LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

PROGRAMMING LANGUAGES AND ARTIFICIAL INTELLIGENCE

Titel der Arbeit

Ruben Triwari

Bachelorarbeit
im Studiengang ’Informatik plus Mathematik’

Betreuer: Prof. Dr. Johannes Kinder

Mentor: Moritz Dannehl, M.Sc.

Ablieferungstermin: 12. September 2024

Inhaltsverzeichnis

Abstract

1 Einführung

In den letzten Jahren gab es große Fortschritte in der natürlichen Sprachverarbeitung, besonders hervorzuheben sind Large Language Models die sich mittlerweile in vielen Bereichen der Informatik in die Lösungsansätze für Problemen in jeweiligen Bereichen eingeschlichen haben. Diese Arbeit untersucht nun, ob diese Fortschritte in der natürlichen Sprachverarbeitung eine Hilfestellung leisten können um Source Code Funktionen semantisch sinvoll in einen Vektor mit reelwertigen Zahlen zu codieren. Diese Vektoren können dann später als Label verwendet werden um ein Modell zu trainieren was Binary Code als Input nimmt und diesen ebenfalls in einen semantischen Vektor mit reelwertigen Zahlen codiert. Das resultierende Modell kann hinterher verwendet werden um Reverse Engeneering zu erleichtern. Ein einfaches Beispiel ist folgendes: Man stelle sich vor, dass man eine Funktion die in Binary Code vorliegt, mühsehlig manuell verstanden was für eine Aufgabe die Funktion in der Code Base hat. Nun kann man diese Funktion codieren und über die Gesamte Code Base einen Nearest Neighbor Search durchführen und all ähnlichen Funktionen ausgeben lassen. Das spart zeit, denn nun hat man eine Idee was diese anderen Funktionen für eine Aufgabe in der Code Base erfüllen könnten.
Das oben berschriebene Problem Source Code Vektoren in sinvoll semantische reelwertige Vektoren zu codieren ist sehr ähnlich zu einen Problem in der natürlichen Sprachverarbeitung und dort bereits gelöst. Die rede ist von dem Problem einen gegebenen Satz in einen semantisch sinvollen Vektor abzubilden. Es ist nageliegend zu versuchen dieses Ergebnis der natürlichen Sprachverarbeitung zu benuzten um eine Lösung für unser Problem zu konstruieren. Die intuivste Idee ist es einfach die Funktionsnamen, die in natürlicher Sprache verfasst sind als beschreibung der Funktion zu verwenden. Diese Beschreibunf können wir nun mühelos codieren, da sie in natürlicher Sprache vorliegt. Eine zweite Idee ist, die Kommentare der Funktionen, die in natürlicher Sprache verfasst sind, als Beschreibung der Funktion zu verwenden. Am viel versprechsten ist es die Funktionen von einen Large Language Modell in natürlicher Sprache beschreiben zu lassen. Als letztes habe ich noch ein bestehendes Modell Code2Vec verwendet und es für dieses Problem angepasst.

2 Grundlagen und Termini

2.1 Maschienelles Lernen

Heutzutage ist maschienelles Lernen weitverbreitet und wird in nahezu jeden Bereich der Informatik verwendet. In diesen Abschnitt wird zunächst maschinelles lernen definiert und dann darauf aufbauend grundlegende Trainingsarten vorgestellt. Maschinelles Lernen wird überall dort eingesetzt wo eine anayltische Lösung eines Problems zu aufwendig oder gar überhaupt nicht existiert. Diese Lösung durch maschinelles Lernen versucht aus den Daten ein Muster abzuleiten. Bei einer endlichen Menge an Daten ist meist, das resultierende Modell nur eine Approximation der gesuchten Lösung.

2.1.1 Definition

Trotz des bekanntheitsgrades, gibt es den irreglauben, dass maschinelles lernen nur was mit Neuronalen Netzwerken zu tun hat, diese Anahme ist im allgemeinen falsch. Generell kann ein Problem das mit maschinelles Lernen gelöst wird, wie folgt formulliert werden:

Definition 1 *Sei X eine beliebige Input Menge, Y eine beliebige Output Menge, $f \in \{X \to Y\}$ die gesuchte Lösung des Problems, $\mathbb{D}$ eine beliebige Menge aus gegebenen Datenpunkten,*
$H_1 \subset \{X \to Y\}$ ein Hypothesenraum, und $A_1 : \mathscr{P}(\{X \to Y\}) \times \mathscr{P}(\mathbb{D}) \to \{X \to Y\}$ ein Lernalgorithmus. Dann ist das ziel, bei gegebenen Daten, den Hypothesenraum H_1 und den Lernalgorithmus A_1 so zu wählen, sodass

$$A_1(H_1, \mathbb{D}) \approx f.$$

Maschinelles lernen ist also die suche nach einen Lernalgorithmus und Hypothesenraum, die dann in kombination mit gegebenen Daten, die optimale Lösung approximieren. Dabei ist hervorzuheben das der Datensatz das Herzstück jeder Problemstellung im Bereich des maschinellen Lernens ist. Ist der Datensatz zu klein oder überhaupt nicht representativ für das gegebene Probelm, wird der Lernalgorithmus die falschen Muster erkennen und dadurch eine fehlerhafte Approximation produzieren.

