Objekt vom Aussehen genauer beschreibt. Die benutzten Wörter können das Objekt auf verschiedene Arten veranschaulichen. Am Beispiel des Objekts „Tür“ lässt sich dies gut demonstrieren. Man kann eine Tür zum Beispiel nach dem Aussehen kategorisieren. Dabei werden beispielsweise Wörter wie „wooden“ oder „sliding“ verwendet. Man kann aber auch die Raumzugehörigkeit spezifizieren, da je nach Raum unterschiedliche Modelle in ShapeNet geladen werden können. Eine Haustür sieht anders aus als eine Küchen- oder Badezimmertür. Die dritte Variante, die verwendet wurde, ist eine Beschreibung des Zustands, in dem sich die Tür befindet. Die Tür kann geöffnet oder geschlossen sein. Je nachdem kann ein anderes Türmodell in die Szene geladen werden und gibt dem Raum dementsprechend ein anderes Flair. Bei der Annotation wurde versucht nicht immer nur eine Beschreibungsart zu nutzen, sondern zwischen diesen abzuwechseln. Dies war nicht immer möglich, da manche Passagen gewisse Beschreibungen schon im Kontext sehr genau vorgeben und der wichtigste Gesichtspunkt war, den Kontext nicht zu verfälschen.

5.5.1. Integration in den TextImager

Bisher ist der ObjectTagger ein eigenständiges Programm, das als Eingabe einen Text benötigt, der dann verarbeitet wird. Die Rückgabe, die der User nach dem Ausführen erhält, ist dann einerseits eine Liste an Masken, die durch Wörter ersetzt wurden. Andererseits wird der verarbeitete Text, der zusätzlich um die Masken beziehungsweise die Wörter, die der ObjectTagger eingesetzt hat, ausgegeben, sodass ihn der User manuell weiterverarbeiten kann. Dies ist in dem Konstrukt an Tools wie TextImager und VAnnotatoR nicht praktikabel, da diese einen hohen Grad an Automatisierung bieten. Daher wurde eine Schnittstelle geschaffen, die es ermöglicht, den ObjectTagger direkt vom TextImager aus aufzurufen und die Ergebnisse, die zurückkommen, weiterzuverarbeiten. Der in Java entwickelte TextImager, bietet eine grundlegende Schnittstelle an, die Python-Skripte ausführen und deren Ausgaben im Java-Programm verarbeiten kann. Die Rückgabe, die aus einer Liste von (Präfix, Basiswort) Paaren besteht, muss so konvertiert werden, dass sie in die Datenstruktur des TextImagers hinzugefügt werden kann.

Der TextImager nutzt grundsätzlich XMI als Dateiformat, welches zu den *Markup Languages* gehört. Die XMI-Datei enthält die Ergebnisse der Verarbeitung durch den TextImager. Jeder Eintrag in der XMI beschreibt ein Objekt, welches als Attribut immer eine ID zur genauen Identifizierung beinhalten muss. Die wichtigste Klasse an Objekten, die der TextImager enthält, sind die Token, da auf sie am häufigsten verwiesen wird. Im TextImager sind bereits verschiedenste Klassen zur Wortverarbeitung vorhanden. Dabei eignen sich vor allem zwei Klassen zur Speicherung der Daten vom ObjectTagger.

Die erste davon ist die Klasse *IsoSpatialEntity* (ISO 2014; ISO 2020), welche ISO genormt ist und wie der Name schon sagt, physische Objekte beschreibt. Neben Gegenständen, die in dieser Arbeit wichtig sind, können damit aber auch Personen oder Orte beschrieben werden. Die wichtigsten Anpassungen gegenüber den Basisklassen, von denen diese Klasse die grundlegenden Strukturen erbt, sind die Beschreibungen des Objekts im Raum. Dabei ist der Ort durch Koordinaten und einer Höhe über NN dargestellt. Drei weitere Attribute halten die Position, Rotation und Skalierung des Objekts fest. Wenn der TextImager die Daten vom ObjectTagger erhält, erstellt er automatisch für jedes Objekt, das klassifiziert wurde, eine *IsoSpatialEntity*. Es kann noch nicht die Informationen des Objekts zur Lage im Raum enthalten. Daher werden diese, wie auch die meisten anderen Felder, erst mal leer gelassen. Sie werden erst im VAnnotatoR mit Daten befüllt, da vorher das 3D-Objekt noch nicht vorhanden ist. Im TextImager enthalten die *IsoSpatialEntity* Objekte als Attribute nur eine

ID, das Wort als Kommentar und Anfangs- und Endposition des Wortes im Text, sodass eine Referenz zum Text besteht.

Die zweite relevante Klasse, die der TextImager mit den Daten des ObjectTaggers befüllt, sind die *Meta Links*. Sie enthalten Informationen, die mit Objekten des Typs *IsoEntity* verknüpft werden sollen. Die Klasse *IsoEntity* bildet die Basis für die verschiedenen ISO genormten Entitäten, wie auch die *IsoSpatialEntity* Klasse. Die *Meta Links* werden genutzt, um die Präfixe, die der ObjectTagger ausgewählt hat, mit den Objekten zu verknüpfen. Dabei werden als Attribute das Wort, die zum Basiswort zugehörige *IsoSpatialEntity* und ein Relationstyp gesetzt. Der Relationstyp ist wichtig, da es *Meta Links* mit unterschiedlichen Relationen in der XMI geben kann. In diesem Fall ist der Typ „Mask“, welcher für alle Links gesetzt wird, die eine Maskierung im Text ersetzen. Sobald alle Objekte erstellt wurden, werden sie zu der XMI-Datei des aktuellen Texts durch den TextImager hinzugefügt und können dann unter anderem durch den VAnnotatoR genutzt werden.

5.6. Erkennen von Teil-Ganz Relationen

Wie bereits in Abschnitt 1.1 erwähnt, ist ein zentraler Bestandteil der Arbeit die Erkennung von Teil-Ganz Relationen. Um diese gut erkennen zu können, wurde ein Neuronales Netz trainiert, das das entsprechende Wort als Ergebnis ausgibt. Die Teil-Ganz Relationen wurden aber nicht in beide Richtungen umgesetzt, sondern nur die Erkennung des Ganzen betrachtet. Dies ergibt sich daraus, dass die Beziehung vom Ganzen zum Kleineren für das Ergebnis nicht relevant ist. Da das Ziel die Extraktion von Objekten aus einem Text ist, ist es relevant ein passendes Objekt aus der ShapeNet-Datenbank zu suchen (Chang u. a. 2015). Dafür muss beispielsweise erkannt werden, dass es sich bei einer Rückenlehne um einen Stuhl handelt, der gesucht werden muss. Andersherum ist es aber nicht von Bedeutung aus welchen Teilen der Stuhl besteht, da nur fertige Objekte in die Szene geladen werden.

5.6.1. Aufbau des Neuronalen Netzes

Zur Erkennung von Holonymen, wurde ein *Feed-Forward Network* eingesetzt. Das Netz wurde von Unity Technologies entwickelt und lässt sich somit problemlos in die VR-Plattform Unity einbinden (Juliani u. a. 2020). Dadurch, dass die Holonym Beziehung nur vom Objekt und nicht von anderen Faktoren abhängig ist, wird lediglich das Objekt als Input für das Neuronale Netz genutzt. Da der Input aber nur aus einer Reihe an Zahlen bestehen darf, muss der String, der das Wort enthält zu einer Zahl beziehungsweise zu einem Vektor transformiert

werden. Hierfür wurde GloVe (Pennington, Socher und Manning 2014) verwendet. Die Autoren stellen bereits verschiedene Word-Vektoren Datenbanken als Textdatei bereit, wobei in diesem Projekt die Daten aus Wikipedia (2014) und Gigaword 5 verwendet wurden[3]. Der Datensatz enthält ca. 400.000 Wörter, die als Vektoren vorliegen.

Die Datensätze werden beim Programmstart geladen, sodass die Verknüpfung zwischen Wort und Vektor global im VAnnotatoR verfügbar ist. Unity und C# stellen aktuell zwar Klassen für Vektoren bereit, allerdings beschränken sich diese auf zwei- und dreidimensionale Vektoren. Da die in GloVe enthaltenen Vektoren eine Dimension von 50 haben, musste dafür eine Klasse geschrieben werden, die die zentralen Eigenschaften eines Vektors beinhaltet und einige Funktionen unterstützt, die benötigt werden (siehe Abbildung 5.11). Die zentralen Punkte, die verfügbar sein müssen, sind neben dem Vektor auch seine Länge und Dimension. Diese werden vor allem beim Überprüfen der Ausgabe des Neuronalen Netzes wichtig.

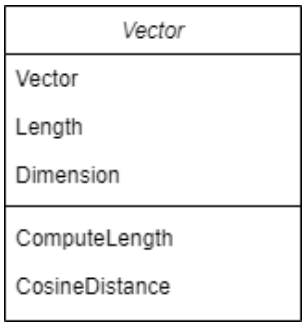

Abbildung 5.11.: Aufbau der Datenstruktur für Vektoren, welche unabhängig von der Dimension des Vektors ist

Das Neuronale Netz braucht bei einer Eingabe von einem Wort folglich 50 Inputneuronen, wobei jedes Neuron eine Stelle des Vektors als Eingabe erhält. Die Anzahl der Outputneuronen des Netzes ist dementsprechend auch 50, da hier ebenfalls ein Wort als Vektor dargestellt wird. Es gibt drei Hiddenlayer mit jeweils 512 Neuronen.

Als Neuronales Netz wird ML-Agents (Juliani u. a. 2020) von Unity genutzt, welches direkt in die VR-Umgebung eingebunden werden kann. ML-Agents ist allerdings keine reine Unity Bibliothek, sondern baut auf Python auf und basiert unter anderem auf Googles Tensorflow Bibliothek (Martín Abadi u. a. 2015). Wie in Abbildung 5.12 zu sehen ist, dienen Agenten als Schnittstelle zwischen dem Behavior (Neuronales Netz) und der restlichen Applikation.

[3] Übersicht über die verfügbaren Datensätze: `https://nlp.stanford.edu/projects/glove/`; abgerufen 27.01.2021

Die Agenten sammeln zum einen den Input für das Netz und zum anderen geben sie die Entscheidung aus, die danach weiterverarbeitet werden kann. Dieser Austausch findet in jedem Updateschritt vom VAnnotatoR statt. Aus der Ausgabe des Neuronalen Netzes wird ein Vektor erstellt, der anschließend wiederum mit der Vektor-Wort Datenbank von GloVe verglichen wird. Als Grundlage für den Vergleich jeweils zweier Vektoren wird die Kosinus Distanz verwendet, da sie genormte Werte zwischen -1 und 1 ausgibt und somit eine gute Vergleichbarkeit bietet. Sie ist folgendermaßen definiert:

$$Kosinus\ Distanz: \quad \cos(\alpha) = \frac{\vec{x} \cdot \vec{y}}{|\vec{x}| \cdot |\vec{y}|}$$

Je ähnlicher sich folglich zwei Vektoren sind, umso näher ist der Wert des Kosinus an 1. Der vom Netz ausgegebene Vektor wird nun mit jedem Vektor aus der GloVe-Datenbank verglichen und es wird der Vektor in der Datenbank gesucht, bei dem der Vergleich 1 am nächsten ist. Dieses Wort wird übernommen und als Holonym gesetzt. Damit die Performance des VAnnotatoRs nicht unter den Vergleichen leidet, wird die Überprüfung der Ausgabe in asynchronen Funktionen ausgeführt und das Ergebnis nach der Berechnung bereitgestellt. Dadurch muss zwar der User im Vergleich zur Verwendung von synchronen Funktionen eventuell etwas länger auf ein Ergebnis warten, allerdings wird es auch keine Aussetzer der Anwendung aufgrund von Berechnungen geben, die in der VR-Umgebung störend sind.