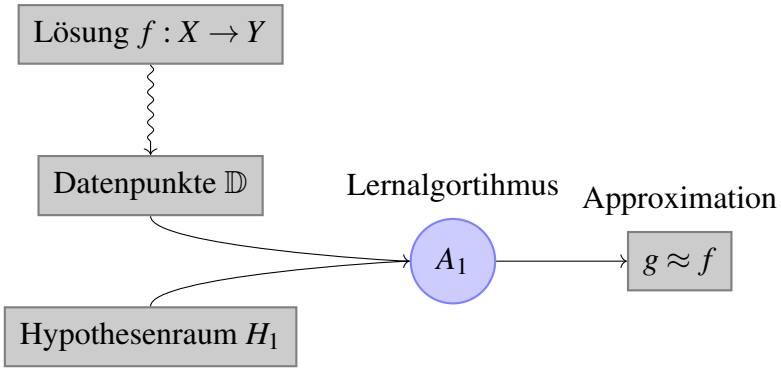

Abbildung 2.1: Grundlegendes maschinelles Lernen Problem

In der Figur 2.1 ist die Problembeschreibung nochmal bildlich dargestellt, erwähnenswert ist,

dass die Datenpunkte nicht immer in abhängigkeit mit $f : X \to Y$ stehen. Beispielsweise können die Datenpunkte einfach nur aus den Eingabewerten bestehen: $\mathbb{D} = \{x_1, x_2, x_2, \ldots, x_n\} \subset X$. Die Struktur des Datensatzes kann sehr unterschiedlich sein, dass hängt auch mit unterschiedlichen Lernmehthoden zusammen.

2.1.2 Deep-Learning

Bevor die Lernmethoden genauer betrachtet werden, wird kurz Deep-Learning vorgestellt. Deep-Learning ist ein Neuronales Netzwerk mit mehreren Layern zwischen Input und Output Layer. Zunächst müssen wir jedoch Neuronale Netzwerke definieren.

Definition 2 *Ein Neuronales Netzwerk (NN) ist eine Funtion $N : \mathbb{R}^q \to \mathbb{R}^p$, wobei $q \in \mathbb{N}$ die Anzahl der Inputs und $p \in \mathbb{N}$ die Anzahl der Outputs ist. Sei $(L_i)_{i\in\{1,\ldots,n\}}$ die Layer, $(K_i^l)_{l=1,\ldots,n,i=1,\ldots r_l}$ die Konten im jeweiligen Layer $l \in \{1,\ldots,n\}$ und $r_l \in \mathbb{N}$ die Anzahl der Knoten im Layer L_l. Jeder Koten im Layer L_l ist mit jedem Knoten im Layer L_{l+1} verbunden, mit $l \in \{1,\ldots,n-1\}$. Jede Verbindung besitzt ein Gewicht $W_{i,j}^l$, wobei $l \in \{1,\ldots,n-1\}$ und das Gewicht der Verbindung $K_i^l \to K_j^{l+1}$ zugeordnet ist. Daraus ergibt sich eine Famillie von Matritzen $(W_l)_{l=1,\ldots,n-1}$, wobei $W_l \in \mathbb{R}^{r_l \times r_{l+1}}$. Nun hat jeder Layer noch ein sogenannten Bias $(B_l)_{l=2,\ldots,n}$, dieser ist ein Zeilenvektor $B_l \in \mathbb{R}^{r_l}$. Als letztes braucht jeder Knoten eine Aktivierungsfunktion, dass heißt für jeden Layer gibt es r_l Funktionen: $(F_l)_{l=1,\ldots,n}$, mit $F_l \in \{\mathbb{R} \to \mathbb{R}\}^{r_l}$. Es ist hilfreich die Funktionsanwendung auch für den Vektor F_l zu definieren: Sei $x \in \mathbb{R}^{r_l}$, dann setze*

$$F_l(x) := \begin{pmatrix} f_1(x_1) \\ \vdots \\ f_{r_l}(x_{r_l}) \end{pmatrix}.$$

Dann ist die Funktion $N : \mathbb{R}^q \to \mathbb{R}^p$ wie folgt definiert:

$$N(x) = h_1(x)$$

,wobei

$$h_l : \mathbb{R}^{r_l} \to \mathbb{R}^{r_l+1}$$

$$h_l(x) = \begin{cases} h_{l+1}(F_{l+1}(W_l x + B_{l+1})) & ,if\ l < n \\ x & ,sonst \end{cases}.$$

Wir bezeichnen L_1 als Input-Layer, L_n als Output-Layer und L_i, mit $i \in \{2,\ldots,n-1\}$, als Hidden-Layer.

Also ist Deep-Learning ein bestimmer Hypothesenraum, denn alle Neuronale Netzwerke sind höher dimensionale reelwertige Funktionen. Schließlich gilt für den Hypothesenraum:

$$H = \{\mathbb{R}^n \to \mathbb{R}^k\},\ \text{wobei } n,k \in \mathbb{N}$$

Die Definition von einem Neuronalen Netzwerk erscheint zunächst länglich und unintuitiv, diese wird aber anschaulich anhand eines Beispiels.

Beispiel 2.1 *Sei $N : \mathbb{R}^2 \to \mathbb{R}^2$, $(L_i)_{i=1,2,3}$ Layer. Der Input-Layer besitzt zwei Knoten $r_1 = 2$, der erste Hidden-Layer besitzt $r_2 = 3$, der zweite Hidden-Layer besitzt $r_3 = 3$ Knoten und der Output-Layer besitzt $r_4 = 2$ Knoten. Mit den Knoten $(K_i^l)_{l=1,2,3,i=1,\dots,r_l}$ und zufälligen Gewichten:*

$$W_1 = \begin{pmatrix} 0.2 & 0.7 & 0.4 \\ 0 & 0.7 & 0.8 \end{pmatrix} \in \mathbb{R}^{2\times 3}, W_2 = \begin{pmatrix} 0.6 & 0 & 0.4 \\ 0.1 & 0.7 & 0.8 \\ 1 & 0.33 & 0.2 \end{pmatrix} \in \mathbb{R}^{3\times 3}, W_3 = \begin{pmatrix} 0.2 & 0.45 \\ 0.1 & 0.23 \\ 1 & 0.33 \end{pmatrix} \in \mathbb{R}^{3\times 2}.$$