5.6.2. Training vom Neuronalen Netz

Um das Neuronale Netz zu trainieren, müssen erst Daten verfügbar sein. Dazu werden elf verschiedene Kategorien betrachtet, unter anderem das Wohnzimmer, Badezimmer, Auto, Computer etc., wobei man die Daten in einer Textdatei sammelt. Da es immer nur eine eins-zu-eins Relation ist, bestehen die Daten aus Tupeln (Teil, Ganzes). Weil das Netz lernen muss, dass ein Stuhl schon der Oberbegriff ist, werden auch (Ganzes, Ganzes) Tupeln hinzugefügt, sodass es diese Objekte ebenfalls richtig zuordnet. Insgesamt gibt es 308 verschiedene Datensätze.

Das Training wurde mit zehn Agenten durchgeführt, die alle Daten an das Netz schicken und eine Antwort vom *Behavior* erwarten. Dadurch kann schneller trainiert werden und der Reward, den die Agenten beim Vergleich mit den vorgegebenen Ergebnissen erhalten und über die Zeit maximieren möchten, wächst schneller. Dabei ist die Rewardfunktion so aufgebaut, dass sie falsche Ergebnisse bestraft und bessere Distanzwerte belohnt (siehe Algorithmus 1). Für das Training wurde eine *Learning Rate* von 0.0003 eingestellt, da so eine konstante Ver-

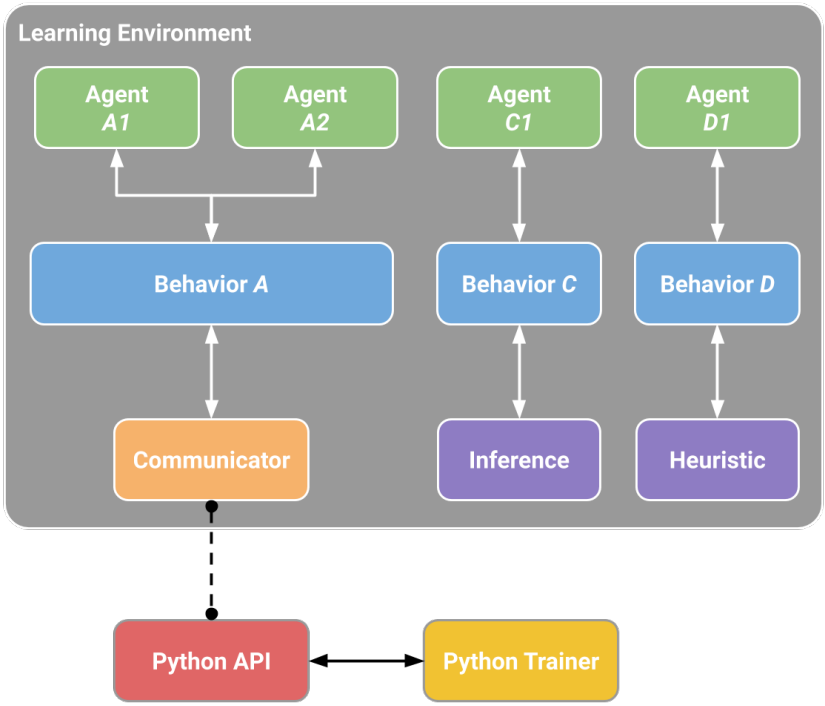

Abbildung 5.12.: Grundlegender Aufbau der Bibliothek ML-Agents (Unity Technologies 2020)

besserung des Rewards der Agenten auf abschließend 0.9 von maximal 1 erreicht wird. Dadurch ließen sich bereits gute Ergebnisse erzielen. Da es mit 308 Datensätzen allerdings nur wenig Daten gibt, wurde Semisupervised Learning (Abschnitt 2.3) angewendet, um die Anzahl an Datensätzen zu erhöhen. Dazu braucht man neue größere Datenbanken, die eine Sammlung an Objekten beinhalten. Aktuell gibt es aber kaum reine Objektdatenbanken, weswegen die ShapeNet-Datenbank als Grundlage benutzt wird (Chang u. a. 2015), da sie in den Metadaten für jedes Objekt auch einen Namen und Tags für die Objekte enthält. Diese Daten wurden ausgelesen und dadurch ließen sich die Tupel an Objekt und Hypernym aufbauen.

Mit diesem neuen Datensatz, der ungefähr 2500 Objekte enthält, wurde das Netz weiter trainiert, sodass die Anzahl an Objekten, die das Netz zuordnen kann, gestiegen ist.

Hierfür mussten zusätzlich die Trainingsparameter etwas angepasst werden, da ansonsten ab einem bestimmten Punkt kein Trainingsfortschritt zu sehen war, obwohl das Neuronale Netz durchschnittlich nur einen Reward von 0.3 erzielte. Durch die Anpassungen unter anderem von der Learning Rate wird im Training ein durchschnittlicher Reward von 0.82 erzielt.

```
1: Vector resultVector = new Vector(vectorAction)
2: double cosineDistance = resultVector.CosineDistance(expectedResult)
3: if cosineDistance > 0.9 then
4:     AddReward(1.0f);
5: else if cosineDistance > 0.8 then
6:     AddReward(0.9f);
7: else if cosineDistance > 0.75 then
8:     AddReward(0.88f);
9: else if cosineDistance > 0.7 then
10:    AddReward(0.7f);
11: else if cosineDistance > 0.6 then
12:    AddReward(0.6f);
13: else if cosineDistance > 0.5 then
14:    AddReward(0.5f);
15: else if cosineDistance > 0.4 then
16:    AddReward(0.3f);
17: else if cosineDistance > 0.2 then
18:    AddReward(0.2f);
19: else if cosineDistance > 0 then
20:    AddReward(0.1f);
21: else if cosineDistance > −0.2 then
22:    AddReward(−0.1f);
23: else
24:    AddReward(−1.0f);
25: end if
```

Algorithmus 1 : Er beschreibt die Aufteilung des Rewards, unter Berücksichtigung der Kosinusdistanz

5.7. Erkennen des Kontexts

Die Erkennung des Kontexts basiert auf dem gleichen grundlegenden Ablauf wie die Bestimmung von Holonymen (siehe Abschnitt 5.6). Hier wird ebenfalls ein Neuronales Netz verwendet, welches allerdings nicht das Holonym ausgibt, sondern ein Wort, das den Kontext beschreibt. Dabei ist es wichtig zu wissen, dass sich der Kontext ausschließlich auf den Ort, an dem die Szene stattfindet, bezieht. Andere mögliche Beschreibungen werden in dieser Arbeit nicht betrachtet.

5.7.1. Aufbau des Neuronalen Netzes

Um die Szene zu erfassen, muss man anders als bei der Bestimmung von Teil-Ganz Relationen nicht nur ein Wort, sondern einen größeren Bereich betrachten. Für jedes Wort, das ein Objekt beschreibt, wird der Kontext festgesetzt. Dafür werden zusätzlich Textstellen links und rechts von diesem Wort in das Neuronale Netz eingegeben. Die Kontextbestimmung bezieht dabei aber nicht alle Wörter in einem Bereich mit ein, sondern nur Objekte, die durch den ObjectTagger (siehe Abschnitt 5.5) als physische Objekte klassifiziert wurden. Konkret werden an das Neuronale Netz fünf Wörter übergeben. Neben dem aktuell betrachteten Wort, werden jeweils die beiden Objekte links und rechts davon miteinbezogen. Diese werden nach demselben Prinzip wie in Unterabschnitt 5.6.1 als Vektoren dargestellt. Die fünf Vektoren werden dann an das Neuronale Netz weitergegeben, welches aufgrund der Dimension der Vektoren 250 Inputneuronen besitzt. Aus der Rückgabe der 50 Outputneuronen wird wiederum ein Vektor erstellt, der mithilfe der Kosinusdistanz in einen String und damit zu einem Wort konvertiert wird.

5.7.2. Trainieren vom Netz

Das Neuronale Netz wird auch hier in zwei Schritten trainiert. Der erste Schritt ist ein Training mit reinem *Supervised Learning*, welches eine Grundlage schafft und dann auf einen größeren Bereich ausgeweitet wird. Für das *Supervised Learning* werden die gleichen Kategorien und Wörter als Grundlage gewählt wie bereits für das andere Training. Der Unterschied liegt allerdings darin, wie die Trainingsdaten aufgebaut sind. Sie bilden keine eins-zu-eins Beziehung wie in Unterabschnitt 5.6.2, sondern enthalten einen Oberbegriff, dem verschiedene im Raum enthaltene Objekte zugeordnet werden. Die Daten bilden daher die Struktur (Kategorie, Begriff 1, Begriff 2, ...). Es gibt insgesamt 25 verschiedene Kategorien, mit denen das Neuronale Netz trainiert, wobei sie hauptsächlich Wohnräume darstellen.

Genauso wie beim Training der Teil-Ganz Relationen wird eine *Learning Rate* von 0.0003 in Verbindung mit mehreren gleichzeitig trainierenden Agenten verwendet, da sich der Reward ebenfalls konstant verbessert. Der Unterschied zwischen beiden Trainings besteht allerdings in der Rewardfunktion. Während sie bei der Holonymbestimmung in kleine Schritte unterteilt ist, ist sie in diesem Fall gröber, da ein stärkerer Bezug auf der Erkennung des richtigen Kontexts liegt. Daraus folgt, dass falsche Antworten stärker bestraft werden.

Das trainierte Neuronale Netz wird dann zur Vermehrung der Daten genutzt. Hierfür dient ebenfalls die ShapeNet-Datenbank als Grundlage, aus welcher die Tags der enthaltenen Objekte in die vom Netz ausgegebene Kategorie eingeordnet werden. Dies erzeugt insgesamt 3840 Datensätze, die daraufhin als gelabelte Daten für das Training des Netzes zur Verfügung stehen. Durch das *Semisupervised Learning* können nun mehr Objekte dem Kontext zugeordnet werden.

5.8. Machine Learning in der VR

Für den User bleibt die Verarbeitung durch die verschiedenen Systeme (Abbildung 4.2) größtenteils verborgen, da er als Ergebnis nur die Objekte beziehungsweise den mit Informationen angereicherten Text sieht. Um die Verarbeitung in der VR ebenfalls sichtbar zu machen, wurde ein Objekt, dass in verschiedenen Kontexten existiert als Grundlage für einen Satz in dem Kontext genommen. Hierfür wurde der Tisch genutzt, welcher einmal als Esstisch und andererseits als Schreibtisch kategorisiert werden kann. Dazu wurden die beiden Sätze „A student sits on a table in his office chair with his laptop and writes his thesis.“ und „Oscar cooked his meal in a pan and eats it on a plate at the table.“ verwendet. In Abbildung 5.13 wird sichtbar, dass auf Grundlage des ersten Satzes ein Schreibtisch geladen wird, das System bei einer Eingabe vom zweiten Satz einen Esstisch lädt (siehe Abbildung 5.14).

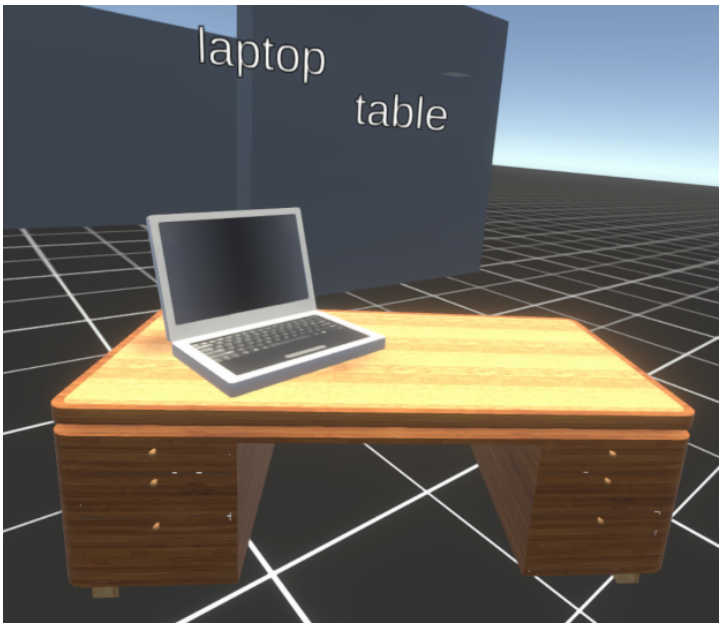

Abbildung 5.13.: Wenn das Objekt Tisch im Kontext Büro und Arbeiten steht, wird ein Schreibtisch geladen. Der Satz dazu lautet: „A student sits on a table in his office chair with his laptop and writes his thesis.“

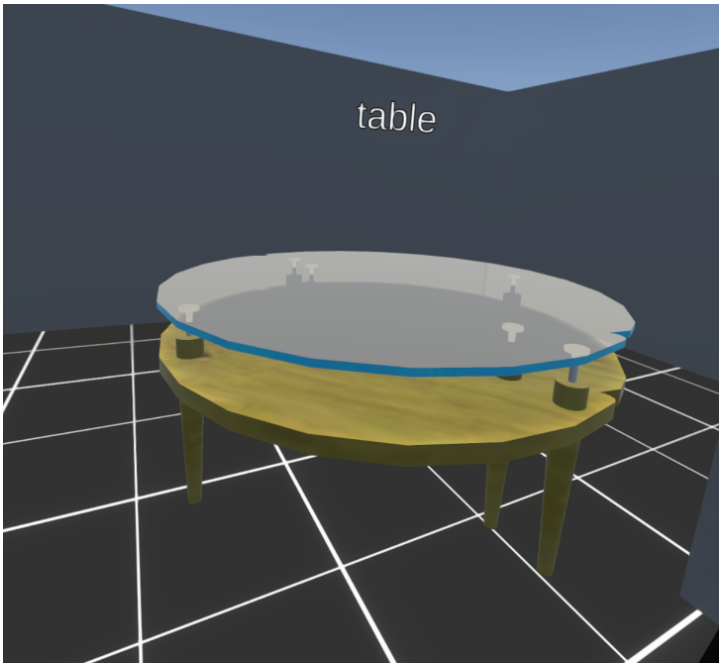

Abbildung 5.14.: Für das Objekt Tisch wird ein Esstisch geladen, wenn der Kontext Kochen oder Essen ist. Der folgende Satz wurde verwendet: „Oscar cooked his meal in a pan and eats it on a plate at the table.“

6. Evaluation

6.1. Evaluation des ObjectTaggers

Zur Überprüfung der Genauigkeit des ObjectTaggers wurde dieser anhand von zwei verschiedenen Merkmalen kurz evaluiert. Erstens wurde überprüft, welche Wörter am häufigsten als Präfixe übernommen wurden und zweitens wurden für vier ausgewählte Präfixe die Wörter herausgesucht, vor die sie vom ObjectTagger gesetzt wurden. Dabei wurde als Grundlage die Erzählung „Die Verwandlung“ von Franz Kafka genutzt. Danach wurden die Ergebnisse des ObjectTaggers mit dem Modell von DistilBERT verglichen, das noch kein Feintuning erhalten hatte.

Bei den häufigsten Präfixen wurden jeweils die 20 am meisten auftretenden Wörter für das DistilBERT Modell (Abbildung 6.1) und den ObjectTagger (Abbildung 6.2) geplottet. Wenn man beide miteinander vergleicht, fällt sofort auf, dass der ObjectTagger deutlich mehr Wörter enthält, die den Raum beschreiben, wie zum Beispiel *wooden, bedroom* oder *front (door)*. In der Standardversion von DistilBERT wird das Modell *distil-bert-uncased* verwendet, welches den Unterschied zwischen Groß- und Kleinschreibung der Wörter nicht beachtet. In Abbildung 6.1 wird offensichtlich, dass die Auswahl sich von der des ObjectTaggers unterscheidet, da sehr viele Wörter, die Personen beschreiben, vorkommen (*younger, elderly, adoptive*). Es gibt zwar auch Präfixe, die den Raum beschreiben, allerdings treten diese wesentlich seltener auf als die Personen beschreibende Wörter. Allein *adoptive* auf Platz eins wurde fast doppelt so häufig von DistilBERT verwendet wie *living* von *living room* auf Platz zwei.
Auch auf den Plätzen 10 bis 20 sieht man noch deutliche Unterschiede bei den genutzten Wörtern. Wie Abbildung 6.2 zeigt, verwendet der ObjectTagger auch einige Farben oder Materialien wie Messing. Dahingegen ersetzt DistilBERT die Maskierungen weiterhin mit einem Mix aus Wörtern, die Objekte und Personen beschreiben.

Der zweite Vergleich betrachtet die Worte, vor die die Präfixe gesetzt werden. Da die Geschichte in der Verwandlung hauptsächlich in einem Raum spielt, wurde der Fokus auf die Präfixe *living, wooden, front* und *bedroom* gelegt. Die Auswahl orientierte sich neben den Räumen, in denen die Geschichte spielt, zusätzlich an den annotierten Wörtern. Dabei wurden

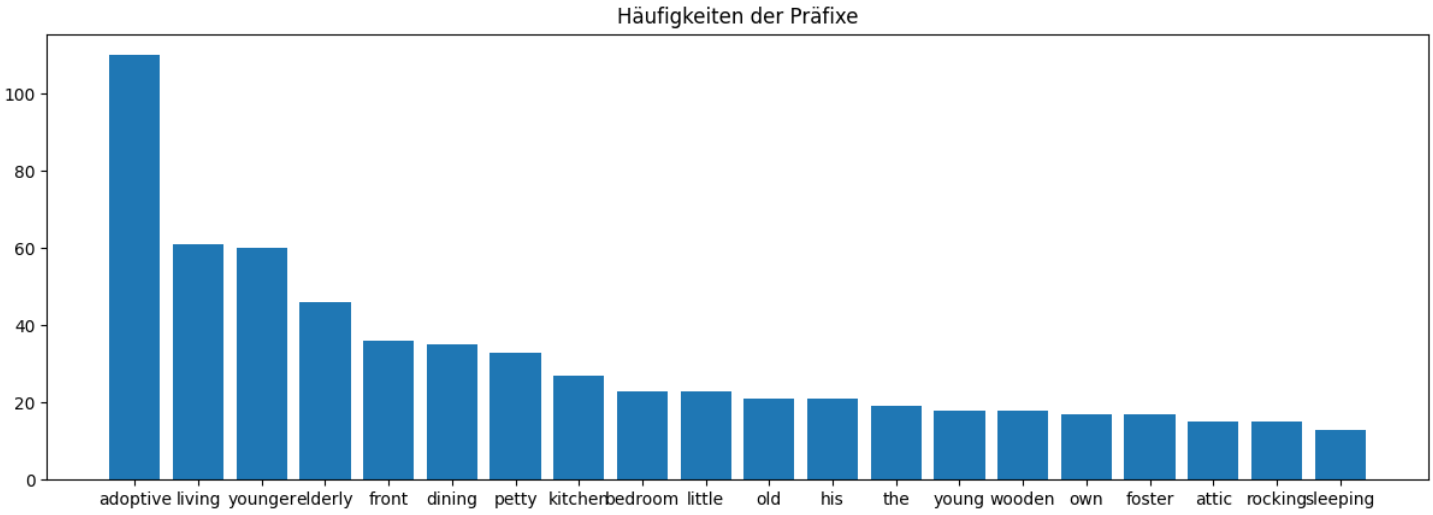

Abbildung 6.1.: Die 20 meistgewählten Präfixe, bei der Nutzung des Standard DistilBERT Modells

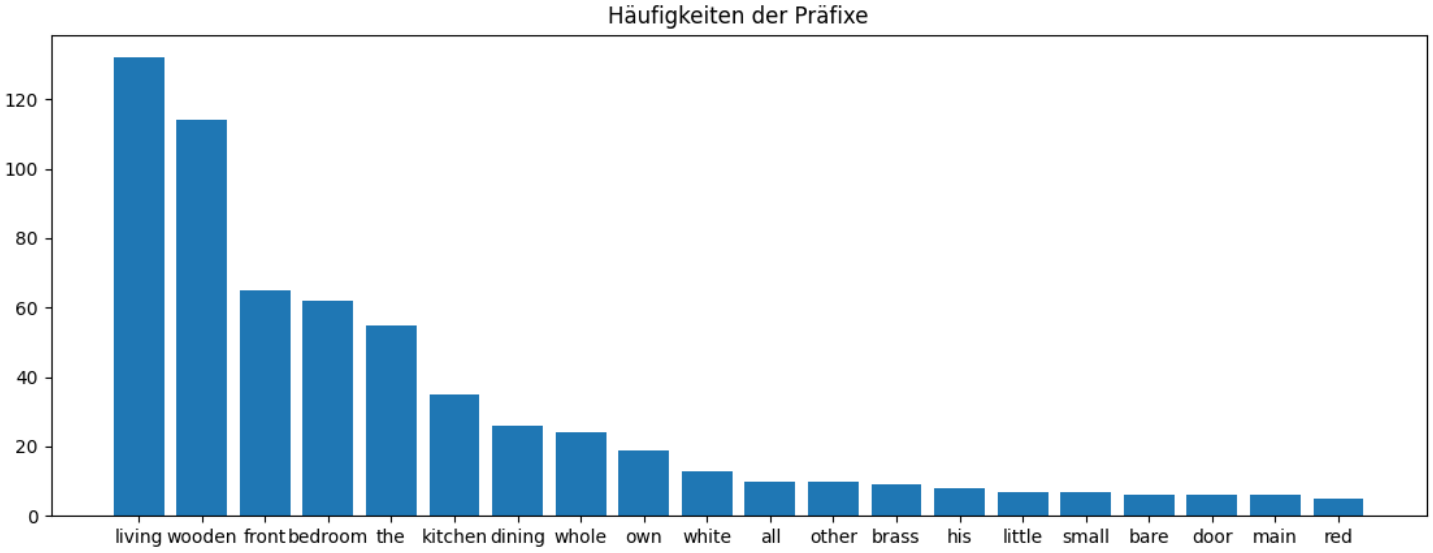

Abbildung 6.2.: Die 20 meistgewählten Präfixe, bei der Nutzung des ObjectTaggers

diese Wörter oft als Präfix gesetzt, was sich ebenfalls an den Häufigkeiten in Abbildung 6.2 widerspiegelt.