Für den Bias setzen wir:

$$B_2 = \begin{pmatrix} 0.1 \\ 0.2 \\ 0.3 \end{pmatrix} \in \mathbb{R}^3, B_3 = \begin{pmatrix} 0.4 \\ 0.5 \\ 0.6 \end{pmatrix} \in \mathbb{R}^3, B_4 = \begin{pmatrix} 0.7 \\ 0.8 \end{pmatrix} \in \mathbb{R}^2,$$

Außerdem setzen wir alle Aktivierungfunktionen:

$$(F_l)_i = tanh, \text{ wobei } l \in \{1,2,3\}, i \in \{1,\dots,r_l\}$$

Dann gilt für das Neuronales Netzwerk $N : \mathbb{R}^2 \to \mathbb{R}^2$:

$$N(x) = tanh(\begin{pmatrix} 0.2 & 0.45 \\ 0.1 & 0.23 \\ 1 & 0.33 \end{pmatrix} tanh(\begin{pmatrix} 0.6 & 0 & 0.4 \\ 0.1 & 0.7 & 0.8 \\ 1 & 0.33 & 0.2 \end{pmatrix} tanh(\begin{pmatrix} 0.2 & 0.7 & 0.4 \\ 0 & 0.7 & 0.8 \end{pmatrix} x + \begin{pmatrix} 0.1 \\ 0.2 \\ 0.3 \end{pmatrix}) + \begin{pmatrix} 0.4 \\ 0.5 \\ 0.6 \end{pmatrix}) + \begin{pmatrix} 0.7 \\ 0.8 \end{pmatrix})$$

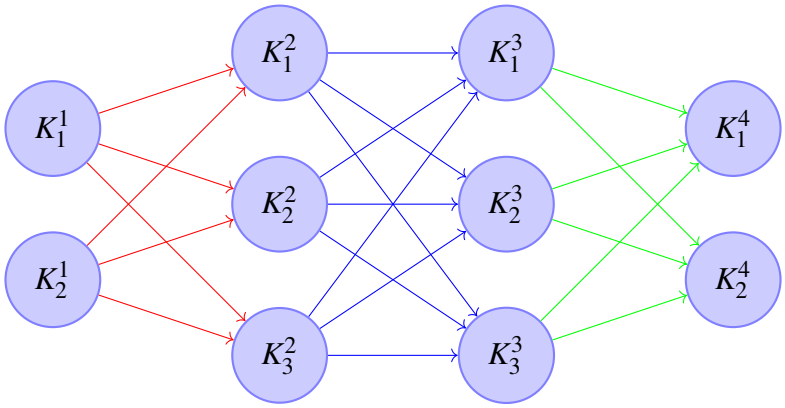

Abbildung 2.2: Neuronales Netzwerk bildlich als Graph dargestellt

Ein Gewicht zwischen zwei Knoten $K_1^1 \to K_1^2$ kann nun einfach nachgeschaut werden:

$$(W_1)_{1,1} = 0.2$$

Bei dem Beispiel oben handelt es sich um Deep-Learning, da das Neuronale Netzwerk zwei Hidden-Layer besitzt. Heutzutage haben Deep-Learning Models zwei bis drei stellige Anzahl an Hidden-Layer. Diese Dimensionen sind aber für ein Beispiel eher ungeignet.

2.1.3 Überwachtes Lernen

Das überwachte Lernen ist die meist eingesetzte Trainingsmethode, deswegen auch die wichtigste. Bei diesem Ansatz liegt immer die korrekte Lösung für jeden Input bei, demtentsprechend ist der Datensatz ein Tuple aus Input und korrekten Output:

$$\mathbb{D} = \{(x_1, y_1), (x_2, y_2)(x_3, y_3), \ldots, (x_n, y_n)\} \subset X \times Y.$$

Ein klasisches Beipiel für diesen Ansatz ist die Bilderkennung, hierbei wäre ein möglicher input ein Vektor mit Grauwerten und der Output ein Label bzw. die Bezeichung für das Bild.

Beispiel 2.2 *Sei $X = \mathbb{R}^{4096}$ und $Y = \{Katze, Hund, Auto\}$, dann könnte der Datensatz wie folgt aussehen:*

$$\mathbb{D} = \{(\begin{pmatrix} 0.2 \\ 0.9 \\ 0.5 \\ \vdots \end{pmatrix}, Hund), (\begin{pmatrix} 0.1 \\ 0.1 \\ 0.6 \\ \vdots \end{pmatrix}, Hund), (\begin{pmatrix} 0.6 \\ 0.7 \\ 0 \\ \vdots \end{pmatrix}, Katze), \ldots, (\begin{pmatrix} 0.4 \\ 0.3 \\ 0.9 \\ \vdots \end{pmatrix}, Auto)\}$$

Der entscheidende Punkt ist also hier, dass wir das richtige verhalten unseres Modells kennen und deswegen direkt wissen wenn es Fehler macht. Bei vielen anderen Arten ist dieser Aspekt, der sehr natürlich erscheint nicht so selbstverständlich. Beim **selbst-überwachten Lernen** generiert der Lernalgorithmus die richtigen Lösungen und jeden Input aus gegeben Daten selber.