Auf den ersten Blick sind keine großen Unterschiede zwischen den Diagrammen festzustellen. Wenn man sich die beiden Diagramme zum Präfix *bedroom* anschaut (Abbildung 6.6 - DistilBERT, Abbildung 6.5 - ObjectTagger) dann wird deutlich, dass *bedroom*, bis auf die Ausnahmen *floor* und *furniture*, immer vor die gleichen Präfixe gesetzt wurde. Dies zeigt, dass DistilBERT bereits ohne ein feineres Training gute Ergebnisse liefert. Der Unterschied zwischen dem fein justierten Modell des ObjectTaggers und dem Standard DistilBERT Modell lässt sich jedoch an zwei verschiedenen Punkten demonstrieren.

Erstens, sind die Reihenfolgen, wann welches Präfix gewählt wird, unterschiedlich. In den Diagrammen sind immer die Top drei Lösungen angegeben, mit denen das [MASK] Token von DistilBERT ersetzt wird. Diese sind nach absteigenden Wahrscheinlichkeiten von BERT sortiert (Wolf u. a. 2020, vgl.). Zwar wird das Präfix *bedroom* vor die gleichen Wörter gesetzt, allerdings lässt sich ein anderes Ranking erkennen. Beim Standard DistilBERT Modell, werden vor allem die Wörter *door*, *window* und *bedroom* als wichtig erachtet (Abbildung 6.6). Nach dem Feinjustieren liegt der Fokus hauptsächlich auf Wörtern mit dem Wortstamm *door*. Man erkennt folglich, dass durch das Feintuning ein anderer Fokus gesetzt wurde. Nach demselben Prinzip lassen sich auch die beiden Diagramme zum Wort *wooden* interpretieren. Nach dem Training (siehe Abbildung 6.9) werden deutlich mehr Möbelstücke mit dem Wort *wooden* in Verbindung gebracht als vorher (siehe Abbildung 6.10). Statt nur *furniture* zu markieren, wurden zusätzlich *couch* und *chair* in dieselbe Gruppe eingeordnet.

Der zweite Punkt, der das Ergebnis des Feintunings deutlich macht, lässt sich sehr gut am Beispiel der Diagramme zu *living* (siehe Abbildung 6.3 und Abbildung 6.4) zeigen. Diesmal sieht man keine Anpassung des Fokus, da *living* auch nach dem Training weiterhin mit dem Wort *room* verknüpft wird. Der Unterschied liegt aber darin, dass die Verbindung zwischen den beiden Wörtern wesentlich häufiger besteht. DistilBERT bringt die beiden Wörter insgesamt 53-mal in Verbindung, während es nach dem Training 113 Verknüpfungen sind.

Dieser Trend ist bei jedem der vier Präfixe zu beobachten. Immer wenn ein Wort bereits oft im Standardmodell von DistilBERT mit dem Präfix verbunden wird, wird es nach dem Training noch häufiger mit dem gleichen Präfix markiert. Man sieht folglich, dass schon ein kurzer Text mit gerade einmal 930 annotierten Maskierungen einen Unterschied bewirkt und

die Klassifizierung genauer macht. Sollten in Zukunft mehr Daten verfügbar sein, folgt daraus, dass man den ObjectTagger noch präziser trainieren kann. Dadurch werden die Präfixe stärker an den Kontext angepasst, wodurch die Suchanfragen an ShapeNet präziser werden.

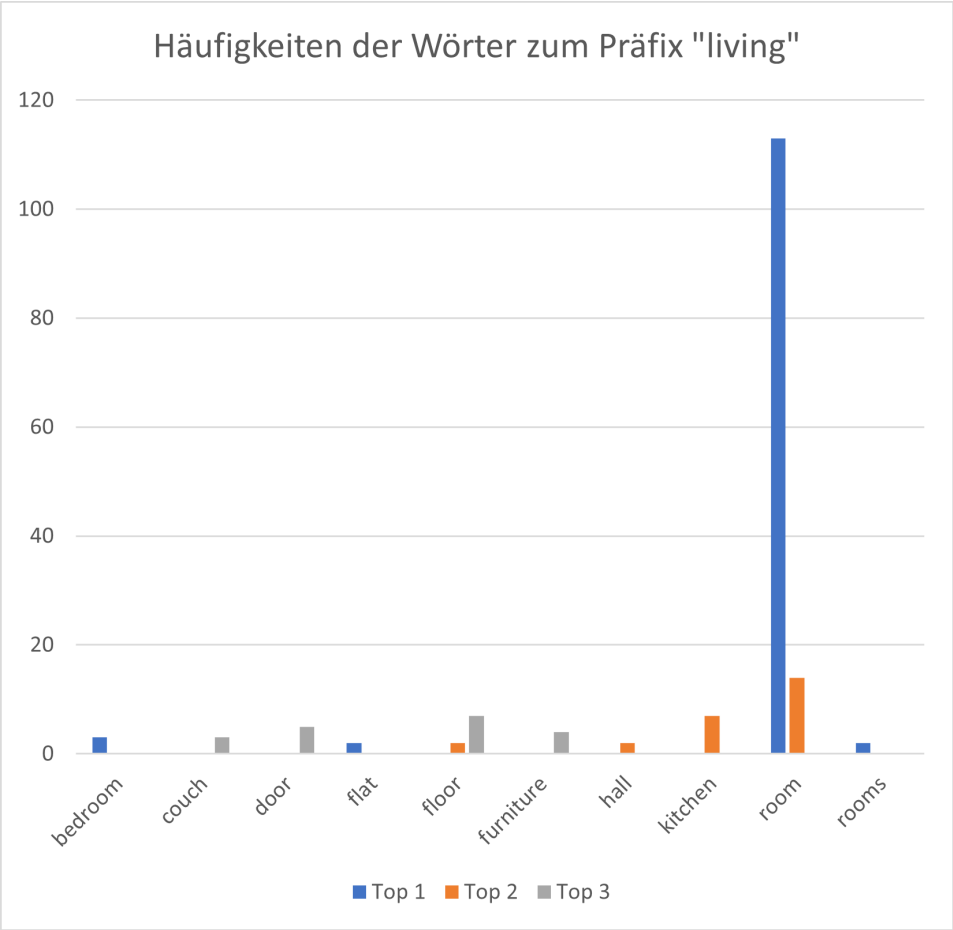

Abbildung 6.3.: Wörter, vor die das Präfix living, vom ObjectTagger gesetzt wurde

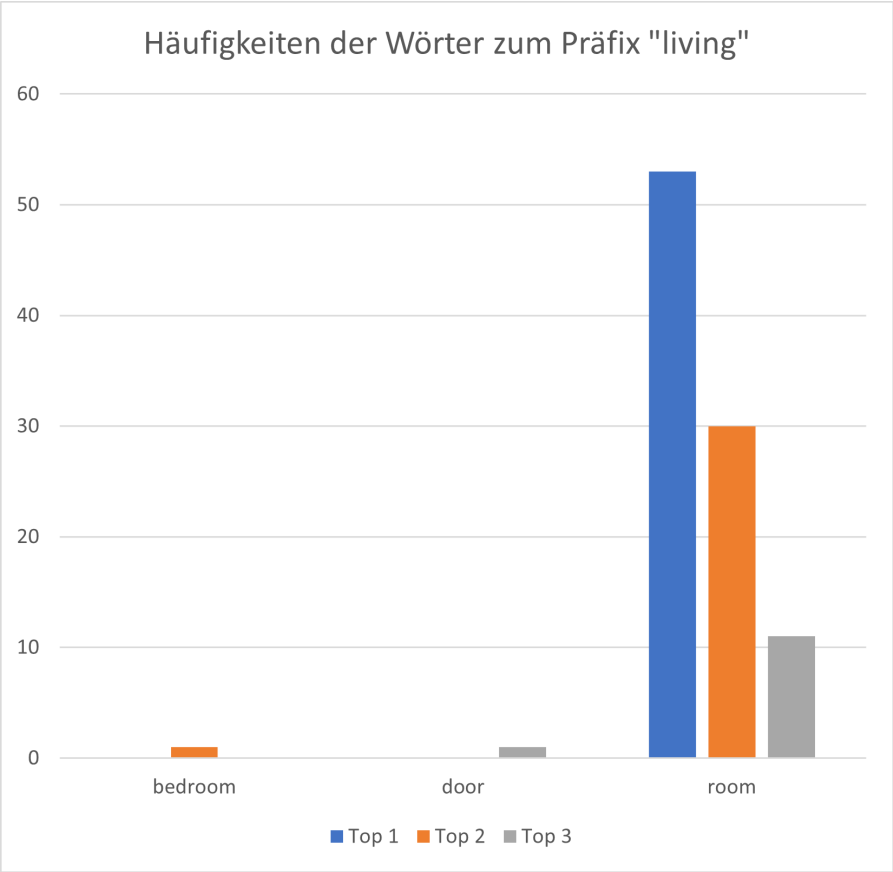

Abbildung 6.4.: Wörter, vor die das Präfix living, von DistilBERT gesetzt wurde

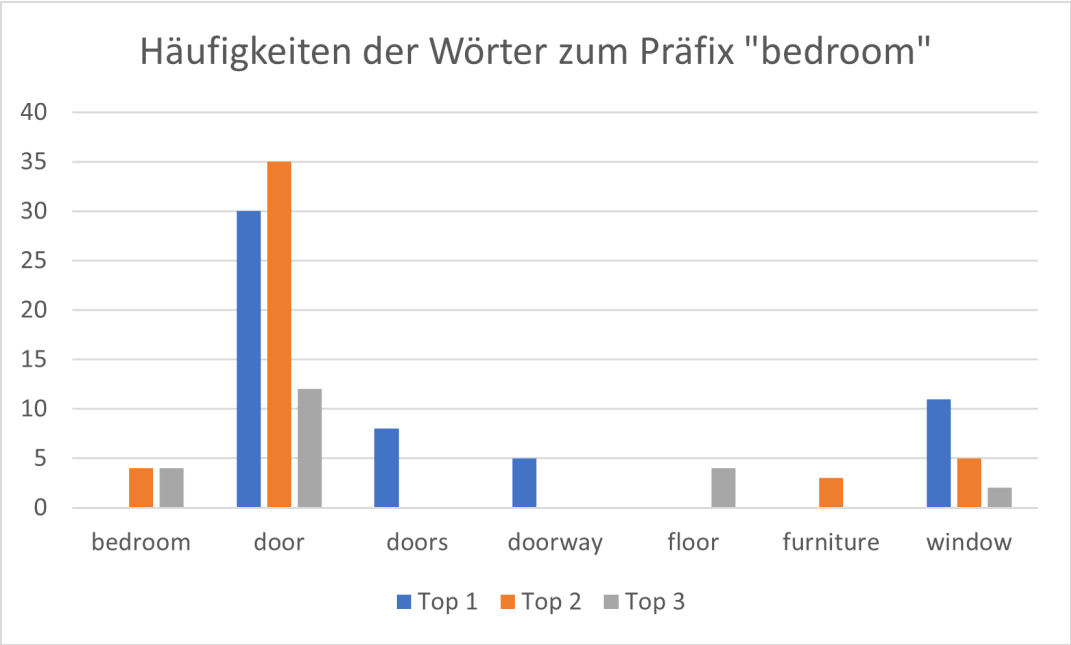

Abbildung 6.5.: Wörter, vor die das Präfix bedroom, vom ObjectTagger gesetzt wurde