2.1.4 Unüberwachtes Lernen

Unüberwachtes lernen ist der Extremfall, denn hier bekommt der Lernalgorithmus ausschließlich die Inputwerte: $\mathbb{D} = (x_1, x_2, x_3, \ldots, x_n) \subset X$. Der Lernalgorithmus erhält also keine hinweise darauf was ein richtiger oder ein falscher Output ist. Unüberwachtes lernen wird meist eingesetzt um in Daten, Strukturen und Muster zu identifizieren. Ein Beispiel ist die Cluster-Analyse, hier bekommt der Lernalogrithmus eine Menge von Daten und gruppiert diese in Teilmengen. In der Abbildung 2.2 ist ein Beispiel für das Resultat einer möglichen Cluster-Analyse dargestellt.

Abbildung 2.3: Cluster Analyse angewendet auf 2-dimensionale Daten

2.1.5 Reinforcement Learning

Das Reinforcement Learning ist nicht so extrem wie das unüberwachte Lernen, bei diesem Paradigma liegen dem Lernalgorithmus, zwar auch nicht die korrekten Outputwerte vor, aber der Lernalgorithmus kriegt für jeden vorhergesagten Wert eine Rückmeldung,wie erwünscht dieser Wert ist. Hier ist ein Beispiel, ein Modell, mittels einen Lernalgorithmus, darauf zu trainieren ein Videospiel zu Gewinnen. Der Lernalgorithmus bekommt ein Abbild von der Umgebung und gibt dem Spiel ein Input, welcher eine Aktion zufolge hat. Falls nun die Aktion dazu beiträgt, den Spieler in eine gute position zu bringen oder gar das Spiel zu gewinnen, bekommt die Aktion eine positive

Bewertung, anderfalls eine negative. Intuitiv könnte man dieses Paradigma auch als „learning by doing“ bezeichnen, da der Lernalgorithmus nach ausreichendem Ausprobieren das gewünschte Verhalten erlernt.

Die Problemstellung im maschinellen Lernen ist allgemein gehalten und abstrakt. Aber genau aus diesem Grund kann maschinelles Lernen in so vielen unterschiedlichen Bereichen eingesetzt werden. In dieser Arbeit werden wir uns hauptsächlich mit dem Bereich der linguistischen Datenverabeitung (engl. Natural Language Processing) befassen, was uns zum nächsten Begriff führt. Liguistische Datenverarbeitung wird im folgenden mit NLP abgekürtzt.

2.2 Semantische Vekorräume

Das semantische kodieren von natürlicher Sprache in einen Vektorraum, ist ein bedeutsames Problem in NLP(Referenz). Der resultierende Vektorraum kann, für verschieden Problemstellungen (engl. downstream task) verwendet werden. Ein Beispiel ist das klassifizierten von Fake-News oder das zusammenfassen von längeren Texten(Quelle?). Semantischer Vektorraum heißt hier das ähnliche Wörter, also Wörter mit ähnlicher Bedeutung, im projizierten Vektorraum einen geringen Abstand zu einander haben. Diese Vektoren werden wir im folgenden Embeddings nennen. In einem optimalen Vektorraum würde im Figur 2.3 gezeigte Beziehung gelten.

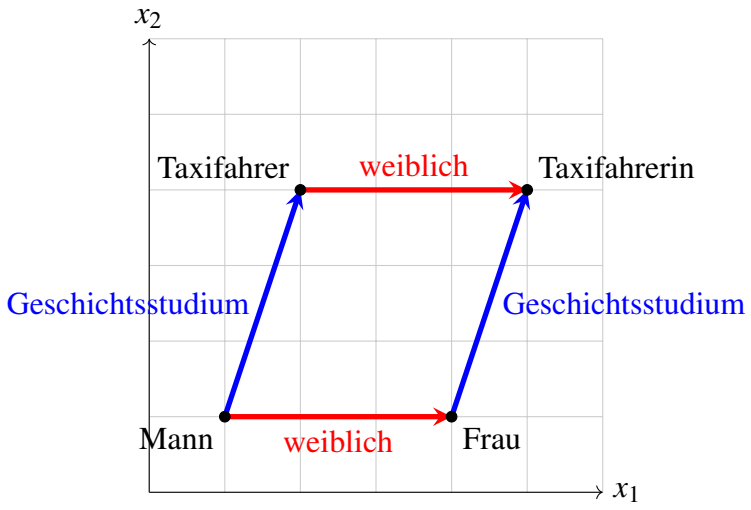

Abbildung 2.4: Optimaler fiktiver semantischer Vektorraum

2.2.1 Bag of Words

Der naivste Ansatz ist es jedem Wort im vorliegenden Text eine Zahl zu zuordnen. Daraus kann zum einen die Häufigkeit eines Wortes abgeleitet werden, aber auch die Sätze in denen ein bestimmtes Wort vor kommt, effizient gefunden werden. Die Bedeutung eines Wortes ist hier komplett unabhängig von der Wahl der zugewiesenen Zahl. Das führt dazu, dass sogar Synonyme einen hohen Abstand haben können.

Beispiel 2.3 *Sei $T = \{Input, Mann, Frau, Eingabe\}$ eine Menge von Wörtern, dann ist die zuordnung zu dem Vektorraum $\mathbb{N}^1$ wie folgt:*

$$Input \to 1, Mann \to 2, Frau \to 3, Eingabe \to 4.$$

Obwohl Eingabe und Input semantisch sehr ähnlich sind, haben sie hier einen sehr unterschiedlichn Wert.