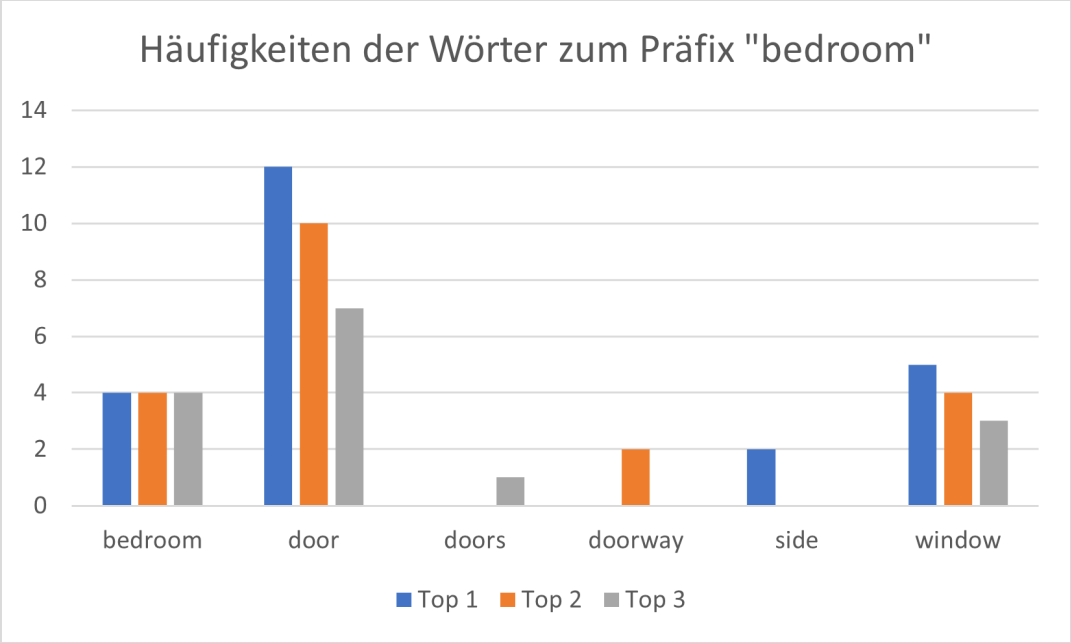

Abbildung 6.6.: Wörter, vor die das Präfix bedroom, von DistilBERT gesetzt wurde

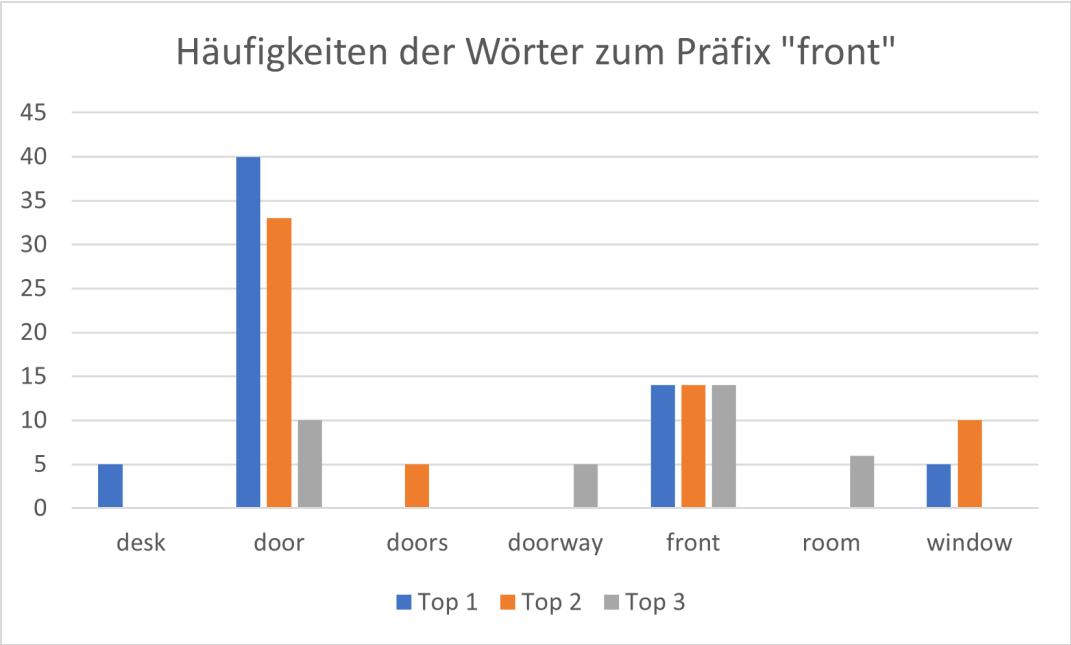

Abbildung 6.7.: Wörter, vor die das Präfix front, vom ObjectTagger gesetzt wurde

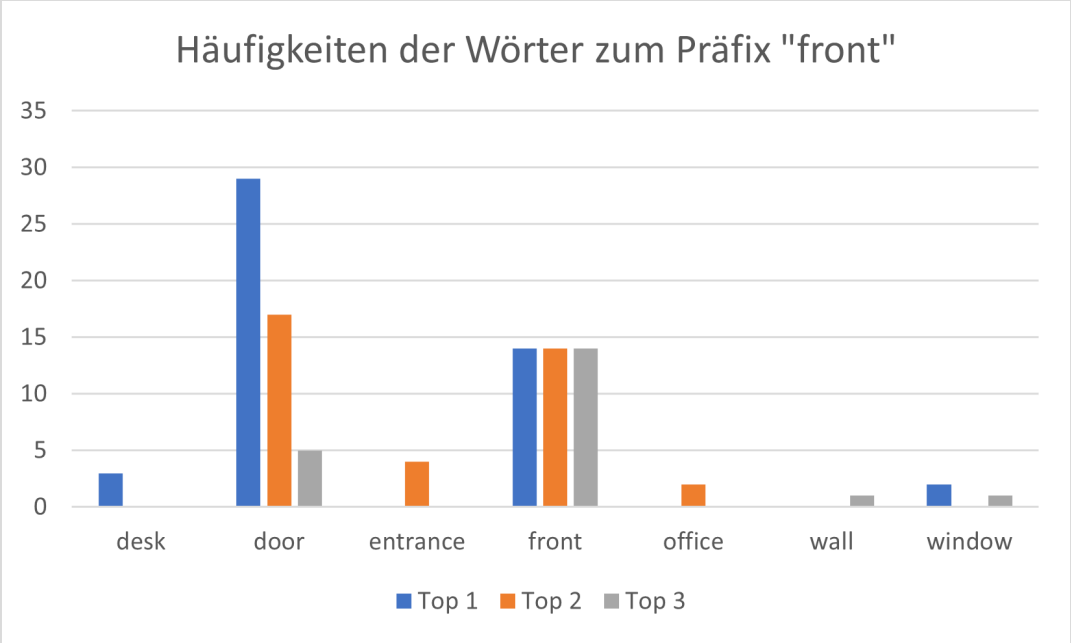

Abbildung 6.8.: Wörter, vor die das Präfix front, von DistilBERT gesetzt wurde

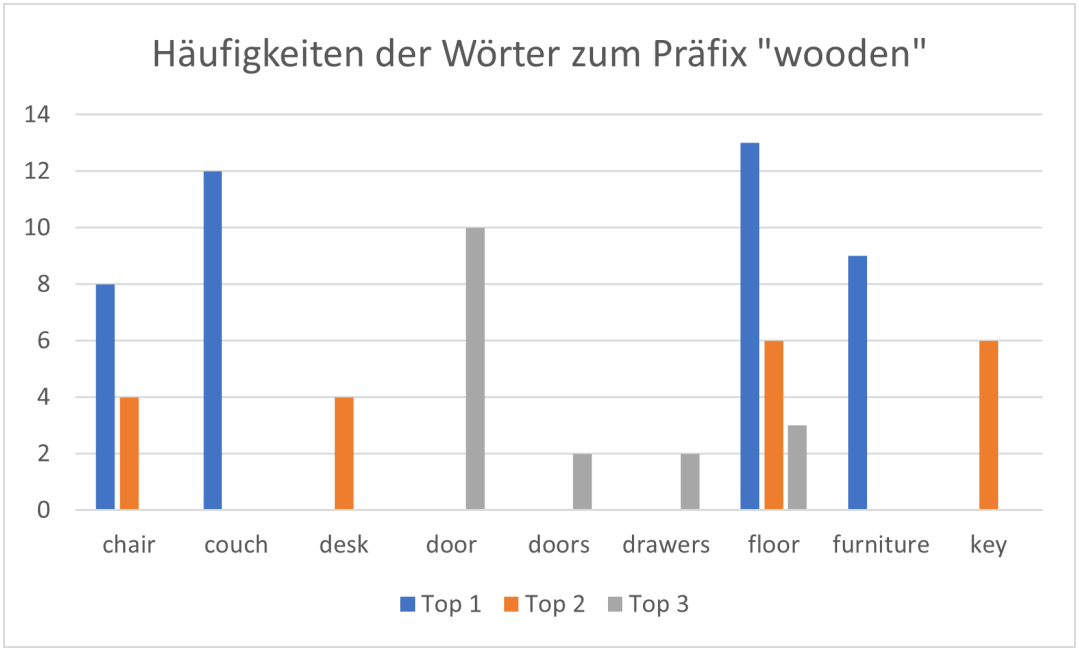

Abbildung 6.9.: Wörter, vor die das Präfix wooden, vom ObjectTagger gesetzt wurde

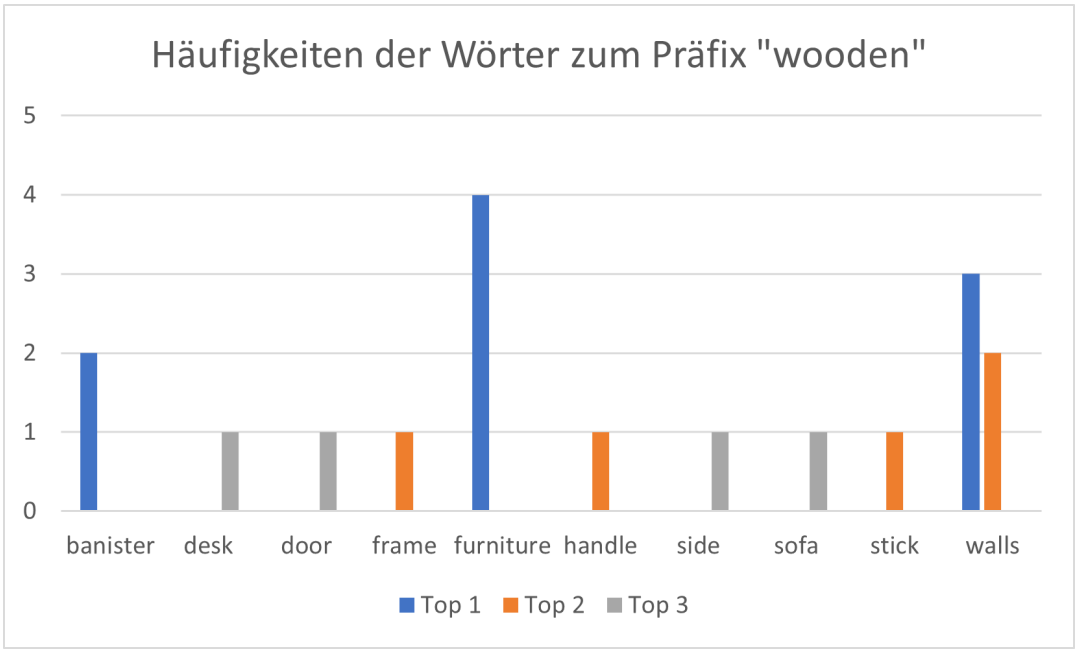

Abbildung 6.10.: Wörter, vor die das Präfix wooden, von DistilBERT gesetzt wurde

6.2. Evaluation der Holonym Relation und der Erkennung des Kontexts

Die Evaluation des Programms soll zeigen, wie genau die Erkennung der Holonyme und des Kontexts ist. Bei beiden Neuronalen Netze wird überprüft, wie gut sie die entsprechenden Beziehungen darstellen. Der entscheidende Faktor ist folglich die Korrektheit der Ausgabe.

6.2.1. Aufbau der Evaluation

Die Evaluation ist einfach aufgebaut und für beide Bereiche identisch. Dem Neuronalen Netz wurden verschiedene Texte übergeben, bei denen sowohl die entsprechenden Holonymbeziehungen für die Wörter als auch der Kontext herausgefunden werden mussten. Für jedes Wort, das ein Objekt darstellt, wurden passende Beispielergebnisse vorgegeben, die im besten Fall mit den Ausgaben des Programms übereinstimmten. Dadurch, dass nicht immer das gleiche Wort ausgewählt, sondern manchmal ein Synonym erfasst wurde, überprüft das System ebenfalls Synonyme, sofern sie in den vordefinierten Ergebnissen enthalten sind.

Ein weiterer wichtiger Punkt war die Generierung von Texten, die durch Bildbeschreibungen erzeugt werden konnten. Insgesamt wurden sechs verschiedene Bilder ausgewählt, die die Räume Wohnzimmer, Badezimmer und Küche zeigen. Zu jedem der drei Zimmer wurden zwei Bilder gewählt. Insgesamt stehen zwölf verschiedene Texte zur Verfügung, die die Bilder beschreiben, also sechs verschiedene Texte pro Bild. Sie folgen keiner Vorlage, sondern wurden als Freitext verfasst, wobei darauf geachtet werden sollte, das Meronyme verwendet werden, damit aus ihnen dann wieder die Holonyme bestimmt werden können. Dies konnte vor allem dadurch erreicht werden, dass die Bilder möglichst genau beschrieben werden, da der Autor dann zwangsläufig Meronyme verwenden muss. Ähnlich wie in Greene (2013) wurde die Texte nach Tokenanzahl und Wortarten ausgewertet. Dabei ergab sich eine durchschnittliche Tokenanzahl von 128. Wie in Abbildung 6.11 zu sehen ist, wurden Nomen am häufigsten verwendet, was nicht verwunderlich ist, da sie potenzielle Objekte beschreiben.

Tabelle 6.1.: Ausgewertet wurde, ob die Bilder richtig erkannt wurden

Raum	Hypernym	Kontext
Badezimmer	85,01%	100%
Wohnzimmer	70,94%	0%
Küche	75,94%	75%
Gesamt	77,30%	58,33%

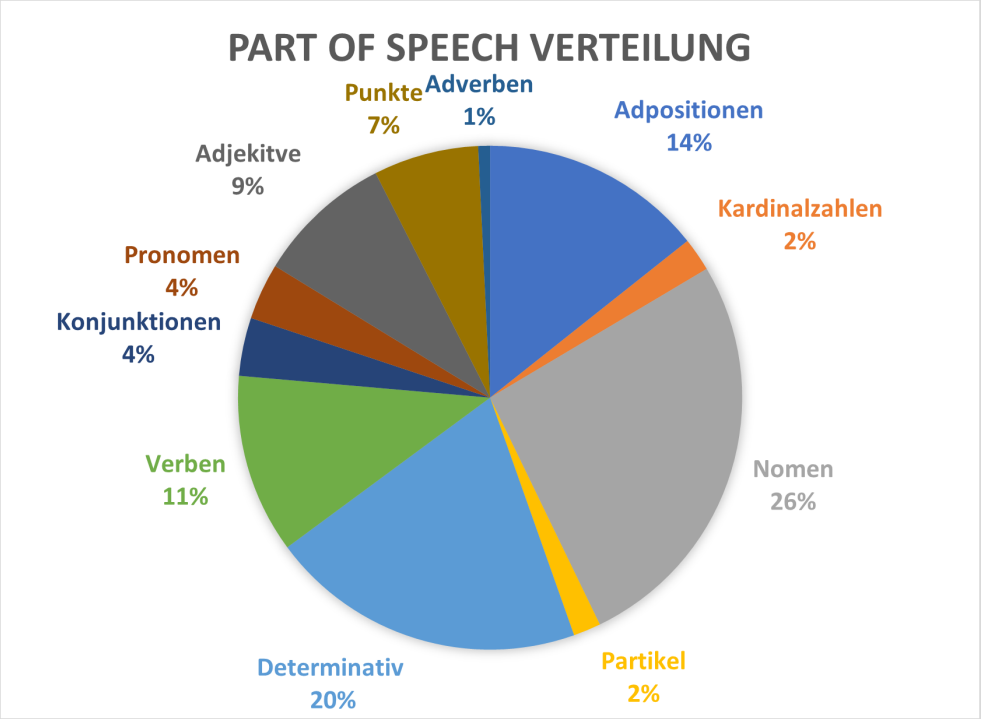

Abbildung 6.11.: Verteilung der Wortarten über alle 36 Texte

6.2.2. Ergebnisse

Nachdem die Texte durch das Programm mit Hilfe der gesamten Pipeline verarbeitet wurden, wurden die ausgegebenen Wörter ausgewertet. In Bezug auf die Holonymbestimmung zeigte sich, dass von allen Objekten, die erkannt werden mussten, 77,30% richtig zugeordnet wurden. Sehr genau in diesem Durchschnitt liegen die Texte zur Küche, wo 75,94% der Hypernyme korrekt verknüpft wurden. Die 75% setzen sich dabei aus den vier Texten zum Thema *Küche* zusammen, wobei die Quote bei drei Texten größer als der Durchschnitt ist. Nur ein Text ist mit 66% darunter. Das liegt unter anderem daran, dass Wörter wie *cooker* oder *plant* nicht richtig zugeordnet wurden, was dazu führt, dass die Quote an korrekt erkannten Objekten niedriger als bei den anderen ist. Ein weiterer Grund für nicht identifizierte

Holonyme beziehungsweise Objekte besteht darin, dass das Netz teilweise unterschiedliche Ausgaben bei einer sehr ähnlichen Eingabe erzeugt. Während in einem Text *plates* richtig zugeordnet wurde, wurde bei einem anderen Text *plate* falsch verknüpft. Das zeigt, dass das Netz bei manchen Entscheidungen noch unsicher ist. Diese Probleme dürften durch noch mehr Trainingsdaten gelöst werden, da das Netz die Neuronen so anpassen kann, dass die Entscheidungen bei noch mehr Objekten deutlich sicherer sind und keine Schwankungen mehr auftreten.

Die Ergebnisse in der Kategorie *Badezimmer* sind sehr gut. Hier wurden mit Abstand die besten Erkennungsraten erzielt (siehe Tabelle 6.1). Auch dort sind die Ergebnisse bei drei Texten sehr gut. Bei einem liegt sie sogar bei 100%. Der vierte Text ist allerdings deutlich schlechter mit gerade einmal 59% richtigen Ergebnissen. Dies dürfte darin begründet sein, dass dem Wort *sideboard* bei jedem Vorkommen *dining room* zugeordnet wurde. Ohne dieses eine Wort liegt die Quote auch wieder bei 77%. Dieses Wortpaar zeigt, dass es wahrscheinlich manche Wörter gibt, bei denen *Overfitting* aufgetreten ist, da ein *sideboard* nicht ausschließlich einem Raum zugeordnet werden kann, aber das Neuronale Netz die Verbindung stark trainiert hat.

Die letzte Kategorie, welche überprüft wurde, ist das Wohnzimmer. Hier wurden knapp 71% der Objekte richtig zugeordnet. Wie schon bei den beiden anderen Kategorien gab es auch hier drei Texte, deren Ergebnisse sehr gut waren und einen Text, bei dem das Ergebnis etwas abfällt. Auch dort lässt es sich wahrscheinlich auf nicht genügend Trainingsdaten zurückführen. Insgesamt kann man zum Schluss kommen, dass die Erkennungsraten pro Kategorie mindesten 70% betragen und damit der größte Teil an Objekten korrekt zugeordnet wird.

Die Evaluation zur Erkennung des Kontexts zeigt, dass insgesamt 58% richtig erkannt wurden. Dabei ist die Aufteilung zwischen den Kategorien sehr unterschiedlich. Bei den Räumen Badezimmer und Küche wurden alle beziehungsweise fast alle Texte richtig eingeordnet. Von den acht Texten wurden sieben richtig klassifiziert. Ein anderes Bild zeigt sich beim Thema Wohnzimmer. Hier wurde kein Text dem richtigen Raum zugeordnet. Dort wurde öfters Küche oder Esszimmer als Raum ausgegeben, was natürlich nicht korrekt war. Die falsche Zuordnung kann daran liegen, dass immer der Kontext für den gesamten Text ausgewählt wird, der am häufigsten vom Netz ausgegeben wird. Dies erhöht zwar einerseits die richtige Zuordnung, wie in den anderen beiden Fällen zu sehen ist, allerdings wird der Kontext dann bei einer falschen Entscheidung auch für alle anderen Worte in dem Text falsch gesetzt.

Die falsche Zuordnung könnte daran liegen, dass ein Wohnzimmer viele Objekte enthält, die andere Räume ebenfalls enthalten und daher die Abgrenzung zwischen den verschiedenen Räumen schwierig ist. Trotzdem lässt sich sagen, dass gute Ergebnisse erzielt wurden und das Netz den richtigen Kontext auswählt.