2.2.2 Word2Vec

Im Jahr 2013 veröffentlichte Google ein Paper, indem sie zwei Modellarchitekturen vorstellen, mit denen ein neuronales Netzwerke effizient lernen kann, semantische embeddings zu produzieren. Das Neuronale Netzwerk hat in beiden Architekturen ein Hidden-Layer, die Gewichte des Hidden-Layers beinhalten nachdem Training die reelwertigen Vektor repräsentationen. Der Aufbau bei beiden Modellen ist in Figur 2.4 dargestellt. Zu beachten ist, dass der Hidden-Layer keine Aktivierungsfunktion besitzt, da er nur als lineare Transformation von einer One-Hot-Kodierung zu einem reelwertigen Vektor dient. One-Hot-Kodierung produziert für jedes aus n Wörten einen Vektor $v \in \{0,1\}^n$. Der Vektor besitzt immer nur eine eins und sonst überall eine null. Jede Position wo eine eins ist, gibt es jeweils nur einmal und kodiert so, mittels der Position, ein Wort. Der Output-Layer hat als Aktiverungsfunktion einen Softmax, der Sofmax gitbt Werte zwischen 0 und 1 zurück. Der Output ist also ein Vektor $v \in (0,1)^n$, daraus kann dann entweder je nach Anwendung ein Wort oder mehrere abgeleitet werden.

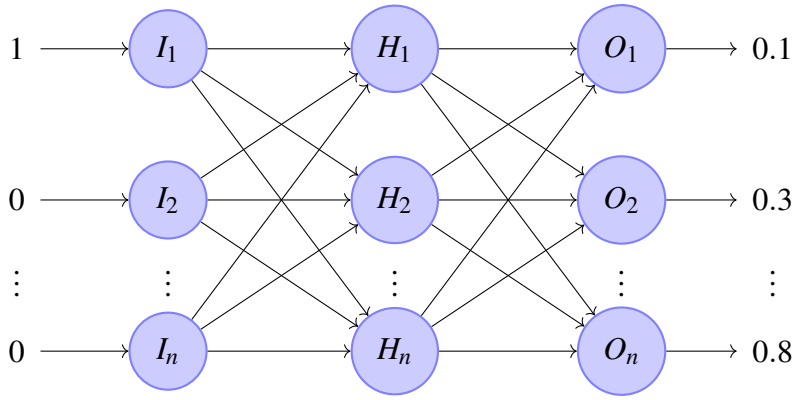

Abbildung 2.5: Neuronales Netzwerk von Word2Vec

Es gibt nun zwei unterschiedliche herangehensweisen, dieses Neuronales Netzwerk zu trainieren. Die erste ist Skip-gram, gegeben ein Wort muss das Modell die naheliegende Wörter vorhersagen. Dass heißt es muss aus einen Wort einen richtigen Kontext zu ordnen. Das Modell wird so lernen wörtern die ähnliche Kontexte haben ähnliche Embeddings zu zuordnen. Die zweite Methode ist Continous-Bag-Of-Words (CBOW), bei der ist es genaum umkegehrt, bei dieser sind Wörter nah an einen bestimmten gegeben und die aufgabe des Modells ist es dieses bestimmte Wort vorherzusagen. Hier ist also der Kontext gegeben und das Wort, welches in den gegeben Kontext benutzt wird muss hervogesagt werden. Da ähnliche Wörter ähnliche Kontexte haben, neigt das

Modell ähnliche Outputs für ähnliche Kontexte zu erlernen. Ähnliche Outputs können dadurch erzeugt werden, das die Embeddings für die Kontextwörter ähnlich sind.
Die Word2Vec Embeddings sind also dann ähnlich bzw. im Vektorraum nah aneinander, wenn ihre Kontexte in denen sie verwendet werden ähnlich sind. Das Problem ist, dass die Kontexte limitiert sind auf eine feste größe, die am Anfang gewählt wird. Sei nun die Kontextgröße $k \in \mathbb{N}$ ungerade, dass bedeutet, das der Kontext eines Wortes alle Wörter sind die $\lfloor \frac{k}{2} \rfloor$ stellen vor dem Wort positioniert sind und $\lfloor \frac{k}{2} \rfloor$ nach dem Wort positioniert sind. Falls das $k \in \mathbb{N}$ zu klein gewählt wird könnte es passieren, dass nur inhaltslose Wörter wie z.B. Artikel oder Präpositionen, im Kontext enthalten sind und somit das Wort mit Inhaltslosen Wörtern assoziert wird. Wenn aber das $k \in \mathbb{N}$ zu breit gewählt ist, könnten verschieden Kontexte verschwimmen und so ungenau Ergebnisse enstehen. Die Kontextgröße ist also ein entscheidender Parameter, für den Erfolg des Modells. Google selber löst das Kontextproblem in ihrem Paper „Attention Is All You Need“, mit ihrer Modellarchitektur namens Transformer.

2.2.3 Transformer

Das Paper „Attention Is All You Need“ ist ein Meilenstein im NLP Bereich. Die vorgestellte Deep Learning-Architektur ist Grundlage für BERT, Sentence Transformer und Large Language Models (wie z.B. Chat-GPT). Das Modell wurde Ursprünglich entwickelt um bei gegebnen Satz, einen sinvollen neuen Satz zu generieren. Die verwendete donwstream task im Paper ist das übersetzen von englisch zu deutsch und englisch zu französisch. Der Transformer kann aber wieder dazu verwendet werden semantische Embeddings zu produzieren, wie wir im Abschnitt zu dem Sentence Transformer sehen werden. Die Architektur hat zwei große Blöcke, die in der Abbildung 2.6 zusehen sind. Der erste Block ist der Encoder, dieser ehält die Eingabe. Beim Übersetzen wäre das der zu übersetzende Text. Der Encoder kodiert den Input in semantische embeddings und gibt den wichtigen Kontext für den Input aus, dass heißt er gibt aus wie wichtig jedes Wort in dem Satz ist für ein gegebenes Wort. Es gibt also keine Kontextgröße, sondern der gesmante Input ist der Kontext, aber der Encoder gibt vor, welche wörter wichtig sind und welcher eher unwichtig sind, für ein gegebenes Wort. Der Decoder kriegt den von ihm selber produzierten Output und das Ergebnis des Encoders, um das nächste Wort zu generieren. Am Anfang bekommt der Decoder ein konstantes Start Embedding und um die generierung zu stoppen gibt er selber ein konstantes Stop Embedding aus. Zu beachten ist, dass man mehrere Encoder und Decoder hintereinander schalten kann.