7. Ausblick

7.1. Fazit

Mit Blick auf die Aufgabenstellung (siehe Abschnitt 1.1), bei der ein System entstehen sollte, welches sowohl Holonyme von Objekten als auch deren Kontext bestimmen soll, lässt sich sagen, dass so ein Programm umgesetzt werden konnte. Dazu musste eine aus verschiedenen Komponenten zusammenhängende Pipeline erstellt werden (siehe Abschnitt 4.1), die sowohl für den VAnnotatoR (Spiekermann, Abrami und Mehler 2018) als auch in Teilen vom TextImager verwendet werden kann. Dabei konnte für die Erkennung von Objekten Wordnet (Miller 1998; Miller 1995) genutzt werden, welches bereits eine Kategorie für physische Objekte enthält. Diese wurden dann im TextImager als IsoSpatialEntities angelegt, was für die Verarbeitung von Vorteil ist, da dieser Klasse eine ISO genormte Datenstruktur zugrunde liegt und somit festgelegte Methoden und Attribute enthält. Zusätzlich zur Erkennung von Objekten wurden sie noch durch ein weiteres beschreibendes Wort ergänzt. Dies diente dazu, Objekte spezifischer zu klassifizieren. Dafür wurde DistilBERT (Sanh u. a. 2020) verwendet (Abschnitt 5.5), welches zum Kontext passende Wörter auswählt und vor die Objekte setzt. Da die erzeugten Wörter des Standardmodells allerdings nicht genau genug waren, wurde darauf aufbauend ein verfeinertes Modell trainiert, welches passender auf Wörter, die Objekte beschreiben ausgerichtet wurde. In Abschnitt 6.1 konnte gezeigt werden, dass dieses Modell besser funktioniert.

Die Informationen, die im TextImager gesammelt wurden und in einer Ausgabedatei zur Verfügung stehen, können vom VAnnotatoR ausgelesen werden und bieten die Grundlage für die Extraktion der in der Aufgabenstellung genannten Beziehungen. Um die Holonyme der Objekte zu bestimmen, wurde ein Neuronales Netz trainiert, welches für ein Wort als Output das eventuell vorhandene Holonym ausgibt. Dieser Output wurde dann in die modifizierte IsoSpatialEntity-Datenstruktur zurückgegeben. Da das Neuronale Netz nur mit Zahlen arbeitet, musste erst eine Konvertierung der Wörter in Vektoren mittels Word2Vec stattfinden. Dazu wurde GloVe genutzt, da es ein bereits fertiges Wörterbuch enthält. In einer Auswertung Unterabschnitt 6.2.2 konnte gezeigt werden, dass durchschnittlich 77% der Objekte korrekt zugeordnet werden konnten.

Das zweite Thema, das in der Aufgabenstellung behandelt wird, ist die Erkennung des Kontexts in Bezug auf Räume beziehungsweise Locations. Hierfür wurde ebenfalls auf den Ergebnissen des TextImagers aufgebaut und ein Neuronales Netz trainiert, das den Kontext des Texts ausgeben soll. Dazu wurde als Eingabe nicht nur das betrachtete Wort in das Netz eingegeben, sondern auch noch dessen Nachbarn. Wie bereits oben ausgeführt, ist die ISOSpatialEntity-Datenstruktur hier ebenfalls die zentrale Schnittstelle, sodass die Wörter um die vom Netz ausgegebenen Informationen ergänzt werden können. Die Evaluation zeigt, dass im Durchschnitt in 58% der Fälle der Kontext richtig gesetzt wurde.

Sobald die ganzen Informationen vollständig ergänzt wurden, konnten die Objekte mit Hilfe einer Suche in der ShapeNet-Datenbank (Chang u. a. 2015) in den VAnnotatoR geladen werden, sodass sie der User dort platzieren kann. Dabei wurden zwei verschiedene User Interfaces geschaffen. Das erste ist ein automatisches System Abschnitt 5.2, bei dem der User nur den Text auswählt und die komplette Verarbeitung im Hintergrund abläuft, sodass er nur die bereits geladenen Objekte platzieren muss. Beim zweiten handelt es sich um einen etwas weniger automatisierten Ansatz, der es dem User ermöglicht, für jedes Objekt ein spezielles ShapeNet-Objekt auszusuchen und manuell in die Szene zu laden.

Text2Scene ist in der aktuellen Forschung ein häufig diskutiertes Thema, welches immer mehr an Bedeutung gewinnt, da es neue Formen der Textvisualisierungen mit sich bringt. Ein bekanntes System ist Wordseye (Coyne und Sproat 2001). Diese Arbeit beziehungsweise die Arbeit des Lehrstuhls bietet das Alleinstellungsmerkmal, dass Text2Scene auch für die VR verfügbar gemacht wird. Da die VR andere Wege der Interaktion mit einer virtuellen Szene ermöglicht als ein 2D-Bild, ist es sicherlich ein Forschungsthema, dessen Bedeutung in den nächsten Jahren noch zunehmen wird.

7.2. Offene Probleme für die Wissenschaft und mögliche Weiterentwicklungen

Obwohl diese Arbeit bereits einige Grundlagen schafft, gibt es Punkte, an denen weitergearbeitet werden kann und bei denen das Programm noch Potenzial für Verbesserungen hat. Dies lässt sich in mehrere Aspekte gliedern.
Aktuell ist die gesamte Textverarbeitung nur auf Englisch verfügbar. Dies könnte auf weitere Sprachen ausgeweitet werden. Dafür müssten unter anderem die beiden selbst trainierten

Neuronalen Netze mit Wörtern aus anderen Sprachen neu trainiert werden. Dies setzt voraus, dass eine zu GloVe ähnliche Word2Vec-Datenbank in anderen Sprachen nutzbar ist. Neben dem eigenen Neuronalen Netz ist man aber auch noch auf die Verfügbarkeit von den anderen in dieser Arbeit genutzten Systemen angewiesen. Dies umfasst sowohl ein WordNet als auch ein BERT Modell in der entsprechenden Sprache.
Außerdem kann man die Texterkennung an sich noch erweitern. Aktuell liegt der Fokus vor allem auf Teil-Ganz Relationen. Da die ShapeNet-Suche allerdings keine Optionen zur spezifischen Suche von Objekten zulässt, muss man möglichst Präzise Suchwörter wählen, um das gewünschte Objekt zu erhalten. Daher könnte man die Textverarbeitung so ausbauen, dass sie neben Teil-Ganz Relationen auch wesentlich mehr Informationen extrahiert wie beispielsweise die Zeit, in der der Text spielt. Um ein Wohnzimmer aus den siebziger Jahren realistisch darzustellen, benötigt man ganz andere Objekte als ein Zimmer aus der heutigen Zeit. Aber nicht nur die Zeit ist ein Faktor, der eine Szene ganz anders aussehen lassen kann. Auch der Ort, an dem die Geschichte spielt, kann einiges an der Einrichtung verändern. Ein Bauernhaus auf dem Land ist anders ausgestattet als eine Wohnung in der Stadt.
Neben der Extraktion von Informationen aus dem Text, kann es wichtig sein, die VR-Szene mit weiteren Objekten anzureichern, die nicht im Text genannt werden. Wenn durch das System erkannt wird, dass der Text in einer Küche spielt, könnten automatisch weitere Gegenstände geladen werden, die in den entsprechenden Raum gehören, aber nicht im Text genannt wurden. Diese könnten, wie auch die im Text vorhandenen Objekte automatisch platziert werden, sodass der User nur den Text auswählen muss und dann als Ergebnis eine fertige Szene sieht, die er sich anschauen kann.
Neben der Textverarbeitung lässt sich auch die VR-Szene erweitern. Aktuell ist man auf die ShapeNet-Datenbank angewiesen. Ein Editor in der VR zum Erstellen seiner eigenen Objekte, könnte die Anzahl und Variationen von Objekten weiter erhöhen. Der User kann dann aus einer Art Baukasten selbst einen Gegenstand kreieren und erzeugen.

Literatur

Abrami, Giuseppe, Alexander Henlein, Attila Kett und Alexander Mehler (2020a). „Text-2SceneVR: Generating Hypertexts with VAnnotatoR as a Pre-processing Step for Text2Scene Systems“. In: Proceedings of the 31st ACM Conference on Hypertext and Social Media. HT 20. Virtual Event, USA: Association for Computing Machinery, S. 177–186. isbn: 9781450370981. doi: 10.1145/3372923.3404791. url: https://doi.org/10.1145/3372923.3404791.

– (2020b). „Text2SceneVR: Generating Hypertexts with VAnnotatoR as a Pre-Processing Step for Text2Scene Systems“. In: Proceedings of the 31st ACM Conference on Hypertext and Social Media. HT '20. Virtual Event, USA: Association for Computing Machinery, S. 177–186. isbn: 9781450370981. doi: 10.1145/3372923.3404791. url: https://doi.org/10.1145/3372923.3404791.

Abrami, Giuseppe, Alexander Mehler und Christian Spiekermann (Juli 2019). „Graph-based Format for Modeling Multimodal Annotations in Virtual Reality by Means of VAnnotatoR“. In: Proceedings of the 21th International Conference on Human-Computer Interaction, HCII 2019. Hrsg. von Constantine Stephanidis und Margherita Antona. HCII 2019. Orlando, Florida, USA: Springer International Publishing, S. 351–358. isbn: 978-3-030-30712-7.

Bengio, Yoshua, Réjean Ducharme, Pascal Vincent und Christian Jauvin (2003). „A neural probabilistic language model“. In: Journal of machine learning research 3.Feb, S. 1137–1155.

Bird, Steven, Ewan Klein und Edward Loper (2009). Natural Language Processing with Python. 1st. O'Reilly Media, Inc. isbn: 0596516495.

Bishop, Christopher M (2006). Pattern recognition and machine learning. Information science and statistics. Softcover published in 2016. New York, NY: Springer. url: https://cds.cern.ch/record/998831.

Brachman, Ronald J. und Thomas Dietterich, Hrsg. (2009). Introduction to Semi-Supervised Learning. Morgan und Claypool. isbn: 9781598295481.

Bucilu, Cristian, Rich Caruana und Alexandru Niculescu-Mizil (2006). „Model compression“. In: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, S. 535–541.

Chang, Angel X. u. a. (2015). ShapeNet: An Information-Rich 3D Model Repository. Techn. Ber. arXiv:1512.03012 [cs.GR]. Stanford University — Princeton University — Toyota Technological Institute at Chicago.

Coyne, Bob und Richard Sproat (2001). „WordsEye: An Automatic Text-to-Scene Conversion System“. In: Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques. SIGGRAPH '01. New York, NY, USA: Association for Computing Machinery, S. 487–496. isbn: 158113374X. doi: 10.1145/383259.383316. url: https://doi.org/10.1145/383259.383316.

Devlin, Jacob, Ming-Wei Chang, Kenton Lee und Kristina Toutanova (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv: 1810.04805 [cs.CL].

Glorot, Xavier, Antoine Bordes und Yoshua Bengio (2011). „Deep sparse rectifier neural networks“. In: Proceedings of the fourteenth international conference on artificial intelligence and statistics. JMLR Workshop und Conference Proceedings, S. 315–323.

Greene, Michelle R (2013). „Statistics of high-level scene context“. In: Frontiers in psychology 4, S. 777.

Hawkins, Douglas M. (2004). „The Problem of Overfitting“. In: Journal of Chemical Information and Computer Sciences 44.1. PMID: 14741005, S. 1–12. doi: 10.1021/ci0342472. eprint: https://doi.org/10.1021/ci0342472. url: https://doi.org/10.1021/ci0342472.

Hemati, Wahed, Tolga Uslu und Alexander Mehler (2016). „TextImager: a Distributed UIMA-based System for NLP“. In: Proceedings of the COLING 2016 System Demonstrations. Federated Conference on Computer Science und Information Systems. Osaka, Japan.

Hinton, Geoffrey, Oriol Vinyals und Jeff Dean (2015). „Distilling the knowledge in a neural network“. In: arXiv preprint arXiv:1503.02531.

ISO (2002). Codes for the representation of names of languages Part 1: Alpha-2 code. International Organization for Standardization, Geneva, CH. url: https://www.iso.org/standard/22109.html.

– (2014). Language resource management Semantic annotation framework (SemAF) Part 7: Spatial information (ISOspace). Standard ISO/IEC TR 24617- 7:2014. International Organization for Standardization, Geneva, CH. url: https://www.iso.org/standard/60779.html.

ISO (2020). Language resource management Semantic annotation framework Part 7: Spatial information. Standard ISO/IEC TR 24617- 7:2014. International Organization for Standardization, Geneva, CH. url: https://www.iso.org/standard/76442.html.

Juliani, Arthur u. a. (2020). Unity: A General Platform for Intelligent Agents. arXiv: 1809.02627 [cs.LG].

Kett, Attila, Giuseppe Abrami, Alexander Mehler und Christian Spiekermann (2018). „Resources2City Explorer: A System for Generating Interactive Walkable Virtual Cities out of File Systems“. In: Proceedings of the 31st ACM User Interface Software and Technology Symposium. Berlin, Germany.

Kühn, Vincent, Giuseppe Abrami und Alexander Mehler (2020). „WikNectVR: A Gesture-Based Approach for Interacting in Virtual Reality Based on WikNect and Gestural Writing“. In: Virtual, Augmented and Mixed Reality. Design and Interaction. Hrsg. von Jessie Y. C. Chen und Gino Fragomeni. Cham: Springer International Publishing, S. 299–312. isbn: 978-3-030-49695-1.

Liu, Tianyi, Shuangsang Fang, Yuehui Zhao, Peng Wang und Jun Zhang (2015). „Implementation of training convolutional neural networks“. In: arXiv preprint arXiv:1506.01195.

Martín Abadi u. a. (2015). TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. Software available from tensorflow.org. url: https://www.tensorflow.org/.

Medsker, Larry R und LC Jain (2001). „Recurrent neural networks“. In: Design and Applications 5.

Mehler, Alexander, Giuseppe Abrami, Christian Spiekermann und Matthias Jostock (2018). „VAnnotatoR: A Framework for Generating Multimodal Hypertexts“. In: Proceedings of the 29th ACM Conference on Hypertext and Social Media. Proceedings of the 29th ACM Conference on Hypertext and Social Media (HT '18). Baltimore, Maryland: ACM.

Mikolov, Tomas, Kai Chen, Greg Corrado und Jeffrey Dean (2013). „Efficient estimation of word representations in vector space“. In: arXiv preprint arXiv:1301.3781.

Miller, George A (1995). „WordNet: a lexical database for English“. In: Communications of the ACM 38.11, S. 39–41.

– (1998). WordNet: An electronic lexical database. MIT press.

Mo, Kaichun u. a. (Juni 2019). „PartNet: A Large-Scale Benchmark for Fine-Grained and Hierarchical Part-Level 3D Object Understanding“. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

Montúfar, Guido, Razvan Pascanu, Kyunghyun Cho und Yoshua Bengio (2014). „On the number of linear regions of deep neural networks“. In: arXiv preprint arXiv:1402.1869.

Nandy, Abhishek (2018). Neural Networks in Unity: C# Programming for Windows 10. 1st edition. isbn: 9781484236734. url: https://learning.oreilly.com/library/view/-/9781484236734.

Pennington, Jeffrey, Richard Socher und Christopher D. Manning (2014). „GloVe: Global Vectors for Word Representation“. In: Empirical Methods in Natural Language Processing (EMNLP), S. 1532–1543. url: http://www.aclweb.org/anthology/D14-1162.

Pustejovsky, James, Jessica L Moszkowicz und Marc Verhagen (2011). „ISO-Space: The annotation of spatial information in language“. In: Proceedings of the Sixth Joint ISO-ACL SIGSEM Workshop on Interoperable Semantic Annotation. Bd. 6, S. 1–9.

Qi, Peng, Timothy Dozat, Yuhao Zhang und Christopher D. Manning (Okt. 2018). „Universal Dependency Parsing from Scratch“. In: Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies. Brussels, Belgium: Association for Computational Linguistics, S. 160–170. url: https://nlp.stanford.edu/pubs/qi2018universal.pdf.

Radford, Alec, Karthik Narasimhan, Tim Salimans und Ilya Sutskever (2018). Improving language understanding by generative pre-training.

Rajpurkar, Pranav, Robin Jia und Percy Liang (2018). Know What You Don't Know: Unanswerable Questions for SQuAD. arXiv: 1806.03822 [cs.CL].

Rajpurkar, Pranav, Jian Zhang, Konstantin Lopyrev und Percy Liang (2016). SQuAD: 100,000+ Questions for Machine Comprehension of Text. arXiv: 1606.05250 [cs.CL].

Rey, Günter Daniel und Karl Friedrich Wender (2018). Neuronale netze: Eine einführung in die grundlagen, anwendungen und datenauswertung. Hogrefe.

Riedmiller, M. und H. Braun (1993). „A direct adaptive method for faster backpropagation learning: the RPROP algorithm“. In: IEEE International Conference on Neural Networks, 586–591 vol.1. doi: 10.1109/ICNN.1993.298623.

Sang, Erik F. Tjong Kim und Fien De Meulder (2003). „Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition“. In: CoRR cs.CL/0306050. url: http://arxiv.org/abs/cs/0306050.

Sanh, Victor, Lysandre Debut, Julien Chaumond und Thomas Wolf (2020). DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. arXiv: 1910.01108 [cs.CL].

Spiekermann, Christian, Giuseppe Abrami und Alexander Mehler (2018). „VAnnotatoR: a Gesture-driven Annotation Framework for Linguistic and Multimodal Annotation“. In: Proceedings of the Annotation, Recognition and Evaluation of Actions (AREA 2018) Workshop. AREA. Miyazaki, Japan.

T., Simpson und Crowe M. (2005). WordNet.Net. url: http://www.ebswift.com/wordnetnet.html (besucht am 17. 02. 2021).

Taylor, Wilson L. (1953). „Cloze Procedure: A New Tool for Measuring Readability“. In: Journalism Quarterly 30.4, S. 415–433. doi: 10.1177/107769905303000401. eprint: https://doi.org/10.1177/107769905303000401. url: https://doi.org/10.1177/107769905303000401.

Unity Technologies (2020). Unity ML-Agents Toolkit Documentation. url: https://github.com/Unity-Technologies/ml-agents/blob/release_4_docs/docs/ML-Agents-Overview.md.

Vaswani, Ashish u. a. (2017). Attention Is All You Need. arXiv: 1706.03762 [cs.CL].

Wang, Alex u. a. (2019). GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. arXiv: 1804.07461 [cs.CL].

Werbos, P. J. (1990). „Backpropagation through time: what it does and how to do it“. In: Proceedings of the IEEE 78.10, S. 1550–1560. doi: 10.1109/5.58337.

Wolf, Thomas u. a. (Okt. 2020). „Transformers: State-of-the-Art Natural Language Processing“. In: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations. Online: Association for Computational Linguistics, S. 38–45. url: https://www.aclweb.org/anthology/2020.emnlp-demos.6.

A. Entwicklerinformationen

Alle Skripte, Szenen und Resources sind im Ordner *Assets/Text2Scene2* enthalten. Dabei gibt es zwei verschiedene Unity-Szenen. Die erste ist die *Text2Scene_Learning* Szene, in der man die Neuronalen Netze trainieren kann und die zweite nennt sich *SceneBuilder* und ist für den Endnutzer gedacht.

A.1. Text2Scene_Learning Szene

Sie beinhaltet alle Funktionen, um die Neuronalen Netze zu trainieren. Das zentrale Objekt in der Szene ist das *TrainingsObject*. An das Objekt ist das *TrainingsScript* angeheftet. Dort kann man auswählen, welches Netz man trainieren möchte. Entweder *Learn Part Net*, *Learn Disambiguation* oder *Learn Containment*. Durch diese Auswahl wird beim Starten des Game-Modus die entsprechenden Daten geladen, mit denen man das Neuronale Netz trainieren kann. Die Datei, von der die Daten geladen werden, kann man in der Klasse *NN_Helper* einstellen. Zum Abschließen der Konfiguration muss man noch die entsprechenden Agenten für das Training auswählen. Dafür gibt es in der Szene die Objekte *PartNetTraining*, *DisamTraining* und *ContainmentTraining*. Sie enthalten jeweils zehn Agenten, die bereits konfiguriert sind und nur aktiviert werden müssen. Damit das Training richtig funktioniert, ist es wichtig, dass immer nur ein Objekt von den dreien aktiviert ist. Folglich reicht es, wenn nur das Objekt und das *TrainingsObject* aktiviert sind.
Neben dem Training gibt es die Möglichkeit das Netz zu nutzen und die Ausgaben auszulesen beziehungsweise abzuspeichern. Dazu wurde die Klasse *Semi Supervised PartNet* erstellt. Sobald das Skript im *TrainigsObject* aktiviert wurde, liest es die Daten ein, zu denen das Neuronale Netz Vorhersagen treffen soll. Zum Einlesen der Daten gibt es im Editor die Felder *ObjectList*, *DisamInput* und *PartNetInput*. Sie nehmen die Ausgangsdaten und die manuell annotierten Daten für den Kontext und die Holonyme an. Zusätzlich müssen wieder die Pfade für die Ausgabe angegeben werden (SemiSupervisedPartNet.cs).

B. Verwendete Bilder

Abbildung B.1.: Quelle: https://pixabay.com/de/photos/badezimmer-bad-wc-waschbecken-2094733/

Abbildung B.2.: Quelle: https://pixabay.com/de/photos/badezimmer-waschbecken-spiegel-2094716/

Abbildung B.3.: Quelle: https://pixabay.com/de/photos/apartment-lounge-wohnzimmer-m%C3%B6bel-3147892/

Abbildung B.4.: Quelle: https://pixabay.com/de/photos/innenraum-wohnzimmer-m%C3%B6bel-zimmer-1961070/

Abbildung B.5.: Quelle: https://www.pexels.com/de-de/foto/weisse-keramikplatte-auf-braunem-holztisch-5824883/

Abbildung B.6.: Quelle: https://www.pexels.com/de-de/foto/kucheninsel-und-barhocker-534151/