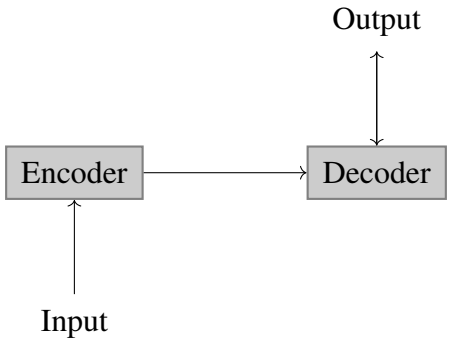

Abbildung 2.6: Transformer Architektur stark vereinfacht

In Abbildung 2.7 ist die Architektur genauer dargestellt. Im folgenden werden alle Bestandteile die in Abbildung vorkommen kurz erklärt. **Input Embedding** gibt dem Input der in Textform vorliegt eine Vektor repräsentation. Danach wird auf dem Vektor ein **Positional Encoding** darauf addiert. Der Transformer verarbeitet alle Wörter Parallel, deswegen geht die Positionsinformation verloren. Um diese Information trotzdem im vektor zu kodieren wird für jede position ein einzigartigen Vektor darauf addiert. Im Block **Add & Norm** werden zwei Inputs addiert und dann normalisiert, damit die Werte in dem Modell nicht zu groß werden. **Feed Forward** ist einfach ein Neuronales Netzwerk mit zwei Layern. Alle Wortembeddings werden nacheinander und identische in das Netzwerk ein gegeben. Der erste Input-Layer hat als Aktiverugnsfunktion ein Softmax und der zweite hat die Identitätsfuntion. Dann gilt für das Netzwerk:

$$FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2.$$

Der **Linear** Block ist wieder ein Neuronales Netzwerk, mit allen Aktivierungsfunktionen: $f_{i,j}(x) = x$. Nun fehlt noch der wohl wichtigste Bestandteil der Architektur das **Multi-Head Attention** Modul. Dieses gibt die Korrelation zwischen ein Wort un allen restlichen im Input aus. Hier wird also der ganze Input als Kontext genommen, aber das Modell lernt auf welche Wörter es im Kontext viel oder wenig aufmerksamkeit schenken sollte. Das **Masked Multi-Head Attention** Modul ist nur beim trainieren anders als das Multi-Head Attention Modul. Beim trainieren ist der Input des Decoders schon der Satz der von dem Modell vorhergesagt werden soll, deswegen müssen alle wörter die es noch nicht vorhergesagt verdeckt (engl. Masked) werden.

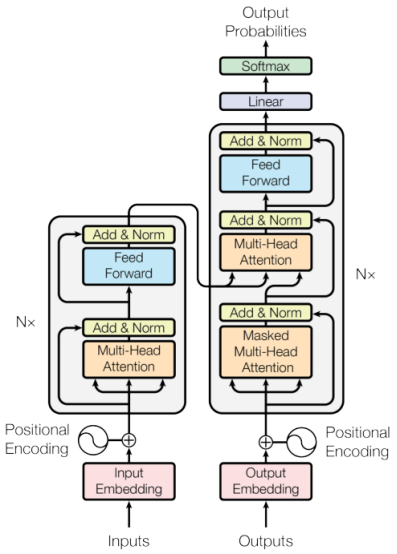

Abbildung 2.7: entommen aus „Attention Is All You Need“

2.2.4 BERT

Das BERT Modell ist wieder ein großer Meilenstein im NLP Bereich und kann als einer der ersten Large Language Models betrachtet werden. Das Modell verbessert die Transformer Architektur indem es die Embeddings der Token verbessert, zwei neue Trainingsformen benutzt und

aussschließlich Encoder verwendet.

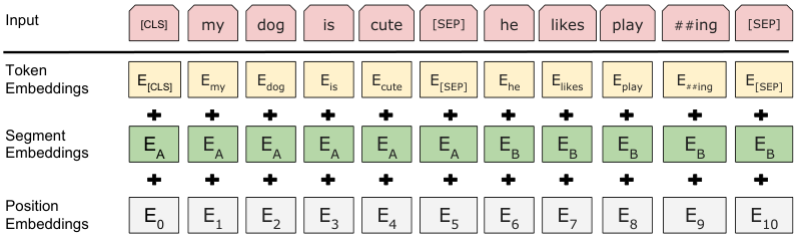

Abbildung 2.8: Token Embedding Prozess entnommen aus dem BERT Paper.

2.2.5 Sentence Transformer

Das Ziel dieser Arbeit ist es, mittels Quellinformationen funktionen in einen semantisches wertvollen Vektor zu kodieren. Semantische wertvoll heißt, dass semantisch ähnliche Funktionen ,wie beispielsweise zwei Sortieralgorithmen, im projezierten Vektorraum, mit einer gewählten Metrik, einen geringen abstand besitzen. Das Sentence-BERT (Reimers et al. 2019) Modell ist Stand der Technik, im semantischen kodieren von natürlicher Sprache in einem Vektorraum. Wenn man nun jeder Funktion, ein Text in natürlicher Sprache zuordnet, der die Funktion beschreibt, kann mithilfe von Sentence-BERT (SBERT) ein semantisch wertvoller Vektor für jede Funktion generiert werden. Die Autoren haben selber auch eine Implementierung bereit gestellt, unter den Namen Sentence-Transformer (LINK??), die in dieser Arbeit verwendet wird. Sie haben das Ursprüngliche von Google entwickelte BERT (Devlin et. al 2018) Modell genommen und angepasst. BERT ist ein Modell welches mittels einer neuen Deep-Learning-Architektur, zwei Textauszüge als Input bekommt und dann eine aussage über ihre semantische ähnlichkeit liefert. Die Hauptarbeit von Reimer und kollegen, war nun das BERT Modell so anzupassen, dass nur ein Textauszug als Input bereitgestellt werden muss, um anschließend einen Vektor zu erhalten, der die Semantik weitestgehend erhält.
In dieser Arbeit werden verschieden Methodiken vorgestellt, einer Funktion einen semantische passenden Text zuzordnen. Ein Tool mit dem man aus Quellinformation ein Text produzieren kann ist das Large Language Model (LLM).

2.3 Code-Llama

2.3.1 Large Language Model

2.3.2 Llama 2

2.3.3 Code-llama

2.4 Code2Vec

2.5 t-SNE

3 Methodik

3.1 Datensatz

Im Maschinellen lernen hat der Datensatz bzw. die Trainingsdaten den größten Einfluss auf die güte des Modells.

3.2 Datenpipeline

3.3 Stabilität von SentenceTransformer

4 Funktionskommentare

4.1 Motivation

4.2 Methodik

5 Code2Vec

5.1 Motivation

5.2 Adaption auf C

5.3 Training

6 Funktionsnamen

6.1 Motivation

6.2 Methodik

7 Coddelama-Erklärungen

7.1 Motivation

7.2 Codellama

7.3 Prompt Engeneering und Temperature

8 Ergebnisse

8.1 Evaluierung durch Experten

8.1.1 Methodik

8.1.2 Auswertung und Ergebnisse

8.2 Qualitative Evaluierung

8.3 Quantitative Evaluierung

9 Limitation

10 Diskussion

11 Fazit

12 Results: Comparing natural language supervised methods for creating Rich Binary Labels

- Stabilität von Sentence Transformer

- Kommentare von Funktionen um Embeddings zu generieren
- Funktionsnamen von Funktionen um Embeddings zu generieren
- Code2Vec um Embeddings zu generieren
- CodeLlama Erklärungen von Funktionen um Embeddings zu generieren
- Evaluierung durch tSNE-Plots
- Evaluierung durch Experten
- Evaluierung durch Formel

$I_k : \mathbf{N} \times \mathbf{N} \times \mathbf{N}^k \to [0,1]$

$$I_k(x,i,v) = \begin{cases} 1 & , \exists j \in \mathbf{N} : x = v_j \wedge i = j \\ \frac{1}{2} & , \exists j \in \mathbf{N} : x = v_j \wedge i \neq j \\ 0 & , \text{otherwise} \end{cases}$$

$E_k : \mathbf{N}^k \times \mathbf{N}^k \to [0,1]$

$$E_k(u,v) = \frac{1}{G_k} \sum_{i=1}^{k} \frac{I_k(u_i,i,v_i)}{log_2(i+1)}$$

wo $G_k := \sum_{i=1}^{k} \frac{1}{log_2(i+1)}$.

$CMP_k : \mathbf{R}^{N \times l} \times \mathbf{R}^{N \times l} \times \{\mathbf{R}^l \times \mathbf{R}^{N \times l} \to \mathbf{N}^k\} \times \{\mathscr{P}([0,1]) \to [0,1]\} \to [0,1]$

$$CMP_k(X,Y,f_k,agg) = agg(\{E_k(f_k(X_{i,j},X), f_k(Y_{i,j},Y)) | j \in \{1,2,3,\ldots N\}\})$$

13 Conclusion

14 Notes on form

14.1 Formatting

This LaTeX template uses the following formatting:

- font: Linux Libertine O (alternatively: Times New Roman)
- font size: 12 pt
- left and right margin: 3.5 cm, top and bottom margin: 3 cm
- align: left
- line spacing: one and a half (alternative: 15 pt line spacing with 12 pt font size)

When implementing the specifications in Word, it is essential to define style sheets.

14.2 Citation

The citation method follows the author-year system. Place reference is in the text, footnotes should only be used for explanations and comments. The following notes are taken from the *language* bibliography template from `ron.artstein.org`:

The *Language* style sheet makes a distinction between two kinds of in-text citations: citing a work and citing an author.

- Citing a work:
  - Two authors are joined by an ampersand (&).
  - More than two authors are abbreviated with *et al.*
  - No parentheses are placed around the year (though parentheses may contain the whole citation).
- Citing an author:
  - Two authors are joined by *and*.
  - More than two authors are abbreviated with *and colleagues*.
  - The year is surrounded by parentheses (with page numbers, if present).

To provide for both kinds of citations, `language.bst` capitalizes on the fact that `natbib` citation commands come in two flavors. In a typical style compatible with `natbib`, ordinary commands such as `\citet` and `\citep` produce short citations abbreviated with *et al.*, whereas starred commands such as `\citet*` and `\citep*` produce a citation with a full author list. Since *Language* does not require citations with full authors, the style `language.bst` repurposes the starred commands to be used for citing the author. The following table shows how the `natbib` citation commands work with `language.bst`.

Command	Two authors	More than two authors
`\citet`	Hale & White Eagle (1980)	Sprouse et al. (2011)
`\citet*`	Hale und White Eagle (1980)	Sprouse and colleagues (2011)
`\citep`	(Hale & White Eagle 1980)	(Sprouse et al. 2011)
`\citep*`	(Hale und White Eagle 1980)	(Sprouse and colleagues 2011)
`\citealt`	Hale & White Eagle 1980	Sprouse et al. 2011
`\citealt*`	Hale und White Eagle 1980	Sprouse and colleagues 2011
`\citealp`	Hale & White Eagle 1980	Sprouse et al. 2011
`\citealp*`	Hale und White Eagle 1980	Sprouse and colleagues 2011
`\citeauthor`	Hale & White Eagle	Sprouse et al.
`\citeauthor*`	Hale und White Eagle	Sprouse and colleagues
`\citefullauthor`	Hale und White Eagle	Sprouse and colleagues

Authors of *Language* articles would typically use `\citet*`, `\citep`, `\citealt` and `\citeauthor*`, though they could use any of the above commands. There is no command for giving a full list of authors.

Bibliography

The bibliography of this template includes the references of the *language* stylesheet as a sample bibliography.

15 General Addenda

If there are several additions you want to add, but they do not fit into the thesis itself, they belong here.

15.1 Detailed Addition

Even sections are possible, but usually only used for several elements in, e.g. tables, images, etc.

16 Figures

16.1 Example 1

16.2 Example 2

Abbildungsverzeichnis

Tabellenverzeichnis

Literatur

BUTT, MIRIAM, und WILHELM GEUDER (Hg.) 1998. *The projection of arguments: Lexical and compositional factors*. Stanford, CA: CSLI Publications.

CROFT, WILLIAM. 1998. Event structure in argument linking. In Butt & Geuder, 21–63.

DONOHUE, MARK. 2009. Geography is more robust than linguistics. Science e-letter, 13 August 2009. URL `http://www.sciencemag.org/cgi/eletters/324/5926/464-c`.

DORIAN, NANCY C. (Hg.) 1989. *Investigating obsolescence*. Cambridge: Cambridge University Press.

GROPEN, JESS; STEVEN PINKER; MICHELLE HOLLANDER; RICHARD GOLDBERG; und RONALD WILSON. 1989. The learnability and acquisition of the dative alternation in English. *Language* 65.203–57.

HALE, KENNETH, und JOSIE WHITE EAGLE. 1980. A preliminary metrical account of Winnebago accent. *International Journal of American Linguistics* 46.117–32.

HASPELMATH, MARTIN. 1993. *A grammar of lezgian*. Walter de Gruyter.

HYMES, DELL H. 1974a. *Foundations in sociolinguistics: An ethnographic approach*. Philadelphia: University of Pennsylvania Press.

HYMES, DELL H. (Hg.) 1974b. *Studies in the history of linguistics: Traditions and paradigms*. Bloomington: Indiana University Press.

HYMES, DELL H. 1980. *Language in education: Ethnolinguistic essays*. Washington, DC: Center for Applied Linguistics.

MINER, KENNETH. 1990. Winnebago accent: The rest of the data. Lawrence: University of Kansas, MS.

MORGAN, TJH; NT UOMINI; LE RENDELL; L CHOUINARD-THULY; SE STREET; HM LEWIS; CP CROSS; C EVANS; R KEARNEY; I DE LA TORRE; ET AL. 2015. Experimental evidence for the co-evolution of hominin tool-making teaching and language. *Nature communications* 6.

PERLMUTTER, DAVID M. 1978. Impersonal passives and the unaccusative hypothesis. *Berkeley Linguistics Society* 4.157–89.

POSER, WILLIAM. 1984. *The phonetics and phonology of tone and intonation in Japanese*. Cambridge, MA: MIT Dissertation.

PRINCE, ELLEN. 1991. Relative clauses, resumptive pronouns, and kind-sentences. Paper presented at the annual meeting of the Linguistic Society of America, Chicago.

RICE, KEREN. 1989. *A grammar of Slave*. Berlin: Mouton de Gruyter.

SALTZMAN, ELLIOT; HOSUNG NAM; JELENA KRIVOKAPIC; und LOUIS GOLDSTEIN. 2008. A task-dynamic toolkit for modeling the effects of prosodic structure on articulation. *Proceedings of the 4th International Conference on Speech Prosody (Speech Prosody 2008)*, Campinas, 175–84. URL `http://aune.lpl.univ-aix.fr/~sprosig/sp2008/papers/3inv.pdf`.

VAN DER SANDT, ROB A. 1992. Presupposition projection as anaphora resolution. *Journal of Semantics* 9.333–77.

SINGLER, JOHN VICTOR. 1992. Review of Melanesian English and the Oceanic substrate, by Roger M. Keesing. *Language* 68.176–82.

SPROUSE, JON; MATT WAGERS; und COLIN PHILLIPS. 2011. A test of the relation between working memory capacity and syntactic island effects. *Language*, to appear.

STOCKWELL, ROBERT P. 1993. Obituary of Dwight L. Bolinger. *Language* 69.99–112.

SUNDELL, TIMOTHY R. 2009. Metalinguistic disagreement. Ann Arbor: University of Michigan, MS. URL `http://faculty.wcas.northwestern.edu/~trs341/papers.html`.

TIERSMA, PETER M. 1993. Linguistic issues in the law. *Language* 69.113–37.

WILSON, DEIRDRE. 1975. *Presuppositions and non-truth-conditional semantics*. London: Academic Press.

YIP, MOIRA. 1991. Coronals, consonant clusters, and the coda condition. *The special status of coronals: Internal and external evidence*, hrsg. von Carole Paradis und Jean-François Prunet, 61–78. San Diego, CA: Academic Press.